LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Potilastietojärjestelmän kalenterikomponentin jatkokehitys Angularilla

LAHDEN AMMATTIKORKEAKOULU
Insinööri (AMK)
Tieto- ja viestintätekniikka
Syksy 2019
Aleksi Rehn

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Rehn, Aleksi	Opinnäytetyö, AMK	Syksy 2019
	Sivumäärä	
	34	

Työn nimi

Potilastietojärjestelmän kalenterikomponentin kehitys Angularilla

Tutkinto

Tieto- ja viestintätekniikka, ohjelmistotekniikka, insinööri (AMK)

Tiivistelmä

Opinnäytetyössä tavoitteena oli jatkaa jo valmiiksi aloitetun potilastietojärjestelmän kalenterikomponentin kehitystä käyttämällä Angularia. Opinnäytetyön toimeksiantajayritys oli teknologiakehitysyritys Nortal Oy.

Työssä tutkittiin Angularin toiminnallisuutta, hakemistorakennetta sekä komentorivikehotteen toimintaperiaatteita. Teoriaosuudessa käytetään esimerkkisovellusta helpottamaan toimintaperiaatteiden ymmärtämistä.

Käytännönosuudessa ensin tutkittiin ja havainnollistettiin toimeksiantoyrityksen kehitysympäristön ohjelmistokirjastoja. Niistä tärkeimpänä esitellään NgRx eli ”Reactive State for Angular”.

Lopulta tehtiin jatkokehitystä potilastietojärjestelmän kalenterikomponenttiin. Käytäntö koostuu kehitystehtävien vaatimuksien esittelystä, vaiheittaisesta ratkaisusta ja yhteenvedosta. Jatkokehitys tehtiin pääasiallisesti käyttämällä TypeScript-ohjelmointikieltä.

Asiasanat

Angular, komponentti, potilastietojärjestelmä

Abstract

Author(s)	Type of publication	Published
Rehn, Aleksi	Bachelor's thesis	Autumn 2019
	Number of pages	
	34	

Title of publication

Development of a calendar component for a patient information system using Angular

Name of Degree

Bachelor of Software Engineering

Abstract

The goal of this thesis was to develop features for a calendar component of a patient information system using Angular. The thesis was made for a technology development company named Nortal Oy.

The thesis takes a detailed look into Angular. It explores the fundamental ways that Angular works, how the file structure works and how the Angular command-line interface is operated.

The thesis explores the different software libraries used in the development environment for the patient information system. Special attention is given to NgRx, also known as "Reactive State for Angular".

The practical section of the thesis consists of several parts. The first part presents the features to be developed, provided by the company. Then there is a description of how the development was done and lastly an overview of how it went. The majority of the development was done using the TypeScript programming language.

Keywords

Angular, component, patient information system

SISÄLLYS

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimi teknologiakehitysyritys Nortal. Nortal on ottanut projektiksi luoda uuden ja paremman potilastietojärjestelmän yksityiselle terveydenhuoltosektorille, sillä osa yksityisen terveydenhuoltosektorin yrityksistä käyttää vanhentuneita potilastietojärjestelmiä. Nortalin tavoitteena on toteuttaa potilastietojärjestelmä, joka korvaa vanhat järjestelmät sekä auttaa helpottamaan tulevaisuuden terveydenhoitoprosessia.

Opinnäytetyön tavoitteena on jatkokehittää toimeksiantoyritys Nortalin potilastietojärjestelmäprojektia Angularilla. Jatkokehitys tehdään potilastietojärjestelmän kalenterikomponenttiin.

Toimeksiantoyritys Nortal on virolainen teknologiakehitysyritys, joka on perustettu vuonna 1998. Nortalilla on yli 800 työntekijää maailmanlaajuisesti, joista Suomessa on 166 (Nortal 2019). Vuonna 2018 Nortalin Suomen liikevaihto oli 24,6 miljoonaa euroa (Asiakastieto 2019).

Aluksi työssä tullaan kuvailemaan Angularia ohjelmistokehyksenä. Työssä kerrotaan tarkemmin Angularin toimintaperiaatteista, mitä etuja Angular tarjoaa sekä millaisia käyttöperiaatteita Angularin komentorivikehotteeseen sisältyy. Toimintaperiaatteiden esittelyssä käytetään esimerkki Angular-sovellusta. Sovellus on yksinkertainen muistiinpanotyökalu, jonka koodia esittelemällä tavoitetaan helpottaa toimintaperiaatteiden ymmärtämistä, sillä samat periaatteet pätevät kaikkiin Angular-sovelluksiin.

Angularin havainnollistamisen jälkeen esitellään potilastietojärjestelmän ympäristöä käymällä läpi ohjelmistokirjastoja, joita projektissa on käytössä. Niistä tärkeimpänä esitellään NgRx-kirjasto eli ”Reactive State for Angular”.

Jatkokehitysvaiheessa esitellään ja toteutetaan kaksi toimeksiantoyrityksen tarjoamaa työtehtävää. Työtehtävistä esitellään vaatimukset, toteuttamistapa vaiheittain sekä yhteenveto.

2 NORTAL

Nortal on Virosta lähtöisin oleva teknologiakehitysyritys. Nortalilla on samanaikaisesti monen asiakkaan projektit kehityksessä. Nortalin asiakkaita suomessa on muun muassa Neste. (Nortal 2019.)

Nortalilla on visio, jossa yhteyskunta toimii saumattomasti kolmen pääpilarin päällä. Nämä kolme pilaria ovat Yritykset, e-Terveydenhuolto ja e-Valtiotoiminta. Tämän opinnäytetyön kohteena oleva potilastietojärjestelmä on osa e-Terveydenhuolto pilaria. (Nortal 2019.)

Uuteen potilastietojärjestelmään liittyviä isoja eroja perinteisestä järjestelmästä on palveluiden sulavuus, datan saatavuus sekä käyttäjädatan hallussapito (Nortal eHealth 2019). Datan saatavuus on iso osa järjestelmää. Nortal pyrkii luomaan järjestelmän, jossa terveystiedot ovat aina saatavilla alustasta riippumatta. Järjestelmässä oleva data on aina ammattilaisen kirjaamaa ja hyväksymää tietoa. (Nortal eHealth 2019.)

Keskustellessani yrityksen edustajan kanssa tuli ilmi myös tulevaisuuden suunnitelmia potilastietojärjestelmälle. Esimerkiksi Nortal pyrkii luomaan järjestelmää, jossa käyttäjästä voidaan saada reaaliaikaista tietoa järjestelmään ja sitä analysoimalla voidaan ennalta ehkäistä mahdollisia sairauksia.

3 ANGULAR

3.1 Yleisesti Angularista

Angular on TypeScriptiin pohjautuva ohjelmistokehys, joka on suunniteltu yksisivuisten verkkosivujen toteuttamiseen. Angular on Googlen tuote, ja se on julkaistu vuonna 2015. Angular on uudelleen rakennettu versio AngularJS-ohjelmistokehyksestä. Googlen päätös uudelleen rakentaa AngularJS johtui muiden ohjelmistokehysten edistyksistä web-kehitys alalla. Jotta AngularJS pysyisi ajankohtaisena kehitysalustana, joutui Google aloittamaan kehityksen täysin alusta, minkä tuloksena syntyi Angular. (Medium 2018.)

Angularin käyttäminen projektissa on järkevää, jos projekti on tarpeeksi suuri. Angularin suurin vahvuus on sen järjestelmällisyys. Angularia käytettäessä kaikki komponentit ja osat sovellusta ovat omissa ympäristöissään. Tämä helpottaa suurien projektien ylläpitoa (Sitepoint 2018.)

Angular tarjoaa suuren määrän työkaluja kehittäjille. Angular tukee direktiivien käyttöä, jota käyttämällä on mahdollista antaa html-elementeille omia toiminnallisuuksia. Mukana tulee myös mahdollisuuksia tehdä muun muassa http-kutsuja, käyttää FormControllia ja hyödyntää tyyppimääritelmiä. Angular-projekteihin on myös sisään rakennettavissa sekä ”unit”- että ”end-to-end”-testaus. Angularin laajuuden takia moni kehittäjä kutsuu Angularia kehitysalustaksi, vaikkei se sitä varsinaisesti ole (Sitepoint 2018.)

Suuri osa kolmannen osapuolen web-kehityskirjastoista on myös saatavilla Angularille. Monesta kirjastosta on tehty versio, joka tukee Angularia.

Tällä hetkellä web-kehityksessä on tärkeää, että uudet ohjelmistot tukevat myös mobiili-laitteita. Angularin vahvuus on, että se toimii kaikilla laitetyypeillä mukaan lukien mobiili-laitteet.

3.2 Angular-CLI:n käyttäminen

Angular-CLI toimii käyttäen Node.js-ohjelmistoa, mikä mahdollistaa JavaScriptin paikallisen suorittamisen. Node Package Manageria (NPM) käytetään Angular-CLI:n lataamiseen (Angular 2019.)

Angular-CLI:n lataaminen NPM-palvelusta onnistuu käyttäen kuvion 1 komentoa komentokehotteessa. NPM toimii kaikissa komentokehotteissa.

```
npm install -g @angular/cli
```

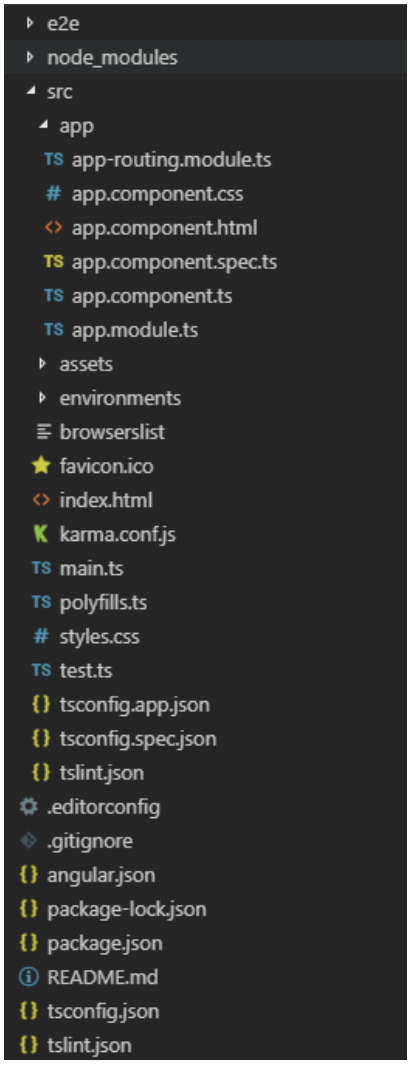
Kuvio 1. Angular-CLI asennus

Uuden Angular-työtilan ja ohjelmistoluurangon luominen tapahtuu kuvion 2 komennolla. Angular-CLI tunnistaa komennon uuden cli-komennon ”ng”-kirjaimista.

```
ng new my-app
```

Kuvio 2. Uuden projektin luonti

Komennon suorittamisen jälkeen NPM suorittaa projektin luontisovelluksen. Sovellus lataa kaikki tarvittavat Node Moduulit NPM-palvelusta sekä luo projektipohjaksi kuvion 3 perus tiedostorakenteen.



Kuvio 3. Angular tiedostorakenne

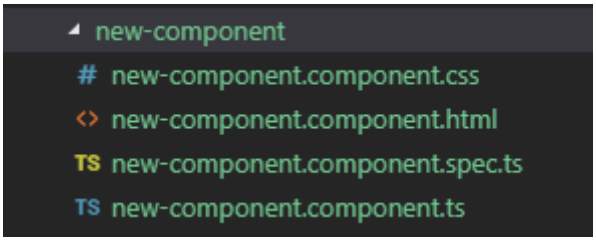
Angular käyttää komponentteja. Komponentit ovat se osa Angularia, joka näytetään verkkosivulla ja jonka sisältöä voidaan muokata dynaamisesti.

Komponentteja luodaan käyttämällä kuvion 4 komentoa komentokehotteessa.

```
ng generate component new-component
```

Kuvio 4. Uuden komponentin luonti

Komennon tuloksena Angular-CLI luo tiedostorakenteeseen uuden kansion nimeltä ”new-component”. Komponentti sisältää sille tarkoitetut .html-, .css-, .ts- sekä spec.ts-tiedostot. (Kuvio 5.)



Kuvio 5. Komponentin osat

Luodut tiedostot sisältävät tarvittavat koodit, jotta tiedostoja voidaan käyttää projektissa. Luontikomento lisää myös tiedostoista viitteen app.module.ts-tiedostoon (kuvio 3), jotta Angular osaa käyttää niitä.

Pääasiallisesti Angularin ominaisuuksien luonti tapahtuu vastaavalla tavalla käyttämällä komentokehotetta. Angular pitää huolen siitä, että tarvittaessa luodut tiedostot lisätään oikeisiin paikkoihin sekä, että niistä lisätään tieto tarvittaviin tiedostoihin.

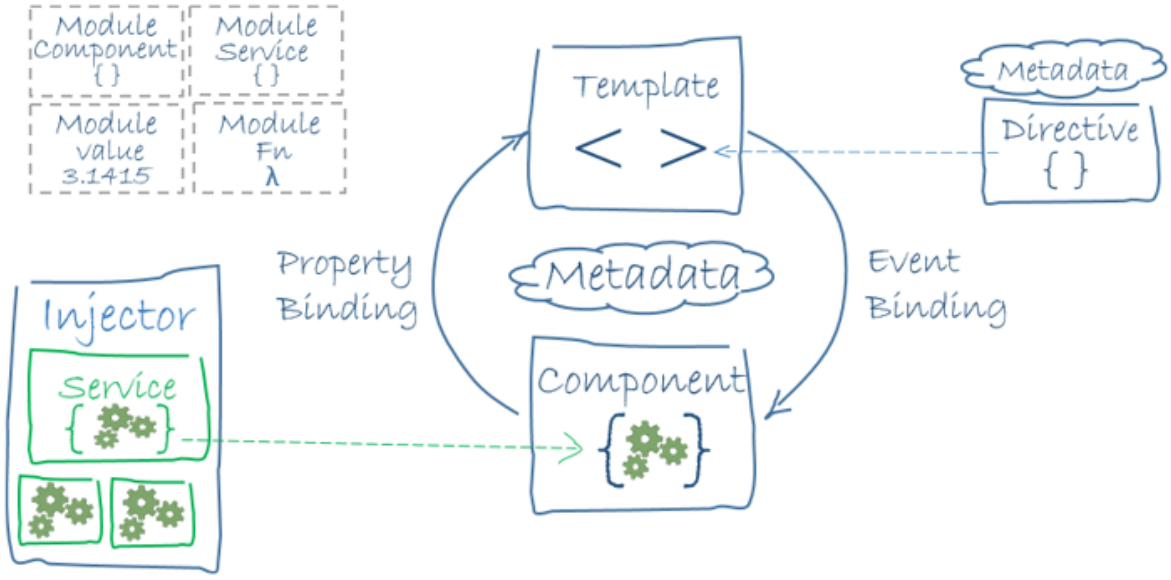
4 ANGULARIN TOIMINTAPERIAATTEET

Angularin toimintaperiaatteiden ymmärtämisen helpottamiseksi työssä käytetään esimerkkisovellusta. Sovellus on yksinkertainen To-do-muistilistasovellus, joka sisältää kaikki esiteltävät käsitteet. Sovellus on tehty Angularia käyttäen ja on vapaasti saatavilla. (Github 2019.)

Angular toimii Moduuleita käyttäen. Moduuleiden tehtävä on toimittaa sisältöä komponenteihin. Komponenteilla rakennetaan käyttöliittymä, jonka käyttäjä näkee. Jokaisessa Angular-sovelluksessa on AppModule-niminen root-moduuli, jonka tehtävä on suorituksen yhteydessä kasata määritetyt moduulit yhteen ja koota niistä toimiva kokonaisuus.

Moduulit ovat jatkokehitetty versio JavaScriptin omista moduuleista. Angular-moduulit pystyvät JavaScript-moduulien tapaan tuomaan toiminnallisuuksia toisista moduuleista sekä viemään omia toiminnallisuuksiaan muihin moduuleihin. Esimerkiksi Router-palvelun käyttäminen onnistuu Angularissa tuomalla Router-NgModuulin toiminnallisuus työympäristöön.

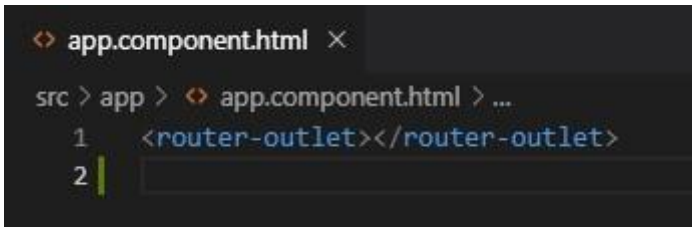
Moduulien käyttäminen vähentää koodin uudelleenkirjoittamista, koska samaa koodia voidaan käyttää useassa moduulissa. Modulaarisesta arkkitehtuurista johtuen voidaan myös hyödyntää ”Lazy-Loading”-tekniikkaa. Ideana on ladata ainoastaan ne moduulit päätelaitteeseen, joita tarvitaan perusnäkymän toimimaan saamiseksi. Tarvittaessa voidaan ladata taustalla lisää moduuleja sen mukaan, mitä käyttäjä tarvitsee. Kyseistä tekniikkaa käyttäen voidaan vähentää sivujen ensilatausaikaa, koska ensilatauksella ladataan vain tarvittavat komponentit.



Kuvio 6. Angularin toimintaperiaatekaavio (Angular 2019)

Kuviossa 6 esitetään yksinkertaistettu versio Angularin arkkitehtuurista. Kuvio esittää Angular komponenttia, johon kuuluu ohjelmalogiikka, sapluuna, injektori sekä niiden välisiä yhteyksiä. Angularin toimintaidea on, että käyttäjän selain lataa sivuston, joka lataa sapluunan (Template). Sapluunat hakevat tarvittavat datat komponentista (Component), johon on tarvittaessa injektoitu dataa käyttäen palvelua (Service). Käyttäjien toimenpiteet Sapluunassa käsitellään komponentissa, jossa suoritetaan tarpeelliset muutokset sapluunaan.

Komponentit ovat osa Angularia, mitä käytetään sisällön näyttämiseen sovelluksessa (Angular Component 2019). Komponentteja voidaan muokata dynaamisesti ohjelmalogiikalla tai niitä voidaan piilottaa ja näyttää tarpeen mukaan.



Kuvio 7. Root-komponentti

Kuviossa 7 on esimerkki Angular-sovelluksen root-komponentin .html-tiedosto. Suorituksen yhteydessä käyttäjälle ladataan tämä näkymä, mihin lisätään tarvittavia komponentteja käyttämällä Router-palvelua. Esimerkiksi Router-palvelulla voidaan ladata esimerkkisovelluksen To-Do-näkymä <router-outlet> tägin sisään.

Reititys on tekniikka, joka Angular käyttää vain tarvittavien komponenttien näyttöön. Normaalissa verkkosivustossa sisältöä vaihdetaan menemällä uuteen URL-osoitteeseen selaimessa. Esimerkiksi kirjautumissivu voidaan avata menemällä www.esimerkkisivusto.fi/kirjautuminen. Selain avaa kokonaan uuden näkymän käyttäjälle ja lataa kaiken uudelleen.

Angularin reititys tarkkailee osoiteriviä ja vaihtaa Kuvion 7 <router-outlet> tägin sisällön vastaamaan käyttäjän pyytämää sisältöä. Esimerkiksi, jos käyttäjä kirjoittaa toisen komponentin osoitteen osoiteriviin, niin Angular vaihtaa <router-outlet> tägin sisällön vastaamaan käyttäjän pyytämää komponenttia, ilman että koko näkymää ladataan uudelleen. Angular sovelluksen reitit määritetään router-tiedostoon, jotta reititin osaa määrittää oikean näkymän selaimeen.

```
const appRoutes: Routes = [
  { path: '', component: TodoComponent}
]
```

Kuvio 8. Reitit

Kuviossa 8 on esimerkki reitistä. Reitti ohjaa käyttäjän suoraan ”TodoComponent”-sisältöön, jos selaimen URL-osoitteeseen ei lisätä mitään.

Komponenttien sisältö luodaan käyttämällä sapluunoita. Sapluunat ovat .html tiedostoja. Sapluunan sisältö voidaan määrittää ennen kuin se näytetään käyttäjälle. Sisällön määritys suoritetaan käyttämällä Property Binding menetelmää. Property binding menetelmällä voidaan etsiä HTML tekstistä ennalta määrättyjä tunnisteista, joiden paikalle, tai sisään laitetaan ohjelmalogiikasta tulevaa sisältöä. (Angular Template 2019.)

Ennen kun näkymä voidaan esittää käyttäjälle, Angular selvittää ja toteuttaa kaikki tarvittavat sisältömääritykset. Angular suorittaa ohjelmalogiikan ja lisää määritettyihin HTML ominaisuuksiin sisältötietoa käyttäjän tarpeiden mukaan. (Angular Template 2019.)

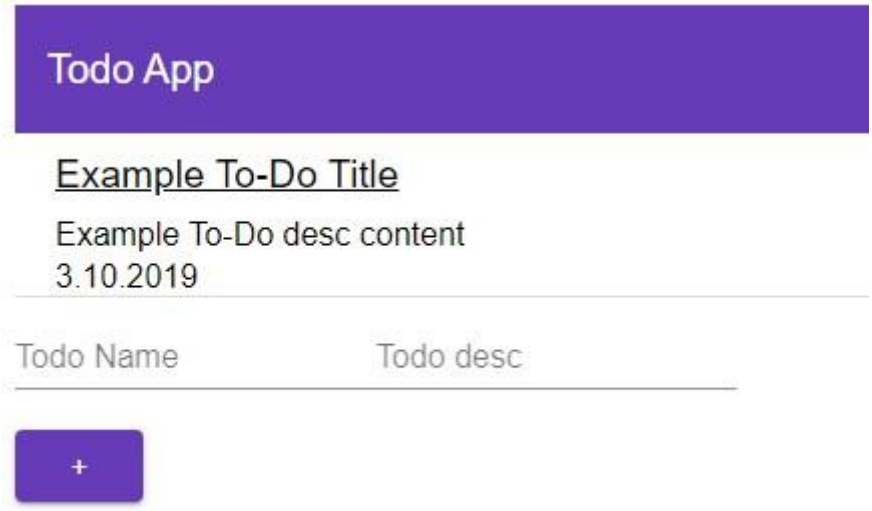
```
<mat-toolbar color="primary">Todo App</mat-toolbar>
<div
  class=""
  *ngFor="let todo of todoItems$ | async"
  (click)="deleteTodo(todo.id)">
  <p class="todoTitle">{{ todo.name }}</p>
  <p class="todoDesc">{{ todo.desc }}</p>
  <p class="todoDesc">{{ todo.initDate }}</p>
  <mat-divider></mat-divider>
</div>

<form (ngSubmit)="addTodo()">
  <mat-form-field>
    <input
      type="text"
      matInput
      [(ngModel)]="newTodoItem.name"
      placeholder="Todo Name"
      name="todoName"
    />
  </mat-form-field>
  <mat-form-field>
    <input
      type="text"
      matInput
      [(ngModel)]="newTodoItem.desc"
      placeholder="Todo desc"
      name="todoDesc"
    />
  </mat-form-field>
  <br />
  <button mat-raised-button color="primary" type="submit">+</button>
</form>

```

Kuvio 9. To-Do näkymän sapluuna

Kuviossa 9 on ToDo-näkymässä olevan .html-sivunsapluuna. Sapluunaan on määritetty elementit, jotka tullaan näyttämään sivulla, kun komponentti ladataan.



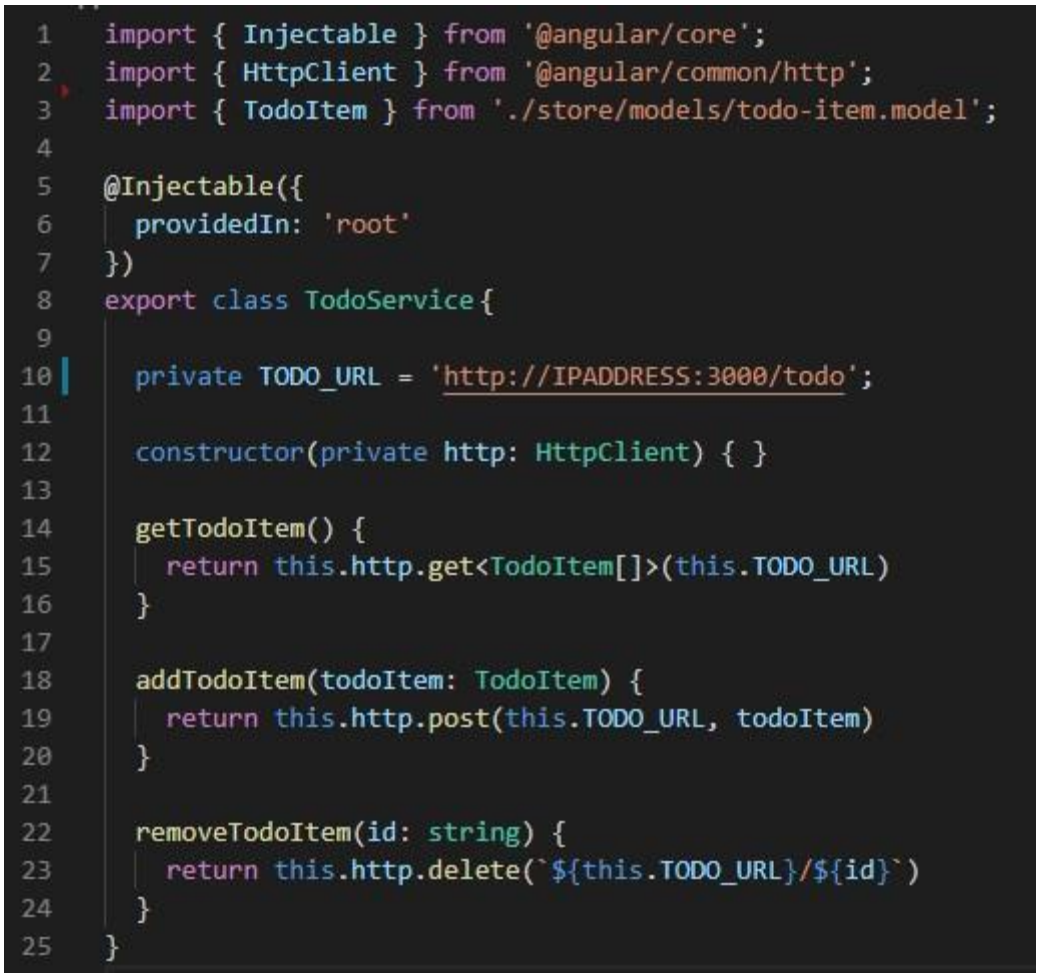
Kuvio 10. Sapluunan tuottama näkymä

Kuviossa 10 nähdään <TodoComponent>-sapluunan tuottama näkymä. Kuviossa 9 rivillä 5 on elementtejä, joihin on määritetty (click)-ominaisuus. (Click) on yksi tapa toteuttaa Kuvion 6 ”Event binding”, eli missä käyttäjä voi vaikuttaa ohjelmalogiikkaan sapluunan puolelta. Esimerkiksi, jos käyttäjä klikkaa elementtiä, jolle on asetettu (click)=”deleteTodo(todo.id)”-ominaisuus, se kertoo ohjelmalogiikalle suorittaa ”deleteTodo”-nimisen metodin.

Angular tukee myös kaksisuuntaista sidontaa. Termillä tarkoitetaan tekniikkaa, jossa käyttäjän muutokset sisältötietoon voidaan tallentaa suoraan ohjelmalogiikkaan. Tämä mahdollistaa esimerkiksi käyttäjän näkymän muokkaamista reaaliaikaisesti samalla, kun se on käyttäjän päätteessä aktiivisena.

Sisältötiedon toimitus ohjelmalogiikkaan Angularissa toimii käyttäen palveluita. Palvelut nähdään Kuviossa 6 nimellä ”services”. Palvelut ovat käytännössä rajapinta datan ja komponenttien välillä. Palvelut vähentävät koodin uudelleenkirjoittamista, sillä palvelu voi toimittaa datan kaikkiin komponentteihin, jotka sitä tarvisevat sen sijaan, että kaikki komponentit noutaisivat sen itse. Palvelun käyttö komponentissa mahdollistetaan lisäämällä ”import”-rivi komponentin .ts-tiedoston alkuun sekä lisäämällä siitä merkintä komponentin ”constructor”-metodiin. Palvelu injektoi tarvittavan datan komponenttiin sovelluksen suorituksen yhteydessä.

Palvelut toimivat myös toiseen suuntaan. Komponentit voivat käyttää palveluita myös datan manipulointiin tallennuspaikassa.



```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { TodoItem } from './store/models/todo-item.model';

@Injectable({
  providedIn: 'root'
})
export class TodoService{

  private TODO_URL = 'http://IPADDRESS:3000/todo';

  constructor(private http: HttpClient) { }

  getTodoItem() {
    return this.http.get<TodoItem[]>(this.TODO_URL)
  }

  addTodoItem(todoItem: TodoItem) {
    return this.http.post(this.TODO_URL, todoItem)
  }

  removeTodoItem(id: string) {
    return this.http.delete(`${this.TODO_URL}/${id}`)
  }
}
```

Kuvio 11. Todo-Service

Kuviossa 11 on todo-palvelu. Tiedoston sisältö määritetään palveluksi lisäämällä sen alkuun @Injectable()-Meta-data-merkintä. Kuviossa nähdään palvelumetodeja, kuten getTodoItems ja addTodoItem, joita käytetään käyttäjän todo-tapahtumien hakemiseen datan tallennuspaikasta. Palvelussa on myös metodeja, kuten removeTodoItem, joilla dataa voidaan poistaa. Palvelu-metodit ovat käytännössä vain GET- ja POST-pyyntöjä, minkä selain tekee palvelimelle (Angular http 2019).

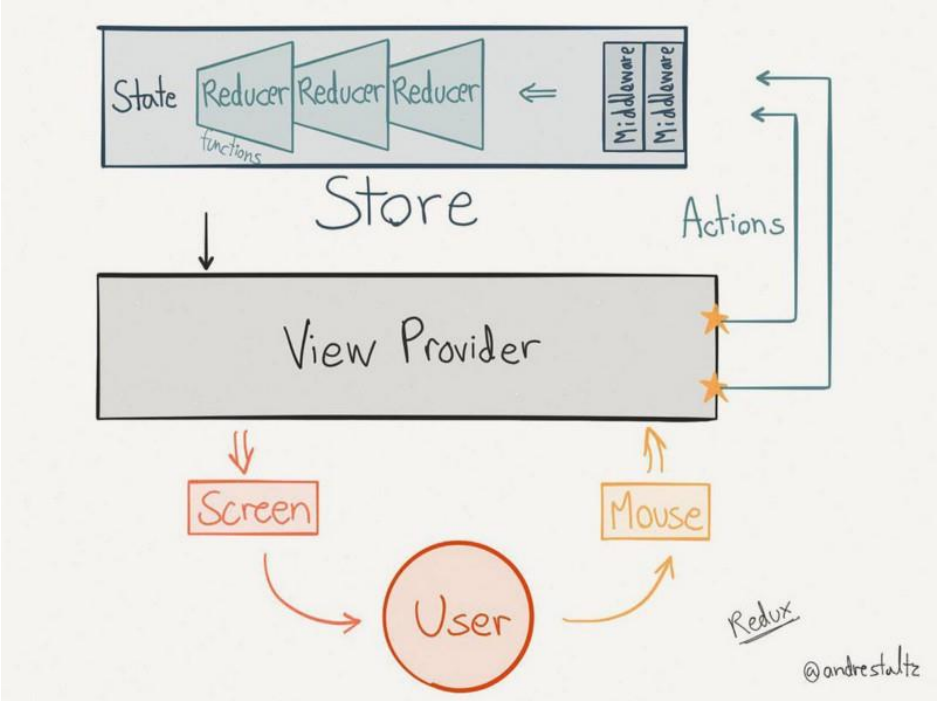
5 ASIAKASYMPÄRISTÖN KIRJASTOT

5.1 Yleistä kirjastoista

Asiakasympäristössä on käytössä kolmannen-osapuolen Angular-komponentteja ja kirjastoja. Niillä toteutetaan esimerkiksi koko sovelluksen tilan hallinta sekä kalenterikomponentti. Tässä luvussa kerrotaan, mitä kirjastoja on käytössä, mihin niitä käytetään sekä miten ne toimivat.

5.2 NgRx – Reactive State for Angular

NgRx on Angularille tehty versio Redux-kirjastosta. Yksinkertaistettuna Reduxin idea on hallita ja päivittää sovelluksen sovellustilaa käyttämällä mahdollisimman vähän ”boilerplate”-koodia. Boilerplate-koodilla tarkoitetaan koodia, joka on kirjoitettuna moneen paikkaan ilman isompia muokkauksia. (Gitconnected 2017.)



Kuvio 12. NgRx-toimintaperiaatekaavio (Gitconnected 2017)

Redux toimii käyttämällä kolmea peruskäsitettä, Actions, Effects, Reducers ja Store. Kuviossa 12 nähdään, että yksinkertaisimmillaan Reduxin toimintajärjestys on: Käyttäjän toimenpide → Action → Effect → Reducer → State muutos → Näkymän päivittäminen.

Kaikki edellä mainitut termit ovat pinnallisesti hyvin yksinkertaisia.

- ”Store” on sovellustila-olio, jossa pidetään koko sovelluksen tilaa. Kaikki tarvittavat tieto on tallennettuna Store-olioon.

- ”Action” tai suomeksi toimenpiteet kertovat Redux Storelle, mitä tehdä.
- ”Effects” on toimenpiteiden takia tehtäviä toimintoja, kuten tiedon hakeminen palvelimelta tai datan muokkaus.
- ”Reducer” vastaanottaa toimenpiteen sekä toimenpiteen jälkeen suoritetun muokkauksen ja päivittää sovellustilan vastaamaan muokkausta.

Asiakasympäristössä käytetään NgRx-Storea hallitsemaan sovellustilaa. Opinnäytetyön kaikki käytännönosuuden osat toimivat Storen ympärillä, joten sen toiminnan vaiheista kerrotaan tarkemmin.

1. Käyttäjä tekee jotain, joka vaatii ohjelmistotilan päivitystä. Esimerkiksi asiakasympäristössä käyttäjä valitsee uuden päivän kalenterista.
2. Angular huomaa käyttäjän toimenpiteen ja käyttää ”Store.dispatch()”-metodia kertomaan Storelle uudesta toimenpiteestä.
3. Store vastaanottaa ”Dispatch”-metodin parametrinä kuvion 13 SET_CALENDAR_RANGE-toimenpiteen nimen ja suorittaa sen.

```
56  export class SetCalendarRange implements Action {
57    readonly type = ScheduleActionType.SET_CALENDAR_RANGE;
58
59    constructor(public payload: CalendarDateRange) {
60    }
61  }
```

Kuvio 13. SetCalendarRange-Action

4. NgRx-Effect huomaa SET_CALENDAR_RANGE-toimenpiteen.

```
112 @Effect()
113 loadCalendarRangeSlotData$ = this.actions$.pipe(
114   ofType<scheduleActions.SetCalendarRange>(scheduleActions.ScheduleActionType.SET_CALENDAR_RANGE),
115   map((action): FilterDateRange => ({
116     startDate: moment(action.payload.start),
117     endDate: moment(action.payload.end),
118   })),
119   concatMap((date) => this.loadRangeSlots(date)),
120 );
```

Kuvio 14. loadCalendarRangeSlotData Effect

Set_calendar_range-toimenpiteen tuloarvoina on uuden kalenteriaikavälin alku- sekä loppumispäivämäärä. Ne lähetetään ”loadRangeSlots”-Effectille.

5. Kuviossa 15 nähdään ”loadRangeSlots”-metodi

```
loadRangeSlots(dateRange: FilterDateRange) {
  return of(dateRange).pipe(
    withLatestFrom(
      this.store.pipe(select(getViewOrEditSchedulesOrAppointments)),
      this.store.pipe(select(getLoadedRange)),
      this.store.pipe(select(getPractitionerId)),
    ),
    filter(([payload, canViewOrEdit, loadedRange]) =>
      canViewOrEdit &&
      !(
        loadedRange &&
        DateUtil.isBetween(payload.startDate, loadedRange.start, loadedRange.end) &&
        DateUtil.isBetween(payload.endDate, loadedRange.start, loadedRange.end)
      )
    ),
    map(([payload, canViewOrEdit, loadedRange, practitionerId]) => new InitiateScheduleRangeLoad({
      ...payload,
      resourceUuid: practitionerId,
    })),
  );
}
```

Kuvio 15. loadRangeSlots-Metodi

Kuviossa 15 tehdään paljon asioita, mutta selityksen yksinkertaistamiseksi keskitytään vain kalenteritapahtumien hakuun. Kuviossa 15 haetaan sen hetkisen käyttäjän uniikki ID, jotta voidaan hakea vain hänelle kuuluvat tapahtumat ja kutsutaan seuraava toimenpide ”InitiateScheduleRangeLoad”.

6. Kuviossa 16 nähdään ”initiateScheduleRangeLoad”-metodi, joka kutsuu toimenpidettä ”LoadAppointments” ja toimittaa sille tarvittavat tuontiarvot.

```
@Effect()
initiateScheduleRangeLoad$ = this.actions$.pipe(
  ofType<scheduleActions.InitiateScheduleRangeLoad>(scheduleActions.INITIATE_SCHEDULE_RANGE_LOAD),
  switchMap(({ payload }) => ([
    new scheduleActions.LoadTimeSlots(payload),
    new scheduleActions.LoadAppointments(payload),
  ])),
);
```

Kuvio 16. initiateScheduleRangeLoad-Effect

7. Kuviossa 17 ” LoadAppointments”-Effect tekee kutsun “scheduleService”-nimiseen palveluun, joka on osa asiakasympäristöä. Palvelu sisältää vain yksinkertaisen http.get-kutsun tietokantaan, johon tapahtumat on tallennettu.

   Jos kutsu palaa onnistuneesti ja tieto saadaan palvelusta, niin Effect kutsuu uutta toimenpidettä ”LoadAppointmentsSuccess”. Kutsun epäonnistuessa käyttäjälle ilmoitetaan ongelmasta ja näytetään virheviesti.

```
@Effect()
loadAppointments$ = this.actions$.pipe(
  ofType<scheduleActions.LoadAppointments>(scheduleActions.ScheduleActionType.LOAD_APPOINTMENTS),
  concatMap(({payload}) => this.scheduleService.getAppointments(payload).pipe(
    map((response) => new scheduleActions.LoadAppointmentsSuccess(response)),
    catchError((response) => of(new scheduleActions.LoadAppointmentsFail(response))),
  )),
);
```

Kuvio 17. loadAppointments-Effect

8. Kuviossa 18 on LOAD_APPOINTMENTS_SUCCESS-Reducer. Se vastaanottaa Effectistä tulevan datan ja liittää sen nykyiseen sovellustilaan.

```
  case ScheduleActionType.LOAD_APPOINTMENTS_SUCCESS: {
    const {ids: uuids, entities} = reduceIdsAndEntities(action.payload, 'uuid');
    return {
      ...state,
      appointmentIds: uuids,
      appointmentEntities: entities,
    };
  }

```

Kuvio 18. LoadAppointmentsSuccess-Action

Riveillä 120-122 näkyvä ”…state” ottaa nykyisen tilan ja liittää siihen ”appointmentIds” sekä ”appointmentEntities” muuttujat. Käyttäjännäkymä päivitetään vastaamaan Reducerin palauttamaa sovellustilaa

Tämä on vain yksi esimerkki asiakasympäristössä toimivasta NgRx-Storesta. Esimerkissä tulee kuitenkin ilmi, miten NgRx-kirjasto ja myös ”Reactive State”-käsite toimii.

5.3 Angular-calendar

Angular 6.0+ calendar on avoimen lähdekoodin kalenterikomponentti, jonka voi lisätä omaan Angular-projektiin. Angular-calendar pohjautuu angular-Bootstrap-calendar komponenttiin. Tämä versio Angular-calendarista ei käytä Bootstrap kirjastoa.

Projektissa oli aluksi käytössä fullcalendar.io (Fullcalendar 2019), mutta se päätettiin vaihtaa Angular-calendar komponenttiin johtuen Angular-calendarin paremmasta yhteensopivuudesta potilastietojärjestelmän tarpeiden kanssa.

5.4 Käyttöliittymäsuunnittelu

Asiakasympäristön käyttöliittymäsuunnittelu tapahtui muutaman työntekijän toimesta. Ensin suunnittelija teki mallikuvat käyttämällä Adobe Photoshop ohjelmaa. Sen pohjalta frontend-kehittäjä teki UI-komponentin, mikä liitetään potilastietojärjestelmään sellaisenaan.

Tämän prosessin takia tässä työssä ei tulla tekemään UI-komponentteja vaan pelkästään ohjelmistologiikkaa.

5.4.1 Ngx-Bootstrap

Käyttöliittymän pohjalla toimii Ngx-Bootstrap-kirjastosta muunneltu versio, jota käytetään pääelementtien toteuttamiseen. Ngx-Bootstrap on kirjasto, mitä käyttäen voi ladata valmiiksi tyyliteltyjä komponentteja suoraan Angulariin. Potilastietojärjestelmässä näitä kyseisiä komponentteja on muunneltu toimimaan paremmin UI-suunnittelijan vision kanssa.

Ngx-Bootstrap käyttää normaalista Bootstrapistä tuttua ”grid”-järjestelmää, jossa käyttäjän ruutu on jaettu 12 osaan. Osia käyttäen voidaan skaalata käyttöliittymän osia sopivan kokoisiksi riippuen päätelaitteen ruudun koosta. Potilastietojärjestelmään on kuitenkin päädytty jakamaan ruutu samalla tavalla osiin, mutta 16 osaan. Niin saadaan jaettua ruutua pienempiin osiin vastaamaan käyttöliittymäsuunnitelmaa.

5.4.2 FontAwesome

Toinen kirjasto, jota käytetään, on FontAwesome. Sen tarkoitus on helpottaa ikonien toteuttamista. FontAwesome on avoimen lähdekoodin vektorigrafiikka kirjasto. Sen kaikki ikonit on suunniteltu tiettyjen sääntöjen mukaan, jotta saadaan aikaan yhtenäinen teema. FontAwesomen kirjasto sisältää yli 5000 valmista ikonia ja niitä tulee jatkuvasti lisää. Ikonien toiminta on testattu kaikilla alustoilla. Ominaisuudet tekevät FontAwesome-kirjastosta helpon valinnan projekteihin, joissa on ikoneja käytössä (Fontawesome 2019.)

5.4.3 Ngx-Perfect-scrollbar

Projektissa on mukana Perfect-scrollbar kirjasto. Perfect-scrollbar on kirjasto, joka parantaa sivun vierintäpalkia. Se luo oletuksena minimaalisen, mutta sulavan vierityspalkin, joka sopii paremmin elementtien sisään verrattuna selainten oletuspalkkiin. Perfect-Scrollbaria käyttäen saadaan myös käyttöön erilaisia metodeja sivun vierittämiseen.

5.4.4 CSS-tyylittelyt

Suuri osa käyttöliittymästä on tehty käyttäen SASS tyylittelyohjeita. Kaikki .scss tyylitiedostot ovat tallennettuna omaan repositorioon, mistä ne liitetään ohjelmistoon käyttämällä NPM komentoa järjestelmän build-vaiheessa.

6 KALENTERIKOMPONENTIN JATKOKEHITYS

6.1 Yleistä jatkokehitysosuudesta

Tässä luvussa kerrotaan ja käsitellään kalenterikomponentin jatkokehitykseen liittyviä käytännön tehtäviä. Kappaleessa tulee olemaan useita tehtäviä johtuen niiden suppeudesta. Jokaisesta tehtävästä tullaan kertomaan mikä tehtävän tavoite on, miksi se tehdään, miten se tehdään ja mitä vaikeuksia sen tekemisessä oli. Tehtävien esittelyssä tullaan käyttämään kuvankaappauksia koodista, jotta siitä saadaan selvä kuva aikaiseksi.

6.2 Tehtävä 1 – Kalenterin viikkonäkymän oikeaan kohtaan siirtäminen

Ensimmäinen tehtävä liittyy kalenterikomponentin viikkonäkymään. Viikkonäkymässä nähdään kaikki viikolle kohdistuvat tapahtumat. Oletuksellisesti viikkonäkymä alkaa aina klo 00:00 eli sivun ylälaidasta.

Description

As a user, I want that when the calendar is opened in the week mode, it is aligned according to the earliest entries. For example, if there are open time slots in my calendar as follows

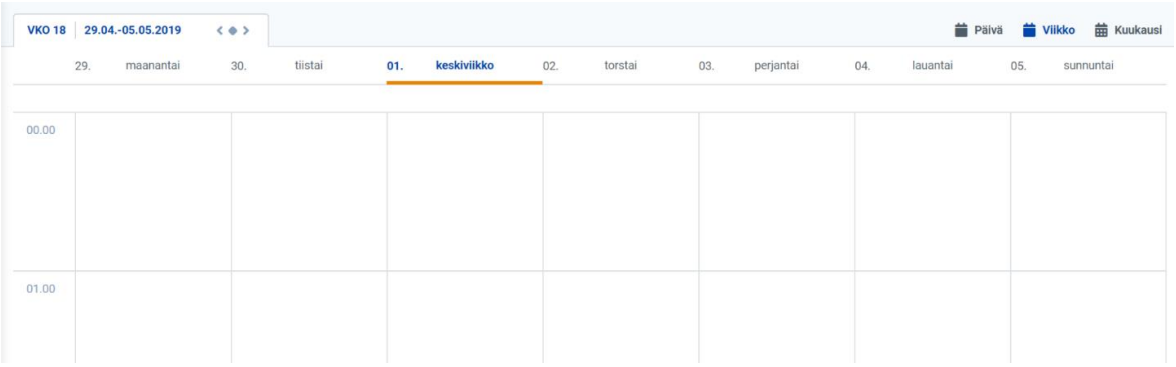
- Mon 11-18
- Tue 10-18
- Thu 8-12

the calendar should be aligned to show the days from 8 onwards.

Kuva 19. Työtehtäväkuvaus

Kuva 19 esittää työtehtävän kuvausta. Työtehtävä on suunnitella ja toteuttaa ohjelmistologiikka, joka vierittää sivun viikon aikaisimman tapahtuman kohdalle. Kuvauksen esimerkissä annetaan kolme kalenterivarausesimerkkiä: maanantai Klo 11-18, tiistai Klo 10-18 ja torstai Klo 8-12. Jotta työ saadaan toteutettua, niin tulee kehittää ohjelmistologiikka, joka vierittää sivun Torstai Klo 8 kohtaan.

Kuvaksessa ei mainita, mitä tehdään, jos viikolla ei ole mitään tapahtumia. Oletuksellisesti kalenterin viikkonäkymä tällöin hakeutuu alkuun eli Klo 00:00 kohtaan Kuvion 20 mukaan.



Kuvio 20. Kalenterinäkymä ennen toteutusta

Kuviossa 20 nähdään kalenterikomponentin viikkonäkymä ennen kuin sille on tehty mitään. Kuten aiemmin mainittu, se oletuksellisesti aloittaa aina Klo 00:00 kohdasta. Kyseisellä viikolla on tapahtumia, mutta ne eivät näy kuviossa.

```
<mwl-calendar-week-view #weekCalendar
  [viewDate]="viewDate"
  [events]="events"
  [eventTemplate]="weekEventTemplate"
  [headerTemplate]="weekHeaderTemplate"
  [hourSegmentHeight]="hourSegmentHeight"
  [hourSegments]="hourSegments"
  [emrCurrentTimeLine]="weekCalendar"
>
</mwl-calendar-week-view>
```

Kuvio 21. Viikkonäkymän sapluuna

Kuviossa 21 on kalenterin viikkonäkymän .HTML-sapluuna. Siinä nähdään, että sapluunaan syötetään ”events”-niminen objektiryhmä. Events sisältö tulee NgRx-Storesta.

```
27    ngOnChanges(){
28      console.log(this.events);
29    }
```

Kuvio 22. ngOnChanges

Jotta nähdään mitä ”Events” toimittaa komponenttiin, laitetaan viikkonäkymän .ts-tiedostoon kuvion 22 mukainen ”ngOnChanges”-metodi ja sen sisään ”console.log(this.events)”-koodilause. Metodi ”ngOnChanges” tarkkailee komponenttiin sisään tulevien datamuuttujien sisältöä ja suorittaa muutoksen huomattua sen sisään kirjoitetut koodilauseet. Rivin 28 koodi tulostaa ”this.events”-dataryhmän sisällön.

```
app-calendar-user-ca…ndar-module.js:7557
▼Array(4) i
  ▶0: {start: Mon Apr 29 2019 11:00:00 GMT+0300 (Eastern European Summer Time), end: Mon Apr 29 2019 11:…
  ▶1: {start: Mon Apr 29 2019 11:20:00 GMT+0300 (Eastern European Summer Time), end: Mon Apr 29 2019 11:…
  ▶2: {start: Mon Apr 29 2019 11:40:00 GMT+0300 (Eastern European Summer Time), end: Mon Apr 29 2019 12:…
  ▶3: {start: Mon Apr 29 2019 12:00:00 GMT+0300 (Eastern European Summer Time), end: Mon Apr 29 2019 12:…
    length: 4
  ▶__proto__: Array(0)
>
```

Kuvio 23. Viikkonäkymän tapahtumat

Events-dataryhmä sisältää sen hetkisen näkymän kaikki tapahtumat. Työtehtävää varten ainoa tärkeä osa dataa on jokaisen tapahtuman ”start”-ominaisuus. Siihen on määritetty, mihin aikaan tapahtuma alkaa.

Tapahtumat on järjestelty päivämäärän mukaan pienimmästä suurimpaan. Eli jos samalla viikolla on toisena päivä aikaisempia tapahtumia niin viikon ensimmäisen päivän tapahtumat ovat silti ensimmäisenä dataryhmässä. Tähän ratkaisuun palataan myöhemmin.

6.2.1 Ratkaisu

Jotta sivua voidaan vierittää automaattisesti, tulee paikantaa, mikä html-elementti pystyy sitä tekemään. Projektissa on mukana käytössä ”Ngx-perfect-scrollbar”-niminen kirjasto, jota käyttäen saadaan aikaa yleistoimiva vierityselementti, jota voidaan käyttää samanlaisena joka paikassa.

```
33    </div>
34    <perfect-scrollbar #ccScrollElement class="c-calendar__body">
35      <div [ngSwitch]="calendarDate.view">
36        <emr-calendar-day *ngSwitchCase="calendarView.day"
37          [viewDate]="calendarDate.selected"
38          [events]="events"
39          [hourSegmentHeight]="hourSegmentHeight"
40          [hourSegments]="hourSegments"
41        ></emr-calendar-day>
42        <emr-calendar-week *ngSwitchCase="calendarView.week"
43          [scrollElement]="directiveRef"
44          [viewDate]="calendarDate.selected"
45          [events]="events"
46          [hourSegmentHeight]="hourSegmentHeight"
47          [hourSegments]="hourSegments"
48        ></emr-calendar-week>
49        <emr-calendar-month *ngSwitchCase="calendarView.month"
50          [viewDate]="calendarDate.selected"
51          [events]="events"
52        ></emr-calendar-month>
53      </div>
54    </perfect-scrollbar>
55  </div>
56
```

Kuvio 24. Kalenterinäkymän sapluuna

Kuviossa 24 nähdään kalenterinäkymän .html-sapluuna. Rivillä 34 luodaan ”perfect-scrollbar”-tägi, jonka sisään on laitettu kaikki eri kalenterinäkymävaihtoehdot. Kaikilla kalenterinäkymävaihtoehdoilla on oma komponentti. Rivillä 34 nähdään myös ”#ccScrollElement”, joka on Angularissa käytettävä ”custom directive”.

Kalenterin viikkonäkymän komponentti ei oletuksellisesti näe tai pysty käyttämään ”perfect-scrollbar”-elementtiä. Se johtuu komponenttien tasojärjestyksestä. Kalenterikomponentti on viikkonäkymä komponentin ”Parent”- tai toisin sanoin ylemmän tason komponentti. Viikkonäkymä on ”child”- tai alemman tason komponentti. Alemmat tasot eivät näe, mitä ylemmillä tasoilla tapahtuu, ellei niille kerrota.

Jotta viikkonäkymä komponentti osaa käyttää ”perfect-scrollbar”-elementin vieritystoimintoa, tulee se syöttää sille käyttämällä ”@Input”-toimintoa. ”@Input” on toiminto, jolla voidaan syöttää ”Parent”-komponentista dataa ”Child”-komponenttiin.

```
12  export class CalendarComponent {
13    @Input() events: CalendarEvent<CalendarEventSlot>[];
14    @Input() calendarDate: CalendarDateRange;
15    @Input() @ViewChild('ccScrollElement') directiveRef?: PerfectScrollbarDirective;
16
```

Kuvio 25. @ViewChild-toiminto

Kuviossa 25 rivillä 15 käytetään ”@ViewChild”-toimintoa kaappaamaan aiemmin mainittu ”#ccScrollElement” ja tallentamaan siitä ”directive reference” muuttujaan ”directiveRef”. ”@ViewChild” on Angularin toiminto, joka etsii saman komponentin sisällöstä hakukriteeriä vastaavaa elementtiä. Se tallentaa ensimmäisen osuman määritettyyn muuttujaan.

TypeScriptiä kirjoittaessa voidaan käyttää muuttujaa määrittäessä muuttujatyyppiä. Rivin 15 lopussa nähdään ”PerfectScrollbarDirective”-muuttujatyyppimääritelmä. Eli tässä tapauksessa ”directiveRef”-muuttujaan voidaan vain tallentaa elementti, joka on tyyppiä ”PerfectScrollbarDirective”.

```
    export declare class PerfectScrollbarDirective implements OnInit,
41      scrollToY(y: number, speed?: number): void;
42      scrollToTop(offset?: number, speed?: number): void;
43      scrollToLeft(offset?: number, speed?: number): void;
44      scrollToRight(offset?: number, speed?: number): void;
45      scrollToBottom(offset?: number, speed?: number): void;
46      scrollToElement(qs: string, offset?: number, speed?: number):
47      private animateScrolling;
48  }
```

Kuvio 26. PerjectScrollBarDirective

Muuttujan ollessa tyyppiä ”PerfectScrollbarDirective”, saadaan käyttöön useita hyödyllisiä metodeja. Tätä työtä varten tarvitaan Kuvion 26 rivin 41 metodia ”scrollToY”. Kyseessä oleva metodi ottaa tuontiarvoina ”y” eli numeraalinen arvo pikseleitä kelattavan elementin yläreunasta, sekä ”speed”, jolla voi vaihtaa kelauksen nopeutta. Metodia kutsuttaessa se vierittää elementtiä määrän ”y” alaspäin käyttäen ”speed” nopeutta.

```
<emr-calendar-week *ngSwitchCase="calendarView.week"
  [scrollElement]="directiveRef"
  [viewDate]="calendarDate.selected"
  [events]="events"
  [hourSegmentHeight]="hourSegmentHeight"
  [hourSegments]="hourSegments"
></emr-calendar-week>
```

Kuvio 27. Viikkonäkymän sapluuna

Jotta kuvion 25 ”@Input” toimii, se tulee määrittää tuontiarvoksi viikkonäkymälle. Kuviossa 27 se tehdään kalenterikomponentin sapluunassa. ”[scrollElement]” on referenssi viikkonäkymäkomponentin muuttujaan ja siihen määritetään aiemmin luotu ”directiveRef”. Kuviossa 27 myös määritetään muuttuja ”[hourSegmentHeight]”.

```
32    ngOnChanges() {
33      if (this.scrollElement) {
34        this.scrollElement.directiveRef.scrollToY(
35          this.hourSegmentHeight * this.weekStartHour() - this.scrollOffset,
36          this.scrollSpeed
37        );
38      }
39    }
40  }
```

Kuvio 28. Automaattivieritys ngOnChanges-toiminnossa

Kaikki tarvittavat osat ovat nyt kasassa ja voidaan aloittaa ratkaisun kirjoittaminen. Kuviossa 28 sivun automaattinen vieritys suoritetaan ”ngOnChanges”-logiikalla. Sitä käytetään ”ngOnInit” sijaan, koska NgRx-Store on asynkrooninen, tarkoittaen että ”events” saapuu sivulle pienellä viiveellä. Näin voidaan varmistaa, että ”events” sisältää dataa ennen kuin toimintoa suoritetaan.

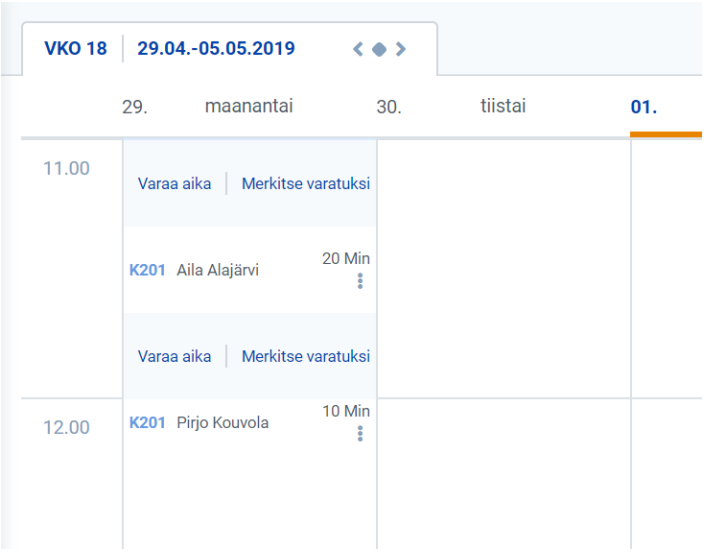
Rivillä 33 alkavassa if lauseessa tarkistetaan, onko vieritettävä elementti latautunut sivulle. Sen jälkeen kutsutaan aiemmin mainittua ”scrollToY”-metodia ”PerfectScrollbarDirective”-kirjastosta. Sille syötetään ensimmäiseksi ”y” tuontiarvoksi laskutoimitus, missä on muuttuja sekä komponentin metodi. ”this.hourSegmentHeight” on kuviossa 20 nähtävissä olevan tuntisegmentin korkeus pikseleinä.

```
18    defaultWeekStartHour = 5;
19    scrollOffset = 50;
20    scrollSpeed = 100;
21
22    weekStartHour() {
23      if (!this.events[0]) {
24        return this.defaultWeekStartHour;
25      }
26      const hours = this.events.sort((a, b) => {
27        return a.start.getHours() - b.start.getHours();
28      });
29      return hours[0].start.getHours();
30    }
```

Kuvio 29. weekStartHour metodi

Toinen yhtälönosa ”this.weekStartHour()” on viikkonäkymässä oleva metodi, joka nähdään kuviossa 29. Metodi ensin tarkistaa if-lauseella onko ”events”-data saapunut komponenttiin. If-lauseessa oleva ”[0]” tarkistaa onko ”events”-taulukossa ensimmäinen ominaisuus. Tämä tehdään, koska ”events” tuodaan komponenttiin tyhjänä, jos kyseisellä viikolla ei ole yhtään tapahtumaa. Viikon ollessa tyhjä, palautetaan paikallinen muuttuja ”defaultWeekStartHour”.

Tapahtumat järjestetään uudelleen käyttämällä JavaScriptin metodia ”array.sort()”. Järjestyssääntönä käytetään tapahtumien ”start”-ominaisuuden tuntien määrää. Uudelleen järjestelty dataryhmä tallennetaan ”hours”-muuttujaan, jossa pienin tuntiarvo on ensimmäisenä. Metodi palauttaa pienimmän tuntimäärän takaisin yhtälöön. Jos viikolla ei ollut yhtään tapahtumaa, metodi palauttaa ”defaultWeekStartHour” eli kuvion 29 mukaan numeraalisen arvon 5. Kuviossa 29 nähdään myös yhtälön viimeinen osa eli ”scrollOffset”

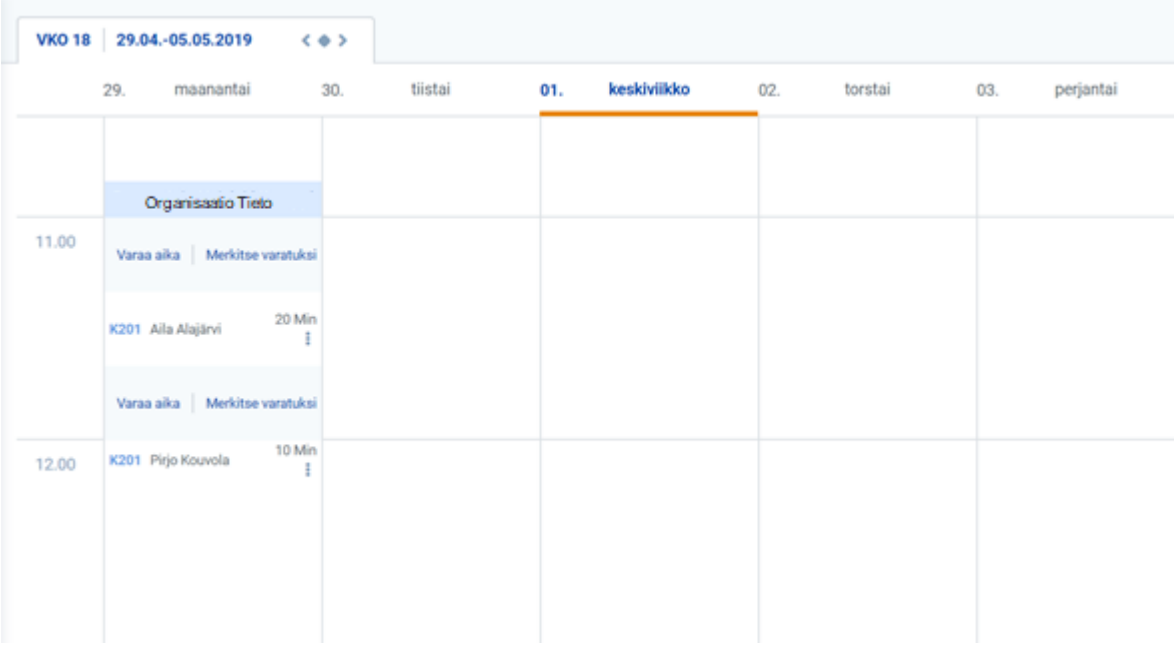


Kuvio 30. Viikkokalenterinäkymä automaattisen vierityksen jälkeen

Kuviossa 30 nähdään mitä tapahtuu, jos yhtälö suoritetaan ilman ”scrollOffset”-muuttujaa. Tapahtuman paikkatietopalkki jää näkymän ulkopuolelle, koska muuttuja ”hourSegmentHeight” kerrottuna metodin ”weekStartHour” palautusarvolla on ensimmäisen tapahtuman alkamisaika. Elementti vierii siihen asti, joten tapahtuman yläpuolella oleva paikkatietopalkki menee piiloon. Sen takia yhtälöstä vähennetään 50 pikseliä, joka on tarpeeksi, jotta paikkatietopalkki saadaan näkyviin. Luku ”50” saadaan tapahtumapaikkatietoelementin maksimi korkeudesta. Lopullinen laskutoimitus vierityksen määrään on:

Tuntisegmentin korkeus pikseleinä * viikon ensimmäisen tapahtuman alkamistunti * ylivierityksen esto.

”weekDefaultStartHour”-muuttujan arvo palautetaan yhtälön toiseen muuttujaan, jos viikolla ei ole yhtään tapahtumaa, eli siinä tapauksessa kalenteriviikko vieritetään alkamaan klo 5.



Kuvio 31. Viikkokalenterinäkymä scrollOffsetin jälkeen

Kuviossa 31 on näkymä, johon kalenteri latautuu, kun sivu ladataan. Kyseisen viikon ensimmäinen tapahtuma on klo 11. Kalenteri vierittää elementin hieman klo 11 yläpuolelle, jotta tapahtumanpaikkatieto on vielä näkyvissä. Se tekee myös kalenterista esteettisesti paremman näköisen käyttäjälle, jos asiat, mitä pitää katsoa, eivät ole aivan kiinni elementin yläreunassa.

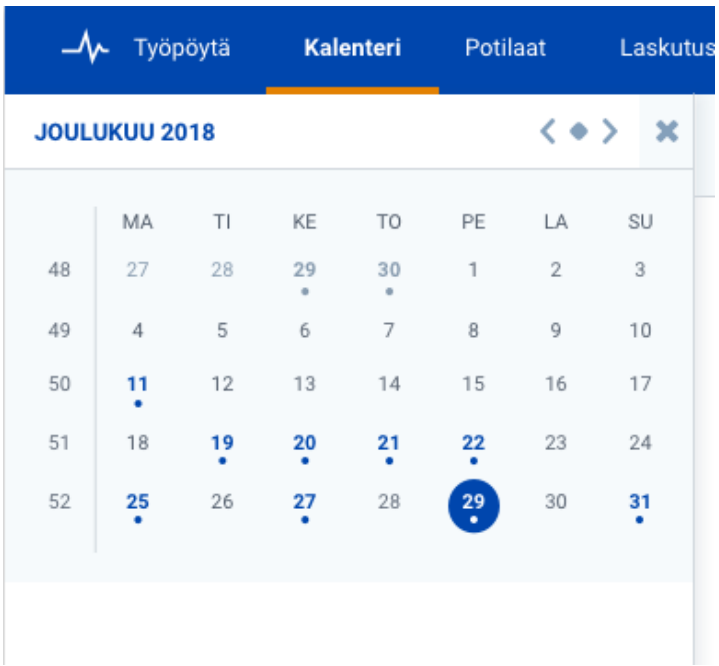
6.2.2 Yhteenveto tehtävästä 1

Lopputulos työstä 1 on hyvä. Keskustelua käytiin kelausmetodien ”scrollToY” sekä ”scrollTo” kesken. ScrollTo mahdollistaisi sivun kelauksen samalla tavalla, mutta päädyttiin kuitenkin ”scrollToY”-vaihtoehtoon. Päätös johtui metodien tuloarvoista. ”ScrollTo” ottaa kolme tuloarvoa: x, y ja speed. ”ScrollToY” ottaa vain: y ja speed.

Jotta koodi pysyy helppolukuisena, tulee välttää metodeihin pelkkien numeraalisten arvojen syöttämistä. Siksi kaikki staattisetkin arvot on asetettu muuttujiin tiedoston alussa. ”ScrollToY” metodia käyttäen säästettiin yksi turha muuttujanimi tiedostosta. Näin saadaan pidettyä tiedostot mahdollisimman lyhyinä sekä selkolukuisina.

6.3 Tehtävä 2 – Tapahtumapäivien esittäminen sivupalkin kalenterissa

Toisen tehtävän idea on yksinkertainen. Kun käyttäjä käyttää sivun sivupalkin ”date-picker”-toimintoa, tulee hänen nähdä kaikki päivät, joissa on varauksia.



Kuvio 32. Työtehtävän 2. tavoite

Kuviossa 32 nähdään päivät, jolloin on tapahtumia merkittynä eri tavalla. Ideana tämä ei ole kovinkaan monimutkainen ja jossain määrin itsestään selvyys.

Sivupalkin date-pickerin toiminta on tarkasti määritelty. Käyttäjän tulee pystyä selailemaan kuukausia eteen- ja taaksepäin ilman, että kalenterinnäkymä muuttuu mitenkään. Näkymä muuttuu vasta, kun käyttäjä valitsee jonkun päivämäärän date-pickeristä. Päivän valittua kalenterinäkymän tilan tulee pysyä samana, esim. viikkonäkymänä, mutta vaihtaa päivämäärä vastaamaan valittua päivää.

Työssä on kaksi selvää ongelmaa ratkaistavaksi. Ensin tulee selvittää, miten voidaan merkitä tapahtumapäivät eri tavalla. Tästä tekee hankalaa date-picker-komponentin toiminnallisuus.

Toinen selvityksen kohde on, että miten saadaan tapahtumat NgRx-Storesta date-picker-komponenttiin. Siitä vaikean tekee Store-state, jossa tapahtumat ovat tallennettuna. Tapahtumat valitaan kalenterinäkymän mukaan. Jos kalenteri on viikon X kohdalla niin NgRx-Storessa on vain sen viikon tapahtumat. Tapahtumadatan lataaminen Storeen onnistuu vain muuttamalla kalenterinäkymän päivämääriä. Tätä ei voida tehdä date-picker-komponentista, sillä se rikkoo järjestelmän toiminnalle asetettuja sääntöjä.

6.3.1 Ratkaisu

Jotta päivien ulkonäköä voidaan muuttaa, tulee ne pystyä erottamaan jollakin tavalla. Se saadaan aikaan päivämerkinnän sapluunassa.

```
15  <ng-template #customDay let-date="date" let-currentMonth="currentMonth" let-selected="selected"
16    <div class="btn-light ngb-datepicker-day-view"
17         [id]="getUniqueId(date)"
18         [class.today]="isToday(date)"
19         [class.bg-primary]="selected"
20         [class.text-white]="selected"
21         [class.text-muted]="!selected && (currentMonth !== date.month || disabled)"
22         [class.outside]="!selected && (currentMonth !== date.month || disabled)"
23         [class.active]="focused">
24      {{date.day}}
25    </div>
26  </ng-template>
```

Kuvio 33. customDay-sapluuna

Kuviossa 32 näkyvät päivä merkinnät luodaan kuvion 33 ”customDay”-sapluunalla. Päivälle määritellään Class-määritelmiä tyylittelyitä varten. Päivän erittelyä varten täytyy päiväelementille antaa ”[id]”, joka toimii sen uniikkina tunnistustietona. Se saadaan määritettyä kutsumalla komponentin metodia ”getUniqueId()”, jolle annetaan päivän objekti tuontiarvona.

```
{
    "day": 2,
    "month": 5,
    "year": 2019
}
```

Kuvio 34. Date-objekti

Jokaisella päivällä on kuvion 34 mukainen ”date”-objekti, joka sisältää kyseisen päivän päivä-, kuukausi- sekä vuositiedon. Date-objekti on date-picker-komponentin luoma.

```
55    getUniqueId(date) {
56      return date.year.toString() + '_' + date.month.toString() + '_' + date.day.toString();
57    }
```

Kuvio 35. getUniqueId-metodi

Kuviossa 35 nähdään komponentin ”getUniqueId”-metodi, jota kutsutaan kuviossa 33. Metodi vastaanottaa date-objektin ja palauttaa merkkijonon, joka sisältää kyseisen date-objektin ominaisuudet. Palautettavan tiedon muodoksi tulee: Vuosinumero_KuukaudenNumero_PäivänNumero. Tällä tavalla saadaan jokaiselle päivälle [id] tieto, joka on uniikki kyseiselle päivälle. Selaimen kehitystyökaluja käyttämällä nähdään, että päiväelementille on nyt ”id” merkittynä kuviossa 36.

```
<div _ngcontent-c27 class="btn-light ngb-
datepicker-day-view has-events" id=
"2019_4_23"> 23 </div> == $0
```

Kuvio 36. Elementin tiedot

```
19    <emr-date-picker #dp [modelDate]="activeDate"
20      [events]="events"
21      (selectDate)="selectNewDate($event)"
22      [navigation]="'none'"
23      [showMonthNames]="false"
24      [addBackground]="true"
25      ></emr-date-picker>
```

Kuvio 37. Tapahtumat date-picker-komponenttiin

Kalenterinäkymän tapahtumat syötettiin kuvion 37 ”date-picker”-komponenttiin, jotta on jotain dataa, jolla testata ratkaisua. Tapahtumat syötettiin käyttäen aiemmin läpikäytyä ”@Input”-toimintoa.

```
@Effect()
updateCalendarRangeOnSelect$ = this.actions$.pipe(
  ofType<scheduleActions.SelectCalendarRangeDate>(scheduleActions.ScheduleActionType.SELECT_CALENDAR_RANGE_DATE),
  map((action) => action.payload),
  withLatestFrom(this.store.pipe(select(getCalendarRange))),
  filter(([payload, range]) => !DateUtil.isBetween(payload, range.start, range.end)),
  map(([payload, range]) => {
    // const format = ScheduleUtil.calendarViewToDateFormat(range.view);
    return new scheduleActions.SetCalendarRange({
      ...range,
      selected: payload.format(),
      start: payload.startOf('month').format(),
      end: payload.endOf('month').format(),
    });
  })
);
```

Kuvio 38. updateCalendarRangeOnSelect-Effect

Testausta varten muutetaan tapahtumien hakemisesta vastaavaa NgRx-Effectiä. Muutetaan sen tapahtumien hakuväliä niin, että näkymästä huolimatta aina haetaan kokonaisen kuukauden verran tapahtumia.

```
for (let i = 0; i < this.events.length; i++) {
  const _year = this.events[i].start.getFullYear().toString();
  const _month = this.events[i].start.getMonth() + 1;
  const _date = this.events[i].start.getDate().toString();
  const _elementId = _year + '_' + _month.toString() + '_' + _date;
  if (!document.getElementById(_elementId).classList.contains('has-events')) {
    document.getElementById(_elementId).className += ' has-events';
  }
}
```

Kuvio 39. For-loop

Päiville, missä on tapahtumia, tulee luoda oma class-määritelmä. Se voidaan tehdä käymällä läpi kaikki ”events”-dataryhmän jäsenet. Kaikista jäsenistä otetaan ”.start”-päivämäärän vuosi, kuukausi ja päivä. Niistä muodostetaan saman muotoinen merkkijono kuin aikaisemmin annetut [id]-merkinnät. Merkkijono tallennetaan väliaikeiseen muuttujaan ”_elementId”. Nyt voidaan käyttää ”document.getElementById”-metodia ja asettaa päivälle ”has-events”-class-määritelmä.

```
24  .has-events {
25    color: #024fb6 !important;
26    display: inline-flex;
27    font-weight: bold;
28    padding-left: 0.55rem !important;
29  }
30
31  .has-events::after{
32    position: relative;
33    top: 0.7rem;
34    right: 0.6rem;
35    font-size: 1.4rem;
36    content:"•";
37  }
```

Kuvio 40. Css-tyylittelyt

Kuviossa 40 määritetään ”has-events”-class-määritelmälle css-tyylittelyt. Jos elementillä on ”has-events” niin sen fontti lihavoidaan sekä sen väriä muutetaan tummempaan siniseen. Kuviossa 41 nähtävä piste lisätään päivämäärän alle käyttämällä Css::after psuedo-class-lisämääreellä. Sillä saadaan lisättyä elementtiin tyylittelyjä sen jälkeen, kun pää-tyylit on ladattu.

Kuvio 41. Ratkaisun tulos

Kuviossa 41 nähdään, että ratkaisu toimii ja jokaisella päivällä on nyt eri ulkonäkö, jos päivällä on tapahtumia. Oikeiden tapahtumien esittämiseksi date-pickerissä, tulee sille luoda oma tapahtumat sisältävä muuttuja, joka eristetään pääkalenterista.

Kuvio 42. Date-picker header-elementti

Kuviossa 42 on date-picker-komponentin header-elementti. Käyttäjä selaa kuukausia kuviossa nähtävillä nuolilla.

```
8   <button class="c-toolbar__btn c-toolbar__btn--sm" type="button" (click)="changeMonth(-1)">
9     <i class="fas fa-angle-left"></i>
10  </button>
11  <button class="c-toolbar__btn c-toolbar__btn--sm" type="button" (click)="selectToday()">
12
13  <button class="c-toolbar__btn c-toolbar__btn--sm" type="button" (click)="changeMonth(1)">
14    <i class="fas fa-angle-right"></i>
15  </button>
```

Kuvio 43. Header-elementin sapluuna

Nuolet ovat sapluunassa yhdistetty ”changeMonth()”-metodiin käyttämällä (click)-sidontaa. ”changeMonth()”-metodille lähetetään tuontiarvona ”-1” tai ”1”. ”-1” lähetetään, jos käyttäjä haluaa palata edelliseen kuukauteen date-picker-näkymässä. ”1” jos käyttäjä haluaa nähdä seuraavan kuukauden.

```
52  onSelectNewMonth(date: moment.Moment) {
53    this.store.dispatch(new SelectDatePickerRangeDate(date));
54  }
55
56  changeMonth(direction: number) {
57    this.shownDate.add(direction, 'month');
58    this.datePicker.navigateTo(this.shownDate);
59    this.onSelectNewMonth(this.shownDate);
60  }
61
```

Kuvio 44. changeMonth metodi

Kuviossa 44 nähdään edellä mainittu ”changeMonth()”-metodi. Sapluunasta tuleva tuontiarvo tuodaan metodiin nimellä ”direction”. Se sisältää arvon ”-1” tai ”1”.

Rivillä 57 nähtävä ”shownDate”-muuttuja sisältää tämän hetkisen näkyvissä olevan kuukauden päivämäärän. Muuttujan kuukausi päivämäärätietoa voidaan muuttaa käyttämällä ”.add()”-metodia. Metodi vastaanottaa kaksi arvoa. Ensimmäinen tuontiarvo on lisättävien tai vähennettävien aikajaksojen määrä. Toinen on aikajakson formaatti.

Add-metodille syötetään ”direction”-tuontiarvo, joka sisältää arvon eteen- tai taaksepäin selaamisesta, sekä ”month”-aikajaksoformaatti. Rivillä 58 date-picker-komponentille ilmoitetaan, että date-pickerin pitää esittää uusi valittu kuukausi. Rivillä 59 kutsutaan uutta metodia ”onSelectNewMonth()”.

OnSelectNewMonth-effectmetodi vastaanottaa ”shownDate”-muuttujan sisällön tuontiarvona. Metodi käyttää Store.dispatch-toimia uuden NgRx-Actionin kutsumiseen. Storelle lähetetään arvona uuden valitun kuukauden alku- ja loppupäivämäärät.

```
166  @Effect()
167  updateDatePickerRangeOnSelect$ = this.actions$.pipe(
168    ofType<scheduleActions.SelectDatePickerRangeDate>(
169      scheduleActions.ScheduleActionType.SELECT_DATE_PICKER_RANGE_DATE),
170    map((action) => action.payload),
171    withLatestFrom(this.store.pipe(select(getDatePickerRange))),
172    map(([payload, range]) => {
173      return new scheduleActions.SetDatePickerRange({
174        ...range,
175        selected: payload.format(),
176        start: payload.startOf('month').format(),
177        end: payload.endOf('month').format(),
178      });
179    })
180  );
```

Kuvio 45. updateDatePickerRangeOnSelect-Effect

Jotta uusi aikavälivalinta voidaan asettaa Storeen, niin tulee ensin valita vanha tieto. Se nähdään kuvion 45 rivillä 171. Storeen tehdään ”select”-valinta, joka palauttaa nykyisen aikavälin. Seuraavaksi uuden aikavälin alku- sekä loppupäivämäärät ja nykyisen aikavälin formaatti lähetetään ”SetDatePickerRange”-Store Actionille.

```
119   @Effect()
120   loadDatepickerRangeSlotData$ = this.actions$.pipe(
121     ofType<scheduleActions.SetDatePickerRange>(
122       scheduleActions.ScheduleActionType.SET_DATE_PICKER_RANGE),
123     map((action): FilterDateRange => ({
124       startDate: moment(action.payload.start),
125       endDate: moment(action.payload.end),
126     })),
127     concatMap((date) => this.loadDatePickerRangeSlots(date)),
128   );
```

Kuvio 46. loadDatepickerRangeSlotData-Effect

SetDatePickerRange-effectmetodi vastaanottaa aikavälin ja erittelee sen muuttujiin ”start-Date” ja ”endDate”. Erittelyyn käytetään moment-nimisen kolmannenosapuolen kirjastoa, jonka toiminnot käsittelevät aikaa. Moment-kirjaston ”.start” ja ”.end” laskevat annetusta päivämäärästä alku- sekä loppuajankohta. Ajankohtat lähetetään seuraavaan metodiin ”loadDatePickerRangeSlots”.

```
288   loadDatePickerRangeSlots(datePickerRange: FilterDateRange) {
289     return of(datePickerRange).pipe(
290       withLatestFrom(
291         this.store.pipe(select(getViewOrEditSchedulesOrAppointments)),
292         this.store.pipe(select(getLoadedDatePickerRange)),
293         this.resourceIdOfSchedule$
294       ),
295       map(([payload, canViewOrEdit, loadedRange, practitionerId]) =>
296        new scheduleActions.LoadDatePickerSlotEntries({
297         ...payload,
298         resourceUuid: practitionerId,
299       })),
300     );
301   }
```

Kuvio 47. loadDatePickerRangeSlots-metodi

Kuviossa 47 loadDatePickerRangeSlots-metodi vastaanottaa valitut ajankohdat date-pickerin tapahtumien lataukseen. Metodin tehtävä on ottaa käyttäjän uniikki ID-arvo ja lisätä se datapakettiin, joka lähetetään seuraavalle toiminnolle. Käyttäjän ID nähdään rivillä 298. ”practitionerId” on jokaisella käyttäjälle oleva uniikki tieto, jota käyttäen tallennetaan henkilökohtaiset tapahtumat ja saadaan selville, kuka sekä mikä käyttäjän rooli on: esimerkiksi lääkäri vai järjestelmänvalvoja.

```
141   @Effect()
142   initiateDatePickerRangeLoad$ = this.actions$.pipe(
143     ofType<scheduleActions.LoadDatePickerSlotEntries>(
144       scheduleActions.ScheduleActionType.LOAD_DATE_PICKER_SLOT_ENTRIES),
145     switchMap(({ payload }) => (this.scheduleService.getScheduleSlotEntries(payload).pipe(
146       map((response) => new scheduleActions.LoadDatePickerSlotEntriesSuccess(
147         { filter: payload, response: response }
148       )),
149       catchError((response) => of(new scheduleActions.LoadDatePickerSlotEntriesFail(response))),
150     )),
151     ));
```

Kuvio 48. initiateDatePickerRangeLoad-Effect

Kuviossa 48 on NgRx-effect, joka odottaa LOAD_DATE_PICKER_SLOT_ENTRIES-Actionin kutsua. Effect vastaanottaa käyttäjän valitseman aikavälin, jota käytetään oikeiden tapahtumien hakuun. Aikaväli syötetään projektin schedule-servicen getScheduleSlotEntries-metodille.

```
getScheduleSlotEntries(filter: ScheduleSlotEntryFilter): Observable<ScheduleSlotEntry[]> {
  const resourceUuid = filter.resourceUuid;
  return this.http.get<ScheduleSlotEntry[]>(`/schedule/slot/entry/${resourceUuid}`, {
    params: buildQueryParams({from: filter.startDate.toISOString(false), to: filter.endDate.toISOString(false)}),
  });
}
```

Kuvio 49. getScheduleSlotEntries-metodi

Schedule-service vastaanottaa aikavälin. Kuviossa 49 nähdään, että getScheduleSlotEntries ottaa tuloarvoista myös käyttäjän henkilökohtaisen ”resourceUuid” tunnisteen. Tunnistetta käytetään, jotta palvelu osaa hakea vain tietyn henkilön tapahtumia. Kuviossa 49 myös nähdään, että funktio rakentaa hakuparametrit backend-palvelua varten. Parametreihin tulee käyttäjän valitseman kuukauden alku- sekä loppupäivämäärä. Lopuksi tehdään http.get-kutsu backendin-palveluun, josta tulevat tiedot palautetaan takaisin NgRx-effectiin. Kuviossa 48 nähdään, että jos palvelu palautuu onnistuneesti, niin kutsutaan uutta NgRx-Actionia nimeltä ”LoadDatePickerSlotEntriesSuccess”

```
331   export class LoadDatePickerSlotEntriesSuccess implements Action {
332     readonly type = ScheduleActionType.LOAD_DATE_PICKER_SLOT_ENTRIES_SUCCESS;
333
334     constructor(public payload: {
335       filter: ScheduleSlotEntryFilter,
336       response: ScheduleSlotEntry[]
337     }) {
338     }
339   }
```

Kuvio 50. LoadDAtePickerSlotEntriesSuccess Action

LoadDatePickerSlotEntriesSuccess-Actionin payloadissa kulkee mukana parametrit “Filter” ja “response”. Filter sisältää tiedon hakuun käytetyt päivämäärät. Response sisältää backend-palvelusta vastaanotetut tapahtumatiedot.

```
149   case ScheduleActionType.LOAD_DATE_PICKER_SLOT_ENTRIES_SUCCESS: {
150     const loadedDatePickerRange = {
151       start: moment(action.payload.filter.startDate).format(),
152       end: moment(action.payload.filter.endDate).format(),
153     };
154     return {
155       ...state,
156       loadedDatePickerRange,
157       datePickerSlotEntries: action.payload.response.map(scheduleSlotEntry =>
158         new ScheduleSlotEntry(scheduleSlotEntry.appointment, scheduleSlotEntry.timeSlots,
159           scheduleSlotEntry.patients, scheduleSlotEntry.locations,
160           scheduleSlotEntry.organization, scheduleSlotEntry.encounter
161         )
162       ),
163     };
164   }
```

Kuvio 51. Switch case-toiminto

NgRx suorittaa valitun switch case-toiminnon, kun LOAD_DATE_PICKER_SLOT_-ENTRIES_SUCCESS-Actionia kursutaan. Switch case-toiminto ottaa ”payload”-parametristä valitut alku- sekä loppupäivämäärät, ja asettaa ne muuttujaan nimeltä ”loadedDatePickerRange”. Sen lisäksi toimito käy läpi kaikki ”response”-parametrin tapahtumat ”map()”-toiminnolla ja lisää ne muuttujaan nimeltä ”datePickerSlotEntries”. Molemmat ”loadedDatePickerRange” ja ”datePickerSlotEntries” liitetään NgRx-Storen tilaan niin, että ne korvaavat vanhat tiedot. Nyt Storessa on tallennettuna oikea valittu aikavälitieto sekä kyseiselle aikavälillä kaikki tapahtumat.

```
<emr-user-calendar-sidebar
  [activeDate]="selectedDate$ | async"
  (selectCalendarDate)="onSelectNewDate($event)"
  [dpEventDays]="datePickerSlotEvents$ | async"
>
</emr-user-calendar-sidebar>
```

Kuvio 52. Sivupalkin-sapluuna

Tapahtumat saadaan date-picker-komponentin sisältämään side-bar-komponenttiin tekemällä valinta NgRx-Storeen. Valinnan palauttamat tapahtumat sidotaan ”datePickerSlotEvents”-muuttujaan. Muuttajan jälkeen laitetaan vielä kuvion 52 mukaan ”| async”, joka tekee valinnan toiminnasta asynkronisen. Näin saadaan muuttuja tilaamaan Store valinta, joten se aina muuttaa sisältöään, jos Storeen tulee uutta dataa (Angular 2019).

```
export const getDatePickerScheduleSlotEvents = createSelector(
  getScheduleState, scheduleSelectors.getDatePickerScheduleSlotEvents
);
```

Kuvio 53. Valitsin

Kuviossa 53 on valitsin, mitä komponentti käyttää valitsemaan NgRx-Storesta oikeat tiedot. Valitsin kutsuu toista valitsinta nimeltä ”getDatePickerScheduleSlotEvents”.

```
80  export const getDatePickerScheduleSlotEvents = createSelector(
81    getDatePickerRange,
82    getDatePickerSlotEntries,
83    (range, slots): CalendarEvent[] => {
84      let prevDate = null;
85      const indicatedDays = [];
86      slots.filter(ScheduleUtil.filterSlotsByRange(range)).map(slot => {
87        if (!moment(slot.getStartDate()).isSame(prevDate, 'day')) {
88          prevDate = slot.getStartDate();
89          indicatedDays.push({
90            start: new Date(slot.getStartDate()),
91            end: new Date(slot.getEndDate()),
92            title: null,
93          });
94        }
95      });
96      return indicatedDays;
97    }
98  );
```

Kuvio 54. Määränvähennysvalitsin

Kuvion 54 valitsin tekee muutaman asian. Se valitsee Storesta kaikki tapahtumat, jotka käyttäjällä on aikataulussa valitulla aikavälillä. Se myös vähentää lähetettävien tapahtumien määrää niin, että vain yksi tapahtuma lähetetään yhtä päivää kohden.

Rivillä 86 nähdään ”map”-metodi, joka käy yksitellen läpi kaikki tapahtumadataryhmän jäsenet. Metodin callback-toiminto vertaa käsiteltävää päivää edelliseen käsiteltyyn päivämäärään. Jos tapahtuma on samalla päivällä kuin edellinen, niin tapahtumalle ei tehdä mitään. Jos päivämäärä on eri niin siitä otetaan alku- ja loppupäivämäärä ylös ja lisätään uuteen tapahtumadataryhmään nimeltä ”indicatedDays”. Lopuksi ”indicatedDays” palautetaan takaisin NgRx-Storeen, mistä se lähetetään date-picker-komponenttiin.

Tämä data määrän vähennys tehdään, jotta se ei hidastaisi käyttöliittymän käyttöä liikaa ylimääräisellä datalla. Esimerkiksi hoitohenkilökunnan jäsenellä voi olla useita satoja tapahtumia viikossa, mutta vain maksimissaan 31 päivää, jota näytetään date-picker komponentissa. On tehokkaampaa tallentaa vain yksi tapahtuma per päivä verrattuna useampaan.

6.3.2 Yhteenveto tehtävästä 2

Tehtävässä 2 on esimerkki käyttöliittymävisiosta, joka vaikuttaa yksinkertaiselta toteuttaa, mutta todellisuus on toisin. NgRx-Storen huono puoli tulee myös esiin. Store on kätevä ja toimiva, kunnes sitä pitää alkaa muokkaamaan jotain tämän tyyppistä varten. Moni Storen toiminto on riippuvainen muista toiminnoista. Jos halutaan lisätä toinen tapahtumat-data-ryhmä Storen tilaan, joka ei ole kytköksissä kalenterinäkymään, niin tarvitsee kirjoittaa kokonaan uusi toimintaketju, jotta se saadaan aikaan.

Tehtävän 2 toteutus on toimiva, mutta siinä on vielä mahdollisuuksia jatkokehitystä varten. Koska kyseessä on iso web-pohjainen sovellus, joten kaikki mahdollinen odotus- ja latausaika pitää minimoida.

Tätä ratkaisua olisi tulevaisuudessa mahdollista optimoida niin, että date-pickerin NgRx-tapahtumaketju ei missään välissä ota vastaan koko kuukauden tapahtumia. Palvelimella toimivaa backend-palvelua tulisi muokata niin, että siihen lisätään uusi toiminto pelkästään date-pickeriä varten. Palvelu palauttaisi ainoastaan päivämäärät, joilla on tapahtumia. Niin saadaan minimoitua palvelun ja verkkosovelluksen välinen datansiirto. Samalla poistetaan selaimelta prosessointi kuormaa, koska ei tarvitse enää Kuvion 54 mukaista toimintoa erittelemään päivämääriä.

7 YHTEENVETO

Tämän opinnäytetyön tavoitteena oli tehdä jatkokehitystä toimeksiantoyrityksen potilastietojärjestelmäprojektin kalenterikomponenttiin. Työ toteutettiin yhteistyössä toimeksiantoyritys Nortal Oy:n kanssa.

Työssä kuvailtiin Angularin toimintaperiaatteita sekä läpikäytiin Angularin komentorivityökalun käyttöperiaatteita. Angularin toimintaperiaatteita esiteltäessä käytettiin esimerkkisollusta selkeyttämään asioiden ymmärtämistä. Lisäksi työssä havainnollistettiin tarkemmin NgRx ohjelmistokirjaston toimintaa.

Opinnäytetyön jatkokehitys osassa esiteltiin toimeksiantoyrityksen tarjoamia työtehtäviä. Työtehtävistä kerrottiin tavoite mihin pyritään, toteutustapa vaiheittain sekä yhteenveto.

Työtehtävä 1 toteutettiin ilman ongelmia. Tehtävän lopputulos oli yksinkertainen ja toimiva ratkaisu, joka tekee tehtävänsä kuormittamatta järjestelmää liian paljoa.

Työtehtävän 2 ratkaisun löytämisessä oli ongelmia. NgRx kirjaston monimutkaisuuden takia työn tekemisessä meni huomattavia määriä aikaa kirjaston toiminnan tutkimiseen. Lopulta saatiin tehtyä ratkaisu, joka toimii, mutta jatkokehitys on tarpeellista. Jatkokehitys tarpeita on käytettävän datan määrän optimointi. Ratkaisussa järjestelmä pyytää enemmän dataa, kun se oikeasti tarvitsee.

Jos opinnäytetyötä aloitettaisiin nyt kirjoittamaan uudelleen, niin asia, mikä huomioitaisiin aikaisemmin, on NgRx-Redux-kirjasto. NgRx-kirjaston toimintaperiaatteisiin tultaisiin keskittymään tarkemmin. Kirjasto on hyvin monimutkainen, joten sen ymmärtämiseen menee pitkään, jos yrittää opetella sen käyttöä vain lukemalla olemassa olevaa koodia.

Potilastietojärjestelmäprojekti on monen eri osa-alueen kokonaisuus. Tämä työ on vain pieni osa potilastietojärjestelmän kalenterikomponentin toimintaa. Potilastietojärjestelmän kehitys jatkuu toimeksiantoyrityksen toimesta.

LÄHTEET

Angular 2019. Architecture [viitattu 2.5.2019]. Saatavissa:
https://angular.io/guide/architecture

Angular 2019. Async Pipe [viitattu 30.10.2019]. Saatavissa:
https://angular.io/api/common/AsyncPipe

Angular 2019. Component [viitattu 2.12.2019]. Saatavissa:
https://angular.io/api/core/Component

Angular 2019. Http [viitattu 2.12.2019]. Saatavissa:
https://angular.io/guide/http

Sitepoint 2018. Angular Introduction [viitattu 2.5.2019]. Saatavissa:
https://www.sitepoint.com/angular-introduction/

Angular 2019. Template [viitattu 2.12.2019]. Saatavissa:
https://angular.io/guide/displaying-data

Asiakastieto 2019. Nortal Oy [viitattu 16.11.2019]. Saatavissa:
https://www.asiakastieto.fi/yritykset/fi/nortal-oy/07281351/taloustiedot

FontAwesome 2019. [viitattu 2.5.2019]. Saatavissa:
https://fontawesome.com/

Fullcalendar 2019 [viitattu 2.5.2019]. Saatavissa:
https://fullcalendar.io/

Github 2019. Todo [viitattu 2.10.2019]. Saatavissa:
https://github.com/Alreh/todo

History of Angular 2018. The history of Angular [viitattu 2.5.2019]. Saatavissa:
https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7

Nortal 2019. About us [viitattu 2.5.2019]. Saatavissa:
https://nortal.com/about-us/

Nortal 2019. eHealth [viitattu 2.5.2019]. Saatavissa:
https://nortal.com/what-we-do/digital-healthcare/

Gitconnect 2019. Redux flow simple summary [viitattu 2.5.2019]. Saatavissa:
https://levelup.gitconnected.com/redux-logic-flow-crazy-simple-summary-35416eadabd8