Vanessa Donkor

WEB-SOVELLUS: ELOKUVAKATALOGI

Tekniikka
2023

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Vanessa Donkor
Opinnäytetyön nimi	Web-sovellus: Elokuvakatalogi
Vuosi	2023
Kieli	suomi
Sivumäärä	40
Ohjaaja	Mikael Jakas

Tässä työssä toteutetaan web-sovellus, jolla voidaan ylläpitää omaa henkilökohtaista katalogia katsotuista elokuvista sekä elokuvia, joita haluaa katsoa tulevaisuudessa. Työssä tutustutaan valittuihin työkaluihin ja toteutusmenetelmiin sekä käydään läpi, miten sovellusta on tarkoitus käyttää ja miten se toimii.

Toteutusmenetelmiksi on valittu ohjelmoida sovellus käyttäen HTML-, CSS- ja Python- ohjelmointikieliä ja tietokantana on valittu MongoDB. Työssä käydään läpi, miksi kyseiset menetelmät on valittu ja miten niitä hyödyntäen sovellus on toteutettu.

Kirjauduttuaan tai luotuaan uuden käyttäjätunnuksen, käyttäjä pääsee tallentamaan itse haluamiaan elokuvia sovellukseen. Näistä elokuvista muodostuu taulukko, josta käyttäjä voi painaa yksittäisen elokuvan kohdalta uudelle sivulle. Tältä sivulta hän pystyy lisäämään erilaisia tietoja elokuvalle, kuten arvosanan tai milloin on viimeksi katsonut sen.

Avainsanat[1] HTML, CSS, Python, ohjelmointi

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Vanessa Donkor
Title	Web Application: Movie Catalog
Year	2023
Language	Finnish
Pages	40
Name of Supervisor	Mikael Jakas

The purpose of this thesis is to develop a web application that allows the user to keep a personal catalog of films that they either have watched or want to watch in the future. The thesis goes through the different methods that have been used to develop the application, and how the application is supposed to be used and how it works.

The application was developed by using HTML, CSS, and Python as programming languages. For the database, MongoDB was used. The thesis goes through why said methods were chosen and how by utilizing them, the application was developed.

After the user logs in or creates a new account, they can add new films. The films are displayed in a table from which the user can click to go to a new page where they can add additional information about the film such as a rating and the last watched date.

Keywords HTML, CSS, Python, MongoDB

SISÄLLYS

KUVALUETTELO

1 JOHDANTO

Nykyään ihmisten jokapäiväisessä elämässä on todella paljon eri informaatiota muistettavana eri asioihin liittyen. Sen sijaan, että kirjaisi tämän kaiken muistettavan, vaikka paperille ylös, niin voidaan hyödyntää teknologiaa. Suurella osalla ihmisistä on nykyään jokin laite, jolla on pääsy internettiin ja nimenomaan laite, josta löytyy tai on ainakin ladattavissa verkkoselain. Juuri tästä syystä web-kehitys ja web-sovellukset ovat yleinen keino toteuttaa eri ratkaisuja, koska sillä voidaan tavoittaa paljon erilaisia käyttäjiä. Tyypillisesti web-sovelluksia tehdään JavaScript -ohjelmointikielellä, mutta tässä työssä keskitytään siihen, miten web-sovelluksen voi toteuttaa Pythonilla yhdessä HTML:n ja CSS:n kanssa.

1.1 Yleiskatsaus

Jatkuvasti julkaistaan uusia elokuvia, joita ihmiset haluavat katsoa. Samalla, kun tulee uusia julkaisuja, on silti vielä lukuisia aikaisempia julkaisuja, joita voidaan haluta katsoa uudelleen. Tätä varten voisi olla hyödyllistä olla jokin keino tai systeemi, millä pystyy tarkkailemaan, mitä elokuvia on jo katsonut, mitä haluaa katsoa uudelleen sekä mitä toivoo katsovansa tulevaisuudessa. Web-sovellus, jolla käyttäjä voisi ylläpitää katalogia elokuvista olisi kätevä tapa tarkkailla omia lempielokuviaan.

1.2 Tavoitteet

Tämän työn tavoite on toteuttaa web-sovellus, jolla käyttäjä voi ylläpitää omaa persoonallista elokuvakatalogia. Sovelluksessa käyttäjä pystyy autentikoimaan itsensä, rekisteröinti ja sisäänkirjautuminen. Käyttäjä pystyy itse lisäämään haluamiaan elokuvia ja niihin liittyviä tietoja täyttämällä lomakkeen. Sovellus esittää elokuvat visuaalisesti näyttävällä tavalla taulukossa. Sovellus muistaa, mitä elokuvia käyttäjä on lisännyt eli, kun käyttäjä kirjautuu sisään uudelleen, vain hänen lisää-

mänsä elokuvat näkyvät. Tiedot tallennetaan tietokantaan. Käyttäjälle syntyy persoonallinen katalogi elokuvista ja hän pystyy ylläpitämään tietoja, kuten esimerkiksi milloin on viimeksi katsonut elokuvan ja minkä arvosanan on antanut sille. Sovelluksen idea on olla yksinkertainen, käyttäjäystävällinen, moderni ja hauska tapa ylläpitää omaa henkilökohtaista elokuvakatalogia.

2 TYÖKALUT JA TOTEUTUSMENETELMÄT

Työssä tehdään web-sovellus käyttäen HTML-, CSS- ja Python- ohjelmointikieliä. Tässä luvussa tutustutaan tarkemmin valittuihin työkaluihin ja toteutusmenetelmiin.

2.1 HTML & CSS

HTML eli HyperText Markup Language on merkintäkieli, jota käytetään web-sivun rakenteen luomiseen. Sillä kerrotaan selaimelle, mitä erilaisia elementtejä halutaan nettivisulta löytyvän. Lisäksi sillä määritetään, mikä on näiden eri sisältöjen, jota nettisivulta löytyy, tarkoitus. Esimerkiksi, mikä osa sisällöstä on otsikko, kappale, kuva jne. Eli siis, että selaimet osaavat lukea ja tulkita nettisivun sisältöä tarkoitetulla tavalla. **(Kuva 1.)** /1/

```
<!DOCTYPE html>
<html lang="en">
<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">
    <title>Example of a HTML Boilerplate</title>
</head>
<body>

</body>
</html>
```

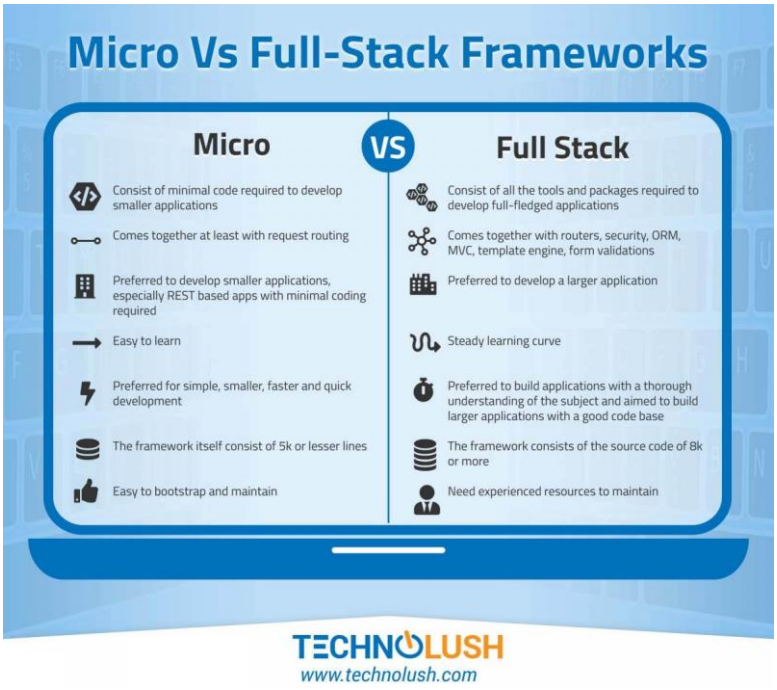
Kuva 1. HTML Boilerplate

CSS eli Cascading Style Sheets on tapa määrittää dokumentille tai verkkosivuille tyyliohjeita. Sitä käytetään kuvaamaan selaimelle, miltä halutaan HTML-elementtien ja sisällön näyttävän, miten halutaan esittää sisältö käyttäjälle. CSS avulla voidaan määritellä esimerkiksi sisällön värejä, fontteja ja layout. /2/

Lyhyesti sanottuna HTML on sivun rakenne ja CSS on se, miltä sivu näyttää visuaalisesti käyttäjälle. Tässä työssä käytetään pääasiallisesti puhdasta HTML:lää ja CSS:sää ilman mitään erillisiä kirjastoja.

2.2 Python & Flask

Python on yleinen ohjelmointikieli moniin eri tarkoituksiin. Tässä työssä Pythonia käytetään vastaamaan web-sovelluksen toiminnallisuudesta ja kommunikointiin tietokannan kanssa. Pythonin ohella käytetään Flaskia, jotta voidaan toteuttaa web-sovellus. Flask on Pythonille kehitetty mikroverkkokehys. Mikroverkkokehykset usein määritellään sillä, että niissä vähemmän kuin 5000 riviä koodia eli ne ovat pienempiä kuin full-stack ohjelmistokehykset. **(Kuva 2.)** Jos kehitettävä sovellus ei ole todella laaja kooltaan, niin pelkkä mikroverkkokehys voi olla riittävä projektille. /3/



Kuva 2. Mikroverkkokehys Vs Full-Stack Ohjelmistokehys /3/

2.3 MongoDB

Tarvitaan jokin tietokanta, jotta käyttäjätiedot sekä käyttäjän syöttämä data web-sovellukseen voitaisiin tallentaa jonnekin, että seuraavan kerran kun käyttäjä käyttää web-sovellusta, se muistaa käyttäjän. Tähän tarkoitukseen tässä projektissa on valittu käytettäväksi tietokannaksi MongoDB, joka on dokumenttitietokanta. Se valittiin yksinkertaisuuden ja yhteensopivuuden takia. /4/

MongoDB:n valinta projektiin esimerkiksi MySQL:än sijaan on mm. sen joustavuuden takia. MySQL:ssä, joka on relaatiotietokanta, tieto tallennetaan käyttäen tauluja ja rivejä. Näiden taulujen välille luodaan yhteyksiä. Tämä vahvistaa esimerkiksi turvallisuutta, mutta relaatiokannoissa kuitenkin tietomalli tulee määrittää etukäteen ja tiedon on vastattava täysin tätä mallia, jotta se voidaan tallentaa kantaan. Tämä puolestaan vähentää joustavuutta sen suhteen, mitä tietoja halutaan lisätä eri vaiheissa. MongoDB:ssä tieto tallennetaan JSON tyylisesti niin sanottuihin dokumentteihin. Nämä dokumentit sisältävät sarjan avain/arvo -pareja, jotka voivat vaihdella tyypeiltään, mukaan lukien taulukot ja sisäkkäiset dokumentit. Varsinainen pääero on se, että näiden avain/arvo -parien rakenne voivat vaihdella lukuisilla eri tavoilla dokumenttien välillä. Dokumentit ovat ns. itseään kuvaavia. /5/

3 KEHITYSYMPÄRISTÖN ASETTELU

3.1 Web-selain

Mitä tulee web-kehitykseen, niin on hyvä jo ensimmäisenä pitää mielessä, mitä selainta aikoo käyttää projektin aikana, koska sovellus pyörii selaimessa. Myös on hyvä varmistaa, että selain on päivitetty uusimpaan versioon. Tyypillisimmät selainvalinnat usein ovat Google Chrome tai muut Chromium-pohjaiset selaimet, kuten Brave. Vaihtoehtoisesti myös Mozilla Firefox on yleinen selain. Näistä vaihtoehdoista löytyy kätevästi ”Dev Tools”, joka on hyödyllinen työkalu devaajalle nähdä ja kokeilla, miltä koodi näyttää selaimessa ja eri kokoisilla laitteilla, koska huomioon on hyvä ottaa myös se millä laitteella käyttäjä selaa sivua.

Web-sovellusta kehittäessä on hyvä ottaa huomioon, että sovelluksen ulkonäkö ja toiminta saattaa hieman vaihdella eri selainten välillä. Jokaisella selaimella on omat oletustyylit ja -muotoilut. Yksi tapa miten varmistaa, että sovellus on yhtenäinen eri selainten välillä, on käyttää CSS Reset -tyylisivua projektissa. Sen idea on poistaa selaimen oletustyylit ja -muotoilut sekä muut mahdolliset epäjohdonmukaisuudet. Tämän ohella projektissa on tavalliseen tapaan omat luodut tyylit sovellukselle. Lisäksi koska eri selainvaihtoehtoja on niin monta, pienemmässä projektissa voisi riittää, että testataan pelkästään yleisimpi selaimia. Tämän voi toteuttaa manuaalisesti käymällä läpi toimiiko sovellus halutulla tavalla selaimissa.

Kuva 3. Yleisimmät webselaimet /6/

3.2 IDE / Tekstieditori

Yksi tärkeimmistä vaiheista ennen kuin varsinaisen web-sovelluksen kehitys alkaa, on valita sopiva IDE (Integrated Development Environment) tai tekstieditori. Visual Studio Code on suosittu vaihtoehto sen hyvän yhteensopivuuden eri ohjelmointikielten takia ja se myös tarjoaa laajat työkalut refaktorointiin sekä virheenjäljitykseen. Myös sen laaja valikoima erilaisia laajennuksia ja plugineja auttavat ohjelmistokehittäjiä heidän projekteissaan. Lisäksi siihen on kätevästi sisään rakennettu terminaali, joka on yksi syistä, miksi tässä työssä on päädytty käyttämään sitä. **(Kuva 4.)**

Kuva 4. Visual Studio Code

3.3 Virtuaaliympäristön asennus

Voidakseen käyttää Flaskia web-sovellusten tekemiseen, ensin on luotava virtuaaliympäristö, johon Flask voidaan asentaa. Olettaen, että koneessa on asennettuna Python 3, virtuaaliympäristö ja Flask voidaan asentaa tässä tapauksessa Windowsille seuraavilla komennoilla. **(Kuva 3.)**

```
Windows

Create the virtual environment:

    py -3 -m venv .venv

Activate the virtual environment:

    .venv\Scripts\activate

Install Flask via pip:

    pip install Flask
```

Kuva 5. Virtuaaliympäristön asennus

Virtuaaliympäristöä käytetään Pythonissa pakettien ja riippuvaisuuksien hallintaan projektikohtaisesti. Sen avulla voidaan käyttää tiettyä haluttua versiota itse Pythonista projektin sisällä sekä tiettyjä versioita ulkoisista kirjastoista. Tällä pystytään välttämään globaaleja asennuksia, jotka voivat aiheuttaa erilaisia ongelmia, kuten esimerkiksi rikkoa systeemityökaluja tai muita projekteja.

3.4 Projektihakemisto

Alustava rakenne projektihakemistolle on seuraavanlainen. **(Kuva 6.)**

```
elokuvakatalogi             # This is the root folder
  | - .flaskenv             # Flask-specific configuration
  | - .git                  # Git folder
  | - .venv                 # Virtual environment
  | - README.md             # Project description
  | - requirements.txt      # Requirements file
  | - .gitignore.txt        # Git ignore file
  | - app/                  # Flask app code
      | - __init__.py       # App definition
      | - routes.py         # Blueprint for routes
      | - templates/        # HTML Templates etc.
      | - static/           # CSS, Images etc.
```

Kuva 6. Projektihakemiston rakenne

App -kansio sisältää itse sovelluksen kaikki koodit. Sovelluksen määritys tapahtuu ”__init__.py” -tiedoston sisällä. Reititykset eli, mitä tapahtuu, kun liikutaan sovelluksen sisällä, on tehty ”routes.py” -nimiseen tiedostoon. Reitit voisi laittaa suoraan init-tiedostoon, mutta helppolukuisuuden vuoksi niitä varten on erillinen tiedosto. Kaikki HTML löytyy templates-nimisen kansion alta ja CSS static-nimisestä kansiosta. Virtuaaliympäristö löytyy ”.venv” -kansiosta ja ”requirements.txt” -tiedoston avulla pidetään yllä projektin riippuvaisuuksia.

3.5 Versionhallinta

On tärkeää, että projektin tiedostoja ja koodia voi pitää turvallisesti tallessa, mm. sen takia, että voi pitää yllä versionhallintaa eri koodin vaiheista sekä muutoksista ja myös, jos menettää tiedostot paikallisesta koneesta niin ne pysyvät kuitenkin tallessa muualla. Tätä varten tämän työn versionhallinnasta vastaamaan on valittu Git ja GitHub. Nämä tulevat myös myöhemmin hyödyksi, jos halutaan julkaista web-sovellus jollakin julkisella palvelimella, että muutkin käyttäjät pääsevät kokeilemaan sovellusta.

4 WEB-SOVELLUKSEN TOTEUTUS

Tässä luvussa esitellään web-sovellus. Tutustutaan sen vaatimuksiin, mitä odotetaan lopputulokselta, käydään läpi sen toiminta ja selitetään yksityiskohtaisemmin eri piirteistä. Tämän web-sovelluksen tarkoitus on olla sovellus, jolla käyttäjä voi lisätä itse sovellukseen haluamiaan elokuvia ja muodostaa näistä katalogin. Käyttäjä voi ylläpitää tietoa mm. siitä, milloin on viimeksi katsonut elokuvan, kuka on ohjannut sen, mikä vuosi se on julkaistu ja lisätä kuvauksen elokuvasta. Tämä onnistuu täyttämällä lomakkeen, jossa on valmiiksi annetut kentät näille eri tiedoille, jotka käyttäjä voi sitten itse lisätä. Painamalla painiketta, sovellus lisää elokuvan tietokantaan ja esittää elokuvan sovelluksen etusivulla olevalla taulukossa, josta sitten voi painaa eri lisäämien elokuvien kohdalta lisätietoja varten, joita myös käyttäjä voi itse myöhemmin halutessaan tarkentaa ja muokata.

4.1 Yleiskatsaus sovelluksesta

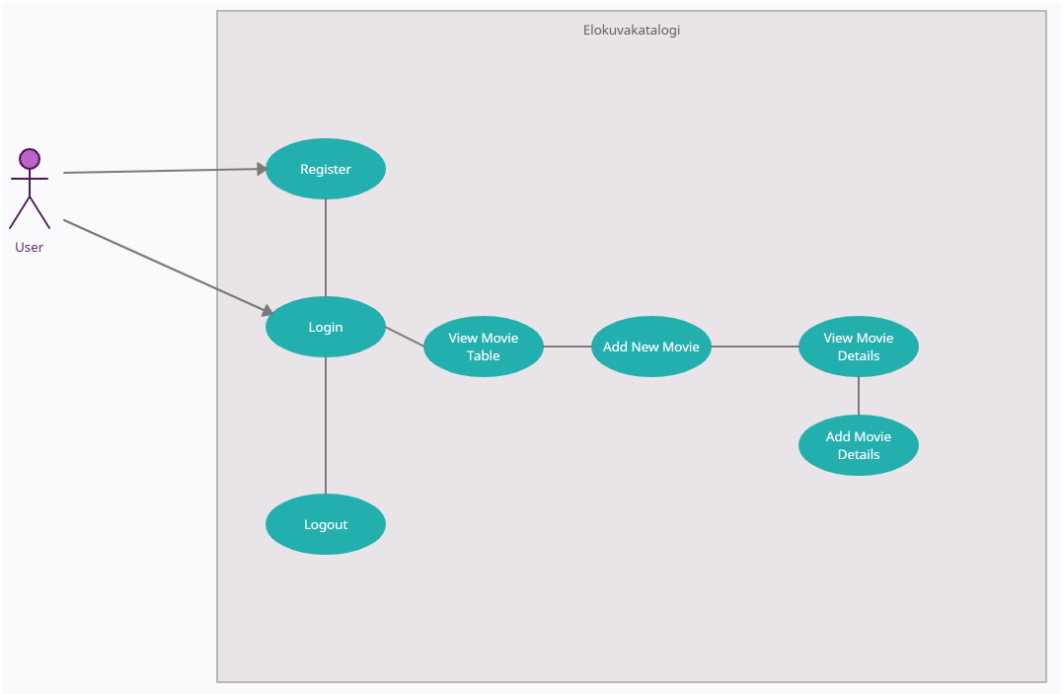
Sovelluksen päätarkoitus on ylläpitää henkilökohtaista katalogia omista lempielokuvista. Käyttäjän luotuaan itselleen tilin ja kirjauduttua sisään, tulee esille elokuvasivu, josta käyttäjä voi itse lisätä haluamiaan elokuvia. Elokuvat näytetään taulukossa, josta voi painaa yksittäisen elokuvan kohdalta lisätiesivulle. Tältä sivulta käyttäjä voi halutessaan lisätä elokuvalle tarkentavia tietoja. Sovelluksen tavoite on olla helppokäyttöinen ja käyttöliittymältään yksinkertainen, mutta visuaalisesti miellyttävä. Alla on lista olennaisista ominaisuuksista, mitä sovellus sisältää:

- Rekisteröinti / Sisäänkirjautuminen
- Lisää uusi elokuva (lomake)
- Tarkastele lisäämiä elokuvia (taulukko)
- Lisää elokuvalle lisätietoja
- Muokkaa elokuvan tietoja
- Kirjautunut käyttäjä näkee vain omat elokuvat
- Dark mode eli tumma tila (osa käyttöliittymän miellyttävyyttä)

4.2 Use Case Diagrammi (Käyttötapauskaavio)

UML-mallinnuksen mukainen käyttötapauskaavio kertoo tiivistetysti toimijan vuorovaikutuksesta käyttöjärjestelmän kanssa. Toisin sanoen tässä tapauksessa, miten esimerkiksi käyttäjä (eng. user) voi käyttää sovellusta. /7/

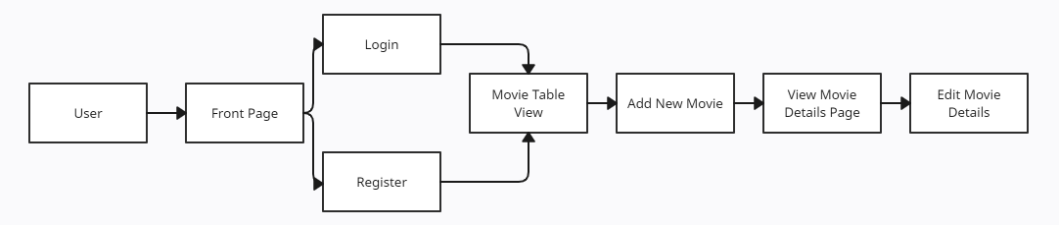
Kuvassa 7 näytetään käyttötapauskaavio työn sovelluksesta. Käyttäjä voi rekisteröityä ja sisään kirjautua sovellukseen sekä myös sisään kirjauduttuaan ulos kirjautua. Käyttäjä voi katsella elokuvataulukkoa, lisätä uuden elokuvan ja katsella elokuvan lisätietoja sekä lisätä elokuvalle lisätietoja.



Kuva 7. Käyttötapauskaavio

4.3 Sovelluksen käyttövaiheet

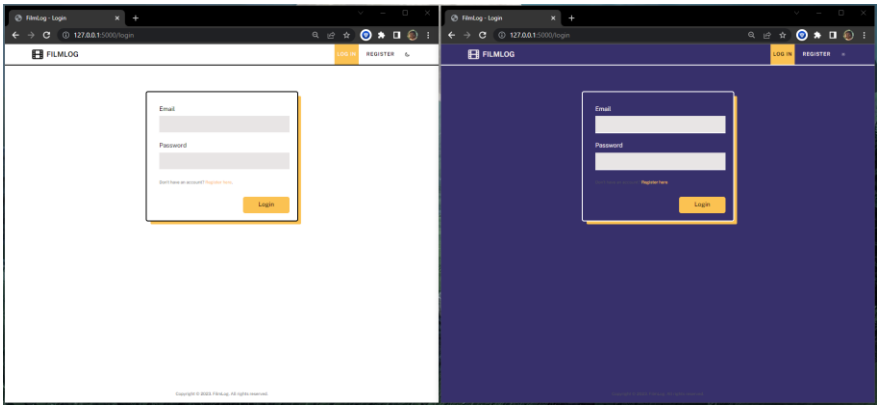
Sovelluksen käynnistyessä käyttäjälle avautuu sivu, jolla on ensin näkymä kirjautua sisään tai vaihtoehtoisesti rekisteröityä uutena käyttäjänä. Tunnistauduttuaan sovelluksen sisään, käyttäjälle tulee näkyviin elokuvasivu, joka on aluksi tyhjä, ellei ole lisännyt aikaisempia elokuvia järjestelmään. Tältä sivulta käyttäjä pystyy klikkaamaan painiketta, josta voidaan lisätä uusi elokuva. Painike ohjaa käyttäjän täyttämään lomakkeen, jolla voidaan lisätä uusi elokuva sovellukseen. Lisättyään uuden elokuvan, sovellus palaa takaisin elokuvasivulle, jolla on nyt näkyvissä uusi lisätty elokuva. Myöhemmin, kun lisää yhä enemmän elokuvia, ne näkyvät siististi taulukossa. Elokuvan kohdalta voidaan klikata painiketta, joka ohjaa käyttäjän elokuvan lisätietosivulle, jolla käyttäjä voi vapaaehtoisesti lisätä kyseiselle elokuvalle lisätietoja painamalla editointipainiketta. Tämä painike ohjaa jälleen lomakkeeseen, jolle käyttäjä voi lisätä uusia lisätietoja sekä myös muokata elokuvan tietoja. Kuvassa 8 on yksinkertaistettu kaavio sovelluksen kulusta.



Kuva 8. Vuokaavio

4.3.1 Sovelluksen ”walkthrough”

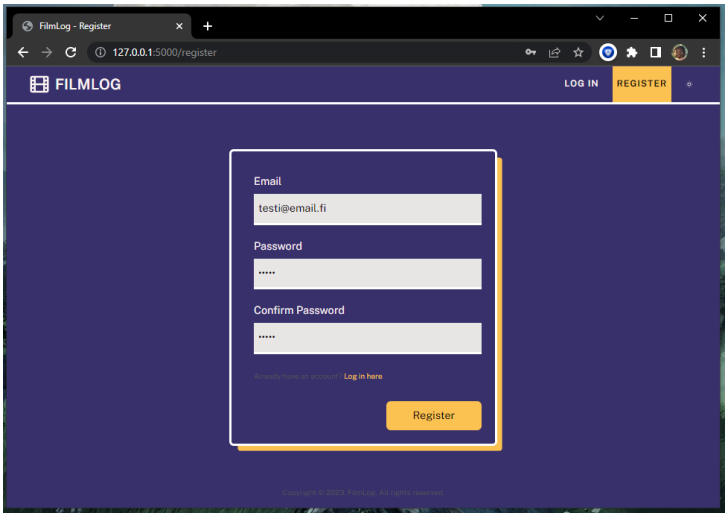
Tässä kappaleessa käydään läpi kuvineen sovelluksen toiminta käyttäjän näkökulmasta eli toisin sanoen, mitä käyttäjä voi tehdä sovelluksella sekä, miltä käyttöliittymä näyttää käyttäjälle. Ensimmäisenä kun sovellukseen menee, käyttäjälle näkyy kuvan 9 mukainen näkymä:



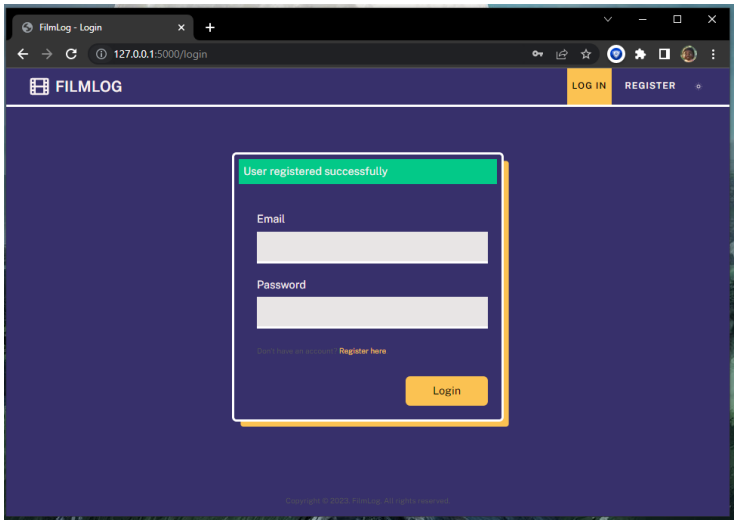
Kuva 9. Etusivu

Etusivulla on näkymä kirjautua sisään tai luoda uusi käyttäjä ennen kuin voi päästä käyttämään varsinaista sovellusta. Sovellusta pystyy käyttämään joko vaaleassa tai tummassa tilassa. **(Kuva 9.)**

Valitessaan uuden käyttäjän luomisen, sovellus ohjaa käyttäjän rekisteröitymään **(Kuva 10.)**, jonka jälkeen se ilmoittaa, että uusi käyttäjä on luotu onnistuneesti ja lisäksi ohjaa käyttäjän kirjautumaan sisään, että voi päästä käyttämään sovellusta. **(Kuva 11.)**

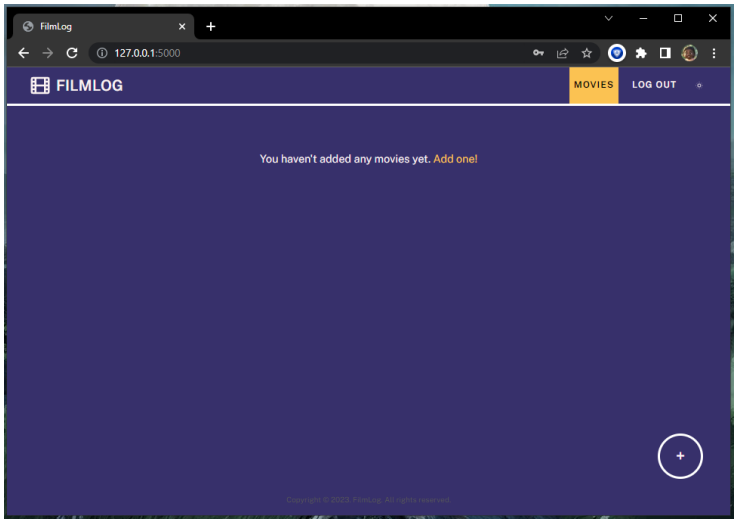


Kuva 10. Luo uusi käyttäjä

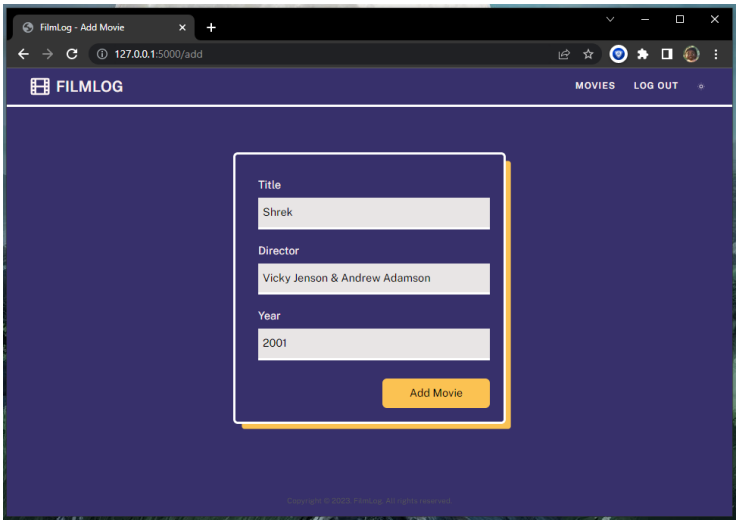


Kuva 11. Käyttäjä luotu onnistuneesti

Kuvassa 12 näkyy elokuvasivu, joka on aluksi tyhjä. Painamalla lisäyspainiketta päästään täyttämään ”uusi elokuva” -lomake. **(Kuva 13.)**

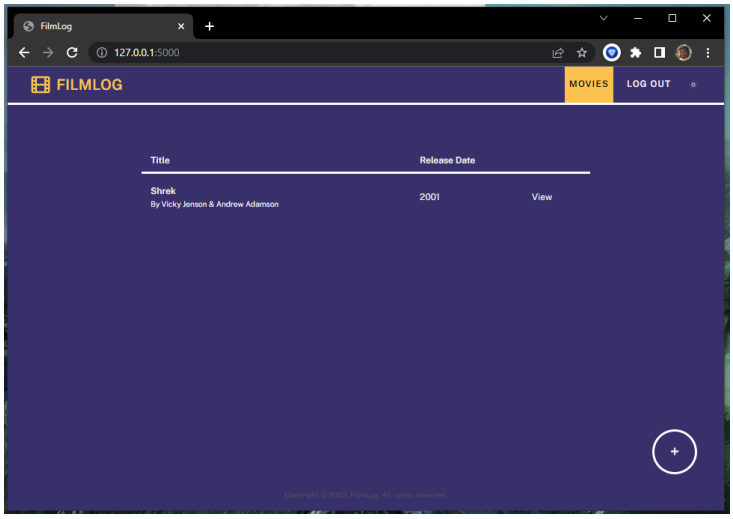


Kuva 12. Elokuvanäkymä tyhjä

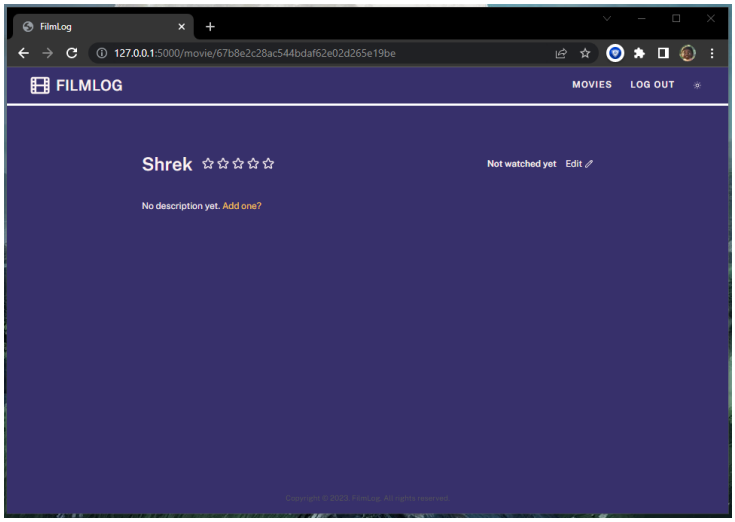


Kuva 13. Lisää uusi elokuvalomake

Lisättyään uuden elokuvan, elokuvasivun näkymä muuttuu. **(Kuva 14.)** Nyt lisätyt elokuvat näkyvät taulukkonäkymässä, josta voidaan klikata lisätietosivulle. **(Kuva 15.)** Lisätietosivu näkyy tyhjänä lukuun ottamatta, että tähtiarvosanan ja viimeksi katsonut päivämäärän voi suoraan muokata painamalla niiden kohdalta.

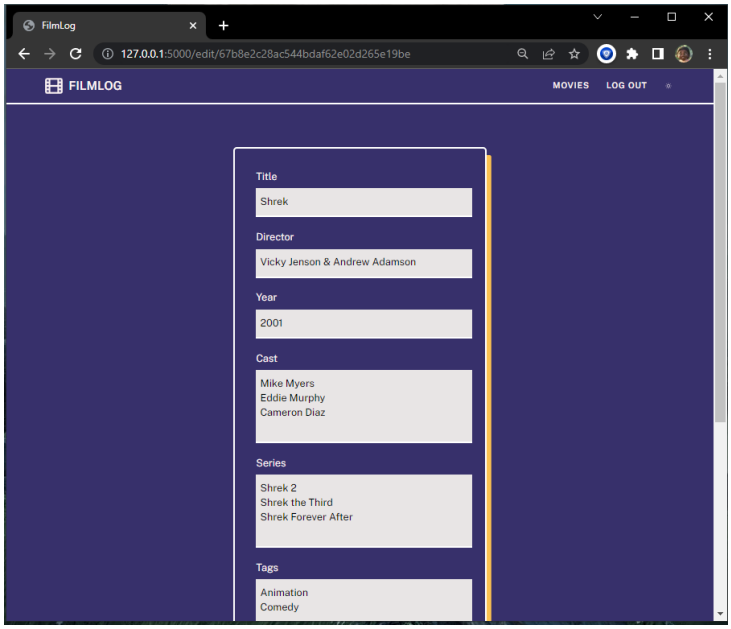


Kuva 14. Elokuvanäkymä: uusi elokuva lisätty

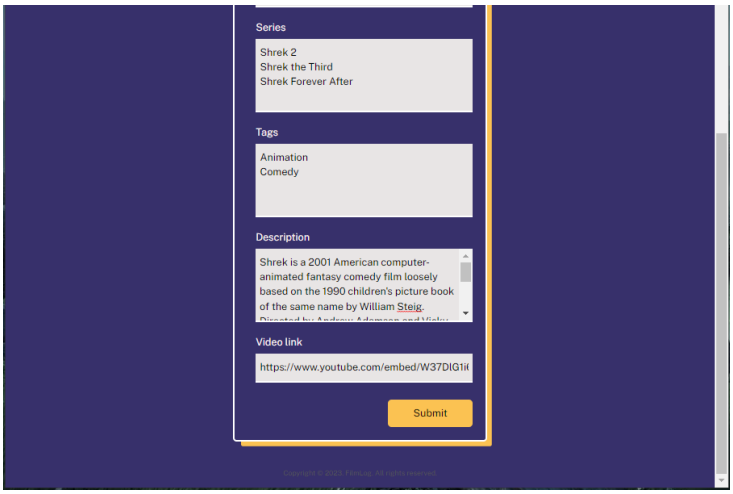


Kuva 15. Elokuvalisätietosivu tyhjä

Painamalla editointipainiketta **(Kuva 15.)**, sovellus ohjaa käyttäjän täyttämään lisätietolomakkeen. **(Kuvat 16.–17.)**

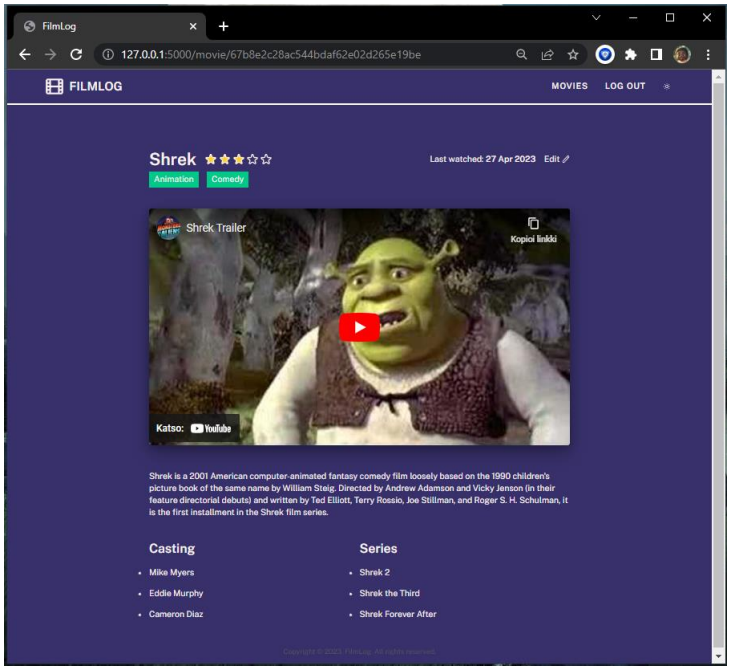


Kuva 16. Elokuva lisätieto lomake osa 1



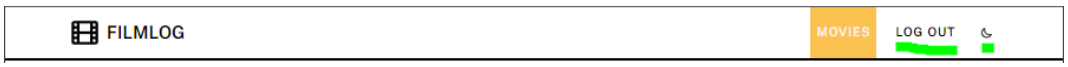
Kuva 17. Elokuva lisätieto lomake osa 2

Kuvassa 18 näkyy, miltä täytetty elokuvalisätietosivu näyttää.



Kuva 18. Elokuva lisätietosivu

Lopuksi käyttäjä voi halutessaan mennä takaisin elokuvasivulle, josta näkyy taulukko lisäämistä elokuvistaan **(Kuva 14.)** tai vaikka kirjautua sovelluksesta ulos, jolloin palataan sisäänkirjautumisnäkymään. **(Kuva 9.)** Kuvissa 19 ja 20 on näytetty tarkempi näkymä sovelluksen ylävalikosta.



Kuva 19. Ylävalikko vaalea tila

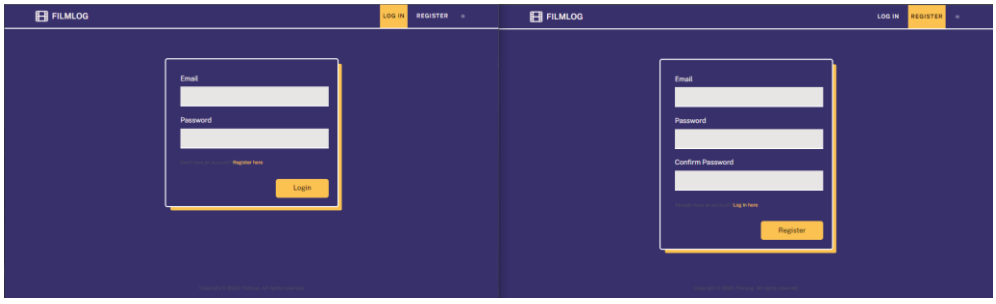
Kuva 20. Ylävalikko tumma tila

4.4 Käyttöliittymän rakenne ja toiminta

Ideana on, että päästääkseen käyttämään sovellusta, käyttäjällä on oltava tili tai hänen täytyy luoda uusi tili sovellukseen. Tätä varten sovellukseen on tehty rajoituksia, mitä on aluksi näkyvillä ei-sisäänkirjautuneelle käyttäjälle. Käyttäjän on aina oltava sisään kirjautuneena nähdäkseen sovelluksen kaikki sivut.

4.4.1 Login- ja register-sivut

Ensimmäisenä näkyvissä ovat vain kaksi sivua, sisään kirjautuminen sekä rekisteröinti, ennen kuin voi päästä pidemmälle. **(Kuva 21.)**



Kuva 21. Sovelluksen näkymä ennen sisään kirjautumista

Ohjatakseen käyttäjän eteenpäin, sovellus katsoo, löytyykö sen sessiosta sähköpostiosoitetta eli toisin sanoen, onko käyttäjä sisäänkirjautunut. Jos ei ole, niin sovellus pyytää sisään kirjautumaan tai rekisteröitymään. Jos sähköposti löytyy, niin sovellus ohjaa käyttäjän indeksisivulle eli toisin sanoen sovelluksen elokuvanäkymään. Kuvassa 22 näytetään rekisteröintisivun reitti.

```python
@pages.route("/register", methods=["POST", "GET"])
def register():
    if session.get("email"):
        return redirect(url_for(".index"))

    form = RegisterForm()

    if form.validate_on_submit():
        user = User(
            _id=uuid.uuid4().hex,
            email=form.email.data,
            password=pbkdf2_sha256.hash(form.password.data),
        )

        current_app.db.user.insert_one(asdict(user))

        flash("User registered successfully", "success")

        return redirect(url_for(".login"))

    return render_template(
        "register.html",
        title="FilmLog - Register",
        form=form
    )
```

Kuva 22. Rekisteröintisivun reititys

Ensin tarkastetaan, onko jo käyttäjä sisään kirjautuneena. Kun käyttäjä painaa lomakkeen submit-nappulaa, sovellus tarkastaa, että kaikki kentät ovat täytetty sekä luo käyttäjälle uniikin id:n sekä salasana hashataan. Käyttäjän tiedot tallennetaan tietokantaan. Lomake näyttää käyttäjälle viestin, jos käyttäjätili on luotu onnistuneesti. Lopuksi sovellus ohjaa käyttäjän kirjautumissivulle. **(Kuva 22.)**

```python
@pages.route("/login", methods=["GET", "POST"])
def login():
    if session.get("email"):
        return redirect(url_for(".index"))

    form = LoginForm()

    if form.validate_on_submit():
        user_data = current_app.db.user.find_one({"email":
form.email.data})
        if not user_data:
            flash("Login credentials not correct",
category="danger")
            return redirect(url_for(".login"))
        user = User(**user_data)

        if user and pbkdf2_sha256.verify(form.password.data,
user.password):
            session["user_id"] = user._id
            session["email"] = user.email

            return redirect(url_for(".index"))

        flash("Login credentials not correct",
category="danger")

    return render_template(
        "login.html",
        title="FilmLog - Login",
        form=form
    )
```

Kuva 23. Login-sivun reititys

Sisään kirjautumista varten jälleen katsotaan, onko jo käyttäjä kirjautuneena. Kirjoittaessaan kirjautumistietoja, lomake etsii tietokannasta, että vastaako sähköposti jo olemassa olevaa käyttäjää. Jos se vastaa olemassa olevaa käyttäjää, niin tarkastetaan vastaako syötetty salasana käyttäjän tietokannassa olevaan salasanaan. Lomake myös ilmoittaa, jos syötetyt tiedot eivät vastaa olemassa olevia tunnuksia. Kun syötetyt tiedot vastaavat oikeaan käyttäjätunnukseen, käyttäjän id sekä sähköposti tallentuu sessioon ja käyttäjä ohjataan indeksisivulle. **(Kuva 23.)** Ulos kirjautuessa sessio nollataan ja käyttäjä ohjataan takaisin kirjautumissivulle. **(Kuva 24.)**

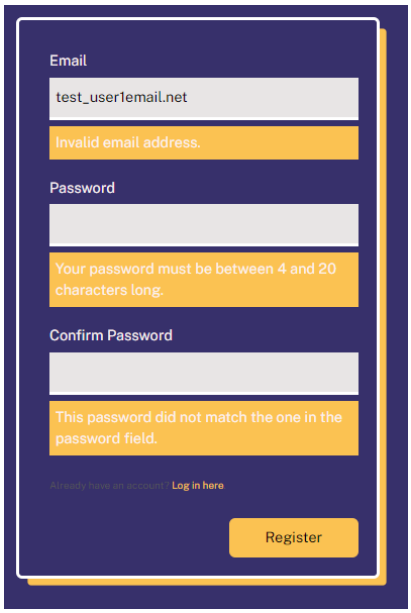
```python
@pages.route("/logout")
def logout():
    current_theme = session.get("theme")
    session.clear()
    session["theme"] = current_theme

    return redirect(url_for(".login"))
```

Kuva 24. Logout-reititys

4.4.2 Flask-WTF & WTForms

Kaikkiin sovelluksen lomakkeiden luomiseen on käytetty Flask-WTF -nimistä laajennusta, joka mahdollistaa WTForms -nimisen kirjaston käytön. WTForms on kirjasto, joka mahdollistaa joustavien lomakkeiden renderöinnin web-sovelluksiin. Se tarjoaa kaikki lomakkeiden peruselementit, kuten tekstikentän ja salasanakentän. Sillä voidaan käyttää niin sanottuja ”validaattoreita”, jotka helpottavat tiedon oikeellisuuden tarkistamisessa. Esimerkiksi, jos tietty kenttä on pakollinen täyttää. Kuvassa 25 on esimerkkejä siitä, minkälaisia virheilmoituksia lomake voi näyttää käyttäjälle, jos tiedot eivät ole täytetty oikein kenttiin.



Kuva 25. Rekisteröintilomakkeen virheilmoitukset

Kuvassa 26 esitetään miten ”validaattoreita” on käytetty rekisteröintilomakkeen tekoon. Lomake tarkistaa, onko sähköposti syötetty oikein, onko salasana vaaditun pituinen sekä vastaako uudelleen syötetty salasana ensimmäisen salasanan kanssa. WTForms-kirjasto pitää sisällään lukuisia eri kenttiä, joita lomakkeen rakentamiseen voidaan käyttää. Rekisteröintilomakkeessa on käytetty StringField -kenttää sähköpostille, PasswordField-kenttää salasanaa varten sekä SubmitField-kenttää submit-nappulaa varten, joka vie tiedon eteenpäin. **(Kuva 26.)**

```
class RegisterForm(FlaskForm):
    email = StringField("Email", validators=[InputRequired(),
Email()])

    password = PasswordField(
        "Password",
        validators=[
            InputRequired(),
            Length(
                min=4,
                max=20,
                message="Your password must be between 4 and 20
characters long.",
            ),
        ],
    )

    confirm_password = PasswordField(
        "Confirm Password",
        validators=[
            InputRequired(),
            EqualTo(
                "password",
                message="This password did not match the one in
the password field.",
            ),
        ],
    )

    submit = SubmitField("Register")
```

Kuva 26. Rekisteröintilomakkeen rakenne

4.4.3 Käyttöliittymän mukavuus

Käyttöliittymän mukavuuden visuaalista puolta ajatellen sovellusta on mahdollista käyttää joko vaaleassa tai tummassa teemassa. Tätä varten on hyödynnetty CSS-muuttujia **(Kuva 27.)**, jotka helpottavat määrittämään, miltä sovelluksen elementit näyttävät selaimessa, kun tietyt ehdot täyttyvät.

```
:root {
    --text-dark: #000;
    --text-light: #fbf2f2;
    --text-muted: #595959;

    --background-color: #fff;
    --accent-colour: #FBC252;
    --accent-colour-2: #03C988;
    --tag-colour: #e5e5e5;

    --border: 3px solid #000;
}
/* Dark Mode: */
:root.dark-mode {
    --text-dark: #fbf2f2;
    --text-light: #000;
    --text-muted: #595959;

    --background-color: #37306B;
    --accent-colour: #FBC252;
    --accent-colour-2: #03C988;
    --tag-colour: var(--accent-colour-2);

    --border: 3px solid #fff;
}
```

Kuva 27. CSS-muuttujat

Sovellus tarkistaa sessiosta, onko vaalea vai tumma teema päällä ja vaihtaa sen haluttuun tilaan, kun käyttäjä painaa teemamerkin kohdalta. Sovellus myös varmistaa, että teemaa vaihdettaessa pysytään samalla sivulla. **(Kuva 28.)**

```python
@pages.get("/toggle-theme")
def toggle_theme():
    current_theme = session.get("theme")
    if current_theme == "dark":
        session["theme"] = "light"
    else:
        session["theme"] = "dark"
    # make sure when theme toggle icon is clicked,
    # we don't navigate to a different page:
    return redirect(request.args.get("current_page"))
```

Kuva 28. Tumma / Vaalea teema

4.4.4 Jinja2

Voidakseen esittää web-sovelluksen data selaimessa, tässä sovelluksessa on käytetty Jinja2 yhdessä HTMLn kanssa. Jinja2 on niin sanottu web-mallinnusmoottori (eng. Web templating engine). Muuten sovelluksessa HTML toimii normaalin tuttuun tapaan, mutta käyttäen Jinja2, HTML-koodin sekaan on sisälletty muuttujia (eng. placeholder), joiden tilalle tulee näkyville haluttu data, kun selain renderöi sovelluksen. Selaimet eivät kykene lukemaan itse Python-koodia, joten web-kehityksen puolella Pythonissa käytetään erilaisia mallinnusmottoreita. Flaskin kanssa tyypillinen mallinnusmoottori on Jinja2. Kuvassa 29 on esimerkki, miltä Jinja2 näyttää sovelluksen HTML-koodissa.

```
<tbody>
        {% for movie in movies_data %}
            <tr>
                <td class="table__cell">
                    <p class="table__movieTitle">{{ movie.title }}</p>
                    <p class="table__movieDirector">By {{ movie.director }}</p>
                </td>
                <td class="table__cell">{{ movie.year }}</td>
                <td class="table__cell"><a href="{{ url_for('pages.movie', _id=movie._id) }}" class="table__link">View</a></td>
            </tr>
        {% endfor %}
</tbody>
```

Kuva 29. Jinja2 esimerkki

Jinja2 on sisälletty aivan normaalin HTML-tiedoston sekaan. Tyypillisesti Python-web-projekteissa HTML-tiedostot löytyvät templates-nimisen kansion sisältä, jotta selain pystyy löytämään ne ja esittämään tiedon oikein. Mitä tulee Jinja2 syntaksiin, niin esimerkiksi sen avulla voidaan käyttää normaaleja lauseita (mm. for ja if -lauseet), kun ne laitetaan ” {% %} ” -merkkien sisälle ja ” {{ }} ” -merkkien sisälle kirjoitetaan mitä tahansa määritelmiä / dataa, joka halutaan näkyville, kun selain renderöi sivun.

4.5 Back-end

Elokuvien ja käyttäjien tallentamista tietokantaan varten niistä muodostetaan niin sanottuja malleja käyttäen dataluokkia (eng. dataclass), sillä jokaisella yksittäisellä elokuvalla ja käyttäjällä on useampi ominaisuus. Jokaisella elokuvalla on uniikki id, nimi, ohjaaja, julkaisuvuosi jne. sekä jokaisella käyttäjällä on oma uniikki id, sähköposti, salasana ja lista lisäämistään elokuvista. Dataluokka on keino muodostaa yksittäinen objekti, joka pitää sisällään ryhmän erilaista dataa ja näin myös helpottaa, mitä sillä kaikella datalla voi tehdä.

4.5.1 Dataluokat

Muodostetaan luokka, joka edustaa yksittäistä elokuvaa ja pitää sisällään eri tiedot, joita elokuvalle voidaan antaa. **(Kuva 30.)**

```python
from dataclasses import dataclass, field
from datetime import datetime


@dataclass
class Movie:
    _id: str
    title: str
    director: str
    year: int
    cast: list[str] = field(default_factory=list)
    series: list[str] = field(default_factory=list)
    last_watched: datetime = None
    rating: int = 0
    tags: list[str] = field(default_factory=list)
    description: str = None
    video_link: str = None
```

Kuva 30. Elokuva dataluokka

Luokalle määritetään parametrejä sekä niiden tietotyypit. Parametreille on annettu oletusarvot, jotta niille ei tarvitse syöttää arvoja, kun luodaan objekti. Tämä on sen takia, että kun luodaan uusi elokuva, sille ei tarvitse syöttää saman tien kaikkia sen tietoja.

Joillekin parametreille on annettu field-objekti ja ”default-factory”. Tämä on tehty, että voidaan paremmin hallita oletusarvon toimintaa. Se antaa määrittää funktion oletusarvoksi ja kun objekti luodaan funktion palautusarvoa, käytetään parametrin arvona. Tämä auttaa käsittelemään muuttuvia arvoja, kuten tässä tapauksessa listoja. Objektia luodessa, funktio luo aina uuden listan ja määrittää sen parametrin arvoksi. Näin jokaisella elokuvaobjektilla on omat listansa ja yhden elokuvan listaa muokatessa, muut listat pysyvät entisellään. **(Kuva 31.)**

```
cast: list[str] = field(default_factory=list)
```

Kuva 31. Field-objekti

Kuvassa 32 näytetään, miten dataluokkaa voidaan käyttää. Muodostamalla konstruktorin elokuvasta ja kutsumalla asdict() -nimistä funktiota se muunnetaan sanakirjaksi ja tallennetaan tietokantaan.

```
@pages.route("/add", methods=["GET", "POST"])
@login_required
def add_movie():
    form = MovieForm()

    if form.validate_on_submit():
        movie = Movie(
            _id=uuid.uuid4().hex,
            title=form.title.data,
            director=form.director.data,
            year=form.year.data,
        )

        current_app.db.movie.insert_one(asdict(movie))
        current_app.db.user.update_one(
            {"_id": session["user_id"]}, {"$push": {"movies": movie._id}}
        )


        return redirect(url_for(".index"))

    return render_template(
        "new_movie.html",
        title="FilmLog - Add Movie",
        form=form # passing MovieForm() object to the form template
    )
```

Kuva 32. Elokuvan lisäys

Lisäksi sovellus päivittää tietokantaan kirjautuneelle käyttäjälle elokuvan tunnistimen (id), jotta tiedetään, kuka on lisännyt elokuvan ja mitä elokuvia tietylle käyttäjälle kuuluu. Myös tästä syystä käyttäjän dataluokassa on tehty parametri, jonka arvoksi on annettu lista elokuvien id:tä varten. **(Kuvat 32.–33.)**

```python
@dataclass
class User:
    _id: str
    email: str
    password: str
    movies: list[str] = field(default_factory=list)
```

Kuva 33. Käyttäjä dataluokka

4.5.2 MongoDB-yhteys

MongoDB-yhteyttä varten käytetään PyMongo -nimistä kirjastoa, jonka avulla saadaan käyttöön MongoClient. Se ottaa yhteyden vaadittuun URI-osoitteeseen, johon päästään käsiksi, kun luodaan niin sanottu ryhmä (eng. cluster) MongoDB Atlaksen palvelimella. **(Kuva 34.)**

```python
import os
from flask import Flask
from dotenv import load_dotenv
from pymongo import MongoClient

from app.routes import pages

load_dotenv()


def create_app():
    app = Flask(__name__)
    app.config["MONGODB_URI"] = os.environ.get("MONGODB_URI")
    app.config["SECRET_KEY"] = os.environ.get(
        "SECRET_KEY", "..."
    )
    app.db =
MongoClient(app.config["MONGODB_URI"]).get_default_database()
    app.register_blueprint(pages)
    return app
```

Kuva 34. MongoDB-yhteys

Itse URI-osoite laitetaan env-nimisen tiedoston sisälle. **(Kuva 35.)**

```
MONGODB_URI=mongodb://mongodb_user:mongodb_password/127.0.0.1:27017/default_databas
e
```

Kuva 35. Env-tiedoston esimerkki

Turvallisuutta ajatellen, MongoDB Atlaksen puolella, kun on luotu uusi ryhmä (eng. cluster), voidaan tehdä ryhmälle uusi käyttäjä ja antaa sille salasana. Ainoastaan autentikoineella käyttäjällä on pääsy tietokantaan. Käyttäjätiedot täytetään aikaisemmin mainittuun URI-osoitteeseen, jonka avulla saadaan yhteys tietokantaan. Myös IP-osoitteita, joista pääsee käsiksi tietokantaan, voidaan rajoittaa. Esimerkiksi jos haluaa vain, että omalta koneelta voidaan saada yhteys ryhmään.

5 PÄÄTELMÄT

Menetelmiä, joilla voi toteuttaa web-sovelluksen, on lukuisia ja ne kehittyvät päivä päivältään. Tyypillistä web-kehityksen puolella on käyttää JavaScriptiä tai Reactia, mutta tätä työtä varten valinta osui Pythonin kohdalle. Sen mahdollisuudet ovat erittäin monipuolisia ja hyödyntäen Flaskia, Pythonin avulla voi toteuttaa vaikka mitä erilaisia web-sovelluksia. Yhdessä HTML:n ja CSS:n kanssa sujui onnistuneesti tavoitellun sovelluksen toteutus.

Tavoitteena oli toteuttaa web-sovellus, jolla autentikoinut käyttäjä pystyy lisäämään haluamiaan elokuvia järjestelmään ja ylläpitää omaa henkilökohtaista katalogia niistä. Käyttäjä pystyy lisäämään yksittäiselle elokuvalle erilaisia tietoja, kuten arvosanan, katsomispäivämäärän ja kuvauksen elokuvasta.

Oli tärkeää, että käyttöliittymä on yksinkertainen ja mielekäs käyttää. Tässä onnistuttiin työssä toteuttamalla minimalistinen ulkoasu ja rakenne. Lisäksi käyttömukavuutta lisäämään, sovelluksesta löytyy erikseen tumma tila. Myös yksinkertaisuutta ajatellen, käyttäjä pystyy itse lisäämään lomakkeen avulla haluamansa elokuvan ja lisätietoja sille. Sovellus tallentaa onnistuneesti sekä elokuvat tietokantaan että myös käyttäjätunnukset. Tietoturvallisuutta ajatellen salasanat ovat suojattuja.

Työssä onnistuttiin toteuttamaan tavoiteltu sovellus ja sille asetetut vaatimukset. Kuitenkin tulevaisuutta ajatellen on erilaisia mahdollisuuksia, mitä uusia ominaisuuksia sovellukseen voisi lisätä, kuten esimerkiksi käyttäjät voisivat nähdä myös muiden lisäämiä elokuvia ja jättää kommentteja niiden alle.

LÄHTEET

/1/HTML: HyperText Markup Language. Viitattu 16.2.2023. https://developer.mozilla.org/en-US/docs/Web/HTML

/2/CSS: Cascading Style Sheets. Viitattu 16.2.2023. https://developer.mozilla.org/en-US/docs/Web/CSS

/3/Micro Vs Full-Stack Frameworks. Viitattu 9.3.2023. https://www.technolush.com/blog/micro-vs-full-stack-frameworks

/4/Why Use MongoDB and When to Use It. Viitattu 20.2.2023. https://www.mongodb.com/why-use-mongodb

/5/MongoDB vs. MySQL Differences. Viitattu 10.3.2023. https://www.mongodb.com/compare/mongodb-mysql

/6/Desktop Browser Market Share Worldwide – February 2023. Viitattu 11.3.2023. https://gs.statcounter.com/browser-market-share/desktop/worldwide

/7/UML Use Case Diagram Tutorial. Viitattu 27.3.2023. https://www.lucidchart.com/pages/uml-use-case-diagram