YRKESHÖGSKOLAN
NOVIA

Ett Symfony-JavaScript projekt

Projektskapande med ramverket Symfony

Hannes Silfverberg

Examensarbete för ingenjör (YH)-examen

El- och automationsteknik

Vasa 2025

EXAMENSARBETE

Författare: Hannes Silfverberg
Utbildning och ort: El- och Automationsteknik, Vasa
Inriktning: informationsteknik
Handledare: Susanne Österholm

Titel: Ett Symfony-JavaScript projekt

Datum: 27.03.2025 Sidantal: 41

Abstrakt

Mitt examensarbete utförde på basis av ett uppdrag jag fick av företaget Nyholm Solutions Oy Ab. Examensarbetet beskriver hur jag har utvecklade en webbapplikation som använder sig av ramverket Symfony. Resultatet av mitt examensarbete går till en kund. Detta examensarbete består av två delar. Den första delen är teoretisk. Där tas upp om de system, kodspråk och ramverk som använts när jag har utvecklat webbapplikationen. Den empiriska delen innehåller först en del om hur man gör en enkel webbapplikation med Symfony och den andra delen beskriver hur jag utvecklat webbapplikationen.

I arbetet används databassystemet PostgreSQL och som grundkodspråk har skriptspråket PHP använts. I projektet har jag använt mig av flera pakethanterare. Dessa är Composer, npm, yarn och PHPs egna inbyggda pakethanterare.

Resultatet av projektet är en webbapplikation med ett fungerande säkerhetssystem där man kan logga in sig i egenskap av användare. Användare kan ha olika tilldelade roller såsom till exempel administratör och dessa roller ger olika rättigheter åt användaren. I webbapplikationen finns fyra tabeller. För att kunna se tre av tabellerna måste man ha rollen administratör. Sidan innehåller också två widgetar där den ena visar antalet producerade produkter under en dag och den andra visar antalet producerade produkter under en månad. Slutligen finns det också två grafer som visar information om produkter som producerats under ett visst tidsintervall, där den ena visar mera noggrann information än den andra.

Språk: svenska
Nyckelord: Symfony, webbapplikation, JavaScript, PHP

OPINNÄYTETYÖ

Tekijä: Hannes Silfverberg
Koulutus ja paikkakunta: Sähkö- ja automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto: Tietotekniikka
Ohjaaja: Susanne Österholm

Nimike: Symfony-JavaScript-projekti

Päivämäärä: 27.03.2025 Sivumäärä: 41

Tiivistelmä

opinnäytetyöni perustuu toimeksiantoon, jonka sain Nyholm Solution Ab (Oy) nimiseltä yritykseltä. Opinnäytetyö esittelee, kuinka olen kehittänyt verkkosivustoa, joka perustuu Symfony nimiseen ohjelmistokehykseen. opinnäytetyöni tulos lähetetään asiakkaalle, jonka nimeä ei salassapitosyistä voi mainita. opinnäytetyö jakautuu kahteen osaan. Ensimmäinen osa on teoreettinen. Siinä esittelen ne järjestelmät, koodikielet ja ohjelmistokehykset, joita olen käyttänyt kehittäessäni verkko- sivustoa.

Empiirinen osa jakautuu kahteen osaan. Ensimmäinen esittelee, kuinka yksinkertainen verkkosivu luodaan Symfonyn avulla ja toinen osa esittelee, kuinka olen kehittänyt verkkosivua, jonka parissa olen työskennellyt. Työssäni olen käyttänyt PostgreSQL-tietojärjestelmää ja peruskoodikielenä PHP kirjoituskieltä. Projektissa olen käyttänyt useita paketinhallintatyökaluja. Nämä ovat Composer, npm, yarn ja PHP:n omia paketinhallinta työkaluja.

Työn tulos on verkkosivu, jolla on turvajärjestelmä mihin voi kirjautua sisään käyttäjänä. Käyttäjille voi jakaa rooleja, joilla on eri valtuudet, kuten järjestelmänvalvoja-rooli, jolla on valtuudet päästä kaikkiin järjestelmävalvojan sivuille. Verkkosivu sisältää neljä taulukkoa, joista kolme on vain järjestelmänvalvojien käytössä. Sivusto sisältää myös kaksi widgetiä, joista toinen näyttää päivittäin tuotetut tuotteet ja toinen kuukausittain tuotetut tuotteet. Lopulta sivusto sisältää myös kaksi kaaviota, joista toinen näyttää miten paljon tuotteitta on tuotettu valitun aikavälityypin aikana ja toisessa näytetään myös tuotetyypin mukaan miten paljon tuotteita on tuotettu aikavälityypin aikana.

Kieli: ruotsi
Avainsanat: Symfony, verkkosovellus, JavaScript

BACHELOR'S THESIS

Author: Hannes Silfverberg
Degree Programme: Electrical Engineering and Automation, Vaasa
Specialisation: Information Technology
Supervisor(s): Susanne Österholm

Title: A Symfony-JavaScript project

Date: 27.03.2025 Number of pages: 41

Abstract

This is an assignment I have made for the company Nyholm Solutions AB. The product of this assignment is made for a customer which, for security reasons, will not be mentioned in this assignment. This assignment describes the development of a website which uses the PHP-based framework Symfony. This paper will consist of three parts, one theoretical part and two empirical parts. The theoretical part will go through the different systems, code languages and frameworks that have been used in the development of this assignment.

The first empirical part is a simple step-by-step guide on how a Symfony website is made while the other part describes the development process of the website I made. The end of the paper describes the conclusions I have made about the project and an opinion of the functionality of the main framework, Symfony.

The result of the project is a website with a security system where you can login and have assigned roles with different authorities. The website includes four tables whereof three are only accessible by users with the Administrator-role. The website also includes two widgets where one shows daily produced products and the other shows monthly produced products. The site also includes two graphs, where one is more detailed than the other.

Language: Swedish
Key words: Symfony, web application, JavaScript

Innehållsförteckning

Ordförklaringar

AJAX	Står för Asynchronous JavaScript And XML. AJAX är ett kommunikationssätt för att få information i XML format från servern till en sida. Man gör ett anrop via JavaScript till servern för att få information.
Bootstrap	Ett CSS-ramverk som underlättar CSS-kodning. Designat för att man lätt ska kunna bygga upp webbapplikationers utseende med Bootstraps rutsystem
Rekursiv akronym	En rekursiv akronym är en förkortning som använder sig av sin egen förkortning i förkortningen.
Modal	En modal är ett användargränssnittselement som visas upp på webbsidans huvudsida. Den gör också ofta en transparent bakgrund till huvudsidan för att få fokuset till själva modalfönstret.
PLC	Står för Programmable Logic Controller och kan på svenska översättas till ett programmerbart logiskt styrsystem. En PLC är en typ av maskin som används för att styra andra maskiner, funktioner eller motorer. PLC:n används ofta inom automation.
Backend	Backend är den del av koden som körs på serverns sida på en webbapplikation. Den behandlar oftast den data som ska skickas till frontend-sidan.
Frontend	Frontend är den del av koden som körs på klientens (eller användarens) sida. Den innehåller oftast sådana koder som kan skrivas ut på en webbläsare.
Skriptspråk	Ordet skriptspråk kommer från engelskans ”*scripting language*”. Skriptspråk är programmeringsspråk för så kallade skript. Skript är koder som man inte kompilerar utan koder som översätts till maskinkod vid körning.
Pakethanterare	En pakethanterare är ett program som automatiskt installerar, uppdaterar, håller reda på eller konfigurerar program. Pakethanteraren använder sig av så kallade paket. Dessa kommer i form av filer som man antingen kan packa ihop eller hämta skilt.
Mall	I detta arbete används mallar för att beskriva färdiga dokument som kan göras till HTML-sidor med en så kallad mallmotor.
Mallmotor	En mallmotor är ett program som kan kompilera så kallade mallar till HTML-sidor som kan skrivas ut på webbläsare.
TypeScript	Ett kodspråk som förenklar JavaScript-programmering och gör så att man kan skriva ut datatyper i koden.

Metronic	Ett frontend ramverk som kommer med en stor verktygslåda med olika UI-verktyg. Det finns flera olika versioner av Metronic anpassat för olika andra ramverk.

1 Inledning

Detta examensarbete beskriver utvecklingen av en webbapplikation gjord med PHP-ramverket Symfony. Webbapplikationen ska ta emot data från en maskin och visa data om produkter dels i tabellform, dels som diagram. Arbetet gjordes på uppdrag av företaget Nyholm Solutions Oy Ab i Jakobstad.

1.1 Uppdragsgivare

Nyholm Solutions är ett företag som i huvudsak skapar program enligt kundernas behov, bland annat telefonapplikationer, embedded & Iot-kommunikationssystem, webbapplikationer, systemintegrationer och PLCn, som är en förkorting av Programmerbart logiskt system. De tjänster som de har skapat är bland annat Trolle. Trolle är ett program som är avsett för att organisera arbete för företag som verkar inom olika servicearbeten, inklusive underhållsbranschen och byggnadsbranschen.

De har också utvecklat program åt Asctel Oy, ett system som styr och övervakar olika enheter på fältet med hjälp av ett visuellt gränssnitt. Det finns två versioner av systemet. Version 1 fungerar med en Windows-baserad server och dess klienter fungerar med embedded PIC mikrokontroller. I version 2 är både servern och klienten Linux-baserade enheter. Denna version har också förmågan att kartlägga olika brandbilar och övervaka deras kommunikation.

2 Ramverk

Det är svårt att hitta en exakt definition på begreppet ramverk på grund av begreppets breda användning, men de flesta definitioner kan oftast sammanfattas i lexikonet Cambridge definition som är ”*a supporting structure around which something can be built*”, som kan översättas med en stödande struktur kring vilkens något kan byggas. I detta arbete används en mera specifik definition på ramverk, det vill säga objektorienterade ramverk (Ragnarsson, 2008).

Objektorienterade ramverk utgör en modell för något slag av domän som byggs upp av olika klasser och objekt. Ofta används de för att lätt kunna implementera olika komponenter till ett program.

Man kan använda en eller flera olika existerande ramverk när man håller på att skapa en applikation. Detta ger en bra grund när man försöker skapa något då man lätt kan använda sig av de olika användbara strukturerna som finns färdigt i ramverket. Dessutom kan man i många ramverk tillsätta olika användbara komponenter (Rehle, 2000).

Det finns många olika typer av ramverk. Oftast är objektorienterade ramverk baserade på programmeringsspråk såsom JavaScript-baserade ramverk eller PHP-baserade ramverk.

2.1 PHP-baserade ramverk

Det finns en hel kategori av ramverk som är baserade på skriptspråket PHP. PHP-ramverk är populära delvis på grund av att PHP är lätt att lära sig. De har ofta öppen källkod och är ramverk för flera plattformar. Dessutom är de ofta lätt kompatibla med andra komponenter såsom SQL. Några exempel på PHP-ramverk är Symfony, Laravel, CakePHP och CodeIgniter.

2.2 Symfony

Symfony är ett PHP-ramverk med öppen källkod som används för att skapa webbapplikationer. Ramverket inkluderar en stor mängd komponenter som har olika funktioner, som man kan använda sig av. Det har också en struktur som man lätt kan förstå.

Komponenterna i Symfony installeras oftast med programmet Composer och det finns ett stort antal komponenter som man kan ladda ner.

Symfony skapades av det franska företaget SensioLabs och de håller fortfarande på att utveckla Symfony. Den första stabila versionen alltså av Symfony var Symfony version 1.0 och den kom ut i början av år 2007. Man gjorde ytterligare fyra olika versioner av Symfony 1 och den sista versionen var Symfony 1.4. Denna stöddes ännu i 3 år efter att den blivit utvecklad. År 2011 kom Symfony 2.0 ut. Version 2 hade en krävande start eftersom den blev utvecklad på nytt helt från grunden. Före Symfony använde sig av Composer för att

installera olika komponenter var den beroende på Git undermoduler för att installera komponenter. Under åren har Symfony utvecklats mera och den nyaste versionen var 7.1.5 då när detta arbete skrevs (Wojciech, 2015).

2.3 Problem med ramverk

Användning av ramverk kan också medföra olika motsättningar och problem i fråga om design.

För det första så kan det vara en hel del att lära sig när man använder sig av ramverk, vilket gör att det går åt en hel del tid bara till att förstå hur man skall använda sig av själva ramverket. Beroende på projektet kan det också gå snabbare att göra ett projekt från grunden av den här anledningen.

Ett annat problem med en del ramverk är att de inte alltid är kompatibla med andra kodspråk och kan därför bara användas i språket som ramverket är designat att användas med (Johnson, 1997).

2.4 Frontend-ramverk

Frontend-ramverk är ramverk som fungerar som grund för frontend-sidan, alltså den del av programmet som körs på klientens eller användarens sida. Frontend-ramverk ger olika verktyg som underlättar arbetet med att göra ett fungerande användargränssnitt för klienten. Många frontend-ramverk kallas också för JavaScript-ramverk på grund av att de ofta använder JavaScript.

Exempel på frontend-ramverk är:

- Angular som är utvecklad av Google och är en av de större frontend-ramverken och kallas därför även för plattform. Typescript används ofta i Angular-projekt. I ramverket finns det inkluderat verktyg som används för att konvertera från Typescript kod till JavaScript. Annat som ramverket fokuserar på är att kontrollera användargränssnitt, att på ett behändigt sätt skicka AJAX-förfrågan och att hantera flera applikationer tillsammans. Dessutom har Angular ett blankettvalideringssystem. Angular får också regelbundet uppdateringar.

- Vue är ett frontend-ramverk med öppen källkod. Ramverket är storleksmässigt ett litet ramverk. Som minst kan en Vue-app utan tillägg vara 16 kb och med alla Vue-komponenter 27 kb. Det är också möjligt att använda sig av Typescript. En sak som är behändigt när man kodar med Vue är att det är möjligt att visualisera ändringarna i användargränssnittet samtidigt som man gör ändringar i koden. Ramverket skapades 2014 av Evan You (Anonymous, 23.01.2025).

- React används ofta även inom mobilapplikationer. Ramverket är känt för att vara lätt att använda. Det är också möjligt att använda sig av Typescript i React (Vyas, 2022).

3 Uppbyggnaden av Symfony

Följande kapitel presenterar Symfonys bakgrund och om olika andra program som används för att hålla upp ett Symfony-projekt. Kapitlet innehåller även ett underkapitel om PostgreSQL som kan användas som databassystem för ett Symfony-projekt.

3.1 Php-grund

En stor del av Symfony är det skriptspråk som den bygger på, alltså PHP. PHP är ett skriptspråk med en öppen källkod, som är gjord för allmänt bruk, men huvudsakligen designat för att användas inom webbutveckling på serversidan och kan också sättas in i en HTML-kod. PHP är en rekursiv akronym för PHP: Hypertext Preprocessor.

Första versionen av PHP gjordes 1994 av Rasmus Lerdorf. I detta skede var PHP en samling med körbara koder, som är skrivna med programmeringsspråket C. De skulle fungera som Common Gateway Interface-koder. I detta skede kallades samlingen Personal Home Page Tools eller PHP Tools. Första versionen, som var öppen för allmänheten, kom ut år 1995. PHP har sedan gått genom en lång historia med olika ändringar och uppdateringar. PHP bytte till och med namnet temporärt till FI som står för ”*Forms Interpreter*”, men fick ”PHP” tillbaka med i namnet samma år.

Versionen, som äntligen började se mera ut som versionen vi har idag, är PHP 3.0. Den versionen fick sin början 1997 när några universitetsstuderanden från Tel Aviv i Israel fann att den dåvarande versionen av PHP, alltså PHP/FI 2.0, inte hade tillräckligt med funktioner

för att göra deras egen e-handelsapplikation. Då började de själva utveckla en ny version av PHP/FI. De tog kontakt med Rasmus Lerdorf, utvecklaren av PHP/FI, och de började utveckla den nya versionen tillsammans. I detta skede gjorde de ett helt nytt programmeringsspråk, som nu fick det officiella namnet PHP: Hypertext Preprocessor. För närvarande använder PHP en ny programmotor som heter Zend engine.

Ända till denna dag utvecklas PHP och den nuvarande versionen är PHP 8. (Mehdi, o.a., 2024).

3.2 Composer

Composer är ett verktyg som hanterar olika bibliotek och komponenter man kan behöva till ett PHP-program. Composer är i huvudsak inte gjort för att installera olika bibliotek globalt, utan den installerar i stället olika beroenden skilt för varje projekt i en vendor-mapp. Därför insisterar man på Composers egen hemsida att Composer borde kallas för en dependency manager i stället för en vanlig pakethanterare. Den har dock också en förmåga att installera beroenden också globalt.

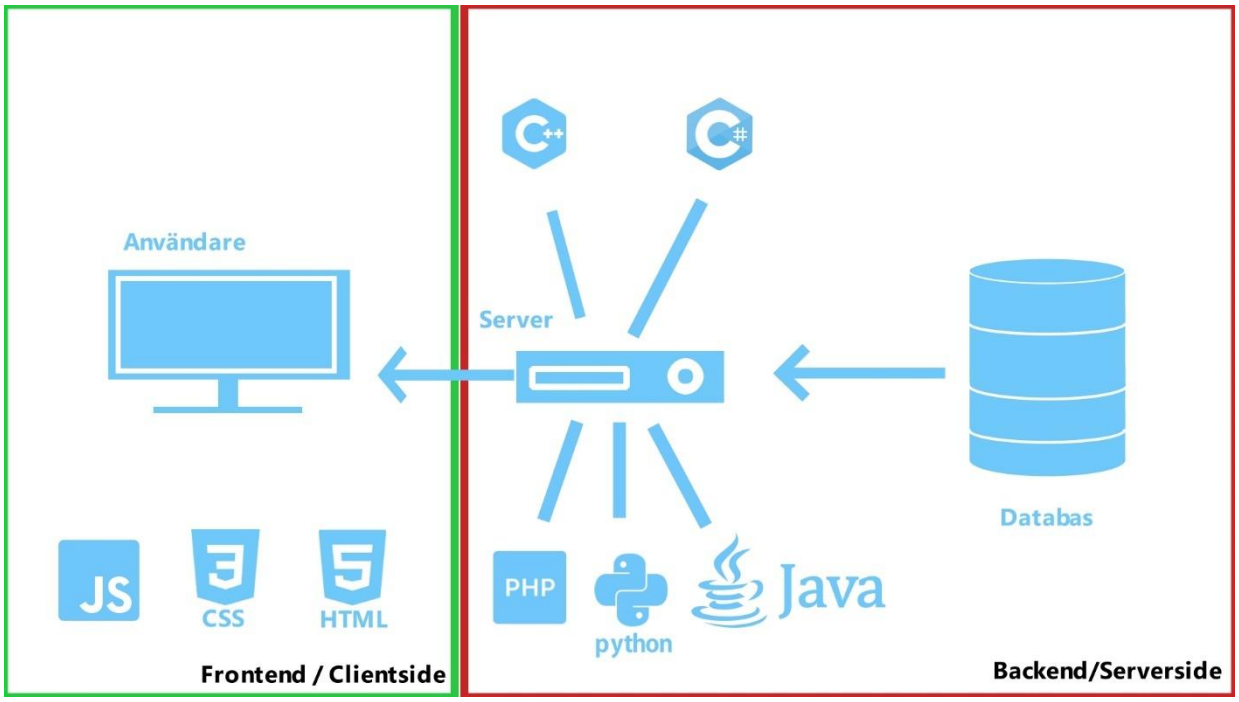
Med Composer kan man också relativt lätt hitta nya uppdateringar till olika program. Alla beroenden man använder sig av är inskrivna i en fil, som heter composer.lock. Denna fil visar vilka komponenter som skall uppdateras eller installeras. Om man till exempel vill flytta ett projekt, men de olika PHP-biblioteken man har installerat med Composer tar för mycket utrymme, kan man ta bort vendor-mappen och sedan flytta projektet dit man vill ha det. Därefter kan man automatiskt via Composer installera om alla komponenter som är inskrivna i composer.lock-filen. (Adermann, Boggiano, & others, u.d.).

3.3 Backend- och Frontend-system

Ett webbprojekt kan man oftast indela i två delar. Den första delen är backend, även känd som server-side. Till denna del hör allt som körs på serverns sida, vilket betyder att denna kod inte alls körs på användarens sida. Backend-sidans uppgifter består oftast av att hantera data, kommunicera med databasen och skicka data till klienten. Några vanliga

kodspråk man kan använda för backend-programmering är C++, C#, Java, Python och PHP. I detta arbete har jag använt mig av PHP.

Till frontend-sidan eller client-side hör allt som körs på klientens eller användarens sida oftast med en webbläsare. När man programmerar på frontend-sidan kodar man oftast layouten på själva sidan, vilket bland annat innefattar vilka webbelement som skall finnas på sidan och hur den ska se ut. Man kan även programmera en del logik med JavaScript på frontend-sidan. De vanligaste kodspråken på frontend-sidan är bland annat: HTML, CSS och JavaScript (Palaksinghal9903, 10.9.2024). I detta arbete har alla de nämnda frontend-kodspråken använts och av backend språken har jag använt PHP.



Figur 1: Bild på frontend – backend struktur.

3.4 Twig

Twig är en mallmotor för PHP som snabbt kan kompilera så kallade mallar, på engelska Templates, och tar PHP till en enklare form.

Det finns många större projekt som använder sig av Twig så som eZPublish, phpBB, Drupal8 och Symfony. Det finns dessutom möjligheter att använda Twig i andra ramverk, bland annat Laravel, Slim, Yii, och Codeigniter (SensioLabs, 2018).

3.5 PostgreSQL

PostgreSQL är ett objektrelationellt databassystem med en öppen källkod, som jag har använt i detta projekt. PostgreSQL använder sig av en utökad version av SQL-språk för att hantera data (PostgresSQL Global Developement Group, 2024).

3.5.1 PostgreSQLs Historia

Historien om hur PostgreSQL utvecklades går tillbaka till år 1973 där det fick sin början i form av ett projekt vid Californias universitet i Berkeley. Projektet gavs namnet Ingres. Inom ramen för projektet presenterade och utvecklade man olika koncept för hur man kan förvara data. Ingres blev också en kommersiell produkt.

År 1986 fortsatte man att vidareutveckla koden för Ingres. Detta nya projekt leddes av ett nytt team med professor Michael Stonebraker som ledare. I detta skede fick projektet namnet POSTGRES. Så småningom lyckades man år 1987 göra en funktionerande demo-version som visades upp nästa år under ACM-SIGMOND konferensen, det vill säga en internationell konferens som specialiserats på datahantering och grundats 1976. Den senaste ACM-SIGMOND konferensen hölls år 2024 (Worsley & D.Drake, 2002).

Version 1 av POSTGRES publicerades till ett fåtal användare år 1989. Efter att den första versionens regelsystem blev utsatt för kritik, gjorde man om regelsystemet i version 2, som kom ut år 1990. År 1991 kom version 3 ut.

POSTGRES-projektet sponsorerades av flera olika organisationer, såsom DARPA, ARO, NSF och ESL. POSTGRES har använts inom många områden, såsom i spårning av asteroider, medicinsk datahantering och olika geografiska informationssystem.

En ny version av POSTGRES utvecklades år 1994 av Andrew Yu och Jolly Chen som hette Postgres95. I denna version lade man SQL-funktionalitet till POSTGRES. Postgres95 använder sig av ett nytt program vid namn psql för att hantera databasen. År 1996 bestämde man sig att döpa om Postgres95 till det nuvarande namnet PostgreSQL (PostgresSQL Global Developement Group, 2024).

3.6 Doctrine

Doctrine är en samling PHP-bibliotek som är fokuserade på kommunikation med databaser och formande av så kallade objekt. Kärnan inom Doctrine är Doctrine ORM som står för Object Relational Mapper och DBAL som står för Database Abstraction Layer.

Inom Doctrine ORM använder man sig av så kallade entiteter, på Engelska entity. De är PHP-objekt som kan identifieras med hjälp av en unik identifierare eller primär nyckel.

4 Projektskapande i Symfony

Här beskrivs den praktiska delen av arbetet. Den praktiska delen kommer att delas in i två delar, det vill säga en del om hur man kan bygga upp en webbapplikation med hjälp av Symfony och en annan del om mitt arbete. I arbetet kommer jag att gå igenom hur man bygger upp ett Symfony- projekt. Projektet har varit att skapa en webbapplikation med ramverket Symfony som lagrar data i en PostgreSQL-databas. Webbapplikationen har ett fungerande säkerhetssystem med användarautentisering där vissa användare har administratörsrättighet.

Projektet, det vill säga applikationen som arbetet baserar sig, innehåller en API via vilken man kan ta emot information från en annan typ av enhet. Frontend ramverket Metronic har använts som grund för frontend-utvecklingen.

4.1 Början av projektet

För att skapa ett Symfony- projekt måste behöver man en lämplig version av PHP. När arbetet började var den rekommenderade versionen av PHP, det vill säga den nyaste, stabila versionen, PHP 8.2, men enligt Symfonys officiella dokumentation så går det också att göra med nyare versioner. Man behöver också Composer för att installera Symfony och andra PHP-baserade komponenter. När man har installerat allt man behöver kan man göra en verifikation, det vill säga programvarutestning, för att få reda på om Symfony fungerar ordentligt. Testningen kan göras på den terminal man använder med hjälp av kommandot som visas nedan.

Kommando 1: Kontrollera om Symfony kan användas i programomgivningen.

```
$ symfony check:requirements
```

Efter verifikationen som visar ifall installationen av Symfony fungerar ordentligt kan man påbörja ett Symfony-projekt. Det första man ska göra för att börja ett Symfony-projekt är att man navigerar med en terminal till ett ställe där man vill ha sitt projekt. Sedan kan man köra kommandot nedan:

Kommando 2: Skapa ett nytt Symfony version 7.1 projekt.

```
$ symfony new my_project_directory --version="7.1.*" --webapp
```

Sedan kan man starta själva Symfony-webapplikationen med att först navigera till den skapade projekt-mappen som har samma namn som man har gett åt projektet med kommandot: symfony server:start. Symfony startar nu en egen server på porten 8000. Sedan kan man navigera med en browser till sidan med adressen localhost:8000. Nu är man på projektets första sida. Det kommer dock inte att visa så mycket då det inte finns någon frontend ännu.

4.2 Projektmiljön

Jag började med att göra Symfony-projektet på operativsystemet Windows, men rekommenderades sedan i stället av mina arbetsgivare att skapa projektet på en Linux-baserad virtuell dator. Jag valde i detta fall Ubuntu på grund av att man lätt kan sätta upp en virtuell Ubuntu på windows datorer, utan att ha några extra virtuella maskinprogram. Jag installerade den nyaste versionen av PHP, som i det skedet var PHP 8.2, till datorn.

4.3 Backend grund

För att få kontakt med databasen så behöver vi Doctrine. ORM/Doctrine kan man installera via Composer med hjälp av kommandot nedan.

Kommando 3: Installera komponenten orm-pack.

```
composer require symfony/orm-pack
```

En annan komponent som är mycket användbar är make-komponenten som används för att automatiskt skapa olika viktiga filer i Symfony-projektet. Här nedan finns installationskommandot:

Kommando 4: Installera komponenten maker-bundle.

```
composer require --dev symfony/maker-bundle
```

När man har installerat Doctrine till sitt Symfony-projekt kan man börja med att försöka ansluta projektet till en databas. Till detta behöver man upp en databasserver. För att kunna ansluta till databasen skall man navigera till den automatiskt skapade .env-filen i projektet. I filen finns det färdigt olika mallar på adresser som Symfony-projektet kan ansluta till. När man har en fungerande databasserver som kör så kan man välja en adress enligt databastyp och sätta in de rätta värdena så som man har konfigurerat databasservern. Oftast är det fråga om databaslösenord, IP-adress och port. Ifall man har en lokal databasserver så kan man använda den automatiskt skapade IP-adressen som är 127.0.0.1.

För att kunna navigera igenom ett Symfony-projekt till olika sidor behöver man en, eller flera, kontroller. Man kan också skapa ett sådant med make-kommandot som syns nedan.

Kommando 5: Skapa en ny kontroller i webbapplikationen.

```
php bin/console make:controller ProductController
```

Kommandot skapar färdigt en PHP-fil som fungerar som kontroller. Den hittas i src/Controller-mappen i projektmappen. Make-kommandot skapar också en färdig html.twig-indexmall där man sedan kan göra frontend-programmering. I kontrollerfilen hittar man de nödvändiga komponenterna som behövs för att den kontroller man har skapat ska fungera. Därefter hittar man huvudklassen, som innehåller alla kontrollerfunktioner i ifrågavarade kontroller. Till klassen skapas också en färdig funktion. Denna funktions uppgift är att rendera eller skapa en frontend-sida till klienten så att klienten kan se sidan som skall visas upp för dem. Första delen av funktionen är rutten, den använder sig av Route-komponenten och reserverar första raden av funktionen. Raden innehåller URL-rutten som kör själva funktionen. På samma rad finns också namnet på

rutten. Inne i funktionen finns det bara en return-funktion som skickar tillbaka till klienten en renderad sida baserad på den indexmall som tidigare blev skapad med make-kommandot. Klientens webbläsare tar emot sidan och bygger upp den till en visuell form som användaren kan se. Här är ett exempel på en kontroller som har blivit skapat via make kommandot:

Kodexempel 1: Strukturen av en kontroller med en funktion som öppnar en sida.

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class ConferenceController extends AbstractController
{
    #[Route('/conference', name: 'app_conference')]
    public function index(): Response
    {
        return $this->render('conference/index.html.twig', [
            'controller_name' => 'ConferenceController',
        ]);
    }
}
```

Doctrine använder sig av så kallade entiteter. Dessa kan man skapa med hjälp av en make-kommandot som syns nedan.

Kommando 6: Skapa en ny entitet.

```
$ php bin/console make:entity
```

Med kommandot skall man fylla in entitetens namn. Därefter går man igenom olika kolumner som finns i entiteten. Där ska man fylla i kolumnens namn, datatyp, parameter beroende på datatyp, ja eller nej ifall fälten i kolumnen kan vara tomma och så vidare. Därefter kommer nästa kolumn som ska fyllas i. För att avsluta kommandoprompten

trycker man in ett tomt värde då det frågas efter kolumnnamn. På detta sätt har en entitetsfil skapats.

För att sätta in entiteten till databasen så måste man först göra en migration för entiteten. Detta görs med make-kommandot som syns nedan:

Kommando 7: Skapa en ny migration.

```
$ php bin/console make:migration
```

Efter att man har gjort en migration med kommandot så kan man migrera den till databasen. Den gör man med kommandot nedan.

Kommando 8: Kör migrationsändringarna till databasen.

```
php bin/console doctrine:migrations:migrate
```

Varje gång man gör en ändring i entiteterna ska man använda sig av migration-processen så att också databasen ändras enligt ändringarna.

Här kan man börja sätta in rader i entiteten:

Ett sätt att automatiskt kunna sätta in testdata eller så kallad dummy-data i en databas i Symfony är att använda sig av så kallade fixtures. Fixtures är PHP klasser som innehåller objekt, som sedan kan skrivas ut i en databas. För att kunna använda fixtures behöver man DoctrineFixturesBundle-komponenten, som man kan installera med Composer-kommandot nedan:

Kommando 9: Installera komponenten doctrine fixutres bundle.

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

När man har installerat komponenten färdigt så skapas det automatiskt en PHP-fil, som heter AppFixtures.php. I den hittar vi klassen AppFixutres och en funktion som heter load. I load kan man programmera in olika objekt som ska sättas in i databasen. För att göra detta behöver man också inkludera de entiteter som objekten baserar sig på. AppFixtures.php-filen använder sig av komponenten ObjectManager som finns färdigt inkluderad i början på PHP-filen för att man ska kunna sätta in värden i databasen. Först gör man en variabel, som

man gör till ett nytt entitets objekt enligt den entitettyp man vill sätta in. Man använder sig av entitetens värdefunktioner för att sätta in värden i entitetsobjektet. Dessa hittar man i själva entitets-filerna som skapades tidigare. Sedan användes persist-funktionen, som finns i ObjektManager-komponenten för att spara ett objekt för användning. Därefter kan man fortsätta med nästa objekt. Man kan också använda sig av en loop, även kallad slinga, för att skapa objekt med slumpmässiga värden, så att man inte behöver lika mycket kod. När man har alla de objekt man behöver så kan man spara dem i databasen med hjälp av flush-funktionen. Kodexempel 2 är ett exempel på en fixture-fil.

Kodexempel 2: En fixture-fil som skapar dummy-data i databasen.

```php
<?php
namespace App\DataFixtures;

use App\Entity\ExempelEntitet;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;


class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $exempelEntitet = new ExempelEntitet();
        $exempelEntitet -> setColumnEtt("föremål1");          //string
        $exempelEntitet -> setKolumnTvå(1);                   //integer
        $manager->persist($exempelEntitet);

        $exempelEntitet = new ExempelEntitet();
        $exempelEntitet -> setColumnEtt("föremål2");
        $exempelEntitet -> setKolumnTvå(2);
        $manager->persist($exempelEntitet);

        $manager->flush();
    }
```

4.4 Skapande av tabeller med Twig

När man har skapat en fungerande backend, som har fungerande databaskommunikation, så kan man på frontend-sidan börja göra visuella tabeller som klienten kan använda. När man skriver ut tabeller så gör man det vanligtvis med JavaScript. Symfony har dock ett eget

sätt att skriva ut data på frontend-sidan. Denna metod är att använda sig av Twig-funktioner. De flesta html-sidor i Symfony kommer alltså i form av html.twig-filer. Twig gör det lätt att snabbt skriva ut saker på frontend-sidan, utan att behöva använda sig av någon JavaScriptkod. I Twig finns det olika funktioner som man kan skriva direkt in i html-koden. Bland annat kan skriva ut olika variabler, göra loopar eller if-satser.

Före man börjar skriva ut tabellen behöver man också få fram den data som behövs i tabellen. Därför måste man exportera den från backend-sidan. Detta gör man genom att man hämtar data från databasen till kontrollern med hjälp av en repository-fil, som skapats på samma gång som man skapade en entitet. När man har gjort en migration så skapas automatiska funktioner i repository-filen, så som findAll() och find()-funktioner. Dessa kan man använda via entitymanager-komponenten för att få antingen all data eller en rad från databasen beroende på entitet. I det här fallet använder man sig av findAll()-funktionen, vilket betyder att man i början av koden behöver en referens till Entitetens repository-fil. När man har dessa kan man använda entitymanager för att använda sig av repository-filens findAll()-funktion för att sätta in all data i en variabel. Sedan kan man exportera den i samma render-funktion som renderar tabellsidan, där man tillägger variabeln i den andra parametern tillsammans med det exporterade namnet på kontrollern. På samma sätt som man ger namn åt kontrollern så ger man ett namn för variabeln och sedan sätter man in själva variabeln. När man har gjort detta kan variabeln användas av på frontend-sidan via Twig. I kodexempel 3 kan man se hur kontrollerfunktionen kan se ut med de nya tilläggen:

Kodexempel 3: En kontrollerfunktion som öppnar en sida tillsammans med data från databasen.

```
class ConferenceController extends AbstractController
{
   #[Route('/conference', name: 'app_conference')]
   public function index(ExempelEntitetRepository $exempel_entitet_repo): Response
   {

      $exent = $exempel_entitet_repo->findAll();

      return $this->render('conference/index.html.twig', [
         'ExporteradObjekt'=> $exent,
         'controller_name' => 'ConferenceController',
      ])  ;
   }
}
```

För att skriva ut en tabell med hjälp av Twig kan man göra en simpel for-loop i en html table där man loopar igenom det tabell-objekt, som man har exporterat från backend-sidan. Kodexemplet 4 är ett exempel på en Twig-loop. I den kan man observera hur Twig-koden sätts in i webbdokumentet. Twig-koden använder sig av olika klamrar för att sätta in funktioner och variabler i mallen. {%...%} -klamrarna används för att köra kod som inte i sig själv skriver ut ett resultat. {{…}} -klamrarna används för koder som skriver ut dess resultat i mallen (SensioLabs, 15.11.2024).

Kodexempel 4: En enkel Twig-loop som skriver ut flera rader med data.

```
<table>
   {% for Föremål in ExporteradObjekt %}
      <tr>
         <td>{{Föremål.ColumnEtt}}</td>
         <td>{{Föremål.KolumnTvå}}</td>
      </tr>
   {% endfor %}
</table>
```

4.5 Säkerhet och kopplingar

De flesta webbapplikationer har något slag av säkerhetssystem eller åtminstone inloggningssystem där användare har olika möjligheter beroende på deras auktoritet på domänen. Symfony har också ett eget säkerhetssystem som man kan använda sig av. För att kunna använda det behöver man dock komponenten security-bundle som kan installeras enligt kommando 10.

Kommando 10: Installera komponenten security-bundle.

```
$ composer require symfony/security-bundle
```

För att få ett ordentligt säkerhetssystem behöver man användare. För att skapa användare måste man göra en användarentitet enligt kommando 11.

Kommando 11: Skapa en User-entitet.

```
$ php bin/console make:user
```

Kommandot startar en kedja med frågor på samma sätt som när man skapar en vanlig entitet. Först ger man namnet på själva användarentiteten. Nästa fråga är om användarens data ska bli sparad i databasen. Som svar på följande fråga ska man sätta in en kolumn, som ska ha ett unikt värde, som till exempel e-postadress eller användarnamn, för att kunna identifiera användaren. Till sist kommer två frågor relaterade till lösenordet, det vill säga ifall användaren ska ha ett lösenord och ifall lösenordet ska krypteras. Det lönar sig att kryptera lösenorden så att ingen ska kunna komma åt dem. Liksom gällande andra entiteter skapas också en entitetsfil till användarentiteten, och liksom gällande andra entiteter ska man göra en till migration så att databasen uppdateras med de nya ändringarna.

När man har skapat en användarentitet kan man börja konfigurera auktoritetsinställningar. I användarentiteten finns det färdigt en kolumn som står för roller. En användare skall kunna ha flera roller. Därför kommer rollvariabeln i form av ett objekt datatyp, vanligtvis i JSON-format. Auktoritetsinställningsfilen hittar man i config/packages och den heter

security.yaml. När man har öppnat inställningsfilen kan man börja konfigurera auktoritetsystemet. Det finns olika sätt på vilka man kan ställa in hur det väljs vem som har tillgång till vad.

Ett enkelt sätt på vilket man kan konfigurera auktoriteter är att använda sig av rutter. Om man till exempel har en rutt som bara administratören ska ha tillgång till, kan man reservera en rutt under den färdiga rubriken access_control, som finns under security-rubriken som är bara för administratörer. Det betyder att bara användare med rollen administratör har tillgång till allt under den rutt man konfigurerat. Se kodexempel 5.

Kodexempel 5: Ett auktoritetsfigurations exempel.

```
access_control:
- { path: ^/admin, roles: ROLE_ADMIN }
```

Ifall man vill vara mera specifik om vilka kontroller-funktioner som klienten/användaren ska ha tillgång till kan man använda sig av isGranted-attributet. Man måste dock referera till attributet i början. Se kodexempel 6.

Kodexempel 6: Demonstration av auktoritetsystem funktionen IsGranted() i en kontroller funktion.

```
use Symfony\Component\Security\Http\Attribute\IsGranted;

class ConferenceController extends AbstractController
{
    #[IsGranted('ROLE_ADMIN')]
    #[Route('/conference', name: 'app_conference')]
    public function index(): Response
    {
        return $this->render('conference/index.html.twig', [
            'controller_name' => 'ConferenceController',
        ]);
    }
}
```

Ett annat sätt på vilket man kan ställa in auktoritetsystem är att man gör rollhierarkier. Detta kan man göra genom att man i security.yaml-filen skriver in vilka andra roller en roll

har tillgång till, som i exemplet nedan där administratören har tillgång både till vanliga användar-rollen och arbetar-rollen:

Kodexempel 7: Exempel på roll hierarkisystem i security.yaml-filen.

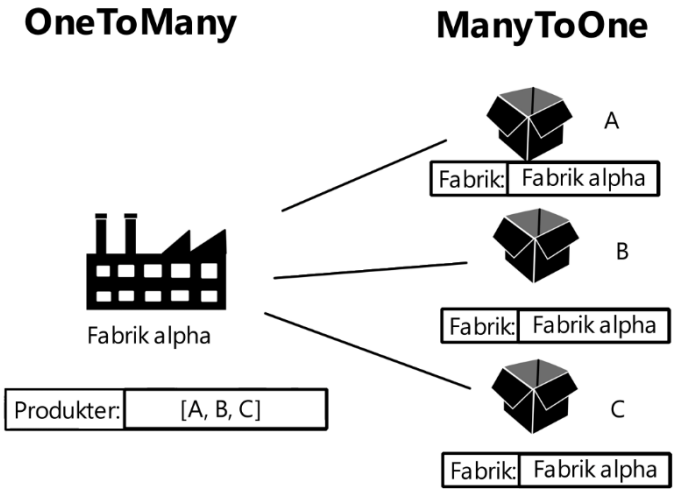
```
role_hierarchy:
  ROLE_ADMIN:       [ROLE_USER,ROLE_WORKER]
```

4.6 Kopplingar

Databaser har ofta olika så kallade kopplingar som kopplar ihop olika tabeller på olika sätt. I Symfony har man också ett eget system som implementeras i deras Doctrine-entiteter. För att implementera kopplingar i en entitet kan man använda sig av samma make-kommando som man skapade själva entiteten med. När man har kört kommandot skriver man in samma namn i prompten som man hade skrivit in då man skapade entiteten. Sedan ska man skriva in ett kolumnnamn för kolumnen som ska visa värden från en annan tabell. Nästa steg är att man ska skriva in relation, koppling på svenska, som datatyp. Nu kommer det en fråga om vilken typ av koppling man vill ha.

I Doctrine finns flera olika typer av kopplingar. Nedan finns exempel på några av dessa kopplingar hur de kan fungera:

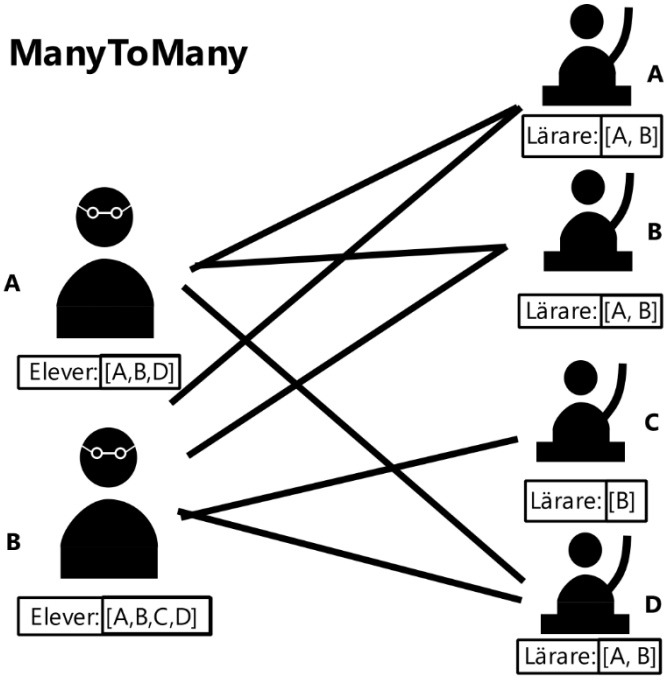
1. Ett exempel på en OneToMany- eller en ManyToOne-koppling kunde vara att en fabrik kan ha flera produkter men en produkt tillhör en enda fabrik. Se figur 2.



Figur 2: Ett exempel på en OneToMany- och ManyToOne-Koppling

2. Ett exempel på en OneToOne-koppling kunde vara en Huvudstad som bara kan tillhöra ett land och landet kan ha bara en huvudstad.

3. ManyToMany- kopplingar är mera ovanliga jämfört med andra typer av kopplingar. Ett exempel på en ManyToMany-koppling kunde vara att en lärare har många elever och en elev har flera lärare. Se figur 3



Figur 3: Ett exempel på en ManyToMany-koppling

En annan sak man måste ta i beaktande ifall kopplingen skall fungera båda vägarna är följande: Det finns två typer av varje koppling, nämligen bidirectional och unidirectional. När en koppling är så kallad unidirectional betyder det att information går bara ena vägen. Ifall man till exempel har en tabell som har en koppling till en annan tabell kan den första tabellen fritt få information från den andra tabellen, medan den andra tabellen inte alls kan få information från den första tabellen, vilket gör kopplingen unidirectional. Då en koppling är bidirectional går kopplingen båda vägarna, vilket betyder att båda tabellerna får ett fält som kan användas för att få information från den motsatta tabellen.

Det är viktigt att man också i detta fall migrerar de ändringar man har gjort på entiteterna till databasen, så att ändringarna kommer i kraft och man kan använda sig av dem. När man har skapat kopplingar kan man börja använda sig av dem för att genom en specifik entitet

få mera information från andra entiteter. Ett sätt genom vilket man snabbt kan få information, till exempel i en ManyToOne, är att man först söker en rad från första tabellen med find() eller använder findAll()-funktionen för att hämta alla värden från en tabell. Ifall man använder sig av findAll()-funktionen kan man direkt använda andra tabellens värde i Twig med hjälp av att man plockar ut andra tabellens värden genom kopplings objektet på samma sätt som andra objekt. Här är samma Twig-kodexempel som förut, men nu finns där värden från en annan tabell via kopplings kolumnen.

Kodexempel 8: Twig-loop som använder sig av en koppling.

```
<table>
  {% for Föremål in ExporteradObjekt %}
  <tr>
    <td>{{Föremål.ColumnEtt}}</td>
    <td>{{Föremål.Kolumntvå}}</td>
    <td>{{Föremål.ExempelTwo.Namn}}
  </tr>
  {% endfor %}
</table>
```

4.7 Webpack med CSS och JavaScript

Ifall man behöver programmera djupare på frontend-sidan och det inte lyckas med de automatiska funktionerna i Symfony, kan det löna sig att använda sig av Javascript. För att använda sig av CSS eller Javascript i ett Symfony-projekt behöver man Webpack encore och en package manager så som npm eller yarn. Här är kommandot för att först installera Webpack encore.

Kommando 12: Installera komponenten Webpack Encore bundles.

```
composer require symfony/webpack-encore-bundle
```

Sedan ska man installera en package manager. Här är kommandon för att installera yarn och npm

Kommando 13: Installera antingen npm eller installerar alla valda npm-komponenter till omgivningen.

```
npm install
```

Kommando 14: in Installera antingen yarn eller installerar alla valda yarn-komponenter till omgivningen.

```
yarn install
```

När man har installerat en av dessa kan man börja programmera med JavaScript och CSS. I assets-mappen som skapats kan man nu börja sätta in JavaScript filer som kan användas för olika sidor. För att testa webpacks JavaScript-funktionalitet kan man till exempel skapa en JavaScript-mapp i assets-mappen och där göra en JavaScript-fil. För att testa JavaScript-filen kan man skriva in en simpel JavaScript-kod, som till exempel använder sig av console.log()-funktionen, för att skriva in en simpel rad i klientens webbläsarkonsol. JavaScript-filer måste dock först kompileras via Webpack för att kunna användas. Därför måste man först registrera den JavaScript-fil man vill använda i webpacks konfigurationsfil, som har skapats i projektets rotmapp.

I den filen, där det redan med hjälp av en .addEntry()-funktion är registrerat för en Javascriptfil med namnet app.js, kan man på samma sätt registrera en Javascriptfil man själv har skapat. Man kan alltså på samma sätt som app registreringen också registrera sina egna Javascriptfiler med .addEntry()-funktioner. Som första parameter ger man alltså Javascriptfilen ett variabelnamn som Webpack kan använda sig av och som andra parameter ger man rutten till själva Javascriptfilen. Till sist måste man ännu kompilera den kod man har tillsatt i Webpackkonfigurationsfilen. Detta kan man göra med följande kommandon i yarn och i npm:

Kommando 15: Kompilera alla JavaScript och CSS filer med hjälp av npm.

```
npm run dev
```

Kommando 16: Kompilera alla JavaScript och CSS filer med hjälp av yarn encore.

```
yarn encore dev
```

Kommando kan man köra så att koden automatiskt kompilerar när man har gjort en ändring:

Kommando 17: Npm kommando som Kompilerar varje gång man gör ändringar i JavaScript eller CSS-filerna.

```
npm run watch
```

Kommando 18: Yarn-kommando som kompilerar varje gång man gör ändringar i JavaScript eller CSS filerna.

```
yarn run watch
```

För att koden skall köra måste man ännu inkludera JavaScript-filen med hjälp av Twig i den HTML-fil som ska använda sig av den. Detta gör man genom att i Head-delen av koden inkludera ett Twig JavaScripts-block, där man använder Twig-funktionen encore_entry_script_tags(), vilken som parameter ska ha den Webpacknamnvariabel man fyllde i Webpackkonfigurationsfilen. Sedan kan man testa ifall det fungerar genom att man startar servern och navigerar till själva sidan som använder sig av javascript. Ifall man använde sig av console.log()-funktionen, märker man ifall den fungerade genom att man kollar webbläsarens egen konsol, som oftast kan öppnas med CTRL + SHIFT + C-knapp kombinationen.

4.8 CRUD, tabelldatahantering och forms

När man har en fungerande tabell i ett webbprojekt, lönar det sig att implementera CRUD-funktionalitet eller create, read, update and delete-funktionalitet. I det här arbetet har detta gjorts manuellt även om det också finns ett automatiskt sätt att göra det inom Symfony. Det har redan tidigare förklarats hur man kan avläsa en tabell, alltså har read-funktionaliteten redan blivit förklarad.

Symfony har en egen komponent som kan användas för att skapa formulär, som heter forms. Komponenten kan installeras med kommandot:

Kommando 19: Installera form-komponenten.

```
composer require symfony/form
```

När man har installerat komponenten kan man börja skapa så kallade Formtype-klasser. Man kan använda sig av ett make-kommando för att automatiskt skapa en Formtype-klassfil genom att man som första parameter namnger formtype-filen och som andra parameter bestämmer vilken entitet som denna Formtype-fil ska använda sig av.

Kommando 20: Skapa en Formtype-fil.

```
Symfony console make:form exempelentitetsform ExempelEntitet
```

När man har skapat en Formtype kan man sätta den som blankett på den sida man vill ha den. Före det måste man dock för det första importera den Formtype man vill ha i kontroller-funktionen som renderar själva sidan. Detta gör man genom att man först inkluderar en rutt till den Formtype man har skapat i kontrollern. Som följande steg skapar man en ny instans av den i kontroller funktionen med funktionen $this->createForm(). Som första parameter ska man sätta in vilken Formtype som funktionen ska använda sig av. Den andra parametern är en variabel på den entitet man använder sig av. Därefter kan man tillsätta den skapade Form-variabeln i objektet i render-funktionen på samma sätt som när man exporterar andra variabler till frontend-sidan.

Då man har inkluderat form-variabeln kan man placera ut den som en blankett på sidan med Twig-funktioner.

För att placera ut blanketten sätter man först in Twig-funktionen {{form_start()}} som början med Form-variabeln man exporterade som parameter. Sedan markerar man slutet på blanketten med {{form_end()}} det vill säga med samma Form-parameter som den första funktionen. Ifall man inte har några andra blankett funktioner mellan markörerna i början och slutet skrivs blanketten automatiskt ut. Ifall man själv vill sätta ut blankettkomponenterna på eget sätt, finns det specifika Twig-funktioner för det. För att placera ut en inmatningslåda kan man använda sig av funktionen {{form_widget()}}. Som parameter kallar man på den kolumn i blankettobjektet man vill ha, i formen formobjekt.kolumn. Om man också vill placera ut titlarna manuellt på sidan kan man använda sig av funktionen {{form_label()}} med samma parameter som i förra funktionen. I Kodexempel 9 ser vi ett exempel på hur en Twig blankett kan skrivas ut.

Kodexempel 9: Exempel på en Twig-blankett.

```
<div>
    {{ form_start(exent_form) }}
    <br>
    {{ form_label(exent_form.ColumnEtt) }}
    {{ form_widget(exent_form.ColumnEtt)}}
    <br>
    {{ form_label(exent_form.KolumnTwo) }}
    {{ form_widget(exent_form.KolumnTwo)}}
    <br>
    {{ form_label(exent_form.ExempelTwo) }}
    {{ form_widget(exent_form.ExempelTwo)}}
    <br>
    <input type="submit" value="">
    {{ form_end(exent_form) }}
</div>
```

4.8.1 Create

Nu kommer förklaringen på hur man implementerar skapande av rader i databasen från klientens sida, alltså create-funktionalitet. I denna guide kommer blanketthanteringen att ske med skilda kontroller-funktioner. För att kunna hantera blanketten med en skild kontroller funktion behövs en skicka-knapp med en så kallad action-tagg. Som parameter i action-taggen kommer rutten till den kontroller-funktion som man använder sig för att

hantera blankettinlämningen. Denna tagg kan man också tillsätta med den kontroller-funktion man skapar själva blanketten med. I kodexempel 10 finns det ett exempel på hur man kan sätta in action-taggen som en parameter när man deklarerar blankettvariabeln i koden.

Kodexempel 10: Demonstration av createForm-funktionen med action-taggen.

```
        $exent_formtype = $this->createForm(ExempelentitetformType::class,
$exent_entity,array( 'action' => '/conference/create'));
```

För att koda själva inlämningsfunktionen kan man börja med att skriva in funktionen i samma kontroller som man renderar själva sidan i. Som parameter för funktionen behöver man en Request-variabel som använder sig av den Request-komponent som ska importeras i början på kontrollern. I denna variabel kommer den inlämnade blankettens information.

Man börjar funktionen med att skapa en ny blankettvariabel och använder sedan blankettens inre funktion, handleRequest, för att få den info den behöver från Request-variabeln, vilken har den skickade informationen. Innan man sparar infon i databasen måste man först kontrollera ifall den information som behövs har kommit. Detta kan man göra genom att använda en If-sats där man sätter blankettvariabelns isSubmitted()-funktion som parameter. Efter detta lönar det sig att kontrollera ifall informationen är giltig för användning. I så fall gör man en likadan If-sats där man i stället sätter in blankettvariabelns isValid()-funktion.

Efter detta kan man börja göra ändringarna i databasen. Det enklaste sättet att sätta in den nya informationen i databasen är att använda sig av blankettfunktionen getData() och sätta informationen från den i en ny variabel.

Efter det behöver man en entityManagerInterface-variabel som parameter i själva kontroller-funktionen. Dessutom behöver man också en referens till EntityManagerInterface från Doctrine. När man har entityManagerInterface-variabeln kan man spara data i databasen med entityManagerInterface-funktionerna persist(), som har som parameter den datavariabel som har all info från blanketten och flush()-funktionen som kör alla ändringar till databasen.

I slutet av funktionen kan man sätta en redirectToRoute()-funktion i returneringen. Denna funktion kan man använda för att ladda om sidan vilket gör att det nya värdet visas upp. Som parameter i funktionen sätter man samma namn som name-taggen i rutten för huvudsidan. Ett exempel på detta är kodexempel 11 nedan.

Kodexempel 11: Demonstration av en Create-kontrollerfunktion.

```
#[Route('/conference/create', name: 'app_conference_create')]
    public function CreateExample( EntityManagerInterface $entityManager,
Request $request ) : Response
    {


        $exent_entity = new ExempelEntitet();


        $exent_formtype = $this->createForm(
            ExempelentitetformType::class,
            $exent_entity);

        $exent_formtype -> handleRequest($request);

        if($exent_formtype-> isSubmitted()){
            if($exent_formtype-> isValid()){

                $exent_entity_new = $exent_formtype -> getData();

                $entityManager->persist($exent_entity_new);
                $entityManager->flush();
            }
        }


        return $this-> redirectToRoute('app_conference')
```

4.8.2 Delete

Ifall man vill ha ett sätt att ta bort ett värde i databasen behöver man en kontroller-funktion som tar in en identifierare för att kunna hitta rätt rad att ta bort.

Ett lätt sätt att få bara en identifierare till kontroller-funktionen är att man skickar identifieraren via kontroller-funktionens sökväg. Detta kan göras genom att man sätter med en variabel inkapslad i {}-tecken i slutet på rutten. När man har en identifierare kan man använda sig av den för att hitta rätt rad med find()-funktionen, som finns i repository:n

för den entitet som tabellen använder sig av. Därefter kan man använda sig av remove()-funktionen i EntityManagerInterface. På samma sätt som i create-funktionen ska man ännu använda sig av flush()-funktionen för att ändringarna ska ske i databasen.

Kodexempel 12: Ett exempel på hur en Delete-funktion kan se ut.

```
#[Route('/conference/delete/{exentitet}', name: 'app_conference_delete')]
    public function DeleteExample($exentitet, ExempelEntitetRepository $ex-
empel_entitet_repo, EntityManagerInterface $entityManager ) : Response
    {

        $exempel_entitet = $exempel_entitet_repo -> find($exentitet);

        $entityManager->remove($exempel_entitet);
        $entityManager->flush();

        return $this-> redirectToRoute('app_conference');
    }
```

När man vill skicka med en identifierare i kontroller-funktionens URL kan man använda sig av Twig för att sätta in en Id i en href-länk, på samma sätt som man skriver ut andra importerade värden.

Kodexempel 13: Ett exempel på samma Twig-loop som tidigare, men med möjlighet att ta bort varje rad.

```
<table>
    {% for Föremål in ExporteradObjekt %}
        <tr>
            <td>{{Föremål.ColumnEtt}}</td>
            <td>{{Föremål.Kolumntwo}}</td>
            <td>{{Föremål.ExempelTwo.Namn}}</td>
            <td>
                <a href="/conference/delete/{{Föremål.id}}">Ta bort</a>
            </td>
        </tr>
    {% endfor %}
</table>
```

4.8.3 Edit

I kodexempel 14 är ett exempel på hur man skapar redigeringsfunktionalitet i arbetet. Man kommer att kunna välja en rad i tabellen där man kan trycka på en redigeringsknapp. Redigeringsknappen öppnar en ny sida med en blankett. Där kan man göra de ändringar man vill göra och sedan skicka in blanketten. Ifall inlämningen lyckas så skickas användaren tillbaka till huvudsidan.

Man kan börja processen med att göra en ny html.twig-sida. Där kan man göra en blankett för den entitet man använder sig av. Man kan också sätta in de föregående värdena i blankettinmatningarnas value-attribut via Twig.

Kodexempel 14: Ett exempel på ett Edit-blankett.

```
{{ form_start(exent_form) }}
    <br>
    {{ form_label(exent_form.ColumnEtt) }}
    {{ form_widget(exent_form.ColumnEtt,{
        'attr': {
            'value': ExporteradObjekt.ColumnEtt
        }
    })}}
    <br>
    {{ form_label(exent_form.KolumnTwo) }}
    {{ form_widget(exent_form.KolumnTwo,{
        'attr': {
            'value': ExporteradObjekt.KolumnTwo
        }
    })}}
    <br>
    {{ form_label(exent_form.ExempelTwo) }}
    {{ form_widget(exent_form.ExempelTwo,{
        'attr': {
            'value': ExporteradObjekt.ExempelTwo.namn
        }
    })}}
    <br>
    <input type="submit" value="Submit">
{{ form_end(exent_form) }}
```

Därefter kan man börja skapa en kontroller-funktion för redigeringen. Det lättaste sättet är att kombinera create-funktionen, delete-funktionen och funktionen som laddar upp huvudsidan. På samma sätt som i delete-funktionen kan man ta in en identifierare via URL-

adressen. Denna kan man använda för att hitta rätt entitet genom entitetens repository-fil. Man kan sedan sätta in den som andra parameter när man deklarerar entitetens formtype. Därefter kan man skriva fortsättningen på samma sätt som i create-funktionen ända till slutet av validerings if-satsen. Man kan flytta dit redirectToRoute()-returfunktionen så att man automatiskt flyttas tillbaka till huvudsidan när man har skickat in blanketten. I slutet på funktionen kan man i stället sätta en render-returfunktion som renderar sidan. Man lägger också med blanketten till sidan och den valda radens entitet.

Kodexempel 15: Ett exempel på en Edit kontrollerfunktion.

```
#[Route('/conference/edit/{exentitet}', name: 'app_conference_edit')]
public function EditExample(
    $exentitet,
    EntityManagerInterface $entityManager,
    Request $request,
    ExempelEntitetRepository $exempel_entitet_repo
) : Response
{
    $exent = $exempel_entitet_repo->find($exentitet);
    $exent_formtype = $this->createForm(
        ExempelentitetformType::class,
        $exent,
    );
    $exent_formtype -> handleRequest($request);
    if($exent_formtype-> isSubmitted()){
        if($exent_formtype-> isValid()){

            $exent_entity = $exent_formtype -> getData();
            $entityManager->persist($exent_entity);
            $entityManager->flush();

            return $this->redirectToRoute('app_conference');
        }
    }
    return $this->render('conference/Edit.html.twig', [
        'ExporteradObjekt' => $exent,
        'exent_form' => $exent_formtype,
    ])   ;
}
```

Allt som ännu behöver göras är att sätta in redigera-knappar i huvudsidans Twig-loop. Dessa kan man sätta in på samma sätt som i delete-exemplet.

5 Arbetet

I detta kapitel presenteras arbetsprocessen av skapandet av en webbapplikation under examensarbetet.

5.1 Symfony-projektets start (Grunden till projektet)

Jag började utvecklingen av projektet genom att göra en databas med PostgreSQL. Sedan skapade jag själva Symfony-projektet där jag installerade baskomponenterna för Symfony. Därefter skapade jag den första kontrollern, vars funktion är att behandla informationen från en av tabellerna. Jag anslöt databasen till projektet och skapade den första entiteten. Sedan satte jag in en simpel dummydata-tabell i databasen med hjälp av databasfixturer (Fixtures). Jag använde den skapta entiteten som modell. I detta skede skapade jag också en enkel frontend sida.

5.2 Första tabellen

Jag gjorde därefter den första tabellsidan på frontend-sidan. Detta gjorde jag på det sättet att backend-sidan först läser databasen med de funktioner som färdigt finns i den automatiskt skapade repository-filen. När man sedan har datan kan man skicka den via render-funktionen som en Array-parameter. När man gjort så består sidorna av twig.html-filer, som är HTML sidor där man kan sätta in data. Jag har nu alltså gjort en liten enkel Twig-loop, som gör en ny HTML-tabellrad med data för varje rad från datan som den mottar från backend- sidan. Jag har på samma gång också gjort en extra sida som visar infon för varje specifik rad som listar ut de olika kolumnernas värden.

5.2.1 Entitetskopplingar och säkerhetssystemets start

Som följande sak började jag sätta med andra entiteter som jag skulle ha med. Därefter började jag i allt större utsträckning integrera säkerhetssystemet, som använder sig av

Security-komponenten för Symfony. För detta ändamål behövde jag också skapa en ny entitet för användare, så att jag skulle kunna spara användare på databasen utan att behöva använda mig av temporära användare.

Följande steg var att försöka implementera kopplingar mellan de olika entiteterna i programmet. För att få detta gjort uppdaterade jag entiteterna, som skulle ha kopplingar med Doctrine-projektets Make-kommando. Jag använde mig av Make-kommandot för att implementera Doctrine-kopplingarna, så som ManyToOne- kopplingen, och på samma gång satt jag också med tidsbundna kolumner in i entiteterna. Dessutom uppdaterade jag också databasfixturerna så att de refererade också till andra tabeller.

Efter arbetet med kopplingarna började jag jobba mera med användar-komponenterna. Jag satte med två färdiga användare in i fixturerna och gjorde också ett sätt att skapa användare med hjälp av konsol-kommandon. Sedan började jag testa programmet genom att använda roller för användare. Jag hade i detta skede två roller, det vill säga administratör och användare, men dessa hade inte ännu i detta skede en hierarkisk ordning.

En annan sak som jag också gjorde var att använda TWIG-funktioner för att göra användargränssnittet olika för användaren och för administratören. Med denna funktion kunde jag också göra två knappar, en för att logga in, som visas då man inte är inloggad, och den andra för att logga ut när man är inloggad.

Jag tillsatte tillägget blameable entity. Tilläggets funktion är att det alltid visar vem som har gjort en ändring i en entitet. När jag använde blameable entity satte jag med den i alla entiteter så att det skulle framgå vem som hade gjort ändringar.

Jag tillsatte också tillägget timestampable och använde det i alla entiteter som jag har. Den skapar automatiskt kolumner i entiteten för att visa när en rad har skapats eller uppdaterats.

5.3 Webpack och sidstil

För att programmet ska se professionellt ut ska det ha en lämplig stil. För att kunna kompilera CSS och JavaScript måste jag använda mig av Webpack. För att lätt få ett bra rutsystem för alla webbelement installerade jag också Bootstrap, som har ett lämpligt dynamiskt kolumnsystem för webbelement så att man lätt kan göra ett sidofält på sidan. I det här skedet uppdaterade jag också administratörsidan, som nu bara kan öppnas av någon som har en administratörroll, eftersom jag nu också hade implementerat hierarki-systemet i konfigurationen av säkerhetsmodulen. Jag ändrade också användarentiteten så att den använde sig av e-postadress i stället för användarnamn. Jag ändrade sedan sidofältet så att det skulle ändra storlek när man ändrade storleken på rutan.

5.4 Tabellsidor för administration

Till projektet skulle det göras tre tabeller som bara administratören har tillgång till. I början satt jag alla tabellerna för administratörerna på samma sida. Man kunde välja mellan olika tabellobjekt där man senare kunde editera eller ta bort olika rader. Jag gjorde också en skapa-knapp för varje tabell. Genom att trycka på knappen öppnas en Bootstrap modal-ruta, där man kan skriva in allt som skall vara med i det nya tabellobjektet som skapas.

5.5 Implementering av DataTables

Tabellerna behövde få en lämplig stil och jag blev rekommenderad att använda JavaScript biblioteket DataTables. Med detta kodbibliotek kan man lätt göra tabeller som ser representabla ut och som dessutom fungerar bra tillsammans med AJAX för att man enkelt ska kunna kunna skicka olika sök-, filtrering- och sorteringsförfrågningar till backenden. I detta skede implementerade jag inte ännu någon AJAX funktionalitet utan använde mig bara av tabellen som finns på frontend-sidan.

5.6 Implementering av Metronic

För att snabbare få upp webbapplikationens utseende till den standard som arbetsgivaren önskade behövde jag använda mig av ramverket Metronic. I det skede då jag började arbetet fanns det inte riktigt någon version av Metronic, som var designad att fungera med Symfony, vilket ledde till att jag valde en version som använder sig av Vue. Detta ledde till en svår process när jag försökte sammanfoga två olika projekt som inte fungerade på samma sätt. Det innebar också ett stort problem på grund av att det mesta av koden är i JavaScript och det tar en otrolig lång tid för webpack att kunna kompilera koden. Det som vanligtvis skulle ta ca 20 sekunder tog nu mellan en halv timme och över en timme för att kompileras. Detta ledde till att en lång del av arbetsprocessen bestod av att jag först försökte åtgärda ett problem i javascripten, varefter jag hamnade att kompilera under ca 40 minuter i genomsnitt. Därefter uppstod det ett nytt fel, varefter jag analyserade problemet och sedan repeterade hela processen.

Men som tur släppte Metronic i något skede ut en ny version som är gjord för att använda med Symfony. Detta underlättade arbetet massivt, kompileringstiden sänktes tillbaka till ett genomsnitt på 20 sekunder och det blev mycket lättare att försöka sammanfoga systemen.

Det första jag gjorde efter att jag hade fått Metronic implementerat var att sätta in administratörstabellerna i den nya stilen. Jag skapade sedan en ny PHP-kontroller. I stället för att skapa en administratörsida skapade jag en ny sida för varje administratörtabell, där tabellerna fungerade med DataTables.

Efter det här började jag bygga upp projektets huvudtabellsida som vanliga användare skall ha åtkomst till och då använde jag mig igen av DataTables.

I det här skedet slutade flera av elementen som hördes till Metronic fungera, eftersom jag hade en sökväg till en sida som hade flera än ett /-tecken. Det här var problematiskt för om jag till exempel ville ha en huvudkategori i sökvägen såsom /huvudgren så kunde jag inte skriva /huvudgren/undergren utan att det ledde till att jag fick en massa fel och Metronic-element som inte alls visades upp. Jag åtgärdade det här problemet genom att sätta med en ny /-tecken framför några av sökvägarna i Metronic-koden.

5.7 DataTables med Ajax

Att överföra all data på en gång kan vara tungt för en server när det blir frågan om över hundra- eller tusentals rader med data. Därför kan man i stället använda AJAX för att göra en förfrågan från frontend-sidan till backend-sidan om den specifika datan som frontend-sidan kräver, varefter backend-sidan kan göra de operationer som behövs för att få rätt data från databasen.

Min fokus i detta skede var att hantera data med DataTables med hjälp av AJAX. Jag gjorde först en JavaScript-fil till tabellsidan. Där skapade jag ett DataTables-objekt där jag först ställde in den så att den fungerade på serversidan. Sedan gjorde jag ett AJAX-anrop in i objektet till en utvald adress för att få data till tabellen. På backend-sidan tas AJAX-anropet emot på den utvalda adressen. Den utvalda adressen leder till en kontroller-funktion som jag skapade. I funktionen hämtas data via en entityrepository där det finns färdiga doctrine-funktioner för att hämta all data från en tabell i databasen. Sedan skickas resultatet tillbaka till kontroller-funktionen. All inkommande data sparas i en variabel, som man sedan loopar igenom för att skriva ut innehållet i JSON-format, som sedan kan skickas tillbaka till frontend-sidan via AJAX.Nu har DataTables-objektet all data det behöver från backend-sidan och kan automatiskt skriva ut hela tabellen. Jag lade också till några kolumninställningar i DataTables-objektet för att visa i vilken ordning kolumnerna skulle komma.

5.8 Tabellfunktioner och filtrering

Därefter implementerade jag en funktion till tabellen som gör att man när man trycker på en rad kan man läsa extra detaljer om varje rad. I samma uppdatering började jag arbeta med den nya versionen av huvudtabellen, där jag bland annat inkluderade en ny version av tabellen. På samma gång inkluderade jag också några TWIG-översättningstaggar för framtida implementeringar.

För att lätt kunna navigera genom tabeller med mycket data, kan man underlätta det genom att man filtrerar vilken data man vill se. Därför gjorde jag en sökruta för varje

kolumn. För att kunna få nya data till tabellerna måste man göra ett AJAX-anrop, som skickar sökorden till backend-sidan. I anropet tas alla sökningar med, så att doctrine skall kunna plocka rätt data från databasen.

För varje tabell gjorde jag en funktion som tar in alla filtervärden som finns. Sedan kontrolleras det för varje variabel ifall den har ett värde. Ifall den har ett värde körs en doctrine andWhere-funktion, som väljer vilket värde kolumnen skall ha när doctrine söker igenom tabellen. Till sist skickas den rätta datan tillbaka till klienten.

I detta skede uppdaterades tabellen varje gång man tryckte på Enter-tangenten, vilket förosakades av att DataTables automatiskt satte sorteringsknappen i header-delen av tabellen på samma plats som filtreringssökrutorna. Följden av det här var att varje gång man försökte söka något i en kolumn och tryckte på Enter-tangenten, aktiverades sorteringsknappen och skickade ut ett nytt AJAX-anrop till backend-programmet. Först ledde det till min fördel, men förorsakade senare problem då jag försökte implementera sortering i tabellen. I det här skedet hade jag inte ännu märkt att det var sorteringen som förorsakade anropen.

Senare ändrade jag alla filtersökrutor för numeriska värden så att de började fungera som intervallfilter i stället. Först sätter man in ett startvärde och ett slutvärde. Sedan filtreras allting utanför intervallet bort. På backend- sidan tar jag emot båda värdena och gör en doctrine- funktion, som i stället väljer alla värden mellan start och slut. Därefter uppdateras tabellen med de nya sökresultaten.

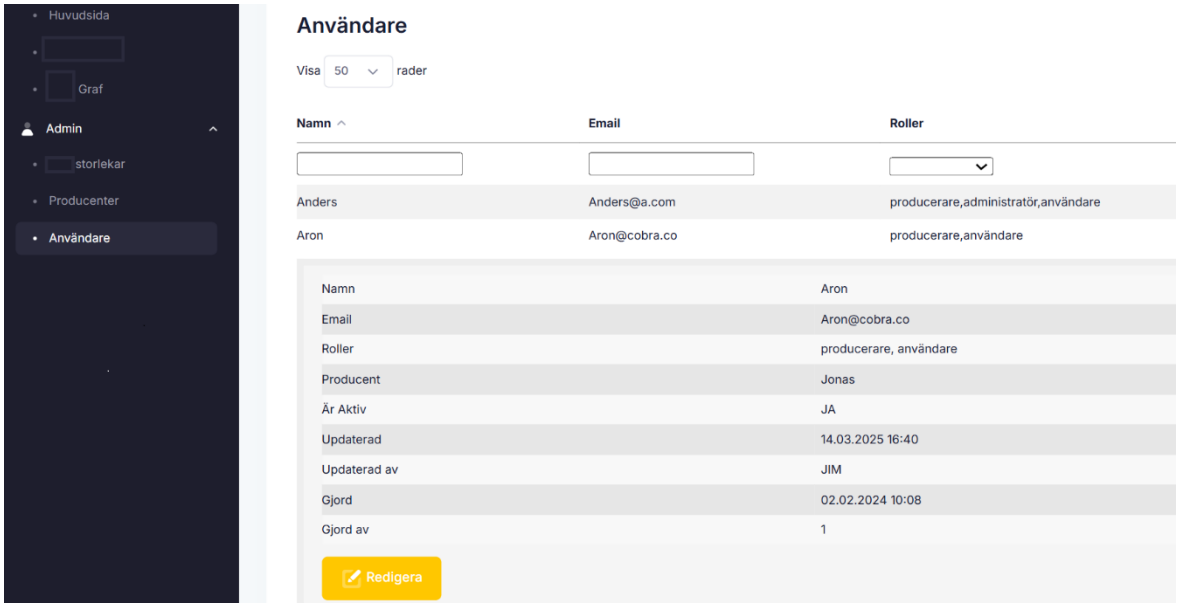
Arbetsgivaren önskade att jag ännu skulle sätta en funktion till tabellen som gör att när man trycker på en rad skapas en ny ruta under raden, där det skrivs ut extra detaljer om raden i fråga. Dessutom önskade arbetsgivaren också jag skulle sätta till en funktion som gör så att om man i denna ruta trycker på en editeringsknapp så ändras rutan till ett formulär, där man kan sätta in nya värden för raden.

Jag började med att göra ändringar i JavaScript-filen för en av tabellsidorna. Först ändrade jag inställningarna för DataTables-tabellens kolumner så att de tillhör en specifik klass. I detta skede började jag använda mig av jQuery för att enkelt kunna referera till olika existerande element på sidan. Jag aktiverade jQuery genom att avkommentera raden .autoProvidejQuery() i webpack.config.js-filen. Därefter tillsatte jag en jQuery konstant i

JavaScript-filen med tecknet-$. Med $-tecknet kunde jag nu använda mig av jQuery funktioner.

När jag hade installerat jQuery gjorde jag en jQuery-funktion som körs när man trycker på ett element med den specifika klassen som tillsattes i kolumnelementet. I detta fall består varje rad i tabellen av ett antal kolumnelement. Det här betyder att jag med jQuery kunde hitta vilken rad användaren hade tryckt på genom att använda jQuery funktionen closest. Funktionen closest kan ta in en webbelementstyp som parameter, såsom tr i detta fall. Detta betyder att funktionen försöker hitta det närmaste tabellradelementet.

När jag då alltså hade en radelementvariabel, kunde jag använda mig av DataTables egen row -funktion, som kan välja en rad i DataTables-objektet. Från resultatet av row-funktionen kan jag också få radens id, som jag skickade som gömt data när jag hämtade data till tabellen. Då jag hade radens id kunde jag göra en AJAX-förfrågan till backend-sidan för att få extra data för den specifika raden. I kontrollern på backend-sidan skickade jag med en html.twig-mall där jag satt med all data jag behövde för raden. Mallen består av en vanlig HTML-tabell där jag hade en kolumn för namnet på raden och en annan kolumn för värdet på raden. Jag satte också in radens id till tabellens id-attribut för senare användning. När frontend-sidan får den data som skickas tillbaka från backend-sidan, skrivs mallen ut under den rad man har valt.



Figur 4: Tabell på användare.

5.9 Widgetar och grafer

Jag experimenterade också med att skapa så kallade kort, som är lite som små rutor på sidan där man kan ha information. De är baserade på Metronics egna exempelkort, som finns framme på deras egen huvudsida med exempel för olika projekt.

Nu började jag också implementera så kallade widgets till programmet, det vill säga små rutor som man kan ha på huvudsidan, som visar olika saker. Jag använde de färdiga korten som modell.

Den första widget jag gjorde tar emot data från olika tabeller för att skriva ut summorna av olika produkter som producerats under dagen. Jag har lagt till en skild TWIG-template, det vill säga en HTML/TWIG-kod som inte visas direkt för användaren, för widgeten som man kan sätta in i ett annat TWIG-dokument.

Datan till widgeten kommer via JavaScript, som skickar ett AJAX-anrop till backend-sidan. Jag hade gjort en kontroller specifikt för widgetar, så att det är lätt att hitta deras backend-koder.

I kontrollern gjorde jag en funktion som tar emot AJAX-anropet och skickar datan till två olika repository-funktioner för att de ska få rätt data. Den första räknar ut produkter för dagen per typ av produkt, medan den andra räknar ut totala mängden av produkter för dagen. Sedan skickas infon tillbaka till frontend-sidan där den listas ut på kortet.

Jag skulle sedan göra en graf som visar total antal producerade produkter per kvartal i ett tidsintervall. Man kan välja mellan perioderna: dagar, veckor och månader. När man väljer en period visar grafen en mängd totalvärden för varje period.

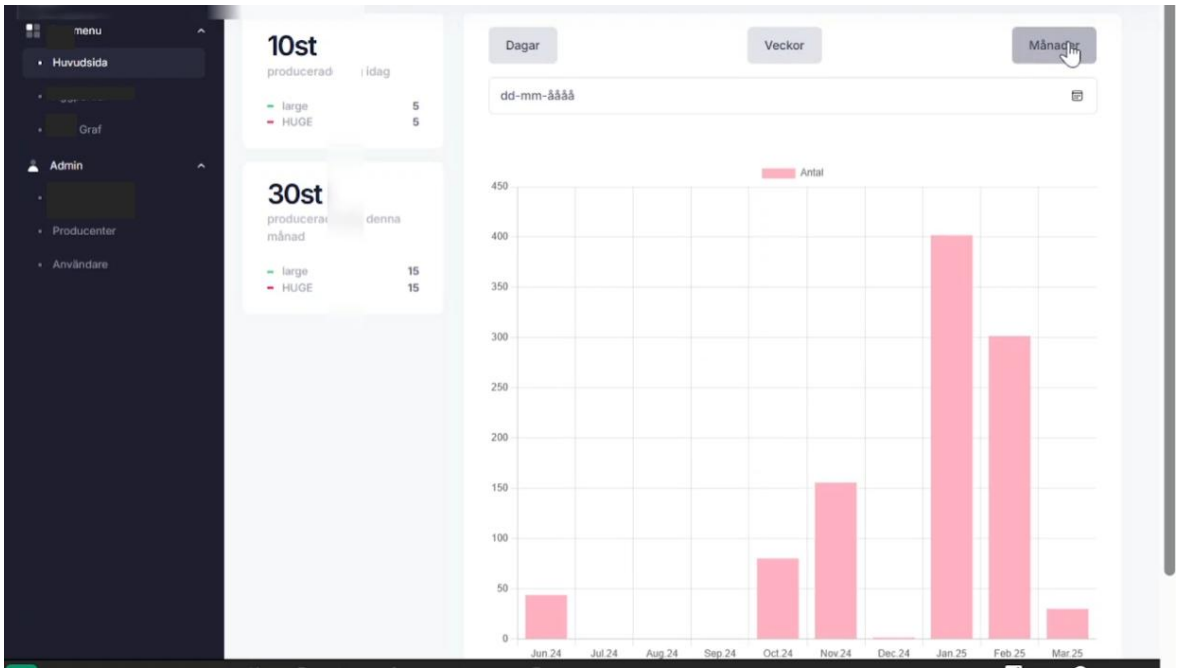
Jag gjorde också på en skild sida en mera avancerad graf. Den visar värden på samma sätt som förra grafen med skillnaden att den också visar antalet för varje typ av produkt. Den använde sig av fyllda, staplade linjediagram som kunde konfigureras i Chart.js-objektet. Först använde jag mig av slumpmässiga färger för att urskilja de olika produkterna, men fick sedan rekommendationen att jag i stället skulle använda mig av en lista på olika färger, som kommer i en viss ordning, för att urskilja olika typer av produkter i grafen.

En annan sak som också skulle finnas med i grafen var total mängden för varje tidsintervall. Chart.js innehåller färdigt en egenskap som gör att alla summor på de olika produkterna

listas ut i en ruta, när man håller musen ovanför en tidsintervall. Rutan kan konfigureras i options-objektet som finns i chart.js-objektet. Jag gjorde alltså en liten loop som summerar mängden av alla produkter och insatte resultatet i slutet av produktlistrutan.

Dessa grafer är gjorda med chartjs, ett JavaScript grafbibliotek. Graferna tar data från backend-sidan och konfigureras på frontend-sidan

I slutet gjorde jag ytterligare en version av widgeten som visar dagens produkter. Men i stället för att visa dagliga produkter visar den alla produkters summor under en månad



Figur 5: Huvudsidan med widgetar.

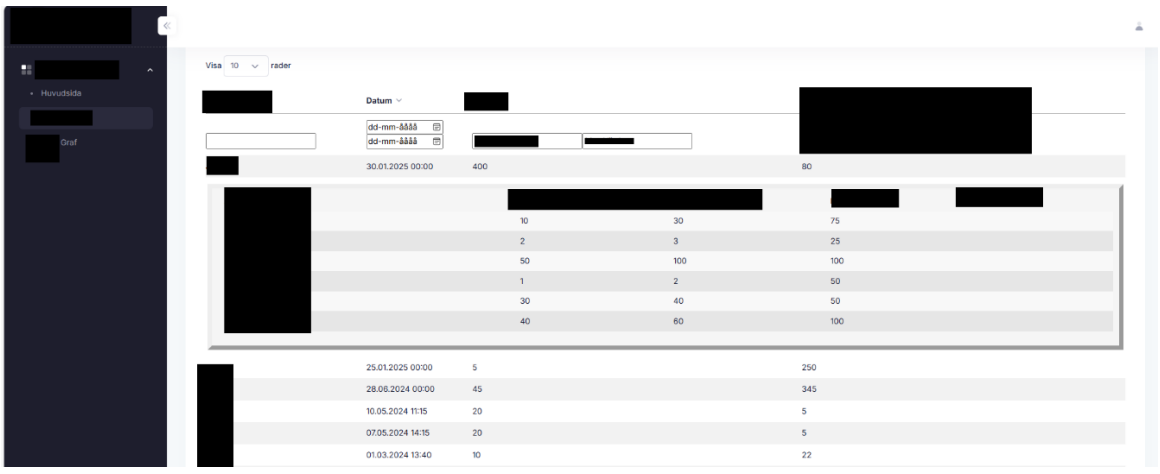
6 Resultat och konklusion

I följande kapitel presenteras de konkluderande delarna av arbetet. Först kommer resultatet som innehåller vad min uppgift har varit och vad jag har åstadkommit under examensarbetet. Nästa underkapitel handlar lite om de olika utmaningar och problem jag hade när detta arbete utfördes. Till sist kommer ett kort underkapitel som innehåller lite utvecklingsmöjligheter för programmet som har skapats under detta arbete.

6.1 Resultat

I det här arbetet har jag använt Symfony för att skapa en webbapplikation på uppdrag av Nyholm Solutions Oy Ab. Jag gjorde en webbapplikation med fungerande databassystem och databashantering. Applikationen har också ett säkerhetssystem där det kan finnas användare som kan ha olika roller med olika auktoriteter. Dessutom har applikationen sidor som bara användare med en specifik roll kan se. Det finns flera olika tabeller som visar olika produkter, användare och annat. I dessa tabeller kan man trycka på varje enskild rad för att få mera detaljer om dem. Man kan också filtrera sökresultat genom sökrutor som finns i varje kolumn. I vissa tabeller finns det möjlighet att skapa och editera olika rader. Det finns också två olika grafer med en mängd detaljer. I anslutning till graferna kan man välja ett visst datum och en tidsintervall för att se uppgifter om en längre eller kortare period, mätt i dagar, veckor eller månader bakåt från det datum man har valt. Slutligen finns det också två enkla widgetar på huvudsidan där den ena mäter mängden producerade produkter för en dag och den andra för en månad.

Under detta projekt har jag lärt mig att använda mig av språket PHP och ramverket Symfony. Jag har fått en bättre förståelse av hur man kan använder sig av AJAX, hur man skapar en struktur med Bootstrap och hur man kan använda sig av ett frontend-ramverk för att för att bli av med extra arbete.



Figur 6: Huvudtabellen från en vanlig användares perspektiv.

6.2 Problem och utmaningar

En av de större utmaningarna var att implementera Metronic i Symfony. Då jag första gången skulle implementera Metronic fanns ingen version av Metronic som var gjord för Symfony. Jag blev rekommenderad att använda en version av Metronic som var gjord för att användas med Vue. Jag försökte sätta in frontend- delen i mitt projekt, men eftersom sidan var fokuserad på att vara kodad i Vue och mycket av koden var skriven i JavaScript ledde detta till att när jag försökte kompilera koden tog det en halv timme för varje kompilering. Det uppstod också ett oräkneligt antal kompileringsfel. Det här ledde till en långvarig process där jag först kompilerade och sedan väntade cirka en halv timme. Sedan kom det ett kompilerings fel, varefter jag fixade problemet, kompilerade igen och sedan kom nästa fel. Denna process tog för lång tid. Till all lycka visade det sig att Metronic under arbetets gång hade gett ut en ny version av Metronic som använder sig av Symfony. Jag bytte då till den versionen i stället.

6.3 Utvecklingsmöjligheter

En API som kan ta emot data från en maskin och lagra denna data i databasen ska ännu göras till programmet men jag har inte ännu fått tillgång till den maskinen så jag har inte ännu skapat den under den tid som jag skrev det här examensarbetet.

7 Källförteckning

Adermann, N., Boggiano, J., & others. (u.d.). *Documentation*. Hämtat från Composer A Depency Manager for PHP: https://getcomposer.org/doc

Anonymous. (23.01.2025). *Frequently Asked Questions*. Hämtat från vuejs.org: https://vuejs.org/about/faq

Johnson, R. (1997). Frameworks = (Components + Patterns). *Commun. ACM*, 4.

Mehdi, A., Friedhelm, B., Anthony, D., Nuno, L., Hannes, M., Georg, R., . . . Jakub, V. (den 4 1 2024). *PHP Manual.* Hämtat från php.net.

Palaksinghal9903. (10.9.2024). *Frontend vs Backend*. Hämtat från GeeksforGeeks: https://www.geeksforgeeks.org/frontend-vs-backend/

PostgresSQL Global Developement Group. (2024). *postgreSQL 17.0 Documentation.* Hämtat från postgreSQL:

https://www.postgresql.org/files/documentation/pdf/17/postgresql-17-US.pdf

Ragnarsson, A. ó. (2008). Importance of Design Patterns and Frameworks for Software Development. *Tölvumál.*, 4.

Rehle, D. (2000). *Framework Design: A Role Modeling Approach.* Zürich: Swiss Federal Institute of Technology at Zurich.

SensioLabs. (15.11.2024). *Twig for Template Designers*. Hämtat från Symfony Twig: https://twig.symfony.com/doc/3.x/templates.html

SensioLabs. (2018). *Introduction*. Hämtat från Twig Symfony: https://twig.symfony.com/doc/3.x/intro.html

Shambhavi, R., Clinton, D., & Manish, A. (2023). *Fundamentals of information Technology.* Tampa: University of South Florida.

Vyas, R. (2022). Comparatie analysis on front-end frameworks for web applications. *International Journal for Reasearch in Applied Science and Engineering Technology*, 298-307.

Wojciech, B. (2015). *Symfony2 Essentials.* Pact Pub Ltd.

Worsley, J., & D.Drake, J. (2002). *Practical PostgreSQL.* Sebastopol,CA: O'reilly & Associates, Inc.