SPA-toteutus

Kuvaus toteutuksesta

Toisen vaiheen alussa tein suuren päätöksen ja päätin toteuttaa harjoitustyön käyttäen Reactia. Tämä päätös oli osittain lisäpisteen toivossa ja osittain oman taidon kasvattamiseksi, sillä olen aikaisemmin käyttänyt Djangoa, ja React on kiinnostanut viime aikoina yhä enemmän.

React on Single Page Application -periaatteella toimiva, sillä se vain päivittää samaa selaimessa näkyvää dokumenttia. Ohjelmassani on erillisiä ”sivuja” mutta selain ei oikeasti vaihda sivua, tieto vain välitetään palvelimelle.

Otteita toteutuksesta

```
import React from 'react'

const Posts = ({ posts }) => {
  return (
    <div>
      {posts.map((post) => (
        <div class="post">
          <div class="post-heading">
            <div class=" header-text">{post.title}</div>
            <div class="date">Id: {post.id}</div>
            </div>
          <div class="post-text">{post.body}</div>
        </div>
      ))}
    </div>
  )
};

export default Posts
```

Ote posts-moduulista. Tämä on siis yksi julkaisu. Reactin yksi hienous on sen modulaarisuus, eli samoja komponentteja voi ja kuuluu käyttää uudelleen.

3 helppoa/hankalaa tekijää

1. React oli itselle kokonaan uutta asiaa, joten asioiden oppimiseen meni huomattavasti aikaa
2. Käyttötarpeeni Reactin kanssa olivat hyvin spesifejä, joten hyvien ohjeiden löytymien oli hankalaa.
3. Olen positiivisesti kuitenkin yllättynyt reactin tehokkuudesta ja mukavuudesta.

Hyödyllisiä linkkejä

- https://www.w3schools.com/REACT/default.asp
- https://medium.com/@olinations/build-a-crud-template-using-react-bootstrap-express-postgres-9f84cc444438
- https://pusher.com/tutorials/consume-restful-api-react
- https://www.freecodecamp.org/news/fullstack-react-blog-app-with-express-and-psql/
- https://medium.com/@Elijah_Meeks/interactive-applications-with-react-d3-f76f7b3ebc71
- https://blog.pusher.com/getting-started-with-react-router-v4/

Koodit ja dokumentaatio GitHubiin

Kuvaus toteutuksesta

Tietotekniikan opiskelijana Git on itselle jo hyvin tuttu asia. Heti projektin alussa loin paikallisen repositorion komennolla ”git init”. Jokainen vaihe on lisätty omana committinaan, joka näin jälkikäteen ajatellen ei ole paras vaihtoehto, vaan olisi kannattanut tehdä pienempiä committeja. Eri palautukset olisi voinut erotella, vaikka tageilla.

Kolmannen vaiheen alussa kokeilin uusia ominaisuuksia, jotka osoittautuivat huonoiksi, joten oli hyvin helppo palata versionhallinnan kanssa takaisinpäin. Lisäksi myös projekti on julkisesti pyörimässä omalla kotiserverillä, johon se oli helppo siirtää gitin avulla, sekä saada lisätyt muutokset nopeasti sinne.

Hieman huolimattomasti menin lisäämään tietokannan salasanan repoon, jonka kuitenkin sain poistettua käyttämällä git rebase -toimintoa. Linkki repositorioon alla.

Lisäpiste: Avoin lisenssi

Koodi on julkaistu avoimella lisenssillä.

Otteita toteutuksesta

```
C:\Users\Veikko\ohjelmointi\ohsiha>git log
commit 016e3917866140cd9ea93e3a6aa9e54a0525e24a
Author: Veikko Nieminen <veikko.nieminen@tuni.fi>
Date:   Thu Apr 16 13:19:15 2020 +0300

    nelj<C3><A4>s palautus

commit 515d0c142b7ec1ac3039139c4571244e991e867c
Author: Veikko Nieminen <veikko.nieminen@tuni.fi>
Date:   Fri Apr 3 14:32:35 2020 +0300

    vaihe 3 osa 2

commit 254bda4d0fbd7c5f1f6c257e178b7015ecaacf60
Author: Veikko Nieminen <veikko.nieminen@tuni.fi>
Date:   Fri Apr 3 09:48:34 2020 +0300

    vaihe 3 osa 1

commit cf73a3154756bf3ac3e30e32740aaa6d722dbd48
Author: Veikko Nieminen <veikko.nieminen@tuni.fi>
Date:   Wed Apr 1 10:44:24 2020 +0300

    toinen vaihe

commit 4d59cbe75b0a96902e13f35f09136b93562c8738
Author: Veikko Nieminen <veikko.nieminen@tuni.fi>
Date:   Fri Feb 21 15:43:48 2020 +0200

    eka vaihe
```

Kuvassa eri vaiheiden committeja. Loppuvaiheessa committien määrä tuplaantui kun pieniä muutoksia tapahtui paljon.

3 helppoa/hankalaa tekijää

1. Gittiä on kouluprojekteissa käytetty jo sen verran paljon, että sen kanssa ei juurikaan ollut ongelmia.
2. Jälkikäteen ajatellen olisi voinut käyttää myös tageja eri palautusversioiden ”julkaisemiseen”.
3. Jälkikäteen huomasin lisänneeni tiedostoja, jotka sisältävät tietokannan salasanan. Ongelma kuitenkin hoitui git rebase toiminnolla

Hyödyllisiä linkkejä

- https://github.com/veikkoni/ohsiha
- https://git-scm.com/
- https://stackoverflow.com/questions/29741476/how-do-i-properly-remove-sensitive-data-pushed-to-a-git-repo/29741586

HTML5-pohjainen käyttöliittymä

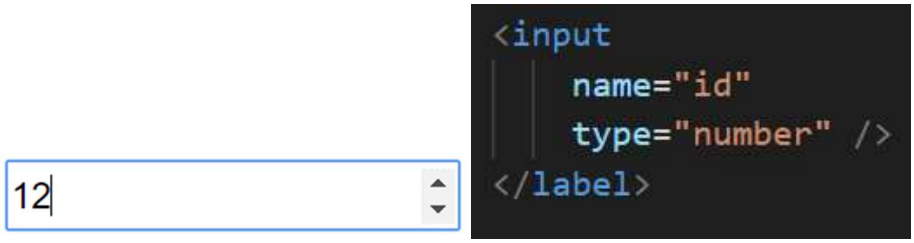
Kuvaus toteutuksesta

Huomaamattani satuin käyttämään projektissa kahta eri HTML5 ominaisuutta.

Käyttäjätunnuksen tallentaminen käyttöliittymän puolella osoittautui hankalaksi, sillä en käyttänyt projektissani tilanhallinnan kirjastoa (esim Redux). Asiaa tutkimalla löysin kuitenkin localStorage objektin, jota kehotettiin käyttämään Reactin kanssa. LocalStorage on WebStoragen yksi ominaisuus.

Toinen ominaisuus oli julkaisujen poistossa käytettävä syöttökohta. Käytin tässä number tyyppistä syötettä, johon voi syöttää vain numeroita. Tämä on sikäli kätevää, ettei erillistä numerollisuuden tarkistusta tarvita. Tähän käyttötapaukseen tosin negatiivisten numeroiden syöttäminen on ”turhaan” mahdollista, mutta ei aiheuta ongelmia.

Otteita toteutuksesta



Vasemmalla numerovalinta käyttöliittymässä ja oikealla koodissa

```
handleInputChange(event) {
  this.setState({
    [event.target.name]: event.target.value
  });
  localStorage.setItem('username', event.target.value)
}
```

```
localStorage.getItem('username'))
```

Vasemmalla tallennus ja oikealla tiedot noutaminen

3 helppoa/hankalaa tekijää

- LocalStoragen kanssa oli hankaluuksia jotka loppujen lopuksi johtuivat reactista (tai osaamattomuudesta)
- Numerosyötteen käyttö oli hyvin yksinkertaista.
- LocalStoragen käyttö on myös kaiken kaikkiaan yksinkertaista.

Hyödyllisiä linkkejä

- https://www.w3.org/TR/webstorage/
- https://www.robinwieruch.de/local-storage-react

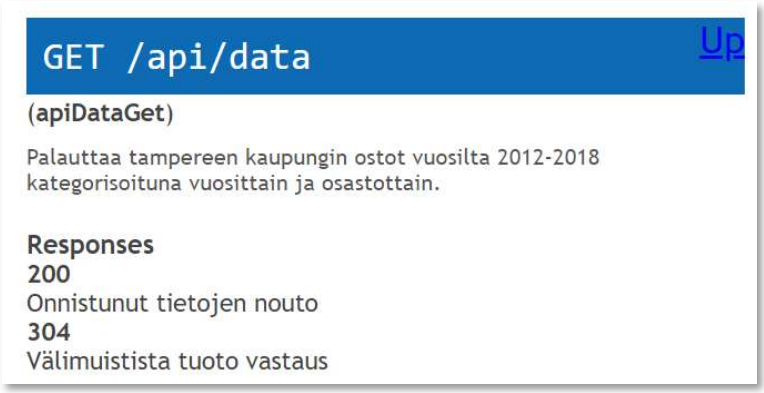
Sovelluskehittäjän rajapinta

Kuvaus toteutuksesta

Käyttöliittymä on toteutettu Reactilla, joten se vaatii erillisen taustaserverin toimiakseen datan kanssa. Tällä tavoin sovelluskehittäjän rajapinta tuli myös samalla toteutettua (Tämä on myös yleinen tapa jakaa sovellus eri osiin). Toteutin rajapinnan käyttämällä express ja express-router kirjastoja. Tässä jokaiseen tarvittavaan osoitteeseen (esim. /api/data) luotiin oma logiikka, josta kuva alempana.

Loin rajapinnasta dokumentaation käyttäen Swagger-palvelua. Tämän palvelun avulla dokumentaatio ”loi itse itsenstä” ja vaati vain pieniä muutoksia. Dokumentaatio on nähtävillä projektin juurikansiossa ja alla olevassa osoitteessa.

Otteita toteutuksesta



Kuva rajapintadokumentaatiosta

```
router.get('/api/post', (req, res, next) => {
  if (req.query.user == null){
    res.status(400).send();
  } else {
    values = [req.query.user]
    pool.query("SELECT * FROM posts WHERE creator = $1", values, (q_err, q_res) => {
      if (q_res == undefined) {
        res.status(400).send();
      } else {
        res.setHeader('Content-Type', 'application/json');
        res.end(JSON.stringify(q_res.rows, null, 3));
      }
    })
  }
});
```

Ote yhdestä rajapinnan toteutuksesta

3 helppoa/hankalaa tekijää

- Datan vieminen rajapinnan kautta osoittautui monimutkaisemmaksi kuin luulin
- Myös dokumentaation luomisessa meni yllättävän kauan, vaikka käytin työkalua
- Rajapintojen luominen oli aikaisemmilta kursseilta tuttua

Hyödyllisiä linkkejä

- http://hoodienkuninkaat.com/ohsiha/api/doc Dokumentaation osoite
- http://hoodienkuninkaat.com:5000/api Rajapinnan osoite
- https://swagger.io/resources/articles/documenting-apis-with-swagger/

Ajax

Kuvaus toteutuksesta

Reactissa yleisesti käytettä tapa hakea tietoa erillisestä palvelusta on ajax pohjainen fetch toiminto. Tätä ominaisuutta käytin käyttöliittymän ja rajapinnan välisessä kommunikoinnissa (Datan hakeminen, julkaisujen hakeminen, poistaminen ja lisääminen. Käyttöliittymä ensin luo näytettävät objektit, ja vasta sitten hakee tiedot ja lisää tiedot näkymään. Tämä ominaisuus oli dokumentoitu hyvin.

Otteita toteutuksesta

```
componentDidMount() {
    fetch('http://hoodienkuninkaat.com:5000/api/posts?user=' + localStorage.getItem('username'))
        .then(res => res.json())
        .then((data) => {
            this.setState({ posts: data })
        })
        .catch(console.log)

}
```

Kuvassa haetaan julkaisut. Samankaltaista hakua käytetään neljässä muussa kohdassa.

3 helppoa/hankalaa tekijää

- Seuraamani ohjeet, ja ilmeisesti myös yleisesti käytäntö, hakivat tietoja tällä tavalla, joten erillistä tutkiskelua ei tarvittu.
- Yhdessä kohdassa oli hankaluuksia saada haettuja tietoja hyödynnettyä.
- Tämäkin ominaisuus osoittautui tehokkaaksi.

Hyödyllisiä linkkejä

- https://pusher.com/tutorials/consume-restful-api-react