Dokumentatsioon

Sissejuhatus

Filmikoda on veebirakendus, mis võimaldab kasutajatel avastada laia valikut filme, leida sobivaid seansse ning broneerida pileteid mugavalt. Rakendus pakub mitmeid funktsioone, sealhulgas erinevate filtrite kasutamine, kinosaalis parimate vabade kohtade soovitamine ning võimalus piletite broneerimiseks.

Kasutajaliides

Filmikoda pakub lihtsat ja loogilist kasutajaliidest, mis on kujundatud nii, et kasutajad saaksid mugavalt navigeerida, uudistada erinevaid filme ning osta pileteid vaid mõne hiireklõpsuga. Rakendus on loodud kasutajasõbralikult, võimaldades hõlpsat juurdepääsu kogu funktsionaalsusele.

Kasutatud tehnoloogiad

- Backend: Spring Boot
- Frontend: React.js
- Andmebaas: PostgreSQL
- Andmebaasi haldus: PgAdmin
- Muud tehnoloogiad: Node.js, npm, React Router

Paigaldus juhised

Andmebaasi setup

- Laadige alla PostgreSQL andmebaas ja PgAdmin tööriist.
- Installige PostgreSQL vastavalt juhistele ja seadistage superkasutaja parool ning port.
- Kasutage PgAdmin tööriista, et luua uus andmebaas nimega "kino".

Backendi ühendus andmebaasiga

- Avage application.properties fail ja lisage andmebaasi ühendamiseks vajalikud konfiguratsioonid, sealhulgas kasutajanimi ja parool. (spring.datasource.username=postgres ja spring.datasource.password=password)
- Veenduge, et olete määranud õige andmebaasi URL-i ja seaded, et Spring Boot rakendus saaks edukalt andmebaasiga ühenduda.

Backendi käivitamine

- Veendu, et on alla laaditud JDK (Java Development Kit) ning Maven
- Laadige alla või kloonige backendi repositoorium oma arvutisse. (package demo)
- Avage IntelliJ ning jooksutage DemoApplication (src/main/java/com/example/demo/DemoApplication.java)

Frontendi käivitamine

- Installige Node.js ja npm. ()
- Laadige alla frontendi repositoorium(nodejs.org)
- Liikuge frontend kausta (cd frontend) ja käivitage käsk npm install, et installida kõik vajalikud sõltuvused.
- Käivitage rakendus käsu npm start abil ja avage seejärel veebibrauseris aadressil http://localhost:3000. (Peaks ka automaatselt selle ise avama)

Kasutajaliidese kasutamine

Kui käivitad rakenduse, ootab sind ees sisselogimisvõimalus. Sul on võimalik luua uus konto või sisse logida olemasolevaga.

Registreerimine:

- Vajuta "Registreeri" nuppu.
- Sisesta soovitud e-mail ja parool ning vajuta "Registreeri".

- Pärast registreerimist suunatakse sind tagasi sisselogimise lehele, kus saad oma uue kontoga sisse logida.

Sisselogimine:

- Sisesta e-mail ja parool ning vajuta "Logi sisse".
- Soovitame kasutada järgmist sisselogimisinfot:
    - E-mail: ma.olen.eva@gmail.com
    - Parool: kass123

Filmide uurimine:

- Vali "Kodu" navigatsiooniribalt.
- Siin lehel näed kõiki Filmikojas saadaolevaid filme.
- Kasutades ülal asuvaid filtreid, saad kitsendada näidatavate filmide valikut keele, vanusepiirangu või žanri järgi.
- Allapoole kerides näed teksti "Sulle soovitatud". Vajutades sellele, saad soovitusi filmide kohta, lähtudes sinu vaatamisajaloost.
- Kui oled valinud mõne filtri, vajuta uuesti "Sulle soovitatud", et näha soovitusi vastavalt filtritele.
- Filmide üksikasjade nägemiseks vajuta filmi peale. Seal saad lugeda filmi sisu kohta ja näha suuremat filmi pilti.

Seansside valimine:

- Vali "Seansid" navigatsiooniribalt.
- Vali sobiv seanss, millele soovid pileteid broneerida.
- Vajuta "+" nupule, et lisada pileteid või vali istekohad käsitsi.
- Vajutades istekohale, saad selle eemaldada või vajutades nupule D, saad kõik eemaldada.
- Tšeki saamiseks vajuta "Tšekk" nupule ja kinnita ost, kui soovid piletid osta.
- Pärast ostu kinnitamist suunatakse sind tagasi seansside lehele.

Välja logimine:

- Vajuta navigatsiooniribal "Logout" nupule.
- Kinnita väljalogimine vajutades "Logout" nupule.

Selline on ülevaade rakenduse kasutamisest. Edukat rakenduse avastamist ja filmide nautimist!

Oluliseimad klassid ja nende funktsioon

Backend

Paketi "filmid", "istekohad", "kasutaja", "saalid" ja "seansid" klassid moodustavad iga tabeli jaoks loodud klasside kogumi. Kõik need klassid on struktureeritud ja üles ehitatud sarnaselt, järgides samu põhimõtteid.

Klassid: Film, Istekohad, Kasutaja, Saal, Seanss

Need klassid on olulised selleks, et määratleda iga tabeli veerud ja nende tüübid. Nende hulka kuuluvad näiteks unikaalsed identifikaatorid, kasutajate e-posti aadressid, paroolid, filmide nimed ja saalide istekohad. Iga klass sisaldab konstruktorit ning meetodeid nende väärtuste seadmiseks ja lugemiseks. Need meetodid on olulised, et saaksime andmeid iga klassi objektist hankida ja manipuleerida.

Controllerid

Kontrollerid on olulised, kuna need loovad ühenduse kasutaja ja rakenduse vahel, võimaldades määrata URL-mustrid (path).

- GetMapping meetodid võimaldavad hankida kõikide klasside objekte või kindla identifikaatori järgi konkreetseid andmeid.
- PostMapping meetodid võimaldavad andmeid lisada andmebaasi.
- DeleteMapping meetodid võimaldavad vastavate andmete kustutamist andmebaasist.
- Siin on oluline märkida, et @CrossOrigin annotatsiooni kasutamine väldib probleeme, mis võivad tekkida andmete pärimisel ja vahetamisel front-end'i ja back-end'i vahel.

Service

Service-klassid on olulised, kuna need vastutavad päringute töötlemise eest vastavalt kasutaja soovidele. Nad töötlevad päringuid, et otsida, lisada või kustutada andmeid andmebaasist vastavalt rakenduse vajadustele. Need klassid suhtlevad tavaliselt repository'ga.

Repository

Repository-klassid võimaldavad andmete leidmist andmebaasist vastavalt päringutele, mida teenindab service-klass. Need on seotud otse andmebaasiga ja pakuvad vajalikke meetodeid andmete haldamiseks.

Config

Config-klassid on olulised, kuna need võimaldavad rakendusel lähtestamisel automaatselt lisada andmeid andmebaasi. Näiteks võib neid kasutada andmete eelseadistamiseks või rakenduse konfigureerimiseks vastavalt keskkonna nõuetele.

Backendi arendusprotsess

Alustasin projekti backendi loomisega Spring Boot raamistikus. Kuna mul ei olnud eelnevat kogemust Spring Bootiga, oli see minu jaoks uus ja põnev väljakutse. Õppimiseks kasutasin Amigoscode'i YouTube'i kursust, mis tutvustas põhjalikult Spring Booti loomist ja selle ühendamist andmebaasiga(https://www.youtube.com/watch?v=9SGDpanrc8U&t=961s). Kursuse abil sain vajaliku juhendi samm-sammult, alustades Spring Initializr'i kasutamisest ja vajalike failide konfigureerimisest, et luua ühendus PostgreSQL andmebaasiga. Andmebaasi käivitamiseks kasutasin PgAdmini, kuhu tuli määrata kasutajanimi ja põhiparool.

Järgnes tabeli struktuuri loomine vastavalt õigele mallile ning esimese paketi loomine nimega "Kasutaja". Lõin Kasutaja klassi oluliste väljadega ning jätkasin KasutajaService'i ja KasutajaControlleri loomisega, et hallata kasutaja loomist ja andmete töötlemist. Samuti lõin KasutajaRepository andmebaasiga suhtlemiseks ning KasutajaConfig konfiguratsiooni faili, et lisada andmeid tabelisse commandlinerunnerilt.

Järgnevad packaged said loodud vastavalt Kasutaja klassi järgi. Iga klassi loomisel mõtlesin hoolikalt läbi, millised andmed on olulised ning milliseid tuleb tabelisse lisada, näiteks unikaalne ID, parool, nimi või vaatamiste ajalugu jne. Algselt lõin kõik kontrollerid, serviced ja config failid samasugused. Frontendi arendamise käigus lisasin kontrolleritesse juurde uusi pathe ja GetMappinge, kui tundsin vajadust konkreetsete andmete kiiremaks kättesaamiseks.

Alguses oli keeruline mõista klasside vahelist struktuuri ja omavahelist seotust ning milline klass mida määrab. Selleks katsetasin palju teste ja andmebaasi päringuid, et veenduda, et kõik toimib nagu vaja.

Kokkuvõttes oli backendi loomine üsna sujuv protsess tänu hästi struktureeritud ja põhjalikule kursusele. Iseseisev katsetamine ja videotundide jälgimine aitasid mul mõista Spring Booti põhimõtteid ning nüüd, kui olen mitu õhtut projekti kallal töötanud, on uute päringute lisamine järjest lihtsam.

Abi otsimine ja lahendatud probleemid

Käsitlesin mõningaid probleeme, mis tekkisid arendusprotsessi käigus ChatGPT-iga. Esines CORS poliitika probleeme, mille lahenduseks oli @CrossOrigin annotatsiooni kasutamine, et vältida probleeme andmete vahetamisel front-end'i ja back-end'i vahel. Lisaks tekkisid raskused filmide filtreerimise implementeerimisel, kus oli vaja kasutada @RequestParam annotatsiooni erinevate päringute jaoks, et saavutada erinevaid filtreerimise tulemusi. Tekkis

probleem ka tabelite loomisel, kui mingi tabel loodi ennem kui teine. Selle lahenduseks kasutasin @Order() annotatsiooni. Samuti kasutasin abi, kui soovisin lisada andmebaasi suure hulga erinevaid andmeid korraga ja ChatGPT mõtles ise välja vastavad seansid, filmid või istekohad.

Kokkuvõttes aitasid need lahendused mul mõista ja rakendada keerukamaid kontseptsioone Spring Booti arendusprotsessis ning aitasid vältida ootamatuid probleeme ja takistusi.

Frontend

React.js-i kasutamine ja erinevad funktsionaalsused/detailid õppisin Youtube video järgi. (https://www.youtube.com/watch?v=O_XL9oQ1_To)

App.js

- React Router (BrowserRouter, Routes, Route): Need komponendid võimaldavad rakenduses navigeerimist ja dünaamiliste lehtede kuvamist vastavalt URL-ile. See on oluline, sest see annab kasutajale võimaluse liikuda erinevate lehtede vahel ja tagab sujuva kasutuskogemuse. Selle kasutamise võimaluse näitas mulle ette ChatGPT.

- Kasutajate seisundi haldamine (useState): useState kasutamine võimaldab kasutajate oleku haldamist rakenduses. Selle abil saab jälgida, kas kasutaja on sisse logitud või mitte ning vastavalt sellele kuvada või peita erinevat sisu. See on oluline, sest see võimaldab kasutajal teha sisselogimise ja väljalogimise toiminguid ning kuvada vastavalt sellele erinevat sisu.

- Kasutaja sisselogimise ja väljalogimise vormid (Login, Logout): Need komponendid võimaldavad kasutajal sisse logida või välja logida. Sisselogimise vorm võimaldab kasutajal autentida ennast ning kui autentimine on edukas, määratakse kasutaja olek ja kuvatakse rakenduses vastav sisu. Väljalogimise komponent võimaldab kasutajal oma sessiooni lõpetada ja kuvada sisselogimise vormi uuesti. Need komponendid on olulised, kuna need tagavad rakenduse turvalisuse ja privaatsuse.

- Navigeerimisriba (Navigationbar): See komponent kuvab kasutajale navigeerimisriba, mis sisaldab linke erinevatele lehtedele. Navigeerimisriba muudab kasutajale lihtsaks ja intuitiivseks erinevate osade vahel liikumise ning annab ülevaate rakenduse kättesaadavatest funktsioonidest.

- Komponendid filmide ja seansside kuvamiseks (Filmid, Seansid, FilmDetailid, Saal): Need komponendid vastutavad filmide, seansside ja saalide andmete kuvamise eest vastavalt kasutaja tegevustele ja valikutele. Näiteks võimaldab Filmid komponent kuvada loendit filmidest, Seansid komponent kuvab saadaolevaid seansse jne. Need komponendid on olulised, kuna need pakuvad kasutajale sisu, mida ta saab vaadata ja kasutada.

Navigationbar.js

Selle koodilõigu loomisel kasutasin Reacti NavLink komponenti, mis võimaldab luua dünaamilisi navigeerimise linke. Esiteks importisin NavLink komponendi 'react-router-dom' moodulist. Seejärel defineerisin funktsiooni Navigationbar, mis tagastab navigeerimisriba koos kolme lingiga: "Kodu", "Seansid" ja "Logout". Iga link on loodud NavLink komponendiga, mis võtab atribuudina sihtkoha (to) ja aktiivse klassi nime (activeClassName), kui link on aktiivne. Lõpuks eksportisin Navigationbar funktsiooni, et seda saaks kasutada teistes osades rakenduses.

Film.js

See komponent haldab filmide kuvamist, filtreerimist ja soovitamist vastavalt kasutaja valikutele.

handleŽanrChange, handleVanusepiirangChange, handleKeelChange

Funktsioonid võimaldavad kasutajal valida filtreid, nagu žanr, vanusepiirang ja keel, mis määravad, milliseid filme kuvatakse. Iga filtri muutmine käivitab vastava muutuse vastavas olekus, salvestades uue valiku. Seejärel koostatakse päring, mis põhineb valitud filtritel, ja laetakse vastavad filmid andmebaasist. Näiteks handleŽanrChange salvestab uue žanri valiku ja värskendab vastavalt filmide nimekirja vastavalt valitud žanrile. Sarnaselt käituvad handleVanusepiirangChange ja handleKeelChange, võimaldades kasutajal filtreerida filme vastavalt vanusepiirangule ja keelele.

useEffect

useEffect on funktsioon, mis võimaldab teha teatud toiminguid, kui komponent laetakse või muudetakse teatud tingimusi. Selles failis kasutatakse useEffecti kahe olulise toimingu jaoks: kasutaja vaadatud filmide laadimiseks ja kõikide filmide laadimiseks. useEffect käivitub komponendi laadimisel ning seejärel käivitatakse fetch päringud, et laadida andmed kasutaja vaadatud filmide ja kõikide filmide kohta. Saadud andmed salvestatakse seejärel vastavatesse olekutesse, et neid saaks rakenduse teistes osades kasutada. See võimaldab komponendil reageerida andmete muutumisele ja värskendada end vastavalt.

soovitaFilme

Soovitatud filmide leidmiseks analüüsitakse kasutaja vaadatud filme ja nende žanreid. Selleks koostatakse map žanrite ja nende esinemiste arvu kohta kasutaja vaadatud filmides. Seejärel leitakse kõige populaarsemad žanrid, filtreeritakse vaatamata filmid ja nende järgi vastavate žanrite filmide. Lõpuks kuvatakse ekraanil filmid, mis vastavad populaarsetele žanritele ja mida kasutaja pole veel vaadanud.

return

Rakendus kuvab filmid, mis vastavad kasutaja valitud filtritele, ning ka soovitatud filme. Filmiobjekt kuvatakse nii, et see sisaldab filmi pilti ja pealkirja ning võimaldab kasutajal klõpsata filmi üksikasjade vaatamiseks. Filmide kuvamiseks kasutatakse dünaamilist kuvamise loendit, mis põhineb filmide olekul, mis laaditakse ja värskendatakse vastavalt valitud filtritele ja kasutaja tegevusele.

Seansid.js

Haldab seansside kuvamist vastavalt API-st saadud andmetele.

useEffect

Kasutatakse useEffecti, et laadida seansside andmed API-st. Pärast andmete laadimist salvestatakse saadud seansside andmed komponendi olekusse. (setSeanssid)

return

Kuvatakse iga seanss eraldi Link komponendi sees, mis viib kasutaja vastava saali detailvaatesse. Iga seanss kuvatakse eraldi div-ina, mis sisaldab seansi filmi pealkirja, pilti, kuupäeva ja algusaja ning saali nime. Kasutatakse format funktsiooni, et kuupäeva ja aja formaati kohandada vastavalt vajadusele. Selle teegi kasutamise sain siit: https://stackoverflow.com/questions/34590369/formatting-a-date-string-in-react-native

Istekohad.js

Komponent võimaldab kasutajal valida istekohti saalis, lisada need ostukorvi, eemaldada neid, vaadata tšekki ja lõpuks osta piletid.

useEffect

useEffect funktsiooni kasutatakse, et laadida istekohtade andmed API-st vastavalt antud saali ID-le. Pärast andmete laadimist salvestatakse saadud istekohtade andmed

komponendi olekusse. See tagab, et komponent renderdataks õiged istekohad vastavalt valitud saalile. useEffect'i sõltuvuseks on saalId, et see käivituks iga kord, kui saali ID muutub.

groupIstekohadByReaNr

Funktsioon groupIstekohadByReaNr vastutab istekohtade grupeerimise eest rea numbri järgi, võimaldades neid kuvada rea kaupa. See funktsioon läbib iga istekoha ja lisab selle vastavasse rea numbri all olevasse listi. Seejärel tagastatakse grupeeritud istekohtade nimekiri.

handleClick

Selles funktsioonis, handleClick, kontrollitakse, kas antud istekoht on juba võetud (kasVõetud omadus). Kui istekoht ei ole juba võetud, siis kontrollitakse, kas see istekoht on juba valitud (valitudIstekohad listis). Kui antud istekohta ei ole veel valitud, lisatakse see valitudIstekohad listi, vastasel juhul eemaldatakse see listist. Lõpuks uuendatakse komponendi olekut vastavalt uutele valitud istekohtadele. See funktsioon võimaldab kasutajal klõpsata istekohal ja lisada või eemaldada selle valitud istekohtade listist.

leiaKeskmineKoht

Funktsioon leiaKeskmineKoht võtab sisendiks istekohtade nimekirja ja keskmise rea indeksi ning tagastab selle rea keskmise istekoha. Funktsioon esmalt sorteerib istekohad rea numbri järgi. Seejärel filtreerib need istekohad, mis kuuluvad antud keskmise rea indeksile. Kui selliseid istekohti leitakse, sorteeri need järjekorras vastavalt koha indeksile reas. Seejärel arvutatakse keskmine koht selle rea keskmiseks istekohaks. Kui selliseid istekohti ei leita, tagastatakse null.

Funktsiooni kasutatakse handleAddSeats funktsioonis selleks, et leida vabu istekohti ja lisada need valitud istekohtade nimekirja.

handleAddSeats

Funktsioon handleAddSeats võtab sisendiks arvu (arv), valitud istekohtade nimekirja (valitudIstekohad) ja kõigi istekohtade nimekirja (istekohad). Funktsiooni eesmärk on lisada valitud arv istekohti valitud istekohtade nimekirja, võttes arvesse saadaolevaid vabu istekohti. Alustuseks kopeeritakse valitud istekohtade nimekiri, et vältida otsest muutmist.

Seejärel filtreeritakse kõik saadaolevad vabad istekohad, eemaldades need, mis on juba valitud istekohtade nimekirjas.

Kui saadavalolevate istekohtade arv on väiksem kui soovitud arv, siis sätitakse vastav staatuse silt ja funktsioon lõpetatakse.

Järgnevalt algab tsükkel, kus soovitud arv istekohti lisatakse. Tsükkel jätkub seni, kuni soovitud arv istekohti on lisatud. Igas tsükli sammus proovitakse leida keskmine istekoht, kasutades funktsiooni leiaKeskmineKoht, mille tulemusel saadakse keskmine vaba istekoht reast. Kui keskmine istekoht on leitud (st see pole null), siis muudetakse see istekoht valituks, lisatakse uuendatudValitudIstekohade nimekirja ning eemaldatakse see saadaolevate vabade istekohtade nimekirjast. Seejärel vähendatakse soovitud istekohtade arvu ühe võrra.

Kui keskmine istekoht ei ole leitud (st see on null), siis muudetakse keskmist rida vastavalt loogikale, et otsida keskmist istekohta järgmisest reast. Seejärel jätkatakse tsükli järgmise sammuga kuni soovitud istekohtade arv on saavutatud.

handleRemoveSeat

Funktsioon handleRemoveSeat võimaldab kasutajal eemaldada ühe valitud istekoha ostukorvist.

removeAllSeats

Funktsioon eemaldab kõik valitud istekohad. Selleks läbib ta iga istekoha nimekirja, kontrollides, kas iga istekoht on ka valitud istekohtade nimekirjas. Kui leitakse ühtivus, muudetakse istekoha kasVõetud omadus false-ks, mis tähistab, et istekoht on valimata. Pärast kõigi valitud istekohtade eemaldamist värskendatakse istekohtade nimekirja vastavalt ning valitud istekohtade nimekiri tühjendatakse.

handleOstaPiletid

See funktsioon käivitub, kui kasutaja soovib osta valitud istekohtade jaoks pileteid. Kõigepealt arvutab see välja kogu valitud istekohtade arvu korrutades selle hinna ühe pileti eest, mis on 5 eurot. Seejärel koostab see päringu API-le, et saata valitud istekohtade andmed serverisse, et piletid oleksid ostetud. Pärast päringu objekti koostamist teostab see päringu, kasutades fetch funktsiooni, mille sihtkohana on serveri API endpoint, kuhu piletite ostuandmed saadetakse. Pärast vastuse saamist kontrollib see vastuse olekut. Kui vastus on edukas, kuvatakse konsolisse teade "Piletid ostetud edukalt!". Kui vastus on ebaõnnestunud, kuvatakse konsolisse veateade "Viga piletite ostmisel:" koos vastuse oleku

tekstiga. Kui päring ebaõnnestub, püütakse kinni viga ja kuvatakse konsolisse veateade koos veateatega.

handleNaitaTsekki

Funktsioon handleNaitaTsekki muudab olekut, et näidata ostu tšekki.

return

See osa koodist tagastab kasutajaliidese, kus kuvatakse istekohad ja nendega seotud toimingud. Iga istekoht renderdatakse vastavasse rea konteinerisse, kus saab neid klõpsates valida või eemaldada. Lisaks on olemas nupud istekohtade lisamiseks ja eemaldamiseks ning hoiatussõnum, kui valitud istekohtade arv ületab lubatud piiri. Kui valitud istekohti on, kuvatakse ka nende loend ning võimalus tšekki kinnitada, kuvades ostetud piletite kokkuhinda ja kinnitamisnuppu.

Filmidetailid.js

useEffect

useEffect kasutatakse selleks, et laadida filmi detailid API-st vastavalt filmi ID-le, mis saadakse useParams hookiga. Kui filmi ID muutub, laetakse uued detailid. See hook käivitatakse ainult siis, kui filmi ID muutub.

fetch

fetch meetodiga saadetakse päring filmi detailide laadimiseks API-st. Kasutatakse filmi ID-d, mis saadakse useParams abil, et saada õige filmi andmed.

Renderdamine

Kui filmi detailid on laaditud, renderdatakse need ekraanile. Näidatakse filmi pealkiri, pilt, žanr ja sisu. Lisaks on olemas nupp, mis viib tagasi seanssidele, kasutades Link komponenti.

Login.js

useState

useState hookiga haldame komponendi olekut, sealhulgas kasutaja sisestatud e-posti, parooli, kasutaja sisselogimisrežiimi (login or register), ja vigade olekut.

handleSubmit

Sündmusfunktsioon, mis käivitub vormi esitamisel. Selles funktsioonis kontrollitakse, kas kasutaja soovib sisse logida või registreerida. Kui kasutaja soovib sisse logida, saadetakse päring serverile, et kontrollida kasutajanime ja parooli. Kui kasutajanimi ja parool on õiged, siis kasutaja sisselogitakse ning kuvatakse vastav teade. Kui kasutajanimi ja parool ei ühti, kuvatakse vastav teade. Kui kasutaja soovib registreeruda, saadetakse uue kasutaja loomiseks päring serverile. Kui kasutaja loomine on edukas, kuvatakse vastav teade ja lülitatakse tagasi sisselogimisrežiimi. (Kuna registreerimisel kirjutad oma emaili ja parooli, siis username saadakse kätte emaili nimest enne @-i)

return

Kuvatakse vorm, kus kasutaja saab sisestada e-posti ja parooli. Samuti kuvatakse vastav teade, kui esineb viga sisselogimisel või registreerimisel. Kui kasutaja soovib sisse logida, kuvatakse nupp "Logi sisse". Kui kasutaja soovib registreeruda, kuvatakse nupp "Registreeri". Lisaks on olemas nupp, mille abil saab vahetada sisselogimisrežiimi.

Kui kasutaja soovib vahetada sisselogimisrežiimi, siis muudetakse vastavat olekut ja muudetakse nupu teksti vastavalt ("Logi sisse" või "Registreeri").

Logout.js

handleLogout

Funktsioon käivitub nupule klõpsamisel ja muudab kasutaja olekuks null, mille tulemusena kasutaja logitakse välja.

return

Kuvatakse lihtne vorm, mis sisaldab pealkirja "Logi välja" ja nuppu "Logout". Nupule klõpsamisel käivitatakse handleLogout funktsioon, mis logib kasutaja välja.

Abi otsimine ja lahendatud probleemid

Kogu minu kogemus React.js-i õppimisega algas algul videotest ja ChatGPT-st saadud abiga, eriti esimese komponendi Film.js loomisel. Kuid mida rohkem ma ise katsetasin ja uurisin, seda enam hakkasin mõistma Reacti süntaksit ja põhimõtteid. Oluliste nüansside mõistmiseks pöörusin tihti ka ChatGPT poole, eriti kui tuli luua midagi täiesti uut või kui tekkisid keerulisemad probleemid, mida ise lahendada ei osanud. Sellised olukorrad on dokumenteeritud ka minu GitHubi commitides. Esialgu oli funktsioonide loomine väljakutse, kuid ChatGPT abil sain kiiresti aru õigetest süntaksidest ja olulistest funktsioonidest.

Autentimise rakendamine React.js-is oli veidi erinev võrreldes minu varasema kogemusega Vue.js-iga, seega pidin lugema erinevaid artikleid ja dokumentatsiooni, et leida parimad lahendused. Kogu õppeprotsessi jooksul avastasin, et React.js on väga õpitav ja kui olen juba tuttav teiste front-end tehnoloogiatega, siis on see veelgi lihtsam. Üldiselt võib öelda, et lõpuks sain React.js-iga päris hästi läbi ja see muutus minu jaoks tuttavaks ja mugavaks tööriistaks.

Mida teeksin paremini või muudaksin? Mis oli raske?

Kahjuks tõsiasi on see, et aega pole päevas 48h ja kõike, mida soovid saavutada ei jõua ehk peab kuskilt natuke midagi kärpima. Siin on mõned funktsioonid, mille arendamiseks tulevikus sooviksin teha muudatusi ning kuidas ma kavatsen neid parandada ja täiustada.

soovitaFilme funktsionaalsus Filmid.js

Funktsiooni "soovitaFilme" saab arendada nii, et see võtab arvesse mitte ainult vaadatud filmide žanrit, vaid ka vanusepiirangut ja keelt. Selleks laadime alla kõik vaadatud filmide andmed ja analüüsime, millised žanrid, vanusepiirangud ja keeled on kõige populaarsemad. Seejärel valime välja soovitatud filmid, võttes arvesse nende populaarsust. Lõpuks kuvame need kasutajale.

Istekohad.js

handleAddSeats

See funktsioon oli minu projekti peamine takistus, kuna soovisin, et see toimiks algse plaani kohaselt. Ta peaks leidma keskmise rea numbri, selleks oleks ta pidanud eraldi funktsiooniga loendama kõik unikaalsed reanumbrid ja seejärel leidma keskmise. Seejärel toimuks sama põhimõttel nagu praegu, kui keskmist istekohta ei leita, liigub funktsioon järgmisele reale jne, kuni jõutakse viimasesse ritta. Seejärel naaseks see keskmise rea juurde ja liiguks sealt allapoole, kuni jõutakse reanumbrini 0, mil funktsioon lõpetatakse.

Probleem seisnes peamiselt selles, et see ei osanud õigesse ritta minna, kui reas polnud vabu kohti. Ma usun, et nüüd, kui arendaksin selle funktsionaalsuse praeguse funktsiooni põhjal, siis suudaksin sellega hakkama saada. Komistuskiviks oli see, et kartsin, et teised osad jäävad poolikuks, seega lõin hetkel sellise funktsiooni, mis töötab minu nelja reaga saalides. Ning see töötab, võttes keskmise rea, näiteks 2, seejärel liigub neljandale, seejärel 1 ja siis 0 reale. Samuti lisaksin siia funktsionaalsuse, et kõrvuti istmed on kõrvuti. Kuigi ta enamasti teebki seda by default, siiski leidub momente, kus ta ühe koha paneb keskmisele reale ja ülejäänud järgmisele reale keskele.

Sellel komponendi (Istekohad.js) loomine oli kõige keerukam osa projekti juures, kuna soovisin vältida valmislahenduste kasutamist internetist ning proovisin ise ekraanile kuvada kõiki istekohti. Lihtne oleks olnud võtta seating chart valmis koodina, kuid tahtsin ise pusida. Returnis aitas ChatGPT mul mõista, kuidas muutujate oleku määramisel saab näidata ekraanile asju, mis olid varasemalt peidus. Samuti õppisin olulisi süntakseid ja meetodeid, mis aitasid mul komponente efektiivsemalt luua. Kuigi see klass nõudis palju läbimõtlemist ja pingutust, oli see äärmiselt põnev ja õpetlik kogemus ning aitas mul oluliselt areneda.

Siin komponendis leidub veel kaks probleemi, mille lahendused teeksin järgnevalt:

Esiteks, kui kasutaja ostab piletid, soovin, et funktsioon handleOstaPiletid salvestaks ostetud filmi ning lisaks selle kasutaja vaadatud filmide hulka. Olen juba loonud vajaliku PUT-mappingu serveripoolses koodis. Selle järgmiseks sammuks oleks edastada Istekohad komponendile kasutaja ja seansi ID. Kui piletit ostetakse, peaksime otsima vastava kasutaja vaadatud filme ning leidma vastava seansi filmi. Seejärel kasutaksime PUT-mappingut kasutaja kontrolleris, et lisada see uuendatud vaadatud filmide hulka.

Teiseks, praegune piletite ostufunktsioon mõjutab kõigi sama saali seansside võetud istekohti. Selle lahendamiseks peaksin täiendama istekohtade tabelit, lisades sinna unikaalse seanssi identifikaatori. See võimaldaks igal seansil omada eraldi istekohtade olekut, tagades, et iga seansi istekohad on unikaalsed. Selline lähenemine võimaldaks meil salvestada istekohtade olekuid samades saalides, kuid erinevate seansside jaoks eraldi.

Kokkuvõte

Alguses oli väljakutse, kuna mul polnud varasemat kogemust Spring Booti ja React JSiga. Siiski aitasid YouTube'i videod mul mõista olulisi detaile ja õpetasid, kuidas projekti üles ehitada. Kuigi see protsess polnud algselt lihtne, oli see pigem põnev ja õpetlik. Kokku kulus aega umbes 20h (ma arvan). See kogemus aitas mul mõista, et veebirakenduste arendamine on väga põnev valdkond, ja sooviksin oma oskusi selles valdkonnas veelgi täiendada ja rakendusi edasi arendada. Kuigi tekkisid takistused, suutsin neist õppida ja tulla tugevamana välja.😎