OHSIHA 2020, Harjoitustyön vaihe 4

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.

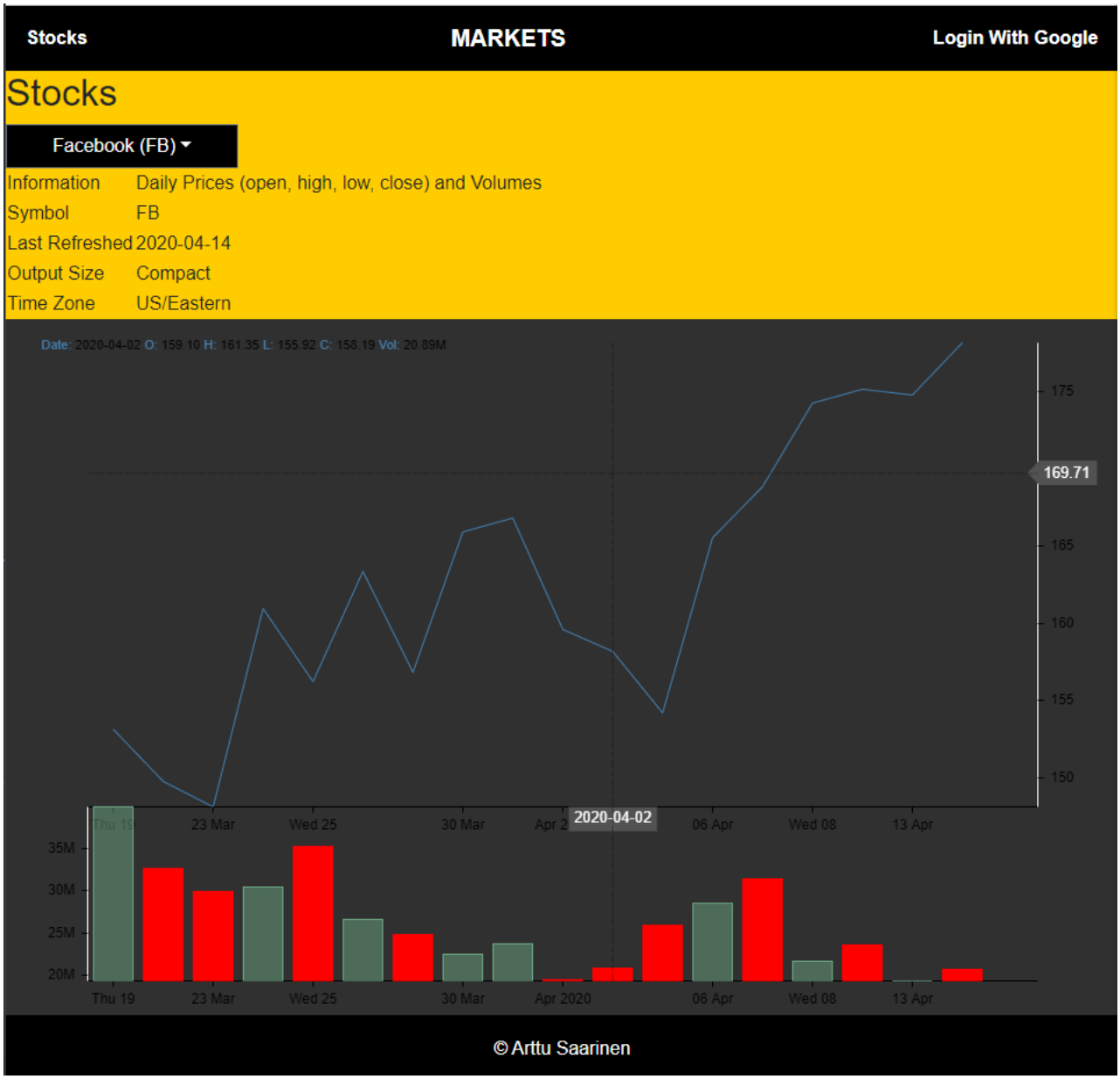
Käytetyt teknologiat

Etenkin tässä vaiheessa käytetyt teknologiat **boldattuna.**

- 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, **react-stockcharts, d3.js**
- Hyödynnettyjä kolmannen osapuolen rajapintoja: AlphaVantage

Osakedatan visualisointi

Käytin react-stockcharts-kirjastoa luomaan osakedatan kuvaajan, jossa olisi hintakehityksen lisäksi palkkikaavio volyymistä sekä kursorin paikan mukaan päivittyvä datapisteselite (jossa näkyy avaus- ja lopetushinnat, korkein sekä alin hinta, päivämäärä ja volyymi. Lisäksi kuvaajaa voi liikutella sivuttain sekä zoomata. Screenshot kuvaajasta alla:



Kuvaaja toteutettiin react-komponenttina, jonka propertyt yhdistettiin komponentin stateen. Tämän jälkeen komponentti päivittyy aina datan päivittyessä. Alla ote koodista:

```
class StockLineChart2 extends Component {
  constructor(props) {
    super(props)
    this.state = {
      crosshairValues: [],
      data: {},
    }
  }

  renderTimeSeries() {
    var timeseries = this.props.data['Time Series (Daily)']
    var timeSeriesArray = []
```

```
  for (var i in timeseries) {
    var open = parseFloat(timeseries[i]['1. open'])
    var high = parseFloat(timeseries[i]['2. high'])
    var low = parseFloat(timeseries[i]['3. low'])
    var close = parseFloat(timeseries[i]['4. close'])
    var volume = parseFloat(timeseries[i]['5. volume'])
    var date = new Date(i)
    timeSeriesArray.unshift({
      date: date,
      open: open,
      close: close,
      high: high,
      low: low,
      volume: volume,
    })
  }
  return timeSeriesArray
}
render() {
  const initialData = this.renderTimeSeries()
  const type = 'hybrid'

  const { width, ratio } = this.props
  const xScaleProvider = discontinuousTimeScaleProvider.inputDateAccessor(
    (d) => d.date
  )
  const { data, xScale, xAccessor, displayXAccessor } = xScaleProvider(
    initialData
  )
  const xExtents = [xAccessor(last(data)), xAccessor(data[data.length - 20])]

  return (
    <ChartCanvas
      ratio={ratio}
      width={width}
      height={600}
      margin={{ left: 70, right: 70, top: 20, bottom: 30 }}
      type={type}
      pointsPerPxThreshold={2}
      seriesName="AAPL"
      data={data}
      xAccessor={xAccessor}
      displayXAccessor={displayXAccessor}
      xScale={xScale}
      xExtents={xExtents}
    >
      <Chart id={1} height ={400} yExtents={(d) => [d.close]}>
        <XAxis axisAt="bottom" orient="bottom" />
        <YAxis
          axisAt="right"
```

```
            orient="right"
            // tickInterval={5}
            // tickValues={[40, 60]}
            ticks={5}
          />
          <MouseCoordinateX
            at="bottom"
            orient="bottom"
            displayFormat={timeFormat('%Y-%m-%d')}
          />
          <MouseCoordinateY
            at="right"
            orient="right"
            displayFormat={format('.2f')}
          />
          <LineSeries yAccessor={(d) => d.close} />

          <OHLCTooltip forChart={1} origin={[-40, 0]} />
          <CrossHairCursor />
        </Chart>

        <Chart id={2} origin={(w, h) => [0, h - 150]} height={150} yExtents={d => d.vol
ume}>
          <XAxis axisAt="bottom" orient="bottom"/>
          <YAxis axisAt="left" orient="left" ticks={5} tickFormat={format(".2s")}/>
          <BarSeries yAccessor={d => d.volume} fill={(d) => d.close > d.open ? "#6BA583
" : "red"} />
        </Chart>
      </ChartCanvas>
    )
  }
}
```

Haasteet ja helpot asiat

1. **Haaste:** Erilaisia kirjastoja graafien tekemiseen reactin avulla on paljon, ja niiden valinta oli hankalaa. Luotasin useita läpi (d3.js, react-vis, victory. reCharts) ja jopa yritin implementoida niitä, mutta huonolla menestyksellä ennen kuin päätin ottaa käyttöön suoraan d3.js:n pohjalle tehdyn react-stockchartsin. Edellä mainitut eivät joko spekseiltään olleet tarpeeksi kehittyneitä (react-vis, victory. reCharts) tai olivat liian kehittyneitä ts. vaikeita ja nollasta lähteviä (d3.js)
2. **Haaste:** React-stockcharts on varsin kehitysvaiheessa oleva kirjasto, ja sen dokumentaatio on käytännössä olematonta. Ts. jos graafia halusi muokata, piti lukea kirjaston lähdekoodia eikä dokumentaatiota.
3. **Helppo asia:** react-stockchartsiin oli melko helppo syöttää tietoa ja pienillä muokkauksilla tehdä omanlaisensa graafi

Hyödyllisiä lähteitä

React-stockchartsin dokumentaatio (ei paljoa apua, tein suurimman osan devauksesta lukemalla kirjaston lähdekoodia ja kokeilemalla valmiita esimerkkejä codesandboxissa: http://rrag.github.io/react-stockcharts/documentation.html#/overview

d3.js ja React: https://wattenberger.com/blog/react-and-d3