Lopputyönä tiimeillä tulee **ostojamyynti kanava** sovelluksen tekeminen.

Sovelluksen pitää olla valmiina **1.10.2025**.

Tällöin tiimit esittelevät luokassa c303 oman työnsä koko muulle luokalle.

Työssä tullaan tarvitsemaan mm. seuraavia asioita:

* node osaamista
* tietokanta osaamista
* js osaamista
* css osaamista
* html osaamista
* salausmenetelmä osaamista
* testaus osaamista
* jne…

Jatkokehitystehtäviä, kun **ostojamyynti kanava** sovelluksen jatkamiseen:

* Rekisteröitymislomakkeen toiminnan tutkiminen ja parantaminen
  + Nyt lomakkeella pystyy rekisteröimään ”tyhjän” käyttäjän
  + Kaikissa kentissä ei ole tarkastuksia
  + Miten korjataan?
* Omien tietojen muokkaaminen
* Salasanan vaihtaminen
* Admin käyttäjälle hallinta sivun lisääminen
  + voi muokata käyttäjän tietoja
  + voi poistaa käyttäjän (pitääkö tehdä muutoksia tietokantaan, esim. jos käyttäjällä ilmoituksia => voi poistaa siitä huolimatta)
* Etusivulle admin ja mod käyttäjille painikkeet ilmoituksien muokkaamiseen ja poistamiseen.

Osto ja myynti Node + hbs

npm init -y

npm i express mysql dotenv nodemon bcryptjs cookie-parser jsonwebtoken express-handlebars

!HUOM kirjastot voidaan asentaa kaikki kerralla tai sitä mukaan kun tarve ilmenee. Esim. bcryptjs vasta sitten kun käsitellään salasanan kryptausta.

luo tiedosto **app.js**

sisältö:

import express from 'express';

const app = express();

app.get('/', (req, res) => {

    res.send("<h1>Home page</h1>");

});

app.listen(3030, () => {

    console.log("Server started on port 3030");

});

Lisätään **package.json** tiedostoon skripti:

"start": "nodemon app.js"

Lisätään samalla ES6 syntaksin edellyttämä **type module**:

"description": "",

  "type": "module",

  "main": "index.js",

Annetaan komentoriville loitsu: **npm start**

Avataan selain osoitteeseen: **localhost:3030**

Lisätään kantayhteys **app.js** tiedostoon:

import express from 'express';

import mysql from 'mysql';

const app = express();

const db = mysql.createConnection({

    host: 'ec2-13-48-195-46.eu-north-1.compute.amazonaws.com',

    user: 'root',

    password: 'my-new-password',

    database: 'ostojamyynti'

});

app.get('/', (req, res) => {

    res.send("<h1>Home page</h1>");

});

app.listen(3030, () => {

    console.log("Server started on port 3030");

    db.connect( (error) => {

        if(error) {

            console.log(error);

        } else {

            console.log("Connected to database");

        }

    });

});

Muuta database virheelliseksi (esim lisää n kirjain loppuun) huomaat, että konsoliin tulee virheilmoitus. Korjaa database oikeaksi.

Listään vielä app.js tiedoston loppuun:

*export* default db;

Näin päästään helposti hyödyntämään tietokantayhteyttä sovelluksen muissa osissa.

Lisätään turvallisuutta käyttämällä .env tiedostoa kantayhteyden tietojen piilottamiseen.

(HUOM! Jos käytössä ei ole nodemonia täytyy sovellus käynnistää uudelleen!)

Luodaan juureen .env tiedosto johon tallennetaan oman tietokannan tiedot:

HOST = ec2-13-48-195-46.eu-north-1.compute.amazonaws.com

DATABASE = ostojamyynti

USER = root

PASSWORD = my-new-password

Muokataan app.js:

import dotenv from 'dotenv';

dotenv.config();

const app = express();

const db = mysql.createConnection({

    host: process.env.HOST,

    user: process.env.USER,

    password: process.env.PASSWORD,

    database: process.env.DATABASE

});

Tehdään ensimmäinen html sivu handlebareja hyödyntäen.

Luodaan juureen kansio **public**, jonka sisälle kansio **assets**, jonka sisälle kansio **styles**, jonka sisälle tiedosto **styles.css**

body {

    background-color: aqua;

}

Luodaan projektin juureen kansio **views**, jonka sisälle kansio **layouts**, jonka sisälle tiedosto **main.hbs**

<!DOCTYPE html>

<html *lang=*"fi">

<head>

    <meta *charset=*"UTF-8">

    <meta *http-equiv=*"X-UA-Compatible" *content=*"IE=edge">

    <meta *name=*"viewport" *content=*"width=device-width, initial-scale=1.0">

    <link *rel=*"stylesheet" *href=*"/assets/styles/styles.css">

    <title>Osto ja myyntikanava</title>

</head>

<body>

    {{{body}}}

</body>

</html>

Lisäksi luodaan **views** kansioon tiedosto **index.hbs** tiedosto:

<h1>Osto ja myyntikanava</h1>

Otetaan handlebars sekä public kansio käyttöön muokkaamalla **app.js** tiedostoa:

import dotenv from 'dotenv';

import { create } from 'express-handlebars';

dotenv.config();

const app = express();

app.use(express.static('public'));

const hbs = create({ extname: '.hbs'});

app.engine('hbs', hbs.engine);

app.set('view engine', 'hbs');

Jotta sivut saadaan näkyviin, muutetaan app.get res.send -> res.**render**. Handlebar hakee layouts kansiosta main.hbs tiedoston ja nyt sen {{{body}}} sisällöksi asetetaan index.hbs

app.get('/', (req, res) => {

    res.render('index')

});

Lisätään valikko sivustolla liikkumista varten: **main.hbs**

Tämän käyttö vaatii, että laajennus Mithril Emmet on asennettuna:

div.nav>ul>li\*3>a

<body>

    <div *class=*"nav">

        <h3>Osto ja myyntikanava</h3>

        <ul>

            <li><a *href=*"/">Etusivu</a></li>

            <li><a *href=*"/login">Kirjaudu</a></li>

            <li><a *href=*"/register">Rekisteröidy</a></li>

        </ul>

    </div>

    {{{body}}}

</body>

Muokataan **index.hbs**:

div.ilmoitus>h2+p

<div *class=*"ilmoitus">

    <h2>Ilmoituksen otsikko</h2>

    <p>Ilmoituksen kuvaus</p>

</div>

Muokataan styles.css:

body {

    background-color: #EAE7DC;

}

.nav {

    background-color: #D8C3A5;

    color: #E85A4F;

    margin: 0;

    padding: 30px 60px;

    display: flex;

    justify-content: space-between;

    border-radius: 5px;

}

.nav ul {

    display: flex;

    justify-content: space-around;

    align-items: center;

}

.nav li {

    list-style: none;

}

.nav li a {

    text-decoration: none;

    letter-spacing: 0.03em;

    color: #5f5e5c;

    font-weight: bold;

    padding: 5px 8px;

}

.nav li a:hover {

    text-decoration: none;

    color: #E85A4F;

}

.ilmoitus {

    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.6);

    background-color: #D8C3A5;

    color: #5f5e5c;

    max-width: 60%;

    margin: auto;

    text-align: center;

    border-radius: 5px;

}

Luodaan rekisteröitymissivu **views** kansioon, lisäämällä tiedosto **register.hbs**:

div>div.universalForm

form>div>label[for=userName]+input.form-control

div>label[for=email]+input:email.form-control

button:submit.button

<div>

    <div *class=*"universalForm">

        <h1>Rekisteröidy</h1>

        <form *action=*"">

            <div>

                <label *for=*"userName">Käyttäjätunnus: </label>

                <input *type=*"text" *name=*"kayttaja\_tunnus" id="kayttaja\_tunnus" *class=*"form-control"

*placeholder=*"Käyttäjätunnus" />

            </div>

            <div>

                <label *for=*"email">Sähköposti: </label>

                <input *type=*"email" *name=*"kayttaja\_sahkoposti" id="kayttaja\_sahkoposti" *class=*"form-control"

*placeholder=*"Sähköposti" />

            </div>

            <div>

                <label *for=*"password">Salasana: </label>

                <input *type=*"password" *name=*"kayttaja\_salasana" id="kayttaja\_salasana" *class=*"form-control"

*placeholder=*"Salasana" />

            </div>

            <button *type=*"submit" *class=*"button">Rekisteröidy</button>

        </form>

    </div>

</div>

Lisätään **style.css** tiedostoon tyyli:

.universalForm {

    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.6);

    background-color: #D8C3A5;

    color: #5f5e5c;

    max-width: 60%;

    margin: auto;

    text-align: center;

    border-radius: 5px;

}

Nyt kun sivuja alkaa tulla enemmän tehdään reiteille oma kansio/tiedosto:

Luo juureen kansio **routes**, jonka sisällä tiedosto **page.routes.js**:

import express from 'express';

const routes = express.Router();

routes.get('/', (req, res) => {

    res.render('index');

});

routes.get('/register', (req, res) => {

    res.render('register');

});

routes.get('/login', (req, res) => {

    res.render('login');

});

*export* default  routes;

Kokeile rekisteröidy linkkiä! Huomataan, että sivut eivät toimi, koska sovellus ei pääse käsiksi reitteihin

Korvaa app.get -> app.use **app.js** tiedostossa:

app.get('/', (req, res) => {

    res.render('index');

});

app.use('/', pages);

Lisää pages import **app.js** tiedoston importteihin:

import pages from './routes/page.routes.js';

Nyt rekisteröidy sivu toimii!

Luodaan routes kansioon **register.routes.js**:

import express from 'express';

import registerUser from '../controllers/user.controller.js';

const routes = express.Router();

routes.post('/', registerUser);

*export* default routes;

Luodaan juureen kansio **controllers**, jonka sisälle tiedosto **user.controller.js**:

const registerUser = (req, res) => {

    console.log(req.body);

    res.send("Form submitted");

}

*export* default registerUser;

Muokataan **register.hbs** teidostoa:

<form *action=*"/register" *method=*"POST">

Kokeillaan toimiiko rekisteröidy sivu -> saadaan virhe ” Cannot POST /register”! Keskustelua miksi näin ja miten voisi korjata?

Ratkaisu: lisätään app.js tiedostoon tarvittava reitti:

app.use('/register', register);

Lisätään import **app.js** tiedoston importteihin:

import register from './routes/register.routes.js';

Täytetään lomake ja kokeillaan rekisteröityä. Huomataan että sivulle ilmaantuu ”Form submitted”, mutta konsoliin tulee undefined,

Lisätään **app.js** tiedostoon urlencoded ja json:

app.set('view engine', 'hbs');

app.use(urlencoded({ extended: false }));

app.use(json());

app.use('/', pages);

Muutetaan expressin import muotoon:

import express, { urlencoded, json } from 'express';

Kokeillaan uudestaan rekisteröitymistä. Nyt nähdään konsolissa lomakkeella lähetetyt tiedot.

Lisätään vielä salasanan vahvistamiskenttä **register.hbs** tiedostoon buttonin yläpuolelle:

            <div>

                <label *for=*"password">Salasana: </label>

                <input *type=*"password" *name=*"kayttaja\_salasana\_varmistus" id="kayttaja\_salasana\_varmistus"

*class=*"form-control" *placeholder=*"Salasana uudelleen" />

            </div>

            <button *type=*"submit" *class=*"button">Rekisteröidy</button>

Lisätään tiedoston loppuun osio, jossa voidaan näyttää viestejä.

Tässä yhteydessä voi kertoa miten handlebareissa if rakenne toimii.

        </form>

    </div>

    <div>

        {{#if message}}

        <h4 *class=*"alert">{{message}}</h4>

        {{/if}}

    </div>

</div>

Muokataan **user.controller.js** tiedostoa:

const registerUser = (req, res) => {

    console.log(req.body);

    res.render('register', {

        message: " Lomake lähetetty!"

    });

}

*export* default registerUser;

Kokeillaan taas rekisteröitymistä ja tarkastellaan miten message toimii.

Muokataan **user.controller.js** tiedostoa siten, että se lisää käyttäjän tietokantaan:

HUOM! kommentoidut esimerkkinä, että asiat voi tehdä pitkän tai lyhyen kaavan kautta.

Tässä yhteydessä voidaan kysyä missä muuttujan message arvo tulee näkyviin.

import db from '../app.js';

const registerUser = (req, res) => {

*// const kayttaja\_tunnus = req.body.kayttaja\_tunnus;*

*// const kayttaja\_sahkoposti = req.body.kayttaja\_sahkoposti;*

*// const kayttaja\_salasana = req.body.kayttaja\_salasana;*

*// const kayttaja\_salasana\_varmistus = req.body.kayttaja\_salasana\_varmistus;*

    const {

        kayttaja\_tunnus,

        kayttaja\_sahkoposti,

        kayttaja\_salasana,

        kayttaja\_salasana\_varmistus,

    } = req.body;

    db.query(

        "SELECT kayttaja\_sahkoposti FROM kayttajat WHERE kayttaja\_sahkoposti = ?",

        [kayttaja\_sahkoposti],

        (error, results) => {

            if (error) {

                console.log("Error in query: " + error);

            }

            if (results.length > 0) {

                return res.render("register", {

                    message: "Sähköposti on jo käytössä!",

                });

            } else if (kayttaja\_salasana !== kayttaja\_salasana\_varmistus) {

                return res.render("register", {

                    message: "Salasanat eivät täsmänneet!",

                });

            }

        }

    );

}

*export* default registerUser;

Kokeile rekisteröitymissivun toiminnot (sähköpostilla, joka löytyy tietokannasta ja salasanan tarkistuksen toiminta). Jos tietokannassa ei ole vielä käyttäjiä, lisää yksi käyttäjä phpMyAdminin avulla.

Lisätään **user.controller.js** tiedostoon query (INSERT INTO) käyttäjän lisäämiseksi tietokantaan:

                    message: "Salasanat eivät täsmänneet!",

                });

            }

            db.query(

                "INSERT INTO kayttajat SET ?",

                {

                    kayttaja\_tunnus: kayttaja\_tunnus,

                    kayttaja\_sahkoposti: kayttaja\_sahkoposti,

                    kayttaja\_salasana: kayttaja\_salasana

                },

                (error, results) => {

                    if (error) {

                        console.log("Error in query: " + error);

                    } else {

                        return res.status(200).render('register', {

                            message: "Rekisteröityminen onnistui!",

                        });

                    }

                }

            );

Lisätään yksi käyttäjä ja varmistetaan phpMyAdminilla, että käyttäjän lisääminen onnistui. Huomataan, että salasana ei ole salattu. Voidaan antaa tehtäväksi etsiä sopiva salausmenetelmä. Esimerkissä käytetään bcryptiä.

Lisätään **user.controller.js** tiedostoon:

Alkuun importti

import bcrypt from 'bcryptjs';

Ennen INSERT INTO queryä:

let hashedPassword = await bcrypt.hash(kayttaja\_salasana, 8);

Queryssä muutetaan:

kayttaja\_salasana: hashedPassword

Tallennetaan ja huomataan, että sovellus kaatuu. Virheilmoituksista nähdään, että bcrypt hash on await joka toimii ainoastaan async function sisällä. Voi käydä keskustelua mihin async pitää lisätä ja miksi bcrypt on await.

Lisätään async funktion alkuun:

[kayttaja\_sahkoposti],

*async* (error, results) => {

Lisätään uusi käyttäjä ja varmistetaan tietokannasta, että salasana on kryptattu.

Lisätään kirjautumissivu.

Luodaan **views** kansioon tiedosto **login.hbs:**

<br>

<div *class=*"universalForm">

    <form *action=*"auth/login" *method=*"POST">

        <div *class=*"container">

            <div *class=*"two-col-header">

                <h1>Kirjaudu palveluun</h1>

            </div>

            <div *class=*"itemA">

                <label *for=*"email">Sähköposti: </label>

            </div>

            <div *class=*"itemB">

                <input *type=*"email" *class=*"form-control" id="kayttaja\_sahkoposti" *name=*"kayttaja\_sahkoposti"

*placeholder=*"Sähköposti" />

            </div>

            <div *class=*"itemA">

                <label *for=*"password">Salasana: </label>

            </div>

            <div *class=*"itemB">

                <input *type=*"password" *class=*"form-control" id="kayttaja\_salasana" *name=*"kayttaja\_salasana"

*placeholder=*"Salasana" />

            </div>

            <div *class=*"loginButton">

                <button *type=*"submit" *class=*"button">Kirjaudu</button>

            </div>

        </div>

    </form>

</div>

<div>

    {{#if message}}

    <h4 *class=*"alert">{{message}}</h4>

    {{/if}}

</div>

Lisätään tyylit styles.css tiedostoon:

.container {

    display: grid;

    grid-template-columns: auto auto;

    padding-top: 20px;

}

.itemA {

    text-align: right;

}

.itemB {

    text-align: left;

}

.two-col-header {

    grid-column-start: 1;

    grid-column-end: 3;

    grid-row-start: 0;

    grid-row-end: 1;

}

.loginButton {

    grid-column-start: 1;

    grid-column-end: 3;

    grid-row-start: 3;

    grid-row-end: 4;

}

Kokeillaan kirjautumista -> saadaan virhe: Cannot POST /auth/login

Lisätään **routes** kansioon tiedosto **auth.routes.js**:

import express from 'express';

import login from '../controllers/auth.controller.js';

const routes = express.Router();

routes.post('/login', login);

*export* default routes;

Lisätään **controllers** kansioon tiedosto **auth.controller.js**:

import db from '../app.js';

import bcrypt from 'bcryptjs';

import jwt from 'jsonwebtoken';

const login = *async* (req, res) => {

    try {

        const { kayttaja\_sahkoposti, kayttaja\_salasana } = req.body;

        if (!kayttaja\_sahkoposti || !kayttaja\_salasana) {

            return res.status(400).render('login', {

                message: "Anna käyttäjätunnus ja/tai salasana!",

            });

        }

        db.query(

            "SELECT \* FROM kayttajat WHERE kayttaja\_sahkoposti = ?",

            [kayttaja\_sahkoposti],

*async* (error, results) => {

                if (

                    !results ||

                    !(await bcrypt.compare(

                        kayttaja\_salasana,

                        results[0].kayttaja\_salasana

                    ))

                ) {

                    res.status(401).render('login', {

                        message: "Sähköposti tai salasana on virheellinen!",

                    });

                } else {

                    const id = results[0].kayttaja\_id;

                    const token = jwt.sign({ id }, process.env.JWT\_SECRET, {

                        expiresIn: process.env.JWT\_EXPIRES\_IN,

                    });

                    const cookieOptions = {

                        expires: new Date(

                            Date.now() + process.env.JWT\_COOKIE\_EXPIRES \* 24 \* 60 \* 60 \* 1000

                        ),

                        httpOnly: true,

                    };

                    res.cookie('jwt', token, cookieOptions);

                    res.status(200).redirect('/');

                }});

    } catch (error) {

        console.log(error);

    }};

*export* default login;

Huomataan, että tiedostossa on kolme uutta **process.env** muuttujaa:

process.env.JWT\_SECRET

process.env.JWT\_EXPIRES\_IN

process.env.JWT\_COOKIE\_EXPIRES

Lisätään nämä **.env-**tiedostoon

JWT\_SECRET = mysupersecretpassword

JWT\_EXPIRES\_IN = 90d

JWT\_COOKIE\_EXPIRES = 90

Kun yritetään kirjautua, saadaan virhe ”Cannot POST auth/login”. Mietintään miten ratkaistaan.

Lisätään reitti **app.js** tiedostoon muiden reittien joukkoon:

app.use('/auth', auth);

Ja sille tarvittava importti:

import auth from './routes/auth.routes.js';

Avataan kehittäjätyökalusta välilehti ”Application ja varmistetaan, että kekseistä löytyy jwt tokeni

Kuva, joka sisältää kohteen pöytä

Kuvaus luotu automaattisesti

Kokeillaan kirjautua sisään väärällä tunnuksella / salasanalla ja sen jälkeen kokeillaan kirjautua uudelleen. Saadaan virhe ”Cannot POST auth/auth/login”. Korjataan **login.hbs**:

<form *action=*"/auth/login" *method=*"POST">

Aiemmassa osoite muotoa ”auth/login” -> voidaan miettiä virheellistä osoitteen muodostumista.

Muokataan **main.hbs** tiedostoa siten, että jatkossa kirjautuneelle ja kirjautumattomalle näkyy eri linkit.

        <ul>

            <li><a *href=*"/">Etusivu</a></li>

            {{#if user}}

                <li><a *href=*"/auth/logout">Kirjaudu ulos</a></li>

            {{else}}

                <li><a *href=*"/login">Kirjaudu</a></li>

                <li><a *href=*"/register">Rekisteröidy</a></li>

            {{/if}}

        </ul>

Tehdään kontrolleri isLoggedIn **auth.controller.js** tiedostoon:

const isLoggedIn = *async* (req, res, next) => {

    if (req.cookies.jwt) {

        try {

            const decoded = await promisify(jwt.verify)(

                req.cookies.jwt,

                process.env.JWT\_SECRET

            );

*//Check if user exist*

            db.query(

                "SELECT \* FROM kayttajat WHERE kayttaja\_id = ?",

                [decoded.id],

                (error, results) => {

                    if (!results) {

                        return next();

                    }

                    req.user = results[0];

                    return next();

                }

            );

        } catch (error) {

            console.log("Error in login query: " + error);

            return next();

        }

    } else {

        next();

    }

};

Lisätään alkuun import promisifylle ja loppuun isLoggedInille export:

import { promisify } from 'util';

...

*export* default login;

*export* { isLoggedIn };

Muokataan **page.routes.js** tiedosto:

Lisätään import:

import { isLoggedIn } from '../controllers/auth.controller.js';

Muokataan **routes.get**:

routes.get('/', isLoggedIn, (req, res) => {

    res.render('index', {

        user: req.user

    });

});

Lisätään **app.js** tiedostoon cookieparser:

import cookieParser from 'cookie-parser';

app.use(json());

app.use(cookieParser());

app.use('/', pages);

Kirjaudutaan sisään ja yläpalkissa on linkit etusivulle ja uloskirjautumiseen.

Lisätään **auth.controller.js** tiedostoon logout funktio ja sille export:

const logout = *async* (req, res) => {

    res.cookie('jwt', 'logout', {

        expires: new Date(Date.now() + 2 \* 1000),

        httpOnly: true,

    });

    res.status(200).redirect('/');

};

*export* default login;

*export* { isLoggedIn, logout };

Lisätään **auth.routes.js** tiedostoon reitti logoutille

routes.get('/logout', logout);

Muokataan import:

import login, { logout } from '../controllers/auth.controller.js';

Tässä vaiheessa voi kertoa, miksi logout importataan tällä tyylillä vs loginin importtaus.

Kokeillaan että uloskirjaus toimii ja varmistetaan kehittäjätyökalun avulla, että jwt tokeni, on hävinnyt.

Tehdään linkki ilmoituksen lisäämiseksi **main.hbs** tiedostoon:

HUOM! Ilmoituksen voi lisätä vain rekisteröitynyt käyttäjä. Voidaan käydä keskustelua mihin kohtaan linkki tulee lisätä.

{{#if user}}

                <li><a *href=*"/newAdvert">Lisää ilmoitus</a></li>

                <li><a *href=*"/profile">Profiili</a></li>

Ilmoituksen laatimista varten luodaan **views** kansioon tiedosto **new-advert.hbs**:

<div>

    <h1 *style=*"text-align: center;">Lisää ilmoitus</h1>

    <form *action=*"/newAdvert/" *method=*"POST">

        {{> advert-form}}

    </form>

</div>

Luodaan **views** kansioon **partials** kansio, jonka sisälle tiedosto **advert-form.hbs**:

<div *class=*"universalForm">

    <div *class=*"container">

        <div *class=*"itemA">

            <label *for=*"ilmoitus\_laji">Ilmoituksen tyyppi: </label>

        </div>

        <div *class=*"itemB">

            <select *class=*"form-control" id="ilmoitus\_laji" *name=*"ilmoitus\_laji">

                <option *value=*"1">{{advertType advert.ilmoitus\_laji}}</option>

                {{#if (selectAdvertType advert.ilmoitus\_laji)}}

                <option *value=*"1">Myydään</option>

                {{else}}

                <option *value=*"2">Ostetaan</option>

                {{/if}}

            </select>

        </div>

        <div *class=*"itemA">

            <label *for=*"ilmoitus\_nimi">Ilmoitusksen nimi: </label>

        </div>

        <div *class=*"itemB">

            <input *type=*"text" *class=*"form-control" id="ilmoitus\_nimi" *name=*"ilmoitus\_nimi"

*value=*"{{advert.ilmoitus\_nimi}}" *placeholder=*"Lisää ilmoituksen nimi" />

        </div>

        <div *class=*"itemA">

            <label *for=*"ilmoitus\_kuvaus">Ilmoituksen kuvaus:</label>

        </div>

        <div *class=*"itemB">

            <textarea *class=*"form-control" id="ilmoitus\_kuvaus" *name=*"ilmoitus\_kuvaus"

*placeholder=*"Lisää kuvaus">{{advert.ilmoitus\_kuvaus}}</textarea>

        </div>

        <div *class=*"editAdvertButton">

            <button *type=*"submit" *class=*"button">Tallenna</button>

        </div>

    </div>

</div>

Tässä yhteydessä voidaan käydä läpi handlebarin partials ominaisuus (vrt. Reactin komponentit)

Lisätään **page.routes.js** tiedostoon uusi reitti:

routes.get('/newAdvert', isLoggedIn, (req, res) => {

    res.render('new-advert', {

        user: req.user

    });

});

Kokeillaan lomakkeen toimintaa -> saadaan virhe ” Error: Missing helper: "advertType"

Tutustutaan miten handlebareihin voidaan tehdä omia helppereitä.

Luodaan projektin juureen kansio **helpers** jonka sisälle tiedosto **handlebars-helpers.js:**

let hbsHelpers = {

    advertType: (ilmoitus\_laji) => {

        let ilmoitusLaji = "";

        if (ilmoitus\_laji === 2) {

            ilmoitusLaji = "Ostetaan";

        } else {

            ilmoitusLaji = "Myydään";

        }

        return ilmoitusLaji;

    },

    selectAdvertType: (ilmoitus\_laji) => {

        let ilmoitusLajiBoolean = false;

        if (ilmoitus\_laji === 2) {

            ilmoitusLajiBoolean = true;

        }

        return ilmoitusLajiBoolean;

    }

}

*export* default hbsHelpers;

Lisätään **app.js** tiedostoon import:

import hbsHelpers from './helpers/handlebars-helpers.js';

ja lisätään hbs muuttujaan helpers:

const hbs = create({

    extname: '.hbs',

    helpers: hbsHelpers

});

Lisätään tyylitiedostoon **styles.css** uusi luokka:

.editAdvertButton {

    grid-column-start: 1;

    grid-column-end: 3;

    grid-row-start: 4;

    grid-row-end: 5;

}

Lisätään **page.routes.js** tiedostoon tarvittavat reitit ja importti:

import newAdvert from '../controllers/advert.controller.js';

routes.get('/newAdvert', isLoggedIn, (req, res) => {

    res.render('new-advert', {

        user: req.user

    });

});

routes.post('/newAdvert', [isLoggedIn, newAdvert], (req, res) => {

    res.render('new-advert', {

        user: req.user

    });

});

Luodaan **controllers** kansioon tiedosto **advert.controller.js**:

import db from '../app.js';

import jwt from 'jsonwebtoken';

import { promisify } from 'util';

const newAdvert = *async* (req, res, next) => {

    const { ilmoitus\_laji, ilmoitus\_nimi, ilmoitus\_kuvaus } = req.body;

    const decoded = await promisify(jwt.verify)(

        req.cookies.jwt,

        process.env.JWT\_SECRET

    );

    db.query(

        "INSERT INTO ilmoitukset SET ?",

        {

            ilmoitus\_laji,

            ilmoitus\_nimi,

            ilmoitus\_kuvaus,

            ilmoitus\_paivays: new Date(Date.now()),

            ilmoittaja\_id: decoded.id,

        },

        (error, results) => {

            if (error) {

                console.log("Error in query: " + error);

            } else {

                res.redirect("/");

            }

        }

    );

};

*export* default newAdvert;

Varmistetaan tietokannasta, että ilmoituksen luominen onnistuu.

Lisätään ilmoituksien listaus etusivulle. Tyhjennä **index.hbs** sisältö ja muokkaa muotoon:

<div>

    {{> list-adverts}}

</div>

Luodaan **partials** kansioon tiedosto **list-adverts.hbs**:

{{#each list}}

<div *class=*"ilmoitukset">

    <div *class=*"container-adverts">

        <div *class=*"ilmoitus" *key=*{{ilmoitus\_id}}>

            <div>

                <h1>{{ilmoitus\_laji}}: {{ilmoitus\_nimi}} </h1>

                <h3>{{ilmoitus\_kuvaus}}</h3>

            </div>

            <div>

            </div>

            <div *class=*"container-advertBottom">

                <div *class=*"itemB">

                    <h5>Ilmoittaja: {{ilmoittaja\_id}}</h5>

                </div>

                <div *class=*"itemA">

                    <h5>Ilmoitus jätetty: {{ilmoitus\_paivays}}</h5>

                </div>

            </div>

        </div>

    </div>

</div>

{{/each}}

Lisätään **advert.controller.js** tiedostoon ja lisätään export:

const listAdverts = *async* (req, res, next) => {

    try {

        db.query("SELECT \* FROM ilmoitukset", *async* (error, results) => {

            req.list = results;

            next();

        });

    } catch (error) {

        console.log(error);

    }

};

*export* default newAdvert;

*export* { listAdverts };

Muokataan **page.routes.js** tiedostoa:

Muokataan import:

import newAdvert, { listAdverts } from '../controllers/advert.controller.js';

Lisätään etusivun reittiin listAdverts

routes.get('/', [ isLoggedIn, listAdverts ], (req, res) => {

    res.render('index', {

        user: req.user,

        list: req.list

    });

});

Ilmoitukset listautuvat nyt etusivulle.

Lisätään tyylit styles.css tiedostoon

.ilmoitus h1 {

    border-radius: 5px;

    letter-spacing: 0.05em;

    background-color: #bba688;

    color: #E85A4F;

}

.ilmoitus h3 {

    color: #5f5e5c;

}

.ilmoitukset {

    background-color: #EAE7DC;

}

.container-advertBottom {

    display: grid;

    grid-template-columns: auto auto;

    padding-left: 20px;

    padding-right: 20px;

}

Huomataan, että ilmoituksissa näkyy osa tiedoista numero arvoina ja ilmoituksen jättämisen ajankohdan muotoilu ei ole kunnossa.

Mietitään, miten saadaan ilmoituksen laji näkyviin -> hyödynnetään aiemmin tehtyä helpperiä **advertType,** muokataan **list-adverts.hbs** tiedoston ilmoituslaji kohta muotoon:

 <h1>{{advertType ilmoitus\_laji}}: {{ilmoitus\_nimi}} </h1>

Voidaan antaa tehtäväksi selvittää, miten ilmoituksen laatimispäivämäärä saadaan muotoiltua järkevästi.

Lisätään päivämäärän muotoileva helpperi **handlebars-helpers.js** tiedostoon:

formatDate: (ilmoitus\_paivays) => {

        let dateFromDb = ilmoitus\_paivays;

        let year = dateFromDb.getFullYear();

        let month = ('0' + (dateFromDb.getMonth() + 1)).slice(-2);

        let date = ('0' + dateFromDb.getDate()).slice(-2);

        return date + '.' + month + '.' + year;

    }

ja muokataan **list-adverts.hbs** tiedostossa ilmoitus jätetty muotoon:

<h5>Ilmoitus jätetty: {{formatDate ilmoitus\_paivays}}</h5>

Saadaksemme ilmoituksen laatijan näkyviin tehdään ensin kontrolleri **user.controller.js** tiedostoon ja lisätään sille export:

const listUsers = *async* (req, res, next) => {

    try {

        db.query("SELECT \* FROM kayttajat", *async* (error, results) => {

            req.users = results;

            next();

        });

    } catch (error) {

        console.log(error);

    }

};

*export* default registerUser;

*export* { listUsers };

Otetaan kontrolleri käyttöön **page.routes.js** tiedostossa:

import { listUsers } from '../controllers/user.controller.js';

routes.get('/', [ isLoggedIn, listAdverts, listUsers ], (req, res) => {

    res.render('index', {

        user: req.user,

        users: req.users,

        list: req.list

    });

});

Lisätään helpperi **handlebars-helpers.js** tiedostoon (hyvä harjoitus opiskelijoille mietittäväksi miten toteutetaan):

getUser: (users, kayttaja) => {

        let user = "";

        users.forEach((u) => {

            if (u.kayttaja\_id === kayttaja) {

                user = u.kayttaja\_tunnus;

            }

        });

        return user;

    },

Muokataan **list-adverts.hbs** tiedostossa ilmoittaja kohta muotoon:

<h5>Ilmoittaja: {{getUser ../users ilmoittaja\_id}}</h5>

Käydään läpi miksi pitää olla **../users**

Ilmoitukset listautuvat nyt vanhimmasta uusimpaan. Tehdään helpperi tiedostoon **handlebars-helpers.hbs**, joka kääntää listan uusimmasta vanhimpaan:

reverseList: (list) => {

        let reversedList = list.reverse();

        return reversedList;

    }

Muokataan vielä **list-adverts.hbs** tiedostossa:

{{#each (reverseList list)}}

Tehdään käyttäjälle profiilisivu omien tietojen ja ilmoituksien hallintaa varten.

Luodaan **views** kansioon tiedosto **profile.hbs:**

<div *class=*"profile">

    <div *class=*"container">

        <div *class=*"itemB">

            <h4>Kirjautuneena: {{user.kayttaja\_tunnus}}</h4>

            <h4>Sähköposti: {{user.kayttaja\_sahkoposti}}</h4>

            <h4>Omia ilmoituksia: {{list.length}}kpl</h4>

        </div>

        <div *class=*"itemA">

            <a *href=*"/editUser/{{user.kayttaja\_id}}"><button *class=*"button">Muokkaa omia tietoja</button></a>

        </div>

    </div>

</div>

<div *class=*"row">

    <div *class=*"col-8">

        <h2 *style=*"text-align: center;">Omat ilmoitukset</h2>

        <div>

            {{#each (reverseList list)}}

            <div *class=*"ilmoitukset">

                <div *class=*"ilmoitus" *key=*{{ilmoitus\_id}}>

                    <div>

                        <h1>{{advertType ilmoitus\_laji}}: {{ilmoitus\_nimi}}</h1>

                        <h3>{{ilmoitus\_kuvaus}}</h3>

                        <h5>Ilmoitus jätetty:{{formatDate ilmoitus\_paivays}}</h5>

                        <a *href=*"/editAdvert/{{ilmoitus\_id}}"><button *class=*"button">Muokkaa</button></a>

                        <a *href=*"/deleteAdvert/{{ilmoitus\_id}}"><button *class=*"button">Poista</button></a>

                    </div>

                </div>

            </div>

            {{/each}}

        </div>

    </div>

</div>

Lisätään linkki profiiliin **main.hbs** tiedostoon:

            {{#if user}}

            <li><a *href=*"/profile">Profiili</a></li>

Tehdään kontrolleri käyttäjän ilmoituksien listaamista varten **advert.controller.js** tiedostoon ja lisätään export:

const listUserAdverts = *async* (req, res, next) => {

    try {

        const decoded = await promisify(jwt.verify)(

            req.cookies.jwt,

            process.env.JWT\_SECRET

        );

        db.query(

            "SELECT \* FROM ilmoitukset WHERE ilmoittaja\_id = ?",

            [decoded.id],

*async* (error, results) => {

                req.list = results;

                next();

            }

        );

    } catch (error) {

        console.log(error);

    }

};

*export* default newAdvert;

*export* { listAdverts, listUserAdverts };

Lisätään reitti **page.routes.js** tiedostoon ja importataan kontrolleri:

import newAdvert, { listAdverts, listUserAdverts } from '../controllers/advert.controller.js';

routes.get('/profile', [isLoggedIn, listUserAdverts], (req, res) => {

        if (req.list) {

            res.render('profile', {

                list: req.list,

                user: req.user

            });

        } else {

            res.redirect('/login');

        }

    }

);

Lisätään buttoneille ja käyttäjän tiedoille tyylit **styles.css** tiedostoon:

button {

    display: inline-block;

    padding: 0.35em 1.2em;

    border: 0.1em solid #5f5e5c;

    margin: 0.5em;

    border-radius: 0.25em;

    box-sizing: border-box;

    text-decoration: none;

    font-family: OpenDyslexic;

    font-weight: 300;

    background: #bba688;

    text-align: center;

    transition: all 0.2s;

}

button:hover {

    color: #000;

    background-color: #EAE7DC;

}

.profile {

    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.6);

    background-color: #D8C3A5;

    color: #5f5e5c;

    margin: 30px auto;

    padding: 10px;

    max-width: 40%;

    border-radius: 5px;

    text-align: center;

}

Ilmoituksen muokkaamista varten luodaan **views** kansioon tiedosto **edit-advert.hbs**:

<div>

    <h1 *style=*"text-align: center;">Muokkaa ilmoitusta</h1>

    <form *action=*"/editAdvert/{{advert.ilmoitus\_id}}" *method=*"POST">

        {{> advert-form}}

    </form>

</div>

Lisätään tarvittavat kontrollerit **advert.controller.js** tiedostoon:

const getAdvert = *async* (req, res, next) => {

    try {

        db.query(

            "SELECT \* FROM ilmoitukset WHERE ilmoitus\_id = ?",

            [req.params.id],

*async* (error, results) => {

                req.advert = results[0];

                next();

            }

        );

    } catch (error) {

        console.log(error);

    }

};

const updateAdvert = *async* (req, res, next) => {

    const { ilmoitus\_kuvaus, ilmoitus\_nimi } = req.body;

    try {

        db.query(

            "UPDATE ilmoitukset SET ilmoitus\_kuvaus = ?, ilmoitus\_nimi = ? WHERE ilmoitus\_id = ?",

            [ilmoitus\_kuvaus, ilmoitus\_nimi, req.params.id],

*async* (error, results) => {

                db.query(

                    "SELECT \* FROM ilmoitukset WHERE ilmoitus\_id = ?",

                    [req.params.id],

*async* (error, results) => {

                        req.advert = results[0];

                        next();

                    }

                );

            }

        );

    } catch (error) {

        console.log(error);

        next();

    }

};

*export* default newAdvert;

*export* { listAdverts, listUserAdverts, getAdvert, updateAdvert };

Lisätään tarvittavat reitit **page.routes.js** tiedostoon:

routes.get('/editAdvert/:id', [isLoggedIn, getAdvert], (req, res) => {

    const user = req.user;

    const advert = req.advert

    if (advert) {

        res.render('edit-advert', {

            user,

            advert

        });

    } else {

        res.redirect('/login');

    }

});

routes.post('/editAdvert/:id', [isLoggedIn, updateAdvert], (req, res) => {

    const user = req.user;

    const advert = req.advert

    if (advert) {

        res.render('edit-advert', {

            user,

            advert

        });

    } else {

        res.redirect('/login');

    }

});

Muokataan importti:

import newAdvert, {

    listAdverts,

    listUserAdverts,

    getAdvert,

    updateAdvert

} from '../controllers/advert.controller.js';

Kokeillaan ilmoituksen poistaminen -> saadan virheilmoitus Cannot GET /deleteAdvert/27

Lisätään kontrolleri **advert.controller.js** tiedostoon:

const deleteAdvert = *async* (req, res, next) => {

    try {

        db.query(

            "DELETE FROM ilmoitukset WHERE ilmoitus\_id = ?",

            [req.params.id],

*async* (error, results) => {

                res.redirect('/profile');

                next();

            }

        );

    } catch (error) {

        console.log(error);

        next();

    }

};

*export* default newAdvert;

*export* { listAdverts, listUserAdverts, getAdvert, updateAdvert, deleteAdvert };

Lisätään reitti **page.routes.js** tiedostoon (HUOM! muista lisätä import):

routes.get('/deleteAdvert/:id', deleteAdvert);

Kokeillaan, että ilmoituksen poistaminen onnistuu.

Tehdään vielä viestien hakutoiminto.

Muokataan **partials** kansiossa olevaa **list-adverts.hbs** tiedostoa:

Lisätään alkuun

{{#if list}}

ja loppuun

{{/each}}

{{else}}

<div>

    <p>

    <h2 *style=*"text-align: center;">Haku ei tuottanut yhtään tulosta :/</h2>

    </p>

</div>

{{/if}}

Lisätään **index.hbs** tiedoston alkuun hakukenttä ja jätetään ilmoituksien listaus tiedoston loppuun:

<div *class=*"searchBar">

    <form *action=*"/" *method=*"POST">

        <div *class=*"container-search">

            <div *class=*"itemC">

                <label *for=*"searchAdvert">Hae ilmoituksista: </label>

            </div>

            <div *class=*"itemD">

                <input *type=*"text" id="searchAdvert" *name=*"searchAdvert" *placeholder=*"Kirjoita hakusana" />

            </div>

            <div *class=*"itemB">

                {{#if (isEmpty searchAdvert)}}

                <button *type=*"submit" *class=*"button">Hae</button><button *type=*"submit" *class=*"button">Tyhjennä</button>

                {{else}}

                <button *type=*"submit" *class=*"button">Hae</button>

                {{/if}}

            </div>

        </div>

    </form>

</div>

<div>

    {{> list-adverts}}

</div>

Lisätään helpperi **handlebars-helpers.js** tiedostoon:

isEmpty: (searchAdvert) => {

    let isEmptyBoolean = false;

    if (searchAdvert) {

        isEmptyBoolean = true;

    }

    return isEmptyBoolean;

}

Lisätään tyylit **styles.css** tiedostoon:

.searchBar {

    padding: 20px;

    text-align: center;

}

.container-search {

    display: grid;

    grid-template-columns: auto auto auto;

    padding-top: 20px;

}

.itemC {

    padding-top: 0.5em;

    text-align: right;

}

.itemD {

    padding-top: 0.4em;

    text-align: left;

}

Muokataan ilmoituksien listaus kontrolleria **advert.controller.js** tiedostossa:

const listAdverts = *async* (req, res, next) => {

    let { searchAdvert } = req.body;

    if (searchAdvert === "") {

        searchAdvert = undefined;

    }

    try {

        if (searchAdvert != undefined) {

            db.query(

                "SELECT \* FROM ilmoitukset WHERE MATCH(ilmoitus\_nimi, ilmoitus\_kuvaus) AGAINST (? IN BOOLEAN MODE)",

                [searchAdvert + "\*"],

*async* (error, results) => {

                    req.list = results;

                    req.searchAdvert = searchAdvert;

                    next();

                });

        } else {

            db.query("SELECT \* FROM ilmoitukset", *async* (error, results) => {

                req.list = results;

                next();

            });}

    } catch (error) {

        console.log(error);

    }};

Lisätään ilmoituksen haulle (post) reitti **page.routes.js** tiedostoon:

routes.post('/', [isLoggedIn, listAdverts, listUsers], (req, res) => {

    res.render('index', {

        user: req.user,

        users: req.users,

        list: req.list,

        searchAdvert: req.searchAdvert

    });

});

Muokataan myös (get):

routes.get('/', [isLoggedIn, listAdverts, listUsers], (req, res) => {

    res.render('index', {

        user: req.user,

        users: req.users,

        list: req.list,

        searchAdvert: req.searchAdvert

    });

});