Fullstack 2022

**Harjoitustyö**

24.04.2022

TTC2080

Timo Laakkonen AA4598

# 

Sisällysluettelo

[1 Routing 1](#_Toc65210399)

# Johdanto

Koska strava-palvelussa ei ole juurikaan haku ja yhteenveto mahdollisuutta, tehdään se ja onneksi uusin ja olemaasa oleva data voidaan haeka api-rajapinnan kautta ja tellentaa mongodb-tietokantaan. Ei kaikkea dataa vaan tietyty kentät.

Data on mahdollista editoida ei kaikkea vaan commentti kentää

Menu josta valitaan

# Suunnitelma

Pääkäyttäjällä on lisäominaisuuksia/-oikeuksia mm. esim. hakea strvasta data ja ladata ne mongoon, vain tietyt tiedot ei kaikkia kenttiä!!!! Katso Python harjoitus.

Josta aina ladataan data nöäytettäväksi data\_frameen pandas-tyyppisesti

Olisko sittenkin niin että rekisteröidessä talletetaan kunkin käyttäjän stravaid ja hän voisi omia tietojaan hakea, mutta sovellus on minun tililläni käytettävissä serverin kautta

Siis tämä sovellus kekityy minuun, mutta muut rekisteräityneet/kirjautuneet voivat myös päästä katselemna tietojani hakea tietoa mutta data ei ole heidän vaan minun suorituksistani. Jatkokehitykseen niin että kukin voi tsekata omiaan.

Heataan dataa strava apin kautta pandas-ttyyppiseen javascript kirjasto käyttäen eli data-frame. Näytetään graafina (olisko pyhon harjooitusksen ttyyppinen suoraan tehty javascriptille). Tietoja voidaan tallentaa mongoDB:hen ja myös lukea tietyn tallnus otsikon alta. Jottta saadan crud niin käsin syöttö mahdollisuus, mutta mitä järkeä kun stravaan pitäisi päivittää eikä CRUD -mongoon.

Tehdään skema mongoon datagramen pohjalta niin saadaan kenties helpoiten…

MERN2021 sisältää CRUDin

# Danfo,js

Vaatii downgradata uusimmasta versioon 4.0.3

npx create-react-app **frontti** --scripts-version 4.0.3

# Heroku

Sovelluksen luonti Herokuun (tähän sitten sovellus pushataan)

heroku login

heroku create mycycling

<https://mycycling.herokuapp.com>

## API

<https://www.strava.com/settings/api>

Timo Laakkonen

[https://www.strava.com/athletes/**741808**](https://www.strava.com/athletes/741808)

[**https://www.strava.com/api/v3/athletes/741808**](https://www.strava.com/api/v3/athletes/741808)

[https://www.strava.com/api/v3/athletes/741808?access\_token=a0a2d8e07ce5da68e78c4edce87858299cc3379f](https://www.strava.com/api/v3/athletes/741808?access_token=33f24f0fc006f046db91cc09586beec7afedd889)

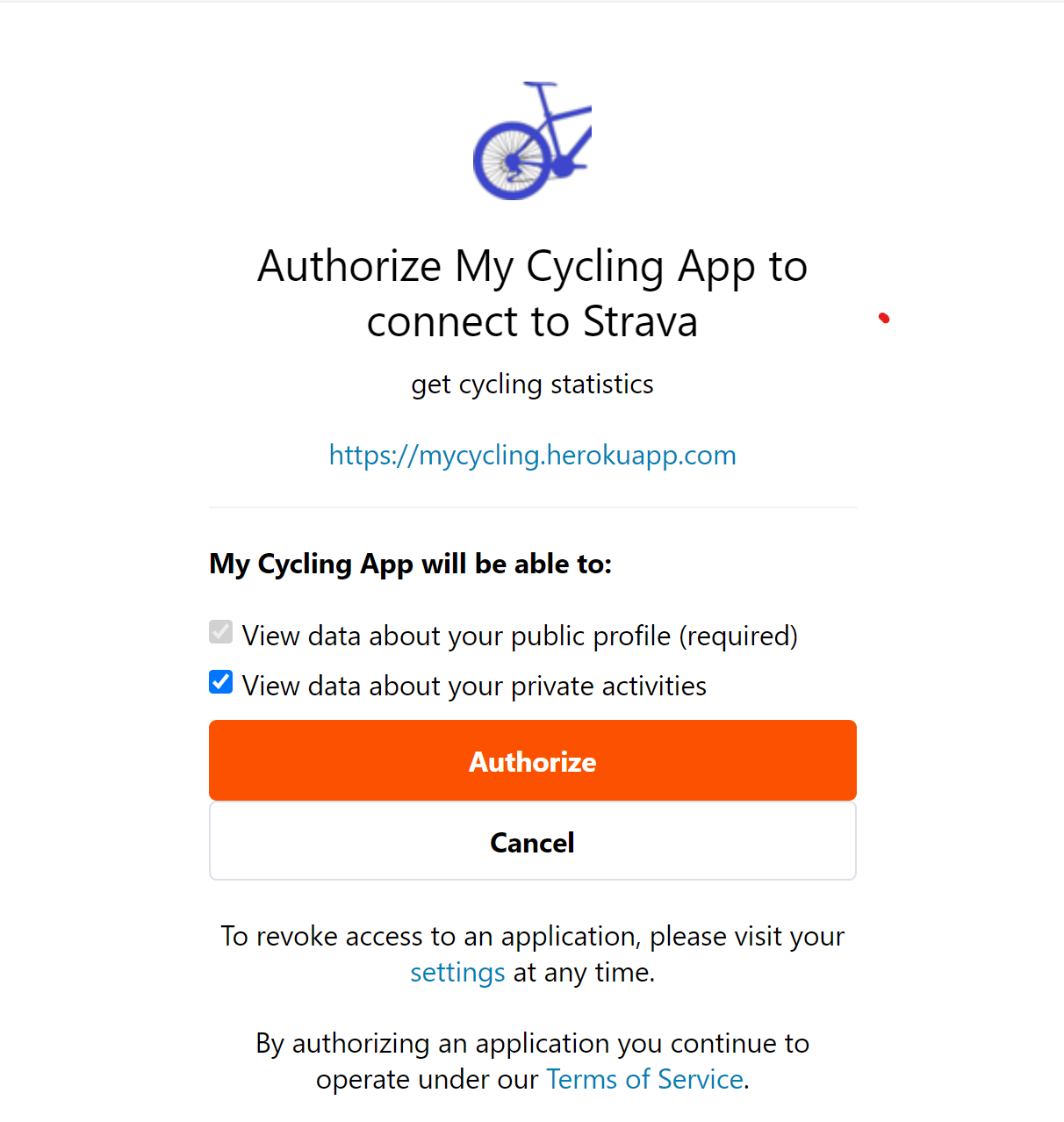
Lisää selostukseen esim. seuraava ongelma ja kuinka se ratksaistaan tyyli: tarinankerrolisesti

Kuinkas tuo em. Access\_token on voimassa ja miten sen saa jatkossa ettei tarvii kydä katsomassa sivulta erikseen????

<https://developers.strava.com/docs/getting-started/#oauth>

step1

https://www.strava.com/oauth/authorize?client\_id=77276&response\_type=code&redirect\_uri=http://localhost/exchange\_token&approval\_prompt=force&scope=activity:read\_all



http://localhost/exchange\_token?state=&code=**a0c97102e48ab9d954f2dfd740a58e00672c57b9**&scope=read,activity:read\_all

https://www.strava.com/oauth/token?client\_id=77276&client\_secret=b90fcaa7490fd8d5d3871f66484ae6cd42921c99&code=**a0c97102e48ab9d954f2dfd740a58e00672c57b9**&grant\_type=authorization\_code

client\_secret b90fcaa7490fd8d5d3871f66484ae6cd42921c99

code **a0c97102e48ab9d954f2dfd740a58e00672c57b9**

grant\_type authorization\_code

em.palauttaa access\_tokenin yms.

Pitäis saada em. kutsuttua javasciptillä -> mah.d CORS onfelma

<https://howtofix.io/fetch-data-from-strava-api-and-return-data-to-table-id513434>

**REACT**

<https://javascript.plainenglish.io/strava-api-react-app-326e63527e2c>

Haetaan access\_token refresh\_tokenin (tämä saatiin edellä ollella tavalla code:n avulla) avulla

const callRefresh = `https://www.strava.com/oauth/token?client\_id=${clientID}&client\_secret=${clientSecret}&refresh\_token=${refreshToken}&grant\_type=refresh\_token`

Get Activities

<https://www.youtube.com/watch?v=sgscChKfGyg&t=263s>

**GET** <https://www.strava.com/api/v3/athlete/activities?access_token=f6f460ca52026a7a7111f8bd745a605c1ebfbd72>

Kokeile: vaihtoehtona **Javascript**-ratkaisu HTML versioon verrattuna

<https://developers.strava.com/docs/reference/>

GET

/athlete/activities

HUOM seuraavassa fonttina Source Code Pro

ar StravaApiV3 = require('strava\_api\_v3');

var defaultClient = StravaApiV3.ApiClient.instance;

*// Configure OAuth2 access token for authorization: strava\_oauth*

var strava\_oauth = defaultClient.authentications['strava\_oauth'];

strava\_oauth.accessToken = "YOUR ACCESS TOKEN"

var api = new StravaApiV3.ActivitiesApi()

var opts = {

'before': 56, *// {Integer} An epoch timestamp to use for filtering activities that have taken place before a certain time.*

'after': 56, *// {Integer} An epoch timestamp to use for filtering activities that have taken place after a certain time.*

'page': 56, *// {Integer} Page number. Defaults to 1.*

'perPage': 56 *// {Integer} Number of items per page. Defaults to 30.*

};

var callback = function(error, data, response) {

if (error) {

console.error(error);

} else {

console.log('API called successfully. Returned data: ' + data);

}

};

api.getLoggedInAthleteActivities(opts, callback);

**Backendissä**

Koska strava API ei sisällä filltröintiä palautettavien tietojen/kenttien suhteen niin tehdään se paluudataan map-funktiolla.

Siis kun data on noudettu ja filtteröity tietokenttien suhteen, kirjoitetaan data mongodb:hen.

VOIS OLLA myös valittavissa yksittääinen suositus esim. to rten ja saada myös kartalla lenkki esitettynä eli olisko sitten kantaan enemmän dataa mutta filtteröidään kunkin tilanteen/tarpeen mukaan

Mites panda tyyppinen frame????

**Datan käsittely**

Voitaisiin noSQL:llä hakea dataa mongodb:stä mutta nyt luetaan kaikki data mongosta ja kokeillaan pandas-typpistä lähetysmistapaa.

<https://danfo.jsdata.org/examples/using-danfojs-in-react>

<https://danfo.jsdata.org/getting-started>

npm install danfojs

Refreshtoken ???? kuinka token systeemi toimii

Siis pyydetäänkä aina tokenia refressin kautta

Javascript

<https://www.youtube.com/watch?v=W_-Ai33_8f8>

Voitaisiin lukea suoraan frontista mutta kokeillaan backin kautta

<https://developers.strava.com/playground/>

Olisko mongossa stravan secret ja muut tiedot

**Kirjautuminen (tsekkaa mwrqn doku ja source)**

<https://fullstackopen.com/osa4/token_perustainen_kirjautuminen>

refresh asiaa

<https://www.gatsbyjs.com/plugins/gatsby-source-strava-activities/>

Auth…

<https://github.com/tobyDickinson/strava-oauth2>

Esimerkki backend frontend

[**https://github.com/jarnovirta/activity-app-backend**](https://github.com/jarnovirta/activity-app-backend)

**Token**

Muista määritellä aika, huom token myös staravaa mutta sitä köyttää vain admin

Määritellään käyttäjälle myös rooli

**User**

Määritellään käyttäjälle myös rooli, admin tai user, rekisteröinnissä voisi olla vaihtoehtoja rooleihin mutta miten…

**Responsitiivisuus bootstrap (tulee aikanaan Koistisen kurssilla myös)**

# Tästä eteenpäin vanhaa ------------------------

# Middleware

<https://expressjs.com/en/guide/writing-middleware.html>

# Reiti

Kuinka estetään syöttämällä url

kirjautumattomat voivat nähdä pääsivun sekä harrastajien tulokset. Kirjautuneet ammattilaiset voivat nähdä lisäksi myös ammattilaisten tulokset. "Omat tulokset"-sivu sekä sen navigointivalinta näytetään vain kirjautuneille (voi olla harrastelija tai ammattilainen)

# 

# Ajankäyttö

Katos MERN2021 kurssin markdown-doku jossa kerrottu vaatimuksist mm. validonnin suhteen TOKEN

Tunteja en ole laskenut mutta paljon pitkässä aikaikkunassa, työ oli oppimisprosessi ja se kokonaisuuden kannalta tärkein soveltaa ja kerrata palauetlla mieliin oppimaansa sekä käyttää luovuttaan/mielikuvitusta ongelmanratkaisussa. Oli aikaisempi kokmeus fulldtackin tekemisestaä nimeomaan MERN-stack, mutta se silloin oli haastavaa ja lähdekkodeista tuli bullshittiä ei kestä pöivänvaloa, mutta toimi ja se silloin riitti koska.

Haluan nyt perustella miksi Reactin valitsin, toki jokin näkemys/komeus ja sitten tulikin kutssi eteen ja syvntämään omia aikaisempia räpellyksi toki ”raamattuna#” toimi fullstackopen

Työ perustui monelta osin soveltaen fullstackopen-kurssin materiaalin oppeja, tämän WEB-kehittäjä kurssikokonaisuuden NodeJS-osuus toi syvyyttä ja selvennystä miksi ja miten jotkut asiat toimivat.

KYLLÄ OLI ÄRSYTTÄVIÄ ongelmia eräs kovin arvostamani opetaja Saconista mainitsi kerran, että kun satuu tiettyjä selittämättömiä ongelmia niin se on kuin maanpöäällinen helvetti, juuri näin.

<https://dev.to/tuanlc/mongoerror-e11000-duplicate-key-error-collection-587l>

|  |  |
| --- | --- |
|  | [MongoError: E11000 duplicate key error collection ?? - DEV Community](https://dev.to/tuanlc/mongoerror-e11000-duplicate-key-error-collection-587l)  What happened here? To understand, I would go further a bit with Indexing and unique keys in Databases. Database Indexes Wikipedia defines:; A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.  dev.to |



Pitkiö async await ketjuja app.js:ään olisi toki voinut käyttää context-globaali (selvitä mikä on teknologi tälle)

Aluksi kun ajattelin aihetta niin tuli mieleeni aikaisemmin tehdyn sovelluksen parantaminen, mutta päädyinkin kertaamaan MERN.

Siellä oli koodia, joka ei kestä päivänvaloa ja se jäi silloin deadlinen painaessa päälle harmittamaan. Toimivia rakaisuja mutta mutta...

Kiitos.

NodeJS.kurssi toimi hyvin kertauksena ja syventävänä sill fullstagopen.

Nyt varmaan tekisin uuden sovelluksen Angularilla oppiakseni sillä Reactiin verrattuna luultavasti pääsee vähemmällä koska Angulariin sisäänrakennettu paljon ja kuten manitsit teityy arkkitehtuuri.

No onhan tuohon Node-kurssiin tekemistä, mutta askel kerrallaan.

Itselle Node-kurssi lienee jonniin verran kertausta ( helposti unohtuu kun ei aktiivisesti käytä) kun lopputyöhön tuli backendiä käytyä läpi lähinnä <https://fullstackopen.com/> , toki aina uutta/syvempää asiaa, uusia näkökantoja ja oivalluksia tulee kun palauttelee mieliin.

Uskonvahvistusta.

lisäsin rivin docker-compose.yml tiedostoon kohtaan mongo-express environment

ME\_CONFIG\_MONGODB\_SERVER: mongo

Ja sitten

Ihmettelin myös tätä puolen päivää koska Tommin videolla kuitenkin toimi ohjeen mukaan. Sitten ajattelin "järjellä" (maalaismalli), että koska localhost selaimessa näyttää express niin  osoite kantaan ele docker mongon ip vaan localhost.

Kyllähän Angular - Nest.JS olisi jos...

Jos en olisi ottantu muita opistoja niin olisin ottanut Angular - Nest.JS  haltuun tekemällö harjoitustyön niillä.

Se miksi työn Frontin tein Reactilla, johtui siitä että halusin virkistää muistiwni kun fullstackopen-materiaalia hyödyntäen tein erään harjoistustyön Savonian, aihe oli vapaa ja reunehot mysö enkä oikein keksinut muuta kuin erään virityksen.

Ja tuosta jäi hieman motivaatiota muistell vanhoja ja en ollut täysin tyytyväin kaikkiin ratkaisuihin vaikka ne toimivat, mutta se ei ole pääasia vaan uudelleenkäytettävyys, ylläpidettävyys, jatkokehitettävyys ja dynaminen

Se mitö opin että aikoinnan jotakin C# reflectionia niin noSQL schemat jotenkin liityy ajatuksenn dynaamisuudesta

Tuli tehtyö erös harjoitustyö 2019 kun en oikein keksinyt mitään niin tein sääsovelluksen tähän liite ja video (mistä voi striimata? ehkä jamkin pilvestä vois jakaa) se miksi valitsin REACTin niin halusin palauttaa mieliin ja rauhallisemmassa tahdissa kun silloin aikoinnan tuli tehtyä deadlinen lähetyessä 24/7 suoritus. Nyt on vaan taas se tilanne, että unohtanut varsin lahjakkasti mitä tuli tehtyä.

ps.

Mikä linux sinulla käytössä?

TITLEN VOIS MUUTTAA

<https://mern2021.azurewebsites.net/>

<https://mern2021.herokuapp.com/>

ps.

Jostain syystä oli mennyt opintopiteet väärälle, ja jotain syystä on 2

# KOEJAA sovn titile ja icon pitä vaihtaa!!!

Tsekkaa scheman rajoittimet mm email StudentSchemasta mallia

Username unique!!!!

Selvitä routerin parametrit sill Tukan jutuissa autohorize on yhtenä välissä ennen varsinaista routekontrolleria

# Deploying Heroku

Tästä puutttuu puuttuu build proc env yms. fullstackopennissa sovellus internetiin

frontend: npm run build

heroku login

heroku create greattrainingdiary

git remote -v

jo ei ole oikein niin seuraava käsky

heroku git:remote -a greattrainingdiary

kopioidaan syntynyt build-hakemisto backendin juureen

lisättän backendiin

app.use(express.static('build'))

backend:

git init

git add .

git commit -m ”update”

git push heroku master

<https://mern2021.herokuapp.com/>

# Frontend validation

input className="form-control" required

Kentät pitää olla Form-tagin sisällä.

Button type="submit"

# Lisääg

Olisko n´buttonien id turhaa ja sen vökittäminen sllä kun row sisältää state-muuttujan id jonka se saa dat-objektin item jäsenenestä/propertystä item.\_id

Selvitö mikä virheilmoitus kun expression.response näyttää omatekemön vieht ilmoiotuksen/koodin.

Validation

<https://stackoverflow.com/questions/57992373/how-to-get-id-of-a-button-in-a-form-with-two-buttons-both-from-type-submit>

Form.Control required

ei toimi vaan

input className="form-control" required

toimii

nyt submit esityyää

Cannot access 'data' before initialization

Siis type submit ja onClick

VIRHE jos serveri alhaallla käy koko async await ketju läpi ja mitä paauttavat ilmiesti statsu ongelma jossakin koska serveri aljhllaa niin ei saada statustta

data-service.js ilmoitta network error se pitää saada palautettua, edellinen pitää verrata loginiin ehkä exception.respose on omia virheilmoituksia varten

Tiedon syöttämisessä ongelma jos kenttä on tyhjä åitäähuomioida validation että pitää olla jotakin kentässä!!!!

Ei ole submit mutta saako jotenkin validation ehkö laitettava formiin kuten login!!!

Testi: serveri alas ja yritä listäaddIten pitäis paauttaa virheen joka ilmoitetaan samin kuin loginin yhteydessä.

Logout muuttaa urlin ja routerin miten????

Tsekkaa missä consol.logia tapahtumat enne jälkeen toastion tapahtuu jokin vlitila välähtää kun addItem

Kiinnitä huomio handleformin alussa niin että modea ei vaihdeta ennen kuin toimenpiteet tehty!!!!

setMode(\_mode);

Toast delay async await eli odotetaan kunnes suljetaan

Tsekkaa liityykö localStoresta token niin

Toasteja eri CRUD toimintoihin

<http://localhost:3000/ProResults> estettävä jos ei kirjauduttu

## react router redirect if not logged in

<https://fullstackopen.com/osa7/react_router#redirect>

## react router redirect programmatically

## useHistory Hook tai Redirect

import { useHistory } from "react-router-dom";

function App() {

let history = useHistory();

}

history.push('/someRoute')

<https://dev.to/projectescape/programmatic-navigation-in-react-3p1l>

localStrote for token

## react router redirect after login

# Selvitä

Token pohjaisessa kirjutumisessa ei tarvitse useria tallentaa vain token ja token lähetetään ja siit saada käyttäjätunnus ja id.

request.params.id

request.query jos on muuten lisäänyt parametrejä ikään kuin <url:n> perään

request.bodyssä on data

Kuinka siirrytään ohjelmallisesti sivulta tosielle/tai halutulle

Kuinka estetään osoiteriviltä

Testaa kokoeile url siv ulle mytrainings

https://fullstackopen.com/osa5/kirjautuminen\_frontendissa#tokenin-tallettaminen-selaimen-local-storageen

# Miksi nacbar on leveämpi kuin table tai toisinpäin miksei siis yht leveitä

# Mongodb/mongoose Lookup tärkeää

mongoDb kantaan collection tallentuu monikossa esim. user -> users

joten viittaus esim. lookupin yhteydessä monikossa eli s perään

https://stackoverflow.com/questions/53266339/mongodb-lookup-returns-empty-array

# Backend date-format kellonaika pois

{ $project: {"date":{ $toString:"$date"},"repeat":"$repeat","user":"$user"}},

{ $project: {"date":{$toDate:"$date"},"repeat":"$repeat","user":"$user"}},

{ $project: {date: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },"repeat":"$repeat","user":"$user"}},

{ $project: {"date":{$toDate:"$date"},"repeat":"$repeat","user":"$user"}},

{ $group: { \_id: "$date", repeat\_sum: { "$sum": "$repeat" },sum:{$addToSet:"$user"} }},

{ $project: {"pvm":"$\_id","repeats":"$repeat\_sum","users":{"$size":"$sum"},\_id : 0} },

{ $sort: {\_id : -1} }

])

=>

db.data.aggregate([

{ $project: {"date":{$toDate:{$dateToString: { format: "%Y-%m-%d", date: {$toDate:{ $toString:"$date"}}}}},"repeat":"$repeat","user":"$user"}},

{ $group: { \_id: "$date", repeat\_sum: { "$sum": "$repeat" },sum:{$addToSet:"$user"} }},

{ $project: {"date":"$\_id","repeats":"$repeat\_sum","users":{"$size":"$sum"},\_id : 0} },

{ $sort: {\_id : -1} }

])

=>

db.data.aggregate([

{$lookup:{"from":"user","localField":"user", foreignField:"email", "as":"utype"}},

{$match:{"utype.pro":true}},

{$project: {"date":{$toDate:{$dateToString: { format: "%Y-%m-%d", date: {$toDate:{ $toString:"$date"}}}}},"repeat":"$repeat","user":"$user"}},

{$group: { \_id: "$date", repeat\_sum: { "$sum": "$repeat" },sum:{$addToSet:"$user"} }},

{$project: {"date":"$\_id","repeats":"$repeat\_sum","users":{"$size":"$sum"},\_id : 0} },

{$sort: {\_id : -1} }

])

Sovellukseen =>

Data.aggregate([

{ $lookup: { "from": "users", "localField": "user", foreignField: "email", "as": "utype" } },

{ $match: { "utype.pro": pro } },

{ $project: { "date": { $toDate: { $dateToString: { format: "%Y-%m-%d", date: { $toDate: { $toString: "$date" } } } } }, "repeat": "$repeat", "user": "$user" } },

{ $group: { \_id: "$date", repeat\_sum: { "$sum": "$repeat" }, sum: { $addToSet: "$user" } } },

{ $project: { "date": "$\_id", "repeats": "$repeat\_sum", "users": { "$size": "$sum" }, \_id: 0 } },

{ $sort: { \_id: -1 } }

])

# Token

const token = jwt.sign(userForToken, "kuopio")

olisiko tuon kuopion tilalla username

ja samoin tarkastuksessa

# TEHTÄVÄT

Tsekkaa Keijon maili

Olisiko modal-dialog sitettenkin app.js:ssä, mutta miten sitö kutsuttaisiin

App.js tasolla tehdään perustoiminnot esim. yhteydet backendiin, joten UI toiminnot palauttavat app.js tasolle esim. registeFormilta ei axios-kutsua backendiin

# noSql Mongo komentoja

f:\jamk\mongo

docker-compose.yml .tiedosto sisältää mongo-framworkin/expressin

version: '3.1'

services:

  mongo:

    image: mongo

    restart: always

    volumes:

      - tk2\_mongo:/data/db

    ports:

      - 27017:27017

    environment:

      MONGO\_INITDB\_ROOT\_USERNAME: root

      MONGO\_INITDB\_ROOT\_PASSWORD: password

  mongo-express:

    image: mongo-express

    restart: always

    ports:

      - 80:8081

    environment:

      ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root

      ME\_CONFIG\_MONGODB\_ADMINPASSWORD: password

volumes:

  tk2\_mongo: {}

docker-compose up -d käynnistää kontin, joka on kuvattu yml-tiedostossa

docker exec -it mongo\_mongo\_1 mongo -u root -p password

luodaan expressissä Database MERN

$ docker exec -i mongo\_mongo\_1 mongoimport -u root -p password --authenticationDatabase=admin --drop -d MERN -c data < data.json –jsonArray

$ docker exec -i mongo\_mongo\_1 mongoimport -u root -p password

--authenticationDatabase=admin --drop -d MERN -c user < user.json –jsonArray

$ docker exec -i mongo\_mongo\_1 mongoimport -u root -p password

--authenticationDatabase=admin --drop -d MERN -c data < data.json --jsonArray

# Tee SPA-tyylinen React-sovellus

Sovelluksessa on seuraavat sivut: "Etusivu", "Harrastajien tulokset", "Ammattilaisten tulokset", "Omat treenit".

Pääsivulla on sovelluksen nimi TREENIPÄIVÄKIRJA, taustakuva sekä valikkopalkki, jossa navigoinnit em. sivuille. Lisäksi kirjautumattomalle käyttäjälle näytetään valinta "Kirjaudu". Jos käyttäjä on kirjautunut, näytetään hänen etunimensä

## Navigointi (Routing)

<https://fullstackopen.com/osa7/react_router>

### Sovelluksen sivujen rungot

Tehdään ensin hakemisto routes ja sinne Homepage.js AmResults,js ProResults.js MyTrainings.js komponentit (sivut)

Importoidaan em. sivut App.js:ään

import Homepage from './routes/Homepage';

import AmResults from './routes/AmResults';

import ProResults from './routes/ProResults';

import MyTrainings from './routes/MyTrainings';

Sivujen rungot ovat seuraava muotoa

import import React from 'react';

const Homepage = () => {

  return (

    <div>

      <h2>Homepage</h2>

    </div>

  )

}

export default Homepage

### Asennetaan React router

Jos projektin template ei sillä kirjastoa router (package.json)

npm install react-router-dom

otetaan käyttöön App.js:ssä

import {

  BrowserRouter as Router,

  Switch, Route, Redirect, Link

} from "react-router-dom"

Yksinkertainen esimerkki: perusajatus

    <Router>

      <h1>

        TREENIPÄIVÄKIRJA

        </h1>

      <h6>

        Harjoitus 4, created by Timo Laakkonen

        </h6>

      <div>

        <Link style={padding} to="/">Home</Link>

        <Link style={padding} to="/AmResults">Amateur</Link>

        <Link style={padding} to="/ProResults">Professional</Link>

        <Link style={padding} to="/MyTrainings">My trainings</Link>

        <Link style={padding} to="/Login">Login</Link>

      </div>

      <Switch>

        <Route path="/AmResults">

          <AmResults />

        </Route>

        <Route path="/ProResults">

          <ProResults />

        </Route>

        <Route path="/MyTrainings">

          <MyTrainings />

        </Route>

        <Route path="/Login">

          <Login />

        </Route>

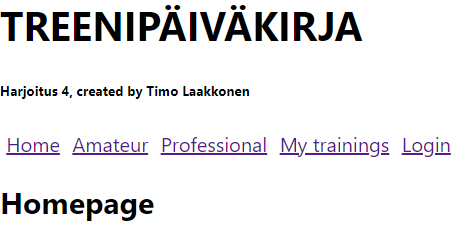
        <Route path="/">

          <Homepage />

        </Route>

      </Switch>

    </Router>



### Bootstrap

Muutetaan edellisen kohdan navigointi käyttämään bootstrap-kirjaston komponentteja

Router

<https://www.npmjs.com/package/react-router-bootstrap>

npm install -S react-router-bootstrap

Bootstrap

<https://react-bootstrap.github.io/getting-started/introduction>

npm install react-bootstrap bootstrap

index.js

<Router><App /></Router>

App.js

import { LinkContainer } from 'react-router-bootstrap';

Käytä tätä tyyliä kun hauat positioida jonkun esim. buttonin oikealle

className='ml-auto'

import React, { useState, useEffect } from 'react'

Icons

<https://www.npmjs.com/package/react-bootstrap-icons>

npm install react-bootstrap-icons –save

https://icons.getbootstrap.com/

// POIS

<https://icons.getbootstrap.com/>

<https://www.npmjs.com/package/bootstrap-icons>

npm i bootstrap-icons

Taustakuvia

<https://www.google.com/search?q=athletic+gym+picture+svg&tbm=isch&ved=2ahUKEwi-sryh44nvAhWPvCoKHfAcC_kQ2-cCegQIABAA&oq=athletic+gym+picture+svg&gs_lcp=CgNpbWcQA1Cc1gJYw9wCYLDmAmgAcAB4AIABYIgBqQKSAQE0mAEAoAEBqgELZ3dzLXdpei1pbWfAAQE&sclient=img&ei=CBQ6YL6OII_5qgHwuazIDw&bih=792&biw=1536&rlz=1C1CHBD_enFI941FI941>

# Login

<https://fullstackopen.com/osa5/kirjautuminen_frontendissa>

<https://fullstackopen.com/osa7/react_router>

history.push(’/’) tarvitaanko?

**Register**

Komponenttia ei käytetän varsinaiseti itsenöisenö vaan login.komoponentin yhteydessä kuten toimintakin rekisteröidöön login-formin kautta

Uusi harjoitus treeni

Lajit on dropdownissa ja lajit ovat mongo-kannassa

Jos epäonnistuu niin jokin ilmoitus mahd. modal

**Validiointi**

Ei saa olla samaa emailia ennestään unique mongon skeemassa huomioitu

  email: {

    type: String,

    unique: true

  }

# Huomioita

Milloin server lähettää .send kutsun onko sille vastinetta siis toista tapaa

res.json palauttaa objektin send stringin

selvitä status ja code

**Input TÄRKEÄ**

      <input

        type="text"

        value={username}

        name="Username"

        onChange={({ target }) => setUsername(target.value)}

      />

**text-muted**

<Form.Text className="text-muted">

We'll never share your email with anyone else.

</Form.Text>

# Kirjautuminen ja käyttäjien hallinta

<https://fullstackopen.com/osa4/kayttajien_hallinta>

<https://fullstackopen.com/osa5/kirjautuminen_frontendissa>

GraphQL Apollo

<https://fullstackopen.com/osa8/tietokanta_ja_kayttajien_hallinta>

# Backend

## Express

F:\Savonia\MERN2021\VIDEOT\Express\GetMP4.mp4

Express/node.js

<https://fullstackopen.com/osa3/node_js_ja_express>

npm init

npm install express

const express = require('express');

const app = express();

Json-parser

npm install body-parser (ei asenneta) sillä express sisältää json-parserin

app.use(bodyParser.json())

korvataan

app.use(express.json());

Routes (middleware)

**Nodemon**

Kehityksen aikainen (kun tehdään muutoksia koodiin niin serveriä ei tarvitse sulkea ja käynnistellä uudelleen vaan nodemon hoitaa asian/uudelleenkäynnistyksen automaattisesti)

npm install nodemon --save-dev

package.jsoniin kijoitetaan serverin kynnistys scriptiin

"start": "nodemon app.js"

Server käynnistyy “npm start”-komennolla

backendin testaus yhteys frontista:

**axios (Frontend)**

npm install axios

**Cors (Backend)**

npm install cors

**Ympäristömuuttujat**

dotenv

npm install dotenv

tiedosto .env jossa ”ei kovakoodattua dataa” esim.

PORT=3001

SECRET=petonen

# Mongo

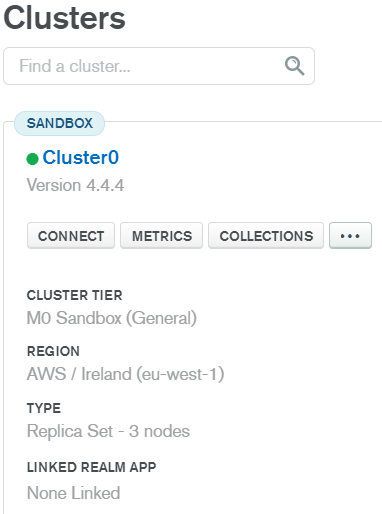
<https://fullstackopen.com/osa3/tietojen_tallettaminen_mongo_db_tietokantaan#mongo-db>

https://www.mongodb.com/cloud/atlas/register

Kirjautuminen:

[timo.laakkonen@edu.savonia.fi](mailto:timo.laakkonen@edu.savonia.fi)

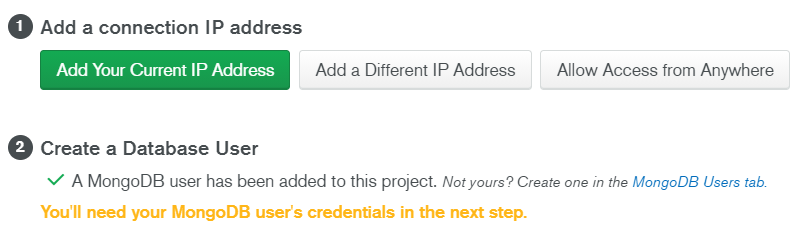
mern#2021

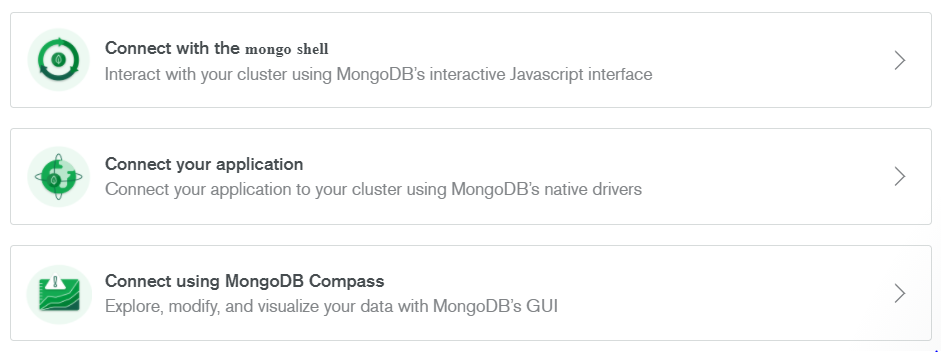


Valitaan CONNECT

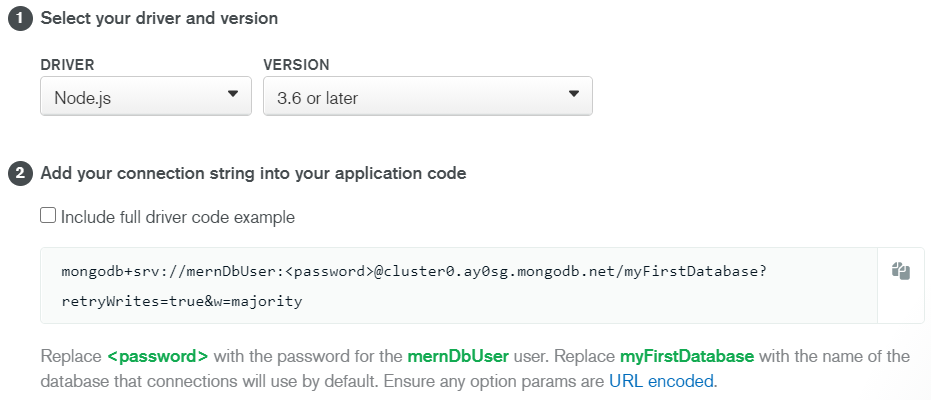
User: mernDbUser

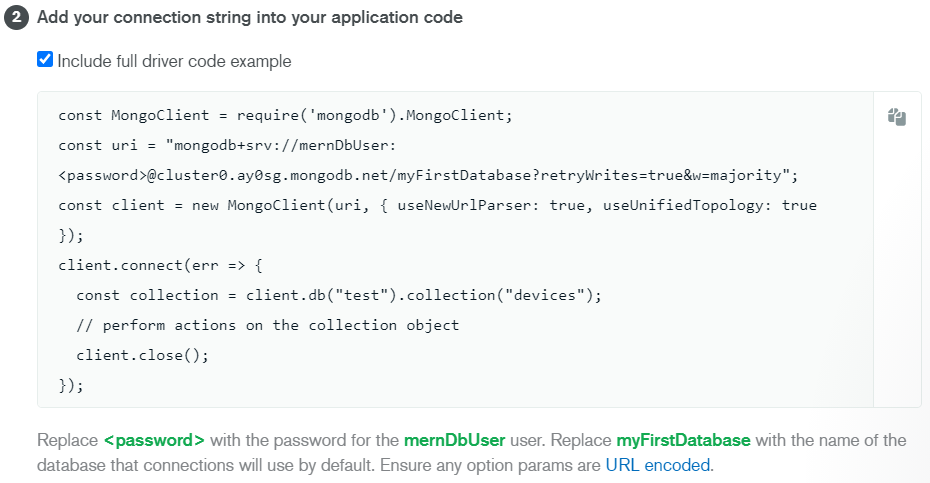
Psw: mern2021





Valitaan Connect your application





const MongoClient = require('mongodb').MongoClient;

const uri = "mongodb+srv://mernDbUser:<password>@cluster0.ay0sg.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";

const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

client.connect(err => {

const collection = client.db("test").collection("devices");

// perform actions on the collection object

client.close();

});

<https://www.w3schools.com/nodejs/nodejs_mongodb.asp>

**Mongoose**

<https://mongoosejs.com/>

npm install mongoose

testaan kantaa erillisellä testikoodilla

**Kannan luonti**

Serverin köynnistyksen yhteydessä luodaan mongoyhteys

const uri = `mongodb+srv://mernDbUser:${password}@cluster0.ay0sg.mongodb.net/training?retryWrites=true`;

luodaan training -kanta johon ”taulut” eli collectionit jotka luodaan seuraavan määrittelyn ohessa

const User = require('../models/user-model')

em. Kutsu/määritys tiedoston viimeinen rivin määritys “User” on luotavan collectionin nimi johon lisätään s (mongon käytäntö)

module.exports = mongoose.model('User', userSchema)

**Käyttäjä-schema**

MongoDB Atlas osaa luoda kannan automaattisesti, jos sovellus yrittää yhdistää kantaan, jota ei ole vielä olemassa.

module.exports = mongoose.model('User', userSchema)

Validation

const uniqueValidator = require('mongoose-unique-validator')

  email: {

    type: String,

    unique: true

  }

const noteSchema = new mongoose.Schema({

content: {

type: String,

required: true,

minlength: 5

},

# Client validation

Input type="email" type="number"

required placeholder="xxx"

# User/register

**Validointi backend/mongo**

Koska validationja email unigue niin siitö pitöis saada feedback jos yrotetään samaa nöin ollen ei tarvitse kollata user-taulua hakemalla onko jo olemassa

Vaikka frontendissä tsekataan jo että on content niin varmistetaan backendissä ennen tallennusta että asia näin on

if (!body.content) {

return response.status(400).json({

error: 'content missing'

})

}

# Login

Validointi myös rekisteröinnin suhteen, backendissä omat..

required type (esim. type=number tai type=email type=password jo osaltaan validointia

tee funktio checkIsValid jossa omia tarkistussääntöjä esim. onko kentät lognissa täytetty

return user.id && user.password; kun molemmissa arvo niin…



**Backend**

<https://fullstackopen.com/osa4/token_perustainen_kirjautuminen>

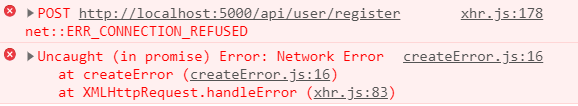
# Token

# Async/await

<https://fullstackopen.com/osa4/backendin_testaaminen#async-await>

# Virhetilanteet

## Serveri alhaalla



Huom erikseen kappaleet validointi forntend/nbackrnd sekä virheidenksittely ja kenties viittau validointiin

<https://fullstackopen.com/osa3/tietojen_tallettaminen_mongo_db_tietokantaan#virheiden-kasittely>

.catch(error => next(error)

Huom kohta

Virheidenkäsittelyn keskittäminen middlewareen

# Toast

https://react-bootstrap.github.io/components/toasts/

# HUOM

## MUISTA virheilmoiukset eli apaluta köyttäjälle jotakin tietoa

Tosin koodissa ei seuraava pääse vlttämättä syntymään että anneta id jota ei ole

JS find-metodi (JSON-taulukkoon)

router.get(’/:id’,(req,res,next)=>{

const orderId=parseInt(req.params.id);

const order=TESORDERS.find(p=>{

return p.id===orderId;

));

res.json({order});

}

app.get('/api/notes/:id', (request, response) => {

const id = Number(request.params.id)

const note = notes.find(note => note.id === id)

if (note) {

response.json(note)

} else {

response.status(404).end()

}}

)

Jos ei löydy ko. Id:tä niin virhe koodi end() takoittaa ei viestiä, mutta

response.status(404).json({message:’Could not find the order’})

Ehto if else tai sitten virhe-ehdon toteutuessa return response.

**MUUTETAAN**

// ketjun viimeinen virhekäsittelijä tänne

app.use((error, req, res, next) => {

    if (res.headersSent) {

        return next(error);

    }

    res.status(error.code || 500)

        .send({ message: error.message || 'Unknown error' });

});

Jolloin routeriin

If(!order){

const error=new Error(’Orde could not be found’)

error.code=404;

return next(error);

}

F:\Savonia\MERN2021\ VIDEOT\Controllers\GetMP4.mp4

Video 21:00

# Icons

Bootstrap svg icon

<https://icons.getbootstrap.com/>

esim. <https://icons.getbootstrap.com/icons/music-note-list/>

<https://visualpharm.com/free-icons>

<https://css-tricks.com/scale-svg/>

Esim.

<svg xmlns="http://www.w3.org/2000/svg" width="32" height="32" fill="#0000FF" class="bi bi-star"

            viewBox="0 0 16 16">

            <path

              d="M2.866 14.85c-.078.444.36.791.746.593l4.39-2.256 4.389 2.256c.386.198.824-.149.746-.592l-.83-4.73 3.523-3.356c.329-.314.158-.888-.283-.95l-4.898-.696L8.465.792a.513.513 0 0 0-.927 0L5.354 5.12l-4.898.696c-.441.062-.612.636-.283.95l3.523 3.356-.83 4.73zm4.905-2.767l-3.686 1.894.694-3.957a.565.565 0 0 0-.163-.505L1.71 6.745l4.052-.576a.525.525 0 0 0 .393-.288l1.847-3.658 1.846 3.658a.525.525 0 0 0 .393.288l4.052.575-2.906 2.77a.564.564 0 0 0-.163.506l.694 3.957-3.686-1.894a.503.503 0 0 0-.461 0z" />

</svg>width height fill voidaan muuttaa halutuksi selvitä pitöisikö viewbox olla myös w:n ja h:n mukainen

Lisää ikoneita mm.

https://iconsvg.xyz/

TEE funktio kutsu ketju kuva

Esim. loginform: app.ddd loginserveice🡪 backend usercontroller---

ja siitä kuinka async/await return toimii

edellinen vain kuviteltu

mutta

users-routes +> user-controllers palauttaa virheen mihin se tulee todellisuudessa

# DATE

**Datepicker**

[**https://www.npmjs.com/package/react-datepicker**](https://www.npmjs.com/package/react-datepicker)

npm install react-datepicker --save

# Object.values

# Hamburger

# Editointi

Pvm komponentti, laji dropdown

**Dropdown**

<https://bootsnipp.com/snippets/M2EEA>

<https://www.geeksforgeeks.org/how-to-use-bootstrap-select-for-dropdown/>

# Tooltip

<https://react-bootstrap.netlify.app/components/overlays/#tooltips>

# PopOver

<https://react-bootstrap.netlify.app/components/overlays/#popovers>

# Sovelluksesta

Kaikki data otetaan aluksi muistiin tilamuuttujaan jos editoidaan dataa ei tarvitse hakea editoitavaksi vasta talletus backendiin vie ja datan lukeminen ”kaikki”

Samoin kun liöstätän uusi niin tallnuus vie backkiin ja pihalle kaikki, am tietty myös deleten osalta

# Testi

Kun user-kanta on tyhjä tai sitä ei ole olemassa.

Tehdään login kutsu

Alkutilanne kun on kirjauduttu

# Async await try catch

Vissiin ei awaitiin try catchia vaan epäonnistuneen kutsun paluuarvo on null

# Select options

<Form.Control as='select'>

<option value=''>Select a food:</option>

<option value='chicken parm'>Chicken Parm</option>

<option value='BLT'>BLT</option>

<option value='steak'>Steak</option>

<option value='salad'>Salad</option>

</Form.Control>

# Dialogit

showDialog on tilamuuttuja App.js:ssä

nimeä uudelleen shwo

<LoginDialog \_show={loginDialog} showDialog={setLoginDialog} func={handleLogin}></LoginDialog>

const LoginDialog = ({ \_show, showDialog, func }) => {

# new Item

Mytrainings

<Form mode={mode} data={data} func={handleForm} />

Handleformiin siis palautetaan jokin mode asettamalla se tilamuuttujaan jolloin form renderöityy uudelleen moden mukaisesti.

Ajatus olisi että app.js -> services eoi app.js tasolle pitää tehdä kutsu addItem

const MyTrainings = (tähän viittaus eli funktio app.js:ssä) => {

backendin validation jos syöttää e joka hyväksytään type=’’Number”

# LONGPRESS

Tooltip vois olla longpress

# logalStorage

const loggedUserJSON = window.localStorage.getItem('loggedMernUser')

    if (loggedUserJSON) {

      const user = JSON.parse(loggedUserJSON)

      setUser(user)

      setToken(user.token)

      setAdmin(user.admin)

    }

  }, [])

  const loginCallBack = (user) => {

    window.localStorage.setItem(

      'loggedMernUser', JSON.stringify(user)

    )

    setUser(user)

    setToken(user.token)

    setAdmin(user.admin)

  }

# MyData

Kirjataanko sähköpostiosoite vai id:ee referenssikis varmaankin id olisi ”ammattimaisempi

myData tilamuuttuja pitää sisällään kaikki treenit, sitä päivitetään silloin kun poistetaan tai editoidaan tai lisätää eli CRUM-toimenpiteiden jälkeen luetaan serveriltä tiedot kunhan ensin muutokset on tallennettu tietokantaan.

# Selvitä popukate

.find({}).populate('datas', { content: 1, date: 1 })

# Axios parametrit

Monesti turhia sill osoite jos siältää queryn ja mahd. bodyssakin on jo tarivttava id tosin token voidaan lähettää getConfi() 3. parametri

# Virheelisten osoitteiden kåisttely

Esim. put operaation ilman id:eetä!!!!

# Internetiin

## Heroku

Backendin juureen tiedosto *Procfile*, joka kertoo Herokulle, miten sovellus käynnistetään

Tiedoston sisältö

web: npm start

**# Harjoitus 4**

Tämä tehtävä sisältää sekä frontendin että backendin toteutusta.

Tee React-osuudet frontend-kansioon, käytä haluamiasi alikansioita.

Tee Node-osuudet backend-kansioon, käytä haluamiasi alikansioita.

Frontendin muotoiluun voit käyttää joko React-Bootstrapia tai

Material-UI-kirjastoa

HUOM! Muut UI-kirjastot eivät kelpaa!

Jokainen tehtävä on 3p => 20\*3p = 60p.

Harjoituksen sisältö: Tee MERN-stackilla fullstack-sovellus

TREENIPÄIVÄKIRJA, jolla voit selata painoilla tehtäviä treenejä. Oheisissa kuvissa on hahmoteltu sovelluksen näyttöjä, mutta tee siitä oman näköisesi.

![Teht.4 luonnos](Dia1.png?raw=true "Tehtävän 4 UI-luonnos")

![Teht.4 luonnos](Dia2.png?raw=true "Tehtävän 4 UI-luonnos")

Tietokantaan talletetaan käyttäjistä siis etunimi, sukunimi, sähköpostiosoite, salasana, onko ammattilainen. Yksittäisen käyttäjän treeneistä talletaan pvm, treenilaji, suorituskerrat, suorituspaino. Voit itse päättää millaisina dokumentteina ja collectioneina nämä talletetaan.

Harrastajasivuille tarvitsee summata yhteen harrastajien treeneistä tiedot seuraavasti: pvm, ko. päivänä kaikkien harrastajien tekemien suorituskertojen summa, kaikkien ko. päivänä mitä tahansa suoritusta tehneiden harrastajien lukumäärä. Samalla tavoin ammattilaissivua varten lasketaan vastaavat tiedot koskien ammattilaisia.

Näitä harrastaja- tai ammattilaissivujen tietoja ei tarvitse tallettaa kantaan, vaan ne voidaan laskea tekemällä kantaan sopivat haut kyseisiä tietoja näytettäessä.

**## 1. Tee SPA-tyylinen React-sovellus**

Sovelluksessa on seuraavat sivut: "Etusivu", "Harrastajien tulokset", "Ammattilaisten tulokset", "Omat treenit".

Pääsivulla on sovelluksen nimi TREENIPÄIVÄKIRJA, taustakuva sekä valikkopalkki, jossa navigoinnit em. sivuille. Lisäksi kirjautumattomalle käyttäjälle näytetään valinta "Kirjaudu". Jos käyttäjä on kirjautunut, näytetään hänen etunimensä.

**## 2. Kirjautumisdialogi frontendiin**

Kyseisestä toiminnosta avautuu modaalinen dialogi, jossa voi syöttää kirjautumista varten sähköpostiosoitteensa ja salasanansa. Jos käyttäjä on kirjautunut, niin näytetään "Kirjaudu"-valinnan tilalla valinta "Kirjaudu ulos".

**## 3. Käyttäjän kirjautumisen toteutus backendissä**

Toteuta backendiin endpoint "user/login", joka ottaa vastaan POST-toimintoja tarvittavat käyttäjätiedot kirjautumista varten. Koodi tarkistaa kannasta, että kyseinen käyttäjä löytyy.

**## 4. Kirjautumisen toteutus tokenin avulla**

Edellinen toteutus siten, että kirjautumisessa palautetaan token, jolla käyttäjä voidaan tunnistaa ja tarkistaa hänen oikeutensa eri palveluihin. Tokeniin on hyvä tallettaa esim. käyttäjän id ja hänen roolinsa eli onko ammattilainen.

**## 5. Rekisteröitymisdialogi**

Kirjautumattomalle käyttäjälle annetaan myös mahdollisuus rekisteröityä. Rekisteröitymistä varten kirjautumisdialogissa on painike "Rekisteröidy", joka avaa joko uuden dialogin tai antaa kirjautumisdialogille lisäkenttiä. Rekisteröinnin yhteydessä kysytään etunimi, sukunimi, sähköpostiosoite, salasana sekä ruksattava tieto onko ammattilainen. Jos rekisteröityjä ei ruksi ko. kohtaa, hän on silloin harrastelija. Tiedot viedään tietokantaan. Rekisteröitymistä varten on endpoint "user/register".

**## 6. Reititys frontendissa kovakoodattuna**

Toteuta frontendissa reititys ja navigointivalintojen näyttäminen siten, että kaikki käyttäjät, myös kirjautumattomat voivat nähdä pääsivun sekä harrastajien tulokset. Kirjautuneet ammattilaiset voivat nähdä lisäksi myös ammattilaisten tulokset. "Omat tulokset"-sivu sekä sen navigointivalinta näytetään vain kirjautuneille (voi olla harrastelija tai ammattilainen), eikä sinne saa päästä esim. urlia käsin kirjoittamalla. Samalla tavoin myös muille sivuille pitää estää pääsy muilta kuin luvallisilta käyttäjiltä. Oikeudet voidaan toteuttaa kovakoodattuina eli ne eivät tarvitse olla kannasta luetun käyttäjätiedon mukaan määrättyjä.

**## 7. Reititys frontendissa kannasta luetun tiedon avulla**

Frontendin reititys siten, että käyttäjän kirjautumistiedot on talletettu palvelimelta onnistuneen kirjautumisen myötä saatuna tokenina + etunimenä + roolina kontekstiin (useContext esim.) ja oikeuksien tarkistus navigoinnissa sekä etunimen näyttö "Omat treenit"-sivulla tehdään sitä käyttäen.

**## 8. "Harrastajat"-sivu, tieto haettu kannasta**

Harrastajat-sivulla näytetään harrastajien tulokset päivittäin yhteenlaskettuina. Tieto näytetään taulukkomaisesti, sarakkeet: Pvm, suoritukset yhteensä, suorittajalkm. Hakua varten backendissä endpoint "result/amateurs".

**## 9. "Ammattilaiset"-sivu, tieto haettu kannasta**

Ammattilaiset-sivulla näytetään ammattilaisten tulokset päivittäin yhteenlaskettuina. Tieto näytetään taulukkomaisesti, sarakkeet: Pvm, suoritukset yhteensä, suorittajalkm. Hakua varten backendissä endpoint "result/professionals".

**## 10. "Omat treenit"-sivu, tietojen näyttö taulukkomaisesti, tieto haettu kannasta, käyttäjä kovakoodattu**

Sivulla näytetään taulukkomaisesti yhden käyttäjän treenit. Sarakkeet: Pvm, Laji, Suorituskerrat, Paino. Hakua varten tehdään backendiin endpoint "result/<id>". Tunniste voi olla kovakoodattu.

**## 11. Edellinen autentikoituneen ja autorisoidun tunnuksen avulla**

Omat treenit voi nähdä vain kirjautunut käyttäjä ja hän näkee siellä vain omat tuloksensa, ei muiden. Tämä perustuu siihen, että sisäänkirjautuneella käyttäjällä on tallessa token, jossa on sisällä käyttäjätunniste. Kun hän hakee henkilökohtaisia tuloksia, backend katsoo vastaako pyynnön mukana tulevassa tokenissa oleva tunniste pyynnön endpointin tunnistetta.

**## 12. Omien treenien muokkaus, käyttäjä kovakoodattuna**

"Omat treenit"-sivulla käyttäjä voi muokata omia suorituksiaan. Backendissä on muokkausta varten endpoint "result/<id>". Tunniste voi olla kovakoodattu.

**## 13. Omien treenien muokkaus autentikoituneen ja autorisoidun tunnuksen avulla**

Muokkauspyynnön mukana tulee token, josta backend tarkastaa, onko käyttäjällä oikeus ko. tietojen muokkaamiseen.

**## 14. Omien treenien poisto, käyttäjä kovakoodattuna**

"Omat treenit"-sivulla käyttäjä voi poistaa omia suorituksiaan. Backendissä on poistoa varten endpoint "result/<id>". Tunniste voi olla kovakoodattu.

**## 15. Omien treenien poisto autentikoituneen ja autorisoidun tunnuksen avulla**

Poistopyynnön mukana tulee token, josta backend tarkastaa, onko käyttäjällä oikeus ko. tietojen poistoon.

**## 16. Omien treenien lisäys, käyttäjä kovakoodattuna**

"Omat treenit"-sivulla on painike, josta käyttäjälle avautuu joko samalle sivulle tai dialogiin tms. kentät, joiden avulla käyttäjä voi lisätä omia treenisuorituksiaan. Backendissä on lisäystä varten endpoint "result/<id>". Tunniste voi olla kovakoodattu.

**## 17. Omien treenien lisäys autentikoituneen ja autorisoidun tunnuksen avulla**

Muokkauspyynnön mukana tulee token, josta backend tarkastaa, onko käyttäjällä oikeus tietojen lisäämiseen. Vain kirjautuneet käyttäjät saavat lisätä treenitietoja, tässä ei tarvitse välttämättä tutkia tokenin sisältöä, ainoastaan se onko token validi.

**## 18. UI-kirjaston käyttö**

Frontendissä on käytetty joko React-Bootstrapia tai Material-UI-kirjastoa tarkoituksenmukaisilla tavoilla ja ulkoasu on siisti ja yhtenäinen. HUOM! Vain jompikumpi näistä kirjastoista kelpaa!!

**## 19. Backendin virheettömyys ja viestitys**

Backendissä täytyy käsitellä kaikki mahdolliset virhetilanteet ja lähettää niistä sopiva koodi ja ilmoitus.

**## 20. Frontendin virheettömyys ja viestitys**

Frontendissä täytyy käsitellä backendin lähettämät ilmoitukset ja viestittää niistä käyttäjälle yhdenmukaisella tavalla käyttäen UI-kirjaston dialogia.