YRKESHÖGSKOLAN
NOVIA

Projekt Praktik 2.0

Webbapplikation för kontakt mellan studeranden och företag

Snellman Björn

2024 Examensarbete för ingenjörsexamen (YH)

El- och automationsteknik

Vasa 2024

EXAMENSARBETE

Författare: Snellman Björn
Utbildning och ort: El och automationsteknik. Vasa
Inriktning: Informationsteknik
Handledare: Kaj Wikman

Titel: Projekt praktik 2.0

Datum: 14.02.2024 Sidantal: 34

Abstrakt

Projekt Praktik 2.0 är ett projekt som startades vid yrkeshögskolan Novia på höstterminen 2021. Projektets syfte var att skapa en webbportal för studeranden och företag. Projektet påbörjades av en grupp av tre studeranden. Studeranden anlitades av Novia FUI för att utveckla en applikation åt Novia.

Gruppen av studerande som deltagit i detta projekt innefattar mig själv och två andra studeranden som har haft som huvudsakliga uppgift att utveckla och designa servermiljön och autentisering för Novia användare och företag. De har hanterat serverutveckling, inloggningsfunktionalitet och databasadministration.

Min uppgift i projektet har varit att utveckla det grafiska användargränssnittet som användare kommer att hantera då de använder sig utav portalen. Jag skulle designa alla sidor och använda mig utav de verktyg som finns tillgängliga från både de ramverken jag arbetar med och de serverresurser som designats åt mig av mina kollegor.

Portalens huvudsakliga syfte var att simplifiera processen för att hitta bra studie och arbetsrelaterade lösningar för studeranden och företag.

För att skapa en sådan webbapplikation som efterfrågats så har vi valt som grupp att använda oss utav körningsmiljön Node.JS. Node.JS är baserat på JavaScript och hanterar servermiljön och databashantering. För att skapa användargränssnittet så har vi valt att använda ramverket Vue.js. Vue.js är ett JavaScript ramverk som tillåter en att enkelt skapa användargränssnitt med integrering av JavaScript och html i samma filer samt enkel tillgång till en egen server-testmiljö. Vårt projekt är nästan 100% baserat på JavaScript.

För att lagra och hantera data så använder vi oss utav en SQLite databas. SQLite är en SQL databasmotor som är baserad på c-kod.

Språk: svenska
Nyckelord: Node.js, Vue.js, GUI, Frontend, Backend

OPINNÄYTETYÖ

Tekijä: Snellman Björn
Koulutus ja paikkakunta: Sähkö ja automaatioinsinööri.
Suuntautumisvaihtoehto: Tietotekniikka
Ohjaaja(t): Kaj Wikman

Nimike: Projekt praktik 2.0

Päivämäärä: 14.02.2024 Sivumäärä: 34

Tiivistelmä

Projekt Praktik 2.0 käynnistyi Novia-ammattikorkeakoulussa syyslukukaudella 2021. Sen tarkoituksena oli luoda verkkoportaali opiskelijoille ja yrityksille. Projektin aloitti kolmen opiskelijan ryhmä, jonka Novia FUI palkkasi sovelluksen kehitystyöhön.

Päätehtävänä on ollut kehittää ja suunnitella palvelinympäristö ja tunnistautuminen Novian käyttäjille ja yrityksille. Hoidettiin palvelinkehitys, kirjautumistoiminnot ja tietokannan hallinta.

Hankkeessa annettu tehtävä on ollut kehittää graafista käyttöliittymää, jonka avulla portaalia käytetään. Kaikki sivut suunniteltiin käyttämällä käytettävissä olevia työkaluja ja palvelinresursseja, jotka muu ryhmä suunnitteli.

Portaalin päätarkoituksena oli yksinkertaistaa hyvien opiskeluun ja työhön liittyvien ratkaisujen löytäminen opiskelijoille ja yrityksille.

Pyydetyn verkkosovelluksen luomiseksi päätettiin käyttää Node.JS-toteutumisympäristöä. Node.JS perustuu JavaScriptiin ja hoitaa palvelinympäristöä ja tietokannan hallintaa. Käyttöliittymän luomiseen valittiin Vue.JS-kehys. Vue.js on JavaScript-kehys, jonka avulla voidaan helposti luoda käyttöliittymiä integroimalla JavaScript ja html samoihin tiedostoihin ja käyttämällä helposti omaa palvelimen testiympäristöä. Projekti perustuu lähes sataprosenttisesti JavaScriptiin.

Tietojen tallentamiseen ja hallintaan käytettiin SQLite-tietokantaa. SQLite on c-koodiin perustuva SQL-tietokantamoottori.

Kieli: Ruotsi
Avainsanat: Node.js, Vue.js, GUI, Frontend, Backend

BACHELOR'S THESIS

Author: Snellman Björn
Degree Programme: Electrical Engineering and Automation.
Specialization: Information Technology
Supervisor(s): Kaj Wikman

Title: Projekt Praktik 2.0

Date: 14.02.2024 Number of pages: 34

Abstract

Projekt Praktik 2.0 is a project started at Novia University of Applied Sciences in the autumn term 2021. The purpose of the project was to create a web portal for students and companies. The project was started by a group of three students. The students were hired by Novia FUI to develop an application for Novia.

The group of students involved in this project included me and two other students who had had the main tasks of developing and designing the server environment and authentication for Novia users and companies. They managed server development, login functionality and database administration.

My task in the project was to develop the graphical user interface that users deal with using the portal. I designed all the pages and made use of the tools available from both the frameworks I work with, and the server resources designed for me by my colleagues. The main purpose of the portal was to simplify the process of finding good study and work-related solutions for students and companies.

To create the web application requested, the Node.JS runtime environment was chosen. Node.JS is based on JavaScript and manages the server environment and database management. To create the user interface, we chose to use the Vue.js framework. Vue.js is a JavaScript framework that allows one to easily create user interfaces with the integration of JavaScript and HTML in the same files, it also provides an easy-to-access server test environment. Our project was almost 100% based on JavaScript. To store and manage data, an SQLite database was used. SQLite is a SQL database engine based on c-code.

Language: Swedish
Key words: Node.js, Vue.js, GUI, Frontend, Backend

Innehållsförteckning

1 Inledning

Uppdraget som detta projektarbete var ämnat att utföra var att skapa en fullt fungerande webbapplikation skapad från grunden. Målet med projektet var att skapa en webbportal där företag, organisation och studerande skulle kunna komma i kontakt med varandra. Studerande ska kunna se olika sorters uppdrag av olika slag som till exempel projekt, slutarbeten, praktik, sommarjobb med mera. Från företagssidan ska det gå att se studerandens profiler med information om vilken linje, år och cv så att det skulle kunna gå ta direkt kontakt med studeranden vid behov.

Projektet i fråga visade sig vara ganska ambitiöst och för att underlätta att det inte skulle falla på en person som skulle ha som uppgift att skapa allting så valdes det av uppdragsgivaren att detta skulle vara ett projektarbete. Således fick tre studeranden som uppgift att designa denna portal.

Vid projektets start så valdes det att två studerandes i huvudsak skulle ha som uppgift att skapa servermiljön som projektet skulle baseras på samt organisera kontakten mellan projektet och de resurser från Novia som skulle vara till behov för att portalen skulle fungera.

Jag skulle ha som huvudsakliga uppgift att designa det grafiska användargränssnittet som alla användare av portalen skulle hantera då portalen väl kommit i bruk. Utöver det grafiska användargränssnittet så fick jag även lov att göra ändringar på servern och databasen på sådant vis att förändringarna skulle vara till nytta för det grafiska användargränssnittet.

Detta examensarbete kommer att gå igenom hur applikationen är designad hur den fungerar från server och databassidan ända fram till användarens upplevelse av det grafiska användargränssnittet.

2 Teori och teknik

Detta är ett tekniskt examensarbete med många begrepp som kan vara svåra att förstå sig på vid första ögonkastet. I denna del kommer relevanta ord, begrepp, tekniker och terminologier att presenteras för att underlätta vid läsning av detta dokument.

2.1 Frontend, backend och GUI

Programmeringen av en webbsida brukar delas in i två olika delar, dessa delar är en frontend och en backend. En GUI (graphical user interface) skapas i frontend och syftar på det grafiska användargränssnittet. En GUI är det som användaren "ser" då användaren hanterar portalen i en webbläsare. I backend delen av portalen kodas server och databashanteringen.

2.1.1 Frontend

Frontend är en term som syftar på den grafiska delen av ett projekt. Det är på frontend delen av ett projekt där en webbsidas GUI kodas. Frontend utvecklare skapar det grafiska användargränssnittet (GUI) som användaren ser då de hanterar webbsidan. Metoder som används utav frontend delen av ett projekt till exempel: HTLM, CSS och JavaScript.

2.1.2 Backend

Backend-utveckling syftar på server delen av en webbsida. Denna del är nästan aldrig synlig för en vanlig användare på en webbsida. Det är i backend som webbsidans server kodas, databaser hanteras och andra grundläggande funktioner placeras.

För att en webbsida ska kunna fungera så behöver det finnas en backend. Frontend och det grafiska användargränssnittet existerar för att en användare enklare ska kunna kommunicera med och hantera de resurser som finns i backend.

2.2 JavaScript

Portalen som skapats i detta projekt är baserat nästan helt på olika ramverk som härstammar från JavaScript. JavaScript är ett dynamiskt programmeringsspråk som används primärt i webbläsare. Med hjälp av JavaScript går det att skapa interaktiva webbsidor för att manipulera html element och annan data.

JavaScript kan användas för att hantera data som sänds till och från en server. Med hjälp av olika ramverk som skapats baserat på JavaScript så går det dessutom använda scriptspråket på server delen av en applikation. Detta projekt är baserat på ett sådant JavaScript ramverk. Ramverket som används av praktikportalen heter Node.js. (IT-Högskolan, 2023).

2.3 JSON

JSON (JavaScript Object Notation) är ett enkelt dataformat som härstammar från JavaScript. Json data kommer i form av vanlig text, strukturerat på ett sätt som väldigt mycket liknar standard JavaScript datatypen **Object**. Json används för att skicka data internt och mellan datorer. Json går att använda sig utav i ett flertal olika programmeringsspråk. (json.org).

2.4 Node.js

Projekt praktik 2.0 är ämnat att vara en webbportal som är användbar för studeranden och företag. Portalen behöver kunna hantera data och spara data i en databas. Portalen behöver vara enkel att använda, flexibel och online. Det finns en rad olika alternativ som skulle kunna användas för att skapa en sådan applikation. Till detta projekt så valdes Node.js som grund för portalen.

Node.js är ett open source, crossplattform JavaScript ramverk. Node.js körs på **"V8 JavaScript motorn"** som är samma motor som Google Chrome är baserad på. Vad V8 motorn gör är att den tar JavaScript kod från en webbläsare och sedan kör koden.

Motorn är en självständig del från webbläsaren där var den körs, det är denna egenskap som tillät skapandet av Node.js. Node.js fungerar på så vis att V8 motorn kör JavaScript kod på en server i stället för på en webbläsare.

Vid skapandet av en webbsida kodas oftast bara frontend och det grafiska användargränssnittet **(client side)** i JavaScript. Fördelen med Node.js är att det också går att skapa sin backend **(server side**) i JavaScript. Med Node.js behöver programmerare inte lära sig ett annat programmeringsspråk utan programmeringen av en hel webbsida går att göra nästan helt på JavaScript. (The OpenJS Foundation, n.d.).

2.5 Vue.js

Vue.js är ett JavaScript ramverk som används för att designa webbapplikationer. Vue är ett ramverk som integrerar JavaScript och html element tillsammans i en fil, traditionellt så brukar Html och JavaScript hållas separerade i varsin fil.

I traditionella webapplikationer så finns Html och JavaScript separerade i olika filer, Detta är inte fallet med Vue.js. Det går skapa JavaScript funktioner och variabler samt Html element i samma kod, på så vis går det att skapa en effektiv fil som hanterar alla dessa element samtidigt.

Det går fortfarande att skapa traditionella JavaScript filer som går att inkludera i en Vue-fil, detta vore lämpligt då ett script designas som bör användas utav fler än en fil.

Vue.js har serverfunktionalitet som kan minimera och kompilerar sin GUI och överföra sin kod från frontend till backend. Det finns ett enkelt kommandogränssnitt för att enkelt starta nya Vue-projekt.

Vue.js har en egen testserver där uppdateringar och förändringar på koden går att se i realtid, detta är behändigt för att testa och se små förändringar utan att behöva kompilera samtliga filer på nytt.

En Vue.js fil är väldigt lik en Html-fil, men har dessutom skilda delar dedikerade för JavaScript och CSS (Cascading Style Sheet, vilket är ett programmeringsspråk för att manipulera stilen av html-element). En Vue.js fil slutar med extensionen “Vue” i slutet av filnamnet. (Vue.js, n.d.).

2.6 Axios.js

Axios.js är ett frontend JavaScript bibliotek i Node.js som tillåter effektiv hantering av Http-anrop till och från en mottagare. Mottagaren kan vara till exempel en websida, en molnserver eller i detta projekts fall en lokal server som finns i backend.

Det finns alternativ som “Fetch API” och “J. Query” etcetera men till detta projekt valdes Axios.js för att det är enkelt och behändigt att använda sig utav. Axios i frontend skickar och tar emot data från backend servern i detta projekt. (Sarjeant, 2020).

2.7 I18n

Då en webbapplikation designas bör det tas i beaktande att alla inte har likadana språkkunskaper. Det är bra att erbjuda alternativa språk vid skapandet av en webbsida. Till detta syfte har ett litet JavaScript bibliotek implementerats i webbportalen för att användare av portalen enkelt ska kunna ändra språk.

Biblioteket som tillämpats kallas Vuei18n.js och i projektets fall så hanteras tre olika språk. Finska, engelska och svenska. (kawaguchi, 2020).

2.8 Express.js

Express.js är ett JavaScript ramverk som kan användas för att enkelt skapa webb-applikationer och webb APIs. Express har tillgång till en stor mängd Http-anrop och används för att hantera alla förfrågningar som kommer till servern där Express körs.

Express.js används för att skapa rutter på node.js backend som kan anropas via Http anrop. Express svarar på korresponderande anrop med rätt kod om det finns en rutt med det namn och typ som anropats. Till exempel om det finns kodat en "**students**" rutt med express som har en GET funktion (GET anrop vill ha svar "en eller flera" av någon typ från en API) för att returnera "students", så går det att få tag i "students" via att anropa "students/GET". (The OpenJS Foundation, n.d.).

2.9 Sequelize

För att kunna spara all relevant data som en server hanterar så behövs en databas. Att kommunicera med databaser kan vara komplicerat, speciellt om det finns flera olika typer av databaser som fungerar på olika sätt. Det finns ett paket för Node.js som kallas Sequelize som gör det väldigt lätt att hantera olika databaser. Sequelize är en ORM (Object, Relational, Mapping) som kan användas för att översätta och hantera olika typer av data mellan databaser och andra program som till exempel projektets server.

Det går att skapa ett Sequelize element i kod, detta element vill veta vilken typ av databas som används samt databasens namn. Då Sequelize får dessa instruktioner så sköter Sequelize om kommunikationen till och från databasen. Sequelize kan således användas för att hantera och simplifiera kommunikationen med flera databaser av olika typ.

Med hjälp av Sequelize behövs inte kunskap om varje databas unika sätt att kommunicera med en server. Det behövs kunskap om hur anrop ska skrivas med SQL, men utöver det så sköter Sequelize om resten. (Sequelize Contributors, 2023).

2.10 SQLite

I projektet så finns en SQLite databas för hantering och lagring av portalens data. För att kommunicera med SQLite databasen från projektets server så används en ORM som kallas Sequelize. Med hjälp av Sequelize så går det väldigt enkelt att lägga upp en förbindelse direkt från projektets serverkod till projektets databas. (SQLite, 2023).

2.11 Multer.js

I detta projekt så används ett ramverk som kallas multer.js. Detta ramverk används för uppladdning och nerladdning av filer på projektets server. Multer hanterar filer som sänds till och från servern och i Multer koden behövs specifikationer om till exempel var filerna skall sparas på servern och hur en fil ska få sitt namn. (github.com/expressjs/multer, 2021).

3 Utförande

Efter att ha presenterat all relevant teori och teknik så följer nu själva utförandet av slutarbetet. I denna del kommer examensarbetets olika moment att beskrivas och förklaras kort och koncist så att man kan förstå hur examensarbetet är gjort.

3.1 Varför Node.js

Det var inte självklart vid projektets start att Node.js skulle bli basen för webbportalen. Alternativ till Node.js finns, ett bra alternativ skulle kunna vara Microsofts .Net ramverk som också är open-source och cross-platform. .NET går att koda i C# F# och Visual Basic. Troligen skulle det ha varit ett helt okej alternativ i stället för Node.

Slutligen valdes Node.js till projektet av ett flertal orsaker. Samtliga programmerare som deltog vid projektets start hade vanan inne att programmera i JavaScript. Utöver vanan så är Node.js ganska enkelt och flexibelt, dessutom så är Node.js väl lämpat för den typ av webbapplikation som eftertraktades. Således var det relativt enkelt att välja Node.js.

3.2 Hur portalen är strukturerad

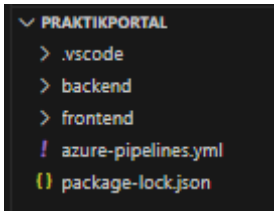
Utöver Node.js som bas för projektet så använder sig projektet utav ramverket Vue.js för frontend programmering och design av det grafiska användargränssnittet. För serverdelen av projektet så används Node.js ramverket “Express.js”. Express.js är användbart för att kunna skapa en flexibel API (Application Programming Interface) som kan sända och ta emot data via http anrop.

På klientsidan (frontend) så används Axios.js. Axios.js är ett frontend-bibliotek som kan användas för att hantera http anrop till en server, I detta fall till Express.js. Projektets server har åtkomst till en databas av typen SQLite. Servern kommunicerar med SQLite databasen med hjälp av biblioteket Sequelize som erbjuder enkel kommunikation till en rad olika databaser.

Praktikportalen är uppbyggd i två huvuddelar, frontend och backend. I Frontend skapas och hanteras alla grafiska element av portalen och i backend delen körs portalens server, databas och andra funktioner. (ComputerScience.org, 2023).

3.2.1 Frontend

Frontend mappen innehåller allt som har att göra med det grafiska användargränssnittet. Det finns en inbyggd Vue.js router som kallas "**Vue Router"** som kan navigerar mellan olika Vue.js filer i det grafiska användargränssnittet. Med Vue-routern kan man bestämma namn på portalens olika sidor och routern erbjuder en rad metoder för att navigera mellan de olika sidorna.



Figur 1: Praktikportalens layout (Visual Studio Code).

I frontend finns en del huvudscript som app.js och main.js vilka har som uppgift att starta portalens GUI. I frontend så finns i18n.js biblioteket som hanterar portalens språkalternativ.

I vue.config.js finns specificerat destinationen dit var koden kommer at kompileras då den omvandlas från utvecklingsläge till produktionsläge. I detta projekt så kommer koden att kompileras till en mapp public som finns i backend. Den grafiska koden testas i frontend men den anländer sist och slutligen till backend.

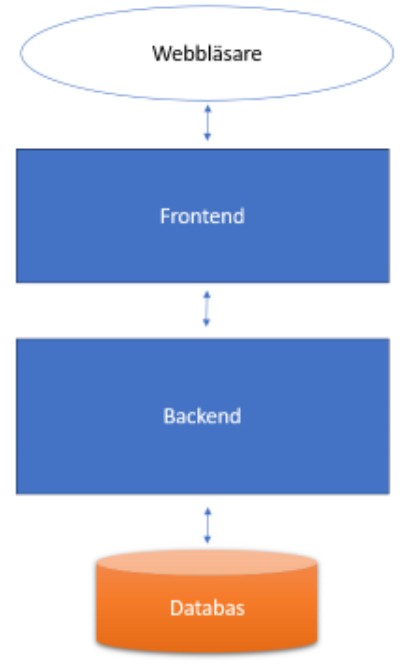
3.2.2 Backend

Backend delen innehåller själva applikationen samt server och databasen som portalen körs på. Backend tar hand om de grafiska komponenterna som sänds från frontend delen av projektet. Dessa filer finns i en mapp kallad public.

Backend servern kör Vue filerna och Vue routern och kallar på backends egna funktioner såsom express rutter och databashantering för att sända och ta emot data mellan de grafiska komponenterna och servern.

Orsaken till att det grafiska användargränssnittet testas och designas i frontend för att sedan överföras till backend är projektets filstorlek. Då koden överförs hamnar hela portalens kod i en mapp (backend) och den grafiska koden som sparas här är en minimerad version av den som finns i frontend. Denna kod tar upp mindre utrymme och kan användas direkt av Express servern.

Frontend kan beskrivas som en testmiljö där den grafiska delen av portalen testas och designas medan den grafiska koden som förs över till backend är mera en "färdig" kod.



Figur 2: Portalens struktur Illustrerad. En användare med en webbläsare hanterar det grafiska användargränssnittet (frontend) som kommunicerar med en server (backend). Backend servern hämtar data från en databas och data går från databasen tillbaka samma väg för att slutligen visas för användaren i webbläsaren.

3.3 Varför Vue.js

Vue.js valdes för att det är ett enkelt och användbart ramverk för att skapa dynamiska frontend miljöer. Vue.js är byggt på vanlig CSS, JavaScript och html, dessa metoder implementeras i en och samma fil. Det går med andra ord att skapa en single page application med bara en fil som innehåller all styling, html och script som en enda helhet.

Vue valdes för att det är enkelt att använda sig utav och har bra funktioner som till exempel testservermiljö och det är byggt på ett sätt som gör det lätta skapa stora projekt med avancerad routing.

Vue är enkelt att använda, installera och kommer med en egen CLI (Command Line Interface) som enkelt kan skapa ett botten (bas) för ett vue.js projekt. Vue CLI kommer med två väldigt användbara script som definieras i en fil i projekt mappen "package.json". Dessa script kallas "serve" och" build". (Vue.js, Last Updated: 11/9/2022).

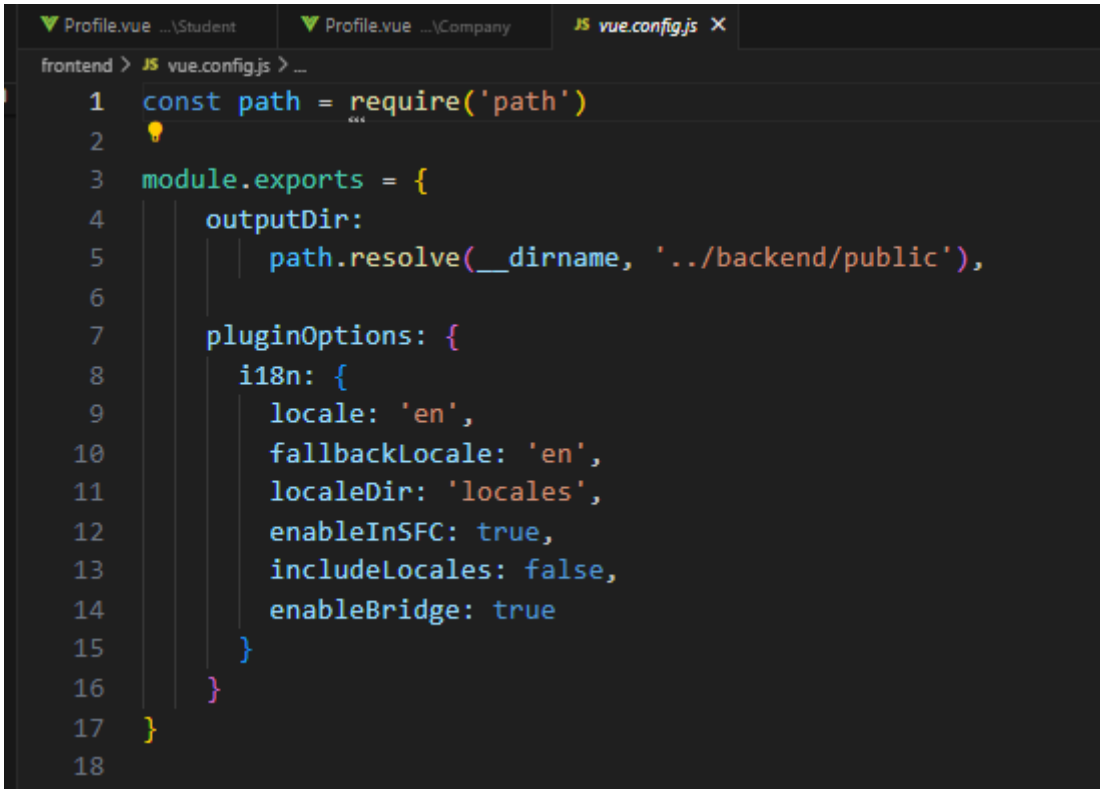
3.3.1 Serve

Serve initierar en utvecklingsserver som använder sig av **Hot-Module-Replacement** (HMR) som tillåter uppdatering och förändring av testserverns kod i realtid. Det går med andra ord att göra uppdateringar som uppdateras live på testservern utan behov av att kompilera och starta om servern på nytt.

3.3.2 Build

Detta kommando skapar en variant av koden som är i produktionsläge. Detta script minimerar all html, CSS och JavaScript och placerar all kod i en specificerad mapp. I detta projekt finns en fil "vue.config.js" där det specificeras bland annat plugins till Vue.js och outputDir (**Output directory**) som är destinationen för koden som skapas med scriptet "build".

Projektet är indelat i en backend där all kod sparas och en frontend där projektets GUI först testas och sedan skickas till backend. I detta projekt sparas all frontend kod till en mapp i backend med namnet public.



Figur 3: Konfigurering så att den grafiska koden ska sparas i destinationen "outputDir" som är mappen public *(backend/public)* i backend (JavaScript).

3.4 Express.js routing

I projektet används Express.js i backend (server delen av projektet) och Express.js kommunicerar med Axios.js i frontend GUI för att få en väldigt enkel http-länk mellan båda delarna av projektet. Express.js skapar rutter (routes) på servern och Axios.js gör anrop till dessa rutter.

```
var express = require('express');
```

Figur 4: Initierar express (JavaScript).

```
var app = express();
```

Figur 5: Deklarerar en ny variabel av typen express (JavaScript).

3.4.1 Routers

Express.js tar emot alla frontend filer som skall användas. Dessa filer finns i en mapp public i backend, vilket är destinationen för alla filer som finns i production mode. Här börjar frontend GUI att användas i dess minimerade format.

```
app.use(express.static(path.join(__dirname, 'public')));
```

Figur 6: Deklarerar att express ska använda de grafiska komponenterna från public mappen (JavaScript).

I app.js importeras och används alla URL routers som servern kommer använda sig utav. Till exempel i frontend koden så kallades rutten "/company/allmissions" med hjälp av Axios. Express.js i backend kommer att svara på detta "GET" anrop.

```
// Make a request to the server to retrieve the company profile data
const response = await axios.get('/company/allmissions/');
```

Figur 7: Ett "GET" anrop skickas från en vue.js fil med hjälp av Axios (JavaScript).

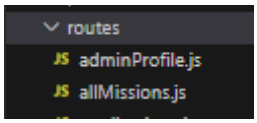
Anropet “company/allmissions” hanteras i filen app.js i backend. Först så åkallar programmet en variabel som kallas “allmissionsrouter”. **allmissionsrouter** importerar alla router funktioner från en fil **allMissions.js** som finns i en mapp **“routes”**.

```
app.use('/company/allmissions', allmissionsRouter)
```

Figur 8: Då company/allmissions anropas så använder koden routern allmissionsRouter (JavaScript).

```
const allmissionsRouter = require('./routes/allMissions')
```

Figur 9: allmissionsRouter använder sig utav en JavaScript-fil “allMissions.js” som finns i en mapp “routes” (JavaScript).



Figur 10: allMissions.js (Visual Studio Code).

3.4.2 allMissions.js

I allMissions.js deklareras användning av express.js egen router, den deklareras med **express.Router()**. I denna fil så kan expressroutern ta emot http-anrop. Http-anrop skickas från frontend via biblioteket Axios. Http anropen kan vara get (hämta), post (sänd), put (ändra), delete (radera). Dessa fyra kommandon brukar förkortas som **CRUD** (Create, read, update och delete).

I frontend koden så skickas anropet “**get(“/company/allmissions/”)**” tecknet **“/”** i slutet syftar på “**get(‘/’)”** om det skulle stå **get(“/company/allmissions/:id”)** så skulle det anropa **get(‘/:id’)**. För det mesta kodas **get(‘/’)** så att alla värden som finns tillgängliga i den databastabell som kommuniceras med kommer att sändas med som svar. Utifall ett “get” anrop skickar med någon typ av id så kommer det returnerade resultatet från databasen för det mesta att vara mera selektiv då koden försöker hitta rätt data med rätt id.

```
let express=require('express')
const router = express.Router()
let mission = require('../data/missions')
router.use(express.json())


router.get('/', async(req, res) => {
    try {
        console.log("got to get all missions");
        let missions = await mission.getAllMissions();
        res.json(missions);
    }
    catch (error) {
        console.error('Error retrieving company missions:', error);
        res.send(null);
    }
})

router.get('/:id', async(req, res) => {
    try {
        let missions = await mission.getMissions(req.params.id);
        res.json(missions);
    }
    catch (err) {
        res.send(null);
    }
})

module.exports=router
```

Figur 11: Express router för missions (JavaScript).

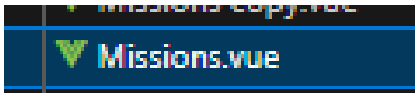
“router.get(‘/’)” funktionen i denna fil anropas och således körs dess kod. Här anropas en funktion “getALLMissions()” som är en extern funktion. getALLMissions() är en funktion som importeras från en annan JavaScript-fil vid namn missions.js som finns i en mapp **“data”**.

Det är i filen missions.js som SQL kod och Sequelize kommunicerar med projektets databas och därifrån kommer ett svar som skickas tillbaka till det grafiska användargränssnittet.

Det värde som funktionen getALLMissions() får tilldelas sedan till variabeln missions (uppdrag) som sedan skickas till frontend som svar i form av Json-data. JSON (JavaScript Object Notation) skickar värden i form av vanlig text.

3.5 Vue filers struktur

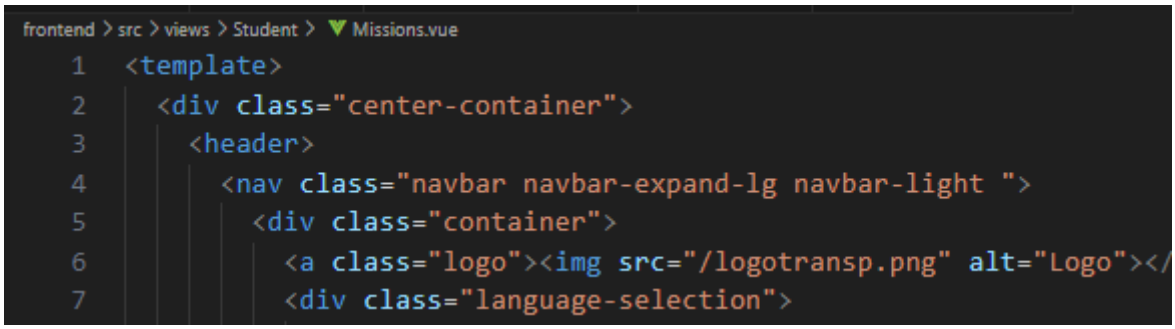
För att göra det enklare att förstå vad en Vue-fil är så kommer detta kapitel att förklara hur en Vue-fil är strukturerad. Som exempel kommer filen Missions.vue att användas. Missions är en sida på portalen som används för att lista olika uppdrag för studeranden som de kan läsa och sedan anmäla sig till om de så vill. Sidan Missions skapas med hjälp av filen Missions.vue.



Figur 12: En Vue.js fil (Visual Studio Code).

3.5.1 Template

Filen börjar med alla element som skulle motsvara klassisk html-kod, denna bit av koden finns innanför en tagg som kallas **template** (med uppbyggnaden: "**<template>** "kod" **</template>)"**. Innanför template så skrivs alla element som syns i det grafiska användargränssnittet (i webbläsaren) såsom text, tabeller sökfält, menyn och så vidare.



Figur 13: All html finns inuti template elementet som börjar med "<template>" och slutar med "</template>" (Vue.js).

3.5.2 Style

Efter template så följer en del som kallas "**style**" (med uppbyggnaden: "**<style>** "kod" **</style>"**). Style delen av koden kodas i CSS. Style delen är inte helt nödvändig utan det går att använda externa bibliotek eller interna taggar i **template** delen för att ändra stil på koden.

Utifall det handlar om många element som bör editeras på samma sätt så är style delen ett bra alternativ. I detta projekt så används ett externt bibliotek som heter Bootstrap.js utöver style delen för att manipulera stilen på portalen. Bootstrap tillåter en att editera stilen på html-objekt genom att ändra på "Class" värdet inne i ett html-element.

```
</template>

<style>
html, body {
  height: 100%;
  margin: 0;
  padding: 0;
}

.center-container {
  position: relative;
  min-height: 100%;
}

.background-image {
  background-image: url('/intrabg.jpg');
  background-repeat: no-repeat;
  background-size: cover;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
```

Figur 14: CSS styling I en Vue-fil (CSS).

3.5.3 Script

Sist i rad så kommer den del av Vue filen som innehåller all JavaScript-kod och alla script för Vue koden. Denna del kallas "**script"** (med uppbyggnaden: "**<script>** "Kod" **</script>**"). Här går det att länka in externa filer såsom JavaScript filer, CSS filer eller dylikt om det behövs.

Som exempel finns biblioteket Axios som används för att anropa servern i backend. För att Axios ska kunna fungera så måste Axios inkluderas i Vue filen. Portalen använder sig utav ett plugin I18n.js som tillåter en att ändra på språk. Detta script finns i en JavaScript-fil som också måste importeras för att språkhanteringen ska fungera.

```
<script>
import i18n from '../../i18n'
import axios from 'axios';
```

Figur 15: Script delen börjar och import av externa bibliotek och funktioner inträffar (JavaScript).

I html delen av koden går det att direkt hänvisa till variabler genom att deklarera dem i script delen av koden. I Missions.vue så finns det tillgängligt tre variabler med olika sorters data. Det finns en variabel **search**, en annan variabel **category** och en lista som kallas **missions**.

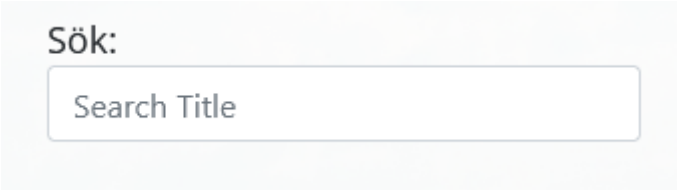
```
export default {
  data: function() {
    return{
      search:'',
      missions: [],
      category:'' ,

    }
  },
```

Figur 16: Deklarerar två variabler och en lista som kan användas inuti Vue <template> delen av koden (Vue.js).

Variabeln “search” implementeras innanför template (html) delen av koden i ett sökfält. “search” binds ihop med det värde som användaren skriver in i sökfältet. Det går med Vue kommandot “**v-model**” att binda ihop variabeln “search” med texten som användaren matar in i sökfältet.

```
<input type="text" class="form-control" id="sök" placeholder="Search Title" v-model="search">
```

Figur 17: Binder ett textfält i html koden (template) till search variabeln i script delen genom Vue metoden V-model (Vue.js).



Figur 18: Textfältet i en webbläsare.

“category” är en variabel som kommer att användas utav ett filter senare i denna fil. “category” får sitt värde från ett urval av fem knappar. Bara en av dessa knappar (kategorier) går att välja och värdet på category variabeln blir vald således.

Figur 19: Ett fält med flera valbara alternativ för variabeln category.

```
<b-card>
<div>
 <input class="form-check-input" checked type="radio" id="check1" name="option1" value="Examwork"  v-model="category">
 <label class="form-check-label">Examensarbete</label>
 <br>
 <input class="form-check-input" type="radio" id="check2" name="option1" value="Internship"  v-model="category">
 <label class="form-check-label">Praktik</label>
 <br>
 <input class="form-check-input" type="radio" id="check3" name="option1" value="Developmentwork" v-model="category">
 <label class="form-check-label">Utvecklingsarbete</label>
 <br>
 <input class="form-check-input" type="radio" id="check4" name="option1" value="Project"  v-model="category">
 <label class="form-check-label">Projekt</label>
 <br>
 <input class="form-check-input" type="radio" id="check5" name="option1" value="Other"  v-model="category">
 <label class="form-check-label">Övrigt</label>
</div>
```

Figur 20: Koden för fältet. Här går att se alla val samt länkar till variabeln "category" med Vue metoden "v-model" (Vue.js).

3.5.4 Kommunikation med backend

I filen Missions.vue finns en tabell som bör fyllas med data om olika uppdrag, till detta syfte så finns det en lista i koden som heter "**missions**". Till "missions" så kommer data om olika uppdrag att hämtas från projektets databas. För att mata in data till "missions" listan så körs en kod som anropas då sidan skapas, detta händer i Vue.js scripthändelsen "mounted".

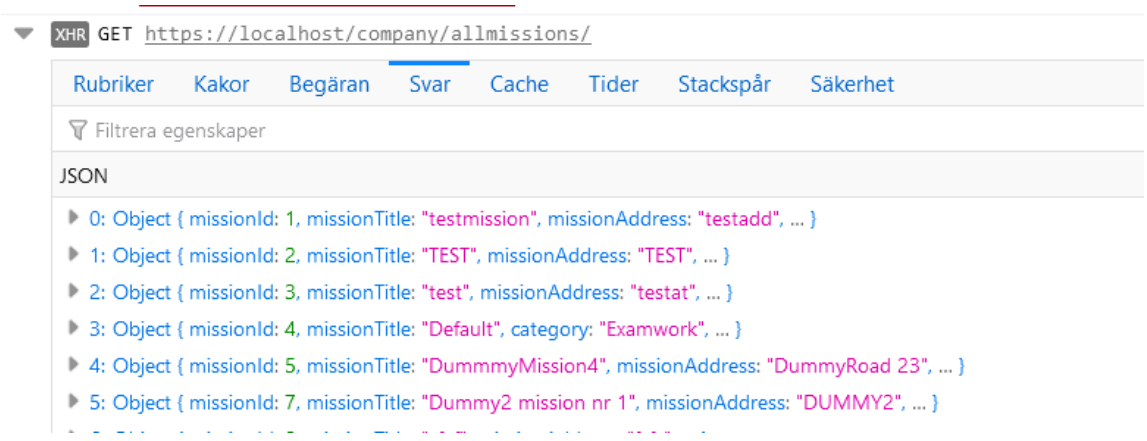
I mounted så körs ett axios "get" anrop till ". /company/allmissions/" som är en URL på servern. servern kommer att hantera alla rutter och anrop internt och svara med den information som efterfrågats. Http anropet "get" används i detta fall, således kommer servern att svara med den "get/" kod som finns på rutten "./company/allmissions/".

All data som skickas som svar i serverns http-respons tas emot i en variabel som heter "**response**". Efter att all data tagits emot så deklareras det att **"missions"** (listan) får värdet som finns i **"response.data"**.

```
async created() {

  console.log(this.category)

i18n.locale = localStorage.Language
},
    async mounted() {
     console.log("should have run: const response = await axios.get('/company/missions'); ");
  try {
    // Make a request to the server to retrieve the company profile data
    const response = await axios.get('/company/allmissions/');


    // Set the retrieved data to the `profiles` data property
    this.missions = response.data;
  } catch (error) {
    console.error('Error retrieving company profile:', error);
  }
},
```

Figur 21: Anropar backend servern via Axios för att få tag i uppdrag till listan missions (Vue.js).

Det går att verifierar i en webbläsare att efterfrågad data har tagits emot från "**localhost/company/allmissions"** genom att gå via webbläsaren och ta en titt på webbläsarens konsol. Där står information om alla http-anrop och svar som skickats och tagits emot.



Figur 22: Svaret på "GET" anropet som JSON format i en webbläsare (JSON).

Som det går att se så skickas det via webbläsaren ett "GET" anrop till rätt URL. Svaret som tas emot är i JSON format. Detta är den data som skickas in till **"missions"** listan i Vue koden.

3.5.5 Funktioner och data manipulering i Vue.js

Innan all data visas åt användaren i användargränssnittet så kommer relevant data att bearbetas av ett filter. Det är här som de tidigare nämnda variablerna "**search"** och **"category"** kommer i användning.

Det finns ett Vue-element i script delen som kallas "**computed**", i "computed" så går det att skriva direkt i filen funktioner som hela tiden kalkyleras baserat på värden på relevanta variabler.

Det finns en annan typ av Vue-element som heter "**methods**" där det går att skriva olika funktioner som går att anropa i template delen av koden om korrekta händelser inträffar så som till exempel att klicka på en knapp. Skillnaden mellan methods och computed är med andra ord att "**computed"** kontrolleras och körs konstant medan "**methods** "måste vänta på att en specifik händelse inträffas först.

I missions.vue på computed delen används ett filter som kallas "**filteredmissions**". Funktionen "filteredmissions" editerar listan **"missions"** och returnerar en modifierad version som filtreras beroende på variablerna **"search"** och "**category"**.

Som tidigare nämnts får **"search"** sitt värde från ett sökfält och **"category"** får sitt värde beroende på vilken kategori som blivit vald. Detta filter tillåter användaren att enkelt söka efter relevanta uppdrag på basis av användarens egna preferenser.

```
computed: {
    filteredmissions: function(){
      console.log(this.category);
  //return this.missions.filter(missions => missions.missionTitle.includes(this.search))
 // return this.missions.filter(mission => mission.missionTitle = 'testmission')// gör alla missions missiontitle till testmission
  //return this.missions.filter(mission => mission.missionTitle = this.search) //gör att alla titlar blir till search
 // return this.missions.filter(mission => mission.missionTitle.includes(this.search))
 return this.missions.filter((mission) => {
    return mission.missionTitle.toLowerCase().match(this.search.toLowerCase()) && mission.category.toLowerCase().match(this.category.toLowerCase());
    //return mission.category.toLowerCase().match(this.category.toLowerCase());
    //return user.name.toLowerCase().match(this.searchInput.toLowerCase());
 });

 }
},
```

Figur 23: Filterfunktion för att kunna söka specifika uppdrag (Vue.js).

Nu så följer den del av koden som slutligen visar listan med uppdrag på det grafiska användargränssnittet. Det finns en tabell som anropar "**filtererdmissions**" funktionen och går igenom den filtrerade listan med en Vue-metod som kallas **v-for**. Det som händer är följande: "För varje **mission** i **filteredmissions**, separerade beroende på **mission.id**" så kommer det att visas en rad med följande data: **Uppdragstitel**, **adress**, **anställare**, **start** och **slutdatum**, **uppdragsbeskrivning** och slutligen **kategori** på uppdraget.

Förutom uppdragets innehåll så finns även en knapp som länkar till en annan sida på portalen där det går att se fler detaljer om det specifika uppdraget.

```
<tbody>
  <tr v-for="mission in filteredmissions" :key="mission.id">
  <!--<tr v-for="mission in missions" :key="mission.id">-->

    <td>{{ mission.missionTitle }}</td>
    <td>{{ mission.missionAddress }}</td>
    <td>{{ mission.missionEmployer }}</td>
    <td>{{ mission.missionStartdate + " - " + mission.mission
    <td>{{ mission.missionDescription }}</td>
    <td>{{ mission.category }}</td>
    <td ><b-button> <router-link :to="{ name:'studMission', p


  </tr>
</tbody>
```

Figur 24: Visar data från den filtrerade listan som tabelldata (Vue.js).

Results

Title	Adress	Employer	Date	Description	category	Actions
testmission	testadd	testemp	26/5/2023 - 26/6/2023	test	Examwork	Open
TEST	TEST	TESTEMpl	26/5/2023 - 26/7/2055	124	Examwork	Open
test	testat	teatat	25/6/2045 - 24/7/2045	12515	Examwork	Open

Figur 25: Tabelldata med uppdrag och knapp i webbläsare.

3.6 I18n språkhantering

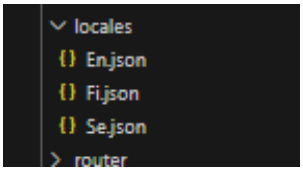
Portalens språkhanteringen ordnas på frontend delen av projektet utav ett Vue-plugin som heter Vue I18n. Scriptet har installerats som ett plugin till Vue.js.

```
import Vue from 'vue'
import VueI18n from 'vue-i18n'

Vue.use(VueI18n)

function loadLocaleMessages () {
  const locales = require.context('./locales', true, /[A-Za-z0-9-_,\s]+\.json$/i)
  const messages = {}
  locales.keys().forEach(key => {
    const matched = key.match(/([A-Za-z0-9-_]+)\./i)
    if (matched && matched.length > 1) {
      const locale = matched[1]
      messages[locale] = locales(key)
    }
  })
  return messages
}

export default new VueI18n({
  locale:  'Se', // changed from en to se
  fallbackLocale:  'Se', // changed from en to se
  messages: loadLocaleMessages()
})
```

Figur 26: Initierar språkanvändning (JavaScript).

Vue I18n fungerar genom att ändra språk beroende på värdet “locale”. Genom att ändra “locale” så kan man hantera olika språk genom ett antal olika Json filer, dessa är de “locales” (språk) som scriptet och portalen har access till. I bilden ovan definieras det att “fallbackLocale” (standard språket) är svenska.

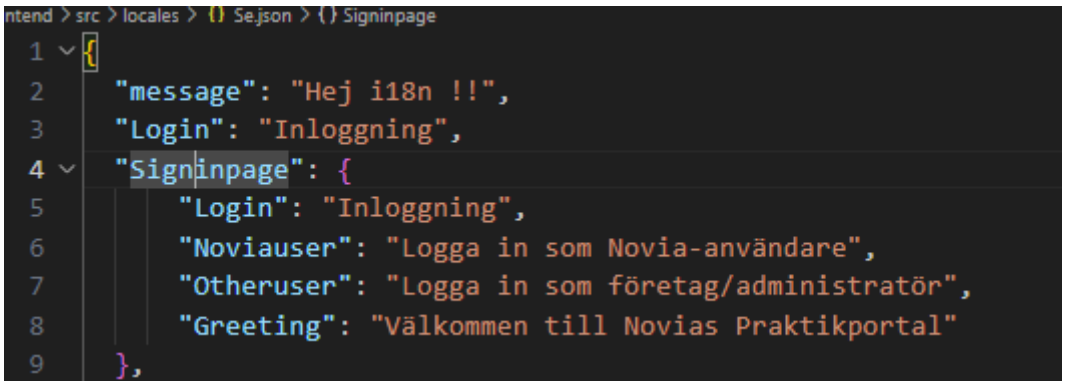
Json filerna som används är sparade i en mapp “locales” i frontend mappen av projektet, filerna är “**En.json**”, “**Fi.json**” och “**Se.json**”. Språken som är tillgängliga baseras på filnamnet innan extensionen “**. json**” (**En, Fi, Se**).



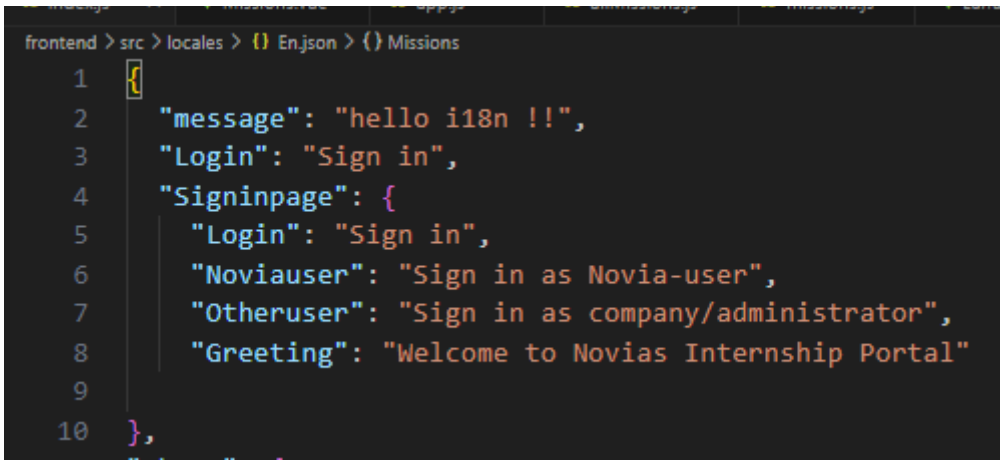
Figur 27: JSON filer för olika språkalternativ (Visual Studio Code).

Dessa filer är uppbyggda på så vis att JSON-objekt deklareras i filen och inuti dessa objekt existerar variabler som får text värden. I bilden nedan så finns till exempel objektet "Signinpage" som har variablerna "Login", "Noviauser", "Otheruser" och "Greeting".

Objekten och namnen på deras variabler är likadana i alla tre språkfiler, det enda som skiljer sig åt är värdet på variablerna. Värdena är olika på så vis att orden är på svenska i den svenska filen och på engelska i den engelska filen samt finska i den finska filen. Som exempel här nedan är "Signingpage" som innehåller text som syns på portalens login sida.



Figur 28: Signinpage på svenska (JSON).



Figur 29: Signinpage på engelska (JSON).

Dessa värden används i en Vue.js fil genom att scriptet Vue I18n först importeras i “script” delen av Vue filen. Sedan skrivs det i “template” delen av Vue koden med speciell syntax namnet på objektet (signinpage) och vilket av objektets värden som ska visas.

```
">{{$t('Signinpage.Greeting')}}</h1>
```

Figur 30: Använder “Greeting” alternativet från Signinpage (Vue.js).

Beroende på vilket värde mellan “Se”, “Fi” och “En” som är aktuell på sidan så kommer texten att uppdateras enligt de värden som hänvisas i Json filerna.

Figur 31: Svenska valt som språk.

Figur 32: Engelska valt som språk.

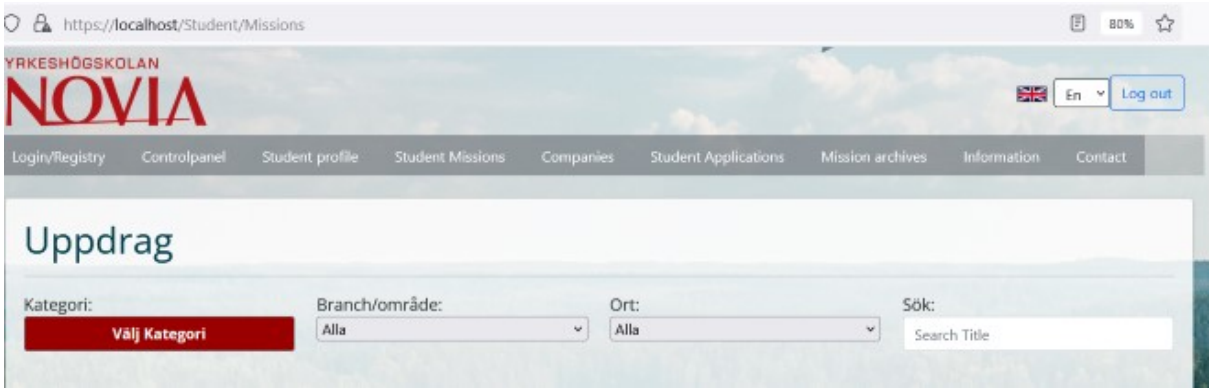
Figur 33: Svenska på webbsidan.

Figur 34: Engelska på webbsidan.

3.7 Vue router

Vue har en egen router som följer med vid installationen av Vue.js i ett projekt. Router paketet kallas "**Vue router**" och via denna router kan funktionen "**Router**" importeras till en JavaScript-fil.

I portalen så implementeras denna router i en fil som heter **index.js**. I denna fil deklareras alla interna rutter och URL länkar som går att använda inom projektet för att navigera användargränssnittet. Förra kapitlet gick i detalj igenom filen "**missions.vue**". Denna fil renderas åt användaren då användaren surfar till portalen och går till rutten "**Student/Missions"**.



Figur 35: Student/Missions i webbläsare.

```
{
  path: '/Student/Missions/',
  name: 'Missions',
  // route level code-splitting
  // this generates a separate chunk (about.[hash].js) for this route
  // which is lazy-loaded when the route is visited.
  component: () => import(/* webpackChunkName: "about" */ '../views/Student/Missions.vue')
},
```

Figur 36: Länk till "Student/Missions" i Vue routern (Vue.js).

I Vue routern deklareras en URL "**/Student/Missions/"** denna URL tilldelas filen "**Missions.vue**". "**Missions.vue**" får ett smeknamn "**Missions**" som går att använda i URL länkar i stället för att specificera filens hela URL.

I "**Missions.vue**" finns i tabellen för uppdrag en knapp som länkar till komponenten "**studMission**". studMission är smeknamnet till en annan Vue fil "**Student/Mission**".

```
<td ><b-button> <router-link :to="{ name:'studMission', params:{id:mission.missionId,title:mission.missionTitle}}"
```

Figur 37: Länk till Student/Mission i en Vue-fil (Vue.js).

I samma routerlänk till "**Student/Mission"** skickas även med två parametrar: "**id** och **title**". Dessa parametrar får sina värden från tabellen med uppdrag i **Missions.vue** och de används som referenser senare i "**Student/Mission**" i filen "**mission.vue**" för att få tag i rätt **id** och **titel** för det uppdrag som valdes i "**Missions.vue"**.

```
{
  path: '/Student/Mission/',
  name: 'studMission',
  // route level code-splitting
  // this generates a separate chunk (about.[hash].js) for this route
  // which is lazy-loaded when the route is visited.
  component: () => import(/* webpackChunkName: "about" */ '../views/Student/mission.vue')
},
```

Figur 38: Länk till Student/Mission och filen mission.vue (Vue.js).

3.8 Sequelize databasautentisering

Sequelize används i projektet för att kommunicera med webbportalens databas. Här specificeras det i kod att sequelize ska köras mot en databas som finns i backend i en mapp "**data**". Databasfilen kallas "**portal.db"** och den är av typen **sqlite**, vilket är databasdialekten som specifieras för sequelize.

```
let db_name = path.join(__dirname, './data/portal.db')
const {Sequelize, DataTypes, QueryTypes} = require('sequelize')
const sequelize = new Sequelize({
    dialect: 'sqlite',
    storage: db_name
})
```

Figur 39: Startar ett Sequelize element och länkar det till databasen "portal.db" av typen sqlite (JavaScript).

Som exempel av användning utav Sequelize så följer nu autentisering för att få uppdrag till "**Missions.Vue**" Detta inträffar i en fil som heter "**missions.js**".

Det första som händer då användning utav sequelize elementet deklareras i koden är att Sequelize autentiserar mot "**portal.db**" (sqlite databasen) så att en fungerande förbindelse uppstår. Sedan körs ett **Query** (förfrågan) via Sequelize till databasen av typen **select**.

SQL anropet som anropas är: "**SELECT * from missions"**. "**missions**" är namnet på en tabell i databasen och **SELECT *** syftar på att all data som finns i tabellen ska returneras.

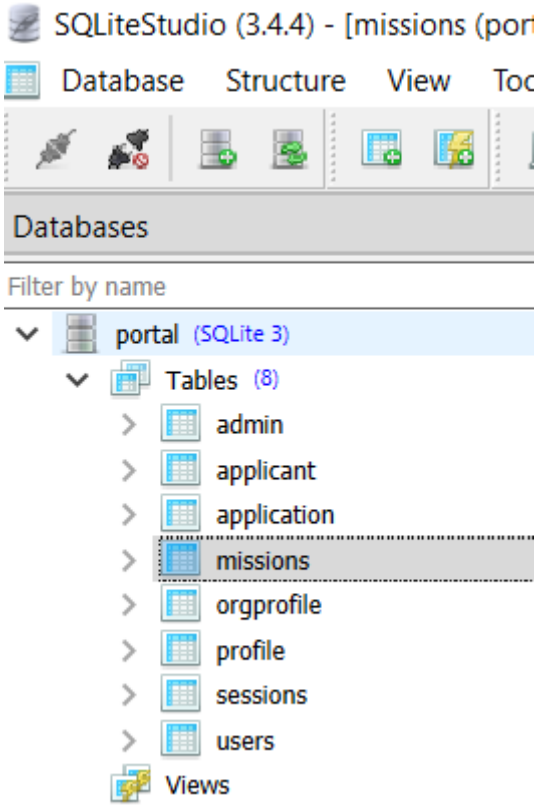
```
exports.getAllMissions = async () => {
    try {
        console.log("got to getAllMissions = async ()")
     await sequelize.authenticate()
        try {
            let missions = await sequelize.query("SELECT * from missions", { type: QueryTypes.SELECT })
            return missions
        }
        catch(error) {
            console.error('Error 1', error)
            return null
        }
     }
     catch (error) {
        console.error('Error 2', error)
        return null

    }
  }
```

Figur 40: Funktion där Sequelize anropar databasen med ett QUERY av typen SELECT (JavaScript).

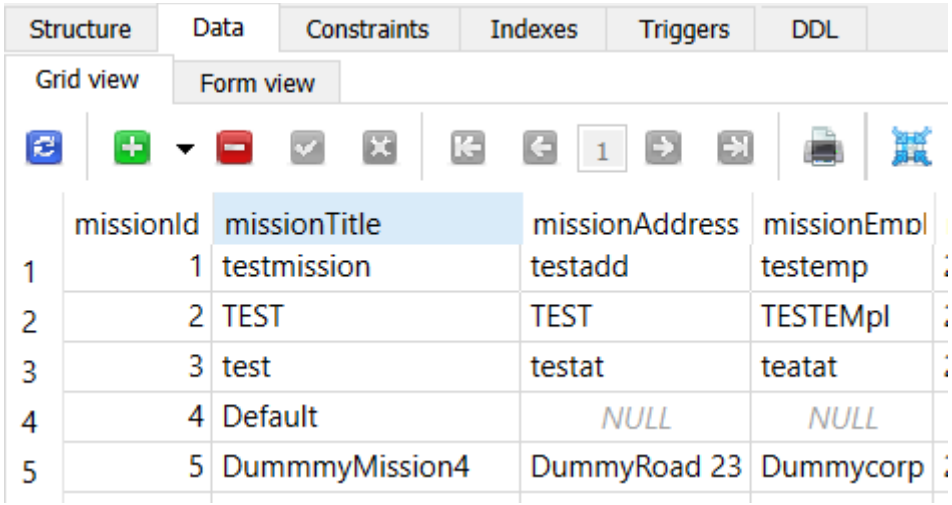
3.9 Databashantering med SQLite

Portalen använder sig utav en databas av typen **sqlite**. SQLite har ett eget verktyg som heter **SQLiteStudio** där det går att manipulera **sqlite** databaser och läsa värden på databastabeller. Här nedan finns databasfilen "**portal.db**" öppnad med "**SQLiteStudio**". Här går det att se databastabellerna som är tillgängliga, den tabell som är av intresse kallas "**missions**".

Tabellen missions innehåller alla uppdrag som finns tillgängliga för projektet och det är härifrån "**Missions.vue**" hämtar sina uppdrag. Det går att se i tabellen "**missions**" samma uppdragsdata som finns att se i tidigare bilder av dokumentet.



Figur 41: Databastabellerna i portal.db (SQLiteStudio).



Figur 42: Databastabellen "missions" öppnad i SQLiteStudio.

3.10 Filhantering med Multer

I detta projekt användes ett Node.js middleware (mellanprogram) som kallas **Multer** för att hantera upp och nerladdning av filer**.** Multer används utav Express.js på backend servern, för att Multer ska kunna fungera så behöver det konfigureras i koden information om hur filerna ska bli mottagna och lagrade.

Express servern sparar studerandens Cv filer till en mapp i backend som heter **"Cv"**. Denna mapp lagrar filerna direkt på samma diskutrymme som resten av projektet. För att undvika att filerna ska kunna få samma namn så sparas filerna med ett unikt numeriskt namn som skapas baserat på filens ursprungliga namn samt på tidpunkten då filen sparas. Då en fil sparas så sparas filens originalnamn i sqlite databasen tillsammans med en länk till var filen sparats på servern (mappen Cv).

```
const multer = require('multer')
```

Figur 43: Inkluderar Multer i en fil på backend (JavaScript).

```
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'Cv')
    },
    filename: (req, file, cb) => {
        console.log(file)
        cb(null, Date.now() + path.extname(file.originalname))
        console.log(cb.filename)
    }

})

const upload = multer({storage:storage})
```

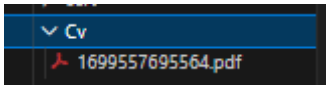
Figur 44: Sparar filer i en mapp "Cv" på servern och skapar unika namn baserat på tidpunkt då filen sparats (JavaScript).

```
},
      async UploadCV() {
    try {
      const Formdata = new FormData()
      Formdata.append("TheFile", this.file)
      console.log(Formdata)
      console.log(this.file)
       await axios.post('/student/CV',Formdata)
       alert('File uploaded');
    }
    catch(error) {
      console.error(error);
    }
```

Figur 45: I en Vue.js fil sänds till backend en fil med alias: "Thefile" via metoden “post” (JavaScript).

```
router.post('/', upload.single("TheFile"), async(req, res) => {
```

Figur 46: Vid en “post” request i backend på rutten ”/student/CV” sparas en fil och filen tas från variabeln "TheFile" som kommer från Vue.js i frontend (JavaScript).



Figur 47: Filen sparas i mappen Cv. Filen får ett unikt namn bestående av siffror (Visual Studio Code).

cv	t
DummyCV.pdf	"

Figur 48: Filens ursprungliga namn sparad i databasen (SQLiteStudio).

Filepath
1699557695564.pdf

Figur49: Länk till filen sparad i databasen (SQLiteStudio).

4 Resultat och diskussion

Projektet har framskridit och börjar sakta men säkert närma sig ett stadium där applikationen börjar bli användbar. Applikationen är dock en bra bit ifrån färdig. Projektet startades med ganska bra takt på hösten 2021 och höll på personligen fram till slutet av vårterminen 2022. Då uppstod personliga åtaganden då jag hamnade att utföra min värnplikt. Jag återkom till studierna till hösten 2023 och fortsatte då med projektet och började ta mera ansvar över server delen och databasen än jag tidigare haft innan jag ryckte in.

Projektet framskred ganska bra takt efter att jag återvänt till projektet. De flesta funktioner som var eftertraktade till portalen blev fullgjorda och i slutändan av år 2023 så hade vi en produkt som faktiskt börjar bli användbar även om den är i ganska enkelt format. Det ursprungliga uppdraget är inte helt genomfört då många nödvändiga detaljer ännu saknas men portalen har ändå kommit en bra bit på vägen.

Slutligen egen åsikt om projektet. Det har varit intressant och utmanande att arbeta i grupp med två andra personer med helt andra åsikter och synpunkter än mig själv på ett långvarigt projekt som detta. Att samarbeta med övriga parter har också krävt noga planering då vi har haft olika arbetstider, scheman, kurser med mera.

Det har också varit intressant att hålla gruppmöten mellan studeranden, arbetsgivare och lärare för att hålla koll på portalens framfart. Personligen tycker jag att projektet i sin helhet har varit en bra och lärorik upplevelse att ha deltagit i.

Även om projektet inte har blivit klart under den tid jag arbetat på det så är jag personligen mycket nöjd med min egen insats.

5 Källförteckning

ComputerScience.org. (2023). *computerscience.org/bootcamps/resources/frontend-vs-backend/*. Hämtat från computerscience.org: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/

github.com/expressjs/multer. (2021). *Npmjs.com/package/multer*. Hämtat från Npmjs.com: https://www.npmjs.com/package/multer

IT-Högskolan. (den 28 04 2023). *iths.se/vad-ar-javascript/*. Hämtat från www.iths.se: https://www.iths.se/vad-ar-javascript/

json.org. (u.d.). *json.org/json-en.html.* Hämtat från https://www.json.org: https://www.json.org/json-en.html

kawaguchi, k. (2020). *kazupon.github.io/vue-i18n/.* Hämtat från kazupon.github.io: https://kazupon.github.io/vue-i18n/

Kumpulainen, T. (02 2021). *theseus.fi/bitstream/handle/10024/490108/Opinnaytetyo_Kumpulainen_Tomi.pdf?sequence=2&isAllowed=y.* Hämtat från Theseus.fi: https://www.theseus.fi/bitstream/handle/10024/490108/Opinnaytetyo_Kumpulainen_Tomi.pdf?sequence=2&isAllowed=y

Sarjeant, J. J. (2020). *axios-http.com/docs/intro*. Hämtat från axios-http.com: https://axios-http.com/docs/intro

Sequelize Contributors. (2023). *Sequelize.org.* Hämtat från sequelize.org: https://sequelize.org/

SQLite. (den 5 12 2023). *sqlite.org/index.html.* Hämtat från www.sqlite.org/index.html: https://www.sqlite.org/index.html

The OpenJS Foundation. (u.d.). *Express.js*. Hämtat från Express.Js: https://expressjs.com/

The OpenJS Foundation. (n.d.). *Introduction to Node.Js*. From Introduction to Node.Js: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Vu, A. (den 05 01 2021). *theseus.fi/bitstream/handle/10024/429165/Anh_Vu.pdf?sequence=2&isAllowed=y.* Hämtat från https://www.theseus.fi: https://www.theseus.fi/bitstream/handle/10024/429165/Anh_Vu.pdf?sequence=2&isAllowed=y

Vue.js. (Last Updated: 11/9/2022). *vuejs.org/guide/cli-service.html*. Hämtat från vuejs.org: https://cli.vuejs.org/guide/cli-service.html

Vue.js. (u.d.). *vuejs.org/guide/introduction.html*. Hämtat från vuejs.org: https://vuejs.org/guide/introduction.html