MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

VIDEO STREAMING APP

LUCRARE DE LICENŢĂ

Absolvent: **Lucian-Alexandru ȘEICAN**

Coordonator ştiinţific: **Asist. Ing. Cosmina IVAN**

2020

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Lucian-Alexandru ȘEICAN**

VIDEO STREAMING APP

1. **Enunţul temei:** O aplicație de tip Video on-Demand care permite utilizatorilor să vizioneze online conținut video grupat în serii.

2. **Conţinutul lucrării:** Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie, Anexe.

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**:

5. **Data emiterii temei:** 1 februarie 2020

6. **Data predării:** 8 septembrie 2020

Absolvent: _____

Coordonator ştiinţific: _____

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) Șeican Lucian-Alexandru,
legitimat(ă) cu CI seria AX nr. 614943
CNP _____, autorul lucrării
VIDEO STREAMING APP
_____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Tehnologia Informației din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Septembrie a anului universitar 2019-2020, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data	Nume, Prenume
07.09.2020	Șeican Lucian-Alexandru

Semnătura

Cuprins

Capitolul 1. Introducere

Scopul acestui proiect este de a crea o aplicație de tip Video on-Demand care permite utilizatorilor să vizioneze fișiere video stocate în cloud fără a fi nevoie de descărcarea lor. De asemenea, aplicația va oferi o interfață de administrator care permite utilizatorilor autorizați să gestioneze utilizatorii și fișierele video grupate sub formă de serii.

1.1. Contextul proiectului

În ultimii ani, progresul tehnologic a dus la o schimbare semnificativă în modul în care consumăm conținut media. Astfel, s-a ajuns de la o piață dominată de televiziune la una dominată de entertainment-ul online, una din componentele majore ale acestuia fiind vizualizarea de video-uri online. Evoluția tehnologiei a permis streaming-ul prin internet de fișiere video de dimensiuni tot mai mari și la viteze mai mari.Așa au luat naștere serviciile de Video on-Demand sau de Live Streaming.

În prezent, există un număr foarte mare de astfel de aplicații, atât gratuite (precum Youtube, Vimeo, Dailymotion și altele) dar și plătite(Netflix , HBO Go, Hulu, Amazon Prime, etc.). Acestea sunt unele dintre cele mai populare moduri de a consuma conținut video , Youtube având un număr estimat de 2 miliarde de utilizatori, iar Netflix aproximativ 160 de milioane de abonați.

Așadar, acest domeniu este unul în creștere, în care noi aplicații apar mereu, și cele existente inovează în continuu pentru a continua să atragă utilizatori.

1.2. Motivația proiectului

Așa cum am menționat și anterior, acest tip de aplicații este unul foarte popular,fiind parte a vieților noastre de zi cu zi .

Motivația pentru crearea acestui proiect este de a cunoaște modul în care funcționează aceste aplicații pe care le utilizăm atât de des, infrastructura din spatele lor, tehnologiile folosite și problemele pe care le pot întâmpina.

Prin realizarea unei astfel de aplicații la o scară mai mică am considerat că pot învăța foarte multe despre ce înseamnă un serviciu Video on-Demand din punct de vedere tehnic.

1.3. Conținutul proiectului

Proiecul va fi structurat astfel:

Capitolul 2 va descrie obiectivele proiectului,atât obiectivul principal cât și obiectivele funcționale sub forma unei liste de funcționalități care se doresc implementate și o serie de obiective nefuncționale.

Capitolul 3 va prezenta o serie de concepte relevante pentru aplicația propusă precum Video on-Demand, Video Streaming, sisteme de recomandare și comparații între aplicațiile și serviciile existente.

Capitolul 4 va descrie arhitectura conceptuală a sistemului, platformele de dezvoltare și serviciile externe utilizate. În plus vor fi prezentate cerințele funcționale, nefuncționale și cazurile de utilizare ale aplicației.

Capitolul 5 va prezenta arhitectura completă a sistemului, tehnologiile folosite ,modul de integrare și de utilizare al acestora pentru implementarea diferitelor funcționalități și structura bazei de date.

Capitolul 6 va descrie modul în care se face testarea și validarea aplicației și tipurile de testare utilizate.

Capitolul 7 va descrie modul de instalare al aplicației și un set de instrucțiuni de utilizare al acesteia.

Capitolul 8 va prezenta o serie de concluzii cu privire la aplicația dezvoltată, o analiză a acesteia și posibilitățile de dezvoltare ulterioară.

Capitolul 2. Obiectivele Proiectului

În cadrul acestui capitol vor fi prezentate principalele obiective ale proiectului din punctul de vedere al funcționalităților puse la dispoziție utilizatorilor, a tehnologiilor folosite pentru implementare dar și al proprietăților non-funcționale.

2.1. Obiectiv principal

Principalul obiectiv al proiectului constă în realizarea unei aplicații web de tip Video on-Demand care va oferi funcționalitățile principale caracteristice acestui tip de aplicații într-o manieră intuitivă și ușor de utilizat.

O astfel de aplicație are ca rol oferirea accesului la o librărie de fișiere video stocate în cloud spre utilizatori. Aceștia vor putea viziona video-uri fără a fi necesară descărcaea lor în avans. În plus, utilizatorii vor putea căuta. Video-urile vor fi grupate în serii, aplicația oferind posibilitatea de a căuta și sorta serii, adaugarea acestora la lista de favorie și vizualizarea de recomandări.

Pe lângă interfața de utilizator care oferă acces la funcționalitățile prezentate, aplicația va oferi o interfață de administrator. Aceasta va permite utilizatorilor autorizați sa gestioneze informațiile care țin de fișierele video și de utilizatori.

2.2. Obiective funcționale

Obiectivele funcționale ale proiectului sunt următoarele:

- *Autentificare*: Aplicația va oferi utilizatorilor posibilitatea de a se loga în conturile proprii existente sau de a crea un cont nou.Utilizatorii trebuie sa fie logați pentru a putea sa vizualizeze video-uri. Vor exista două tipuri de utilizator: utilizator normal, care poate utiliza funcționalitățile principale ale aplicației prin interfața de utilizator, respectiv administratori care vor putea accesa interfața de administrare a aplicației și realiza operații specifice acesteia

- *Căutare și sortare serii:* Aplicația va permite utilizatorilor sa caute o serie după nume și să sorteze seriile după nume sau gen.

- *Vizionare serii:* Aplicația va permite utilizatorilor să acceseze pagina unei serii. Aceasta va conține o listă cu toate episoadele dintr-o serie și un player video care va reda episodul ales de către utilizator.

- *Adăugare serie la lista de favorite:* Tot de pe pagina seriei, utilizatorul are opțiunea de a adăuga seria la lista de favorite.

- *Vizualizare listă de serii favorite:* Aplicația oferă utilizatorului posibilitatea de a vedea o listă de serii favorite.

- *Vizualizare recomandări:* Aplicația oferă utilizatorului sugestii de serii de interes pe baza interacțiunilor trecute între utilizator și seriile existente.

- *Gestiune serii:* Aplicația permite utilizatorilor cu drepturi de administrator sa gestioneze seriile existente. Astfel, se va permite ștergerea, editarea sau adăugarea de serii.

- *Gestiune utilizatori:* Aplicația permite utilizatorilor cu drepturi de administrator să vizualizeze utilizatorii existenți precum și să adauge, să șteargă și să editeze conturi de utiilizatori.

2.3. Obiective non-funcționale

Aplicația își propune să respecte o serie de obiective din punct de vedere non-funcțional.

Din punct de vedere al *utilizabilității* aplicația va oferi o interfață intuitivă și ușor de utilizat , cu meniuri care fac navigarea între pagini foarte ușoară și un aspect similar cu alte aplicații de acest tip. Astfel, utilizatorii se pot familiariza foarte rapid cu aplicația și modul de utilizare al acesteia.

Din perspectiva *securității* , aplicația asigură siguranța datelor sensibile precum parolele prin criptarea lor. În plus, funcționalitățile oferite utilizatorilor nelogați sunt limitate, deci utilizatorii trebuie să se autentifice pentru a accesa multe dintre funcțiile de bază ale acesteia.

Un alt obiectiv non-funcțional important este *scalabilitatea*. Aplicația utilizează servicii externe pentru stocarea de fișiere video și pentru generarea recomandărilor, acestea fiind punctele critice din punctul de vedere al scalabilității. Aceste servicii externe suportă volume de date variate, oferind un grad bun de scalabilitate. Conceptualizarea modelului local se va realiza având la bază o cât mai bună scalabilitate a sistemului(prin definirea și utilizarea unor metadate în acest scop).

Capitolul 3. Studiu Bibliografic

În cadrul acestui capitol vor fi prezentate concepte referitoare la platformele de gestionare de conținut video și mai exact la tehnologia Video Streaming, tehnologie care va sta la baza aplicației propuse.

3.1. Video on Demand (VoD)

Video-on-Demand este un tip de serviciu care permite utilizatorilor să acceseze și să vizualizeze conținut video dintr-un catalog de video-uri existente.Popularitatea serviciilor de tip Video-on-Demand a crescut foarte mult în ultimii ani, ajungând să fie principalul mod prin care utilizatorii consumă conținut video, depășind metodele clasice precum Televiziune sau Cinema.[1]

Platformele VoD existente se împart în două mari categorii:

Free: Platforme care permit utilizatorilor sa vizualizeze video-uri gratuit. Acest tip de platforme oferă conținut foarte divers, produs în general de către creatori de conținut individuali. Printre aceste tipuri de aplicații se numără Youtube, Vimeo, Dailymotion, etc.

Paid: Platforme premium în cadrul cărora utilizatorii trebuie să plătească pentru accesarea vide-urilor. De obicei conținutul de pe astfel de platforme este produs de studio-uri sau alte companii.Exemple de astfel de platforme sunt Netflix, Amazon Prime Video,HBO Go, Hulu, etc. Acestea se împart la rândul lor în mai multe subcategorii în funcție de modul în care utilizatorii pot accesa conținutul video astfel:

- Transactional Video-on-Demand(TVOD) : au la bază o metodă de distribuție care presupune ca utilizatorii sa plătească fiecare „bucată” de conținut video pe care dorește să o acceseze
- Subscription Video-on-Demand(SVOD): are la bază un suscription model care prespune ca utilizatorii să plătească o taxă regulată(ex: lunară) pentru a avea acces la conținut nelimitat. [1]

Platformele de tip VOD se bazează pe tehnologia video streaming care va fi descrisă în continuare.

3.2. Video Streaming

3.2.1. Evoluția sistemelor VoD

Apariția și dezvoltarea sistemelor VoD se datorează în mare parte tehnologiei și serviciilor de tip Cloud Computing. Conform [2] Cloud Computing este un model care permite accesul la cerere la un grup de resurse computaționale neomogene precum servere, stocare, aplicații sau servicii cu posibilitatea de a accesa aceste resurse cu efort

[1] https://en.wikipedia.org/

managerial minim și dependență scăzută față de furnizorii de servicii de internet. Un exemplu de arhitectură Cloud Computing se poate observa în figura următoare:

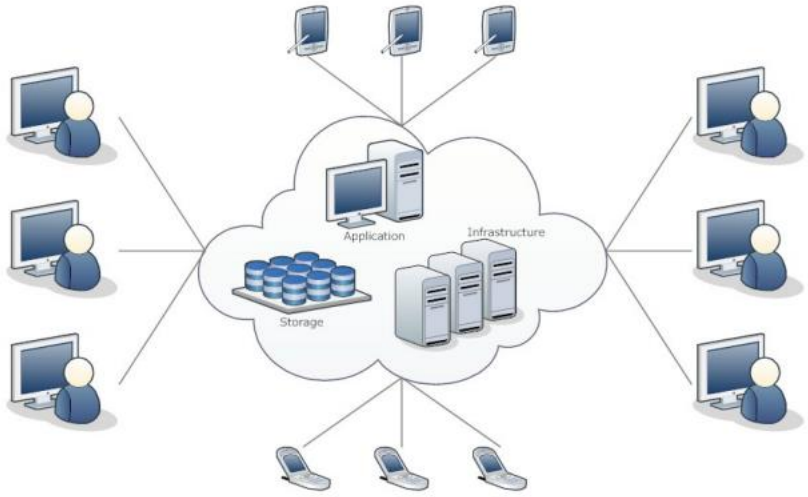


Figura 3.1 Arhitectura de tip Cloud computing preluată din [2]

Așadar, Cloud Computing a schimbat modul în care utilizatorii pot accesaresurse și servicii prin intermediul internetului. Un astfel de exemplu este reprezentat de tehnologiile Video Streaming și Live Streaming care stau la baza serviciilor de tip VoD. Aceste tehnologii vor fi descrise în capitolul următor.

Conform [3] , prima tehnologie de tip ”streaming” a apărut în 1995, aceasta făcând posibilă transmiterea de conținut audio prin internet, și ceva mai târziu și de conținut video.

Primele servicii de tip VoD datează încă dinaintea apariției tehnologiei streaming, acestea facându-și apariția la inceputul anilor 90 precum un serviciu dezvoltat de AT&T in 1990 care folosea casete ca și sursă de streamuri vide în timp real. Până în 1992 s-a făcut trecerea de la casete la discuri și DRAM. Un număr de servicii de tip VoD experimentale au apărut în perioada următoare precum Bell Atlantic în 1993 în SUA, sau Cambrige Digital Interactive Television Trial în 1994 în Anglia. Acestea foloseau tehnologia ADSL. În 1998 Kingston Communications a lansat primul serviciu VoD în totalitate comercial și a început să integreze televiziunea și accesul la internet prin ”cutii” setate folosind IP-uri și care transmiteau datele folosind ADSL. În 2006 a apărut serviciul Sky Anytime pentru PC-uri , folosind tehnologia peer-to-peer. Aceasta a fost apoi implementată și de alți distribuitori de servicii similare, fiind folosită și în ziua de azi de toate serviciile VoD majore.

Popularitatea acestor servicii a continuat sa crească, în 2010 ajungându-se ca 80% dintre utilizatorii de internet din SUA să vizualizeze sau descarce video-uri folosind aplicații de tip VoD[2]

[2]. http://echelonstudios.us/

3.2.2. Video Streaming și Live Streaming

Tehnologia Video Streaming presupune redarea unui fișier de tip video stocat remote fără a fi necesară descărcarea sa prealabilă pe dispozitivul utilizatorului.Astfel,un fișier video poate fi redat în timpul transmiterii acestuia spre dispozitivul client. Pentru a fi transmis prin rețea conținutul video trebuie să fie compresat și apoi decompresat odată ajuns la utilizator pentru a putea fi vizualizat.

Dacă pentru video streaming conținutul video trebuie sa existe și să fie stocat pe un server înainte de a putea fi redat, în cazul tehnologiei live streaming video-urile care ajung spre utilizator sunt generate în timp real.

Deși experiența utilizatorului pentru aceste două tipuri de streaming este similară, modul lor de funcționare este foarte diferit. Pentru proiectul propus, ne vom concentra pe tehnologia video streaming ca soluție de transmitere de conținut video spre utilizatori.

Schema următoare prezintă diferitele tipuri de Video Streaming și Live Streaming existente:

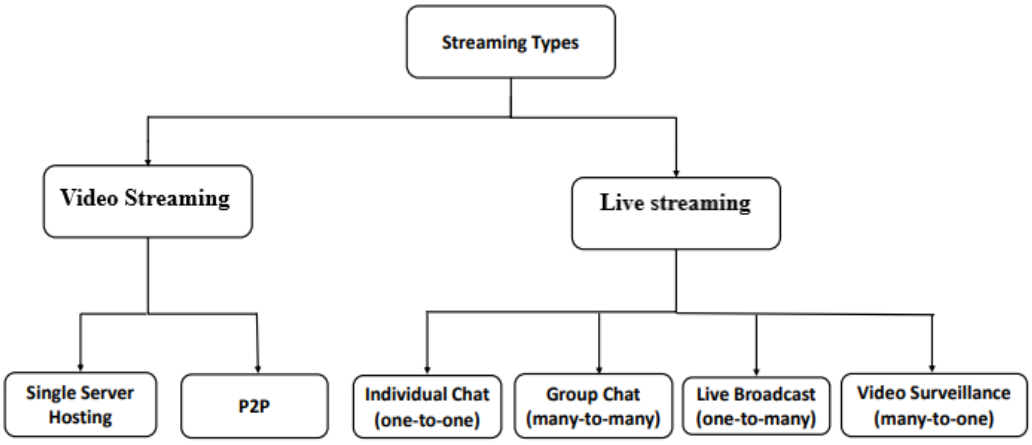


Figura 3.2 Aplicabilitatea soluțiilor de tip streaming preluată din [4]

3.3. Sisteme de recomandare

Conform [5] un sistem de recomandare poate fi considerat orice sistem care produce recomandări personalizate ca și rezultat sau are efectul de a ghida utilizatorul într-o manieră personalizată spre obiecte de interes dintr-o gamă mare de opțiuni posibile.

Odată cu creșterea în popularitate a aplicațiilor și serviciilor web a crescut și importanța sistemelor de recomandare. Astfel, acestea pot fi un factor decisiv de departajare între aplicații similare din punctul de vedere al utilizatorilor.

Există două mari paradigme de funcționare a sistemelor de recomandări: colaborative și bazate pe conținut.

Sisteme colaborative

Principala caracteristică a sistemelor colaborative este faptul că se bazează exclusiv pe intereacțiuni trecute înregistrate între utilizatori și obiectele care reprezintă subiectul interacțiunii. Aceste interacțiuni sunt înregistrate în matrici de interacțiune utilizator-obiect. Conform acestei paradigme, interacțiunile înregistrate sunt suficiente pentru a detecta obiecte similare și a face predicții pe baza acestora .

După cum este menționat în [6], sistemele colaborative se pot împărți la rândul lor în mai multe subcategorii în funcție de relațiile considerate între utilizatori și obiecte precum Sisteme Colaborative bazate pe Obiecte sau Sisteme Colaborative bazate pe Utilizatori.

Principalul avantaj al acestui tip de sisteme este faptul ca nu necesită informații despre utilizatori și obiecte. În plus, cu cât crește numărul de utilizatori care interacționează cu obiectele, cu atât crește precizia predicțiilor. Dezavantajul acesor sisteme constă în faptul că, din cauza faptului că predicțiile se fac pe baza interacțiunilor treute, pentru utilizatori noi este greu să se genereze predicții precise. Această problemă poate fi adresată prin recomandarea de obiecte random utilizatorilor noi, sau prin recomandarea de obiecte populare către utilizatorii noi.

Sisteme bazate pe conținut

Spre deosebire de sistemele colaborative, sistemele bazate pe conținut folosesc informații suplimentare , nu doar despre interacțiuni între utilizatori și obiecte. De exemplu, pentru un sistem de recomandare de filme și seriale, astfel de sisteme ar putea utiliza informații despre utilizator precum vârsta, sexul, ocupația,genurile favorite, actori favoriți sau orice altfel de informații personale. Astfel, ideea din spatele acestor sisteme este de a construi un model bazat pe informațiile disponibile care poate să explice interacțiunile observate. Odată ce acest model este determinat, e foarte ușor de generat sugestii pe baza informațiilor despre utilizatori.

Sistemele bazate pe conținut suferă mai puțin în cazul utilizatorilor noi, singurii care ar suferi de această problemă ar fi utilizatori cu caracteristici noi. Această problemă se diminuează cu cât sistemul este mai vechi.

Aceste două tipuri de sisteme de recomandare pot fi utilizate împreună pentru crearea de sisteme hibride.

3.4. Servicii Video on Demand

Datorită creșterii rapide în popularitate a platformelor Video on-Demand, și numarul acestor platforme este în continuă creștere. Cele mai importante aplicații de tip VoD bazate pe abonamen sunt :

- Amazon Prime Video
- Apple TV+
- HBO Go
- Netflix
- Hulu
- Disney+

O comparație din punctul de vedere al funcționalităților oferite de aceste servicii:

	Suport Pentru Dispozitive Multiple	Vizionare Offline	Streaming Conținut Ultra HD (4K)	Număr Maxim de Streamuri Simultane Pentru Același Utilizator
Amazon Prime Video	Da	Da	Da	3
Apple TV+	Da	Da	Da	6
HBO Go	Da	Nu	Nu	3
Netflix	Da	Da	Da	4
Hulu	Da	Da	Da (limitat la unele dispositive)	1
Disney+	Da	Da	Da	4

3.1 Analiza comparativă a serviciilor VoD

Pe lângă aceste servicii premium, bazate pe abonament, există serviciile de tip Viedo on Demand gratuite, dintre care cele mai populare sunt:

- Youtube
- Dailymotion
- Vimeo

Aceste servicii oferă utilizatorilor posibilitatea de a încărca și edita propriile video-uri. O altă funcționalitate importantă pe care acestea o oferă este reprezentată de partea de statistici sau ”analytics” care permite utilizatorilor să aibă acces la o gamă variată de informații cu privire la video-urile încărcate precum timpul petrecut de alți utilizatori vizionând acest conținut video, locația lor, date demografice, aprecieri , comentarii, distribuiri și altele. Multe dintre aceste funcționalități sunt disponibile gratuit, unele fiind totuși parte din planurile premium, destinate creatorilor de conținut.

Monetizarea acestor platforme se bazează în mare parte pe reclame.

3.5. Servicii hosting video

Pentru stocarea și redarea fișierelor video aplicația propusă va utiliza un serviciu de hosting video, acestea oferind infrastructuri bine dezvoltate care oferă un număr mare de funcționalități.

De obicei, o platformă video onlie oferă o interfață(de obicei o aplicație web sau mobile) prin care utilizatorii pot interacționa cu un serviciu de hosting video. În cazul proiectului de față, aplicația implementată va oferi această interfață care va avea în spate un serviciu de hosting video existent.

Cele mai cunoscute și populare astfel de servicii în momentul de față sunt:

- Dailymotion
- Vimeo
- Youtube

În continuare vom prezenta o comparație între aceste 3 servicii din punct de vedere al specificațiilor tehnice și funcționalităților pe care le oferă fiecare dintre ele:

	Dailymotion	Vimeo	Youtube
Video Player	HTML5 video sau Flash	HTML5 video sau Flash	HTML5 video sau Flash
Video Format	Ogg Theora/Vorbis sau H.264, Sorenson codec	H.264, AV1	H.264, Sorenson codec, VP8, VP9
Max file size(MB)	2048	500 (Basic); mai mult cu planurile plătite	128,000 pentru utilizatorii verificați
Max video length(min)	720	Nelimitat	15 pentru utilizatori normali, 720 pentru utilizatori verificați
Max Resolution	4096×2160	4096×2160	7,680×4320
Video bitrate(kbit)	-	Variabil	Variabil
Audio bitrate(kbit)	96~192	320	64-192
Stereo sound	Da	Da	Da
3D capable	-	Da	Da
60 FPS support	Da	Da	Da

3.2 Specificații tehnice pentru serviciile importante de hosting video

Video player: Tehnologia utilizată pentru redarea unui video în browser.

Video format: Tehnologia folosită pentru „împachetarea” datelor unui fișier video pentru ca acestea să poată fi stocate

Audio and video bitrate: numărul de biți de informație folosiți pentru a codifica o unitate de timp dintr-un fișier video. Bitrate-ul influențează dimensiunea și calitatea unui video: cu cât bitrate-ul unui fișier este mai mare, calitatea sa va crește, dar la fel se va întâmpla și cu dimensiunea acestuia.

3D Capable: posibilitatea de redare de fișiere video care folosesc tehnologia 3D.

60 FPS Support: suport pentru fișiere video care rulează la 60 de cadre pe secundă(Frames Per Second). Numărul de cadre pe secundă reprezintă o măsură a modului în care se afișează mișcarea într-un video. Astfel , cu cât numărul de cadre pe secundă este mai mare, mișcarea va fi mai fluidă.În prezent, 60 FPS este un standard utilizat de un număr foarte mare de platforme.

3.6. Concluzii

Serviciile Video on-Demand sunt unele dintre cele mai populare aplicații în momentul de față, numărul de utilizatori al acestora fiind în continuă creștere.Aceste aplicații sunt foarte complexe,fiind construite sa lucreze cu volume mari de date și integrând un număr mare de tehnologii precum Peer to Peer, Cloud Computing, streaming, sisteme de recomandare, și altele.

Aplicația propusă are ca scop implementarea unui serviciu Video on-Demand la o scară redusă, dar oferind funcționalitățile principale și utilizând tehnologiile adecvate pentru realizarea acestui tip de aplicații.

Capitolul 4. Analiză şi Fundamentare Teoretică

În cadrul acestui capitol va fi prezentată arhitectura conceptuală a sistemului, platformele de dezvoltare utilizate pentru implementarea aplicației, cerințele funcționale și non-funcționale ale sistemului și o descriere a cazurilor de utilizare.

4.1. Arhitectura conceptuală a sistemului

Aplicația va fi construită conform următoarei arhitecturi conceptuale:

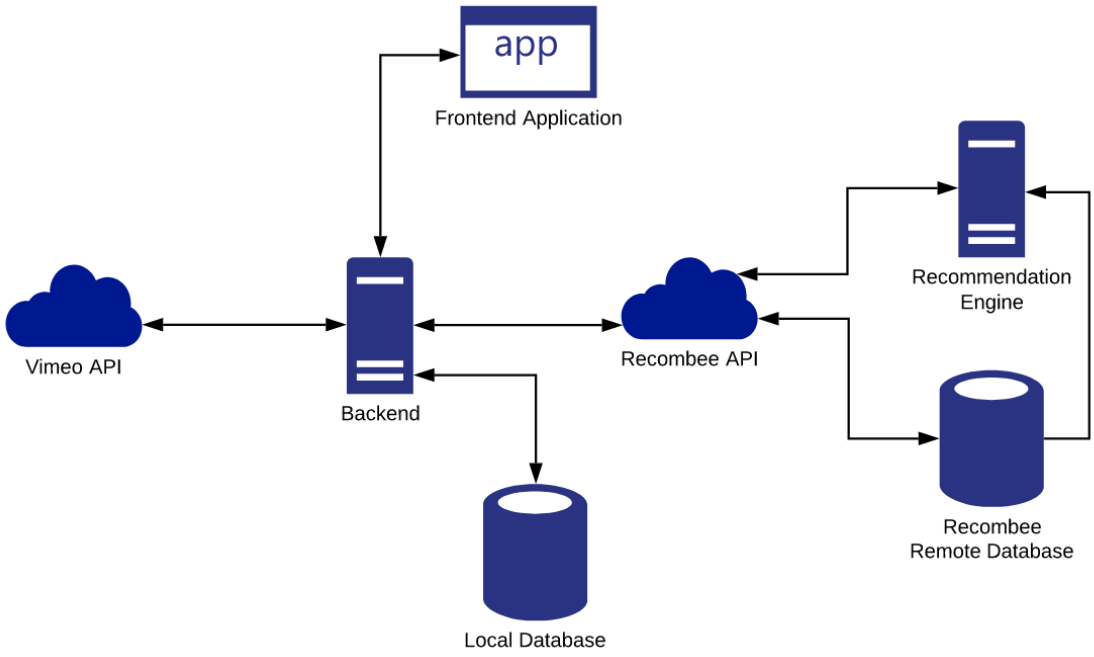


Figura 4.1 Diagrama conceptuală a sistemului

După cum se poate observa din diagrama prezentată, la baza acestui sistem stă aplicația de backend care implementează logica inernă și facilitează comunicarea și transferul de date între toate celelalte componente ale sistemului. Astfel, fiecare dintre aceste componente îndeplinește un rol bine definit după cum urmează:

Baza de date locală: păstrează informații despre utilizatori și video-uri

Vimeo API: o interfață utilizată pentru a realiza gestiunea fișierelor video(upload, edit, delete)

Recombee API: o interfață utilizată pentru gestiunea interacțiunii dintre utilizatori și video-uri precum și returnarea de recomandări

Baza de date remote Recombee: folosită pentru stocarea datelor referitoare la interacțiunile dintre utilizatori și video-uri

Motorul de recomandări: determină recomandările folosind datele stocate în baza de date remote

Aplicația frontend: interfața utilizator care facilitează interacțiunea dintre aceștia și celalale servicii

Așadar, în cadrul acestei aplicații se realizează o distribuție bine definită a datelor și funcționalităților în funcție de tipul acestora. Descrieri mai detaliate a componentelor vor fi prezentate în subcapitolele următoare.

4.2. Platformele de dezvoltare

4.2.1. Backend

Partea de backend a aplicației va fi realizată folosind Spring [3]. Acesta este un framework open source care oferă un model de configurare și programare a aplicațiilor Java. Funcționalitățile de bază ale Spring pot fi folosite pentru orice tip de aplicație Java, dar prezintă și extensii specifice construirii de aplicații web.

Una din cele mai importante funcționalități oferite de către Spring este containerul de Inversiune a Controlului (Inversion of Control container), cunoscut și sub numele de Dependency Injection. Această tehnologie presupune ca obiectele să își definească propriile dependințe doar prin argumente ale metodei constructor sau argumente ale unei metode factory. Astfel , containerul IoC este responsabil pentru managementul ciclurilor de viață al obiectelor, prin gestiunea creării, inițializării și configurării acestora. Obiectele create de către container sunt cunoscute și sub numele de ”beans”.

Am ales să utilizez framework-ul spring datorită flexibilității sale, facând ca aplicația să aibă o structura care ușurează modificarea funcționalităților existente sau adăugarea de funcționalități noi. Așadar, componenta debackend a aplicației va respecta o arhitectură layered după cum urmează:

[3] https://spring.io/

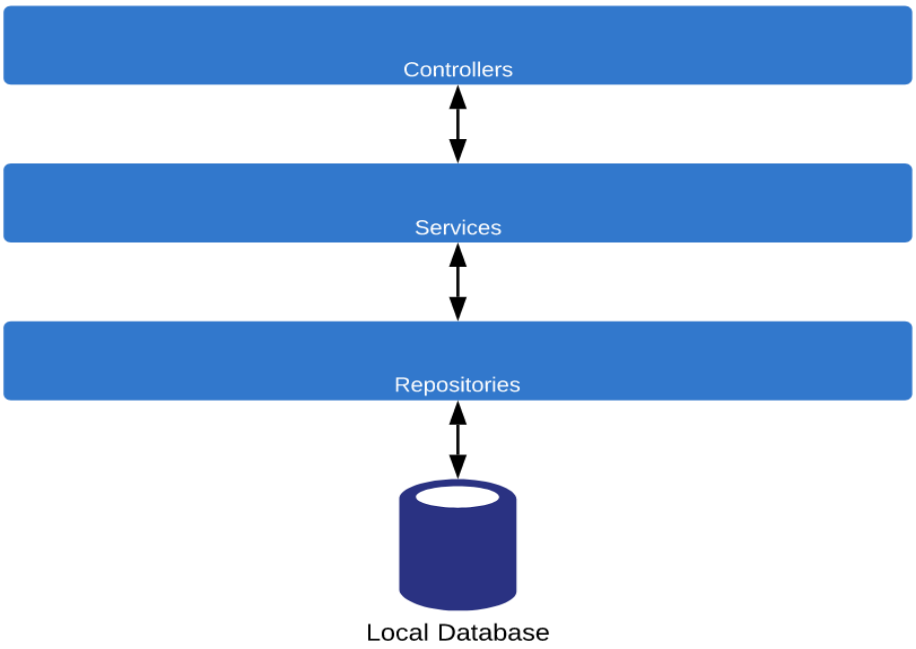


Figura 4.2 Arhitectură generică de tip Layered

Această arhitecură presupune o separare foarte clară a responsabilităților în cadrul aplicației. Așadar, layer-ul de Repositories se ocupă cu facilitarea comunicării cu baza de date prin oferirea de interfețe pentru citire, inserare, modificare sau ștergere de date. Layer-ul service implementează logica aplicației și comunicarea acesteia cu servicii externe, iar layer-ul Controllers definește endpoint-urile prin care se vor apela metodele implementate. Această arhitectură folosește tehnologia Dependency Injection, fiecare obiect din clasele Controllers definește obiecte de tip Services ca argumente în constructor, iar obiectele din clasele Services definesc obiecte de tip Repositories într-un mod similar.

4.2.2. Frontend

Pentru realizarea părții de frontend al aplicației este utilizat Vue Js.[4] Acesta este un framework JavaScript de tip Model-View-Viewmodel(MVVM) folosit pentru construirea interfețelor de utilizator. Vue este un framework reactiv, astfel încat componentele sale reactive se vor actualiza în timp real,este versatil și oferă o performanță bună, fiind rapid fară a necesita eforturi mari de optimizare.

Datorită faptului că este un framework de tip Model-View-Viewmodel,acesta permite o structurare foarte bună și posibilitatea de reutilizare a componentelor. Acest model presupune existența de modele sau componente, care sunt părți mici, independente, care sunt utilizate in view-uri reprezentând paginile propriu zise ale aplicației. În cadrul view-urilor se realizează și partea de transmitere de date și comenzi spre componente, sau partea viewmodel din pattern-ul MVVM.

[4] https://vuejs.org/

Schema următoare ilustrează relațiile dintre cele 3 componente ale acestui pattern:

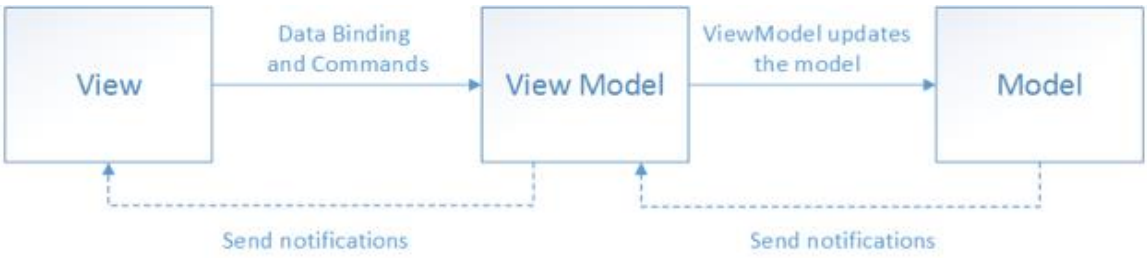


Figura 4.3 Model-View-ViewModel [5]

Comparativ cu pattern-ul MVC unde componenta Controller care manipulează datele Modelelor când se fac cereri de la View, în MVVM se introduce componenta ModelView care nu manipulează direct Modelul. În schimb aceasta permite comunicarea directă între componentele View și Model.

Am ales să folosesc Vue.js pentru implementarea componentei de frontend a aplicației deoarece este un framework foarte flexibil și performant. În plus , structura componentelor face ca proiectul să fie foarte ușor de înțeles și oferă un grad mare de reutilizabilitate ale acestora.

4.3. Servicii externe

Pentru realizarea unora din funcționalitățile importante, aplicația utilizează servicii externe. Astfel , pentru generarea recomandărilor este folosit Recombee API, iar pentru gestionarea stocării fișierelor video este folosit Vimeo API. Acestea vor fi descrise în cadrul acestui capitol.

4.3.1. Recombee

Recombee este un serviciu care oferă utilizatorilor posibilitatea de a accesa o bază de date remote și un motor de recomandări remote. Acesta oferă recomandări în timp real în funcție de informațiile stocate în baza de date.

Pentru determinarea recomandărilor Recombee folosește o serie de metode precum filtrarea colaborativă, filtrarea bazată pe conținut, sau metode bazate pe inteligența artificială. Motorul de recomandări determină automat cele mai potrivite metode care vor fi folosite într-o anumită situație în funcție de datele primite de la utilizatori.

[5] https://docs.microsoft.com/

Astfel, acest serviciu oferă recomandări personalizate în timp real, adaptându-se la cerințele utilizatorului atât din punctul de vedere al tipului lor dar și al volumului de date, oferind scalabilitate bună.

Accesul la aceste resurse se face prin intermediul unui API care poate fi folosit în sisteme bazate pe diverse limbaje precu Javascript, Python, Ruby, Java, PHP sau C#. Integrarea și utilizarea acestui serviciu se face foarte ușor, prin instalarea unui pachet și apelarea de metode pentru introducerea de date în baza de date remote(date despre interacțiunile dintre utiliatori și obiecte) sau pentru generarea de recomandări. Pe lângă API, Recombee oferă și o interfață de UI care permite administrarea bazei de date remote, personalizarea de reguli pentru recomandări sau statistici.

Datorită flexibilității, numărului mare de funcționalități și a ușurinței de integrare și utilizare am considerat ca Recombee este un sistem de recomandări care se potrivește aplicației propuse.

4.3.2. Vimeo

Vimeo [6] este o platformă de video hosting care permite stocarea și gestionarea remote de fișiere video. Aceasta pune la dispoziție un API pentru integrarea sa în aplicații. Prin intermediul API-ului utilizatorii pot încărca sau șterge fișiere video, pot edita titlul și descrierea sau pot schimba setările de securitate ale unui video. Vimeo API oferă suport nativ pentru limbajele PHP , Python și Node.js , dar prin intermediul unor librării third party poate fi utilizat în alte limbaje sau framework-uri.

Datorită serviciului de video hosting robust din spatele Vimeo API și a ușurinței de integrare și utilizare acesta reprezintă o soluție foarte bună pentru multe aplicații, oferind scalabilitate și funcționalități potrivite pentru diverse tipuri de utilizatori.

4.3.3. Firebase Cloud Storage

Firebase Cloud Storage [7] este o soluție de stocare în cloud care permite păstrarea de fișiere remote, oferind suport pentru un număr mare de platforme. Accesul la această platformă de stocare se face folosind un API a cărui utilizare va fi descrisă în cadrul capitolului 5.

Așadar, în proiectul propus va fi folosit Firebase Storage pentru stocarea de imagini aferente seriilor(fiecare serie are o imagine asociată). Astfel, imaginile vor fi stocate în cloud iar în baza de date locală vor fi stocate doar URL-urile spre imagini care vor fi folosite pentru afișarea lor.

Datorită suportului nativ pentru Javascript care oferă metode și clase wrapper ce simplifică utilizarea API-ului dar și a infrastructurii Google Cloud Platform din spatele Firebase Cloud Storage care oferă performanță și scalabilitate bune, am considerat că aceasta este o soluție potrivită pentru proiectul propus.

[6] https://vimeo.com/

[7] https://firebase.google.com/

4.4. Cerințe funcționale

Aceste cerințe definesc comportamentul, sau cu alte cuvinte funcționalitățile pe care o aplicație trebuie să le pună la dispoziție utilizatorilor. Cerințele funcționale ale acestei aplicații sunt:

Gestionare utilizarori	1. Creare cont 2. Login 3. Logout 4. Resetare parolă 5. Vizualizare cont personal 6. Vizualizare utilizatori 7. Editare utilizatori 8. Ștergere utilizatori
Administrare fișiere video	1. Upload video 2. Grupare video in serii 3. Redenumire video 4. Ștergere video
Interacțiune cu conținutul Video	1. Vizualizare listă de serii 2. Vizualizare serii recomandate 3. Căutare și sortare serii 4. Vizionare de serii 5. Adăugare serie la lista de favorite 6. Rating serie

Tabel 4.1 Categorii de cerințe funcționale

4.5. Cerințe non-funcționale

În continuare sunt prezentate cerințele non-funcționale ale sistemului, cerințe care nu fac referire la funcționalități ci la parametrii după care se poate judeca modul de funcționare al sistemului:

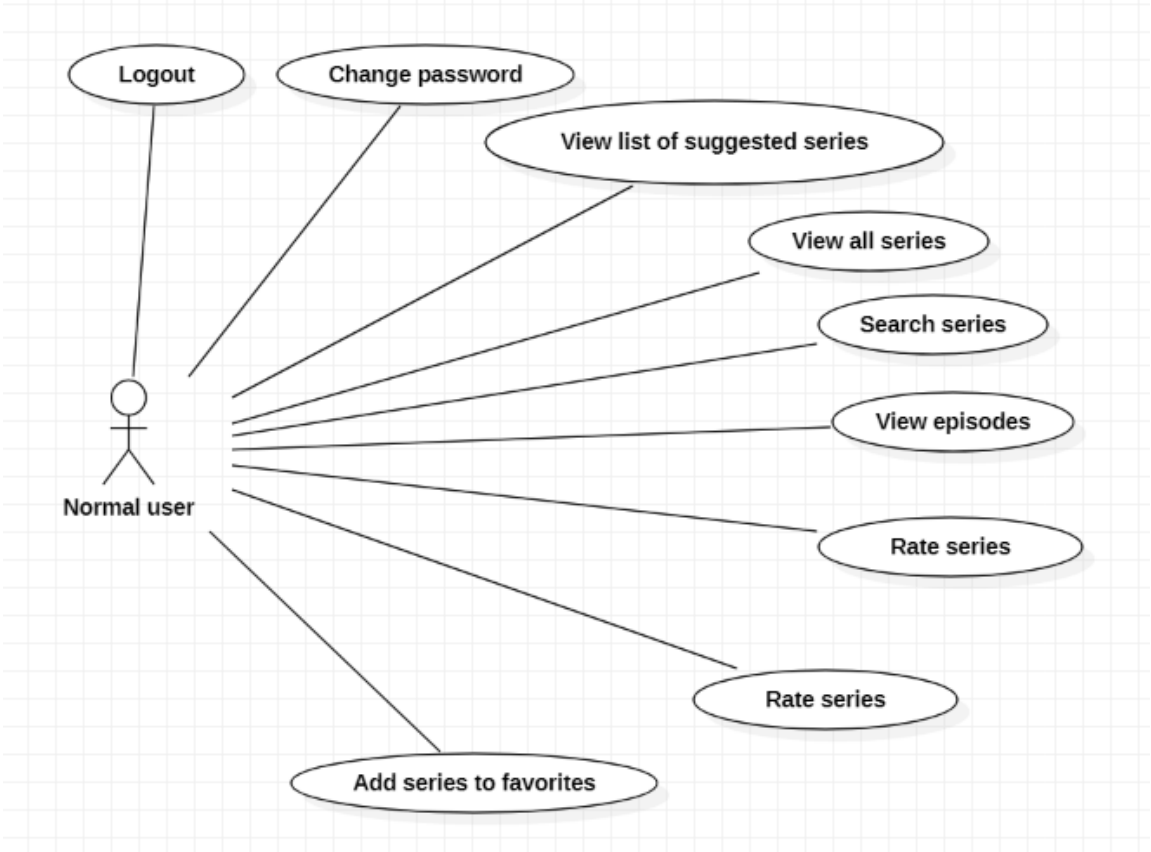
Securitate	1. Date sensibile stocate sub formă criptată(parolele) 2. Restricționarea accesului la anumite pagini și funcționalități în funcție de tipul de utilizator logat.
Utilizabilitate	1. O interfață ușor de utilizat și intuitivă. 2. Posibilitatea de a schimba limba meniului de navigare

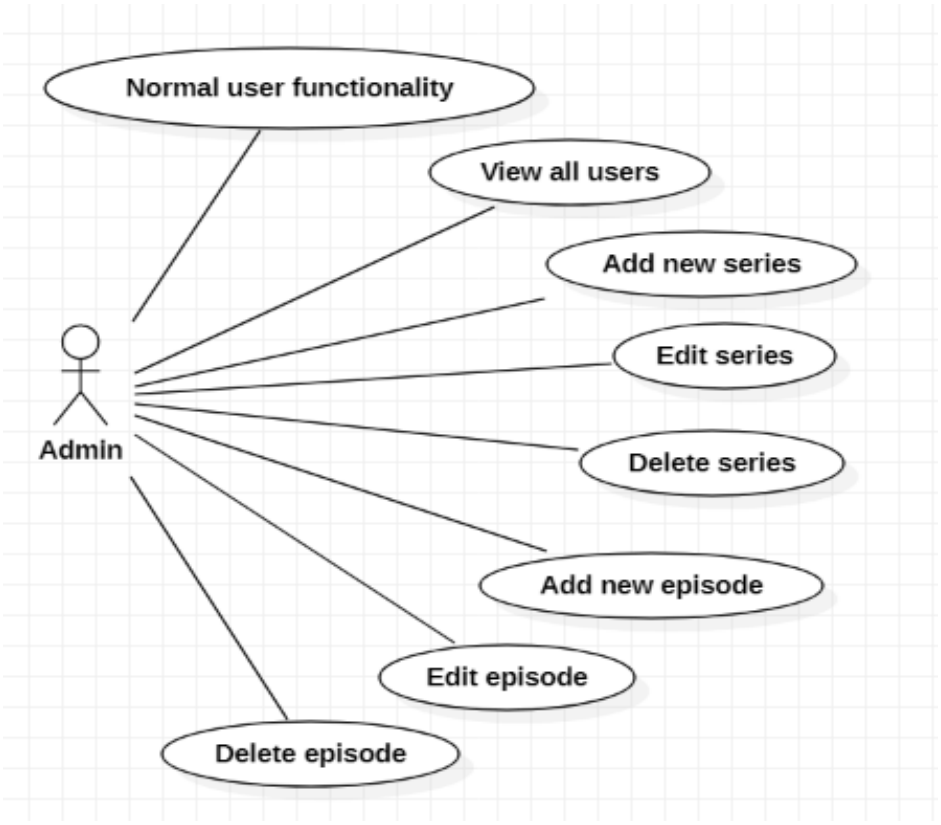
Scalabilitate	1. Această proprietate este realizată prin intermediul serviciilor externe folosite care pot suportă volume de date mari 2. Scalabilitatea este asigurată și prin conceptualizarea modelului local de date astfel încât nu sunt stocate local decât metadate despre datele de dimensiuni mari aflate în cloud.
Testabilitate	1. Structurarea codului astfel încât să faciliteze adăugarea de teste

Tabel 4.2 Categorii de cerințe nonfuncționale

4.6. Cazuri de utilizare

În cadrul acestei aplicații vor exista 3 tipuri de utilizatori pe baza cărora se definesc scenariile de utilizare: utilizator nelogat, utilizator logat obișnuit respectiv utilizator logat de tip admin . Cazurile de utilizare pentru aceste tipuri de utilizatori se pot observa in schemele următoare:





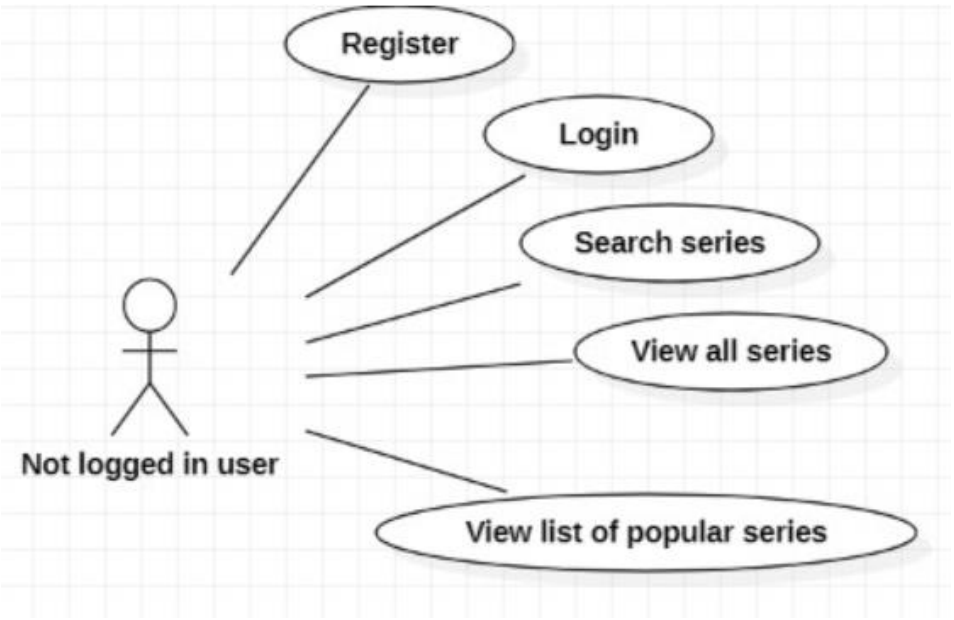


Figura 4.4 Cazuri de utilizare

În continuare urmează o prezentare a cazurilor de utilizare enumerate în aceste diagrame, incluzând pașii necesari unui utilizator pentru a utiliza funcționalitatea descrisă și rezultatele așteptate în urma acțiunilor utilizatorului:

UC 1
Login

Actor: utilizator nelogat
Pași:

- Utilizatorul accesează pagina de Login
- Utilizatorul introduce o adresă de email și o parolă în câmpurile corespunzătoare din formularul de login
- Utilizatorul apasă butonul ”Login”

Precondiții: -

Postcondiții:

- Dacă utilizatorul a introdus un set de credențiale valide, acesta este logat și redirecționat spre pagina de pornire a aplicației
- Dacă utilizatorul a introdus un set de credențiale invalide, un mesaj de eroare corespunzător este afișat

UC 2
Logout

Actor: utilizator logat
Pași:

- Utilizatorul apasă butonul ”logout”

Precondiții:

- Utilizatorul este logat ca și utilizator normal sau admin

Postcondiții:

- Utilizatorul este delogat și redirecționat spre pagina de pornire a aplicației

UC 3
Register

Actor: utilizator nelogat
Pași:

- Utilizatorul accesează pagina de Login
- Utilizatorul completeaza câmpurile din formularul de register
- Utilizatorul apasă butonul ”Register”

Precondiții: -

Postcondiții:

- Dacă nu există un cont asociat adresei de email introduse și datele introduse sunt valide un cont nou este creat și un mesaj de succes este afișat.
- Dacă există un cont asociat adresei de email introduse sau există erori de validare a formularului un mesaj de eroare corespunzător este afișat.

UC 4
Vizualizare serii sugerate

Actor: orice utilizator
Pași:

- Utilizatorul se loghează (dacă nu este logat)
- Utilizatorul navighează la pagina de pornire a aplicației (homepage).

Precondiții: -

Postcondiții:

- Utilizatorul vede o listă o serii sugerate. Acestea se împart în mai multe categorii, în funcție de criteriul pe baza căruia au fost sugerate.

UC 5
Vizualizare listă completă de serii

Actor: orice utilizator
Pași:

- Utilizatorul navighează la pagina ”Toate Seriile”

Precondiții: -

Postcondiții:

- Utilizatorul poate vedea o listă cu toate seriile
- Utilizatorul are opțiunea de a ordona seriile sau de a le filtra după diverse criterii

UC 6
Căutare serii

Actor: orice utilizator
Pași:

- Utilizatorul accesează pagina de pornire a aplicației
- Utilizatorul inserează o valoare pe care dorește să o caute în bara de căutare
- Utilizatorul apasă tasta Enter sau dă click pe butonul de căutare

Precondiții: -

Postcondiții:

- O listă de serii care corespunde termenului căutat este returnată

UC 7
Vizionare serie

Actor: utilizator logat
Pași:

- Utilizatorul accesează pagina de pornire, caută o serie sau accesează pagina ”Toate Seriile”
- Utilizatorul dă click pe numele uneia dintre seriile afișate

Precondiții:

- Utilizatorul este logat ca și utilizator normal sau admin

Postcondiții:

- Utilizatorul este redirecționat la pagina seriei
- Această pagină conține numele seriei, un player video care redă episodul curent,rating-ul seriei și o listă de episoade ale seriei dar și un buton pentru adăugarea seriei la lista de favorite
- Utilizatorul poate da click pe oricare din episoadele din listă pentru a începe redarea acestuia.

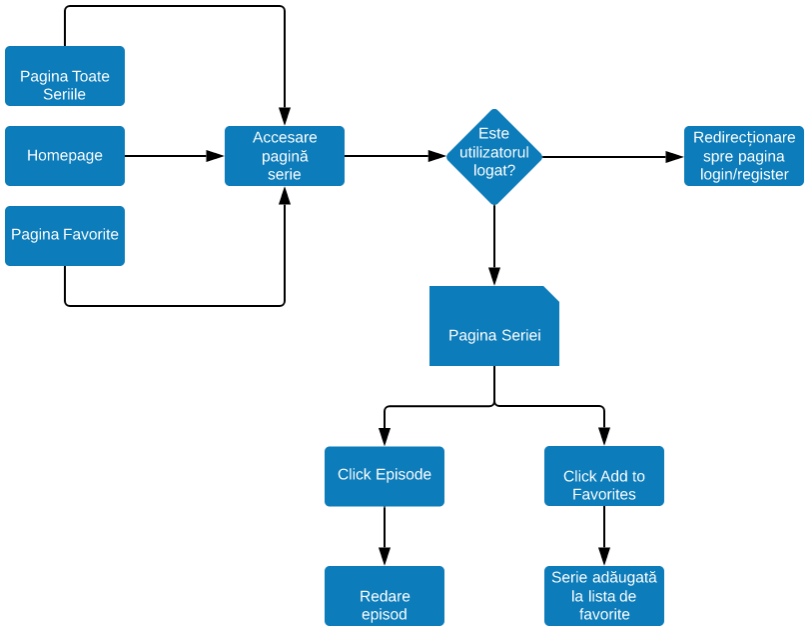


Figura 4.5 Diagramă vizionare serie

UC 8
Adăugare serie la lista de favorite

Actor: utilizator logat
Pași:
- Utilizatorul accesează pagina unei serii existente
- Utilizatorul apasă pe butonul ”Add to favorites”

Precondiții:
- Utilizatorul este logat ca și utilizator normal sau admin

Postcondiții:
- Seria curentă este adăugată la lista de serii favorite al utilizatorului și un mesaj de succes este afișat
- Dacă apare o eroare la adăugarea seriei la lista de favorite un mesaj corespunzător este afișat

UC 9
Vizualizare listă de serii favorite

Actor: utilizator logat
Pași:
- Utilizatorul accesează pagina de serii favorite

Precondiții:
- Utilizatorul este logat ca și utilizator normal sau admin

Postcondiții:
- O listă de serii favorite este afișată
- Pentru fiecare serie din listă există un buton de ștergere a seriei din lista de favorite

UC 10
Ștergere serie din lista de favorite

Actor: utilizator logat
Pași:
- Utilizatorul accesează pagina de serii favorite
- Utilizatorul apasă butonul ”Remove from favorites” pentru una din seriile existente

Precondiții:
- Utilizatorul este logat ca și utilizator normal sau admin

Postcondiții:

- Seria este ștearsă din lista de serii favorite ale utilizatorului și un mesaj de succes este afișat
- Dacă apare o eroare la ștergerea seriei din lista de favorite un mesaj corespunzător este afișat

UC 11
Vizualizare listă de utilizatori existenți

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a conturilor utilizatorilor

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Utilizatorul vede o listă cu toți utilizatorii existenți
- Pentru fiecare utilizator din listă există un buton de ”Edit” și un buton de ”Delete”
- Utilizatorul vede un buton pentru crearea unui nou user.

UC 12
Ștergere utilizator existent

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a conturilor utilizatorilor
- Utilizatorul apasă butonul ”Delete” pentru unul din userii afișați în listă

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Dacă operația de ștergere se efectuează cu succes contul userului ales este șters din baza de date și din lista de useri afișată pe UI.
- Dacă apare o problemă la operația de ștergere utilizatorul va vedea un mesaj corespunzător de eroare

UC 13
Editare utilizator existent

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a conturilor utilizatorilor
- Utilizatorul apasă butonul ”Edit” pentru un user existent

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Utilizatorul este redirecționat la un formular de editare a datelor userului ales
- Utilizatorul poate edita numele, adresa de email, parola sau rolul(administrator sau user normal) al userului selectat
- Utilizatorul poate apăsa butonul ”Save” pentru a salva modificările făcute
- Dacă apar probleme la salvarea modificărilor sau la validarea datelor introduse un mesaj de eroare corespunzător este afișat.

UC 14
Vizualizare listă de serii existente

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Utilizatorul poate vedea o listă cu toate seriile existente
- Pentru fiecare serie afișată există un buton de ”Edit”, un buton de ”Delete” și un buton ”Manage episodes”
- Pe aceeași pagină există un buton de creare a unei serii noi

UC 15
Ștergere serie existentă

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Delete” pentru una din seriile existente

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Dacă ștergerea seriei se realizează cu succes datele corespunzătoare seriei și episoadelor acesteia sunt șterse din baza de date și seria este îndepărtată din lista de serii de pe UI
- Dacă ștergerea seriei eșuează un mesaj de eroare corespunzător este afișat

UC 16
Editare informații ale unei serii existente

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Edit” pentru una din seriile existente

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Utilizatorul este redirecționat spre un formular de editare a seriei selectate. Prin acest formular poate fi editată imaginea, numele, genul și rating-ul corespunzător seriei selectate
- Utilizatorul poate salva modificările făcute seriei. Dacă salvarea se realizează fără erori un mesaj de succes este afișat
- Dacă apar probleme la salvarea informațiilor modificate un mesaj de eroare corespunzător este afișat

UC 17
Vizualizare listă de episoade ale unei serii existente

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Manage episodes” pentru una din seriile existente

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Utilizatorul este redirecționat la o pagină care conține o listă cu toate episoadele seriei
- Numele fiecărui episod din listă este un câmp editabil

- Pentru fiecare episod există un buton de ștergere a episodului respectiv și un buton ”Save” pentru salvarea modificărilor făcute la titlul episodului
- Pe aceeași pagină este afișat un formular de adăugare al unui nou episod la seria selectată

UC 18
Editarea titlului unui episod dintr-o serie existentă

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Manage episodes” pentru una din seriile existente
- Utilizatorul modifică numele unuia din episoadele afișate și apasă butonul ”Save” asociat episodului

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Dacă salvarea modificării titlului episodului se realizează fără erori un mesaj de succes este afișat
- Dacă apar probleme la salvarea modificărilor făcute la titlul episodului un mesaj de eroare corespunzător este afișat

UC 19
Ștergerea unui episod dintr-o serie existentă

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Manage episodes” pentru una din seriile existente
- Utilizatorul apasă butonul ”Delete” pentru unul din episoadele afișate

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Dacă nu apar erori la operația de ștergere informațiile despre episodul selectat sunt șterse din baza de date și episodul este îndepărtat din lista de episoade de pe UI
- Daca apare o eroare la ștergerea episodului un mesaj de eroare corespunzător este afișat

UC 20
Adăugarea unui episod nou la o serie existentă

Actor: utilizator cu drepturi de administrator
Pași:

- Utilizatorul accesează pagina de administrare a seriilor
- Utilizatorul apasă butonul ”Manage episodes” pentru una din seriile existente
- Utilizatorul selectează un fișier video stocat local
- Utilizatorul apasă butonul ”Upload”

Precondiții:

- Utilizatorul este logat ca și administrator

Postcondiții:

- Fișierul stocat în cloud în baza de date Vimeo iar informația primită în urma uploadului(URL-ul video-ului) este stocată apoi în baza de date locală iar lista de episoade afișată este updatată cu noul episod
- Dacă apare o eroare în timpul uploadării unui video un mesaj de eroare corespunzător este afișat

Capitolul 5. Proiectare de Detaliu si Implementare

În cadrul acestui capitol va fi prezentată arhitectura sistemului precum și modul de utilizare al diferitelor tehnologii pentru implementarea diverselor funcționalități ale proiectului.

5.1. Arhitectura sistemului

Sistemul este implementat utilizând o arhitectură de tip client-server. Partea de client constă în aplicația frontend construită folosind Vue JS iar serverul este aplicația backend realizată cu framework-ul spring și limbajul Java. Cele două aplicații comunică prin request-uri HTTP. Astfel , aplicația backend definește endpoint-uri pe care aplicația de frontend le folosește pentru a apela metode implementate pe partea de server.

Pe lângă implementarea logicii aplicației, un alt rol al backend-ului il reprezintă comunicarea cu serviciile externe. În acest caz, sunt folosite două astfel de servicii: Vimeo API, serviciu care se ocupă de stocarea fișierelor video respectiv Recombee API, care se ocupă cu stocarea datelor despre interacțiunile dintre utilizatori și video-uri și generarea de recomandări pe baza acestora.Aceste servicii externe vor fi prezentate în detaliu în subcapitole separate. Arhitectura sistemului poate fi observată în figura următoare:

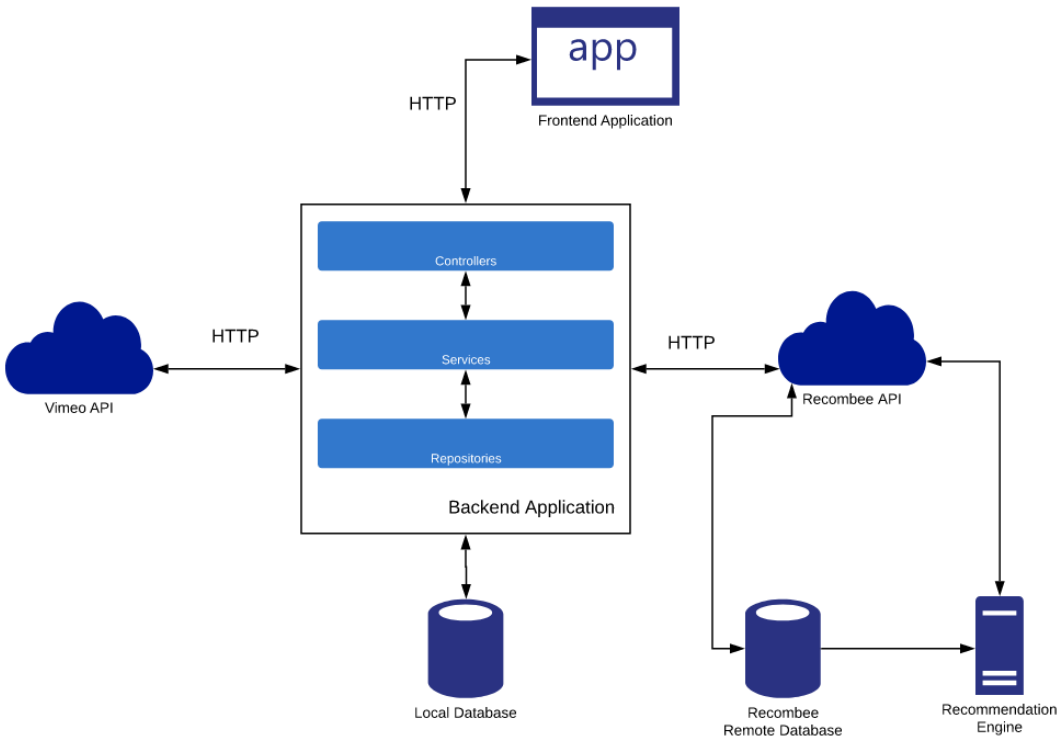


Figura 5.1 Arhitectura detaliată a sistemului

5.2. Tehnologii utilizate

În cadrul acestui subcapitol vor fi prezentate principalele tehnologii folosite în cadrul proiectului. Acesta este împărțit în două aplicații: o aplicație client realizată folosind Vue.js respectiv o aplicație server realizată în limbajul Java folosind framework-ul Spring. Componenta server a proiectului folosește și două servicii externe: Vimeo API folosit pentru gestiunea stocării remote a fișierelor video și Recombee API folosit pentru generarea de recomandări ,utilizarea acestor servicii în cadrul aplicației va fi prezentat în continuare.

5.2.1. Aplicația client

În acest subcapitol va fi prezentată aplicația client care va oferi interfața prin care utilizatorul poate accesa diversele funcționalități ale proiectului. Va fi descrisă în detaliu structura aplicației client, componentele sale și tehnologiile pe folosite. Înainte de implementare am realizat o serie de mockup-uri pentru a modela interfața de utilizator folosind tool-ul online https://moqups.com. (Anexa 1).

5.2.1.1. Arhitectura aplicației client

Aplicația client este structurată după cum urmează în schema următoare:

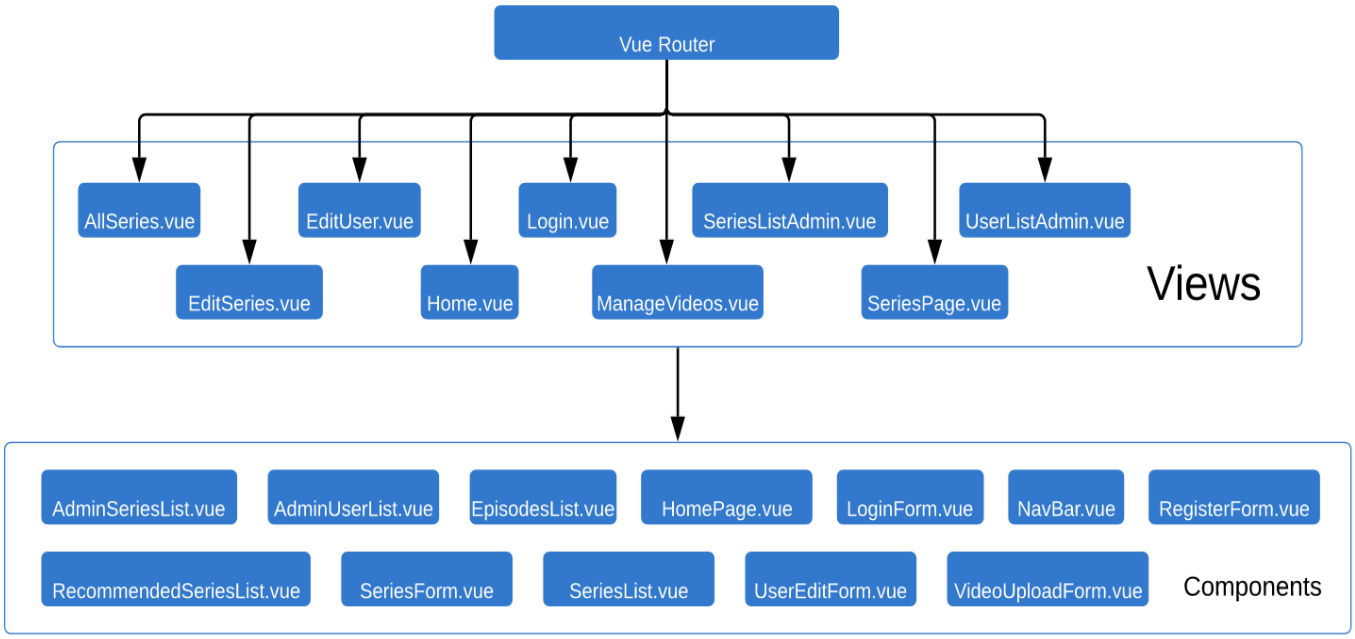


Figura 5.2 Arhitectura aplicației client

După cum se poate observa în această schemă , aplicația este realizată din două mari tipuri de componente și anume Views și Components. Views sunt paginile pe care le poate vedea utilizatorul, fiecare din ele conține una sau mai multe Components(am ales să nu reprezint relațiile dintre fiecare View și Component în acestă schemă pentru simplificarea sa). Corespondența dintre URL-ul vizitat de utilizator și View-ul sau pagina care se afișează este realizată de către Vue Router. Atât Vue Router, Views și Components vor fi descrise în detaliu în subcapitolele următoare.

5.2.1.2. Rutarea

Componenta care se ocupă cu definirea path-urilor și cu rutarea este Vue Router. Acesta conține o listă de path-uri și un View asociat fiecărui path după cum se poate observa în schema următoare:

```
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/series/:id',
    name: "Series",
    component: SeriesPage
  },
  {
    path: '/editSeries/:id',
    name: "EditSeries",
    component: EditSeries
  },
```

Figura 5.3 Definirea rutelor în Vue.js

Aplicațiile Vue.js sunt single-page-applications(SPA), ceea ce înseamnă că pagina curentă este rescrisă mereu cu informațiile noi care vin de la server. Astfel , în momentul în care utilizatorul realizează anumite acțiuni noi date sunt aduse de la server în mod dinamic. Această încărcare dinamică a datelor este realizată de către router.

Vue Router oferă posibilitatea de a defini segmente dinamice în cadrul unui path. Un astfel de segment se definește folosind simbolul : (ex Series path în figura 5.2). Astfel , pentru diferite valori ale segmentului dinamic Vue Router va face matching la View-ul corespunzător. Aceste segmente dinamice pot fi accesat în cadrul View-urilor sau al Componentelor folosind sintaxa **this.$route.params.id** unde ”id” este numele parametrului în acest exemplu.

5.2.1.3. Views și Components

Views și Components sunt similare, motiv pentru care vor fi prezentate în cadrul aceluiași subcapitol. Acestea sunt principalele unități care formează o aplicație Vue.js. Atât Views cât și Components implementează modelul Model-View-ViewModel, structura căruia este prezentată în **4.2.2**. Acestea sunt fișiere împărțite în 3 părți distincte cu roluri bine definite și anume:

- Template
- Script
- Style

Astfel , în secțiunea de Template se află partea de HTML al Componentei sau View-ului, adică elementele de UI implementate.Vom lua ca exemplu componenta LoginForm.vue care descrie formularul de login, parte a pagini de login din cadrul aplicației:

```
<template>
<div>
    <form ref="form">
        <p class="h4 text-center mb-4">Sign in</p>
        <div class="white-text">
            <mdb-input class="inp" label="Your email" icon="envelope" type="email" v-model="form.email"/>
            <mdb-input class="inp" label="Your password" icon="lock" type="password" v-model="form.password"/>
        </div>
        <div class="text-center">
            <mdb-btn v-on:click="loginFunc">Login</mdb-btn>
        </div>
  </form>
</div>
</template>
```

Figura 5.4 Secțiunea template al unei componente în Vue.js

Secțiunea de Script utilizează Javascript pentru descrie comportamentul , sau cum reacționează partea de UI la diferite evenimente(vezi figura 5.5, exemplu de metodă care se apelează în momentul trimiterii formularului de login). Tot în cadrul acestei secțiuni se importă librării sau alte componente externe(vezi figura 5.4) și se realizează comunicarea cu aplicația server (prin apelarea de API-ur, în acest caz folosind librăria Axios care va fi descrisă în detaliu într-un capitol viitor) dar și cu servicii externe.

```
<script>
import { mdbInput, mdbBtn } from 'mdbvue';
import { required, email} from 'vuelidate/lib/validators'
import axios from 'axios';

 export default {
   components: {
     mdbInput,
     mdbBtn
   },
```

Figura 5.5 Importarea și declararea de componente și librării

```
data(){
  return{
    form: {
      email:'',
      password: ''
    }
  }
},
validations:{
  form:{
    email:{required, email},
    password:{required}
  }
},
methods:{
  loginFunc: function(){
      this.$refs.form.submit()
      let currentObj = this;
      this.$v.form.$touch();
      if(this.$v.form.email.$error) {
        alert('Email is not valid')
        return
      }
      if(this.$v.form.password.$error) {
        alert('The password is required')
        return
      }
      axios.post('http://localhost:8080/user/login/',{
        email: this.form.email,
        password: this.form.password
      })
      .then(function(response){
            console.log('this is the login response ' + response.data);
            if(response.data == false){
              alert("Wrong login credentials")
            }
            currentObj.output = response.data;
      })
      .catch(function(error){
            currentObj.output = error;
      })
```

Figura 5.6 Metodă în cadrul secțiunii script

În secțiunea style se folosește CSS pentru a descrie aspectul elementelor de UI.

```
<style>
  .h4{
    color: aliceblue;
  }
  .inp{
    color: aliceblue;
  }
</style>
```

Figura 5.7 Secțiunea style a unei componente

Principala diferență dintre Views și Components este faptul că un Component descrie o singură componentă de UI(care poate fi compusă din unul sau mai multe elemente de UI) de sine stătătoare care poate fi reutilizată pe mai multe pagini pe când Views descriu pagini întregi, folosind una sau mai multe Componente.Astfel, în Views sunt importate elemente de tip Component (figura 5.7) care pot fi incluse în secțiunea template ca și elemente de UI cu tag-uri care au numele componentei.

```
<script>
import LoginForm from '@/components/LoginForm.vue'
import RegisterForm from '@/components/RegisterForm.vue'
export default {
    components:{
        LoginForm,
        RegisterForm
    }
}
</script>
```

Figura 5.8 Declararea de Components într-un View

5.2.1.4. Firebase API

Firebase Storage este serviciul pe care l-am utilizat pentru a stoca în cloud imaginile corespunzătoare fiecărei serii. Accesul la acest serviciu se realizează prin intermediul unui API. Apelarea acestuia este facilitată de către pachetul *npm firebase* care oferă metode wrapper pentru apelarea API-ului. Firebase Storage poate stoca orice fel de fișiere, dar în cazul acestei aplicații inputul este restricționat doar la imagini.

Procesul de încărcare al unei imagini în Firebase Storage este următorul: imaginea care se dorește încărcată este citită din stocarea locală a utilizatorului. Odată citită, aceasta este trimisă spre Firebase Storage prin intermediul API-ului, care , în caz de succes va returna URL-ul prin care se poate accesa imaginea stocată. Acest URL urmează să fie trimis spre aplicația server pentru a fi stocat în baza de date locală.

Modul de încărcare a unei imagini în Firebase Storage poate fi observat în figura 5.5 , unde *imageData* este imaginea citită prin intermediul unui element HTML de tip input care acceptă doar imagini.

```
const storageRef=firebase.storage().ref(`${this.imageData.name}`).put(this.imageData);
storageRef.on(`state_changed`,snapshot=>{
    this.uploadValue = (snapshot.bytesTransferred/snapshot.totalBytes)*100;
}, error=>{console.log(error.message)},
()=>{this.uploadValue=100;
    storageRef.snapshot.ref.getDownloadURL().then((url)=>{
    this.picture =url;
    });
```

Figura 5.9 Încărcarea unei imagini în Firebase Storage

5.2.1.5. Vue i18n

Vue i18n este un plugin folosit pentru internaționalizarea, sau cu alte cuvinte traducerea aplicațiilor vue în mai multe limbi pentru a le face mai ușor de utilizat de către un număr mai mare de utilizatori.

În cadrul aplicației client al proiectului propus am ales să folosesc meniuri atât în limba română cât și în limba engleză, utilizatorul având posibilitatea de a alege între aceste două limbi.

Pluginul Vue i18n face posibilă această funcționalitate prin următorul proces: va exista câte un fișier separat pentru fiecare limbă dorită, fiecare fișier conținând cuvintele sau frazele pe care le dorim traduse sub forma de variabile , fiecare având ca valoare cuvintele în limba corespunzătoare fișierului(vezi figura 5.9, fișierele ro.js și en.js care conțin termeni folosiți în meniurile de navigare în română și în engleză).

```
export default {
    home: 'Home',
    language: 'Language',
    allSeries: 'All Series',
    account: 'Account',
    login: 'Login',
    logout: 'Logout',
};
```

```
export default {
    home: 'Acasă',
    language: 'Limbă',
    allSeries: 'Toate Seriile',
    account: 'Cont',
    login: 'Intră în cont',
    logout: 'Ieși din cont',
};
```

Figura 5.10 Fișierele ro.js și en.js

Odată create aceste fișiere, este nevoie doar de crearea unui obiect de tip VueI18n care va conține setări precum limba default și locația fișierelor menționate anterior(vezi figura 5.10).

```
import Vue from 'vue';
import VueI18n from 'vue-i18n';
import messages from '../translations';

Vue.use(VueI18n);
export const i18n = new VueI18n({
  locale: 'ro',
  fallbackLocale: 'ro',
  messages,
});
```

Figura 5.11 Declararea obiectului i18n

După definirea acestui obiect el este importat în componentele unde dorim sa îl folosim, și prin intermediul său se pot folosi variabilele declarate în fișierele pentru diferitele limbi , comutarea între limbi(și deci între fișierele din care se obțin valorile variabilelor) facându-se pe baza parametrului **locale** transmis acestui obiect.

5.2.1.6. Vue Axios

Axios este un client HTTP pentru browsere prin intermediul căruia se pot face request-uri HTTP, se pot transforma datele de intrare și ieșire ale unui request și care suportă Promise API. Un Promise este o valoare proxy a cărui valoare nu este cunoscută în momentul în care este creeat și poate fi asociat unei acțiuni asincrone. Acesta poate avea una din 3 stări:

- *Pending*: aceasta este starea inițială
- *Fulfilled*: se ajunge în această stare dacă operația asociată s-a executat cu succes
- *Rejected*: se ajunge în această stare dacă operația asociată a rezultat într-o eroare

Vue Axios este un wrapper pentru clientul Axios. Acesta este folosit în cadrul aplicației client pentru a realiza comunicarea cu aplicația client prin apelarea endpoint-urilor definite de către aceasta.

Folosind Axios se poate realiza și interceptarea erorilor. Astfel, dacă API-ul apelat nu este disponibil, request-ul nu a fost făcut în mod corespunzător sau răspunsul primit nu este de forma așteptată putem intercepta eroarea folosind **catch** și o putem gestiona în mod corespunzător, fie prin afișarea sa în consolă fie prin afișarea unui mesaj de eroare la nivelul interfeței grafice.

În continuare va fi prezentat un exemplu de request de tip post făcut spre aplicația server. În corpul request-ului se transmite un obiect de forma așteptată de către server și se folosește **catch** pentru interceptarea posibilelor erori care vor fi apoi afișate în consolă:

```
axios.post('http://localhost:8080/user/register/',{
  username: this.form.username,
  password: this.form.password,
  email: this.form.email
})
.then(function(response){
        var respData = response.data
        alert(respData);
        currentObj.output = response.data;
  })
.catch(function(error){
        console.log(error);
        currentObj.output = error;
  })
```

Figura 5.12 Apelarea unui API folosind Vue Axios

5.2.2. Aplicația server

În cadrul acestui capitol va fi prezentată structura aplicației server, componentele care o alcătuiesc și tehnologiile utilizate în cadrul său.

5.2.2.1. Descriere generală

Aplicația Server este construită folosind limbajul de programare Java și framework-ul spring.

După cum a fost ilustrat și în capitolul **4.2.1** această aplicație respectă o arhitectură layered, fiecare dintre layerele care o compun având un rol bine definit.(Figura 5.12).

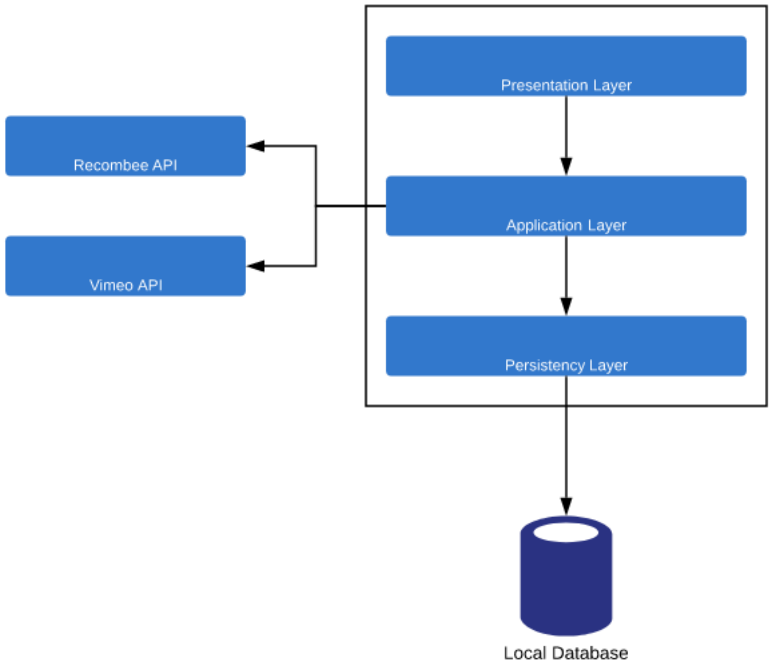


Figura 5.13 Arhitectura layered aplicată pe sistem

Această organizare face ca aplicația să fie ușor de înteles și face foarte ușoară adăugarea de noi funcționalități sau modificarea celor existente. Așadar , layerele și responsabilitățile lor sunt după cum urmează:

- **Layer-ul de Prezentare(Presentation layer)**:acest layer are scopul de a permite accesarea funcționalităților implementate în cadrul acestei aplicații de către utilizatori sau alte aplicații. Acest lucru este realizat prin intermediul unor API-uri care expun diferitele metode implementate de către servicii. API-urile sunt apelate de către aplicația client, realizându-se astfel comunicarea între cele două aplicații. Layer-ul de prezentare este

alcătuit din *Controllere,* acestea fiind componentele în cadrul cărora sunt definite API-urile menționate.

- **Layer-ul de Aplicație**(**Application Layer** sau **Business Logic Layer**): acest layer conține logica aplicației, cu alte cuvinte în cadrul acestui layer sunt implementate majoritatea funcționalităților aplicației. Tot în cadrul acestuia se realizează comunicarea cu serviciile externe. Layer-ul de aplicație este format din componente numite *Servicii* care sunt direct apelate de către Controllerele din Layer-ul de Prezentare
- **Layer-ul de Persistență(Persistence Layer) :** acest layer conține *Entități* care descriu modelele de date și *Repositories* care se ocupă cu accesul direct la baza de date, permițând executarea de operații CRUD. Aceste Repositories sunt utilizate de către serviciile prezente în Layer-ul de Aplicație pentru comunicarea cu baza de date.

Este important de reținut că în această arhitectură, deși datele circulă în ambele sensuri între layere dependințele dintre ele, și ca urmare fluxul de control circulă într-un singur sens. Deci Layer-ul de Prezentare depinde de Layer-ul de Aplicație și accesează metode din cadrul acestuia , dar nu și invers. Aceeași relație există și între Layer-ul de Aplicație și cel de persistență.

Toate componentele menționate care alcătuiesc diferitele layere ale aplicației vor fi prezentate în detaliu în capitolele următoare.

5.2.2.2. Controller

Controller-ele sunt componente aflate în Layer-ul de Prezentare al aplicației. Acestea sunt clase care definesc endpoint-uri API care vor fi apelate de către aplicația client. Framework-ul Spring oferă o serie de adnotări care se pot folosi pentru a defini clasa controller dar și diferitele endpoint-uri din cadrul Controller-ului. Câteva dintre adnotările pe care le-am folosit în cadrul proiectului sunt următoarele:

- *@RestController* : această adnotare include adnotările *@Controller* și *@ResponseBody* astfel încât cea din urmă nu mai trebuie adăugată la fiecare metodă din controller. Adnotarea *@RestController* este folosită pentru a defini o clasă ca fiind Controller, primind astfel rolul de a gestiona request-uri web.
- *@RequestMapping*: este o adnotare care definește calea(path-ul) pe care controller-ul va accepta request-uri.Această adnotare acceptă parametrul path prin care se specifică calea. Acest parametru nu are o valoare default, deci daca acesta nu este specificat, controller-ul va accepta orice request-uri.
- *@CrossOrigin*: Este o adnotare prin intermediul căreia se activează CORS(Cross-Origin Resource Sharing), sau cu alte cuvinte se permite altor aplicații cu origini diferite să facă request-uri spre controller-ul definit.

- *@Autowired:* Folosind această adnotare se realizează Dependency Injection, permițând framework-ului Spring sa injecteze automat beans în clasa curentă.
- *@GetMapping*, *@PostMapping*: Aceste adnotări permit definirea de request-uri de tip GET și POST. Ele sunt forme simplificate ale adnotării *@RequestMapping* cu valoarea argumentului method = RequestMethod.GET respectiv RequestMethod.POST. Cele două adnotări au parametrul value prin care se definește path-ul spre request-ul definit. Acestea acceptă path-uri dinamice prin introducerea zonelor dinamice între {}
- *@RequestBody*: prin această adnotare body-ul request-ului este mapat la un obiect.
- *@PathVariable*: prin acestă adnotare se mapează o parte dinamică a path-ului la un parametru. Astfel request-ul va accepta doar valori de tipul acestui parametru în secțiunea dinamică a path-ului.

În următoarea figură va fi prezentată utilizarea unora dintre adnotările descrise în cadrul SeriesController:

```
@RestController
@CrossOrigin
@RequestMapping(value = "/series")
public class SeriesController {

    private final SeriesService seriesService;

    @Autowired
    public SeriesController(SeriesService seriesService) { this.seriesService = seriesService; }

    @GetMapping(path = "/{id}")
    public SeriesDTO getById (@PathVariable("id") int seriesId){
        return seriesService.getById(seriesId);}

    @PostMapping(path = "/addOrEdit")
    public Boolean addOrEdit(@RequestBody SeriesDTO seriesDTO) { return seriesService.addOrEditSeries(seriesDTO); }
```

Figura 5.14 Definirea SeriesController

Toate controller-ele din Layer-ul de prezentare și request-urile permise de acestea sunt definite în mod similar.

5.2.2.3. Service

Componentele de tip service aparțin Layer-ului de Aplicație și au rolul de a implementa logica aplicației. Ele folosesc componente de tip Repository pentru a accesa baza de date. Pentru injectarea acestora se folosește adnotarea *@Autowired* prezentată

precedent într-o manieră similară cu folosirea ei la nivel de Controller pentru a injecta Serviciile.

O clasă este marcată ca fiind Serviciu folosindu-se adnotarea *@Service*. În figura 5.14 este prezentat un exemplu de definire a unui Serviciu și a Repository-urilor folosite în cadrul acestuia:

```
@Service
public class SeriesService {
    private final SeriesRepository seriesRepository;
    private final VideoRepository videoRepository;

    @Autowired
    public SeriesService(SeriesRepository seriesRepository, VideoRepository videoRepository) {
        this.seriesRepository = seriesRepository;
        this.videoRepository = videoRepository;
    }
```

Figura 5.15 Definirea clasei SeriesService

5.2.2.4. Repository

Componentele de tip Repository sunt interfețe care se află în Layer-ul de Persistență și au rolul de a comunica cu baza de date, permițând realizarea de operații CRUD.

Pentru definirea unui Repository în Spring se folosește interfața JpaRepository care este extinsă de interfața dorită(Repository-ul pe care îl creăm). Un exemplu de astfel de repository se poate observa în figura următoare:

```
public interface VideoRepository extends JpaRepository<Video, Integer> {

}
```

Figura 5.16 Definirea interfeței VideoRepository

JpaRepository primește două argumente: T și ID. T reprezintă entitatea care descrie modelul de date . În acest exemplu , entitatea este clasa Video. Entitățile vor fi discutate în detaliu într-un capitol următor. Al doilea argument, ID, reprezintă tipul de date al cheii primare din tabela descrisă de entitatea T, în acest caz cheia primară fiind de tip Integer.

5.2.2.5. Entități

Entitățile sunt modelele de date în framework-ul Spring. Așadar , clasele marcate ca entități descriu structura unei tabele corespunzătoare în baza de date. Spring oferă o serie de adnotări pe care le-am utilizat pentru definirea de entități și câmpuri corespunzătoare lor:

- *@Entity*: această adnotare marchează o clasă ca entitate, cu alte cuvinte clasa respectivă poate fi mapată la o tabelă din baza de date.
- *@Table:* prin această adnotare și a argumentului ”name” pe care îl primește se poate seta numele pe care îl va avea tabela din baza de date la care este mapată clasa.
- *@Id*: prin această adnotare un câmp este marcat ca și Id sau cheie primară. Adnotarea @GeneratedValue poate fi folosită împreună cu aceasta pentru a seta acest Id ca auto-increment.
- *@Colum:* folosind această adnotare și argumentele pe care le primește se pot specifica proprietățile câmpului mapat precum numele, lungimea, dacă acceptă valori null, și altele.
- *@OneToMany , @ManyToOne, @OneToOne și @ManyToMany:* folosind aceste adnotări se pot seta relații între un câmp din clasa curentă și câmpuri din alte entități specificate prin adnotarea *@JoinColumn*.

În continuare va fi prezentat un exemplu de declarare al unei entități folosind câteva dintre adnotările descrise:

```
@Entity
@Table(name = "watchHistory")
public class WatchHistory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="watchHistoryId", unique = true,nullable = false)
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "user")
    private User user;

    @ManyToOne
    @JoinColumn(name = "video")
    private Video video;
```

Figura 5.17 Definirea entității WatchHistory

5.2.2.6. Data Transfer Objects

Data Transfer Objects (DTO) sunt obicete prin intermediul cărora se face transferul de date la nivelul Layer-ului de Prezentare. Astfel, datele primite și returnate de către aplicație sunt sub formă de DTO-uri în locul Entităților. Deoarece se vrea evitarea transmiterii de date inutile acestea sunt folosite în locul Entităților la anumite nivele ale aplicației. Un DTO poate avea aceleași câmpuri pe care le are o Entitate(fără adnotările specifice Entităților), sau doar o parte din câmpurile din Entitate, în fucție de nevoile de implementare.

```
public class VideoDTO {

    private Integer idVideo;

    private String name;

    private String url;
```

```
@Entity
@Table(name = "video")
public class Video {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "idVideo", unique = true, nullable = false)
    private Integer idVideo;

    @Column(name = "name")
    private String name;

    @Column(name = "url")
    private String url;
```

Figura 5.18 Comparație VideoDTO și Video

DTO-urile sunt folosite în Layer-ul de Prezentare, deci Controller-ele primesc și returnează doar obicete de tip DTO. La nivelul de Repositories sunt folosite doar Entități, deci locul unde se face conversia între Entități și DTO-uri este Layer-ul de Aplicație. Astfel, pentru fiecare pereche DTO-Entitate există metode de conversie între cele două. În ambele sensuri. Un exemplu de metode care fac conversia DTO-Entitate și Entitate-DTO pentru clasele Video și VideoDTO exemplificate anterior este următorul:

```
public class VideoBuilder {
    public static VideoDTO getDtoFromEntity(Video video){
        return new VideoDTO(video.getIdVideo(),video.getName(),video.getUrl());
    }

    public static Video getEntityFromDto(VideoDTO videoDTO){
        return new Video(videoDTO.getIdVideo(),videoDTO.getName(),videoDTO.getUrl());
    }
```

Figura 5.19 Conversia între Entități și DTO-uri

5.2.2.7. Vimeo API

Vimeo API nu oferă suport nativ pentru Java, așa că, pentru utilizarea sa am folosit pachetul Vimeo Clickntap prin intermediul căruia se poate accesa API-ul și toate metodele sale.

Primul pas pentru utilizarea Vimeo API este crearea unui cont Vimeo și a unei noi aplicații din interfața Vimeo Developer [6]. Odată creată aplicația este nevoie de generarea unui token de acces care să ofere permisiuni de upload.

După ce avem token-ul, utilizarea Vimeo API este foarte ușoară. Token-ul va fi folosit ca argument pentru inițializarea unui obiect de tip Vimeo. Prin intermediul acestui obiect se pot face request-uri spre API prin apelarea de metode din această clasă. Pentru upload-ul de fișiere video , este apelată metoda *addVideo()* care primește ca argument calea spre locația fișierului de pe stocarea locală a utilizatorului. Dacă upload-ul se realizează cu succes, metoda returnează informații despre video-ul încărcat(printre care și URL-ul video-ului care va fi salvat în baza de date locală).Dacă upload-ul eșuează, metoda returnează o eroare.

```
public String uploadVideo(String path){
    Vimeo vimeo = new Vimeo( token: "377aed61aedc364d76f90b94c4582023");
    try {
        String video = vimeo.addVideo(new File(path),  upgradeTo1080: true);
        VimeoResponse info = vimeo.getVideoInfo(video);
        return info.toString();
    } catch (IOException | VimeoException e) {
        e.printStackTrace();
        return "";
    }
}
```

Figura 5.20 Upload video folosind Vimeo API

5.2.2.8. Recombee API

Pentru accesarea motorului de recomandări Recombee este utilizat Recombee API. Recombee oferă suport nativ pentru Java prin pachetul Recombee Api-Client care acționează ca un wrapper peste API .

Pentru a începe utilizarea Recombee API, este nevoie de crearea unui cont. Apoi, folosind interfața Recombee Admin[7] trebuie creeată o bază de date care va stoca obiectele și interacțiunile dintre acestea și generat un token pentru a putea accesa această bază de date din cadrul aplicației server.

După parcurgerea acestor pași de setup, utilizarea Recombee API este foarte asemănătoare cu cea a Vimeo API. Se inițializează un obiect de tip Recombee Client folosind token-ul de acces generat și numele bazei de date create la pasul anterior folosind constructorul *new RecombeeClient(dbName, accessToken).*

[6] https://developer.vimeo.com/
[7] https://admin.recombee.com/

Pentru a genera recomandări este nevoie de a urma un anumit proces. În primul rând, este nevoie ca toate obiectele care participă în interacțiuni(în acest caz utilizatori și serii) trebuie sa fie stocate în baza de date remote. Astfel vom avea metode care adaugă un obiect în baza de date Recombee folosind Recombee API astfel:

```
public void addUser(int userId){
        try {
            client.send(new AddUser(String.valueOf(userId)));
        } catch (ApiException e) {
            e.printStackTrace();
        }
}
```

Figura 5.21 Adăugarea unui obiect în baza de date Recombee

Astfel de metode de adăugare de useri și serii se apeleaza de fiecare dată când un user sau o serie nouă este creată. Următorul pas este înregistrarea interacțiunilor dintre obiecte. Pentru a înregistra o interacțiune între două obiecte, ambele trebuie să existe în baza de date Recombee, de aceea pasul anterior este foarte important. Un exemplu de interacțiune este atunci când un utilizator vizionează o serie, aceasta se înregistrează astfel:

```
public void addUserWatchesSeries(int userId, int seriesId){
    try {
        client.send(new AddDetailView(String.valueOf(userId),String.valueOf(seriesId))
        .setTimestamp(new Date()));
    } catch (ApiException e) {
        e.printStackTrace();
    }
}
```

Figura 5.22 Salvarea unei interacțiuni între utilizator și serie

Există multe tipuri de interacțiuni care pot fi înregistrate folosind Recombee API, acestea fiind folosite de motorul de recomandare pentru generarea recomandărilor. Odată ce înregistrăm interacțiunile între obiecte, putem obține recomandări prin intermediul API-ului. Cu cât sunt înregistrate mai multe interacțiuni, cu atât calitatea recomandărilor crește.

```
public RecommendationResponse getRecommendedSeriesForUser(int userId){
    try {
        RecommendationResponse recommended = client.send(new RecommendItemsToUser(String.valueOf(userId),  count: 5));
        String[] id = recommended.getIds();
        return recommended;
    } catch (ApiException e) {
        e.printStackTrace();
    }
    return new RecommendationResponse();
}
```

Figura 5.23 Generarea de recomandări pentru un utilizator

5.2.3. Baza de date

Baza de date folosită este una MySQL, toate tabelele și relațiile dintre acestea fiind generate din Entități folosind adnotările descrise în capitolul *5.2.2.5*. Structura datelor reprezentate este inspirată de bazele de date ale aplicațiilor similare descrise în [7], cu modificări pentru adaptarea la nevoile aplicației dezvoltate. Diagrama bazei de date este următoarea:

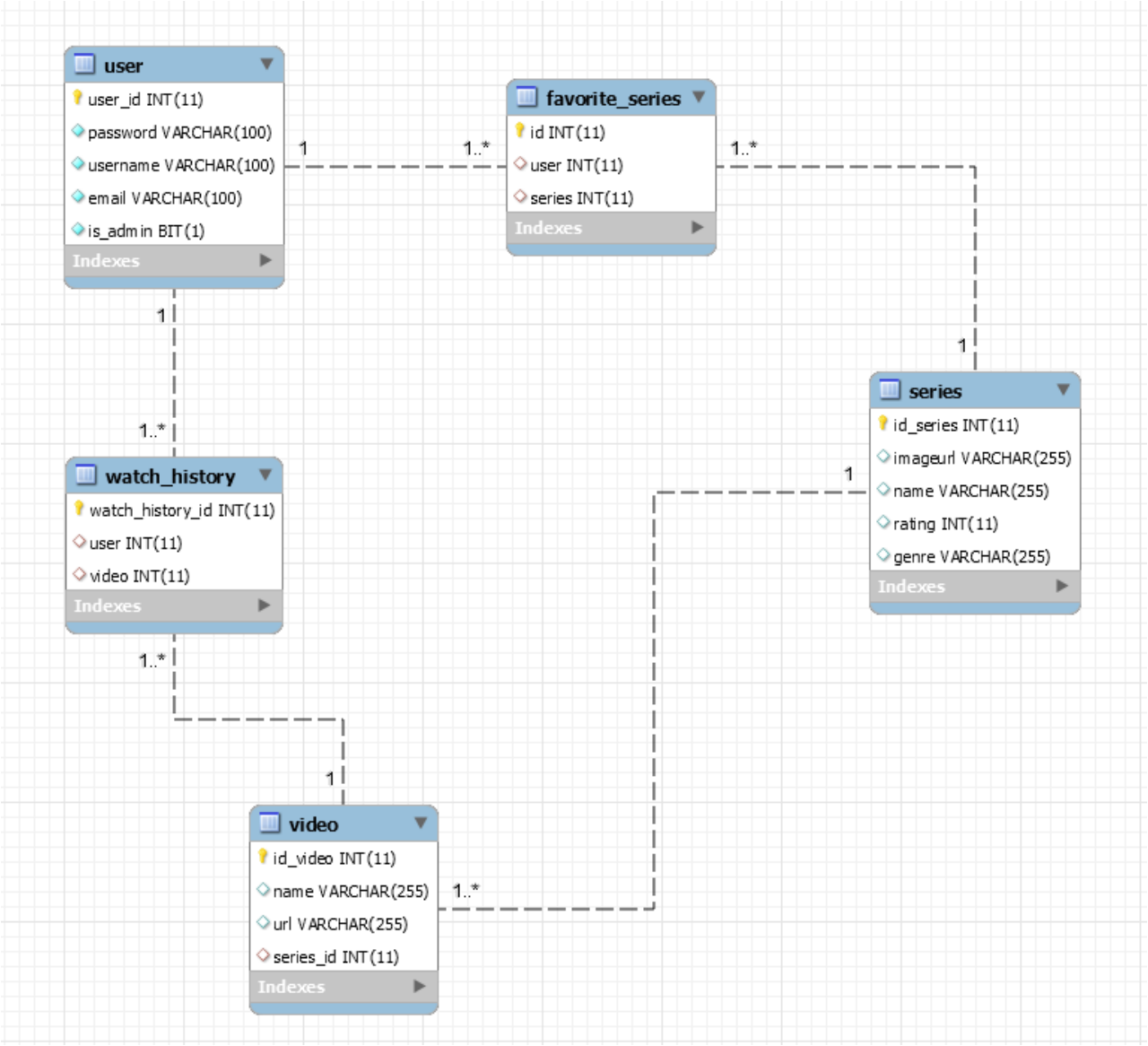


Figura 5.24 Diagrama bazei de date

Baza de date respectă nivelul 4 de normalizare(4NF). În continuare vor si prezentate tabelele care o alcătuiesc și rolul câmpurilor acestora:

Tabela user: conține datele despre conturile utilizatorilor existenți. Aceasta este formată din următoarele câmpuri:

- *User_id*: cheia primară a tabelei, de tip int
- *Password*: parola criptată a contului utilizatorului, stocată sub formă de string
- *Username*: string care conține numele de utilizator al contului
- *Email*: string, stochează adresa de email asociată contului utilizatorului
- *Is_admin*: boolean, un flag care arată dacă utilizatorul are sau nu drepturi de administrator

Tabela series: conține datele despre serii. Aceasta este formată din următoarele câmpuri:

- *Id_series*: cheia primară a tabelei, de tip int
- *Imageurl*: string, URL-ul imaginii asociate seriei
- *Name*: string care conține numele seriei
- *Rating*: int care conține rating-ul seriei, sau mai bine zis categoria din care face parte o serie din punct de vedere al calității.Conform [8] seriile se pot clasifica în 4 categorii diferite (Terrible, Poor, Average,Excellent).Pentru această aplicație am considerat o categorie suplimentară, și anume Good care se situează între Average și Excellent. Acest câmp va conține valori cuprinse între 1(reprezentând Terrible) și 5(Excellent).
- *Genre*: string, numele genului seriei

Tabela videos: conține datele despre fișierele video. Aceasta este formată din următoarele câmpuri:

- *Id_video*: cheia primară a tabelei, de tip int
- *Name*: string care conține numele local al fișierului video(este numele care va fi folosit în cadrul aplicației, nu trebuie să corespundă cu numele fișierului din Vimeo)
- *Url*: string care conține url-ul video-ului din Vimeo
- *Series_id*: int , cheie străină reprezentând seria căreia aparține video-ul

Tabela favorite_series: conține datele despre seriile favorite ale utilizatorilor. Aceasta este formată din următoarele câmpuri:

- *Id*: cheia primară a tabelei, de tip int
- *User*: int, cheie străină care reprezintă utilizatorul
- *Series*: int, cheie străină care reprezintă seria

Tabela watch_history: conține datele despre istoricul de vizualizare a fiecărui utilizator. Aceasta este formată din următoarele câmpuri:

- *Id*: cheia primară a tabelei, de tip int
- *User*: int, cheie străină care reprezintă utilizatorul
- *Series*: int, cheie străină care reprezintă seria

Capitolul 6. Testare şi Validare

În cadrul acestui capitol va fi prezentat modul în care s-a realizat testarea aplicației, sau modul în care aceasta îndeplinește cerințele funcționale și nonfuncționale stabilite.

Din punctul de vedere al testării la nivel de componente, acestea au fost testate individual în timpul implementării. Astfel , pentru componentele aplicației server, fiecare API a fost testat folosind aplicația Postman pentru a observa răspunsul la diferite date de input respectiv timpii de răspuns.

Pentru testarea componentelor de frontend, a interacțiunii cu API-urile serverului dar și cu alte componente frontend, am realizat o serie de teste de sistem manuale care recreează anumite flow-uri comune pentru utilizatorii aplicației. În continuare vor fi prezentate câteva scenarii de test folosite penru testarea acestor flow-uri:

Test Case 1: Login
Scenariul 1
Pași:

- Utilizatorul accesează aplicația
- Utilizatorul acceseaza pagina de Login/Register
- Utilizatorul completează formularul de login cu un set de credențiale valide
- Utilizatorul apasă butonul de Login

Rezultate așteptate:

- Un mesaj de succes este afișat și utilizatorul este logat cu succes în contul său.

Scenariul 2
Pași:

- Utilizatorul accesează aplicația
- Utilizatorul acceseaza pagina de Login/Register
- Utilizatorul completează formularul de login cu un set de credențiale invalide
- Utilizatorul apasă butonul de Login

Rezultate așteptate:

- Un mesaj de eroare este afișat și utilizatorul nu este logat în contul său.

Test Case 2: Vizionare serie
Scenariul 1
Pași:

- Utilizatorul accesează aplicația
- Utilizatorul acceseaza pagina unei serii

Rezultate așteptate:

- Utilizatorul este redirecționat spre altă pagină deoarece nu este logat

Scenariul 2
Pași:

- Utilizatorul accesează aplicația
- Utilizatorul se loghează cu credențiale valide

- Utilizatorul accesează pagina unei serii

Rezultate așteptate:

- O listă cu episoadele seriei este afișată
- Uilizatorul poate apăsa pe oricare dintre episoade pentru a începe redarea acestuia.

Test Case 3: Edit Series info

Scenariul 1

Pași:

- Utilizatorul accesează aplicația
- Utilizatorul acceseaza pagina de administrare a seriilor

Rezultate așteptate:

- Utilizatorul este redirecționat la pagina Login/Register deoarece nu este logat

Scenariul 2

Pași:

- Utilizatorul accesează aplicația
- Utilizatorul se logheaza cu un cont fară drepturi de administrator
- Utilizatorul acceseaza pagina de administrare a seriilor

Rezultate așteptate:

- Utilizatorul este redirecționat la pagina Login/Register deoarece nu are drepturile necesare pentru a accesa pagina

Scenariul 3

Pași:

- Utilizatorul accesează aplicația
- Utilizatorul se logheaza cu un cont cu drepturi de administrator
- Utilizatorul acceseaza pagina de administrare a seriilor
- Utilizatorul apasă butonul de editare a informațiilor unei serii
- Utilizatorul completează formularul de editare cu date valide și apasă butonul update series

Rezultate așteptate:

- Un mesaj de succes este afișat și informațiile modificate de către utilizator sunt salvate în baza de date

Capitolul 7. Manual de Instalare si Utilizare

În acest capitol vor fi prezentate resursele necesare pentru instalarea și rularea proiectului, instrucțiuni pentru realizarea acestor pași dar și instrucțiuni de navigare și utilizare a aplicației.

7.1. Cerințe de sistem

Pentru rularea în condiții optime a proiectului sistemul trebuie să îndeplinească următoarele cerințe:

- CPU : CPU x86 64bit
- RAM: 4GB
- Sistem de operare: Windows, Linux sau OS X
- Browser web: Google Chrome, Mozilla Firefox, Microsoft Edge , Safari , alte browsere bazate pe browser-ul open source Chromium(Opera, Brave, etc)

7.2. Instalare și rulare

7.2.1. Descărcarea proiectului

Proiectul este împărțit în două aplicații separate, unul de frontend și unul de backend. Cele două sunt stocate în două repository-uri separate pe platforma GitLab.

Primul pas este obținerea de copii locale a celor două aplicații. Pentru asta trebuie executați pașii următori:

- Instalarea GIT
- Accesarea GitLab și crearea unui cont
- Cererea accesului la cele două repository-uri
- Clonarea lor în directoare locale folosind comanda ”*git clone <repo-url>*”

7.2.2. Crearea bazei de date

Pentru proiectul propus va fi utilizată o bază de date MySQL. Pentru asta trebuie urmați acești pași:

- Instalarea MySQL folosind MySQL Installer
- Pornirea serverului MySQL
- Crearea unei conexiuni și a unui user în MySQL Workbench
- Crearea unei noi scheme cu numele ”licenta”

7.2.3. Setup-ul proiectului de backend

Pentru rularea proiectului de backend trebuie urmați pașii următori:

- Instalarea Java SE Development Kit 8
- Instalarea unui IDE care suportă Java și Spring(ex: Intellij IDEA, Eclipse, etc)
- Instalarea Apache Maven

- Deschiderea în IDE a proiectului clonat la 7.2.1
- Setarea conexiunii cu baza de date în fișierul *application.properties* (se vor completa portul, userul și parola bazei de date locale)

```
database.ip = ${MYSQL_IP:localhost}
database.port = ${MYSQL_PORT:3306}
database.person = ${MYSQL_USER:root}
database.password = ${MYSQL_PASSWORD:root}
database.name = ${MYSQL_DBNAME:licenta}
```

Figura 7.1 Configurarea DB în application.properties

- Setarea portului pe care va rula aplicația(aceasta trebuie sa ruleze pe portul 8080) și a hibernate ddl auto(această proprietate trebuie setată cu valoarea ”create” la prima rulare a aplicației și cu valoarea ”update” la rulări ulterioare). Aceste proprietăți se setează în fișierul *application.properties* menționat la punctul anterior

```
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

server.port=8080
```

Figura 7.2 Configurarea portului și a hibernate ddl-auto

- Rularea proiectului(folosind butonul run din IDE)

7.2.4. Setup-ul proiectului de frontend

Pentru rularea aplicației frontend trebuie urmați pașii următori:

- Instalarea Node.js folosind Node.js Installer
- Instalarea Vue.js folosind comanda **npm install -g @vue/cli**
- Deschiderea proiectului clonat la 7.2.1 folosind un IDE care suportă aplicații web(Intellij IDEA, Visual Studio Code, WebStorm, etc)
- Deschiderea unui terminal și navigarea la locația din interiorul proiectului unde se află fișierul **package.json**
- Rularea comenzii **npm install**
- Rularea aplicației din terminalul IDE-ului folosind comanda **npm run serve**

7.3. Instrucțiuni de utilizare

În acest capitol va fi prezentat modul de utilizare al aplicației, harta paginilor și scenarii de utilizare pe diferitele pagini ale aplicației.

7.3.1. Harta paginilor

În acest capitol va fi prezentată structura paginilor și cum se poate face navigarea între ele dar și tipurile de utilizatori care pot accesa fiecare pagină. Harta paginilor va fi prezentată în figura 7.3:

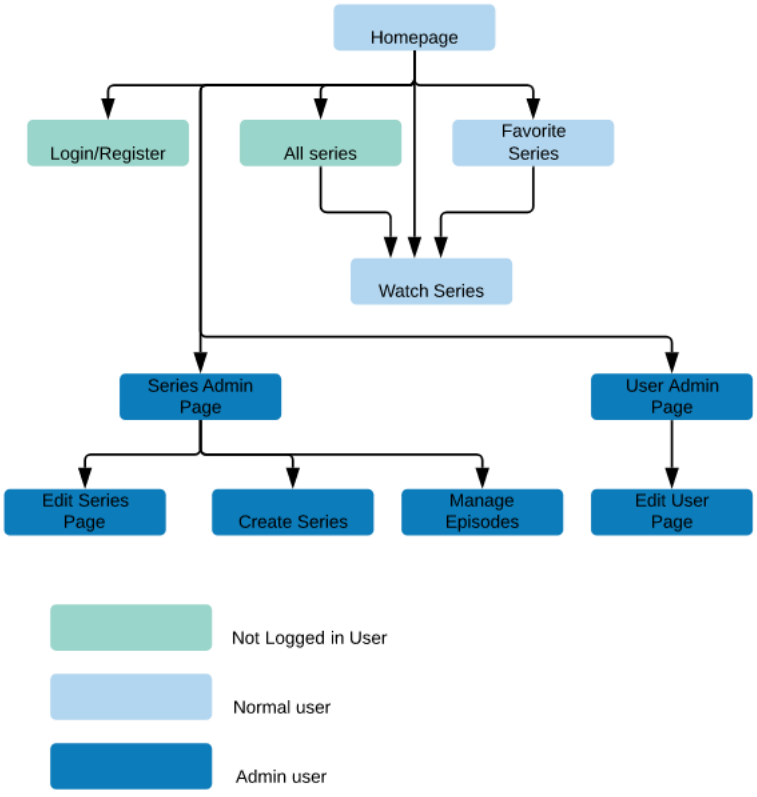


Figura 7.3 Harta paginilor

7.3.2. Utilizarea aplicației

În acest capitol vor fi prezentate diferitele pagini ale aplicației și funcționalitățile oferite de către acestea.

Homepage

Pe această pagină utilizatorul poate vedea o listă de serii recomandate pe baza acțiunilor sale trecute. Daca utilizatorul nu este logat acesta va fi redirecționat la pagina de login/register.

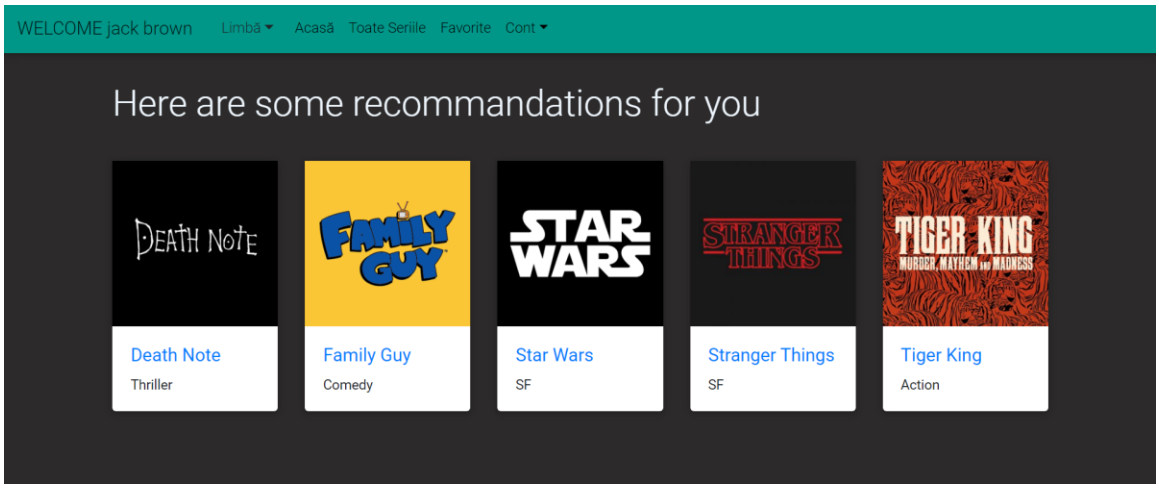


Figura 7.4 Homepage

Login/Register

Pe această pagină sunt afișate două formulare: un formular de login pentru logarea într-un cont existent respectiv un formular de register pentru crearea unui nou cont de utilizator.

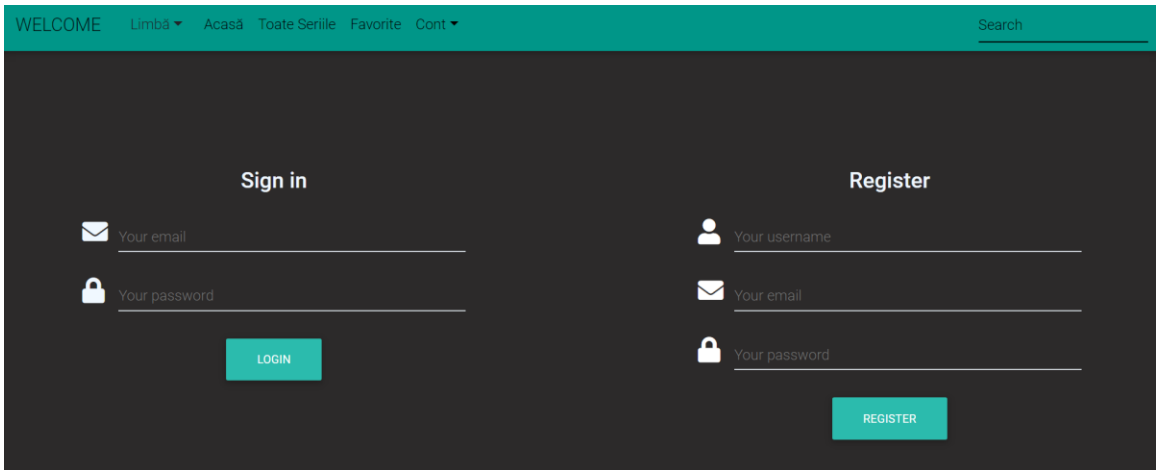


Figura 7.5 Login/Register

Toate Seriile

Pe această pagină utilizatorul poate vedea o listă cu toate seriile existente. Acesta are opțiunea de a filtra seriile în funcție de genul acestora sau de a le sorta după nume sau rating atât ascendent cât și descendent.În plus, utilizatorul va putea efectua o căutare în funcție de numele seriei prin completarea câmpului corespunzător și apăsarea butonului Search. Pentru a reseta rezultatele căutării utilizatorul trebuie sa șteargă input-ul introdus în câmpul de căutare și apăsarea din nou a butonului de Search. Această pagină poate fi accesată de orice utilizatori, chiar și cei nelogați.

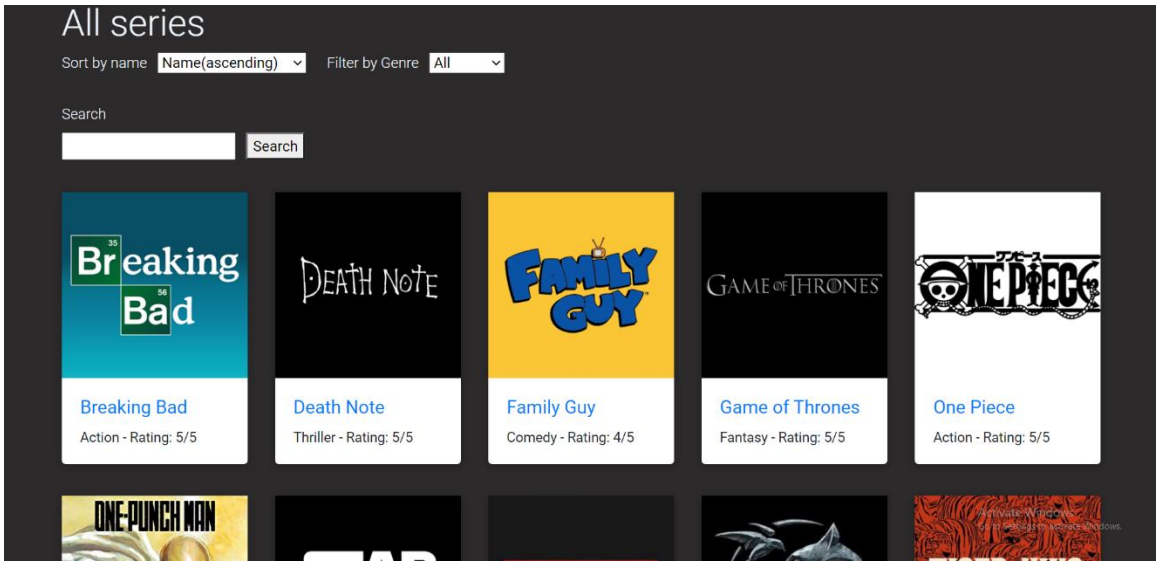


Figura 7.6 Toate seriile

Pagina seriei

Utilizatorul poate ajunge pe această pagină apasând pe una din seriile de pe paginile care returnează serii precum Homepage, Toate Seriile, sau lista de serii favorite. Pe această pagină uilizatorul poate să vadă o listă cu toate episoadele seriei. Apăsând pe unul dintre episoade, se începe redarea acestuia. În plus, pe această pagină utilizatorul poate apăsa butonul ”Add to favories” pentru a adăuga seria la lista de favorite. Această pagină poate fi accesată doar de utilizatorii logați.

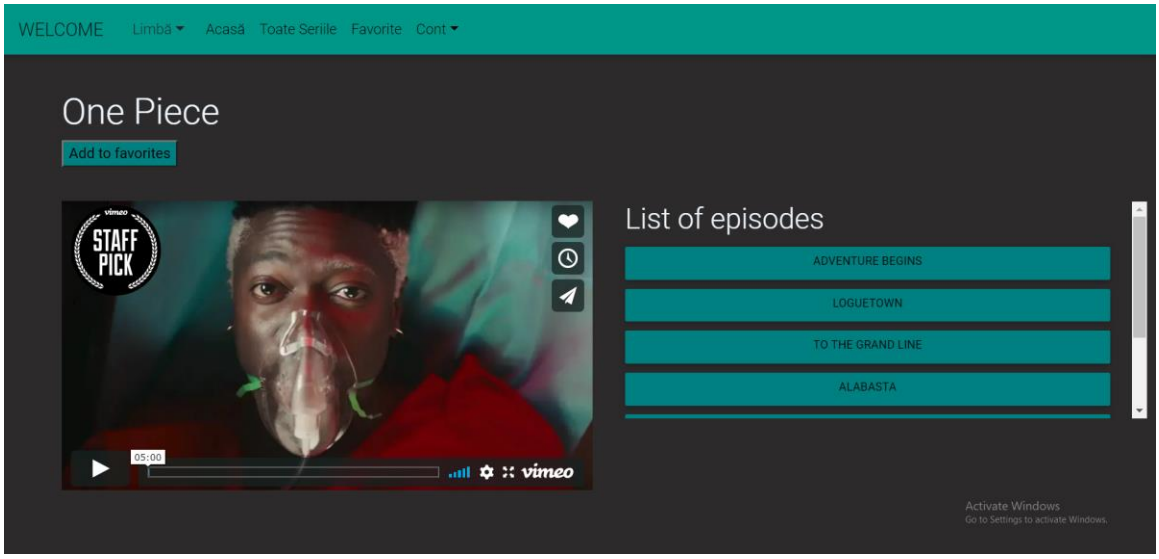


Figura 7.7 Pagina seriei

Lista de favorite

Pe această pagină utilizatorul poate vedea toate seriile pe care le-a marcat ca favorite și are opțiunea de a elimina serii din lista de favorite. Această pagină poate fi accesată doar de utilizatorii.

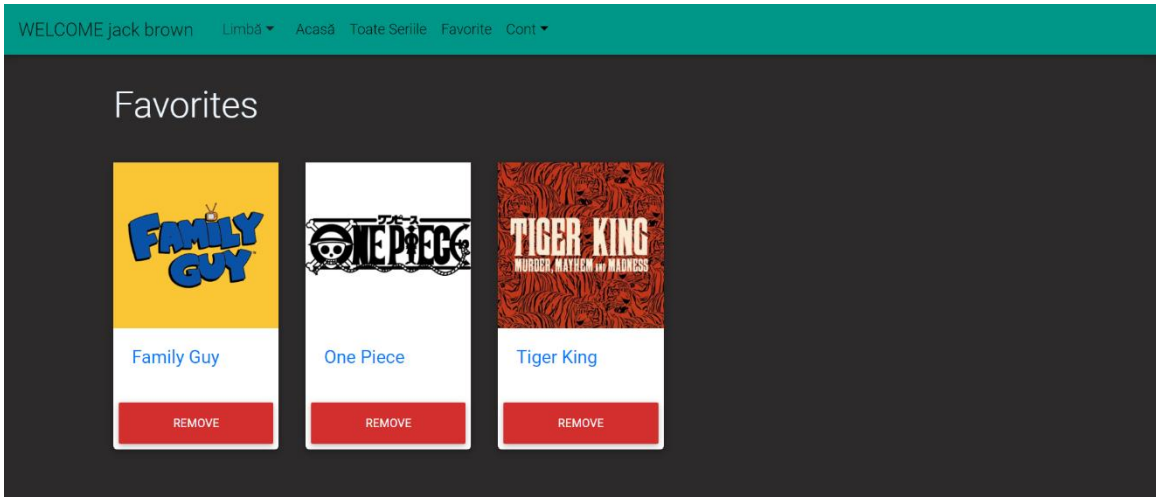


Figura 7.8 Lista de favorite

Administrare serii

Această pagină poate fi accesată doar de către utilizatori cu permisiuni de administrator. Acesta va putea vedea o listă cu toate seriile existente. Pentru fiecare serie, utilizatorul are opțiunea de a edita informațiile acesteia, de a gestiona lista de episoade aferente seriei sau de a șterge seria. Pe aceeasi pagină, există un buton pentru adăugarea unei serii noi.

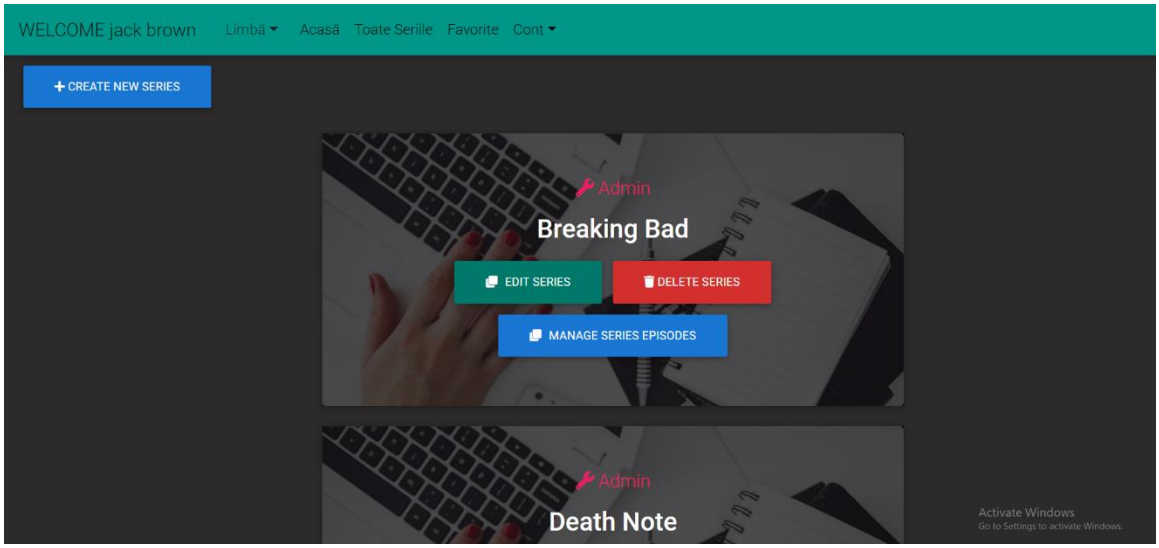


Figura 7.9 Administrare serii

Editare serie

Această pagină poate fi accesată doar de către utilizatori cu permisiuni de administrator. Acesta va putea edita detalii precum numele, genul sau imaginea corespunzătoare seriei.

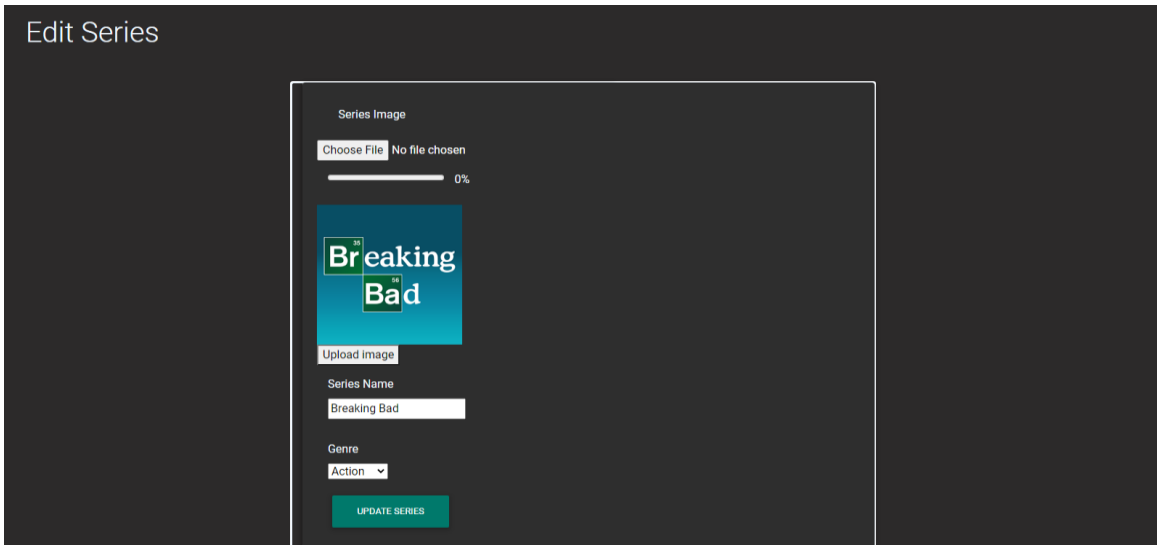


Figura 7.10 Editare serie

Administrare episoade ale seriei

Această pagină poate fi accesată doar de către utilizatori cu permisiuni de administrator. Acesta va putea vedea un formular de creare a unei serii noi. Pentru crearea seriei utilizatorul are opțiunea de a uploada un video nou dar și de a seta numele noului episod și URL-ul spre video(daca utilizatorul a uploadat un video URL-ul va fi generat automat. Pentru crearea unui episod folosind un video deja existent în Vimeo utilizatorul va putea introduce manual URL-ul.

Pe lângă formularul de creare de episoade, pe această pagină sunt listate episoadele existente ale seriei. Pentru fiecare episod existent poate fi editat numele, URL-ul video-ului,sau se poate șterge episodul .

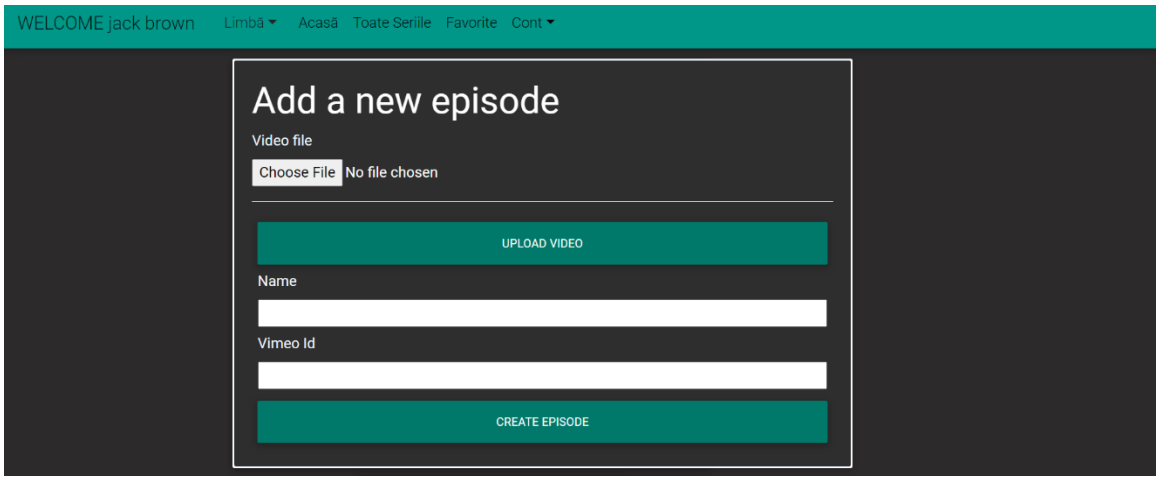


Figura 7.11 Creare episod

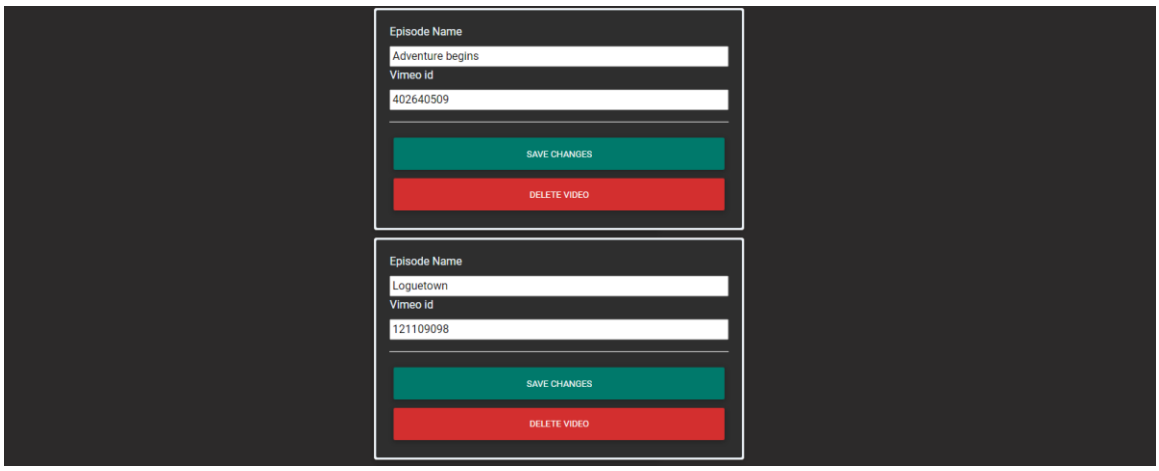


Figura 7.12 Editare episoade

Administrare utilizatori

Această pagină poate fi accesată doar de către utilizatori cu permisiuni de administrator. Acesta va putea vedea o listă cu toți utilizatorii existenți. Pentru fiecare utilizator din listă există opțiunile de a edita informațiile acestuia respectiv de a îl șterge.

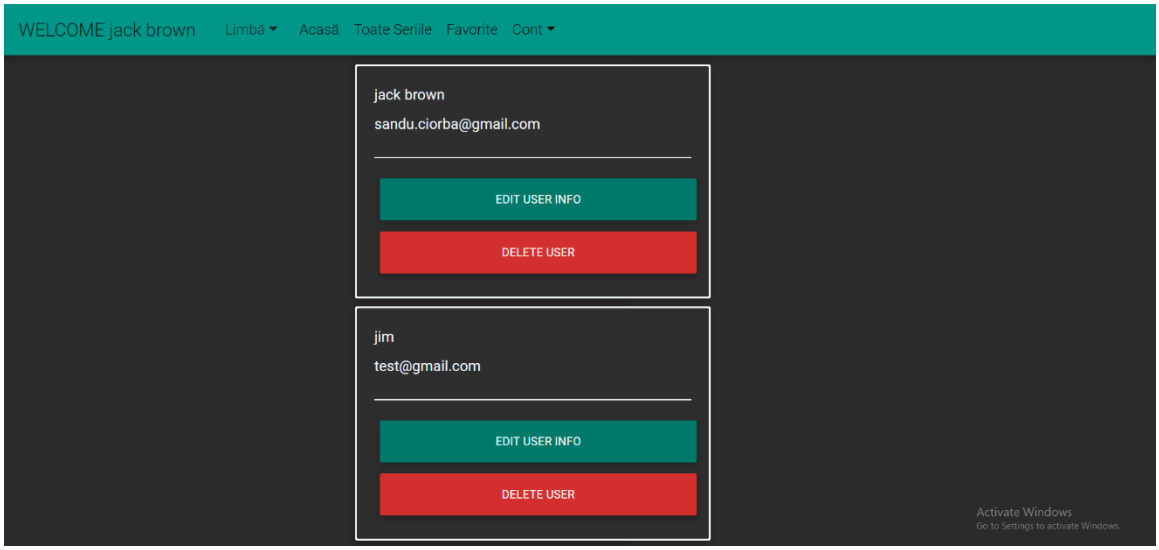


Figura 7.13 Gestionare utilizatori

Editare utilizatori

Această pagină poate fi accesată doar de către utilizatori cu permisiuni de administrator. Acesta va putea vedea un formular pentru editarea informațiilor corespunzătoare contului unui utilizator precum nume, adresa de email, parola, sau rolul.

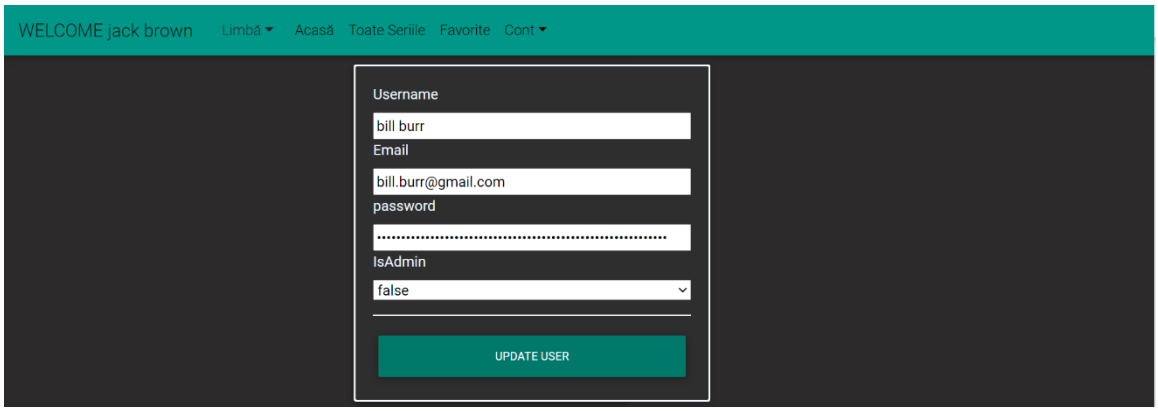


Figura 7.14 Editare utilizator

Capitolul 8. Concluzii

În cadrul acestui capitol vor fi prezentate concluzile obținute în urma implementării sistemului, gradul de îndeplinire al obiectivelor propuse dar și posibile modalități de îmbunătățire a sistemului în viitor.

8.1. Analiza rezultatelor obținute

Aplicația creată reușește să îndeplinească obiectivele principale propuse și anume de a fi o aplicație de video streaming care poate fi utilizată atât de utilizatori normali, dar oferă și administratorilor un mod de a controla conținutul disponibil utilizatorilor normali.

Așadar, utilizatorii normali pot vedea seriile existente în aplicație, primesc recomandări de serii care i-ar putea interesa și pot să vizioneze episoadele fiecărei serii. Utilizatorii de tip administrator au control total asupra conținutului aplicației. Așadar ei pot vedea și edita utilizatorii existenți și pot gestiona seriile . Astfel un administrator are opțiunea de a crea serii noi, dar și de a șterge sau a edita seriile existente. La editarea unei serii, un administrator poate modifica informațiile despre aceasta, dar și sa încarce noi video-uri pentru aceasta sau să editeze informațiile despre vide-urile existente.

De asemenea, prin folosirea serviciilor externe aplicația dispune de un sistem de recomandări(Recombee) robust, cu un număr foarte mare de funcționalități care se poate adapta la modificări ulterioare, dar și de servicii de stocare în cloud (Vimeo respectiv Firebase) care oferă fiabilitate și posibilitate de scalare, fiind astfel la rândul lor adaptabile în cazul modificărilor apărute în cadrul sistemului.

8.2. Dezvoltări ulterioare

O direcție importantă de dezvoltare pentru aplicația propusă este reprezentată de sistemul de recomandări, și anume la funcționalitățile utilizate din cadrul acestuia. În prezent, singurele tipuri de recomandări utilizate sunt colaborative, sau bazate pe interacțiunile utilizator-serie. Cu toate acestea, Recombee suportă și recomandări bazate pe conținut, ceea ce ar presupune cunoașterea anumitor informații despre utilizatori. Pentru a utiliza aceste tipuri de recomandări, ar fi nevoie de implementarea de profiluri pentru utilizatori, pentru ca fiecare utilizator să poată să seteze anumite preferințe.

O altă posibilitate de dezvoltare o reprezintă adăugarea de noi moduri în care utilizatorii pot interacționa cu seriile, precum posibilitatea de a adăuga comentarii sau de a oferi un rating.

O direcție de dezvoltare este reprezentată și de scalarea aplicației. Astfel, dacă se dorește creșterea volumului de date precum numărul și dimensiunea fișierelor video nu este nevoie de modificări în cadrul aplicației, fiind necesară doar tranziția la planuri premium cu mai puține limitări pentru Vimeo respectiv Recombee. În momentul de față sunt folosite planuri basic, gratuite, care limitează conținutul video care poate fi încărcat în Vimeo respectiv numărul de obiecte și interacțiuni care pot fi stocate în baza de date remote Recombee. Un upgrade la aceste planuri ar duce la creșterea acestor limite.

Bibliografie

[1] Eli Brosh , "An Efficient Video-On-Demand System ", Lucrare Licență, Columbia University, 2008, Available online at: https://pdfs.semanticscholar.org/bb43/4eb539525b8d936761357a2ab3c397588d39.pdf

[2] Jerome Mendes de Figueiredo, "CASHED: Cloud-Assisted Adaptive and Scalable Video Streaming for Heterogeneous End-User Devices" , Lucrare Master, Universitatea Tehnică Lisabona, 2014, Available online at: https://fenix.tecnico.ulisboa.pt/downloadFile/395146457574/Dissertation.pdf

[3] Albert Llongueras Clotet , "Implementation of a streaming server", Lucrare Master, Universitatea Maribor, 2011, Available online at: https://upcommons.upc.edu/bitstream/handle/2099.1/13020/Final_Master_Thesis_Albert.pdf?sequence=1&isAllowed=y

[4] Mahmoud K. Darwich, "Cost-Efficient Video On Demand (VOD) Streaming Using Cloud Services" , Lucrare Doctorat, University of Louisiana at Lafayette, 2017, Available online at: http://hpcclab.org/theses/mahmoud17.pdf

[5] Niklas WIETRECK, "Towards a new generation of movie recommender systems: A mood based approach", Lucrare Master, Upsala University, 2019, Available online at: http://www.diva-portal.org/smash/get/diva2:1219240/FULLTEXT01.pdf

[6] Leidy Esperanza Molina , "Recommendation System for Netflix", Lucrare de Cercetare, Universitaetea Vrije Amsterdam ,2018 , Available online at: https://beta.vu.nl/nl/Images/werkstuk-fernandez_tcm235-874624.pdf

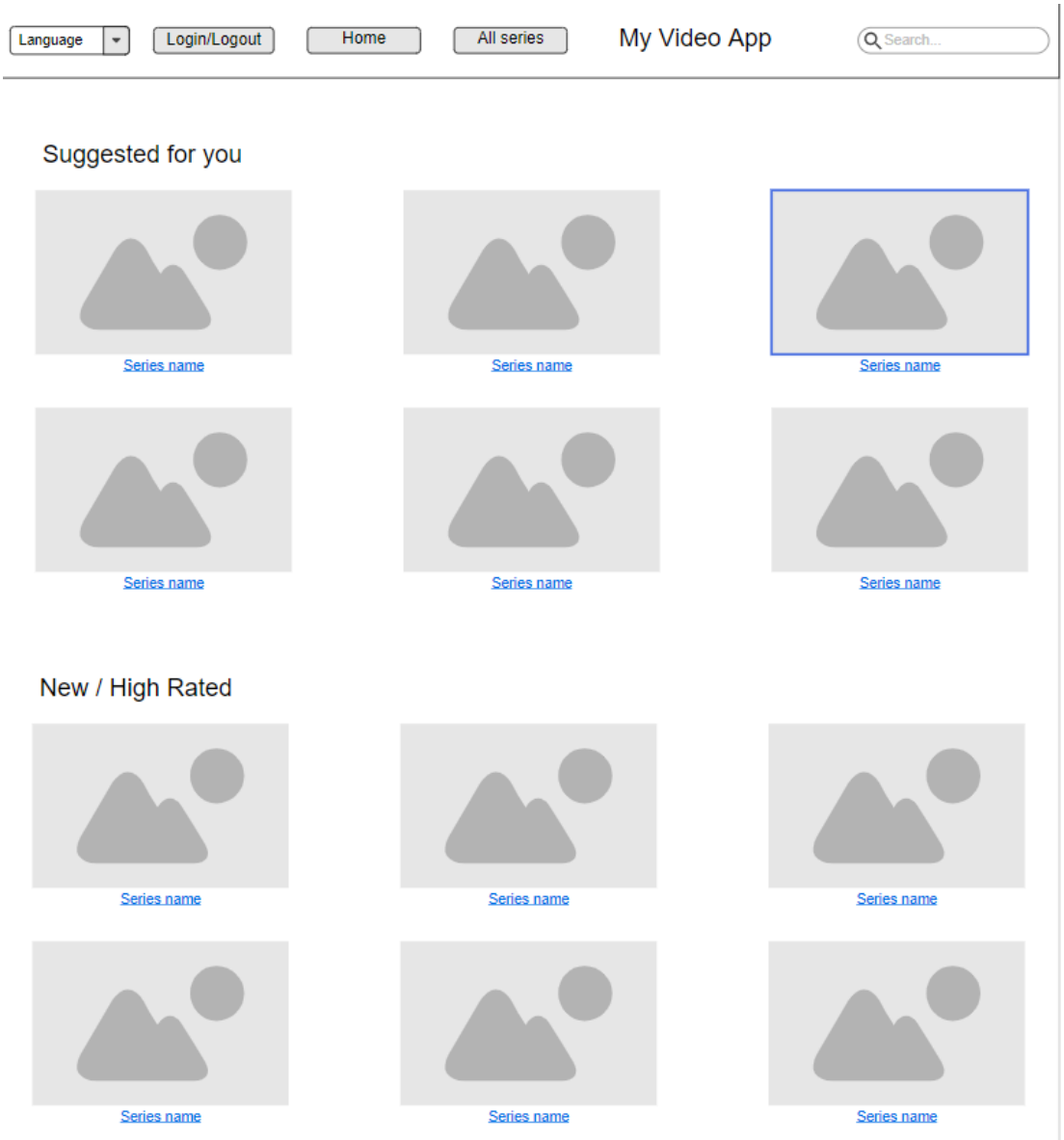
[7] Sami Antila, "Film Archive Management System" , Lucrare Licență, JAMK University of Applied Sciences, 2016, Available online at: https://core.ac.uk/download/pdf/80992638.pdf

[8] Ibnal Asad, Khalid Ahmed, Tanvir Rahman, Md. Saiedur , "Movie Popularity Classification based on Inherent Movie Attributes using C4.5, PART and Correlation Coefficient". International Conference on Informatics, Electronics & Vision 2012, DOI: 10.1109/ICIEV.2012.6317401. Available online at: https://www.researchgate.net/publication/261054688

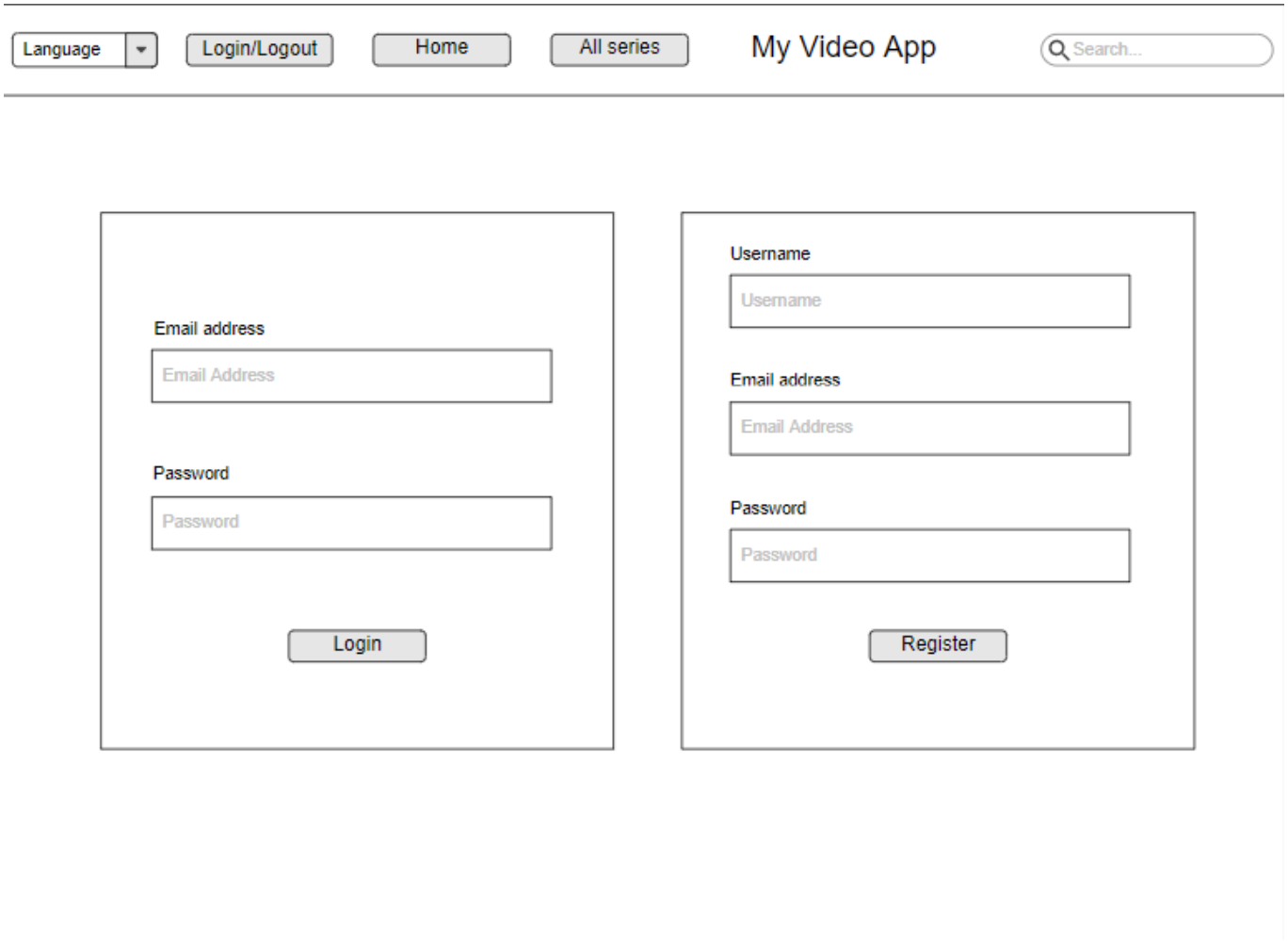
Lista figurilor

Lista tabelelor

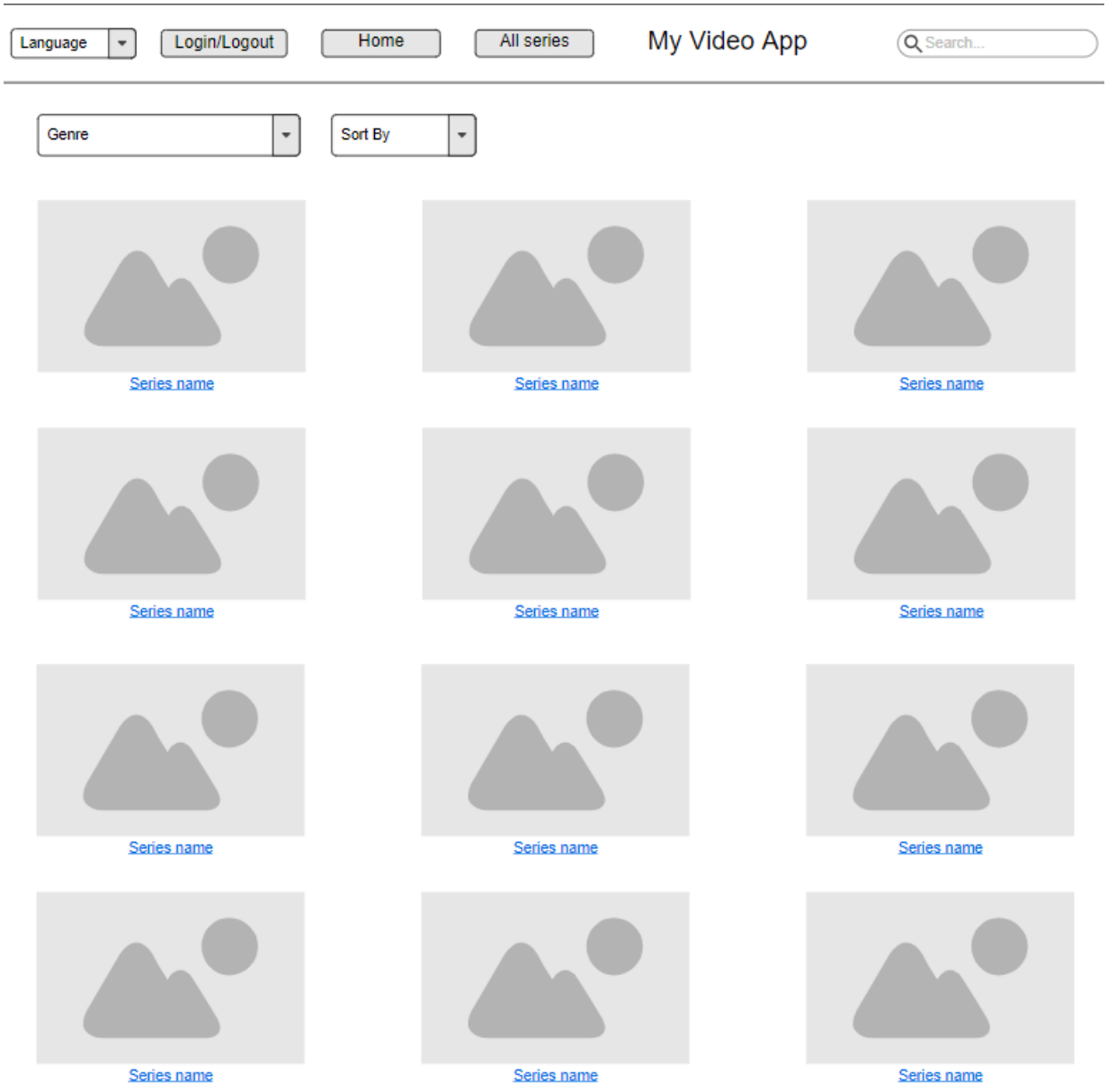
Anexa 1: UI Mockups



Mockup Homepage



Mockup Login/Register



Mockup pagină Toate Seriile

Mockup pagina Seriei