SymfonyCon 2019 Amsterdam Conference Videos

With <3 from SymfonyCasts

Chapter 1: Keynote (Fabien Potencier)

Tip

SymfonyCon 2019 Amsterdam presentation by Fabien Potencier.

Time for Symfony 5! Watch live as Fabien gives you a behind-the-scenes look at the *many* things that go on behind the scenes to release a new version of Symfony.

Transcript & caption for the talk will be added soon.

Chapter 2: HTTP/3: It's all about the transport! (Benoit Jacquemont)

Tip

SymfonyCon 2019 Amsterdam presentation by Benoit Jacquemont.

Talk slides

The announcement of HTTP/3 at the start of November 2018 may have come as a surprise for a lot of us.

Indeed, compared to the 8 years that separated HTTP/1.1 et HTTP/2, this announcement came only 4 years after the release of HTTP/2.

But the biggest surprise is under the hood, with a replacement of the transport layer.

In this talk, we will explain why this version 3 of the HTTP protocol has been designed, especially around the latency topic.

We will cover as well how technically this version works, and what it will bring to our applications, and what are the challenges that will need to be addressed, in order to fully benefit from this new version of the protocol that runs the Web.

Transcript & caption for the talk will be added soon.

Chapter 3: How to contribute to Symfony and why you should give it a try (Valentin Udaltsov)

Tip

SymfonyCon 2019 Amsterdam presentation by Valentin Udaltsov.

Talk slides

I started contributing actively to Symfony three years ago. Not only is it a way to thank the project, but it's also an immense source of knowledge and communication for me. I would like to share my experience and to encourage contributions to Symfony.

What we'll discuss:

- how participation in an open-source project makes you a better developer;
- easy steps to join the Symfony ecosystem;
- where to get an idea for a pull request;
- branches and roadmap;
- coding standards, conventions and backward compatibility;
- rebase flow;
- review process. *

Transcript & caption for the talk will be added soon.

Chapter 4: A view in the PHP Virtual Machine (julien pauli)

Tip

SymfonyCon 2019 Amsterdam presentation by julien pauli.

Talk slides

This talk is about how PHP works. We'll learn together how PHP compiles, optimizes then executes your scripts, both in the Web environment and CLI apps. We'll dive into PHP's source code - written in C - to extract some parts of interest and study them to better understand PHP's behaviors as well as best practices in terms of performances (CPU cycles and memory allocations).

Transcript & caption for the talk will be added soon.

Chapter 5: How Doctrine caching can skyrocket your application (Jachim Coudenys)

Tip

SymfonyCon 2019 Amsterdam presentation by Jachim Coudenys.

Talk slides

When people talk about Doctrine (or any ORM for that matter), the performance issue always comes up fairly quickly. Besides the fact that Doctrine will help you develop faster, so a little overhead doesn't really matter, there are numerous options to increase the performance of the application.

By understanding how the system works in the first place, a lot of issues can be avoided right away.

When you have done everything to avoid these pitfalls, you can bring in the big guns: caching. Doctrine has several caching mechanism and since Doctrine 2.5 "Second Level Cache" was added to our toolbox. After this talk, you should know what the impact is of every cache and how to use them.

Transcript & caption for the talk will be added soon.

Chapter 6: Using the Workflow component for e-commerce (Michelle Sanver)

Tip

SymfonyCon 2019 Amsterdam presentation by Michelle Sanver.

We got the task to make an order API, from open order, to delivered, with payments in between and after. So there are naturally a lot of states, and a lot of transitions where we needed to calculate the prices correctly and handle credit card transfers. Keeping track of all of this, and when we need to do what, ensuring that an order is always up to date, and that it has the data it needs, and that we send good error messages when a user can not do an action, was a challenge for us until we discovered the workflow component.

This is a real happy use case story where I will show you how we did this, and how much more straightforward it was for us to build an otherwise complex system using the workflow component.

Transcript & caption for the talk will be added soon.

Chapter 7: Crazy Fun Experiments with PHP (Not for Production) (Zan Baldwin)

Tip

SymfonyCon 2019 Amsterdam presentation by Zan Baldwin.

Talk slides

I'll show you the crazy things you can do in PHP with streams and autoloader overloading to write your own language features. I'll also show you how you can supercharge your Symfony applications using aspect-orientated programming or encrypt source code on-the-fly using only PHP. As if that wasn't enough, we'll go even further and make PHP a polyglot language by importing esoteric language scripts! These aren't your average hacks and shouldn't be run in production... but let's explore these concepts as fun experiments so you'll never think of PHP as boring again!

Transcript & caption for the talk will be added soon.

Chapter 8: Hexagonal Architecture with Symfony (Matthias Noback)

Tip

SymfonyCon 2019 Amsterdam presentation by Matthias Noback.

Talk slides

Symfony offers many excellent components for your web and console applications. But of course, you still have to implement your own application logic, create your own domain models, and write your own tests. So Symfony has its place, but it's not going to be everywhere in your application.

In this talk I will explain an architectural style called "Hexagonal Architecture", which will help you structure your applications in such a way that you can focus most of your development effort on the core of your application, designing it in a way that makes its production code sustainable and easy to test.

Transcript & caption for the talk will be added soon.

Chapter 9: Crawling the Web with the New Symfony Components (Adiel Cristo)

Tip

SymfonyCon 2019 Amsterdam presentation by Adiel Cristo.

When developing an application, a common feature we need to implement is gathering data from other sources. These data are available in various forms, usually unstructured, and behind some JS application, making them harder to reach.

To make things worse, as the application evolves, we need to get more data, from even more sources. But don't worry, things can be easier!

In this talk we'll use the Symfony's HttpClient, Messenger and Panther to build a crawler, first as a simple console application, then evolving to a distributed one.

Transcript & caption for the talk will be added soon.

Chapter 10: Adding Event Sourcing to an existing PHP project (for the right reasons) (Alessandro Lai)

Tip

SymfonyCon 2019 Amsterdam presentation by Alessandro Lai.

Talk slides

"Event Sourcing", along with "CQRS", have recently become trending terms, and now there is so much theory, blog posts and talks about them.

However, most of these deal with the problem starting from an utopian assumption: having to write a project from scratch, but at the same time with a high domain complexity right from the start, enough to justify the use of a complex technique like event sourcing and CQRS, which carry a fair amount of inherent complexity. But the reality greatly differs: projects are born from small and simple prototypes, and they accumulate complexity only with time, growth and evolution of specifications and features.

This talk is a case history in which I will tell you (with little theory and a lot of practical examples) how we decided to add event sourcing to an already existing project (without eradicating the rest or rewriting it), to solve a specific problem (reporting and its history) for which this methodology proved to be the perfect solution.

Transcript & caption for the talk will be added soon.

Chapter 11: HYPErmedia: leveraging HTTP/2 and Symfony for better and faster web APIs (Kévin Dunglas)

Tip

SymfonyCon 2019 Amsterdam presentation by Kévin Dunglas.

Talk slides

Over the years, several formats have been created to fix performance bottlenecks of web APIs: the n+1 problem, over fetching, under fetching… The current hipster solution for these problems is GraphQL. It's a very smart network hack for HTTP/1, but a hack that is not necessary anymore with HTTP/2 and HTTP/3.

The new versions of the protocol of the web now have native capabilities allowing to create fast and idiomatic web APIs: multiplexing, server push, headers deduplication, compression, persistent connections (Mercure)… Leveraging HTTP/2 and HTTP/3 unveils the true powers of the web (hypermedia architecture) with no compromises with performance.

During this presentation, we'll learn how to design our APIs to be able to maximize the benefits provided by HTTP/2. Better, Symfony already contains all the tools you need to create (WebLink, API Platform) and consume (HttpClient) such APIs. We will discover these gems together.

HATEOAS is the new hype!

Transcript & caption for the talk will be added soon.

Chapter 12: PHP, Symfony and Security (Diana Ungaro Arnos)

Tip

SymfonyCon 2019 Amsterdam presentation by Diana Ungaro Arnos.

Talk slides

Have you ever tried talking to someone about using PHP in secure applications? It's nothing new that we deal with prejudice against PHP every day and the situation is even worse when we talk about security. The latest versions of PHP provide security tools and modern cryptography and Symfony itself make its efforts to deliver robust security features that are simple to implement. We'll learn about the latest language and framework initiatives in this regard and check out short and quick tips for boosting you application's security.

Transcript & caption for the talk will be added soon.

Chapter 13: What happens when I press enter? (Tobias Sjösten)

Tip

SymfonyCon 2019 Amsterdam presentation by Tobias Sjösten.

Talk slides

As a technical interviewer, one of the questions I like to ask the most is "what happens when I write www.example.com in the browser and then press enter?". The answer reveals a lot about the interviewee's understanding of a vast number of technologies that fringes web development.

In this talk, I will go through exactly what happens, down to excruciating detail, so that you will be better prepared for your future job interview.

Hello, can everyone hear me? Good. Cool. So second to last session, are we getting a bit tired today or spirits are up? We're good to go. All right, so my name is Tobias Sjösten. Very happy to see everyone here. So a lot of people in here. The, you've seen my last name. Uh, there's this ring with the two dots. That's a Swedish ER. So my accent didn't give it away. I'm from Sweden and this is my fourth SymfonyCon now. I'm very excited to be here and stumbled up on stage for the first time. Is anyone here for the first time. Ah quite a few. Very cool to see new faces here. So I've been, fourth SymfonyCon. I was also in the SymfonyLive back, back when there was a thing and before that SymfonyDay Uh, so I've been involved in Symfony for more than 10 years now and I've written a lot of code during this time.

First as a freelancer, uh, then later on as running a web agency and then as a consultant, up until last year. So more than a decades worth of uh, writing Symfony code for customers. Last year I was employed by this company Stim. I'm not gonna talk too much corporate sales or anything, but Stim basically. Uh, it's a company where you as a musician, it's a nonprofit organization. You as a musician can become a member and then you own part of the part of the organization. And if you write the music and your music is later on used to make money, we make sure that you get a piece of that money. And my role at Stim is as a software architect and that involves a lot of tasks that are non-coding tasks. For example, I do a lot of technical interviews, so when we have a candidate coming in and they want to work with us at Stim, we sit down for one hour and I get to prod them, nudge them and see where they are.

The Question

Like their technical skill level, was a very weird situation for me as a coder coming into. So I initially I did what any sane person do. You know, you go to Google and type in how to conduct a technical interview. And what Google tells you is that you should ask open ended questions to let the candidate speak freely and talk about whatever is important to them, what they think, you know, to show their width, and the depth of their knowledge. And we, we work mainly with web development at Stim. So I had to think and I came up with this one question that I thought was very clever. So I started asking our candidates this question to see how they talk their way through this problem and explain the situation. And it turned out to give very good answers. Like you could see people coming in as front end developers, they really focused on the front end stuff and they didn't care too much about the servers and things like that.

Whereas back enders were the reverse or full stackers, so it was a, I thought it was very clever, coming up with this question until one candidate came in and said, Oh, this question again. So it's a very common question. But despite that, a lot of the candidates who came in, they couldn't really talk their way through this question very well. So yeah, I, I will, I will show the question actually. Uh, what happens when I press enter, so I will, I will demo this this very quickly. So I will pull up, I think Chrome, I type in stim.se and I hit enter and there is a website. So what happens between me pressing enter and this website being rendered. So a lot of people got, couldn't really give a very good answer. So I thought why don't I go to SymfonyCon with a lot of developers here and maybe I can help you answer this question if you are looking for a job in the future. We are hiring by the way. In Sweden, if you'd like the cold just come talk to me after.

What Happens after pressing Enter

So the grid at the big overview of what does happen is your enter key is being pressed, a electric circuit is being closed, which flows some electric current through your keyboard to the logic controller logic circuit of the keyboard. It scans all the

keys on the keyboard to determine which ones are being pressed. It encodes that as one signal, which it sends to the operating system. In this case, the enter keys number 13 so it sounds that to the operating system, it goes through the drivers of the operating system. For this specific keyboard, the operating system interprets the signal sees that this is signal number 13 so it's the enter key being pressed. The operating systems checks which application is open on your browser, on your computer. And it sounds, in our case it's Chrome.

So the operating system sends to Chrome, Hey the enter keys being pressed do without what you want. Chrome realizes that it needs to do something else. So it looks at the address bar because in the address bar you can, you can type stim.se for example, which is a domain. Or you can type how to conduct a technical interview, for example. And depending on this various cases, Chrome needs to do different things. So Chrome figures out that it needs to contact the server to talk to a server to get a response back. Chrome contacts the server sends a request to the server, the server hands off the request to Apache. Apache of course hands it off to Symfony, right Symfony does some processing, runs your codes, connects to a database, does a lot of calculations. And then the response gets sent back to the browser. The browser renders this HTML in the response and we see our web page. However, I am terrible with electronics, so we're not going to talk any electronics today and I am really bad at what happens with the whole rendering process. That's a whole talk in itself. So we're not going to touch that. We are only going to talk about the backend stuff basically.

Protocol

So first things first, we need to determine what to do with whatever the browser, the user has written in the, in the address field. Hopefully it's a, we're going to assume that it's a, it's a URL of some kind. So this is one of the most complex part, of a URL. So here you have the protocol starting out HTTPS, which tells the browser that how we're going to communicate with the server. We're going to communicate over a protocol, HTTPS. We are requesting a protected resource. So I'm sending my username to be us together with a password, a secret to reach this resource. Then it's the www.stim.se, which is the domain. The sub domain is www. So we know who we're going to talk to. On this machine there might be several applications running and listening for web traffic.

So we can also define which port and thereby which application on this server that we're going to talk to. So you can send in 79 for example. By default it's port 80 or port 443. Yup. Thank comes the /hej, which is Swedish for hello. So if you don't learn anything else today, at least you know some Swedish now. So we want the page, Hey, at this web server. Ah ?q=a, we are telling the server, uh, something that is for the server to interpret. And finally we have a fragment which is #X, in this case, we never send this to the server. So whenever we get the HTML back from the server, our browser can look through the HTML and see if there's an anchor tag called #X. So it knows to jump to that immediately. So we know how to talk to, we know who to talk to. www.stim.se

We know how to talk to it through HTTPS and we know what to ask from it. We want the /hej page. If you remember my surname question, it has this funny character. So domain name kernel have these funny characters in them, in which case we, because in the internet for some reason doesn't speak Swedish. So we need to translate our domain name to our proper domain name, which is done through punical. So the browser also looks at their domain name now to see whether it needs to do that Punic killed translation.

TCP / UDP

Next, given the domain name stim.se, it doesn't give us enough information to know how to reach this domain. So for that we're going to use two protocols, which powers most of the internet. Uh, so these two together, TCP IP are over 50 years old. There's also one other part, called UDP, which is frequently used not, in the web yet. This suit used to be called department of defense model by the way, because it was developed by DARPA back in the, back in the old days. So the IP part that we're going to start talking about here is, uh, the locator basically, which is the address and a computer that connects to the internet is given an address so we can talk to it and it can talk to us. And of course, 50 years ago when they developed this technology, they came up with an address space with which holds about just over 4 million addresses, which of course it's never going to end right?

IP

There's so many address that we were going to last forever until you start connecting your phone, your computer, your fridge, your microwave, and so on. And then we quickly run out of addresses. So they had to develop a new, IP version, which is IP version six, which is more and more being used today. We're still running a lot, a lot on IP version four. So within the IP address, there's a protocol called DNS, the domain name system, which is a way for us to actually translate a domain name into an IP address so that we later on can talk to the computer. So the domain name system, it's something, most browsers, they use the operating systems built in function called getAddr. So they send in the domain name there and they get back an IP address. Chrome for some reason decided not to do that.

DNS

They have implemented their own DNS lookup. I don't know why, but they probably had some good reasons. So what the operating system does when it gets a request from application is it needs to, so it's a bit of a chicken and egg problem here that you have a domain that you want to translate to an IP address. In order to do that, you need to talk to a server to do the translation. The server, you can only be reached by an IP address. So how do you get an IP address in order to get an IP address? Ah, very clever. You have it pre-installed in your computer when you get the computer, or you can also get the list of IP addresses from the network that you connect to through the HCP. So the first thing though that the computer does the operating system does when it gets the request to translate a domain is that it looks at your local hosts file.

So you can override this whole DNS system in your computer. For example, if we have stim.se on the server here, everything's working, the internet is using that server. I want to move it to a new server. Then I take this, the site I copied over to the new server on a new IP address, and then in my local computer I can update my hosts file. So whenever I go to stim.se, I go to this new server. Everything looks fine, I update it so the internet goes to the new server. So that's a useful way to use your internal host system. If you don't have the domain name in your host system, though, the operating system moves on to their resolv.conf file. All of these are text files laying around in your computer and you're free to edit them and do whatever you want with them.

The resolv.conf file contains usually a list of name servers. You can add your own name server like Google has it's 8888 or CloudFlare has 1111 so you can add it yourself if you want a better name server. So we have an IP address for a name server. Finally we can connect to it so we make a connection. We ask name server, Hey, we have stim.se. What's the IP address? What you get back down is a pointer. A pointer can be there an A pointer or an AAAA pointer, whether it's IP version four or version six telling us that this domain name has IP address. It could also give us back our CNAME or an A name pointer, which tells us that this domain name is actually an alias for another domain name, so you have to go to that other domain name and they might be a couple of steps until finally hopefully get an IP address back or a list of IP addresses as well.

Each pointer has a TTL or time to live, which is usually by default it's set very high. So when you want to change your IP address, you might have the experience that it can take up to 72 hours or something like that. And that's the TTL that is set very high, which is a way for everything in between your computer and the name server to cache the the lookup. So the next time you go to stim.se, You don't have to go the whole way to look it up and then back. But you can write in your computer, it can be cached for however long this TTL set to. So next time it will be much faster. So we have one IP address or a list of IP addresses. Next step is to connect to it to start making our requests for what we want from the server and the internet is built upon a communication layers. You might have heard of the OSI model for example, which is an old way of structuring, like you have the electronics in the bottom and then you build upon layers.

TCP

The layer, the deepest layer that we will go through here though is the TCP layer transmission control protocol. So this is what powers all of most of our internet traffic today. FTP, SSH, HTTP and so on. And the way it works is that you take your data that you want to send to another computer, you chop it up into small packages. On each package you can set the number of flags on the metadata in the header of the packages. You number them one, two, three, four and then you send them on through the network. So then the receiving part can get, depending on which route the packages take, you can get number two first and then number three, then the number one. And then the receiving part can put them together and unpack them and get the whole whole picture.

So it's a way to to optimize by sending out all the packages out in several different ways. The packages have one outport and one import. So we are telling the package that from our computer, our outgoing port is one, two, three, four, five for example. So that when the response comes back, then, the server can tell us that this is going back to our application Chrome one, two, three, four, five. It also has an outgoing port, which is the equivalent for the server where we want to send it into the server. And finally they have a bunch of flags, these TCP packages. So these flags is used to communicate where in the process of communication we are.

And in order to establish our TCP connection first, we send one package with a SYN flag set to true the synchronize. So we send that off to the server, the server response setting, the ACK SYN ACK synchronized acknowledge bits, flags on the packages sends it back to us. And finally our client sets, sends one package with the acknowledge flag to the server. So synchronize synchronous, acknowledge and acknowledge back. So now we have done, ah, we have established a connection to the server that we want to communicate with them. We might have IP version four and IP version six addresses IP addresses, in which case we use a technique called happy eyeballs. So the server sends out both an IP four and IP six connection to the server and whoever responds to fastest is the one that we're going to use. So also a way to, to optimize because the way you've set up your IP version six might be very, very slow. So in that case, we want to use the old IP four.

And since we live in a modern web development world, we know that we're not going to just ask for one, one page. We're going to ask for images and JavaScript and CSS. And so on. Right? So the browser optimizes this by making six connections, uh, to, to the server. So it can do a lot of parallel file requests. And why six? It's like much other stuff in the internet it's just an arbitrary number that's, you know, it seems to work. And one interesting fact about these packages going back and forth. So in the nation, state of China, the government knows better than you, right. So in order to not let you connect to whatever server you want to connect to, they intercept this traffic, this first package, and instead of letting the server send back a SYN ACK package to you, they send a reset package. So then the connection is dropped and you can't go to Facebook.

HTTPS

Next up we have our TCP connection established so we can now communicate with the server. And on top of this we might want to put another layer of communication before we are ready to actually start sending data to them. So for example, if you wrote HTTPS colon slash slash in the web browser, then we are telling the browser that we want to make an HTTPS connection. So HTTPS hypertext transfer protocol secure. So it's a way for our browser to know that whoever we're talking to is actually the one we want to talk to and no one can listen on the traffic in between us. So it's encrypted traffic. So either we write in our browser that we want this specific protocol or we there, there's a technological HSTS, which is a way for the server to tell us that going forward from the first request. Now going forward, we want to use HTTPS for everything. You can also, if you have a domain that you know that we will never want to use regular HTTP. You can submit your domain to the browser lists and they will include it by default in the browser. So everyone always gets HTTPS default.

The way this works to establish HTTPS is that the client first on top of the TCP layer sends a package to the guest server saying client hello. Along with this request you also send a list of algorithms for encryption that you are able to, that the browser is able to encrypt. So in encryption you have, two types roughly speaking you have two types of encryption. You have symmetric encryption where you have one key that you use to encrypt it and decrypted you have also asymmetric encryption, which is when you have two keys. So you can use the first key to encrypt it and then you need to use the second key to decrypt it or vice versa. So the client here first in this step it sends a list of algorithms, symmetric algorithms to encrypt with, the server response saying server hello and he picks one of these algorithms to go forward with.

It also sends the certificate of the server to the client and the certificate contains a domain and an expire date. So we can from the get go we can see that this is actually the certificate for stim.se. So we are good to go with this one and it has not expired. The certificate also contains a public key, which is one of these, you know, key pairs and a signature. And based on the signature we can determine if it's a valid, like if it's actually, uh, if it comes from a source that we, uh, that we trust. So in, in your computer, again to solve the whole chicken egg problem, you have a list of certificate authorities, uh, that are companies that issues these certificates that your computer trusts. So given the signature of the certificate, you can check whether it's a valid issued certificate.

We also make a request to this issuer to make sure that it's still a valid request because sometimes these issuers get hacked and then you are in a world of hurt because then whoever hacked them can issue a new certificate. So you don't want to trust those. So we make an extra check just to make sure that the certificate is actually still valid. Now we know that this is a valid certificate. Then we have the public key contained in the certificate and we on the client side, we generate a session key. So the session key is what we're going to use later on going forward. We take the session key, we encrypt it with the public key of the server, we send it onto the server. Now since we have a public key and a private key the server only knows the private key, only the server can decrypt it. So now the client knows a key, a secret key and the servant knows a secret key. And we know and we have, we have all decided on an algorithm to use. So everything is in place now. Now we can encrypt our traffic, we know that we can trust this server and we ready to go. So the client says, client ready service says client server ready and we have one layer on top of the other, we're ready to go.

So in order to actually send some data, we need to format it. So that a web server can understand the request. So here's a very simple request. It would look something like this also in clear text being sent over the internet. So we're saying that we want to get the /hej page using protocol HTTP 1.1 and we are asking for this page from the host www.stim.se. Instead of the GET, there's many different, I think you have nine or ten different verbs. You can say that you want to delete this resource or you want to update it with a PUT or PATCH request and so on. The protocol part, I think we've also talked earlier about HTTP 3. So you have HTTP 1.1 which is the most common today. We're, we're slowly rolling over to HTTP 2 which instead of having these six connections simultaneously to send data.

So with, with HTTP you can, you can send a request and then you wait for the server, the server turns a response and that's it. Then you send another one, which is very, very slower and that's why they do six simultaneous connections. So with HTTP 2 you can instead send all the requests at one time and you get back a bunch of responses in the same connection. HTTP 3 instead is going to use that other instead of TCP, it's going to use UDP protocol, which is less reliable, but it's very, very much faster. So to speed up the internet. Along with the along with the first line and GET Hey, uh, we also send a list of headers. So in this case we were only using the host header. You can also send the content type to say that this data that I'm sent, this data that I am sending you is in text format or JSON format.

You can also send the accept header for example, and saying that the data you're to return to me should be in YAML whatever you want back. And there's one funny header called the referer, header. So whenever you're on one website and you click a link, you go to another website, then your browser automatically sends the referer header to the new website so that they know where you came from. Of course in the English, English language referrer is spelled with two Rs. But when they wrote up the spec in the 90s, they used a spelling program called spell. And this spelling program did not have support for referrer. So there was a typo sneaking in to the specification. So this is an email from, from Roy Fielding the big brains behind the rest API architecture. Just talking about how, we made a mistake, blame it on the France, blame it on France and that's why we have misspelled, header in our messages today.

Encoding the Request

So we have assembled our, our GET requests, we're ready almost to send it off to the server. Before we do that though, since the server, the internet does not speak Swedish, nor does it speak English. We need to convert our message into numbers because computer speak numbers, right? We do that with our us gear with Alaska table. So every character is represented by a number or a series of number. But to keep it simple, one number. So if you remember when we pressed the enter key, it was the key number 13. So key number 13 you see character return is the name of that key. And you can look up capital G, capital E, capital E, T and so on to spell out GET and the whole message. So every character gets a number. But there are many different number systems to pick and are a number system in English and Swedish and many other languages, we use the decimal system, right?

So we have 10 digits to represent the number with, going from zero to nine. There are other number systems, for example, the hex decimal system, which goes from 0 to F. So they representing 16 different digits and then finally have the binary number system. And when sending data over the internet, we do it through fiber optics, right? Cause that's the speed of light is very, very fast. But with light, you have a limitation. You can only have the light bulb being on or it can be off. So one of these number systems is better to represent data through fiber optics. So the binary system with a zero and the one it's very, very easily translated to the light being on or being off.

So given our number 13 for example, if we want to send the enter key over the internet, you need to count to 13 in the binary system. And the way we do that in the decimal system is that we count from one, two, three, four up to nine and then we don't have any more numbers. So we add a one and we're dropped down to zero and then up to 19 and then 20 and then 99 100 zone. That's how we keep growing. The binary system works exactly the same. So let's see if I get the rights. We have the decimal system, the decimal system here. You have the binary system there. So we've started counting the number of attendance attendees in the audience today. We started with zero and then go to one one, one and then we'll go to Magnus, then we'll have two, but we don't have the number two in the binary system.

So then we roll over and we do one zero the next one, three one one and so on and so on. So that way we can easily represent, uh, many numbers in the binary system. So doing the translation, we get something like this, the GET request. Uh, the G is number 71, which turns into zero one zero zero, zero blah blah, blah. And this is how we get the whole, the whole HTTP request in binary format. Now finally we have the connection. We know who to talk to. We have encrypted our message and we are ready to go. So we send our message over the internet in tiny little packages. They arrive at the server, the server orders them correctly. One, two, three, four binds them together, unpacks them and extracts the data. It looks at the header of each package.

Apache

See that this is an incoming package for port 80 and port 80 is listened to by this web server application called Apache. So we the server unpacks all the data and sends the data onward to Apache. Apache has a look at this GET request, sees that is that it's uh, going to the domain stim.se Which is powered by Symfony of course. Right? And it sends the GET request onwards to with PHP. So PHP, you can have it running either in a separate process. You can make connections over the internet, local, local network, or over the internet if you want. It could also be a sub-process of, your Apache program.

PHP started running. It takes your code, compiles into op code and executes the code. Uh, Apache also feeds in a lot of data, like for example, that the server, the global server variable, the files variable POST, GET, session cookie and so on. So you have a mess of global variables now ready for you in PHP to start executing and responding to this request. Of course, we use also composer, uh, to handle our loading. So we do, we use the composer autoloading, which when composer pulls down all the packages that we want to use, it builds a map of all these packages that this specific one is in here. This one is there. So whenever our code asks PHP to load this specific class composer can kick in and load that specific file without us having to handle that. We just tell PHP whatever we want, Composer handles it for us.

Symfony Handles Request

Finally we come into Symfony. Symfony takes all of these, Uh, so, so if uh, anyone here was coding web early 2000s, you

know there was very common to use frameworks like CodeIgniter cakePHP and so on Symfony as well calling themselves a MVC framework and model view controller framework. That, it's an architecture, a graphical user interface architecture from way back when, which doesn't really fit in the web. So you know, you had to make modifications to make it work in a web context and I don't know. No, it didn't really solve the problem. So from Symfony 2 and onwards, they said that we're not going to be a MVC framework. We are going to be a request, response framework, which makes a lot of sense in a web context. Right? So the way Symfony expresses this is through this beautiful little interface, the HTTP Kernel, which says that I can handle a request and I will return a response. That's it. That's, that's how we work. But of course this is just an abstraction on an interface, something for us to actually implement and fulfill.

So the way this is being done, is Symfony coming in, when the request comes into Symfony, one of the first thing it does is it creates a request object. It takes all this messy global variables, pulls it down into one request object, which it then can send into the framework to do its work, which is brilliant when you want to test stuff because if you use to write your own code PHP code back in 5.2, uh, it was a mess to, to make sure that all these global variables were up to date. There are no side effects when you, when you called one function, it didn't ruin anything for the old tests. So everything that we need to do is create a request object from all of this Symfony takes care of the rest

When we have the request object Symfony is a very uh, event driven framework. So Symfony, if you know about the solid principles, uh, the O in the solid principles stands for the open close principle, which says that your code should be open for extension but closed for modification. So you write your code once, you should ideally never have to touch it again, but you should still be able to extend it and add functionality or modify functionality. Using an event system that Symfony has, achieves exactly that. So we never have to touch Symfony ever again. But by using listeners to these events that goes on into Symfony, we can extend it and do a lot of fun stuff with it. So using this request object. We uh, we uh, sorry I lost my thread there. Using this request object we tell all our listeners that, Hey guys, we have a, we have a request coming in here. Do with that what you want.

Your listeners that you set up can respond with a response straight there saying that it looks at their, you can, you can look at their request to see that this comes from IP, blah, blah, blah. We don't want to serve those guys, so we respond with a 404 for this specific IP. If none of the listeners returns a response, the processing goes on to the router listener, which is a built in listener in Symfony framework. The router listener obviously uses a router to take this request and look up which controller should execute this functionality. It takes the controller and sets it on the request object. It doesn't execute anything it just prepares the request. With this data.

We move onto the controller resolver, which is responsible for extracting an actual controller from the, from the request. We have already prepped it so we can, we can grab the controller straight from the request because the router listener already put it in there and then we fire another event. This is telling everyone who's listening to this event that now we have a controller ready to do its work so you can plugin your functionality to do things like if there's an incoming request that states that it wants version 1.1 of this or version 2 of this request of this resource that you have exposed, you can, you can steer it to another controller. Uh, so that way you can do like API versioning for example. You can also see that we have an incoming request which has an ID in it, but this controller that we want to call requires a user. So we can take this ID, we'll look up a user and we inject it into the, into the controller arguments.

Moving on, our controller is ready to be called, we call the hej method on it, which basically just instantiates a response object, returns that. We have a response. Again, we call all our listeners. If we have not returned a response in our controller, we might have just returned on error, for example. There's a proceeding event called a kernel.view where you can hook in and you can do your transformation to make it an actual response. Either way, we have our response, we can listen to it, we can add some JavaScript to the response. We can do some logging or whatever you want to do. Add some headers. Symfony takes the response, sends it. This creates a, again, a text representation of the response. So this is the actual HTTP response being sent back. So again, we are using HTTP 1.1 we say that the result of this request, the response is a 200. Okay.

You can have many different responses here. The 200 series is that something went well. 300, it's do something else where. 400, you did something wrong. 500, I did something wrong. Um, and there might be, ah so here we also defined that this content that we send back is a HTML format, which obviously it's not. It should be a text. Uh, you can have many different, uh, headers tagged onto this response. You can add, for example, you want to allow anyone from the server and onwards to save this measure message. You can add some caching headers for example, or add some cookies for future requests.

PHP returns the response to Apache. Apache does not respond to the server because now we've put a reverse proxy in between. A reverse proxy listens to this cache headers that we have set up, sees that, okay, this, this a response that came in, it came back here, has been configured to be saved for one day, for example. So the next time someone requests the /hej page, we don't have to go into PHP, we don't have to go into Apache or our controller. We can just return it immediately. It speeds it up very, very much. Uh, in the next slot there's going to be a talk about HTTP caching. Someone's going to go into that here. Finally, varnish returns the response to the server. The server goes sends it into tiny little packages all over the internet, which can be intercepted unless we do HTTPS. They can be intercepted by proxies along the way that also can

listen to this, uh, HTTP headers, cache it for future requests so that in your building, your company building, you might have our proxy standing to speed up requests within the company or whatever

The browser receives the response. It also caches it for future requests. So if you cache it in this layer here and you make a request, again, you don't even have to go out to the internet to do any DNS lookup, it is very, very quickly. Now we do the rendering, which I have no idea how it works. It just paints a pretty picture on the website. We also need to fetch new images, JavaScript, CSS and so on. So we use the six connections that we have to send more and more requests until we have everything that we need and we can fully render the page. Then, we send one final package to the server with a flag called FIN for finish, send it off to the server saying that we're done. The TCP connection breaks. And that concludes my talk. Thank you very much.

Chapter 14: Configuring Symfony - from localhost to High Availability (Nicolas Grekas)

Tip

SymfonyCon 2019 Amsterdam presentation by Nicolas Grekas.

All apps need configuration: the database host, background color of your client's theme, API token to some service, number of items per page on the blog, etc. Where should all this configuration live? Operating systems, PHP and Symfony provide many options: environment variables, .env files, plain PHP constants, dependency injection parameters, key-value databases, "secrets" vaults, etc. You have many choices! And with great choices comes great responsibility... and complexity!

In this talk, we'll review all the ways to store configuration, which is best for each "type" of configuration and the benefits and drawbacks of each. We'll also talk about Symfony 4.4's new "secrets vault" as one of the ways to store sensitive settings. By the end, you'll be ready to configure anything from localhost up to a cluster of high resiliency microservices.

Transcript & caption for the talk will be added soon.

Chapter 15: HTTP Caching with Symfony 101 (Matthias Pigulla)

Tip

SymfonyCon 2019 Amsterdam presentation by Matthias Pigulla.

HTTP caching is a powerful technique to improve response times for site visitors, make more efficient use of bandwidth and reduce server load. We will have a look at the basic concepts, the different caching strategies and HTTP headers used to implement them. I will then show you a few ways how to implement HTTP level caching in your Symfony application, how to leverage the HTTP Cache included in the framework and resolve the mystery around ESI.

My hope is that when you return to work on monday, you will bring some new tools to your project that you can start using right away.

So welcome everybody. I hope you have been enjoying the first conference day so far. And um, I hope you all brought a bowl of popcorn for this talk. This is, um, the HTTP caching fundamentals talk and I've designed this for the beginners track. So my assumption will be that you know some the basics of Symfony and have used it for some time, but you need not bring any thing, any knowledge regarding HTTP caching in particular. I'm going to cover all the basics. Regarding questions, I might be a bit short on time, so I hope you don't mind if you could just keep note of your questions and we'll postpone them and have a Q&A section in the end or you just come up in the, in the, in the lobby and ask me after talk.

Simple math questions

So I would like to start with a maybe simple math question for you. Just check the answer. How much is 34 times 17? 578. Awesome. Great. Okay. Um, sorry, let me try this again. How much is 34 times 17? Great. Okay. And now that was way faster. And what you've just seen in the front here is caching. So you've done some compute intensive work, hot labor and we're able to put the answer on shelf and reuse it just in case someone else comes around later and asks the same question again. So if I'd speak HTTP, I might have asked you something like, like this: a GET request, GET /multiply, whatever. And um, if you were caches you would have built what is called the effective URL. So, you would basically put all this information together into one URL that contains everything, assuming we're talking over an encrypted connection, you would also put this HTTPS in there. And with this effective URL you would build a cache key.

Cache key

So, the cache key is basically for a key-value store where you have a key to put stuff and get it back later on. And in this cache key you put the HTTP method that you've been, that we've been using - this is GET in this case - you put in the effective URL that we've just built and there may be some other stuff that also goes in there, but for now we can just ignore that. And then you wait for the response to come along and, oh, not, sorry, not wait for the response, but you would only do this for some kinds of requests.

Cacheable HTTP Methods

For GET obviously and for HEAD requests and um, the standards even allow you to do this for POST requests. Now this is something you might keep in mind for the social event tonight as a fun talking point. But, um, you should really try to post, um, to cache POST requests in production and there are lots of quirks and must do and edge cases where you may or not do this.

So, almost any cache will anyway just ignore the POST requests, but it's in there in the spec, but just as a fun fact. So then you'd wait for the answer to come along.

Status codes cacheable by default

And um, yeah, there, there are quite some status codes that come back that you might want to put into your cache. Obviously 200 is successful response as the first one you want to cache, but it's also perfectly fine to cache other things like the 301 "Moved permanently" or the "Not found" 404 status code because that's all information that might still be valid quite a few minutes or sometime later. This is a quite a long list of status codes that permit caching. And in fact those even permit caching by default, that is without any further tweaks or configuration and had a sent these status codes are allowed to be put in a cache.

A picture of the Internet

Now I've sketched up a little, a picture of the internet. So what you can see, um, on this side, on the right hand for you, it's the left hand side, sorry. Um, is what the standards called the user agents. These are the clients that issue requests to your service. And on the other side we have what is called the "origin". So the origin is the server that knows the ultimate answers to those requests. And in between those two, there may be some caches, some even some layers of caches. Um, yeah, you maybe have heard of CDNs, content delivery networks, which are basically just HTTP caches that try to be as close as possible to the, um, to the user agents to answer requests as fast as possible. Or you may have a, well, reverse proxy. Sometimes also it's called a gateway cache, a load balancer in front of your application service in case you're using several of them.

All these, um, 3 on the right hand side of together called the surrogate caches because those are caches that are typically under your control. You can configure them, you choose to deploy them and they are acting as um, on your behalf. So you control those. Now I've put these, um, green and bluish dots inside the components to show where the actual, the actual caching takes place. We may have a cache in the browser, which is called a private cache and it means that this cash keeps information only for one particular user agent for one, maybe person. Whereas the other caches with these bluish dots are called the public or the shared caches because they star requests and responses for more than one user agent. So that's the one distinction I would like you to take away from this slide. And between those components we will be speaking HTTP.

So I guess the browser is not using HTTP to communicate with its own internal cache, but, um, when we move across process boundaries or host boundaries, then HTTP will be the way of talking to each other. So if you want to read more on this, um, this is, um, these other standards, the place where you can look it up.

Live presenation on Symfony sandbox application

And that brings us to the part of the presentation I'm a bit afraid of because, um, now we see what happens. So what I have prepared for you is a simple Symfony sandbox application, that I will use to demonstrate caching. I will be giving you the address of the GitHub repo later on so you can use it and play around yourself. What I would like to demonstrate you first is that... Is this readable from from all the way back. Yep. Okay. Thank you. So the first thing I would like to show you is um, I have a simple view that just shows the headers sent by the user agent that are relevant to caching when I request this page.

Okay. So, um, in this case... Also this is readable that I like this. Um, in this case nothing particular has happened. And also when I follow a link, which in turn takes me to the same page, nothing happens. If, however I entered the address in the address bar, you'll see that now there's another header, which happens to be the same when I press the reload button. And yet another one is submitted when I use a force reload that is using shift and pressing reload. So what I would like to point your attention to is that when trying to understand or debug why caching works or does not work the way you expect it to be careful when doing this from your browser because there's always some state and a browser on which page you have been before and which is the page you're currently at and how are you trying to reload it.

cURL as a command line tool for fetching content over HTTP and more

And it may be hard to, to really see what's going on if, if you're using this. So instead it would be better if, um, if you add another tool to your toolbar and I'm sure most of you have heard about this, this is called cURL. cURL is the command line tool for command line tool for, yeah, just fetching content over HTTP and doing lots of stuff. And um, for example, I could just make a simple cURL that's a capital -I - it says I want to make a request and please just show me the response headers. I add an "-X GET", which tell cURL that I want it to make a GET request because when I use this ice which usually just makes a HEAD but I want it to be a GET and then I have to add the address, which is this one.

Cache control headers

So, and and what I get back is the, um, basically the um, the HTTP response and the cache control headers. So this is more reliable when trying to understand what's going on because there's also no cache built into cURL. So you get clean and predictable responses every time.

If you have a look at this, a cache control header, cache control response header, this is the way of the server telling the client whether it's okay to do caching or under which conditions to do. And for now, just note that no cache private is added by Symfony. And this is a pretty, yeah, it's a safe default because it basically tries to tell the user agent, don't try any caching at all unless I tell you otherwise. So let's make this more interesting and let's try to write some controller method that shows the timestamp. That returns a response. I want to render a time, I should probably go that route. If all goes well. I seem to have some DNS issues. Okay. So this, um, this works. So it's, um, this template is just showing the, um, the time at which the response has been generated on the server. And I also included some JavaScript to show the time in the browser whenever this page is displayed. And, um, there is nothing in particular happening right now. But what I would like to do now is to tell the browser that it's okay to cache this response. And in order to do so I need to take this response, return it, and in between I

add an additional header.

Max-age header

Sorry, wrong URL, thank you. I'm going to rely on you the next 40 minutes. Thank you. Okay. Um, so now we have a cache control max-age=10, max-age=10, um, tells the user agent that is okay to keep this response and the cache for a lifetime of 10 seconds and you need not come back. Ask me about this for the next 10 seconds. So this is basically the, the fastest response you can get is one you don't even have to ask for because you know the answer already. So, if we try this in the browser, and I'm going to reload this, I don't know what's wrong about my DNS. So I've just fetched this and now I'm going to click this link again. Click it again, click it again. And you can see that only the time in the browser advances so this page is rendered again, but it has not been fetched from the server, from scratch, which will happen right now because now this 10 second period has expired.

So, once this response has exceeded this 10 second lifetime, um, it is called to be "stale". So, unless the time is, um, past due, it's fresh. And after that it transitions to a stale state. Being stale, however, does not mean that the browser or the user agent has to just discard everything. But it merely means that it has to revalidate. That has, it has to check back with the server and fetch, no ask again if it's still okay to still use the same response. So to show you how this validation thing works, I'm going to add another.

ETag header

I'm going to add another method. This, this view just shows the current week day. We also need a round here. Okay. That works. And um, now I'm going to add two additional headers to this response, which are called the ETag. I'll just put in some ETag there, explain on that in a minute, and I'll also say that, um, this thing has been last changed or was it today midnight? Yeah, the day changed at the midnight, but I'm not sure if there's a today midnight expression is correct. Let's find out. Yes, it is. So now the response that I'm sending includes those two additional headers, ETag... No, that one and that one.

So, the ETag is basically some identifier that is sent by the server to the client. It does not have any meaning to the client. So it's just an OPAC string and we have this last modified information. And using this information, a client can now check back with the server and ask whether it's still okay to use this response. Can ask has anything changed on the server since I have last seen this ETag or last modified information from you. So, with the ETag, what happens is that the user agent includes an additional request header, which is the capital H switch adds an additional header to send to the server and it says if none match - there's so um, it asks the server to send me this response, but only if none of these attacks that I've given you actually match.

304 "Not modified" response

And what happens is that I get back at 304 "Not modified" response. This is the server's way of telling me that nothing has changed since I have seen this attack. And it's okay to just keep on using the response that I already have. We can also do this with the um, the last modified information, in which case it's not the if not match header but it's if modified since, just send me the response if it has been modified since that point in time.

And again it's a 304, it has not been modified on the server since then. Now you might wonder where this 304 actually comes from because there's nothing special in my controller that I've done to just verify this information. And in fact this is because I'm using Apache as my web server and the web server was smart enough to just get the regular 200 status code response from Symfony, see that there is a matching ETag or last modified information and then it just omitted sending back the response buddy and instead turned it into a 304 "Not modified". So, in Symfony we've gone all the way through this and did all the work of rendering the... the response and running Twig and stuff. So, we do better in this case if we, if we take the risk, the requests, and then we have to create the response.

Sorry, we have to create a response ourselves and then we can use a method. If response is modified, is not modified - request. So, this particular method compares the request that has been made to the response that we are probably going to send out. And if this has not been modified, we can return the response right away in this place. And only in case we see it that there has been a modification, we need to do the actual work of rendering the Twig template. So, in this case we can render it and put it into the response object that we already have. And let's first see that it works at all. It does. So now if I add this, if modified since header, sorry, I want you to be able to read this. Um, if I add this "if modified" since header again, we now get this 304 status code like before. But in this case we've been able to shortcut all the work inside the controller and we did not need to, yeah, render the template and stuff. Just to prove you that this really works I'm going to, I'm going to throw a runtime exception right here. So, sorry.

This one's the worst because it's the one that shortcuts the control action. But if I omit this header and ask for the complete response, we're now getting this internal stop error just because there's the, um, the runtime exception in here. So, what you've now seen is basically a combination of um, yeah, the validation and um, now there another header that we haven't...

oh, it's not a header, it's a, a, a, a directive inside the cache control header, which says must revalidate., Must revalidate, tells the user agent that once this response that it has has turned stale, it must, in any case, check back with the server to make sure that it's still up to date. So you might now wonder on the which conditions this response can be stale at all because we have not specified a lifetime for this response at all. And in fact, user agents are allowed to... when, when the cache control headers present and says, it's okay to keep this and your private cache, then user agents are allowed to calculate a heuristic expirations. So based on this information in the last modified header, which has now I'm about 16 hours ago, the um, user agent may figure out that it is probably safe to just cache this for a lifetime of let's say 5 minutes or so. So if you want to prevent this, heuristic, um, expirations, it will be better to just be explicit about the exploration you would like to use. And we could either add a flag to this response and say "no cache".

No, we have no cache private again, which is also the default issued by Symfony, which in fact does not say it's not okay to put it in the cache. So this does not prevent caching, but it just says: if you cache this, you must not serve it from the cash without asking me. So, no cache is probably one of the big, yeah, I would say I'm a misnomer and this not really intention revealing what is some, what this header does. So yes, keep it in the cache, but um, asked me every time. So this will be one way to prevent this heuristic caching from taking place or the other one would be, um, just as before to add the, um, this max-age thing.

Cache annotations from FrameworkExtraBundle

And I'm going to do this now with a, with the cache annotation from the FrameworkExtraBundle, I can now say, okay, keep this for 10 seconds and now we have a response that can be kept in a cache for 10 seconds and after those 10 seconds have expired, the response becomes stale and we need to perform revalidation by either checking with the ETag which is the preferred way of doing this or by using the last modified information. And if we find out that their response is that, well we can use it for another 10 seconds.

So this is all fine and dandy and we have now responses then that can be kept in our user agents caches. But we have not really achieved any performance improvements on our server side. So we're still basically facing the same requests and um, have to do all the same computational work. And um, to improve on on this side, we need to add another component. We have to add a public or a shared cache to the Symfony application. Now what I'm not sure if you know about this, there is a HTTP cache implementation written in PHP, which is part of the Symfony distribution as you get it. So you do not need to start with Varnish and have complex server setups, but you can just use the Symfony HTTP cache and basically if you can deploy PHP to a server, you can include the HTTP cache as well. And, um, it's pretty much stable and it offers a very fast response times already. So that may be a good thing to start with. So let me show you how this works by adding it to our demo application.

To do so we need to go to the index.php file, which is the front controller, the very first script that is started when, when Symfony actually starts. And we put this cache right here and wrap it around our Kernel. So you may have heard of the decorator pattern and object oriented software, which means that you put another object, you put a new object in front of another one to intercept, handle and modify the requests that are being made.

So, I'll take this one, put it in front of the Kernel and fired the same request again. Now, either I forgot to remove. No, I didn't. You know what's wrong? Nope. Okay. Oh I, I probably chose the wrong cache implementation. Let me try this again. That's the one I want to be using. Okay.

So, I make the same request again and we can now see that there is an additional header that has been added. So the Symfony cache is now in front of the Symfony kernel and all requests that usually were directed to the kernel and now first handled by the Symfony cache. And, um, yeah, the cache can either try to provide a response as fast as possible and return it without even starting the Symfony kernel booting it up at all or otherwise, if it has not yet cached a response, it will need to just do what, what happens to usually. So in this case, um, this, this information is added in, in the debug mode only and it tells me while the cache is in place and I've seen a GET request for a /weekday and um, still valid, forget about this. That's from my tests. I need to just remove what's in the cache. Let me do this again. So this is what you should expect. Um, I've seen the request for /weekday, but that was a miss. I do not have anything in cache for this. And this is because the method that I'm using here is still using this private header. It says you may, you must only keep this in a private cache, but you must not share this information between several users. So in order to make our weekday cachable and the shared cache, we go to this showWeekday() controller. And um, I just use the annotation. I'll say this is a public response. It's okay to use this in a shared cache as well.

No, look at this line. The request was a miss. There was nothing in the cache, but we are now able to store the response and what we see is a response that has been so from the cache. So cache requires to add this Age header. The Age header tells any downstream clients how long this response has been sitting in the cache altogether. So it's important if we know that we may use this for a period of 10 seconds, it may be important to know that it has that it has been sitting for 5 seconds in a cache already because that reduces the time that we are ultimately allowed to make use of this response. So that was a miss, but it has been started. Now I'm going to make this request again. Now the cache says: okay, I had a stale response

because the 10 second period had expired.

I validated that with the backend server and I figured out it's a valid. And so I started and kept it and um, yeah, by the way, here is the response and I got this as a 200 status code. Now I'm going to repeat this two more times and be a bit quicker. Now this time I was so quick that um, I even got a fresh response response from the cache. So the cache did not even have to contact the Symfony kernel at all. We did not even boot the Symfony kernel, but we could just grab the response from disk and um, crank it out. So, um, yeah, that's the best possible case that can happen to you.

Be careful with "public"

Now I need to, um, put a, what a word of caution, um, regarding public caching because if you put this public cache control header on your response, it basically says that it's okay for any cache along the way to keep this response and this even overrides some protection mechanisms that are in HTTP by default.

So, for example, if you're using HTTP authentication and this provides the authorization header and that usually prevents any cache from storing the response. But if you put this public in there, it says it's perfectly fine to keep this and don't worry about the authorization thing. Also, when it comes to cookies, session cookies, you may have some lock in area on your pages. This information is not taking into account when the cache keeps a public response. So be careful when adding public because that can lead to, yeah, just mixing up content that is for different users. And um, you may not even be aware that some user is getting someone else's response because you usually don't see what is going on in the caches downstream. So did I say you should be careful with adding public to your responses.

Vary header

Well if you... if you do this, there's one more header. I'm not going to demo this in detail, but I want you to tell about this. This is called the Vary header, which you can set on your response. And with this Vary header, you give the caches information, um, that they need to put additional information into the cache key I mentioned in the beginning. So when I said we skip this information for now you can tell a cache that it can cache a response, but it also needs to put, for example, the cookie header into that cache key. So you can make sure that different users with different cookies get different responses, which if you make this this way for four sessions, basically you can omit the cash because it will be caching one, one thing per one user. But there may be cases where we are able to yet just find a set of headers that makes a perfect combination. So you pick different responses apart and at the same time still be able to use them for, for similar users.

So, if these are the two options, what, what can we do to have requests in our cache, um, where we maybe want to mix private and public things. So for example, a part of the response is meant to be for one particular, particular user only. Whereas the other part is public. Or maybe we want to have responses where some information is only valid for a very short time and other parts of the information can be used for much longer.

Edge Side Includes aka ESI

And for this use case, there is a cool feature that is also built in Symfony which is called the Edge Side Includes (ESI). Edge Side Includes is originally a technique to get just compose a page of several fragments and to do this as close as possible to the user agents at the edge of the internet or on the edge side caches. But you can also use the Symfony HTTP cache to do this just in front of your back end. And um, I would like to show you how this works by creating a dashboard, lets say dashboard action. So, I want to issue a greeting to some user, which is obviously a private response because it contains the person's name. And I also want to include the weekday and the time as we've seen before and put this all together. So I will write this showDashboard(), response. And I've also prepared a Twig view for this one. We need to add a route again. Now in this dashboard, nothing fancy so far. Let's try this. Yeah, it just fetches the name parameter from the request and hopefully says "Hello SymfonyCon". So nothing fancy so far, but now we're going to just embed what we've, what we've seen before the time and the weekday and we do so by using a Twig helper function called render_esi(), which renders and edge side include. And inside that we need to put a reference to the controller that we would like to be executed in displays. So I'll just go back to this demo controller, copy a reference to this method. Put this in here and I need to recycle the back slashes. Put them in here over there. And I'll do this for the... Sorry?

I'll do this for the weekday as well. Now this is a feature I need to turn on in the framework configuration. So we go to the framework Yaml configuration. We have those two switches, ESI, which you may have noticed before. And also the fragment thing. I'll just turn both on for now and show you what happens. And then the best case, we now have a response that contains both the name and the information that we've had before. You may be surprised that um, those links are missing in the fragments. That's because I am in my, in the base template. I've made sure that, um, I only add the HTML body and stuff once for the entire response. And when I detect that we are adding fragments together, then I just skipped this. So just to yeah, I probably confused you with that.

Now what we see is that, um, we get this entire response, but I would like to show you how this works behind the scenes and

um, to do so I just pretend for a moment that I am and yes, I aware of cache and I, um, I need to first turn off the cache, the front controller for a second because if the cache is present, it's going to do the ESI processing for me. And I cannot pretend to do this myself. So just comment this out for a second and then we will be doing this. This request, do I need to quote this? I'm never sure. And then I will tell the cache, no, the HTTP kernel fact, that I am a Edge Site Include aware cache in front of it. So, you need to know about this one. Um, that's just a, that's just for showing what happens behind the scenes. So this tells the Kernel, um, I'm a surrogate cache in front of you and I'm capable of processing Edge Side Includes. Okay. So, this time I would like to see the response body in fact, and what you can see is that now this dashboard request that I've, that I've made includes this fancy ESI include markup. So this is the first response that is returned by the kernel and put into the cache. And now the ESI processor inside the cache will have to fetch two additional resources, which are referred by this, this, this mockup. And um, yeah, ir will just fetch that, put it in this place and let's again turn this on.

Go back to the usual request. And, um, we can now see that the Symfony cache has processed a whole lot of requests. The first one is the main request, which was a miss. The second one was this fragment with this particularly long URL, which was also a miss. And the third one, well it was just stale and could be validated and could even be start later on. So what the cash has been doing now is that it's basically just juggling with three different responses. It has a cache, each of those with its own cache control information. And whenever a request now hits the cache, the cache will take those responses from the, from, from storage, merge it together and find the best possible combination of know of cache control headers that still satisfy everything that's gone into this. So, because my, uh, my personal greeting page is private and no cash. Obviously the anti response must also be private and no cash. But, um, we can make use of the, did I leave this cacheable?

Well, the, the, the showtime controller is not even public. So, um, we need to make this public as well. And now we have the time that it can be cached for 10 seconds. Um, we have the weekday that can be cached for 10 seconds and now we should be able to, um, repeat this request a few times and now the caches should be warm and um, we have a fresh, so a completely cache-based HEAD for, for both of the fragments but still a miss because we do not store the, the personal information at all and we can build this all together and this can give you pretty good performance boost if you are able to just separate the public and the private content generated by your, by your application.

Yeah. I promise to give you the um, the uh, GitHub link. So this is the address of the repository where I've put this. Um, yeah, I also try to put, um, the changes I've made during this presentation so you can follow up on that later on and maybe you can use it to just play around and yeah, get a feeling of how things work.

Questions

So, if we have time for questions, which I'm not sure when, when shall we finish at? Yeah, quarter past four. Sorry, one minute. Yeah. Okay. So a one minute is a bit tight maybe. So if you want just, just come up after this talk, ask me your questions or ask me at the social events tonight or tomorrow - I will be around for the entire day. Um, yeah, and just, um, see if I can provide any useful answers. Otherwise, have a great conference. Have a great social events tonight, enjoy your stay and thank you for your time.

Chapter 16: How fitness helps you become a better developer (Magnus Nordlander)

Tip

SymfonyCon 2019 Amsterdam presentation by Magnus Nordlander.

We often think of technical skills as the way to level up as developers, but we're not (yet) brains-in-a-vat. Our body and physical health are crucial to be able to work well as developers.

In this talk I speak both about the science behind fitness and nutrition, and my personal journey of losing over 70 kgs, starting to go to the gym, how it affected me as a developer, as well as the shocking secret behind what happened to the Sound of Symfony podcast.

So welcome everyone. Um, I'm here to talk about how fitness makes you a better developer. And just before we start, there are going to be some sensitive topics discussed in this talk. Um, mental health, specifically depression and anxiety. Uh, there's also going to be a talk of obesity and I'm telling you so that if you feel you need to brace yourself mentally, you can do that. Or if this is not something that you're comfortable with, I won't be judging you if you leaving this talk, besides both the talk that Nicholas has and the talk that Matthias has seems awesome. So I can't really blame you if you do. Um, but, uh, my name is Magnus Norlander. I've been a Symfony developer since about 2007 and I run a small consultancy in Stockholm. I'm mostly focusing on Symfony. This is not my first SymfonyCon. I've actually been to all of them except for one.

Uh, for example, in Paris in December 15, I, uh, gave a presentation. Um, I was also in Berlin and back in December, 2016 I didn't talk at that conference, but I had a great time. Uh, back in 2017 I wasn't that SymfonyCon that was the one that I missed. Uh, but I did manage to go to SymfonyLive in San Francisco. This picture isn't from the conference. It's from, we recorded an episode of sound of Symfony there, uh, the now pretty much defunct podcast. Um, and yeah, I mean, clearly something happened. There's a pattern in those pictures. So let's take a closer look at, uh, to one of them and, and a more recent picture to compare. As you can see, I got a new haircut. Um, Joke, jokes aside

I, uh, I, I lost a bunch of weight. Um, and um, I also want to add a couple of, you know, caveats to this presentation. I'm not a physician. I'm not a nutritionist. I'm not a neuroscientist or a personal trainer. Um, it's just lost a bunch of weight. I'm a developer. So, uh, if you need to consult somebody, someone professional, I'm not that person. Uh, but speaking of neuroscience, we're gonna put a pin in my journey and start somewhere else. The brain. Um, since I assume most of us here are programmers and even if you're not, the brain is a pretty important tool to everyone. Um, and I don't know about you, but I like learning about how the tools that I use work. So that's what we're going to do here today. This is a picture of a brain. Um, and like I said, I'm not a neuroscientist.

Parts of the Brain that effect fitness (Hippocampus, Amygdala & Frontal Lobe)

I can't go into every detail about every part of the brain, but we're going to take a closer look at a few interesting parts, um, that has to do with fitness. So I'm going to focus on three different parts. The hippocampus, the amygdala, and the frontal lobe. The hippocampus is a fairly small, or actually there's two, because you have one in each half of the brain, but it's a, it's a part of the, um, um, of, of the temporal lobe. Um, and it plays an important role and consolidating short term memories into longterm memories as well as spatial memory and emotional regulation. And it's named for the seahorse shape it has. Um, the next part would be the amygdala. It's there. I didn't find as nice a picture of that one, but, uh, it plays an important role in memory processing, or actually there's two of that one, again, like I said, two parts of the brain.

Uh, it's also located in the, uh, in the temporal lobe. It's about the size of an almond. And, uh, it plays a primary role in processing and memory decision making and emotional responses like anxiety or fear or aggression. And for lack of a better term, it's one of the more primitive parts of the brain. Uh, and by that I mean that it's a part of the brain that's, that evolved pretty early on, around 200 million years ago. And it's shared not only with all mammals, it's also shared with reptiles. And fear is an important part of the brain. So that's why we've kept this around, uh, because fear keeps us from doing recklessly dangerous things that could kill us, which would be bad. Um, next up, the frontal lobe, which as you can see from this picture, is a pretty big part of the brain. Um, and in contrast to the amygdala, it's a relatively recent addition to the mammalian brain.

It's also, it's very large. It has many functions because it's such a large part of the brain, but most of the functions are, are

higher functions like logical and abstract thinking or executive functions, which is stuff like planning for the future, judgment, decision-making, attention span or inhibition. And something interesting about the frontal lobe is that it can modify the emotions derived from the amygdala, whereas the amygdala gives us the, you know, the, the fear anxiety, those sorts of emotions. They're modulated by the frontal lobe to be expressed in a more socially acceptable way. Like perhaps if you're giving a ah, a presentation, you don't run away because it's, it's scary to be on stage. Um, and this balancing act with regulating the, the brain parts, the keep regulating each other is something that I want to talk about that people here watch. 'em Chernobyl, the TV show.

There was a great scene in there where I'm with Dr. Legasov of explains how an RBMK nuclear reactor balances the different parts of the reaction in order to create a stable one and just like a nuclear reactor, the different parts of the brain to balance each other in order to keep us functioning emotionally and unlike a nuclear reactor, the a in the brain, this happens through hormones and signaling substances. And we're going to talk about some of those two. First up, there's a substance called serotonin and all of these substances have complex, multifaceted roles, but very simplistically, serotonin contributes to a feeling of calm and inner strength. Uh, there's also dopamine, which is part of the reward system of the brain and it plays a key part in motivation. It's also strongly connected to addiction. Um, there's GABA or gamma aminobutyric acid, um, which reduces neural excitability, which means basically that it gets in between the neurons, slowing down their reactions and that calms us.

There's another substance called BDNF or brain derived neurotrophic factor, which is a protein that supports the survival of existing neurons and encourages growth of new neurons or Synopsys. Um, there's endorphins or actually endorphin as a contraction of endogenous morphine, uh, which is because it's a morphine like substance, but it's produced in the body that's endogenous. Uh, and just like the regular kind of morphine, um, it inhibits pain and it can induce a feeling of euphoria. Uh, endorphins is if you've heard of the runners high, that's, that's endorphins. Um, and finally there's a cortisol, which is a hormone that's associated with stress. Cortisol increases our heart rate. It also increases our blood pressure and generally it makes us ready to act on some sort of perceived danger. Uh, and we're going to look a little bit closer at that. Um, specifically cortisol and stress. Cortisol is regulated through a system called the HPA axis.

HPA is an abbreviation of hypothalamus, pituitary, and adrenal gland. And whenever you're in a situation where the brain perceives danger, the amygdala starts signaling the hypothalamus, which in turn, um, through a hormone signals to pituitary, which in turn signals the adrenal gland to start releasing cortisol into your, your bloodstream, which again, like it raises your heart rate and your blood pressure and everything. Something that's kind of interesting is that the amygdala itself actually responds to cortisol, um, by activating it even more. So if it was just the amygdala and the HPA axis there was involved, you would spin out of control into pure panic pretty, pretty quickly. Um, luckily the amygdala is balanced by the hippocampus and by the frontal lobe. Um, and

the, the HPA axis and the, the activation from the amygdala can sometimes misfire in the, in the modern world. Like if you're giving a presentation and you're going to, you're going to start being stressed. That's, that's a natural thing. Uh, and you're going to start releasing cortisol and making you ready to, to run away or to fight, which is perfect. That's just what I need right now. Uh, not necessarily useful. Um, so a big part of this presentation is, is going to be where these things, uh, stress. For example, humans have always had stressors, but over the last century, the characteristics of those kinds of stressors have been changing a lot from how am I gonna get food today? Or how am I going to run away from this bear, uh, to, uh, how am I gonna meet this deadline coming up in six weeks? Or, uh, what if the interest rates go up?

Um, we've gone from a lot of acute stressors to longterm stressors and that's not particularly good. Um, because like I said, stress reactions are caused by cortisol. Cortisol inhibits the hippocampus and the frontal lobe making. And like I said, hippocampus is responsible for memory formation. So, uh, longterm stress makes memories more difficult to form and it also starts affecting our emotional stability, [cough] excuse me. Um, it, um, it sort of gives the amygdala the sort of upper hand and this emotional balancing act and longterm elevated cortisol levels are actually toxic to the hippocampus, making it shrink and breaking it down physically. Um, exercise on the other hand is a good way to combat this because exercise stimulates the frontal cortex, a part of the frontal lobe and adds connections between the frontal cortex and the amygdala sort of strengthening the control over this, this balanced reaction and from, um, from, from the, uh, the frontal lobe.

It also increases resiliency, against stress, uh, basically making it so that in, if you're in a stressful situation, it takes more of it for you to sort of get into that fight or flight mode. Um, and it also makes your brain better at lowering cortisol levels once the threat is gone. Um, and um, stress is also deeply connected to anxiety and whereas stress is acute, uh, short term anxiety is basically what happens over the longterm. Uh, when you have stressors, both are controlled through the same mechanism, the HPA axis and um, it's been shown in scientific studies, uh, that people with anxiety has overactive amygdala, overly active amygdalas and sort of, it would sort of make sense then that strengthening this connection that the giving the frontal lobe more control of our stress reactions, uh, would help with anxiety as well. But that's not the only way that exercise helps against anxiety.

Um, because when things happen in the brain at the same time, um, those, there's a saying called neurons that fire together wire together. So things that happen simultaneously in the brain are going to be wiring together so that if one of them

happens, both of them will happen. The a typical a situation where you, where you could see something like this as PTSD, where if, say you've been in a war zone and that's an that that's definitely an acutely stressful situation and you hear, um, sounds of like loud exploding type type sounds, uh, your brain is going to start associating those sounds with anxiety so that if you start hearing those sounds, even if it's not an stressful situation, you're going to be getting stressed. Um, what happens with exercise though is, and this happens, uh, because stress increases your heart rate and blood pressure, um, that gets associated with stressful situations, but there's also other situations which increases your, your heart rate and your blood pressure and that's exercise.

So by giving, uh, like a second way of having those, those experiences, you start to unwire those, um, uh, those connections. So that getting a high heart rate is probably not going to be asked anxiety inducing. Next up mood specifically depression. Um, this is also connected to stress and anxiety, but it's not directly affected by the HPA axis. I mean, it's all connected, but it's not the same mechanism. Uh, we know that mood disorders like depression are linked with lowered levels of serotonin, dopamine, and noradrenaline. Um, and if you're looking at calling mood stabilizing medications like SSRIs or selective serotonin reuptake inhibitors, um, those work by longterm racing the levels of, um, and of those signaling substances in the cases of an of SSRI, it's serotonin. And, um, there is something else that also raises your, your serotonin levels longterm. And that is exercise. Uh, and some studies have shown that for mild depression, exercise is as effective as, uh, as mood stabilizing medications. And unlike mood, stabilizing medications and exercise habits will actually give you longterm resiliency against mood disorders. Okay.

Memory also important for developing things. Like I said earlier, cortisol is toxic to the hippocampus and a good way to balance this is exercise because exercise stimulates BDNF production, which helps protect or even grow the hippocampus, allowing longterm memories to be more easily formed. And it's more difficult for them to get broken down.

Cardio Experiment

And if we have the time, which I believe we do a, I'm going to tell a short story about a scientific experiment on scientists in the U S took 120 people. They measured their hippocampus size via MRI, MRI machines, and then they split them into two groups. So group A regularly did cardio and group B did calmer activities like stretching. And after a year, uh, of this, uh, they measured again for group B, the results was pretty much what you would be expecting. The hippocampus had shrunk by about 1%. That's entirely normal. The hippocampus typically shrinks by about 1% per year. Um, but for group a, the one that had done cardio, that was not the case. The hippocampus had actually grown by about 2%. And, um, this isn't the only study of its kind. There's been other studies that have correlated exercise with better memories. Um, but the, uh, the protection and growing of the hippocampus is the likely mechanic. Why that actually works.

Focus is strongly connected to the reward system of the brain. Uh, basically you'll only stay focused as long as the brain considers remaining focused to be the most rewarding behavior. Uh, and this is signaled with dopamine. The brain wants dopamine and if it thinks that you're going to get more dopamine by checking your phone than by keeping doing a presentation on fitness, you're going to have a hard time focusing on the presentation. Um, dopamine also helps tuning out the background. So if you take a moment to listen around, there's an air conditioning system or something running in here. Uh, I'm sure you didn't notice that before I pointed it out because that's basically being filtered. Uh, that's being done in the thalamus and other part of the brain. And this filtering of the, the, the, the function of the thalamus is caused by dopamine because that noise is not important, you didn't notice it.

Um, so the thalamus acts as a filter for the unnecessary information that your brain has access to, but you don't really need. And having a well functioning thalamus allows you to focus better. Um, Oh, and did I mention that dopamine is is increased over long time by exercise? It is. Um, it's also key for creativity because having a finely tuned dopamine level keeps not only the noise out, but it also lets convergent and divergent thinking, which is basically what creativity is. So it makes sure you're not listening to unnecessary stuff, but you're getting just the right amount of new ideas to actually be creative. And creativity is less understood than for example, focus. But it's been well tested. Uh, studies show that exercise correlates strongly with increased creativity.

Cardio

So which physical activity should you be doing? Well, I mean you can basically, basically you can split up any physical activity into two kinds, uh, or probably more. But in this presentation I'm going to be focusing on two kinds, either cardio, which is basically anything that gets your heart rate up. Uh, ideally you would looking at something like 70% of your max heart rate, uh, or above that would be considered cardio. This could be running. Uh, if you're out of shape, walking is definitely going to do this too. It certainly did for me. Um, it could be biking or like those cardio machines that you have at the gym or it could be something more advanced like high intensity interval training and cardio is what's going to be driving most of the changes to the neurology of your brain. Um, so it is important if you want to have these sort of um, these sort of advantages conferred upon you.

So how often? Well, at least two times a week, preferably three to four, uh, and for at least 20 minutes per time, preferably 30 to 40 minutes. And it's important to keep at it because while you are going to be seeing some, some positive effects well actually a lot of positive effects immediately, there's going to be a fair number of them that's going to take a few months before they start like before you start noticing them.

Lifting

So in addition to cardio, there's also resistance training or lifting, which is something that I enjoy very much. Resistance training is basically what builds muscle and it shapes your physique. It's not as important for the brain, but it's not unimportant. Because see, muscle muscle tissue actually helps your body filter cortisol. So that again contributes to cortisol control and yeah, you should totally still do it. Um, so for resistance training, the important part to focus on, at least if you want to build muscle mass, is progressive overload.

That is making sure you increase in some way, either by how much weight you're lifting or how many reps you're doing, um, or even just slowing down and doing slower reps, uh, yes to, to make sure you're continuously challenging your muscles. Because with resistance training, you're basically breaking down your muscles and then the body repairs them, but it repairs them in a way that they're stronger. So you want to keep breaking them down. If you're just doing whatever you're able to, you're not going to be breaking down any muscle tissue. Uh, you want to go at least twice a week. Uh, if you're doing, if you're doing twice a week, you'll be doing full body workouts and I would recommend, uh, learning to use free weights because they're safer. Uh, you could totally do like body weight exercises as long as that's still a progressive overload.

Um, but machines are more prone to injury even though they might seem easier to use. Um, so it's definitely worth the time and to, to learn how to do the, to use free weights and to do like the big compound movements. So what about some other stuff? I mean, exercise is great, right? But what are the options? Uh, medication. So I want to be perfectly clear. Medication works and if you need it, you totally should use it. Physical activity doesn't replace medication. And if, for example, you're depressed, it could be difficult to find the motivation to start exercising. But that being said, if you are able to start exercising studies show that combining medication with physical exercise tends to give better results and it tends to give better longterm results. Um, because medication and physical exercise, uses some of the same pathways in the brain.

Mindfulness

Um, so yeah, also mindfulness, that's a fun thing. And I wish I could talk more about it, but I don't have the time. Uh, it works too. Uh, it's not as effective as exercise. It works basically because you, it's gonna help you like rewire your brain so that you're not getting as many. You're not getting as anxious over stuff that you shouldn't be getting anxious about. Um, and if you're more interested in, if you're interested in mindfulness, ask me after the talking, I'll be happy to talk to you about it. Um, what about diet and because we're now, now I'm done talking about exercise. Really I wanted to take this opportunity to talk a little bit more about diet because while it's not exactly fitness, it does affect your abilities as a programmer. Uh, but there is a connection, which is how I'm making this totally smooth SIG segue.

Diet

So, so, um, yeah, so smooth. Um, poor diet increases cortisol levels, uh, and excessive food intake in relation to your activity, uh, causes obesity. And I just wanted to give us insight. That's a subject near and dear to my heart. Uh, wanted side note on that, obesity is an epidemic. It's affecting 16% of adults in 2016 globally. It's even higher in industrialized nations. Um, the prevalence of obesity has tripled between 1975 and 2016. It's definitely not without problems. It causes cardiovascular diseases. It, um, it's a major cause of diabetes type two or metabolic syndrome. Um, it also causes osteoarthritis. Um, it keeps you chronically inflamed and it multiplies your risk for certain types of cancers by a lot. It's also important to note that adipose tissue, which is fat tissue, is an endocrine organ. It actually affects your hormones. Um, and um, like I said, it correlates with increased cortisol levels.

It's also super easy for obesity to start causing sleep disturbances. For example, I, I was never diagnosed with anything, but I'm very sure I had sleep apnea. I didn't realize until after I'd lost weight how poorly I slept. Uh, there's other ways of doing that and you can have sleep apnea without being obese, but it's, if you can avoid having sleep apnea, that's a good thing. Don't, don't have that. It's, It's not good. Um, and obesity doesn't just stress the joints and cardiovascular system. Like, I don't know why I've mentioned this three, three times now. It increases cortisol levels in the stressing the brain, which can totally turn into a vicious cycle because cortisol also increases fat storage. And if you happen to be someone who tends to eat their feelings, which I am, that's not a good combination because if you're eating your feelings, you're going to be more obese, which increases, which, which, which keeps you producing more cortisol, which means you eat more of your feelings, which means you gain a lot of weight.

So I'm not sure how many of you sort of know because I know I certainly didn't really understand how this worked before I started, um, before I started losing weight. Um, how the energy balance of the body sort of works. Eating food provides you

with nutritional energy and the body uses energy to do like stuff walking around or talking or even just existing. It uses, it uses energy because your heart is pumping and your thinking and your body is generating heat. And if you take in more energy than is used, it's going to be stored somehow. And if you use more energy than you've taken in, that energy has to come from somewhere. It doesn't come out of thin air. It comes from that storage. The body can store energy in several different ways. It can store it in as blood glucose. That is the fastest, most accessible way of having energy stored.

It can store it as glycogen in the liver or glycogen in the muscles, uh, which is also pretty accessible or it could store it as adipose tissue or body fat. It can also use the energy to build stuff like muscle, which it can technically break down later to, to use for energy. But that's not something that the body really wants to do. Um, and to maintain weight, you should eat as much energy as you use to gain weight. If that is a problem, you should eat more than you use, uh, or to lose weight. You should eat less than you use. So how do you know how much to eat? Well, basically there's three ways and the first kinds, sort of the, the, uh, the, the obvious one following from how energy balance works. If you're maintaining your weight, whatever you're eating is how much energy you're using, otherwise you would be either losing or gaining weight.

Bench Marks

So if you log what you're eating and regularly weighing yourself, you can calculate what's known as your daily and your total daily energy expenditure by just summing it up and averaging out. And there's apps to do that. I'm going to be talking a little bit more about that later. Um, that takes a while though because you have to log what you're eating over a longer time. I wouldn't recommend doing it for less than a month. Um, faster but less exact way is to use a formula like the Mifflin St Jeor formula to estimate what's known as your basal metabolic rate. Uh, that formula takes your age, your gender, your height, and your weight, I believe, um, to to estimate how much energy your body is using to just exist. Um, then you can take, uh, what's known as a physical activity factor and multiply that by your basal metabolic rates to get your total daily energy expenditure.

Um, the, the physical activity factor is just basically based on how active you are. You get a number that you multiply the BMR with to get the TDEE. The third option is measuring it, which sounds really good, but it's, it has practical problems. The, there are tools that are super exact for measuring your basal metabolic rate, like a metabolic chamber. Uh, but you probably don't have access to those. There's not a lot of them, um, in the entire world. And the tools that you do have access to aren't actually measuring your metabolic rate. They're just estimating at using a formula like the Mifflin St Jeor formula activity bands like an Apple watch or a Fitbit or whatever. Um, they do measure your activity so they can get a decent estimate of your physical activity and physical activity factor. But the error margin, those are not great. And like I said, they're still using a formula for your basal metabolic rate. Uh, if you have the time, actually logging in weighing is super good, but you might want to get a ballpark estimate using a formula.

Dieting

So are calories, everything that matters? Uh, well no, there's also macronutrients which you've probably heard of. It's fats, carbohydrates and protein, and we all need fats and protein. Um, carbs aren't strictly necessary, but it's good if you want to be able to perform physically. And I could totally go very in depth about macro distribution and what, which macro distribution is good for what. And if you're interested in that, come talk to me afterwards or ping me on Twitter. But I, until then, I just want to say a couple of things. So the normal diet that's recommended by stuff like the, uh, well, sort of any nutrition agency is going to be about 50% energy from carbohydrates, about 30% from fats and about 20% from protein. And that's a good start. Um, if you're looking to build muscle or if you're losing weight and I want to maintain the muscle mass that you have, you probably want to add more protein to that or actually rebalance it so that you, uh, you eat some more protein than that.

Um, and there's a bunch of exclusion diets. I'm sure everyone here has heard of keto, which is basically that you eat barely any carbohydrates and that kicks you into ketosis. Um, but for weight loss, what really matters is how many calories you're eating versus how many calories you're using. Even if you're in ketosis, that is that you're using fats instead of carbohydrates as your main energy source, unless you're actually at a caloric deficit. There's too much energy already coming into your system from your diet. So that doesn't, it's not necessary to take it from, from your fat stores. Um, but what I would just recommend is to find a way of eating that that works for you, that you feel that you can sustain. And if that is eating, eating a low carbohydrate diet, that's, that's fine. Um, but it's not something like that. The important thing is energy balance.

And now, now we have everything, right. You have the energy and we have the macronutrient nutrients. Well, no, we also have micronutrients, uh, which is pretty much vitamins and minerals. Um, so what do you eat? Well you can get, you can get a long way with just common sense, like mostly cook your own food, eat plenty of vegetables. You all know this. Don't eat too much of the things that everyone knows aren't good for you, like deep fried foods or sweets or excessively fatty foods. Um, that takes you a long way. Um, you just have to actually do it. That's the difficult part. It's simple but not easy. Um, if you're, if that isn't accomplishing your goals, it's super useful to log your food intake and there are apps to help you do this. Like my fitness pal or Lifesum, uh, where you just, you have, you can basically search for food items and you add them to your daily log and you weigh them and that makes it easier.

Um, but it's also important to remember that what's important isn't what you're doing once a month or once a year. Or like if, if you're at a conference, that's not what's important. It's important what you're doing every day. So like not caring, particularly what you're eating at, say, new year's eve. Totally fine. Not caring, particularly what you eat every other day. That's probably not gonna fly. Um, and, um, I sort of just want to talk a little bit about my story and, uh, there's caveats for this one too. You are not me. I hope. Um, not everyone needs to lose weight. Um, I, I did, not everyone does. Not everyone is two meters tall, surprisingly, because I thought that's such a nice and round number that that should probably be the average. It turns out it isn't. Um, also, I didn't exactly do everything right either. So, um, uh, yes, do as I say, not as I do.

Magnus' Journey

Um, I was always pretty big. I didn't become obese until my twenties when I started moving away from home. Um, and I was never very active. Um, possibly because I have exercise induced asthma, which I only found out about like a year ago. Uh, getting medication for that super helpful, super helpful. Um, but when I started starting attending a university, uh, stress from school where I suddenly had to apply myself, that's super weird. Uh, and the reduced social control of what I'm eating because I was living alone or I was living in a, in a, a, you know, a student housing thing, but I'm pretty much living alone. That caused me to eat more because I tend to eat my emotions. And when I was stressed, I ate to feel good. Um, I wasn't happy. I was especially not happy with my body and I had a lot of anxieties about my body and my health.

Um, thinking about that or doing anything about that would make it feel more real and um, that would trigger more anxiety, not, not great. And I never really had a single rock bottom moment, but I, I, I'd like say I had three, three pivotal moments, which I would like to talk a little bit about here. So in the summer of 2017, I joined Mensa and it's not important that it is Mensa, uh, but it's a social Mensa. Basically. It's a social organization. It's meetups and drinkups. And yeah, I made a lot of new friends, became more social and that sort of became, it sort of made me a happier person. Um, I, I was never diagnosed with anything. I was likely depressed before, but I never went to see a doctor because going to see a doctor would be anxiety inducing and I did not want to do that.

Um, but occasionally I did like challenge my anxiety somewhat. In December, 2017 I read an article about insulin and diabetes and that article, I don't like to actually link to it because it turns out that's probably not a correct article. But the take that I was getting from it is that if I started eating less carbs, maybe that would prevent me from getting diabetes. That getting diabetes was a major, major fear of mine. It was pretty justified to be honest. But, um, because I had some of the symptoms of pre-diabetes, um, but I thought this is something that I can do. Like eating less carbs is not difficult. I like eating vegetables. I don't need to drink lots of sugar drinks. Um, I can do this. And so I did, I started changing my diet, uh, eating less carbs, um, incidentally eating less calories overall and I started losing inches off my waist and that sort of kicked in the realization that I actually could lose weight.

Initially. I didn't think I could. Um, I thought that weight loss is not something that was understood, which is easy. It's easy to get that sort of a, that sort of impression from seeing all of these weird like pineapple diets and these crash diets where you're supposed to eat nothing at all and then you gain the weight back. And like, I thought that people didn't, people didn't know how to lose weight. It's no one knew. But by doing this, I sort of started understanding that actually maybe I could lose all that weight, which was very motivating. So by spring of 2018 I started exercising more specifically. I started going more for walks. I started closing the rings on my Apple watch. That's pretty motivational. Um, and I bought a scale that I couldn't use. Um, because at that time I, I still don't know what I weighed.

I know it's over 180 kilos because that's the maximum weight for that scale and I was over that. Um, so surprisingly enough I took that in good, good stride because, um, I just thought that, well maybe if not now, at least I'm still losing weight, obviously. Um, I'll be able to use it sometime. By summer of 2018, I set a calorie goal and I started sticking to that and by August I was under 180 kilos and actually started to be able to use my scale. In September I got a gym membership mostly because Sweden is a cold country, so I don't want to be out walking or running in the winter. Um, so I wanted to be able to do cardio in doors, but as I started going to the gym, I also started lifting. Uh, which I found was fun. Uh, cardio was not something that I thought was really fun initially.

Um, but uh, lifting was, that was fun from the beginning. So I also increased my calorie goal because going to the gym surprisingly uses energy. Uh, and in April, 2019, I hit my goal weight of 115 kilos, which is when I started switching over to maintenance. That was not as smooth as it could have been. I would not recommend like going from a steep deficit directly to maintenance. It's probably better to like taper that out so that you're used to how you're supposed to be eating at a higher calorie limit. Also some kind of fun stuff. I was in an article on men's health. I did not expect I would be in an article above Jason Mamoa. That's a, that was pretty cool. Um, so all in all I went from about, I'm guessing 190 to 200 kilos to 110. Um, which is pretty much, I didn't weigh myself today, but that's pretty much where I'm at.

Uh, I'm eating a lot better. I'm now working out every day. That's a relative. You don't need to be working out every day. Um, but that's a, that's a relatively new thing. I do that because I enjoy lifting. Um, I sleep better. I have normal blood blood pressure now. I did not, when I was overweight, I had very high blood pressure. Not good at all. My resting heart rate is around 50 beats per minute, which is pretty good and I'm also happier and less stressed. Um, I'm going to go ahead and add

another, um, another before and after picture. Um, and something that I want to leave you with is this, I'm not special. Like you can do this. If you need to lose weight, you can do it. It's simple, not easy, but um, what you should do is you should focus on sustainability both through exercise or diet.

Lessons Learned

This, this particular slide applies to regardless of if you need to lose weight and you start exercising, focus on a sustainable habit both for exercise and diet. Don't go all in because it's so easy to like go all in and I'm going to exercise twice a day. And then you do that for half a week and then you stop, like start small and scale it up over time as you feel like you can find an activity that you like. For me that was lifting and later running running initially was sort of the least objectionable form of cardio to me. Um, but I actually found, I sort of started enjoying that. Um, and routine is super powerful specifically for weight loss. I would like to add that it's not a race. You don't need to do this fast. You should plan for maintenance. Also for me specifically, I'm a volume eater so I need to go for foods with a low calorie density that is like need a lot of it with not too much calories.

But drinking your calories is not, that's not a good idea for anyone. Don't do that too often. Also weighing yourself is good if you want to lose weight, but you should not be looking at like the day to day value. It's better to look at moving averages because your weight tends to fluctuate based on water or how much salt you had the day before or whatnot. Um, also in the, in the description for this talk, I promise to reveal the dark secret about what happened with the sound of symfony podcast. I just didn't have time. I started going to the gym. I mean, that's more important, and that's pretty much the end of this presentation. If you have any questions, we don't have time to take them now, but you should talk, you can talk to me at the conference, or you can contact me on Twitter or the symfony Slack and, uh, yeah, that's, uh, that's it. Uh, thank you everyone.

Chapter 17: PHPUnit Best Practices (Sebastian Bergmann)

Tip

SymfonyCon 2019 Amsterdam presentation by Sebastian Bergmann.

Talk slides

While PHPUnit is not difficult to set up and writing tests with it is easy, you will get better results and save development time if you know the tips and tricks to leverage PHPUnit more effectively. This session, presented by the creator of PHPUnit, teaches best practices you can use to ensure that your unit testing effort is efficiently implemented.

Transcript & caption for the talk will be added soon.

Chapter 18: Using API Platform to build ticketing system (Antonio Peric-Mazar)

Tip

SymfonyCon 2019 Amsterdam presentation by Antonio Peric-Mazar.

Talk slides

Why is API platform a way to go and the new standard in developing apps? In this talk, I want to show you some real examples that we built using API platform including a ticketing system for the world's biggest bicycle marathon and a social network that is a mixture of both Tinder and Facebook Messenger.

We had to tackle problems regarding the implementation of tax laws in 18 different countries, dozens of translations (including Arabic), multiple role systems, different timezones, overall struggle with a complicated logic with an infinite number of branches, and more. Are you interested? Sign up for the talk.

Hello? can you hear me? Yeah. Cool. Hi. Morning. How are you enjoying the conference? Yeah. Cool. Uh, first I want to thank you, the entire Symfony team for inviting me to speak here. Uh, I did many of conferences. This one was on my bucket list for awhile, so I'm really, really happy to be here. Also, thanks to the sponsors for making this happen. Uh, before I start, I want to get to know more about you. So first, who is using Symfony 5 in production? There is a guy, okay. Who is using API platform. Okay. Like 30% of the people. Cool. Uh, so my name is Antonio Perić-Mažar. I'm from Croatia from split. I'm the CEO of Locastic. I'm also co founder of the Litto, it is not software development agency. It is different industry. And we also, two years ago co found a Tinel Meetup, which is in our industry.

And every month we bring one foreign speaker to our home city to do of course free meetup, with beers, pizza and hanging around. Uh, we are very proud about that. Just few words about company. We are doing bunch of Symfony backend based projects. We are also doing a lot of user experience projects working with the banks, telecom operators. We are not like big company only 20 people, but we are like really do, I would say good things. Uh, what is our, what is the context of my token API platform experience?

Experience

So one of the biggest projects that we did, which is based on Symfony and API platform is ticketing platform for GFNY organization. That is the franchise business that is running in 26 countries and supporting our users from 82 countries in eight different languages including Hebrew and Indonesia. We are having that in production for a year and a half serving approximately 60,000 tickets per year.

The complexity of this project is not like high traffic or like high availability. It's more like in domain that the main is really, really complex on the front end it's React with Redux on the backend is fully Symfony with API platform. With API platform. We also did some social networks chat based, and some matching similar to the Tinder. I assume that some of you know what's the Tinder and also with a few like enterprise CRM, ERP applications so that that is something that I want to share some parts of our experience in this talk. When I planned the talk I was like aiming to talk only about this ticketing system but actually when I start writing on a paper I noticed there is like bunch of things that are repeating so I pulled a few of them and I will try to show you as best as I can what what we are doing and how we are doing the things.

What is the API platform because only like 40% people is using the API platform. I will just do quick introduction to the API platform, so if we want to trust Fabien Potencier air, that's the, that's the most powerful API platform in any language in the world. It is based on Symfony. It is dedicated to API dream projects. It contains PHP library to create fully PHP features, APIs, supporting industry standards, providing some Javascript libraries shipped with Docker and Kubernetes integration and it's now Symfony official API stack. It's very, it's very powerful tool containing many feature like simple creating crud, data validation, pagination, filtering, sorting, hypermedia, GraphQL support, whatever. Basically whatever you need to build modern APIs in matter of minutes. So just for example, I want to show you how simple it is to create crud in literally for the seconds.

So basically the first thing that we need to do is create some model, some entity, regular entity in a Symfony. So it has a name, it has ID, nothing, nothing special. The second thing is that we need to map that to the our ORM system and we need, if we want to have some validation and the only thing that we need to expose this as a resource, as the API resources to do

one line of configuration. I'm using yaml. You can use XML you can use annotations, you can use whatever you want, but this is the only thing that you need to do. And you are getting fully operational crud for API, fully operational API resource with beautiful documentation. You can play it, you can test it here it, it works perfectly. So what we are getting, we are getting two types of operation that's collection operations and that's item operations to our mandatory GET on a collection and GET on the item.

Others can be removed or modified or whatever. Now this is the as as we don't have controllers, we don't have anything here. We just have configuration for the API resources. We have bunch of extension points that we can use to build our business logic. So the best thing about API platform is that it's not focusing you to think about framework itself, it's you, it's you are focusing even more than with Symfony 4 on building your value, business value for your users. Okay, so at the top of the end is the kernel events. That is regular Symfony events. But suggestion is to use other extension points because kernel events works only with the rest APIs and other extension points will work even if you are using GraphQL as your, as your APIs, uh, it is using action domain responder pattern which is like better alternative for HTTP comparing to MVC how it works, so user GET like sends something to the URI which is requesting some action. Action asks domain to do some logic and then responder or gives respond to the end user.

If you divide that in the API platform, we have separation like this on the left side in the action part we have operational resources and actions that we did with configuration. Then we have data providers, data persister and other things which contains some kind of hard domain logic and then we have serialization response which is actually responder with the things that I show you as the extension point. You can basically do the changes in any of these free parts. We have also serialization group divides in two contexts. One is normalized context. Second one is deserialized context. Why this is very powerful. It's because in this way with very simple configuration, we can have different read write models without even writing the custom code for that so we can normalize a context. You can specify which fields are you requiring. For example, for creating or updating the resource and with deserialize you can say what you want to expose to the user.

Okay, so how, how the regular project looks like with the, with the API platform and a Symfony. The things that I will talk today, it's more like about big Symfony projects which are using the API platform as one part of the application. The one part that is actually communicating with any part, with any client's application. It can be your watch, it can be your fridge, it can be your single page application, it can be your computer, it can be your car, whatever you want. We are exposing API to the API platform. We are getting some amazing thing to implement our logic. So today you will see that and our projects, we are quite, I know that we are at the conferences talking about a lot about decoupling from the framework, but I think that some points we are missing to talk how to leverage the framework.

Like for example, in our case, the fully decoupling from the Symfony is not something that you are doing because in the future we will never move from the Symfony even more maybe from the PHP but not from the Symfony. So leveraging the framework, it's a huge advantage for us, especially if we have nice structure of the projects and how we can do that. So first advice that I have for you is for configuration. Use the yaml if you have a longterm project. Did this configuration can be quite extensive and I know that annotations are really easy. Nice to start. But imagine this having inner annotations, this is what I selected. It's the configuration for only one resource. This can be even improved this file, because we can separate this in few files, which will be yaml files of course. But imagine this having in one huge annotation with bunch of other things.

It can be really, really messy. I personally use annotation for a demo because they will save me a bunch of time, but if we are doing like longterm projects, we prefer yaml. You can use whatever you want. This is just my advice.

User Management & Security

Okay. First thing that every application has is users. So let's see. Few tips and tricks, how you can do user management or security. If you have listened to the security talk yesterday, probably some of the things will be familiar to you, but there are few touchpoints that I want you to have intention to that. So basically if you are using API platform, you don't want to use FOSUserBundle. I assume that you are all aware of FOSUserBundle but FOSUserBundle. It's not built for REST. It's built for some quick user management and it brings a lot of over head with itself.

As your project grows, you will have more and more problem overriding the things that are built in FOSUserBundle. For example, I'm working at the moment at one legacy project and I spent the days removing some parts of the FOSUserBundle, finally I'm clean so I can, I can change it. What you can use, it's actually Doctrine User Provider which is included in Symfony 4 so it's simple and easy to integrate. Then you need to, only thing that you need to do is bin/console maker:user. You will get fully working user with everything setup in security and the only thing in rest API that you need to implement some action. For example, registration action. So this is how it looks. It needs to implement UserInterface and I set up already some some groups, some from writes some from read, so normalization and the denormalization context.

Second thing is that I need to map that to my ORM and database and then I need configure the resources, as you can see here, this is a little bigger configuration because I set up the normalization context and denormalization context. Okay, so the next thing is what I want to do, I want to hash password. When I'm saving, when I'm storing user to the database, I want to

implement registration. The only thing that I need to do is create UserDataPersister. In the UserDataPersister I have the method which is the second method which is actually the first after, after the construct which returns do I support this method for this entity? If it is true it will do persist or remove. Depends what we need to do and in the persist method what I'm doing, I'm hashing the plain password, removing the plain password, saving User to the database.

I can send the email here, I can do whatever logic I want so like I am putting some custom logic in a place in the API platform. What we are using mostly for for for security for a login is JSON web tokens can, which are like standards that are lightweight and simply identification system stateless and they are storing, token to the browser, local storage and then can be authenticated. This depends on the, from the project from the project. For some projects you will need or too or something else. But this works for, for for example, for this ticketing system, this works perfectly. It works very simple. You send request to the server with your user name and password. Server signs the token returns this to you and which every new request you are sending that to the server. What is very important thing that I want you to pay notice here is that insight.

When we are decrypting this encoded string, we are actually getting the data about user. We can get any data that we want to start by. For example, we will store email. Why this is okay for our, just before I tell you why this is very important for of course for this, for this authentication, we have two bundles which are LexikJWTAuthenticationBundle and JWTRefreshTokenBundle. JWTRefreshTokenBundle is to use after your token is expired to get a new one. So basically after the login you will get a token and refresh token. Token will sign, we'll have some short time to live. Uh, also one very nice thing too that a lot of developers miss from the documentation is that we have user checker and security component and this can save a lot of time for you. Basically the Symony is using some default UserChecker, but you can implement your own just implementing this interface and then the only thing that you need is put one line of the setup in security.yml.

In this way you can do some additional check. It can use any fields that you want for a user, like is it expire, is it blocked, is it deleted, did I ban it or whatever you want. It gives you very, very good flexibility and two points. Preauthorization and PostAuthorization. Resource and operational level on on API platform are quite powerful to set up. So basically what this configuration says, it says that if you want to access to the resource book, you need to be at least the role user. You need to logged in. If you want to create new book, you need to be role admin. If you want to update the book, you need to be owner of that book and you need to be admin. Okay. And what's even powerful is the using voters. You all know what the voters are. Okay, cool.

So the same way that you are using voters in any Symfony project with this slide, two lines of the configuration, you can use them in a API platform projects.

JWT Tip

When I told you that to pay attention on a JWT token, it was about this for example, you're usually your user is stored in your database. You have DoctrineUserProvider, but sometimes in some projects it can happen that your user is stored to some third party API. And then communication with that third party API is not like all this possible or it's like too, too slow. For example, three seconds. You don't want users to wait every time for three second. So what's the nice thing to do? It's actually to create database last user. Since we have email inside JWT token, we can, we don't need to authorize user to load user every time from the database.

We can just authorize them from the email because we can trust to the JWT token and this is very, very nice thing to have to have in mind how to do this. It's again just the line of configuration. We need to set up the default JWT, lexik_jwt provider and we need to say our firewall for the API. Can you use that provider for the user all done. The only thing that you need to have in mind is that if you want to get user information, you need to create that manually. You cannot do token get user. Okay.

Multi Language API

Creating multi language API. This is something if you are doing like big projects usually today you will need to create a multi language APIs. API platform is not supporting that out of the box so when we got this requirement we need to find some solution about it.

What we actually did as we are doing a lot of it to Sylius and are also doing amazing work same as, API platform's people. We took the idea about Sylius translation, how, how they are working there and implement that with the API platform. We created open source bundle, publish it to the GitHub and also there is blog post explaining how you can use that. It's, it's actually quite easy what you need after you install the bundle you need to extend your entity that you want to be translatable with the AbstractTranslatable and add first method which is createTranslation that method. It's loading the translation entity that we will implement later. And the second thing is that we need to implement a translation relation to to our uh, to our PostTranslation entity. Of course we need to create some virtual fields which will be used to get single language query and then we need to implement translation entity which will contain all the fields that we want to translate.

After we did all of this, we need to set up like few lines of configuration. We need to set up the translation groups for

normalizing context and we need to set up translation groups for some filters or whatever we want to use. And how we are using this. It's quite easy when we are creating a new resource we are sending all the languages that we want to create so for example in this example we are creating English and Dutch, sorry Germany and we are sending all the files. We see that we have local, local code inside. If we want to get some language we can query only one language. Then we will use this virtual properties and we will get like only title and the content for the English language. If we want to get all the languages we need to say to send dynamically groups, translation and we will get all the languages listed.

Second thing that you need to figure out how to solve when you have multi language API is static translations. There are two solutions. One solution is to use LexikTranslationBundle which stores translation to the database but you can export it to the files. We wanted to have solution that writes directly to the file so that we don't need to have some cron jobs or manually dropping the translations to the, to the files. So there is a guide also on our blog how it's, it's too much code but it's really easy concept. So if you want to learn how to write your own transplantation, you can go there and check it.

Manipulating Context

Uh, next thing that I want to talk is manipulating contexts. This example that I have on a slide is purely from the API documentation, but this is very important thing that we have in API Platform.

Why it is important. It is important because for one of the basic examples is that we don't have, we don't need to have two separate APIs for admin and for the regular users. So we can have the same access points, but based on their roles we can have different responses. How this work, again we have some API resource here you can see annotation configuration and then we are adding serializer groups. So you can see at the active that we have book:output and we have admin:input and on the name we have book:output and book:input. So what we are doing, we are creating a service, which is our BookContextBuilder. We are decorating API platform, serializer context_builder and we are building our custom logic inside and this is very simple concept. Again, we are just checking what we are doing like is are we talking about book resource, do are we doing denormalization or normalization is user logged and do they have admin role? If they have just check for the context groups and add group admin:input period, that's it. You will get a different response for user and for admin user.

Symfony Messenger Component

Oh, Symfony Messenger component please. Who's using Symfony Messenger component. Okay. Not much people but like a lot 20%. Symfony Messenger component is a very simple component to start using it. It allows us to have like communication between queues, between different application, easy to implement, easy to use, making asynchronous communication easy. It just works in very simple way. You have sender, you have handler, sender sends a message, handler takes that message, do some logic. We have some bus transportation between that. We can have some middleware, but you can read that in documentation. Why I'm talking about this because with Symfony messenger and API platform, we can have Command Query Responsibility Segregation or CQRS pattern. It's very easy how it works. Basically you just need to set up configuration. Again, you will say a message that we are using messenger, that output is false for example, and response on this action will be 202, like status. Okay.

Then the handler will do other things. We'll do heavy, heavy things. This is the simple thing. I want to talk about more, more advanced thing. Also you can use that in the image in ImageMediaDataPersister for example, data persisters skier, like if you have image that you want to resize, you can dispatch a message than some Lambda or cloud, whatever can do the work, return the image or whatever. Uh, also you can use it in events, but as I said, like try to use other extension points. This is the reason why messenger is very important for me. If you are talking about, this is the project that I'm actually working on. It's fully hosted on a Google cloud platform. It is event sourced, distributed base. I don't know. Architecture. Okay, so what we have in the blue is the thing that is actually hosting our Symfony application in the yellow and orange are cloud functions from the, from the, from the Google cloud platform and Firebase and other things.

We have a bunch of the pub and sub services communicating between and the API platform is this small red.here. Literally API platform exposing the rest API to the, to the client's app and everything in the behind. As I stated in the beginning of my talk is Symfony, so basically Symfony is always like more complex part than just API platform. So if you're talking about Symfony and the messenger, if you read documentation it's always sending the message to some application and then that same application is consuming the message. That works perfectly. But usually that's not how the things work. Usually you are at least communicating between two different Symfony applications only or even in, in better case you are communicating with some third party applications, nodeJS, GO applications, whatever and Symfony works perfect in first case, it works not so good in second case it doesn't work at all out of the box.

In the third case actually you need to do, can I say out of the box? It doesn't work just with configuration that you set up in your Symfony application. Why? This is how Symfony dispatch message it looks like. We have body which actually contains our message. Then we have a bunch of headers which are describing which handler will be used and some which command bus, blah blah blah like a bunch of configuration that is needed in our Symfony application. If you dispatch this message app

message CommunicationMessage to some other Symfony application, there is huge, there is huge chance that you are not using the same namespace so this won't work if you are dispatching the message from the node JS application for example, which was my case, you are not having properties on the headers. You don't have the body which is string actually contains escape, escape JSON and properties and the headers are proper JSON. You actually have this, we will have some JSON like no description, nothing. Then me and colleague, developer and a team, we start working and he was dispatching the message. I was getting errors, errors. I start researching and then the simple solution was can you add the headers and other things that we need and at the first was like, huh, no, but that's the only message that we are consuming at the moment from the other side like, okay, let's do that. That's the quickest way, but that's not like sustainable way. We want to do that better. So how you can do that is actually you can write your custom ExternalJsonMessageSerializer what that message JSON serializer do. It's actually receives your message, decodes from the string to array and then based on some parameter, if you have it in a message, it can checks what is happening, what it is, what it is receiving, and then can create the message that your SymfonyMessengerHandler needs so that it's possible to do.

Of course you need to have some parameter which you can, which you can catch on this point. We were lucky enough so we'd have this key based on the key I know that's testing or communication or caching or whatever and I have variables which I know that they are body of my message. Again, you need to create your message as I said you, we need you. We'll return that. That will work. Also you need to set up some configuration because in in a default configurations Symfony Messenger is using the default Symfony Messenger, deserializer second solution that is also possible is that using the HappyMessageSerializer why I didn't put the code for a message serializer because it has the dependency you need to have body property in a JSON and with this custom serializer we can even outweigh that. Also there is a blog post Symfony Messenger on AWS Lambda that Tobias wrote. I think so, yeah. Also have in mind that messenger component is similar to ORM components. It will work in most of the cases, but if you have some like really, really specific cases, you will need to build something by yourself at some point or at least like really hard customizations.

Handling Emails

Handling emails with new Symfony Mailer component that is easier than ever. Like we have a bunch of emails that we are sending bunch of notifications in, in, in all, in all applications that are mentioned at the beginning. So how to do is like read the Symfony Mailer component. You have really easy integration with Symfony Messenger. So that means you can send them asynchronously. You have also support for the load balancer. So if you have like three or four different transports for the emails, it will spread them like with the round Robin method, uh, it's also high level availability.

If a fail over, if something fails, it will go to second then the third transportation, for the email, it supports out of the box. It supports Amazon, SES, Gmail, MailChimp, Mandrill, Mailgun, PostMark and SendGrid. And that's for me personally, that is the best thing that happened with this component because usually setting up the emails was pain in, really painful. So yeah, that is something if you are not using, check that it's much better than just Swift mailer that we had like a few years ago or even a few months ago.

Creating Reports (exports)

Uh, creating reports and exports. When you are doing ticketing systems, that is like something that you do on a daily basis. What we had as a problem was importing data from the few different sources, storing that in a database, doing some transformation and exporting in a total different format to our users, so how we did the importing first we set up the crawlers with the Symfony Messenger component.

We crawl all the data that we need. We could not do transportation on a fly because there was too much relationship between all the data that we are getting, so we need to store that to the database. Then do transformation. Then again store that to the database to get it in a format that we want to use. Now that when we have the format that we can store in data base that we can read that we can do some operation with that we needed to export that to CSV files. It's the same if you are doing XML it's the same if you are doing XML files, what you need to do is actually create custom operation for export. You need to change the format to the CSV. For example, in this case you need to say what is DTO Data Transformer Object for your output in this case is OrderExport class, which group we will use for in this case OrderExport and that in this case is a POST method.

I got the question why it is POST not GET? The POST because when we are creating something we always use the past and in this situation we are creating some document that does not exist. So that's our logic. Again, you can use GET if you want. Next thing is that you need to create this DTO object. It's simple model. Sam as you have model for for reading and writing to your database by entity. You have this DTO here which contains bunch of properties, get and set methods so you can go with the public that that depends on you. A second thing is that you need to implement DataTransformer, DataTransformer again is a simple class that do only one thing. Transform data from one format to some other format. Not, not a format, wrong word. Like literally from one object to another object and doing some like changes on, on the way how we store the data.

So what we have here, we have export on the left side, which is created in, ah at the beginning of the transfer method. And we are setting the values and transforming if it is needed for exporting to the uh, to the, to the end user. What is very important here if you didn't, uh, if you didn't notice this pagination is enabled false. Why it is that because in the report we want to have all the data that we have in a database. By default we will get 30 of of that. And that's it. That's how you do exports. How you do reports in any format that you want. You just need to create objects that will contain data and export it to the, to the users.

Real-time Applications

Real time applications with API platform. Who, who is using or who heard about Mercure. Okay, cool. So in like, this is like what we can do now easily. And what are the real time application? What the real time applications are. So like if you do update on the web, it will be updated on all mobile devices that are connected together. In the past there was a way to build real time applications, but usually that was like some, let's call it weird solutions. It will be ready with NodeJS combination that they will be subscribed to PHP will publish to the Redis, Node will read it, and then to the web sockets will send it to all the clients that are connected. There was Pusher solution like hosted solution, ReactPHP did something. I think they did actually a good job there. But to be honest, the PHP is not built for real time applications.

PHP is built for request response. We all know that. So what actually changed here is that we have this Mercure HUB which is created by Kevin. Kevin did a lot of amazing work with the Symfony, with the Mercure. If you listen yesterday. So basically what we are doing after we create something, we passed that resource to the Mercure hub. And then Mercure hub sends server-side events to all clients that are connected to Mercure hub. It cannot go in opposite directions. So it's not a duplex communication as web sockets, but in 99% cases this will be solution that will work for your project. So if you are doing chat application, if you are doing push notifications, that will work. Now this is just a short description. It's written in Go, it's based on, it's automatic to HTTP to HTTPS, it has CORS support, Cloud Native, it's open source and bunch of other things.

This is very interesting. It works very easy out of the box with API platform. So basically you only need to say it's Mercure true on any changes that you do with your resource. It will be automatically sent to the Mercure hub and it will be, if you're, if some users are connected, they will get the data. This part of the JavaScript is the freelance of the code that you need to subscribe to the mercure and get the data. Uh, once the event happens.

Testing

If you are doing a big applications like this, you need to write a test. So what what we learned during during this applications is that for example, we don't believe in 100% coverage of the tests because that usually will means that we will lose a lot of time. But we believe in smart testing like we want to test the critical cases, we want to test any bug that happened.

We want to have like really good structure tests. Also for legacy code we have policy like if we got some bug or something we will first cover that with a test and we will resolve that. And also like if you are not very experienced, maybe TDD will be hard at the beginning for you. So just start writing the tests. But I want to talk a little bit about API testing tools. First tool is that you have it out of the box with with API platform is HTTP client in API platform which which manipulates the Symfony HTTPKernel directly. So it give boost of performance and you will have much faster tests. If we are comparing that with the test over the network. Also it's good to consider another tool which is APITestCase. They are quite similar, I would say from the, from the Sylius project and in API test case you get a lot of helper methods that can save you a bunch of time like testing is is JSON equals something?

Is JSON contains something is this, is this matching the schema that we defined for the resource for example book, book object. A second very useful thing is PHP Matcher because if you are creating dummy data you don't want to create data like Antonio surname Peric, SymfonyCon. You want to just check is this string type is this integer type is this date is this array contains something that is possible with the PHP Matcher and this will save bunch of time for you. There is plenty of methods inside that so you can basically check any type of expression that you have inside your response. If you are setting up the new project and you don't have a data Faker, which is, which works with AliceBundle under the hood, it's very, very useful tool to create dummy data which looks real data. Kevin used that yesterday for, for his demo at the end of the talk.

So the data were created with the AliceBundle. Also you can use Postman test. Postman tests are very nice but they will be slower than API test case both from the Sylius and from the API platform because the Postman can later be used as ah interactive specification of your API for your front end developers. The tests are written in NodeJS but they are very, very simple to write and then you can integrate them with the Newman, Newman in your console or in your CI and just run it before deploying to the server to check is everything okay with uh, with uh, with your, with your tests, with are API, sorry. Also some tools for checking test quality like Infection, PHPStan, Continuous Integration, other things. I think you are listening a lot about these tools. So I will just mention them and yeah, use them.

They are good tools. Okay.

Last slide before I'm done is actually the picture that Fabien Potencier tweeted a few days ago about the book he's writing

and about architecture of the modern modern application built with a Symfony. So the applications that I, that I mentioned today that I described some parts of how they work are actually built in this way. We have a bunch of models which are integrated to work together. Some models are communicating directly to the API inside the code. Some are communicating to the message component, depends on the things, what are they doing and the API, the API is like I'll say API is only one part which is exposing that to the user but also it give us like really nice extension points where we can put a bunch of our logic. Usually of course in this kind of applications you have a lot of console command applications which are doing a lot of background job as a workers or as cron jobs or or whatever. Uh, one last thing, like conclusion API platform and a Symfony especially Symfony 4 are really awesome tools and I think we all should be happy that we have that good framework in PHP and that we are using that as a part of our everyday work. Thank you.

Chapter 19: Make the Most out of Twig (Andrii Yatsenko)

Tip

SymfonyCon 2019 Amsterdam presentation by Andrii Yatsenko.

Talk slides

Twig is the most powerful templating engine in the PHP world that enables us to create highly complex projects with hundreds of multi-level extended templates, thousands of blocks, functions, filters, tests, and macros. It also offers a sandbox, a unique but not a widely used feature that is creating secure user-editable templates. In addition, there are a number of handy built-in and external debugging tools available in the Twig ecosystem to simplify the day-to-day work process for a Twig designer.

In this presentation, I will talk about how extensively we use Twig in a complex open-source e-commerce project.

Transcript & caption for the talk will be added soon.

Chapter 20: Mental Health in the Workplace (Stefan Koopmanschap)

Tip

SymfonyCon 2019 Amsterdam presentation by Stefan Koopmanschap.

Talk slides

Mental health is an important part of life, yet mental illness is big. Perhaps bigger than you might think when looking around you. In this talk, you'll be introduced to some basics on mental health and mental illness, and given some tips and handholds on how to handle mental illness on the work floor.

Transcript & caption for the talk will be added soon.

Chapter 21: Importing bad data - Outputting good data with Symfony (Michelle Sanver)

Tip

SymfonyCon 2019 Amsterdam presentation by Michelle Sanver.

The role of our API in Switzerland is to consume a lot of data that was not meant for a digital age and to transform it into beautiful output, for one of the biggest retailer in Switzerland. This is a journey of consuming a lot of data and APIs from different sources and in different formats. Some of them made us laugh, some of us got migraines. We built a smooth architecture to consume and output data. I am proud of our architecture that we seamlessly upgraded to keep the latest versions, now Symfony 4 along the way. I want to share with you how we managed to keep this API up to date for over 5 years and the architecture that we use to make it happen.

Transcript & caption for the talk will be added soon.

Chapter 22: Symfony Serializer: There and back again (Juciellen Cabrera)

Tip

SymfonyCon 2019 Amsterdam presentation by Juciellen Cabrera.

Talk slides

When developing APIs you sometimes need to return the same object with different representations. Furthermore, an object can be complex due to its relationships with other objects, making the serialization process such a hard task.

The Symfony Serializer Component makes it possible to manage how you want to serialize objects for JSON, XML or other formats. Also, you can manage responses using serialization groups, choosing which properties you want, handling serialization depth and much more.

Transcript & caption for the talk will be added soon.

Chapter 23: Eeek, my tests are mutating! (Lander Vanderstraeten)

Tip

SymfonyCon 2019 Amsterdam presentation by Lander Vanderstraeten.

Writing tests is nice, but how are you sure that your tests cover all use cases? Code coverage can give false positive metrics. A new way of working that goes by the name of mutation testing has gained a lot of popularity lately. This talk will explain you what it is, how you can integrate it and contains a demo over the basics of mutation testing with infection and phpspec.

Hello. Good morning everyone. My name is Lander Vanderstraten. It's an unpronounceable name for who is not Flemish or Dutch. I'm, I'm currently a PHP developer at PHPro. I'm working for one of the biggest online gambling casinos. I'm also a maintainer of Grunt PHP, which is a really nice, nice task runner. It's very useful for people who would like to run, for example, coding styles or unit tests before you hit commit. I also have strong interests in the domain driven design, CQRS, event sourcing, and micro service. But before I start, I would like to ask you free quiz questions.

Tests will have bugs

Who thinks, unit tests are a waste of time. Oh no, no. It's still early. But you can get out. Who always writes unit tests. Please, some more hands. Okay. You guys are awesome. It's important that your unit tests are always up to date and like I said, you can use very nice tools like Grunt PHP to make sure that every developer runs their unit test before every hit commit. Are there people who already know what mutation testing is or use? Okay. I see some, some hands over here. Well, eh, this still is more of an introduction, but I still that you guys will learn something today. All right. We are mostly all developers over here. And what we do is recreate software. We are aware that we write bugs all the time. Nobody writes bug free code. So smart developers, as we all are, we write unit tests and these tests are code. So your tests will have bugs. Tests are a good tool to verify your code. But what does verify your test?

Code coverage is a start

Like Sebastian said that this morning, a code coverage. It's great. It's a good start. Most developers try to reach up the 100% percentage. It's the same but 100% code coverage. But are you rethinking that your application is now 100% bug free? No, sorry, it's not good. Code coverage is just a measurement of how many lines that are touched during your test suite. So still it can give a good score with useless tests for developers who are writing these tests to get up until that's 100% code coverage. So this is what good coverage looks like in real life. All the tests passed. Every letter here is inside the letter box, but it's just at the wrong place.

Mutation testing

So how can we now detect, well bla mutation testing. So I will start by explaining a bit more but reading this definition. Mutation testing is a technique to evaluate the quality of your tests by programmatically mutating and making a small series of modification to the code with the goal that your tests no longer pass. Oh, this is a really nice show on discovery channel. How do they do it? Well, let me explain this with a basic example. If books are crimes and your tests are the police mutations are fake crimes to let people see that the police is doing their job. So a bit more technically is to assess the quality of a given test. Mutants are executed against input test to see if seeded faults can be detected.

Wow. All this terminology, mutation, mutants, WTF? So each mutated version is called a mutant and a mutated program with failing tests is a killed mutant. And that's what we actually want to achieve as I said in the definition. A mutated program with passed tests is an escape mutant. There is a modification been done on this, on the code, but still the tests are passed and that's not what you want. So let's measure all these things. Test suites are measured by the percentage of the mutant that they killed and new tests can be designed to kill these additional mutants. So with mutation testing, it's another metric. Than code coverage, it's called a mutation score indicator and it's ideal to integrate this in your CI environment. For example, you can say that we have a threshold percentage of the score indication that we want. And when we add new code, developers add new code, they used Grunt PHP to have these tests running before they git commit. You can check that the threshold is reached, for example, up to 100% that all these mutants are killed every time.

How does it modify? Well based on a list of mutators.

An example is that this is a basic function foo bar, which has an integer and outputs an integer. And for example, we increment this number. Well during our tests with mutation tests or mutation testing framework, it will change it to the increment operator to a decrement operator and check if all the tests will fail or one of the tests will fail. So this is good. Another example, for example, we try to see if the object foo is nullable. Maybe you have written a bad test and it's not nullable. This is a small modification it has been on during these infection or a mutation test suite.

Timings

Now what about timings as Sebastian this morning explained code coverage significantly increases during runtime when you apply your code coverage during your unit tests, especially in big project, we can add a mutation framework on top of it, but the timing will increase exponentially since we have this huge list of mutators, for example, the increment decrement, or nullable, or an equal to, not equals to, all these changes, they run the unit test suite every time over and over again. And this takes a huge time.

Demo example

So enough with the talk, let's go right up to the demo. Good. This is all working. Before I start, I would like to show you a this demonstration project uses PHPSpec instead of PHPUnit. Both testing frameworks are great. I use infection as mutation testing framework and I use the PHP code coverage extension on top of PHPSpec to have this code coverage. It's configured like this in the PHPSpec YAML file. Still quite small, but it's, you know, don't like this also infection. You configure your source directories? You can also exclude folders if you want during your, your tests. You can also have false positives. You can exclude them here as well. And I will now continue by my unit tests that I've prepared. The goal is that I will create a function that has an integer as an argument and it can it should not increment nor decrement, the value zero. So let's start with this one. I also have here just the class, this is the class I would like to write a unit test for. So this foo bar function, when it has the value zero, it should be zero. When we also implement this and we run our tests,

Yay. The first unit test is written, we still have two tests to do.

So, it's also should increment a positive number. So this foo bar of the value let's say five should be six. Now when we run our tests, this is incorrect obviously. So we change our implementation by checking if the number is zero, then we return zero, else we do number plus one. The second test did. Now I will also do this for the decrement function. Yes. So let's say a minus 10 should be minus 11 and then we have, if the number is smaller,

Zero return number minus one

We have successfully created a unit test with PHPSpec. On top of that, we use code coverage and we have a code coverage of 100% no bugs in here. So let's run our mutation testing framework. Now on top of it, we didn't achieve here. 100% well, this is an example where we can change our code since our tests are fine and this is a workaround. So we can either exclude this one or we can just change it

To this. I did a mistake. Number menus one. There we go. Yep. Like this.

So when we run our tests again, everything is still okay. We didn't make any major change, we switched those statements. But when we run our mutation framework on top of it, it's everything is good, well I forgot to mention something here. So let me go back to the initial setup. So sorry about that. Here we have the escaped mutant. Why? We can see it in our infection log file. And here we can see that's on line 15 of our foo bar class. It actually tried to change greater than zero to greater than or equals to zero. But since this is not a case in our code, it can never reach it. Since our first statement is already checking for equals to zero, we we can, you can change it upwards to make sure that everything is working correctly. And now you see that every mutant is killed. Now are these tests actually testing something, no they are not. Why? For this very basic example, there are some tests missing and this is what I would like to call the boundary or the limits. And the first positive number is one, and the first negative number is minus one. But what if we change our coat and let's say free,

These tests are still green and I've done a modification to the code. Obviously in this case this is this has no meaning, but it can happen that you have these huge functions with unit tests. There are actually not testing something. And another developer makes a change and thinks, Oh, the tests are still green, but actually it's not. So let me put this back to zero and add as well the first positive number, that should be two, as well as the first negative number should be minus two. I will now demo it again with the value of free. We will see that one of our tests failed. Now with infection. This is actually a mutator. I currently haven't configured it. That's why it's failing right now as well as the unit tests is failing so they can get correct again, that's about it for this brief demonstration. So, what did we actually learn today?

What did we learn today?

This guy is Peterson Ted is really famous cook in Belgium and at the end of his shows he always asks the candidates a few questions about what did they learned? Well, what did I learn from this is always write unit tests. Writing unit tests is never a waste of time. Make sure that your whole team keeps them always up to date. Next is good coverage can give us a false positive feeling. But mutation testing gives us a slightly better quality indication, but it's not the Holy grail that will solve every application that contains no bugs anymore. So it's another metric that you can use that you can integrate in the CI environment. Still it takes a lot of time. So most developers would not like to run them every time in a git commit on their machine. So that's where I would like to place them. And that's the end of my quick talk. If there are questions, feel free to ask them. And thank you very much.

Hello? Hi, I'm is who feel extra tight costs of I will ask in English. Yeah. How many, how much extra time does it actually cost to run this mutation? That's a lot. A lot really. I, I haven't any metric included in this slide, but let's say that projects with these big unit tests that with code coverage take five minutes, it can go up to 20, 25 minutes. So it increased heavily really on time and that's why a referred to it to include it in a CI environment since we as a developer will not wait to get to it until it's finished.

Thank you.

Questions

Hey, thanks for your talk. Thank you. Yeah, I had a quick question. So the configuration that you showed for the mutators it was for the whole project. Would you recommend that you would have specific mutator configurations per unit test or per unit test suite because obviously it has a exponential effect on the runtime or, or would you say that the whole project has just one big configuration for the entire mutators?

Yeah, that's a tough question. I would recommend to you is the complete list, but this takes so much time. Maybe you can be aware of these mutators that you will check them yourself. But I would recommend to use the whole list actually that you make sure that every tiny mutation is tested. Okay. Thanks.

Hey. so the way that I understood this is that every, a mutation is a shingle single change somewhere and then all the tests gets run. Yes. Is that necessarily so, because why? Why don't we try and apply a number of mutations and run the tests and that way try to improve the overall time that this takes because sure, it will not give you like the exact check to this one change led to this test failure. But if you get a list of like these 10 changes led to these five test failures in like a fifth of the time, then that may already be very helpful to you as well. That's true. That's true. But maybe, maybe you should ask this also to one of the maintainers of a infection, but how do I see it? Is that they want to do one modification and run the complete suite about over it to make sure that this particular mutation is seen by at least one test. I think that's the point of it, but honestly, I don't know. It's more in depth. I'm sorry. Okay. have a great day and enjoy the conference. Thank you.

Chapter 24: Integrating performance management in your development cycle (Marc Weistroff)

Tip

SymfonyCon 2019 Amsterdam presentation by Marc Weistroff.

Good performance is crucial for online businesses and it should be taken very seriously. While this is common knowledge, it is usually not prioritized in development cycles. This leads to higher development cost as teams found themselves in situations of fire-fighting. By integrating performance management early in the development process, the cost are reduced, chances of performance regression are lessened, same as the business risks. Using Blackfire, we will see why performance management should be integrated in a SymfonyCloud project early in every environments from development to testing to production.

Transcript & caption for the talk will be added soon.

Chapter 25: Demystifying React JS for Symfony developers (Titouan Galopin)

Tip

SymfonyCon 2019 Amsterdam presentation by Titouan Galopin.

Talk slides

The world of Javascript is vast and exciting... but it can also be quite scary! It is an innovative world in which new technologies appear all the time, which can make it difficult for developers to keep up to date. One of the most famous of these technologies is probably React JS. It changed the way we conceive User Interfaces, both in the browser and in mobile applications. But understanding it can be challenging. Let's demystify it together so that you can enter the world of Javascript with confidence!

Transcript & caption for the talk will be added soon.

Chapter 26: Head first into Symfony Cache, Redis & Redis Cluster (Andre Rømcke)

Tip

SymfonyCon 2019 Amsterdam presentation by Andre Rømcke.

Talk slides

Symfony Cache has been around for a few releases. But what is happening behind the scenes? Talk focuses on how is it working, down to detail level on Redis for things like datatypes, Redis Cluster sharding logic, how it differs from Memcached and more.

Hopefully you'll learn how you can make sure to get optimal performance, what opportunities exists, and which pitfalls to try to avoid.

So quite a bit. So I'll just do a quick recap on what it is for those that haven't. Ah cache tagging, what that this anyone using that already? Yeah, there's a few, not that many of us with some, uh, there are quite a lot of Symfony applications using it under the hood, you might not actually know it, um, shortly on Memcached versus Redis just uh, for awareness Redis and Redis cluster what it is. Um, and some issues we had over the last year in the Redis adaptor and what ended up with the RedisTagAwareAapter and then we'll do, uh, I'll do a short demo, um, just to show how you can add caching to your application. Nothing fancy there. And then I'll go through some more advanced edge cases to be aware of if you do this and also where the new adapter fits to the old one.

About Me

Um, so me, uh, working for eZ Systems, uh, been doing that for a long time. Try whenever I have the time to contribute to the Symfony FOS. Uh, especially FOS HTTP cache. Um, well only a Composer, long time ago. Um, PHP-FIG long time ago, Docker Compose, um, eZ is a kind of a global system, small company, 75 people spread around with the community and partners. Uh, beyond that, uh, we make, um, we make a CMS, uh, today called eZ Platform in the past called eZ Publish. You might have heard of it. It's a very extendible, very feature rich, flexible ah it can be used for headless or full, uh, full, uh, use. Uh, we've been on Symfony since 2012 and, uh, we're actually in the middle of uh, um, getting a new version out soon with either, either on 44.4 LTS or 5 or both. We haven't really decided, um, we have commercial flavors of that eZ Platform Enterprise and Commerce, which adds additional features.

Cache in eZ Platform

Um, so in our case, we started to use Symfony cache back in 2007. Uh, we used a Stash if you've ever heard about that. Um, pull that out, put in Symfony cache, uh, to instead use Symfony, uh, sort of the cache tagging there because we wanted to take a different approach to how we clear cache. If you are aware of a Stash, it has something called a hierarchical cache. So we can for instance, in for instance, say I want to delete locations slash and it will delete all locations. Uh, if that's an entity, uh, if you have a blog, you can delete all blogs, but if you have something that goes across all these trees or hierarchies, then becomes a bit more, uh, difficult. And you also end up clearing way too much cache in some cases when this gets complex. So after moving to that, we found some, some issues. So going back to that, and that's all sort of already mentioned, we in the end ah decided to contribute optimized tag adapters for Redis file system. So quick recap on Symfony cache for those that haven't used it yet. Uh, PSR6 compliant cache adaptors plus plus. So it adds additional features on top and needs to be fast. Um, I can sign on that compared to Stash. It's definitely faster.

Um, it's also progressively being used all over the place in Symfony itself. So you'll probably see new places being used for that.

Symfony Cache Adapters

There are a lot of adopters right now. This is maybe not even complete, but there's APCu a PHP Array in memory chaining several using Doctrine, um, existing adopters from them. There's FileSystem, there's PHP file and PHP Array using Opcache for those cases where you have cache that never changes. So, um, uh, there's that. And then there's a proxy for using other PSR6 caches. There's Redis and there's Memcached. And finally there is something called TagAware. So this is kind of a

special kind. This is not a backend per se. This is something that wraps, uh, a backend. And to go a bit into tagging, um, what do we use that for? How do we use it and to try to be generic? You can say, depending on an application, it can be a entity type, it can be a placement somehow. If you have some kind of placement in your system for your entities, variant type in commerce systems maybe. So a lot of different things depending on your system.

And where this is relevant is if you have operations in your application that affects, uh, quite a big subset of those. Not the whole thing, but definitely a part like the change to a variation affecting all the variations in that specific kind of variation, for instance. So affecting bulk of entities but not all. So about tagging cache, uh, you can basically directly invalidate on this here. Jumping a bit to an eZ example, we have the key, uh, for eZ content. In our case we have, ah content can be anything, articles, blogs, whatever. So there's ID 66, there's type two. If, um, and that's a article, I think a location 44. So tagging where in the content tree we place this so that if there is an operation on location 44 and everything beneath it, we can say to the cache, please clear this, please clear path, actually 44.

TagAware Example

So to jump to PHP code and look at how this looks like, actually it's quite simple. There's just one method added to what PSR6 provides. There is invalidate tags but of course in the detail there, it does imply a lot of concepts. It doesn't imply a secondary index, it does imply keeping tracking of this and does imply setting those tags or well if you want. Um, as mentioned it wraps your normal adapter and stores the tags in a separate key which you don't have to care about. It's completely internal. Um, it does, when you look up your key, it will do a separate in parallel lookup for to know which tags you have added to this cache. And after knowing that it will do a second look up or third, I'll show you in a second to to actually know if any of those have been validated. So if we look at actually output from web profiler who will see it first, do one round of lookups, to the backend, it will look up what you asked for. It will look up for which tags is assigned to this. And then it will look up all the tags you have assigned in this case. Um, it's a timestamp. So in this case there's false, there's nothing set yet. I just booted up the application. But if there were, you know, relations, it would return a timestamp.

We'll return a bit to the new adapters around this. But first, what about Redis and Redis Cluster? Anyone here uses Redis or maybe specifically Redis Cluster already. Okay. Anyone uses other things like from Netflix or other solutions?

Yeah.

Redis

Um, so starting with Redis, it's not Memcache, it's way more advanced. It has, um, data types. So it has a lot of different data types. It's not just string. There's lists, sets, so list within a key, sorted sets, hashes, bitmaps, hyper log logs, which are not going to try to explain it this evening, streams, which is a complex talk in itself. Uh, there are, um, several talks about it. You can find online if you're interested, quite recent future. And on these data types, there are tons of operations, so it's not just get, set and almost that's it. There is tons more, if we talk about string, there's GET SET, APPEND, BITPOS, Decrement Increment and so on. If it's an ink, no, int value in the string. Um, there's also a generic, um, cluster commands for doing specific things. There's a transactional.

There's um, also something called the pipeline for doing several in one call. Uh, and for SET, which is what we use in RedisTagAware there is add to this app. So add a member, remove a member, pop the last member, uh, dif when you have two sets, uh, in do intersections, unions, moving members from one set to another. So there is a lot of advanced operations that you can do on this. You can use this, uh, for something cool in your application without it actually being a cache. It can be a storage. I'll get back to that. Um, and also, um, if you still compare it to Redis and now to Memcached, it allows you to control much more what happens on a lot of different cases. One important one is what happens when it reaches max memory. So when it reaches maximum memory, can ask it if it should randomly clear things, if it should, uh, respect TTL, if it should respect a last, um, rather than, um, recently used object or frequently used object. Uh, and you can choose if that should only be on those that have a TTL or everything, those even that doesn't have it.

Redis Cluster

So onto Redis Cluster, um, this allows you to scale Redis by running several instances. You can also run several on the same server. So if you want to handle more load than you have a lot of spare CPU and memory, you can even do it per server.

Uh, what it does for you is coordinate a cache across cache slots, which is within each server and deal with the replication and also figure out who is the master. So if something goes down, it will communicate with the different servers, elect a new master in the background and deal with this itself. Um, thinking back to all the different advanced options I talked about earlier with Redis, there are some limitations when you use a cluster unlike Memcached. But of course, uh, it, it supports everything Memcached supports also in cluster mode. So talking about limitations, you can, for instance, not do a pipeline

that would, um, have to go to one server.

Uh, in the case of our clients are the one, most of the use, the PHP Redis one. It mainly supports doing several operations at once using simpler commands like MGET and MSET. Um, and to give an example, uh, if you do a rename of a key from one name to another, this might result in the following, uh, exception cross slot keys in request on the hash to the same slot. There is a way around this I'll show you later. But, um, it's things to be aware of. You need to kind of work around those things.

Memcached vs Redis

Shortly on Memcached versus Redis, whoever uses Memcached. Okay. Where do you used to use it and move to Redis? Yeah, yeah. There is a lot of those, including me. Um, they're there. It's good to know what the difference are because actually Memcached has its strengths also compared to Redis.

Um, to, to talk about them here. I already talked about the strengths of Redis. You have the data types, you have the controller eviction, you have persistence, you have the pipeline Lua capabilities. Um, and there is also a bunch of other things, but those are the ones I felt were important. Um, and then when it comes to multi server, uh, you start to see the difference because um, Memcached is by default meant for running on several servers. While Redis you have to introduce a different thing, right? Redis Cluster or other technologies to run it across several servers. So that's one difference. The second one is multithreading Memcached does multithreading especially in the last five, six, seven years. Um, while Redis today, today it's a single threaded. So this means if you want to take, as I mentioned earlier, take care about extra load, uh, and want to take care about and use a lot of CPU, CPU, of course, you would need to use Redis Cluster to spawn additional processes to take advantage of those.

Redis 6 does come with some background threading capabilities but it's more for the slow operations. Okay. So having set somehow some uh, some background and some context here. Let's talk about the new adapters. Um, there's two of them. FilesystemTagAware and RedisTagAwareAdapter. What they do on the high level is basically instead of like the TagAwareAdapter, where it does several look-ups, it moves tags to be like a relation. So instead of using the expiry time and looking up that it will store it as a separate thing and not have to do a lookup on it, this means it has a one round trip to the server instead of two.

In the case of file system adapter, it uses a file for tags. So kind of like a upended the file every time there's something added, appending to it. If there is invalidation, I think it moves it and then reads it, clear all the keys in it. Uh, in Redis, um, TagAwareAdapter, it was, um, it is using a set, so storing that as a relation and it uses it with out expiring. This is, um, one of the tricky things with this adapter it needs to do that because uh, if the tags are suddenly evicted before the cache itself, if you clear it by the tag and that has been evicted, you basically end up with stale cache. So clear limitation may be I'm not that up there. So all this efforts to try to do one, um, one round trip lookups? Uh, we have cases with customers where they're on Amazon or something else and there are often quite some latency to reach the cache backend if it's Memcached or if it's redis.

Uh, it can be the same. What we had from customers was some were in the range of 0.2 to 0.5 milliseconds per lookup. Um, and if you are on a small instance of that Redis uh, or the ElasticCache, it will be even slower. So simple page, no problem. That might attribute to something like five to 20 millisecond of the total, uh, time spent to generate that page. But um, us, me and Yonnie in the front there, um, coming from eZ we make a CMS, there's like complex news papers or something that our customers tries to load, you know, cases like this or worse where there's like thousands of articles or different kinds of content being shown on one page. All of them need some look up to the cache and this starting to attribute to a lot of the load time, uh, unless the page is cached by Varnish. So whenever varnish isn't caching the page just to look ups to Redis or memcached will attribute to up to one second.

Symfony Cache

And that's just with one Redis instance, it was in some cases worse. So to talk briefly on the optimizations that's been done in, in Symfony Cache over the last year, we first figured out that it was using pipeline. So when you move to Redis cluster, it wouldn't have to then do, if you were asking for five things at the same time, it would do five calls. So instead of using MGET, it ended up being a single GET per in a foreach for each and everyone. So back and forth ping pong to the Redis server. So this was fixed.

There was also a small thing around having to do versioning of cache in Redis Cluster, this, uh, added additional lookups to know which version the caches is on now. And then if you clear everything, the version will be bumped. And this wasn't needed anymore. When Nicholas did some other, uh, improvements here and there and he fixed this one. He fixed a lot of things, uh, also on the TagAwareAdapter, it's, it's way better now. It does some micro TTL caching. So if you do a lookup on those cache to know the expire time, it doesn't have to look that up every single time during your request. So that's a great improvement as well. Um, but a lot of the improvements you need to deal with is actually going to be on your application.

eZ Platform Example

So in our case on the eZ Platform side, we did change this to make sure we could take advantage of looking up several things at the same time. Instead of us having our API where you load one thing at the same time and our, our users are using that, we expose more APIs, to load several things at the same time. We took advantage of it ourself in the cases we could and we started to like encourage our users to use it also. Um, we also then more importantly introduced, um, RedisTagAwareAdapter and maybe what had a biggest impact for us. We introduced application specific in-memory cache, um, kind of discussing Nicholas about trying to build something generic for Symfony around this. But it's, it is application specific. It's only you and your application that knows what can safely be put in memory for a short amount of time. So yeah, it's possible to do it but then be aware of where you can do it. In our case, we do it, um, for maybe our second on metadata lookups, things that doesn't change too often and for anything that is related to the entities, so the content in our case, which can update quite often, we have a very, very short, um, 40, 60 millisecond burst cache just to make sure if there's an inefficient code on top looking up the same data several times, we don't have to ask Redis about that.

So to talk about end result example, in our case we had like a page or a dashboard doing um, 17,000 lookups, to Symfony cache, um, and on Redis cluster due to the issues I talked about before, that was up to 40 to 60,000 lookups. So we're talking like around 30 second wait time for, for the user. The editor is sitting there and waiting to load the dashboard. Only 30 seconds was just for Redis and then additional 200 milliseconds, again or something just to do the PHP execution. Um, after all those fixes I talked about, we went down to 63, so today it, it's not even showing up almost on the, on the profiler, on your running in dev mode. So it's, it's really a great improvement. And if we hadn't done it, uh, we wouldn't be able to launch a couple of larger customers stuff we've done in the last half year. So it was definitely necessary. Okay. Um, before I'm going on to edge cases and some things to be aware of, I'll um, briefly do our small demo. I'll need to switch screen.

Symfony Demo

So anyone here tried out Symfony demo or looked at it or used it or, yeah, some nodding. It's a quite simple application. It kind of has blogs and comments on that and some, um, editorial tags and it has a admin backend to let you log in and edit this. And it's all on Symfony, pure Symfony. So, if we, um, if we start the server, make sure that we have cleared cache to make sure that it's correct. And if we, okay. And that's on the right. Yeah.

So clearing cache again.

So, um, first load now it will basically, you can see that doing a, at the bottom there is, is it readable? No. Yeah, Better? Let's do that here also. Okay. So you see at the bottom there, there's three lookups to the database and 174 lookups to the cache. Actually, it's, if we look at it, it should be a lot of save requests, GET items, save, GET items time, save, save the third. Yeah. And actually it's annotations in Symfony itself doing most of that. So you can see we have 14 look-ups and done by the cache backend. And then if I go back now you can see now it's loading again. There is no database lookups, and now there's 23 lookups to cache. So what did I do to, ah, to demo application, if we have a look in the code here, let's zoom in, you might not see the tree on the side here, but basically in demo there's a BlogController for the front and a BlogController for admin. Is this one readable? Nope

Better?

Nope. Uh, so what's been done here? Uh, imported a couple of, um, uh, interfaces, uh, ItemInterface specific thing here and then you'll see it later. But TagAwareCacheInterface is the most important thing because now, and this is both from contracts by the way, um, now we can do a change to, to the controllers. We can on the IndexController. We can just type in that we want TagAwareCacheInterface and then get the whole cache adapter being already wrapped in this tag aware stuff. Um, and we can start to use it. Um, so changes to the controller here is then to generate a key that is unique to every request coming in, page index, which page we are on. If it's a tag filter. So the editorial tag shows you in the admin or the demo. Although this needs to be added to the key to make sure we look up what is unique to that, to the page. We check if it is a hit and it's not.

So we need to do the loading. Um, this is then code, that was there from before. Um, get the latest post in the end, store it, here, and then what is beyond normal PSR6 is this part, setting the tags. So I put it into a separate function here now to make it a bit more readable, but basically adding one global tag for the list itself and I'll show you why later. And then also for each and every post that is going to be displayed on this page, add the tag with an ID of that post. So return an array of tags and this is being stored together with the cache. And we now have a relation, kind of not a strong one, not a foreign key checked one or anything like that, but we have some kind of a relation. So, but if I now try to edit this post, will it work? So we have the first here. Let's just do something simple, edit, you have the tags there for the editorial part, we saved it and then backend, the edit shows up but not in front end. So I need to do something more. I need to also take care about the admin side of things. So doing that,

So to go over what has been changed. If you go to BlogController again and now I need to zoom, yeah,

Readable, um, TagAwareCacheInterface, same as before and now index not so important. We don't need to cache things in admin. Maybe. New; when a new Post is created we type in that we want a cache and in this case when there is a valid change, we need to clear the index because the ID is new. If we clear on the ID and the right, um, the right ones won't be cleared. So we cleared the index to being sure that whoever's page this ends up on it will kind of be be there. And secondly, when there is an edit, same thing type in the TagAwareCacheInterface and in this case we can safely invalidate on the one with the ID. So in this case, 'blog-post' . $Post->getId() and affecting just the caches where this one showed up. Um, there is actually a bug in this and that is if it's changes to those that are old tags and suddenly this page shows up, on this and this blog post shows up on a different, uh, list. It's, it's not going to show up before the cache, expires. But yeah, that's the life of caching.

Uh, so now we have check this out and we should, I think, be able to just edit again.

Mmm.

So now updated successfully and, it shows up. So that's the basics of using cache tagging. Um, the last thing I wanted to show briefly was how you can just switch to the new adapters. So at the end of this, so I'm pulling in the Symfony 4 here to use the latest version and in the end, configuring App to be a new service, setting up a new service and setting it to the new class. And I want them on this because it will just fail. So I'll save you that time. Um, but I'll talk a bit about the downsides and the things you need to think about around this.

Edge Cases & Downside

So, yeah, I already mentioned one cache bug there and there can be others. So some things to be aware of around caching, um, race conditions. Uh, it's a common one. Um, you have, um, for instance, in order to load your entity you might have to do two look ups to the backend first to figure out, okay, what's the latest version or something like that. And then secondly, do you look up on that? Those kinds of situation can be prone to race conditions. For instance, there are many other cases if in between those who calls someone publishes a new version, it will look up the wrong version and your system might act wrongly then by using that.

When caching data. So kind of the case I was showing here, if you, if you are using transactions and using complex things going on in those transactions, you'll have to deal with that. Hopefully we can see if we can find some solution in Symfony cache to make it simpler. But what we opted to do is kind of like disabled caching during transactions and make sure that we're not sending updates to the servers when we are within the transaction, but kind of just a store, whatever should be changed. So when you hit that the last commit, we commit all the changes to cache that should happen. So without doing that end up with a lot of strange stale cache situations.

And then there's async and stale cache. Um, if you used Varnish, yeah, are probably aware of that, but it's typically if you, you forget to clear something, uh, or if you actually do it on purpose, um, then you need to be aware how to deal with it. Um, then the adapter, so we have the RedisTagAwareAdapter, it has a requirement due to the non expiry on the tags to use volatile, um, cache eviction or no eviction, which is the standard in release. So this means on the pro side, it has just one round trip to the server on the negative side, including consumes more memory one. And two, you risk of running out of memory because it will not be able to clear those tags. So that's kind of a problem for some. And then maybe this adapter is not for you or you need to put much more memory to Redis.

So comparing that to RedisAdapter, the one that we had for quite some time now, uh, uses less memory. All the memory can be freed. You can use whatever eviction strategy you want. A negative side, it does two look ups and done MemcachedAdapter compared to this. Again, even less memory. Um, because of the just one data type and probably other reasons it, it's more efficient on how it stores things. All can be freed and at least, in smaller setups it's capable of handling more traffic normally because of the multi tread in nature. But that's a, not everyone agrees with that. On the con side two look ups to, to do when you use it with tagging. So this is all in regards to, in the context of when you use tagging, if you don't use tagging, there will just be one lookup. So now and then you haven't, don't need the RedisTagAwareAdapter in the first place.

Lastly, some details on the RedisTagAwareAdapter just like RedisAdapter it uses MGET to look up everything it needs. So one call, spread it around on all servers when you're on Redis cluster. So in parallel and on the invalidation in this, a few things, um, here showing how the command will look on the command line, it does a rename and then the existing name and then it does the curly brackets there. So that's the kind of the trick it needs to do to avoid, to hit that cross slot issue I talked about earlier because now we telling the client that, Hey, we want to put it on the same slot as the, that key in the first place, but we add additional, uh, suffix, um, to make sure this now our unique one that we can deal with. Specifically, this temp is a unique cache. So there is no, there's not one string. There is a, so we can safely now use this on our process without interference from other processes. We can read the numbers and we can delete. You can delete that, the set itself and the keys within the set, so then the members. So we can safely do this all the way until it's done without interference from other processes and yup. Done. And that is the last, Oh, you're one.

Um, someone wanted to have a picture. Uh, this will be online by the way. That's okay. Yeah. Okay. So that's the end. Any questions around any of this? Yeah, I'm sorry. Are there any limitations on the tags? I mean, maybe there is a how many you mean? Yeah. Or how long they can be. Yeah. The, the limitation of the set is 4 billion members in Symfony 4.3. We had, um, limit in PHP due to, we used s pop, uh, to for 2 billion, but now, now we're using it, uh, as it is. So it's 4 billion. Okay. Any more questions come by me later? I think everyone is ready for lunch. Thank you everyone.

Chapter 27: Prime Time with Messenger: Queues, Workers & more Fun! (Ryan Weaver)

Tip

SymfonyCon 2019 Amsterdam presentation by Ryan Weaver.

Talk slides

In Symfony 4.4, Messenger will lose its experimental label and officially become stable! In this talk, we'll go through Messenger from the ground-up: learning about messages, message handlers, transports and how to consume messages asynchronously via worker commands. Everything you need to make your app faster by delaying work until later.

We'll also talk about the many new features that were added to Messenger since Symfony 4.3, like retries, the failure transport and support for using Doctrine and Redis as transports.

Let's get to work with Messenger!

Alright. Hello. Welcome. It's ah, I guess that was my subtle cue to begin so we can stay on time here. All right, so let's get into this. We have lots of things to talk about. So my name is Ryan. I work on SymfonyCasts. So I do a lot of tutorials on SymfonyCasts. Hopefully you've heard of us at this point. Um, and what else to know about me? Where's, where's Leanna? This is the fun part. She doesn't come to the conferences anymore. There she is. How many people have met Leanna? Let's get a fair picture of, yeah, round of applause. There we go!

So if you don't know Leanna, this is what she looks like. She enjoys conversation, high fives. Basically making friends with everyone. So afterwards, if you would like to meet Leanna, she would like to meet you. She's all the way up here in the second row and or you can tweet at her right now. You can become remote friends by tweeting at her. That would also be awesome. She is responsible amongst actually running the company for, hold on. Here we go for all the animations on SymfonyCasts. So if you enjoy some of the ridiculous things we do, like it's almost always her. And most importantly, I am the father of my son Beckett. I am a dad. So this means that I get to show pictures of my kid to people and since the room is very tight, there's not really a way to get away from it.

So this is my mandatory "there is my son" picture. Okay, so messenger, I think at this point we mostly understand what messenger is. It's a Facebook product for sending chats to your friends. I use it all the time. It's very nice. It's all so a component in Symfony for sending messages and if you haven't used it before, that still makes it sound like it's kind of just Facebook messenger. You're like, wait, Facebook messenger is also how I send messages. So what does that actually mean? So messenger is, and if you haven't used it before, some of these words won't really have much meaning to you. It's technically a message bus, which a message bus can also be used as a command bus or an event bus or query bus. I'm not going to get into the details of the differences between those. Um, it's of course also a school bus sometimes and maybe most importantly or maybe the reason that you're here is that it allows you to run code asynchronously or so you're told.

Stroopwafel Ordering System

So let's get started with messenger. We are going to do something very important. We are going to make our stroopwafel ordering system. So we're going to make an end points or I guess I already made an end point where I can make a post request to send my stroopwafel order. And here's my very complex looking. Here we go. Controller code. Just going to point my computer at myself a bit more. There we go. Okay, so I'm taking some data, I'm creating some variables and okay, this is the really important part. This is our proprietary secret stroopwafel baking machine recipe. Okay? Here's what we do. We um, we uh, we put some, some dough on the waffle iron. Okay. That takes some time. Right? That takes about 500 milliseconds. We're fast, but it still takes some time. And then we add some caramel and if you have a topping, we add the topping and then at the very end we give it to you.

Right. So that's our very important system here. Takes a little bit of time for us to do this. And then boom, we have stroopwafel. How many people have had stroopwafel since they've been here? Okay, good. Yeah, like half. If you're like, what is a stroopwafel just Google it. You should have them here, you can get them fresh here. I've never had a fresh one until a few days ago. I've had them just not fresh. Yeah, well they're warm and the caramel kind of drips out the side. Okay. So this was delicious, but can we organize our code? Of course we can. So maybe, maybe, maybe something that we decide to do is a stroopwafel is a topping, what size you want and who you want to deliver it to. That's like what you tell the person when

you want your, your stroopwafel. So maybe we decide to put those three arguments, those three parameters into just kind of an object.

Organizing Our Code

Okay. So anytime now that we want to make a stroopwafel, we can just create a StroopwafelOrder. And that's kind of the thing that we'll deal with. And this doesn't change our, our code much right now inside of our code, we can take those three variables and put them into an object. Our stroopwafel machine is now using the order variable. It's using this object to call, get size, get toppings. Uh, this is a very, very small change. It's like, okay, maybe the code is a little bit cleaner. Um, okay, if you're looking at the controller now, so we have this nice object, but we still have all of our code in the middle. So what do we do as Symfony developers we say, Oh, you should refactor that logic into a service and you can have nice skinny controllers. Okay, so let's refactor our logic into a service.

So this is our StroopwafelOrderHandler. We need the logger. So I auto wire the logger into it. Cool. And then I decide to make a function called handleOrder that be called anything. Right? And I'm just, I'm making this up. This is my service to centralize my code and then I just move all the logic into that class. So now in our controller, it's ah, we create our StroopwafelOrder object, put this stuff in it, and then we call the handleOrder method on our new service and pass it the object. Cool. So we haven't talked about messenger at all yet. That was just a sort of um, service refactoring 101 just moving code around, putting it into service. Nothing special at all about that. But we're already very close to using messenger. So when we started, we were calling the, the Stroopwafel Handler directly.

We're calling that ah, or actually not when we started, but there's sort of multiple layers we can do here. So most commonly if you have a service that does some work, you normally just call a method on it directly. Like, I need to make a stroopwafel. So let me get the service that does that and I will call a method on it. And we can do that by passing the maybe three arguments to it, like our, our, our size and our toppings and who it's for or we can go a little bit fancier, right? And have our model object that has those properties inside of it. And we could pass that to the service. But in both cases, we're just passing ah we're calling the method directly on the service. So when we introduce messenger or a message bus, it's just a small difference here.

A message bus is just a middleman. So instead of you calling the method on the service directly, you take this, uh, the, your StroopwafelOrder object and you call dispatch on messenger. It's kind of this middle thing. And then it figures out which service to call and calls it for you. So really is like, what is a message bus? It's just a middle layer. Instead of you calling a service directly, you call this middle layer and then it figures out who to call. That's it. So if message bus idea is somewhat new to you, I hope you feel very underwhelmed at this moment. You're like, that's it. That's what a message bus is. Yes, that's what a message bus is. It's a very simple concept. So in diagram format, so let's say we have these kind of three messages on the left, these three things we want to do.

We want to bake a stroopwafel, brew coffee or clean up, clean up the bathroom. And then on the right here we have our kind of our people or our services that do that work. So instead of, you know, for example the, we're going to bake a stroopwafel that's done by the chef. So instead of telling the chef directly to do that, we're going to kind of pass that to the bus and the bus is going to say, Hm, this is work that's done by the chef. Okay? Same thing with brew coffee. That's the barrista and clean up the bathroom. That's maybe somebody else the custodian. Cool. Okay, so now we're ready to use messenger.

Configuring Messenger

So let's install it, composer require messenger. Of course we give you this nice big, blue message. It looks really good. We'll talk about all this stuff though. And the key thing with the message bus, there has to be a connection between the message and the handler.

How does message, how does the message bus know that when we give a StroopwafelOrder that it should call this service and this method on that service. Uh, and by the way I won't show it, but you can have one message. Uh, you can configure it, so one message has multiple handlers. So you can, you can say, here's my one message and maybe you need to have three different handlers call that. That's legal. I'll just do the kind of one to one relationship. So how does messenger know what to call when we pass it a StroopwafelOrder? And that could, when, when messenger was created, that could've been done in any way. It could have been all in Yaml the way that we came up with, which I really like is a two step process. And of course this is Symfony. So if you want to configure this in a harder way, you can definitely do that.

There's always a harder, more manual way of doing things. I'm going to show you the, uh, the most straight forward, the way that most people will do it, but you always have more control over this process. So the main way we do it is we say, okay, you take that service object and you now make it implement MessageHandler Interface. And all that does is that tells messenger to be aware that this is apparently a MessageHandler. It's not enough to tell it what messages it handles, but it kind of makes the system aware of it. This is actually a rare empty interface. There's no methods you implement. It's a purely a marker interface. So that the system is aware of it. Step two is, messenger needs to know what method to call on your handler. So

even if it knows that this message, should go to this handler, it's like, okay, but what method would I call on it?

So the easiest way to solve that was to use PHPs __invoke. So again, you can configure this, but out of the box you're supposed to make your handlers have an __invoke method. That's what messenger expects there to be unless you tell it otherwise. And then the key part of it is, it's just really simple is once we have added the interface here, Symfony looks at this class, looks at the __invoke method and then looks at the one argument and reads it's type hint. It's like just the simplest thing ever. It says, apparently the StroopwafelOrderHandler should be called when a StroopwafelOrder is passed to the message bus because that's what this class is saying. So that's how the connections made between the two. And like we like to do in Symfony, we like to make that connection without having to go to another yaml file.

So everything's happening inside of the message class and the handler class. Okay. Then using, uh, using the message bus really couldn't be simpler and there's a new service MessageBusInterface, you auto wire that into your controller or into a service and then it really only has, um, Oh Hey, I have that backwards. It really only has one method on it, which I have backwards here with the red and green messageBus, arrow dispatch, there's no other method on it. You dispatch the message into it and then it takes care of the rest. So at this point that message is still synchronous though. This was just a, a really like a design pattern. We just used a message bus, uh, to call our handler for us. We added a middleman, but we didn't really make any significant changes to our application. So that's kind of a first half of messenger. The second half is, okay, now how do we make this asynchronous?

Um, so in general, like why would you have a message bus? Like why, why are we adding this middle layer? It was easier just for me to call my service directly. There was really no problem with that. By centralizing things through the message bus, um, you open up a lot of possibilities. In anytime in your code, if you, if you do something in 10 different places and you suddenly centralize that code, that gives you very powerful opportunities to do something every time you perform that action. So a simple example is you can now log a message every time or log something each time a message is dispatched. That's a silly example, but the point is you might be dispatching many messages to do lots of work from all around your application and you can now write some centralized code that is executed every single time a message goes through the message bus. So there's many more interesting things you can do, but you can run code at that time. Um, Oh yeah, here's a great example and we actually have a middleware that does this. You could, you could wrap each handler automatically in a transaction. That's not something that happens out of the box in messenger, but you can add that. And there's people like Tobias Nyholm uses that all the time. So in his handlers, he just kind of writes doctrine code and there's something else that wraps that in a transaction and then commits the transaction afterwards.

Um, and really the main thing, the most important thing for messenger is it means that you can actually have something in the center, intercept the message, not call the handler and then just store it somewhere. And if we've stored the message somewhere, then you can start to think like, hey, if we have this message sitting over here, I don't know, maybe we save it on the file system, that's not really that important, then we could maybe save it for later and then later read that message and then pass it to the handler. So that's kind of the key thing with the message bus. It gives you that hook for saving these messages somewhere else for later. So the way that you do that is called Transports. Which by chance is also a word that's used in the new mailer component. So the transport is where you want a message to be sent for storage elsewhere.

There are three supported out of the box. We have AMQP which is RabbitMQ. Uh, we have the doctrine transport which we are going to use cause it's probably the most familiar and easiest to people. Uh, which literally means that these messages are stored in a database table. And then we also have a Redis transports that you can put things in as well. And you can also do lots of other ones by using third party libraries. But in core we have those three things. So we'll make sure we have doctrine installed and maybe you already have it installed in your project. And then I'm going to uncomment the MESSENGER_TRANSPORT_DSN and use the doctrine transport. That doctrine://default, the default is referring to your default connection in doctrine. So if you have more than one connection, you would choose whichever connection you want.

So it's nice because there's no, um, you don't have to worry about configuring the database twice. You just say reuse my connection. Okay, so then inside of our configuration file, so config/packages/messenger.yaml we don't really have anything in here at this point. We're going to add something now, which is we're going to configure a transport called async. And that name is not important at all. So you can choose your favorite name for that. You're going to see how we reference that key in a second. We're just calling it async because it seems like a pretty good name. So we're saying this async transport should send to doctrine. Okay, so if we ran our code right now, it's still synchronous, meaning the request is still waiting for us to, you know, do the whole stroopwafel thing before the response is returned. So it hasn't changed that we've just added a transport.

Async Configuration

So the way that you actually say, I want this message to be asynchronous is you route it to a transport. So that's the last piece here. We have a routing key and we're going to say App\Message\StroopwafelOrder should go to async. The moment we do that, when it goes into the message bus, there's something that sends it to the doctrine transport and then just stops. So the handler is never called. And we get that response back very quickly. Okay. So where did what, what does it mean to be sent to the doctor and transport? Um, quite literally you have a table in your database. I mean for me only to be fully honest, only

about six months ago, the whole kind of world of queuing was still new to me. So being able to use a doctrine transport to like learn and see things in a familiar atmosphere was very, very useful.

So that's where I'm using it here. You can literally, once you've sent that first message there, you have a new table in your database called messenger_messages and it looks like that craziness. The key thing here is you're actually seeing a PHP serialized object. So that message you put into there has just been serialized and stored in a database table. And there's a couple other properties like when was it created and things like that. But it's basically just sitting there in your database. Okay. Oh and as I mentioned, mentioned up here, the table is created automatically. So like you're like where did this table come from? It's just created automatically for you. You don't need an entity for it happens behind the scenes. Okay, so this is great. So now they're being stored in the database tables though. Next question is how do we actually say I want you read them and handle them.

This is the thing that's called a worker. Worker being a fancy word for a command line process that reads that message out of the database table and then sends it back through the message bus. But this time saying actually handle it. So in Symfony it's just a bin console command bin console messenger:consume -vv a, I do the -vv for very verbose. It just gives you like more, you can actually see what's happening if you do the -vv um, so we do that. We're going to get a couple of log entries. It's going to say I received the message, then it's going to say this message was handled by that class. Meaning it was handled successfully and Oh sorry. Those are actually split. So it says, first it says the message is being handled basically. And then at the end it says it was handled successfully. And, and acknowledging to transport it. That's a queuing word. Acknowledging to transport is a fancy way of saying I removed it from the database table, like we successfully handled it so it's no longer in the database table and now it's gone and it's handled. And then this, this process just, just hangs there and waits for more messages. Okay. And then we have stroopwafel so we still get a stroopwafel we just didn't have to wait for it. Yes. stroopwafel yeah. Yes.

So messenger, uh, done, uh, the end, I'm just kidding, don't, don't, yeah, don't clap or anything yet. Not actually the end because that's only been 20 minutes, but we could actually stop right there. We have a, a messenger tutorial on SymfonyCasts and I was like reviewing it for this talk. And I realized that in about the first five videos we talked about like all the main things, like you could, it's 45 chapters, 45 videos, the first five is enough to give you your 90% use case. Uh, and then you can dive way deeper after that. Um, but it really is just that simple. So the one thing you, even if you don't want to go too much deeper, you probably need to know a little bit more about that worker. Um, if you've never done this type of system before, and one of the obvious questions is how do I run this on production?

Running workers on Production

So as I mentioned, you run this command and it just sits there and waits and it finds another message. It processes it, waits, finds another message, processes it. Um, so do we just run this manually on production and hope that it runs forever and doesn't error out at any point? Definitely not. So actually you want your worker process to die every now and then. So, uh, you know, one of the questions with this stuff is like, hey, PHP isn't really meant to be a long running process. Aren't there memory leaks or this or that? Maybe, uh, the point is like, you should actually celebrate this. Uh, don't expect your, your worker to handle 10 million messages in a row without having any problems or any memory leaks or any errors or like that. Um, you want your worker to die and then when it dies you'll just restart it again.

But you want it to die on your terms but what you don't want is you don't want it to die because something in your code was not doing is hanging onto variables and your memory is getting bigger and then suddenly in the middle of making a stroopwafel you hit the memory limit. That's not a great failure cause you might be halfway done with the stroopwafel and it might be burning on the waffle iron when your worker dies. So you want it to die but you want it to die on your terms. So, what that means is there are options to the worker which basically say only handle 10 messages, then die. Just stop, just exit or run messages until you take up 128 megabytes of memory or run messages for a maximum of 10 minutes. And actually the more aggressive you are in this, like kind of the safer you are because it's the less likely that you're going to end up in the middle of handling a message.

And then something just, uh, something just dies. By the way, because I won't mention it later. If a message did get halfway through and then you just died, just like a really fatal error, that message will be re-handled. That's kind of a property of a queuing system is like the queuing system will redeliver the message later unless you've actually told it, I did finish this. You still don't want it to die halfway through because it still might mean you did half of the work. And so if you handle again later, you'll do half of the work again and then the rest of the work, um, but it will get redelivered. So that means that we need something on production to make sure that this process is always running. And when it exits, it always restarts. This is kind of the one infrastructure thing you need to worry about if you're on a kind of a really, uh, not really.

But if you're on a traditional server setup, this is most traditionally done with supervisor. We have documentation on that on Symfony. It's just a little configuration file. And you say, make sure that command never stops running, which really means when it dies, that supervisor will start it again. And you can also tell it to run three of these commands at once, cause that's totally legal. Three workers or 10 workers or 100 workers so that you're processing the messages faster. Um, or if you're on

like a platform as a service, like SymfonyCloud, Platform.SH, Heroku, these are usually built in. There's usually a section that says like workers and you give it the command and then and then it runs it. So another issue with this is, think about it. If you, when you make a new deploy, you probably already have a worker process running and maybe it will run for five more minutes before it like hits some limit and then, and then exits and then restarts again.

So when you deploy your code, does the worker process see the new code? And it actually doesn't, cause if you think about it, when it was booted up, that was before your deploy. So it got all of those classes in memory. And then when you deploy changes to those classes, they're not there until that worker actually exits and restarts. So again, you actually want your workers to exit and restart in this. So one of the features new in 4.3 and I'll talk about lots of features that are new in 4.3 and 4.4 is very simply just a way to stop all your workers. So on deploy, this is a deploy step. You say messenger:stop-workers and they'll gracefully exit, they'll finish the message that they are currently working on and then they'll close themselves and then you know your Supervisord or whatever is going to then spin up a new one and it solves this problem. So this is like one of the little things that, it seems little but it makes messenger a lot more usable now than one year ago because this is something you had to kind of worry about handling yourself.

Running Locally

Oh also what about running locally? That's another thing. It's like, Hey, when you're running, when you're doing things locally and you have your worker in the background, if you are actively working on your handler, you need to remember to stop the worker and restart it or else it doesn't see the code. I do this all the time. So if you use the Symfony binary, which, which I highly recommend you, that has a symfony run -d command, which just runs things, uh, just executes things, the -d means run as a daemon and then you can tell it to run symfony console messenger:consume. And then my favorite thing, I just learned this, it has a brand new, I think this is maybe only uh, maybe a week old. It has a watch on it. So this basically says, uh, I want you to run that in the background and watch for changes on these directories and then restart the process if you see changes to those directories. So kind of takes care of restarting itself in the background. This also, um, if I remember correctly, will, uh, stops itself when you do your server stop. So if you're using the web server, a server stop is going to stop all the background processes.

Envelopes & Stamps

All right. Um, so I do have to talk a little bit about envelopes and stamps as a core concept. It's not a new concept if you've used messenger or saw Samuel's presentation last year. So I'll just talk a little bit about it. Um, you're sending messages into message bus. So we thought that was a really cool idea. So Nicolas Grekas thought it would be really cute if we, if we put the message, if we were able to put the messages into envelopes and then put stamps on them. I mean, it's just adorable. So we have this concept in, in a Symfony and messenger of messages inside envelopes with stamps on them. And probably most of the time you don't know or care that this is happening. Um, until you do. So, first of all, one of the things that you can do is instead of dispatching a message directly, you can create this Envelope object.

That's a, an object in the core of Symfony. It's not something I created. You create this Envelope object, you put the message inside of it and then you actually dispatch the envelope. This makes no difference. In fact, if you look inside the MessageBus class, if you pass it a Message object, something that's not an envelope, it creates an envelope for you inputs it inside. So our message was always in an envelope. We just didn't, we don't have to worry about ourselves if we don't care, but you are allowed to put it inside of an envelope. Why would you put it inside of an envelope? Because you want to add a stamp. So the message itself is only concerned about containing information that's needed by the handler. Like what is the topping, what is the size and who is this stroopwafel for. But sometimes there's other information that you kind of need to attach to the message.

Most commonly configuration for the transport. So this is a real stamp called the DelayStamp where you can actually tell the transport, I want to delay this message five seconds before it's processed. Another really common stamp if you're using RabbitMQ is there is an AMQP stamp which lets you control like more specific parameters that are special to RabbitMQ that are special to AMQP. So you can customize like, like you could do a um, if you use AMQP, you can attach a um, a routing key. That's how you're gonna attach a routing. Can you do a specific message is via stamp. There's not a ton of these stamps but they exist and that's how you kind of control various things. Also, this is fun for debugging how the system works. The message bus dispatch method returns an the envelope. It's an, it's kind of your same envelope but as it goes through the process, other people add stamps to it and you can dump this and it's just kind of interesting to see what it adds.

We can see our delay stamp on top. There's actually something called the BusNameStamp. It's an internal thing, but it actually records what bus you were sent to in case you have multiple buses in your app. This case it has a SentStamp. That's actually proof that your message was sent to a transport versus handled immediately. If it were handled, it would have a HandledStamp. So it really is like as this message gets delivered in a sense, like there's other parties like stamping it along the way, it went to this country, then I went to this country and then went here and kind of adding information and tracking things along the way. At the bottom there's a TransportMessageIdStamp, which is actually a stamp that contains the ID in the

database that this got stored on since we're using the doctrine transport.

Priority Transports

All right, so another new feature in Symfony 4.3, was priority transports, which was a really important feature for me. So let's say we have a StroopwafelOrder and then we get a CoffeeOrder and then another StroopwafelOrder then um and then somebody tells us that we need to clean the bathroom, another CoffeeOrder, somebody tells us we need to wash the dishes. Um, these come in whatever order they come in, but it's not necessarily the order that we want to process them. Uh, probably if we go clean the bathroom while someone's waiting for their coffee order, it's not going to be a great user experience. Okay. So the way to handle this, um, it's not the only way to handle this, but the, typically the way this is handled in the queuing world and the way that's handled inside of messenger is not by adding a priority to messages.

This is something you can do. For example, in AMQP they have this concept of like prioritizing messages. But most of the time it's not about prioritizing the messages. It's about two different queues, two different transports, queues. Those are kind of synonyms to different queues and one of them is a higher priority than the other one. So in this case, we're going to create a second transport called async high priority and we are still gonna use the doctrine transport. Uh, but this time when you have a transport, you can pass various options to it. One of the options in the doctrine transport is a queue_name. And I'm just going to set this to high priority. What that literally means in the database is that doctrine has a column on that table called queue_name. If you don't set this, it's default. If you set it, it's set to whatever your value is here.

So we have one database table bore, sort of creating sort of two queues in that table. You know, the ability to select all from the default queue and then select all from the high priority queue. So, but they're all kind of still sitting in the same table. So we have a second, a um, transport now and now we're able to, uh, just send some messages to the high priority one and some messages to the normal one. Now at this point there is nothing different between those two transports. So if you're trying to figure out why one is more important than the other one, other than the name, there's nothing. There are two perfectly normal identical transports or queues. The important thing is when you handle these with your worker. So earlier when we ran messenger:consume, we just said messenger:consume. This time we're going to say messenger:consume and we're going to pass the names of our transports in the order that they should be prioritized.

So async_high_priority first and then async afterwards. So each time that the worker goes and looks for new messages, it first asks the first transport, the async_priority_high if there's any messages there, if there are, it handles them. If there aren't then it goes and asks the next transport if there are any there. And every time it handles one message it goes back to the beginning. So if it does handle one in the low priority one, as soon as it finishes, it doesn't keep working on low priority ones. It goes back to the beginning and says, okay now are there any high priority messages or messages that are in the high priority transport?

Failures & Retries

All right, so another thing that was new in 4.3 that was very, very important is like what happens on failure? Cause it's, it's, it's really important. Like, if something fails during a normal web request, uh, at least the user would see a 500 error and they would know that it didn't work and maybe they would refresh and maybe we'd get an error in our logs or something like that. And so it's even more important when you do something asynchronously that you, you know, that it actually works. You don't want somebody to, um, uh, maybe, um, uh, do a reset password and maybe you then send them the email asynchronously and it fails and they never get the email. It's like, that's kind of lame there, just like sitting there waiting for it. So what should happen when there's an error handling a message? So let's introduce our error.

So we, uh, we have a CoffeeOrderHandler. Um, we're not very good at keeping inventory of our coffee beans. So about four out of five times our handler throws an OutOf BeansException. Okay. So I know that's small. I'm going to describe what this is going through. So we, somebody orders coffee. We already have our worker that's like looking for messages. It's finds this message, it's really excited. It receives the message and then it says error while handling this message. And then it says sending for retry number one, um, 1000 millisecond delay. Cause that's a really important thing. One of the most common ways a handler can fail is because you're probably talking to a third party API or even your database. You're doing something over the network and obviously network connections can fail. And when they do, it's usually temporary. You might want to wait at least a little bit of time before you retry immediately. So this delays one second and then it sends it back. Well I should say it sends it back to the transport and says, wait one second before you give this back to me to handle. So to wait at least one second before we handle it again, then it consumes it again. It has another air, it sends it for retry number two. This time it delays it for two seconds, uh, after it tries to the third time it delays it for four seconds.

And then finally in this case, it actually got it the fourth time. Finally we had beans. It handled it successfully and everyone was happy. So, um, the obvious question is what if it keeps failing? Like does this just get tried forever? Um, so this is all dictated by what's called a retry_strategy. And what you see here are the default settings. Um, this is code I stole from you. I think, um, the settings here, the max_retries are three. Um, the delay is one, 1000 milliseconds, one second, and there's a

multiplier of two. And what that means is you get an exponential delay on there. So the first delay is one second delay is a times two, so two seconds and then two, and then third one is two times two, four, and then four and eight. If you'd kept doing retries, it would be 16 seconds, 32 seconds.

So you can kind of give it this nice curve of being, like, if we're still having problems, maybe we need to wait a little bit longer. And by the way, it's a one-second by default. I know of people I know of use cases where people are delaying by a day, like totally legal. You get a single transport that says the types of things that we receive from this transport really should just be tried tomorrow if they're having problems. And you could put that here. There's also a service because of course this is Symfony. Everything is extendable. So there's a service that says, um, that's uh, that you can control the retry strategy stuff like however you want. All right? Um, but still what happens after three times? So this is another new feature in Symfony 4.3 called the failure_transport. And you do two things first.

Well, I guess first you make a, you make a third transport. Uh, we're calling failed. That's not important. And I'm still using the doctrine transport and I'm giving this another queue name. So if you look at kind of the bottom of this, this is me just creating a third transport, no different than any other transport. But that top key, that failure_transport, that points to it, that activates the failure transport functionality. So before we did this, if it failed three times, it then gets discarded from the queue and it's gone forever. If you activate the failure transport after the three failures, it will actually get sent to that failure transport this, so this says, Hey, I've, I've failed three times, I'm now going to send to the failure transport. So you now have this one transport, this one cue that starts to fill up with messages.

Now you could handle this queue like normal, you could actually consume from this queue, but that's not the way it's meant to be used because probably, you know, these are things that failed three times, probably or more. Um, depending on your configuration. So there might actually be a problem you need to look into. So there's a set of new commands where you can actually look at these and see what went on. So we can say ./bin/console messenger:failed:show list all the messages that have failed when they failed and we can get more information about one of them specifically. It's going to tell us a history of like when they failed, when they were redelivered. You can see the delays, the seconds of delay between those retries, the error class, what messages inside. And also like the full stack trace so you can see the full stack trace of like what happened to cause.

Um, I think this is actually just the most recent error. We don't keep track of all of the stack traces, just the most recent one. And then of course you can retry it. And there's also a command to be like, you know what, we're good. We're going to just remove that. It's not never going to work again. Um, so just, just, just fail it.

Fake Transports

Um, so also new in 4.3, just cause I want to show you guys all the new stuff that has happened since the last 12 months. Uh, fake transports, I call them. So one of them is, uh, one of them. So one of the things that occurred to me is that if you are handling things asynchronously, you might not want to while you're locally developing or you might want to, but you might not want a designer on your team to have things like running asynchronously and have to have a worker set up.

If you have like the whole infrastructure dockerized maybe you can do that easily and it automatically consumes them. So if you want to make all of your messages handled synchronously, you can do that with the sync transport. And it's just that simple. It's S Y N C colon, slash. Slash. It's a fake transport. It's like, yes, I am sending it to this transport. Quick handle it immediately and then it just handles it immediately. The other fake transport is for the test environment. So in the test environment you're like, ah, I don't really want or need things sent to a real queue when I'm doing my functional test. So we have some functional tests here, which makes the, the, the request to the StroopwafelOrder. Uh, I don't have, maybe I don't have, uh, uh, my queuing system set up or I don't, I just, I just don't want to send to the queue.

So the way you can handle that is in this case in .env.test, we will use the in-memory:// transport, which was almost called the null transport cause we're sending it to dev null. We're basically being like, yeah, we'll send it somewhere garbage. Uh, the one subtle difference is that it hangs onto the messages for the rest of that request. So that allows you at the bottom of your test. So this is the bottom of my test. I assert the response is successful. I can actually go and get the container from that last request and fetch that transport from the container. It's always messenger.transport.async. The name of that transport. And that transport has a couple of methods on it. Like, I want you to give me all the messages that were sent on you so you can assert that you actually sent one message and you can actually get the message out of it and make sure it looks correct.

Mailer Support

Um, Mailer supports, uh, I put this in here cause it's really cool but it's a, it's just one slide. Uh, could we send our emails async through messenger? Uh, sure. Just add one, one line to your routing. When Fabien built mailer, he put something in there that said, if messenger is present, I will dispatch a method, I will dispatch a message through messenger in order to send emails. So he, the hook point is already there. All we need to do is just say, when that message is dispatched through messenger, I want it to go to async uh, I want to send to an async transport. And that's it. So this is a really interesting hook

point for third party libraries and things like that to be like, Hey, we're doing some work for you. By the way, in order to do that work, we're following the message bus pattern, which you don't care about except that it allows you to delay this until later if you choose to. So you can imagine somebody putting that in their bundles' Read.me says if you want to do this work later, just, you know, add this line to your routing and you're opting in to doing that work later.

More New Features

Um, so the summary of the things that I don't have time to actually talk about, if you are waiting for the Redis Transport that landed in 4.3 if you are using it, but waiting for delay support is the one thing that was missing in the Redis Transport is it couldn't delay. So your retries were like immediate. Um, that's there in 4.4. A piece ah, in 4.3 we changed the serializer to PHPSerializer. How many people, how many of you used messenger before 4.3? Yeah, several hands. And did you ever have any weird situations where the data was kind of missing as yeah, I was already making eye contact.

Yeah. I'm getting a big nod. Yeah. So the in Symfony 4.2 and before the object was serialized to JSON using the Symfony serializer the Symfony serializer is really great. It turns out that if you don't construct your class just the way it wants it, when you, when the message gets serialized, it probably serializes correctly when it gets deserialized some of your properties were missing. So in 4.3, we were like, look, if your Symfony application is responsible for sending and receiving the same message, let's just use PHP realization. The other one is still there. If you're sending to like third parties, you know, to queue for a third party to consume it. So you can still use JSON but out of the box it uses PHP serializer. Um, it basically just removed these WTF moments when you couldn't figure out why things weren't working.

There's also events in 4.3, so you can hook in and run code while the worker is doing things. Um, this is a really good one. It doesn't sound interesting until you hit the bug that causes, uh, between each, uh, message being handled by the worker. The entity manager is now cleared. Uh, we are getting problems where you would have one message handled here and you'd query for an entity. Five minutes later you would have another message that would query for the same entity and your data would be out of date because doctrine is very smart and says, Hey, you already queried for this. I'm not going to make a second query. I'm just going to return to you the five minutes old data. So that's no longer a problem. They really are independent of each other. from_transport, I won't talk about that in detail, but this is the ability to have a message with two handlers.

But one handler goes to one transport and the other handler goes to another transport like high priority, low priority, um, API Platform integration. That's not really new, but that's just once you get used to messenger, like there's a really good way to make, not really a custom operation, but let's say a custom operation on API Platform where when you, uh, when the user sends data to it, you actually, um, uh, handle that through messenger. So you kind of make a message with all the input you want and then it goes through messenger. And, uh, most interestingly, or at least interestingly depending on who you are, it's stable now, don't. Yes. Yeah.

It's um, you know, I'm, I have to stop very soon, but I was, I was listening to Sam, Samuel Roze, um, presentation last year and right at the very end he said these very famous words, he said it's still experimental, but I don't think it will change very much for Symfony 4.3, check out the Symfony, 4.3 messenger change log. There was some, we surprised ourselves. We've gotten those things behind us. If you look at the Symfony, 4.4 change log, it's much, much smaller. It is stable. We changed lots of things. We've had lots to learn. And the most important thing is that it's, it has the experimental off of it, which means even if there was something that we wanted to change, now what's going to happen in a backwards compatible way. So it's, it's actually safe to use. So to put this all together, um, messenger allows you to use a message bus or kind of use the pattern of a message bus, uh, save work for later and retry, uh, and introspect on errors so you just don't like lose them out of thin air. Um, and it's ready for production. So please use it. I love using it. Um, I've only been using it for seven, eight months now. Um, so still like relatively new to me, but, but I absolutely love it. Alright, so thank you guys very much.

Chapter 28: SymfonyCloud: the infrastructure of the Symfony ecosystem (Tugdual Saunier)

Tip

SymfonyCon 2019 Amsterdam presentation by Tugdual Saunier.

You don't know it yet but you are already using SymfonyCloud every day! Since 2017, the infrastructure of the Symfony community gradually migrated to SymfonyCloud. Let me show you what happens behind the scene and how we leverage every SymfonyCloud features to move faster.

Transcript & caption for the talk will be added soon.

Chapter 29: Together towards an AI, NEAT plus ultra (Grégoire Hébert)

Tip

SymfonyCon 2019 Amsterdam presentation by Grégoire Hébert.

Talk slides

You've heard about AI for some time. But it remains obscure for you. How does it work, what is behind this word? If for you, reading white papers or doctoral papers is not your thing, that the Tensorflow doc is just a big pile of words for you, and if you've seen unclear presentations using the same vocabulary without really giving you an idea of how it works and how to implement an AI at home... This presentation is for you. At its end, you will be able to play with an AI, simple, but that will serve as a gateway to the beautiful world of machine-learning.

Transcript & caption for the talk will be added soon.

Chapter 30: Building really fast applications (Tobias Nyholm)

Tip

SymfonyCon 2019 Amsterdam presentation by Tobias Nyholm.

Talk slides

Let us talk about performance. What can we do to make an application run faster? What should we be considering when starting a new application? There are some quick fixes that I will quickly cover. But to get down to really fast response times requires hard work. I will give my best ideas and tricks for fast apps.

I will show how we can build applications that responds in less that 15ms and then work towards even faster than that.

No, this is not a Varnish talk.

Hey, my name is Tobias and I'm going to talk to you about building really fast applications. This talk is not titled building quick applications. It's not title building, fairly quick applications. It's called building fast applications. The kind of applications you see in the movies, and I'm doing this talk because I think we should care about performance I'm also doing the talk because if you know about performance, it gives you better understanding what your application's actually doing. And it's fun. Performance is fun, eh. Short about me. I work at a company called Happyr. I do plenty of Symfony stuff. I'm also running PHP Meetup in Stockholm. And I do Open source. I started my open source career in um, in a conference. It was a SymfonyCon in Paris. PSR6 just came out. So me and a few other friends were like, yeah, let's, let's build implementations for PSR6 and PSR6 is caching, correct. So we build PHP-cache we built like, 20 different implementations of memcache, Redis, the Couchbase database or whatever. And I do plenty of other things in open source too. The most important, the most fun thing here is the NSA, NSA totally violates class privacy. So it's easier to access private properties, methods using NSA. Um, let's get started. Um, frameworks are slow. This is something I see on the internet time to time. Last time I saw it was today in a tweet. And you all heard this, right? Yes, I heard this. Yes. Thank you. Who here uses the framework? Show of hands.

Fair...

You who rose your hand? What is the framework?

What is a Framework

I've been traveling, traveling Europe, mostly asking these questions to different people both in public and private. And for every person I ask, I get a different answers. Some here in the front whispered like, Oh, a collection of components. And Nicholas Grekas said on stage yesterday that if using the dependency injection container, that's when using Symfony, I dunno, but before we, before we're gonna, before we get to claim that something slow, we need to know what it is. So back in the days, like 10 years ago, I tried to use the Zend, I download the Zend, started to read installation instructions. And there was plenty of files. It was folders after folders, and file after file. I don't even know where to write hello world to make something happen. So I quickly gave up like don't make get wrong. Symfony wasn't too much better.

Like if you did Symfony 10 years ago you did your SVN checkout and you got plenty of files and nowadays that's something different. Like Symfony 5 barely has any files when you start with it, you get like four folders and five files. So in in, in Symfony 5 or in Symfony 4 it's you who are responsible for what code you adding. You are responsible what dependencies you are downloading and if you download slow code, your application will be slower. And so I want to say framework is just our code or the code we responsible of. So let's dig into, is our code really slow?

Let's talk about Performance

Let's start talking about performance. There are three important rules in performance. These rules are always true no matter what you're doing in computer science. Rule number one is buy a better server. Hardware is way, way cheaper than software.

And so this is what you gotta do always like unless you're doing something crazy, this is what your going to do. If work, by crazy, I mean if you have more than 10 servers in production, you should buy another one. Rule number two, use Varnish. Most people do use Varnish. It's super awesome, super great and this is not that kind of talk like using Varnish is basically all

the talk is, Oh let Varnish handle 95% of traffic and everything's super fast, super smooth. There's plenty of resources on this online. There's plenty of talks, plenty of books, plenty of blog articles. You should read them and learn about them. It's, it's Varnish is great. However, it's not what this talk is going to be about. If you see a talk that's named how we handle 1 million requests a day or how we take care of 10,000 requests a second, it's definitely a Varnish talk.

Rule number two is like caching, like, but we're not going to talk more about Varnish. I want to talk about this. I want to talk about rule number three, run less code. This is the only way how you actually speed up the execution of your code. You all recognize this, nobody knows this, but you all recognize this. This is machine code. It's basically when you write in PHP, when write echo hello world, it gets compiled to opcache and opcodes and then it got compiled to machine code. Each of these lines is an instruction for your CPU. Each of these line is like one tick in CPU, so if you have a CPU that is one gigahertz CPU, it means that it can run 1 million of these lines every second and every ever these, every code do, every line of PHP we write it generates multiple of ticks and the more ticks you're having, the slower your application will be. So basically more code, more ticks. It's going to be slower. I researched this a little bit because I'm trying to be a good, a good scientist. So I looked at previous Symfony versions, I looked at how many lines of code that are in them, how many lines of code were executed when running Hello World. How many function calls there was, how much memory and how much time it took to execute.

So the point I'm trying to make here is the less code you have, the less code you're running, the less memory it'll take and the quicker it will go. And this is not super, it's not super accurate. Like you see that Symfony 3 was adding some more code than Symfony 2. And also it's not really fair to Symfony 1, because I run this in development mode, but you get the idea right. Whenever asking questions like I wanted to, yeah, thank you. Sure. Okay. But this is only for the hello world application and our application of doing hello world. Like if we were doing hello world, we should do something like this. This is by far quicker than everything every framework ever. So what we need to figure out is what is, what is our application doing? We're not doing hello world, we're not doing something super generic.

We have a business problem and we need a specific solution to this business problem. So first thing we need to do when creating application, we've got to figure out what is the application doing. And this is very hard to make a generic answer to. The best generic answer is basically Symfony skeleton. Like Symfony skeleton is a great base to build out towards. But if you, if you do something specific like, like my URL shortener, I don't, I don't need something super generic, I just need something that's specific for me. So this is an application that's actually in production and it didn't take more than an hour to to write, test and deploy. And I think now you see what it's doing, just takes the URL and rewrites it and sends to somewhere else. And I think we all can agree that if I was using a framework for this, that would be overkill.

Do you need this?

Right? And you could also argue that since I'm using, if I'm using PHP, that would be overkill. I should use something like Nginx rewrite rules or something, but I don't know Nginx very well, that's why I'm writing this in PHP. So whenever you start a new application, you need to figure out where on this scale your application going to be, is it going to be somewhere over here with my redirect application or is it going to be somewhere over here and a big fat application using forms and stuff. Like you need to think about this when starting a new project. So what I like to do whenever I start a new project and when I download new dependency, I like play a game and it's called do you need this? So basically do you need security, if you're on a private network? Do you need forms, if you're running a micro service? Do you need a router like Symfony 4 is the fastest router ever written in PHP, but if you only have five routes, do you really need it? An IF statement is good enough, do you need a kernel if you have a small code base? Do you need dependency injection dependency injection is great, but do we really need it if you only have 10 classes and also do we need classes?

Do you need HTTP? Like if you're running locally, do you need HTTP and if you do need HTTP, do we need HTTP Foundation or are you happy with superglobals? Like you should know that getting a user IP from superglobals is pretty much impossible and HTTP Foundation helps you with that. However, your application may not need the user IP using the Messenger component is great. I love the Messenger component and it's a great way to read from queues and write to queues, but do really need it. Do you need that fancy code that abstraction, maybe they AMQP extension is good enough. In writing the database. Do you need doctrine? Do you need doctrine ORM or happy with doctrine PDO. What I'm trying to force myself and now telling you to force yourself like you should. It's a balance.

I do need, I do use a library with good well-tested code and nice API and they'll take care of some time. It's an all day everything. Or do you write the code yourself having something that's super specific for your use case, but that means that you have to take care of edge cases yourself and you may have to have rough APIs to work with. So it's basically do I want to use a library and maybe develop really quickly or do I want to go with no library with a specific use case, maybe developing more slowly like the one I'm a quick application is slow development or like it's a balance. And I want you to think about this before it required a new dependency.

Building a small application

So when when building something small like small and less complex than Symfony skeleton I like to start from nothing. I like to start in writing my index.php file myself and the result of this is nothing really I can publish. There's nothing really I can like, Oh this is my new framework, which is super nice because I'm trying to solve a specific business case. And that doesn't really make sense to make it public. I, however, I did write a small library, I call it superslim, which basically is a small framework for educational purposes. So what I'm going to tell you now, the next five or 10 minutes is it's super slim. If you pay attention, don't, don't, you don't need to Google it. Superslim one word.

So what I do, I like to use the middleware pattern. It's the same thing as, it's the same thing as PSR 15 so I start with my index.php file. I create a request from the globals and I create the response and somewhere down there I also echo out the body. The result of this file is as expected nothing, but I want to add more. I want to add more content in my response. The first thing I do when adding an anonymous function with it takes a request and response and then a callable to the next anonymous function. And this specific one is just checks that it's a get request. And if I want to be more cooler and add another anonymous function, and that adds something to the response. So when I have my array of middlewares or an array of anonymous function, I give them all to a runner and then I say, Hey, runner, make sure run yourself and give me back the response. And if I run this now I'll get something at least something a little bit more cool.

So you may think, Oh, this runner's is where every, every, every magic thing happens. But it's really isn't. This is, this is the full runner. It just takes the middleware, run them one by one, like run middleware A, B, and C, and then that's it. So you may think like, Oh, this anonymous functions, they are, they're toxic or they looked bad, or they give me theu give me a really wrong feeling. So if, so we just slap on an interface.

So I wrote, I've written many applications using this pattern. And this is a simple application. I think it's a, I'm a geocoder basically. So the idea here is that all the code in this application, I've written myself, I've know this written this Runner, I written this PHP file, I've written all this middlewares sure. In the cache middleware I decided to use Symfony cache. So, but I made the decision to include that cache library. It's all the responsibilities on me. Um, and it might be tricky to understand, okay, sure I like this index.php file, but where does the magic happen? So in my geocoder the last middleware is a router and the router looks like this. So it's a switch case statement. And if I see a request coming to from IP I just call the ipController and the ipAction and that controller looks like any controller that's, a normal Symfony controller.

So consider this application again, the caching I added a cache layer, I was super happy with it. But when developing it came became an issue because I added cache and then I did a change in my controller and I couldn't really see much change. So it would make sense to have different configuration in development and production. So I decided to add a custom kernel and a custom kernel also allow me to have a yaml configuration and a dependency injection. So I created a class I wrote to myself, I call it Kernel and a constructor with a environment parameter and a Boolean if I'm the debugging or not and in the handle function that's the same code as I had in my index.php file. I add my middlewares and I give it to a runner. Then I run the runner, there's a boot function also.

The boot function basically looks at if they have a CachedContainer, if I have a CachedContainer, use it. If I don't have a CacheContainer, create a new ContainerBuilder, add some parameters to it and then look at the services.yml file and the services_environment.yml file and look at, ah, for configuration and then a compile it and a dump it to the file system for, for optimizations for in future. And then I use the compiled container. So with this small class, I can write configuration like this and this supports auto wiring, this supports out in a figure or not resilient. It's auto wiring without configure, you need to do a trick, you need to add code for this entry boot function. So this is basically saying like if you see a service which implements the EventSubscriberInterface, add this kernel.event_subscriber tag to it.

Or if you see something that looks like a command, add this command to it. And then the Drager system compiler passes to make sure every service is up and running properly. So if you want to have more auto configure functions, you need to add more and more boilerplate like this. And in my case, my, the code stayed like this. I didn't have to add anything else. But if you want to continue working on this project, you may add so much boilerplates. At some point you're like, Oh, it's too much. I can't take the, I don't know what all these lines is doing. And if so, feel free to use the FrameworkBundle because this is actually what a framework bundle does for you. It brings some nice configurations for your components and it registers services to make sure auto configure is working properly.

Improve Performance on Legacy Applications

I also recognize that you don't, all of us don't have the luxury of creating new applications. Most of us are sitting on old legacy applications and we just want to make it faster. So, um, let's try to improve the performance on legacy applications. Uh, you all know the basics. This is the basics. They, and let's make them more generic. And there's a reason that these are in order, like the goes from cheap to very expensive. And there is, again, there's no point of making number three unless, unless it's, you think it's fun or unless you're doing something crazy, like you have more than 20, 30 production servers. So I'm gonna sip some water for effect and then I'm going to start.

So I'm going to try something like this. I'm going to give you some tips now and I got to have a tip counter up there so you can

keep talking. The tips. I will share the slides later. So don't, I mean, so you can go over them and I won't give tips on architecture. Like if you have the wrong architecture, that's a very specific problem. So I can really get tips on this, but I going to try, I'll try to give some tips. So are we ready now? I'm a little bit nervous because this, I'm going to try to have a high speed with these tips, so if you knows something I'm going to quickly move on and I'm not sure how well I'm going to do. So I'm a little bit nervous. Anyhow, the first tip is basically you should look at the bottom of this list to buy a better server, the more powerful services at the bottom.

Symfony Cache

Um, and, but we're going to began to talk about the caching a little bit, caching in Symfony has been way, way improved in the last versions, like Symfony implements, three caching interfaces. They implement the great PSR 6, they implement the okay PSR 16 the simple cache and the implement, the excellent Symfony cache and there's adapters to memory, adapters read, Redis memcache, APCU and file and plenty more. And what you should do is you should have one or two of these chains in the configuring application. So cache can either be small and very fast or slow and very big like memory, small and fast and file is slow and big. Also in this scenario, memcache is also slow and big. So create a chain of caches like this. So you do the fast first. If you are missing, then you go to the slower one and make sure when you configure, Redis, memcache don't use TLS because that could increase the handshake could take up to two millisecond, 200 milliseconds.

Doctrine Cache

You should also make sure to configure the cache with doctrine because this is a lifesaver. Like this is probably if you use Flex it's probably already configured. Please be sure you have checked this because when someone told me this, it's saved like 25% of all requests on my application. So doctrine has three caches that the memcache metadata cache which is basically configuration and have query cache which translates from DQL to SQL and it has a result cache which is basically the result of the query.

Result Cache

Ah here is an example of how using result cache, so say you building a CMS like nobody really cares if your page is one hour old from data is not so you can, you can easily, you can easily store it this for an hour. This will obviously be a way better fit for complex queries.

I was going to say that this will, running this code will actually create DQL and invoke doctrine. If you wrapped his function around a Symfony cache, you will not use doctrine and not using doctrine is less code than using doctrine so please, please try and experiment with this.

Cache HTTP Calls

This is not about super important. Often get, often gets forgotten. You should cache HTTP calls because if we fight and optimize for milliseconds and then you do an HTTP call, that takes a second like what's the point of fighting for the milliseconds and showing you this. This is a simple Twitter client that is fetching all the latest tweets and say that you have two visitors every seconds and it takes a half an hour for this. The average time, this is math saying if we measured for 10 minutes, the average time will be an half second. Every request to make an HTTP called the Twitter and it takes a half a second.

So this is long and boring. I know, but you get the idea, right, because this is building up to something. So if you're using cache, you could, I mean I get everything using Symfony Cache I say cache, please give me the latest tweets. That's my cache key. If it's a miss, then call this, call this callable. And on the callable here, I say, just cache this for one minute. It's not a lot. It's just one minute. And with this only one minute caching, the average time waiting for Twitter is now 4 milliseconds and I've been asking around plenty and they confirm. So please cache your, please cache HTTP calls, even even just a little bit and you don't have to be fancy. It's a cache service. I mean you can use local cache and properties. Say you have a simple class calculating the area of a circle and if I call this, every time I call this, I give this a radius, every time I call it I'll start calculating pie and this is obviously time-consuming and we don't have to do this every call.

We can use a local cache to store result and also yes, I'm blown away. This is how we calculate PI. This is actually an Excel, I think that's funny. Building, ah building a web application, it's all about parsing a request and delivering response. Everything that's not part of delivering response you should postpone to later. Ryan, you showed us in the previous talk like you use async for pretty much everything you can use async for. Something that's super slow is code generation like Symfony dependency injection container helps us with this. No yaml or XML is read in production. Symfony dependency injection does this for us and read everything and generates the PHP code with our, with our configuration. So again, if you're not using the dependency injection container, please use it because it's improves performance massively. So when you

define all your services, you still need to instantiate them and instead, oops instantiating services are still slow.

Lazy Services

So that's why you should use Lazy services. So if you have a fat service with a lot of the dependencies to instantiate this, we need to first instantiate all the dependencies. And then instantiate this class, and this can take a lot of time, and especially if all the services are not used in every, every time your using the service though, if all the dependencies use every time using service. So say and lazy services, you can say this service should be lazy, which means don't instantiate it until we using it. So this is when you writing a RequestListener, for instance, most of your request listeners looks like this. So we check if this is the master request, then you continue. And if this starts with API and then continue, but most of the time it doesn't start with API. So you just cancel, you do nothing. And we don't want to spend a lot of time instantiating this expensive event listener if not going to do any job and the work. So that's why lazy services is super, super helpful to speed up performance. However, when defining a service that's lazy, it is global, maybe you won't only want lazy services on the request listeners, and this is where service subscribers comes into play.

Service Subscribers

I don't know if you know about service subscribers, but that they're, Oh, I barely knows service subscribers if they're fairly fairly new, like in Symfony 3.4, Anyhow, how do you, do you know how to use service subscribers? Nobody raised their hand. Okay, so basically what you do, you're going to say this event request listener, I will be interested in these three services and these three services will not be, will not be instantiated until you say, Hey, locator, give me the service. So this is a great way to have a lot of heavy lifting request listeners, but without affecting performance, they're only heavy in when you actually using them. Makes sense. Lovely. So this is cool and all. Let's move on to doctrine. Look at this five lines of code. It's a form. For yourself, think about potential issues with this code. I'm going to show you a less obvious one. You see, I have a daytime immutable for one year. This will generate a new query to the database every second. If I'm just a little bit smart, I can say that I fixed the time. This will help my data database to cache the query. There is more issues with the data layer. For instance, I forgot to use indexes. This is the simple web profiler. And if I click the explain query, I will see that I have about 300,000 users and this will take a lot of time. So if I just add my index to this query, a table, I can see that it takes, I get what goes from 200 milliseconds down to less than one to get this query. But we're not done. I assume some of you spotted more issues with this code?

Nope.

Um, if there is a hundred user created in last year, this will generate 100 queries and, Oh, sorry. Sorry. This will generate one query but 100 objects because this will, the hydration is the expensive thing in doctrines lifecycle. So I fetch my I fetch all my rows and then I create a user object for all of these rows. And if I'm only using two fields here, like I should only query for what I need. So if I change this code a little bit to make sure to query only the ID and the email, I won't create 100 objects. And this will reduce memory massively.

1 + n problem

Massively, like in your application won't crash when you seen this form. Another doctrine related thing is the 1 + n problem. Basically this query over here, I'm fetching the web shop for the this country and then I'm looping over all the products in the web shop to display the name. This will generate one query for the web shop. And then if I have a hundred products it will generate 100 additional queries because we are using lazy loading. So a better idea would be to actually query for the products immediately. This will only generate one query.

Data Transfer

Another doctrine related thing is a data transfer. Say that the product contains a lot of, a lot of megabytes of data. You have images there, you have a lot of product descriptions and every time you fetch your products from the database you take, you transfer data from the database into memory and maybe not really using that memory, maybe not even using the product description. So there's three possible solutions for this. You can use partial objects, which means that you're only getting few fields. However does issue with that is if I give you a product, you don't know if the partial product or the full product, I can also split this product entity into two tables. Like you have a product and then I have a one to one mapping to metadata and the metadata has all this, all the heavy lifting things.

So when I have a product that's quick and easy, no memory, trans, no memory, not a lot of data is transferred from database to memory or I can use another feature which is new Doctrine 2.4. It's the NEW keyword which basically says I can keep my product just like this, like with all the heavy data, but I can create a model looking like this but only the two properties I, this is my simple product and when I query for this I can say give me a simple product and this is how we instantiated. This will

instantiate just, uh this instantiate a simple product, not the full, fat one.

PHP.ini

Other things you should consider is your PHP.ini file. Like make sure you don't do the default from your vendor. Let's make sure we configure sensible, default yourself. This basically makes sure to enable opcache and it sets a sensible defaults for the real path cache. And bunch of other things. There's plenty, Oh, make sure you know what you're doing and make sure it's someone else tells you this is correct thing for your application.

Front-end

The same thing also goes through with front-end stuff like you may have a super quick backend but the front end is slow and heavy and same rules here like run less code. For instance, here's something on top of my head like make sure you have small HTML pages. Like make sure we have a limited number of cookies, only include the JS and CSS script that you actually are using and you can warm up the DNS cache. You can make sure that your scripts are executed asynchronously. You should always use Gzip and HTTP2 and there's one thing with icons, like if you have icons, make sure you use the font swap. So either. You start loading new page and it's white and then you're waiting for fonts to load and when fonts loaded then you show your content or you show the content at once and then you swap off the fonts when they are getting there. I wrote a blog post how you do do this with Google fonts etc and about four weeks later Google updated so they do it automatically for you. So that means two things. That is, font swap is a good idea, and Google reads my blog.

You should also make sure to optimize and lazy load images, you should use CDNs and you should make sure you have good cache control headers like this will cache it for one year, but it would also tell the browse, this file is immutable. It will never ever change. So even so you still, if you have assets, make sure they are immutable because that will be a performance and webpack encore is excellent for helping you with this like front and assets could be a mouthful, but using webpack encore, it helps you with code splitting in the optimize the images, etc, etc. And you're lucky because there are plenty of tools helping you to audit your frontend like Chrome Lighthouse is one of those tools and it's all right. It's perfect.

I show you these rules. There's actually a fourth one...

Have you ever logged into PayPal? Logging you in securely? Sure you are, sure you are. They have a slow system somehow and they made it a feature. I mean that's super smart but I mean all those other things that might be like you've seen before. Sorry for super being super dark. It's basically placeholders. Facebook does this, Slack does this and this even react plugins that helps you to display something. So it looks like your content loaded a little bit quicker than it actually did.

Profile your Application

So if you apply any of these tips and tricks, you should make sure to profile your application. Like you should make a baseline measurement, do a PR with a change and then profile if it actually was a positive change or not. And I cannot stress this enough, like a few weeks ago I created something was an annotation warmer which makes you annotation, go to Symfony cache. I was pretty happy about it and I told Tiwan and Tiguan like, Hmm, I think this will be slower because good reasons but I couldn't really tell him anything because I forgot the profile, the before and after difference. So make sure to always remember to profile your applications before and after you do changes. And profiling also a good way to figure out where your architecture is strong and is also a good way to figure out where your application's wrong. And there are a few profilers out there. The most popular popular one is Tideways and Blackfire. And here's an example of Blackfire. Could you tell me why this particular application is slow?

Too much code? I'm happy to say that I can give you a small hint. It's basically class, class loading like composer is great doing class loading, especially since PHP 7 but it's not excellent. So I'm going to show you some slides. Like if you were at SymfonyCon three years ago there were a person called Andrew Carter and he showed a lot of great things. But these slides, these slides are the best two slides I've ever seen. So let's play a game. Let's play a game of the HTTP pipeline being a restaurant. So the client is a customer, the server is the restaurant and the request is an order and the response is obviously the food and the application who created the response, the application, the one who creates the response is the chef. And the one who facilitates all this is a waiter or waitress. So if the HTTP pipeline is the restaurant, the customer will enter the restaurant. The waitress would take the order, waitress creates a chef, chefs makes food, waitress gives foods customer, and then the waitress brutally murders the chef. Why? Why would we do this? Like if you were a management consultant and came through this restaurant, what is the one tip you would give them? So, and it takes a lot of time to create a new chef.

So in PHP more technical, we do something like this, a browser connects to Nginx and Nginx talks of FastCGi to PHPfpm, PHPfpm has a pool of processes already warm and running. This is good because creating a process is very expensive. And when the request comes in, we create an application, the application delivers the response and then we kill the application.

We're killing the chef. One request more comes in and we kill it. And you probably have a lot of traffic. So you probably doing this a lot of time. But what if it could not kill a chef? What if we could keep the application loaded when they fancy animation is done I'll click. Excellent. So what if it did something like this instead? So FastCGI talks directly with a PHP process and this will keep the application loaded over multiple requests and we don't kill the chef.

Sure. Since PHP is not built for this kind of Witchery, we have to kill it one time to time to just make sure we don't have memory issues. But this is like this is, this will speed up your application significantly. And if you, if you want to do this, there's a few libraries I like FastCGI Daemon. And if you basically put your application inside the anonymous function, and you can do this with my fancy, fancy self built library or a full Symfony framework too. So this is a way for us not to kill a chef, but you may ask, Hey Tobias, next week, where PHP 7.4 what about preloading? So preloading is an excellent way to doing pretty much exact same thing. So with preloading, you can make sure to have like 90 95% of your chef still there. And that's way better than nothing, right?

PHP 7.4 Preloading

So how to use preloading, it is that Symfony simple. You basically add this line to your php.ini config and Symfony will automatically generate the classes or your chef. The classes that you use the most since this is not a PHP 7.4 is unstable and PHP 7.4 is not stable and there is some issues with this and the latest master. This Symfony feature not is completely stable either. There will be plenty of improvements if you tag your class. If you, you can tag your service as container hot path. It will end up in this file. Also, it will be ways for bundles. Make sure you can add things. Add classes to this file too. So I'm going to end my talk now I'm going to say frameworks are not slow. It is you as a developer who has the responsibility to, you have responsibility. Whatever you add to application, you have responsibility for every code line of every dependency you add to your application. And if you add a lot of things you will have a lot of code and yes that will be slow. But if you follow all the tips and tricks I've showed you, you can easily go under 10 milliseconds. Like in fact, you can go under less than five milliseconds. And this is rarely needed, right? We want to do this because it's fun, but it's super possible. And I've done a talk a few times before and all of a sudden at the end of the talk. People like, Oh, Tobias, have you tested this? Really? Oh, Tobias, can you use this in production? Yes, you can. Thank you for your attention.

Chapter 31: Everything you wanted to know about Sylius, but didn’t find time to ask (Łukasz Chruściel)

Tip

SymfonyCon 2019 Amsterdam presentation by Łukasz Chruściel.

Sylius is an open-source eCommerce platform based on Symfony 4, which reached version 1.6 last September. Is there any reason why should you check Sylius if you are not developing eCommerce websites? What are the reasons to choose Sylius? For who is Sylius designed for? I will answer these questions and give some insight over the newest changes in the platform.

Transcript & caption for the talk will be added soon.

Chapter 32: DevCorp: Choose Your Own Adventure (Pauline Vos)

Tip

SymfonyCon 2019 Amsterdam presentation by Pauline Vos.

Talk slides

You're a software development consultant called into DevCorp with a mission. What started as a hip, informal startup now has investor demands to meet. And they're counting on you to help them become a scale-up. How do you grow the existing team and maintain the codebase?

This interactive talk, intended for any developer of any level, will give you some valuable technical and soft skills to take with you on your real-life professional journey. Using a voting system, the audience decides… and has to live with the consequences. Based on a mix of personal experience, agile methodology, and software design principles, this story has several possible endings.

Will you help lift your team's performance or run DevCorp into the ground?

Transcript & caption for the talk will be added soon.

Chapter 33: One Year of Symfony (Zan Baldwin & Nicolas Grekas)

Tip

SymfonyCon 2019 Amsterdam presentation by Zan Baldwin and Nicolas Grekas

Closing keynote.

Transcript & caption for the talk will be added soon.