OHSIHA 2020, Harjoitustyön vaihe 3

Arttu Saarinen
arttu.saarinen@tuni.fi

GitHub-linkki projektiin: https://github.com/saarinea/markets
Heroku-linkki projektiin: https://tranquil-hollows-95262.herokuapp.com/

Harjoitustyön tarkoituksena on toteuttaa verkkopalvelu, joka hakee rajapinnoista erilaista finanssimarkkinadataa (mm. osakekursseja, pörssi-indeksejä, valuuttakursseja) ja koostaa niistä eräänlainen dashboard omiin informaatiotarpeisiini. Teknologioiksi valitsin alla listatut työkalut, koska olen Web-ohjelmoinnin kurssin käynyt ja halusin oppia uutta sekä uuden kielen (Javascript) että muuten uusien työkalujen avulla.

Käytetyt teknologiat

- Frontend: React
- Backend: Node.js
- Tietokanta: Atlas MongoDB
- Versionhallinta: Github
- Editori: Visual Studio Code
- Serveri/sovellusplatform: Heroku
- Hyödynnettyjä kirjastoja: Passport, Mongoose, Nodemon, Bootstrap, Axios, Cors, Redux
- Hyödynnettyjä kolmannen osapuolen rajapintoja: AlphaVantage

API-kutsun vaihto frontendistä suoraan API:in menevästä kutsusta Reduxiin

Tavoitetilanani oli saada sivulle dropdown-valikko, josta valitaan minkä osakkeen tiedot halutaan hakea ja päivittää alla näkyvät taulukot valintaa vastaavasti. Olin jo aiemmassa vaiheessa suorittanut API-kutsun AlphaVantage-palveluun suoraan allaolevalla demo-API-kutsulla frontendin React-komponentissa, mikä ei luonnollisesti ole pitkäaikainen ratkaisu.

```
componentDidM
ount() {

        fetch('https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol
        =MSFT&interval=5min&apikey=demo')
            .then(res => res.json())
            .then((data) => {
              this.setState({
                metadata: data["Meta Data"],
                timeseries: data["Time Series (5min)"],
                isLoaded: true
              })
            },
```

Tämän jälkeen toteutin kunnollisen API-kutsun seuraavien välivaiheiden kautta:

Välivaihe 1: Haetaan tiedot taulukoivassa React-komponentissa .fetch-kutsulla backendistä ja näytetään aina ainoastaan tietyn osakkeen tiedot ts. näytetään periaatteessa staattista dataa. Tässä vaiheessa keskityin backendin express-serverissä tapahtuvaan tiedonhakuun API:sta. Alla vielä välivaihe 2:ssa sama backendin kutsu API:in tarkemmin avattuna.

Välivaihe 2: Tehdään toinen React-komponentti (dropdown-valikko) ja Reduxia hyödyntäen päivitetään sen avulla dynaamisesti taulukkokomponentti. Mikäli Reduxin peruskäsitteet eivät ole tuttuja, ne kannattaa lukea alimpana olevan lähdelistan ensimmäisestä Redux-otsikon alla olevasta linkistä. Tässä tapauksessa tieto valuu sivulle seuraavien askelien mukaan:

1. **Valitse nappulasta haluamasi osake**
2. **Valikko käynnistää action creatorin, joka luo Reduxiin käskyn suorittaa GET_DATA-tyyppisen actionin, joka puolestaan sisältää dataa hakevan funktion**

```
<a
            className="dropdown-item"
            href="#"
            onClick={() => this.click('Facebook (FB)', 'FB')}
          >
            Facebook (FB)
          </a>
```

```
  click = (text, ticker) => {
    this.toggleOpen()
    this.setMenutext(text)
    this.props.updateData(ticker)
  }
```

```
function mapDispatchToProps(dispatch) {
  return {
    updateData: ticker => dispatch(getData(ticker))
  }
}

export default connect(null, mapDispatchToProps)(StockDropdown)
```

3. **Hae action creatorissa data backendistä käyttäen axios-kirjastoa**

```
export const getData = (ticker) => async dispatch => {
  const res = await axios.get('/data/stocks',
  {params: {
     ticker: ticker
  }})
  dispatch({ type: GET_DATA, payload: res.data })
}
```

4. **Backend kuuntelee ko. kutsua ja palauttaa dataa frontendiin**

```
module.exports = app => {
  app.get('/data/stocks', async (req, res) => {
```

```
    try {
      const result = await axios.get(
        'https://www.alphavantage.co/query',
        {params:{
          function:"TIME_SERIES_DAILY",
          symbol: req.query.ticker,
          apikey: keys.AlphaVantageKey
        }}
      )
      res.send(result.data)
    } catch (error) {
      console.log(error)
    }
  })
}
```

5. **Frontendissä redux-storeen lähetetään action, joka reducerissa käsitellään ts. asetetaan storeen haettu data**

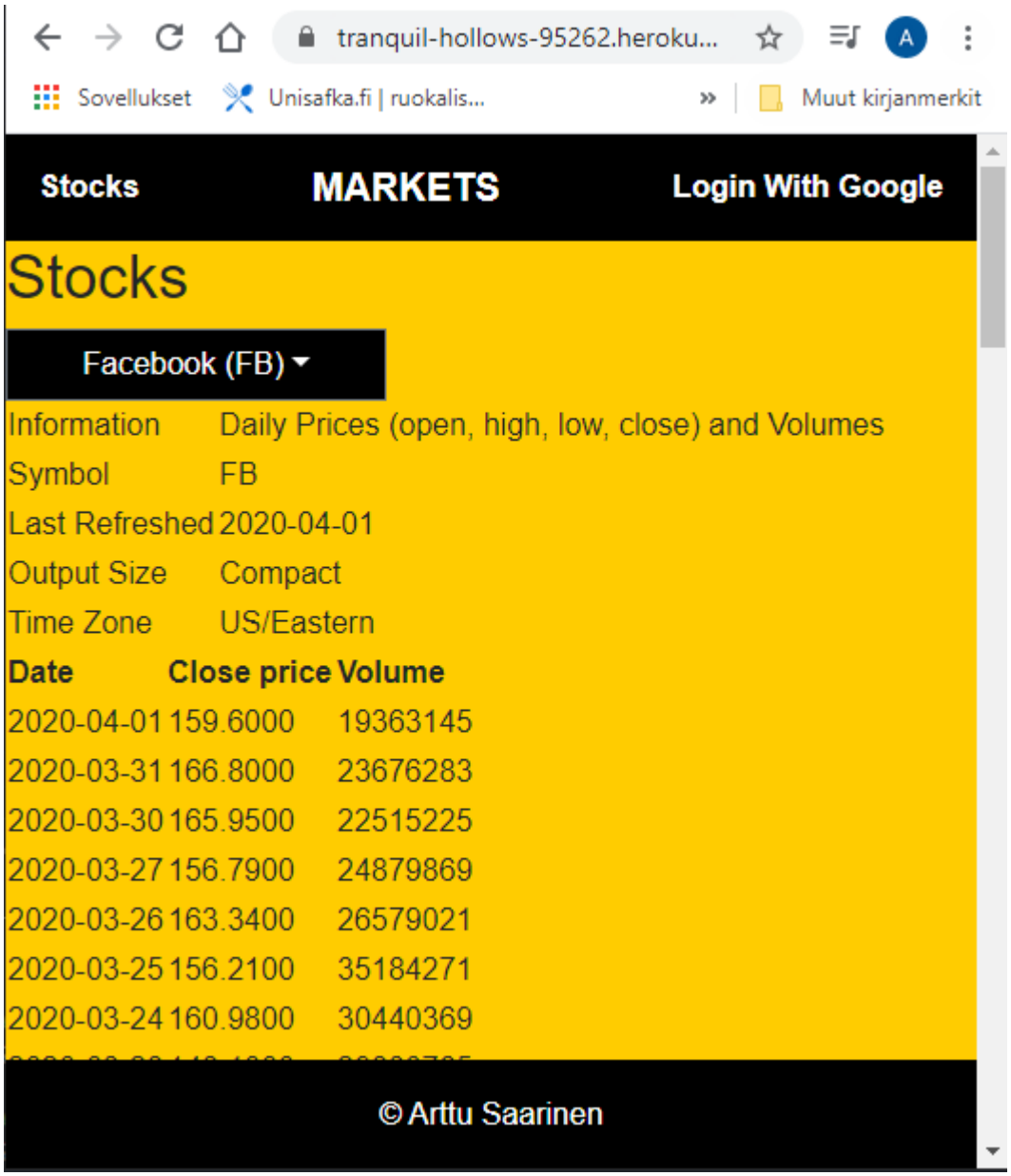
```
export default function(state = initialState, action) {
  switch (action.type) {
    case GET_DATA:
      return Object.assign({}, state, {
        data: action.payload,
        isLoaded: true
      })
    default:
      return state
  }
}
```

6. **Frontendin dataa listaava komponentti on kytketty redux-storeen, ja se päivittyy automaattisesti redux-storen datasisällön päivittyessä**

```
function mapStateToProps(state) {
  return { data: state.stock.data, isLoaded: state.stock.isLoaded }
}

export default connect(mapStateToProps, null)(SingleStock)
```

Tuloksena alla oleva näkymä:



Muut kehityksen alla olleet asiat

- Laitoin projektin Herokuun pyörimään, osoite yllä ja tässä: https://tranquil-hollows-95262.herokuapp.com/. Tuota operaatiota varten piti tehdä muutamat konfiguroinnit, suurimpana package.jsoniin piti lisätä skripti herokua varten sekä expressin index.js:ssä piti lisätä if-lause, jossa ympäristön ollessa tuotantoympäristö urlien käsitteleminen on hiukan erilaista.
- Lisäksi kehittelin ulkoasua hiukan visuaalisemmaksi, vaikkei se vieläkään kovin kehuttava ole

Haasteet ja helpot asiat

1. **Haaste: CORS:** lähettäessä dataa/ kyselyjä frontendistä backendiin, tuli esille virhe: serveri ei hyväksynyt pyyntöjä jotka tulivat eri palvelimelta. Tämä tapahtuu siksi, että create-react-appin luoma sovellus pyörii erillisessä serverissä kehitysympäristössä (react-frontend: localhost:3000, express-backend: localhost:5000). Sen sai korjattua asentamalla cors-kirjaston ja lisäämällä sen toimimaan kehitysympäristössä
2. **Haaste: Debuggaus chromella ja OAuth:** Käytettäessä oauthia eli Googlen kirjautumista, tuli erikoinen ongelma vastaan. Käytettäessä VS Coden debug for chrome-lisäosaa käynnistettäessä

debuggaus avaa VS Code uuden instanssin chromesta. Tämä ko. Chromen instanssi ei kuitenkaan anna käyttäjän kirjautua sisään, vaan kun Googlen kirjautumisikkunassa täyttää tietonsa ja painaa ”Kirjaudu”, sanoo sivu että ”käyttämäsi selain ei ole luotettava”. Tämä johtuu siitä, että chromen instanssi sisältää tiedon siitä, että se on debuggerin käynnistämä instanssi, ja tällaisia instansseja ei kohdella luotettavana Googlen OAuth-kirjautumisessa. Tähän ongelmaan ei oikeastaan ole vastausta, eli jouduin tekemään niin että otin kirjautumispakon pois päältä debugatessa kohdissa joissa se oli päällä oletuksena.

3. **Haaste: Debuggaus yleisesti:** Debuggaus yleisesti on melko paljon monimutkaisempaa web-sovelluksissa verrattuna esim. C++-sovelluksiin verrattuna, esim. asynkroniset tietopyynnöt ja ylipäänsä paljon monimutkaisempi tiedostorakenne tekee asioista hankalanmpia
4. **Haaste:** CSS:n ja bootstrapin ym. visuaalinen konffaaminen aikaavievää ilman parempaa rutiinia, esimerkiksi kun tein navigointipalkkia sekä footer-palkkia sivulleni meni niihin huomattavasti enemmän aikaa kun mitä voisi olettaa ottaen huomioon että kyseessä on mustat palkit parilla linkillä varustettuna
5. **Haaste: Reduxin ja redux-thunkin ja niiden yhteistoiminta Reactin kanssa:** konffaus melkoinen haaste, redux ja redux-thunk todella suuri tietopaketti yhdellä kertaa sisäistettäväksi ja opeteltavaksi
6. **Helppo asia:** sitten kun reactin, reduxin ja redux-thunkin tutoriaaleja saa kahlattua läpi ja yhden käytännön esimerkin toimimaan, paljastuu kuinka helposti sovelluksen ja komponenttien tiloja voi muutella ja kuinka käteviä työkaluja ne ovatkaan tietosisällön dynaamiseen päivittämiseen

Hyödyllisiä lähteitä

Yleisesti Reactista:

- https://alligator.io/react/constructors-with-react-components/
- https://reactjs.org/docs/handling-events.html
- https://redux.js.org/basics/usage-with-react
- https://www.robinwieruch.de/react-pass-props-to-component

Redux, reactin kytkeminen Reduxiin, Redux-thunk:

- https://www.valentinog.com/blog/redux/
- https://stackoverflow.com/questions/53467778/react-redux-launch-state-change-of-another-component
- https://alligator.io/redux/redux-thunk/