



COSC 31093 / BECS 31213

# Enterprise Software Design and Architecture

Group Project – Group 04

Netflix Case study

## **Acknowledgement**

As the students of group 4, we would like to express my sincere gratitude to Dr. Mrs. Toshini Kumarika for her invaluable guidance and support throughout this module. Her expertise and dedication have been instrumental in enhancing my understanding and skills. I would also like to thank all group members for their collaboration and teamwork. Their contributions and commitment have been essential to the successful completion of this assignment.

Thank you all for your encouragement and assistance.

## **Abbreviations**

ERP: Enterprise Resource Planning

ESD: Enterprise software development

SDLC: Software Development Life Cycle

BP: Business process

SP: Supporting process

ADM: Architectural design model

GP: Group project

REST: Representational State Transfer

API: Application Programming Interfaces

UI/UX: User Interfaces/User Experience

DFD: Data Flow Diagram

## **Table of Contents**

Acknowledgement .....	2
Abbreviations .....	2
Table of figures .....	5
List of tables.....	6
Introduction.....	7
What is Netflix? .....	8
Company overview .....	8
Types and categories of Netflix users .....	8
Available functionalities and feature of Netflix .....	8
Functional and non-functional requirements of Netflix system .....	10
Functional requirements .....	10
Non-functional requirements.....	12
Project scope statement.....	13
Technology stack .....	15
Backend development .....	15
Frontend development.....	15
Development environments and services .....	16
Proposed architecture for web service .....	17
System Design .....	17
Sequence Diagrams .....	17
User and Video Management .....	17
View management .....	19
Payment management.....	20
Class diagram .....	22

Use case diagram.....	25
UI designing.....	26
Enterprise application development report for Netflix web service .....	28
Scalability.....	28
Optimization suggestions for further improvements of Netflix web app.....	29
Security.....	29
Further extensions of security on Netflix web service .....	31
Integration .....	32
Customization and flexibility .....	33
Maintainability .....	34
User experience .....	34
Data management.....	37
Compliance and regulations .....	37
Monitoring and analytics.....	37
Collaboration and workflow.....	39
Quality and system testing.....	41
Acceptance criteria .....	41
Conclusion .....	42

## **Table of figures**

Figure 1: Netflix logo, (Nisha, 2023).....	7
Figure 2: Java.....	15
Figure 3: Spring boot .....	15
Figure 4: MySQL.....	15
Figure 5: HTML.....	15
Figure 6: JS .....	15
Figure 7: Bootstrap .....	15
Figure 8: IntelliJ IDEA .....	16
Figure 9: XAMPP Server.....	16
Figure 10: JDK 17.....	16
Figure 11; API architecture with Spring boot, (JavaPoint, 2021).....	17
Figure 12: Sequence diagram for user management.....	19
Figure 13: Sequence diagram for video management.....	20
Figure 14: Sequence diagram for payment management.....	21
Figure 15: Class diagram of Netflix web service - Part 01.....	22
Figure 16: Class diagram - Part 02 .....	23
Figure 17: Class diagram - Part 03 .....	24
Figure 18: Use case diagram of Netflix Web service .....	25
Figure 19: User interface design 01 .....	26
Figure 20: User interface design 02.....	26
Figure 21: User interface design 03 .....	27
Figure 22: Kubernetes.....	29
<i>Figure 23: Nginx</i> .....	29
Figure 24: Redis.....	29
Figure 25: Rabbit HQ.....	29
Figure 26: JWT Authentication and ASE .....	31
Figure 27: OAuth protocol and HTTPS.....	31
Figure 28: Frontend connection with backend.....	33
Figure 29: Backend connection with frontend.....	33
Figure 30: Status code usages .....	33

Figure 31: Exception handling.....	33
Figure 32: Components.....	33
Figure 33: Usages of components.....	33
Figure 34: F&Q in the web service.....	36
Figure 35: Desktop vie.....	36
Figure 36: Dashboard 01.....	38
Figure 37: Dashboard 02.....	38
Figure 38: Dashboard 03.....	39
Figure 39: Project management with MS Projec).....	40
Figure 40: Collaborative development with Git & GitHub .....	40

### **List of tables**

Table 1: Group members .....	7
Table 2: Uses and use cases of Netflix, (Netflix, 2024) .....	10
Table 3: API Testing .....	41

## **Introduction**

This group project is focused on developing a robust and scalable web service application for similar to Netflix which belongs to ESD, a leading subscription-based streaming service renowned for its extensive gallery of movies, TV series, documentaries, and original content. Group aim is to enhance Netflix's user experience by developing essential web services that manage key functionalities like user management, content cataloging, and personalized recommendations video management as well as payment handling with streamline monitoring and analytics.

The project will be executed in three milestones. The first milestone involves requirement analysis and planning, where we will outline the key services and their responsibilities, the technology stack, and the API endpoints. The second milestone focuses on designing the system using UML diagrams to visualize component interactions and system structure. The final milestone will be the implementation of the system as web services, adhering to REST architecture, ensuring scalability, security, and maintainability.

In this document, team has been focused on the project initiation, planning as well as requirements analysis. According to the recognized requirements, such as system and user requirements, the rest of the project will be going on. The system will be following RESTful API architecture to satisfy the BP and SP of Netflix operations.



*Figure 1: Netflix logo, (Nisha, 2023)*

Student number	Group Member
PS/2020/010	A.E.P.P. JAYASEKARA
PS/2020/012	A.M.L. ATHUKORALA
PS/2020/021	M.A.B. KAVEESH
PS/2020/024	K.D.C.D. FERNANDO

*Table 1: Group members*

## **What is Netflix?**

### **Company overview**

Netflix is one of the world's largest entertainment and video such movies, TV series, documentaries, and original content visualizing and presenting services with 270 million of paid customers in more than 190 countries enjoying and entertaining across a broad range of genres and language speakers. Members are free to survive, play, pause and resume watching as much as customers want, anytime, anywhere, and can change their plans at any time. The company aim to different age ranges and target population is very large. (Netflix, 2024)

### **Types and categories of Netflix users**

1. Individual subscribers
2. Family members (Profiles)
3. Administrators
4. Content creators/partners
5. Customer support representatives
6. Marketing team
7. Data analysts
8. System administrators
9. Investors (Shareholders/Owners/Partners)
10. Sponsors (For advertisements)
11. Other stakeholders

### **Available functionalities and feature of Netflix**

User	Use cases
Individual subscribers	Search, sort and explore movies, TV series, documentaries. Stream selected content, play as they want. Create and personalize individual profiles. Do user actions on profiles such as update and remove profile. Upgrade, downgrade, or cancel subscription plans. Provide feedback on watched content.



	<p>Access and manage the viewing history.</p> <p>Download content for offline viewing.</p> <p>Restrict content based on ratings for children.</p>
Family members (Profiles)	<p>Receive content recommendations based on individual viewing behaviors.</p> <p>Stream content tailored to the profile's preferences.</p> <p>Customize profile preferences and settings.</p> <p>Add movies and shows to a personal watchlist.</p>
Administrators	<p>Handle user registrations, profile management, and authentication.</p> <p>Add, update, and remove content in the catalog.</p> <p>Track and analyze system performance and usage statistics.</p> <p>Oversee subscription plans, billing cycles, and payment issues.</p> <p>Create and review various operational and performance reports.</p>
Content creators/partners	<p>Add new movies, TV shows, and documentaries to the platform.</p> <p>Update metadata, descriptions, and categories of existing content.</p> <p>Monitor viewership statistics and user feedback on their content.</p>
Customer support representatives	<p>Help users with login problems, password resets, and account recovery.</p> <p>Provide support for streaming issues and device compatibility.</p> <p>Assist with payment issues, subscription changes, and refunds.</p> <p>Collect and respond to user complaints and suggestions.</p>
Marketing team	<p>Design and implement campaigns to promote new releases and featured content.</p> <p>Study viewership trends and user behavior to tailor marketing strategies.</p> <p>Engage with users on social media platforms to promote content and gather feedback.</p>

	Test different marketing messages and promotions to optimize user engagement.
Data analysts	<p>Study user viewing patterns to generate insights and recommendations.</p> <p>Improve the algorithm for personalized content recommendations.</p> <p>Track key performance indicators to evaluate service performance.</p> <p>Create detailed reports on user behavior and content performance.</p>
System administrators	<p>Ensure the reliability and performance of servers and network infrastructure.</p> <p>Protect against unauthorized access and security breaches.</p> <p>Deploy updates and patches to the system.</p> <p>Manage data backups and disaster recovery processes.</p>

*Table 2: Uses and use cases of Netflix, (Netflix, 2024)*

## **Functional and non-functional requirements of Netflix system**

### **Functional requirements**

#### 1. User management

##### a. User registration and authentication

- Users should be able to register and log in using email and password or social media accounts like Google or Facebook.

##### b. Profile management

- Users should be able to create multiple profiles under a single account.
- The system should allow users to customize profile settings, including update and delete user account.

##### c. Subscription management

- Users should be able to choose, upgrade, or downgrade subscription plans.
- The system should automatically do the billing and payment handling.

##### d. Payment facility

- The users should be able to pay for their subscription plans through the system using credit or debit cards that have sufficient financial security.
- 2. Content management
  - a. Content Catalog
    - The system should allow us to insert, update, delete content and maintain a database of movies, TV series, and documentaries.
    - The system should allow administrators to add, update, and remove content.
  - b. Metadata management
    - The system should be maintained and managed metadata such as titles, descriptions, genres, and release dates.
    - The system should suggest recommendations based on viewing history and preferences.
- 3. Content delivery
  - a. Streaming services
    - The system should be played high-quality streaming of video content.
    - The system should be supported with multiple resolutions (e.g., SD, HD, 4K) and adaptive bitrate streaming.
  - b. Download for offline viewing
    - The system should allow users to download content for offline viewing on supported devices.
- 4. User interaction
  - a. Search functionality
    - The system should allow users to search for content by title, genre, actor, etc.
  - b. Rating and reviews
    - The system should allow users to rate and review content they have watched.
  - c. Watchlist management
    - The system should be able to add content to their watchlist for future viewing.

## 5. Parental controls

### a. Content restrictions

- Some system users should be able to restrict videos to others (Parents can limit or restrict videos to child).

### b. Profile-specific settings

- The system should be included setting parental controls on individual profiles.

## 6. Customer support

### a. Help and support

- The system should have a help center with FAQs and troubleshooting guides.
- The system should provide live chat and email support for user inquiries.

## **Non-functional requirements**

### ▪ Performance

The system should support many concurrent users without performance degradation time latency and bottleneck.

### ▪ Scalability

The platform should be able to scale horizontally to accommodate increasing user load and data volume without slowing the system when data storage is rising and user count increasing.

### ▪ Availability

The system should ensure high availability with minimal downtime through redundant systems and failover mechanisms.

### ▪ Reliability

The system should provide a robust and reliable service with consistent uptime and performance.

### ▪ Security

Protect user data with strong encryption both in transit and at rest. The system should ensure customer data privacy, financial security and data protection.

- Maintainability

The system should be written clean, modular, and well-documented code to facilitate maintenance and updates.

- Compatibility

The system should be compatible with a wide range of devices and operating systems.

- Compliance

The system should adhere to relevant industry regulations and standards, such as GDPR for data protection.

- Monitoring and analytics

The system should have implemented monitoring tools to track system performance and user behavior.

### **Project scope statement**

<b>Implementing Netflix case study as a web service</b>		
The system should be developed following REST API architecture using Spring boot, MySQL and using different web technologies. The final system implementation would be almost similar to the actual Netflix scenario and work similar to the use cases available in Netflix move and theater platform.		
The client	The anonymous client who required web service system like Netflix	
Project starts	01.07.2024	
Objectives	<ul style="list-style-type: none"> <li>▪ To complete requirement analysis and planning before the given deadline.</li> <li>▪ To define appropriate architecture for system design and development for both front-end and backend.</li> <li>▪ To deliver completed system on or before the agreed deadline with proposed deliverables.</li> </ul>	
	In scope	The group is agreed to implement,

Scope description		<ul style="list-style-type: none"> <li>▪ User management</li> <li>▪ Content management</li> <li>▪ Content delivery</li> <li>▪ Customer support</li> </ul>
	Out of scope	<ul style="list-style-type: none"> <li>▪ Watch history and download management</li> <li>▪ System deployment and monitoring</li> <li>▪ Maintaining and troubleshooting</li> </ul>
Deliverables	<ul style="list-style-type: none"> <li>▪ Complete implementation of defined requirements using pre-selected technologies.</li> <li>▪ Test plan and quality management plan (QMP)</li> <li>▪ End user documentation</li> <li>▪ Complete source code</li> <li>▪ User training session for administration operations.</li> </ul>	
Assumptions and inferences	When developing this web service, the development team assumes that the user (Client) needs a system which is exactly similar to Netflix.	
Acceptance criteria	The user will be accepted into the system only if all the predefined requirements have been achieved on or before the deadline.	
Identified risk	There is a risk when developing the system because this is very much similar to the actual Netflix platform.	
Project milestones	Provide requirement analysis and planning report Provide system architecture design with appropriate UML data. Provide final implementation with proposed deliverables.	
Constrains	The team little more worrying about the time given to the project completion.	

## **Technology stack**

### **Backend development**



*Figure 2: Java*



*Figure 3: Spring boot*



*Figure 4: MySQL*

*(PngWing, 2024)*

Spring boot is a very robust and micro framework with minimum configurations that helps to develop microservice for web application's backend and mobile application backends. Due to the high performance, platform independence and there are many services and web applications in the current world that is still running on java platforms. Also, the string boot has a quick approach to creating and deploying applications and services very easily. The MySQL is also free and opensource database widely used I very large enterprises to store their data and manipulate operations.

### **Frontend development**



*Figure 5: HTML*



*Figure 6: JS*

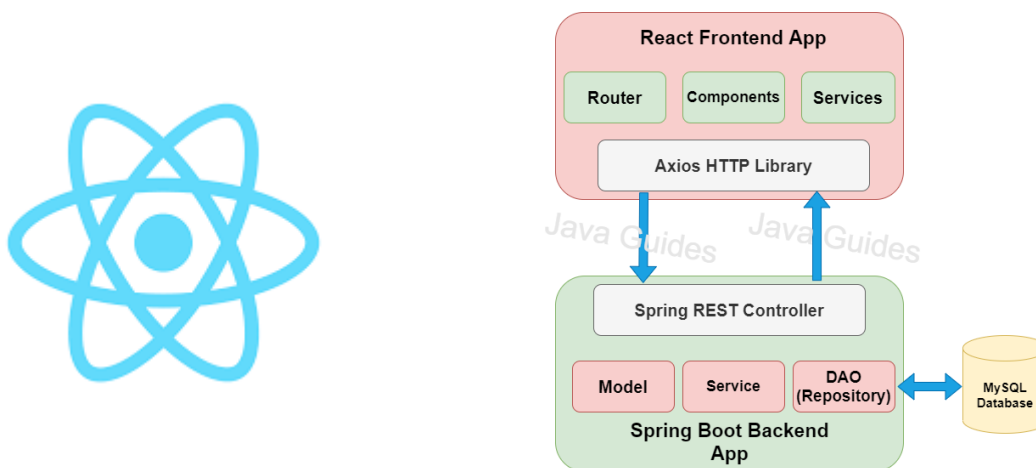
*(PngWing, 2024)*



*Figure 7: Bootstrap*

For the frontend development of this application, team was decided to use simple web technologies such as html for web pages designing, JavaScript for Realtime data manipulation and rendering, bootstrap for style web pages and maintain the consistency throughout the entire system.

But for better performance and utilize API architecture in a much better way, the team was moved to React.js frontend development. React.js was a free and robust frontend tool which is also a JavaScript library developed by Facebook company for UI frontend development with enhancing the feature of reusable components. To achieve these advantages, the team decided to choose React.js frontend for this Netflix webservice.



(Source: <https://www.javaguides.net/2020/07/spring-boot-react-js-crud-example-tutorial.html>)

### Development environments and services

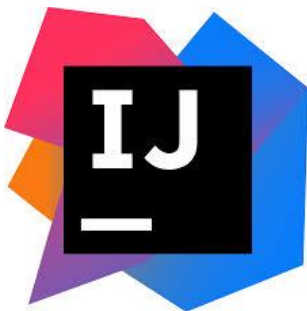


Figure 8: IntelliJ IDEA



Figure 9: XAMPP Server



Figure 10: JDK 17

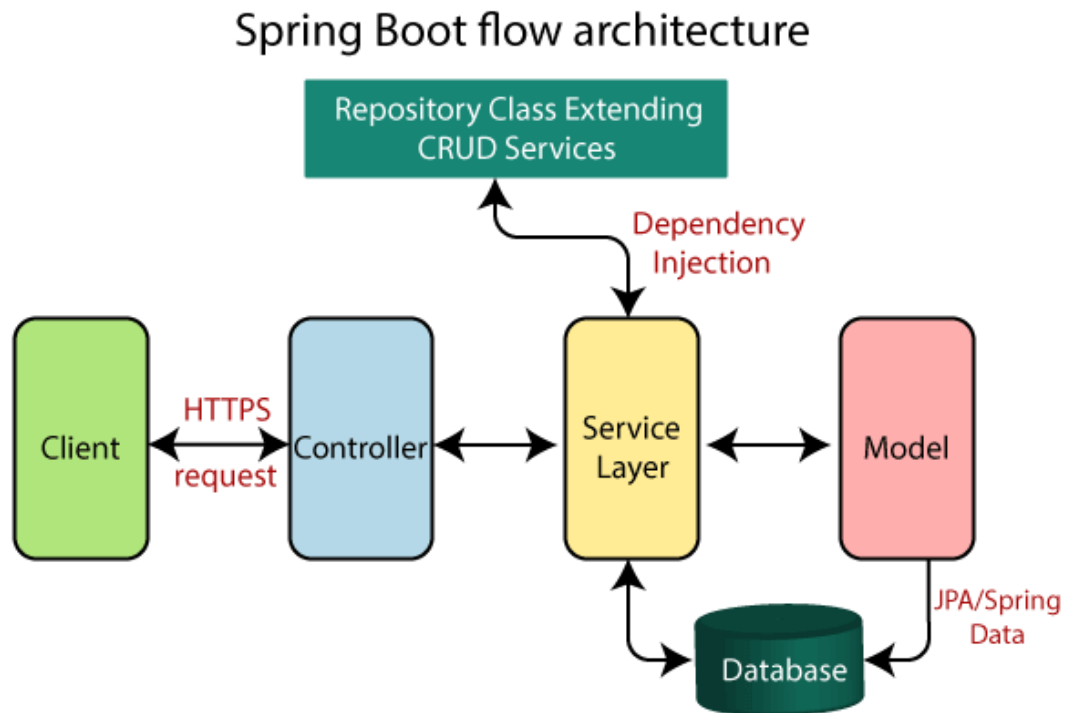
(PngWing, 2024)

IntelliJ is the most compatible IDEA for java development and the team was chosen JDK 17 which support more smoothly for spring boot and xampp was used to work with MySQL. Three of these



tools are freely available and easy accessibility was reason for choosing to develop Netflix application as well as postman will be using for API testing and manual blackbox and whitebox testing will be done for both frontend and backend.

### Proposed architecture for web service



*Figure 11; API architecture with Spring boot, (JavaPoint, 2021)*

### System Design

#### **Sequence Diagrams**

#### User and Video Management

#### Login/Register sequence

- User/Admin interacts with UserController to initiate login or registration.
- UserController calls UserRepository's findByUsername() method to fetch user data.
- UserRepository returns the user data to UserController.
- UserController sends the login/registration result back to the User/Admin.

#### Request video list

- User requests a list of videos from VideoController.

- VideoController calls VideoRepository's findAll() method to get all videos.
- VideoRepository returns the list of videos to VideoController.
- VideoController sends the video list back to the User.

#### View video

- User requests to view a specific video from VideoController.
- VideoController calls VideoRepository's findById(videoId) method to get the video details.
- VideoRepository returns the video details to VideoController.
- VideoController sends the video details to ViewController for viewing.
- ViewController calls ViewRepository's save(view) method to save the view.
- ViewRepository returns the view acknowledgment to ViewController.
- ViewController sends the view acknowledgment to VideoController.
- VideoController sends the view acknowledgment back to the User.

#### Manage users

- Admin requests the user list from UserController.
- UserController calls UserRepository's findAll() method to get all users.
- UserRepository returns the user list to UserController.
- UserController sends the user list back to the Admin.

#### Manage videos

- Admin requests the video list from VideoController.
- VideoController calls VideoRepository's findAll() method to get all videos.
- VideoRepository returns the video list to VideoController.
- VideoController sends the video list back to the admin.

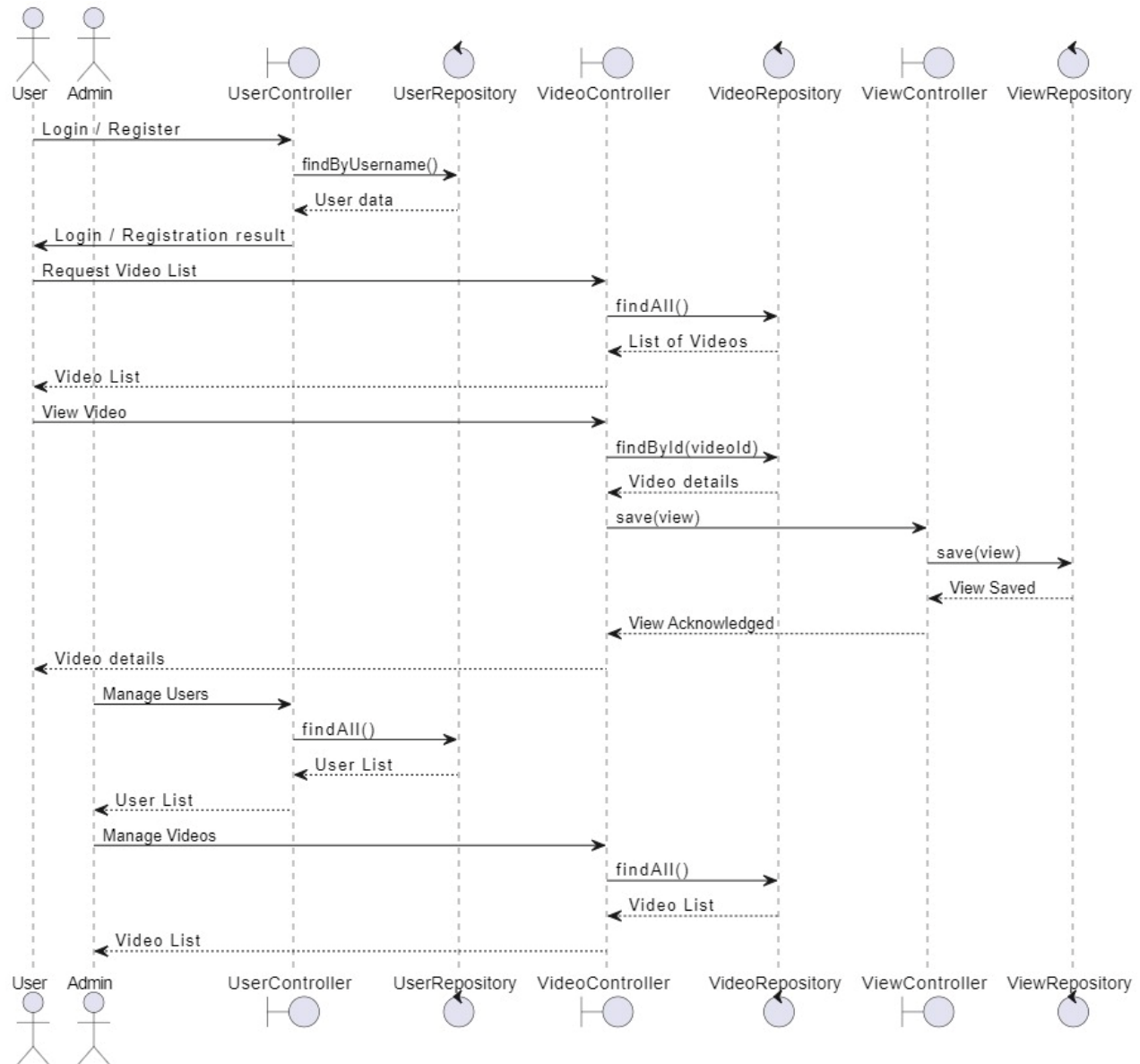


Figure 12: Sequence diagram for user management

## View management

### View video

- User/Admin requests to view a specific video from ViewController.
- ViewController calls ViewRepository's save(view) method to save the view.
- ViewRepository returns the view acknowledgment to ViewController.
- ViewController sends the view acknowledgment back to the User/Admin.

### Manage views

- Admin requests the view list from ViewController.
- ViewController calls ViewRepository's findAll() method to get all views.
- ViewRepository returns the view list to ViewController.
- ViewController sends the view list back to the Admin.

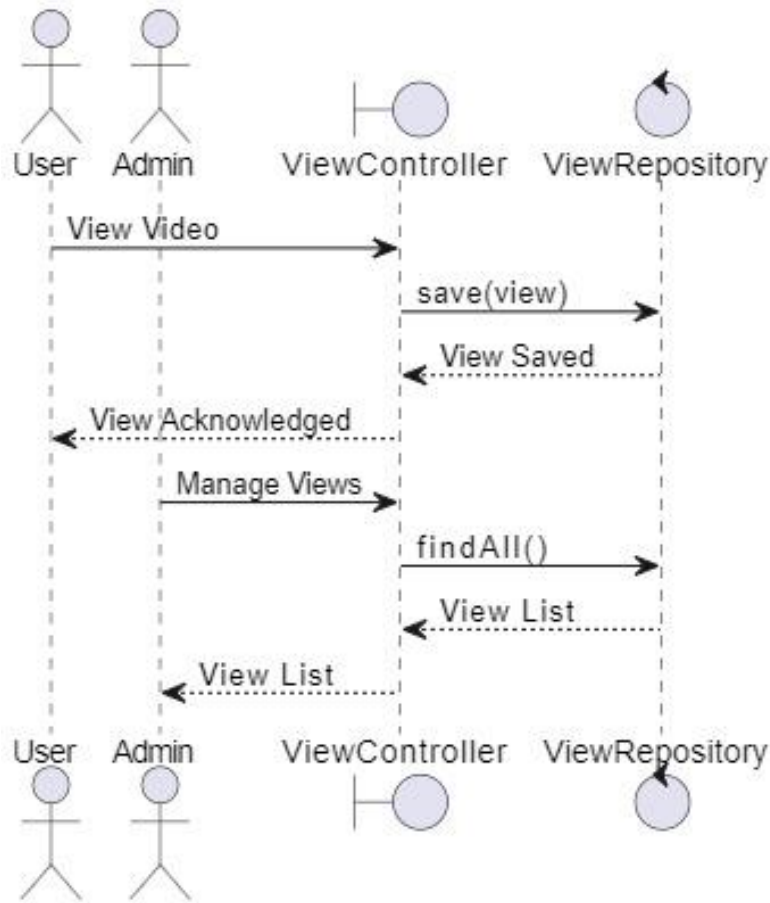


Figure 13: Sequence diagram for video management

## Payment management

### Make payment

- The process starts when a User initiates a payment by sending a "Make Payment" request to the PaymentController.
- The PaymentController processes this request and calls the save(payment) method on the PaymentRepository.
- The PaymentRepository confirms the payment by sending a "Payment Confirmation" back to the PaymentController.

- The PaymentController then communicates this success to the User with a "Payment Success" message.

### Manage payments

- An Admin can manage payments by sending a "Manage Payments" request to the PaymentController.
- The PaymentController handles this request by calling the findAll() method on the PaymentRepository to retrieve all payment records.
- The PaymentRepository returns the "Payment List" to the PaymentController.
- The PaymentController sends the "Payment List" to the admin for review.

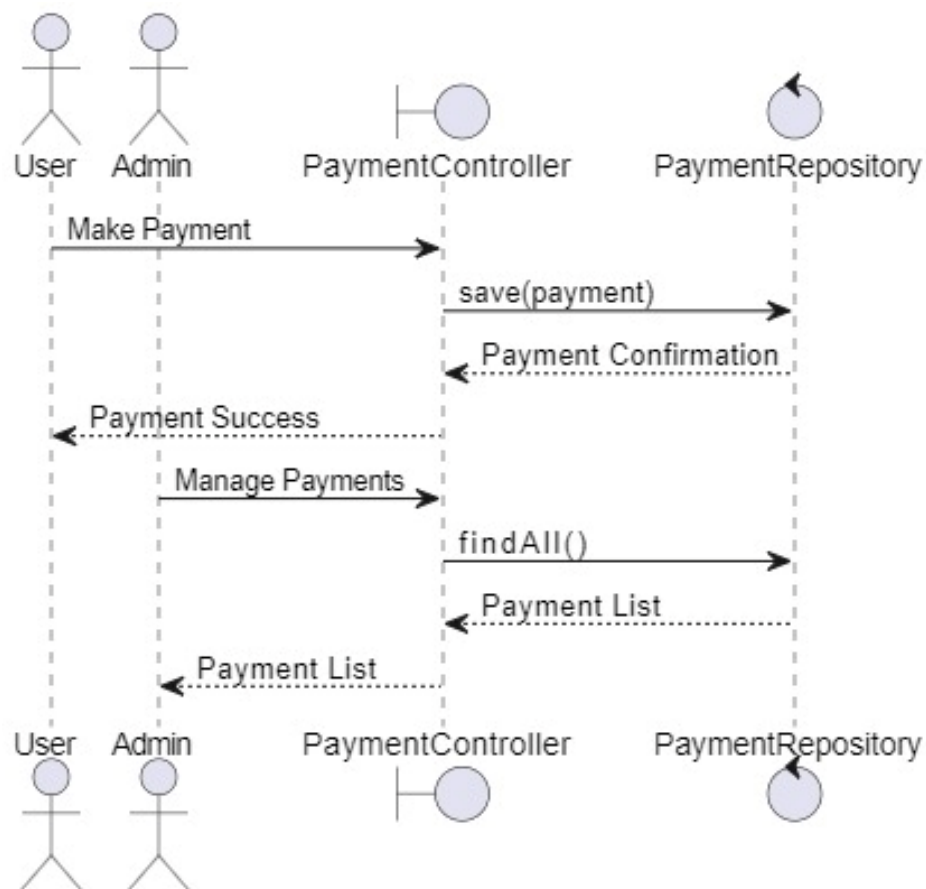


Figure 14: Sequence diagram for payment management

## Class diagram

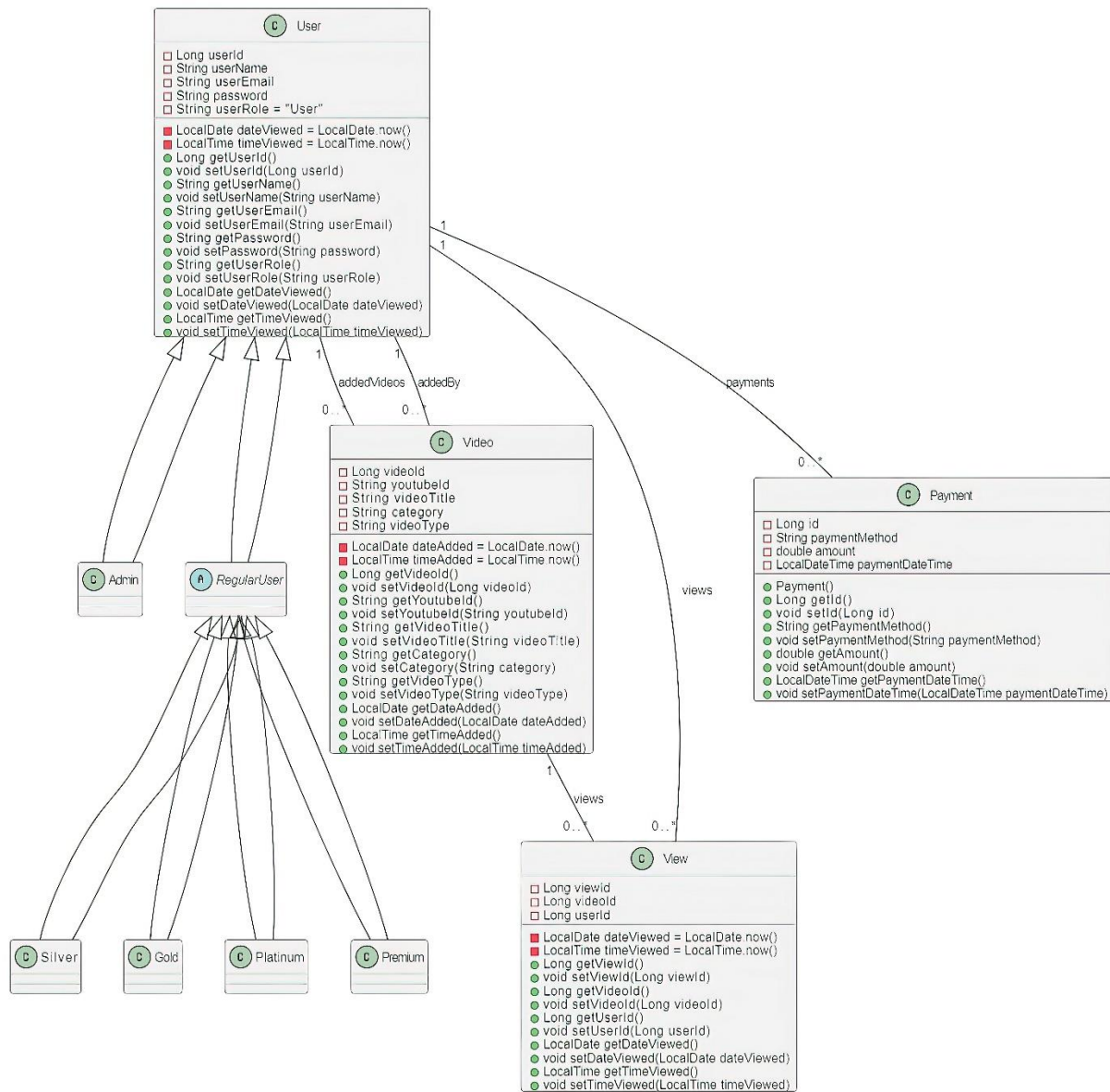


Figure 15: Class diagram of Netflix web service - Part 01

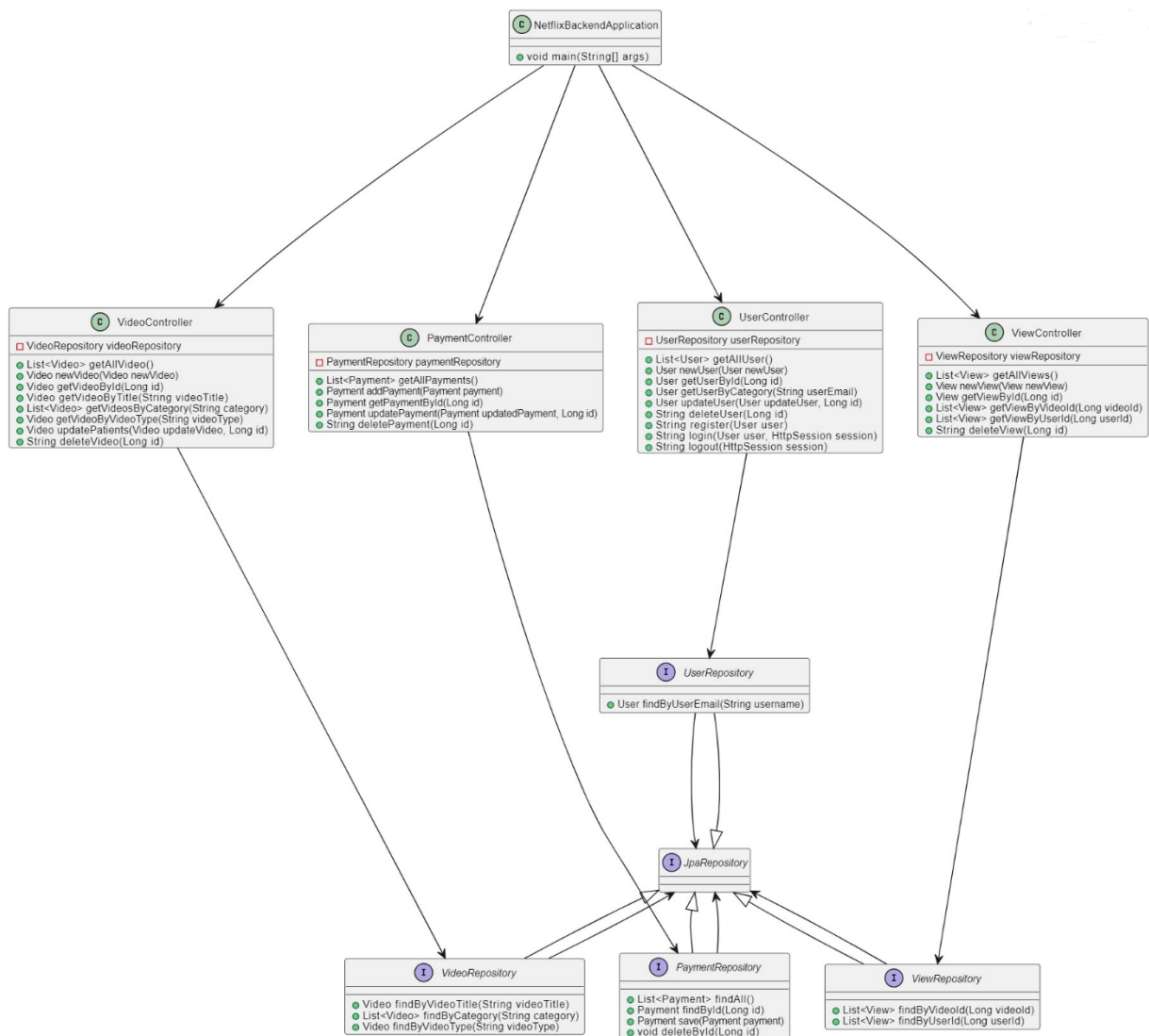


Figure 16: Class diagram - Part 02

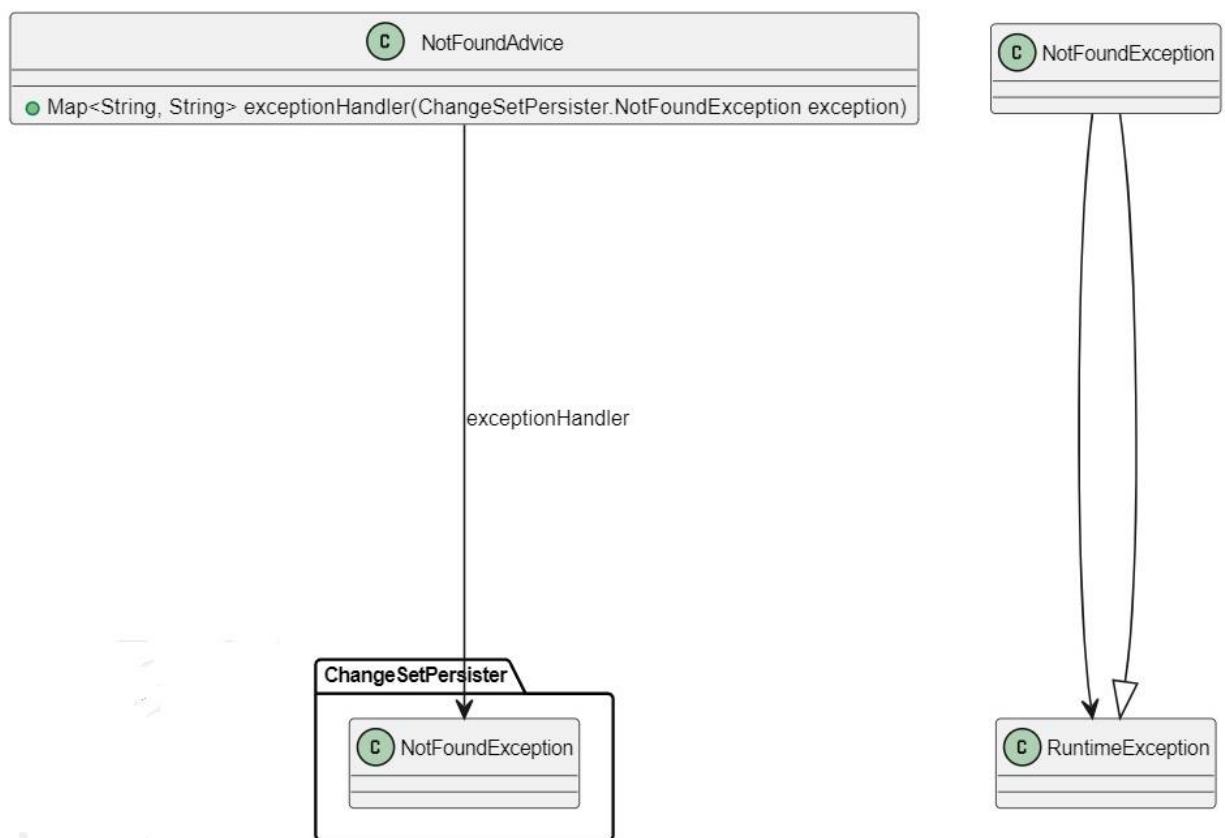


Figure 17: Class diagram - Part 03



## Use case diagram

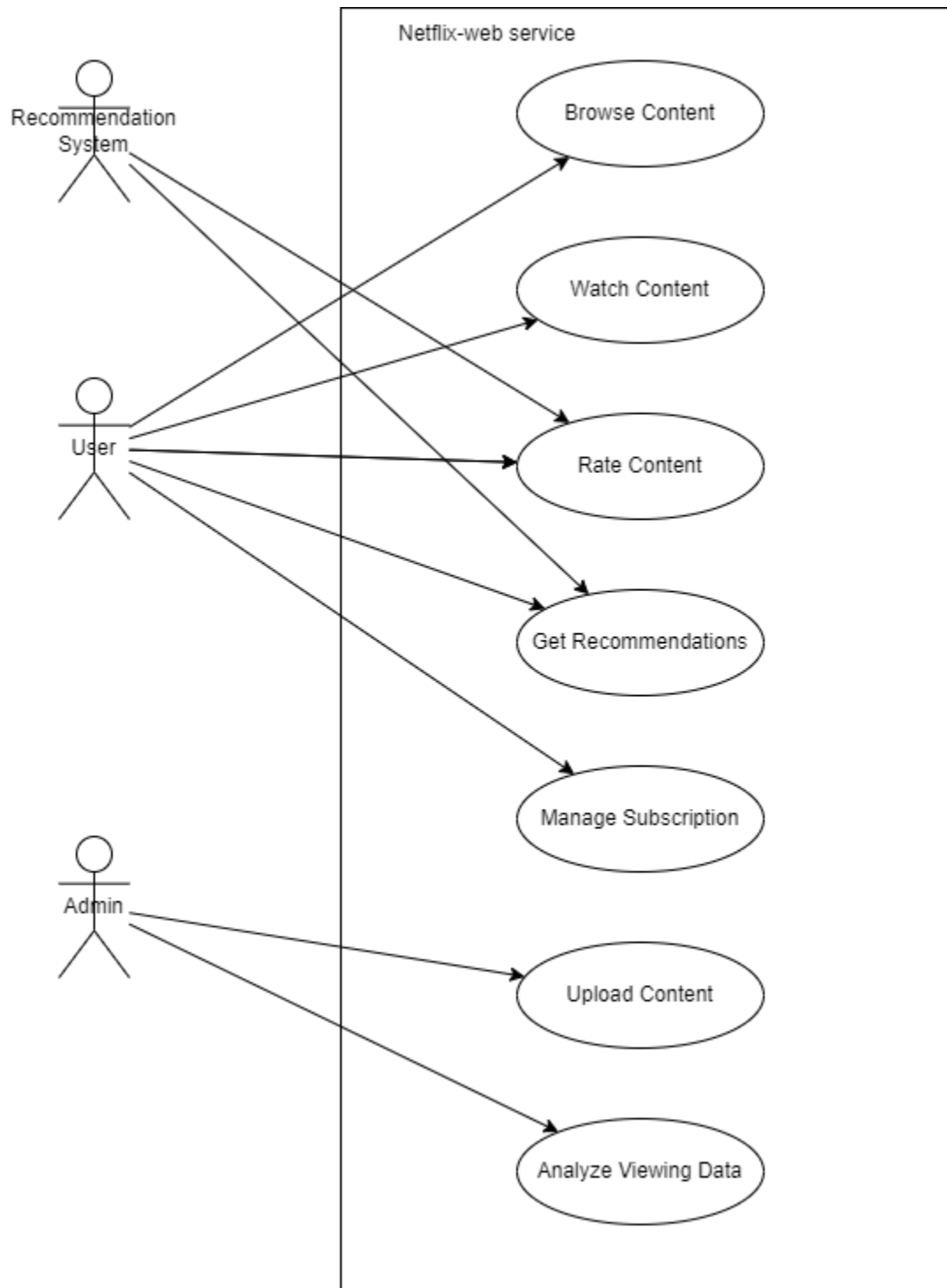


Figure 18: Use case diagram of Netflix Web service

## UI designing

All the UIs the system was developed using html, CSS, JavaScript and bootstrap for maintain the consistency of the final project as well as improve quality and give much better user experience using react.js functional components.

Here there are some of user interfaces,

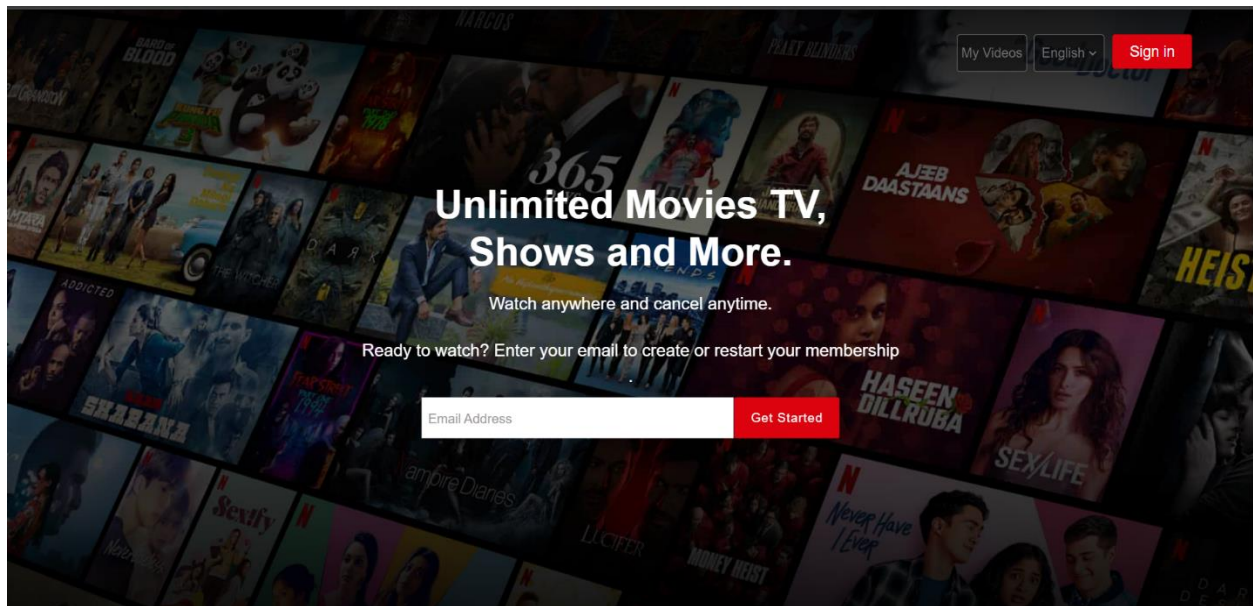


Figure 19: User interface design 01

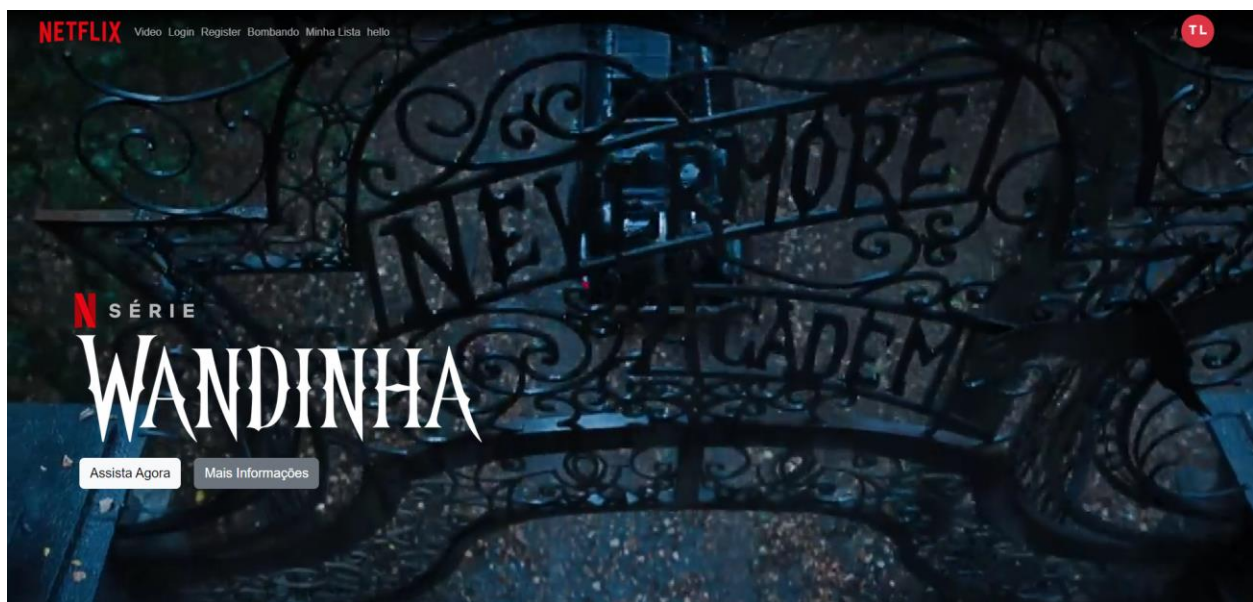


Figure 20: User interface design 02

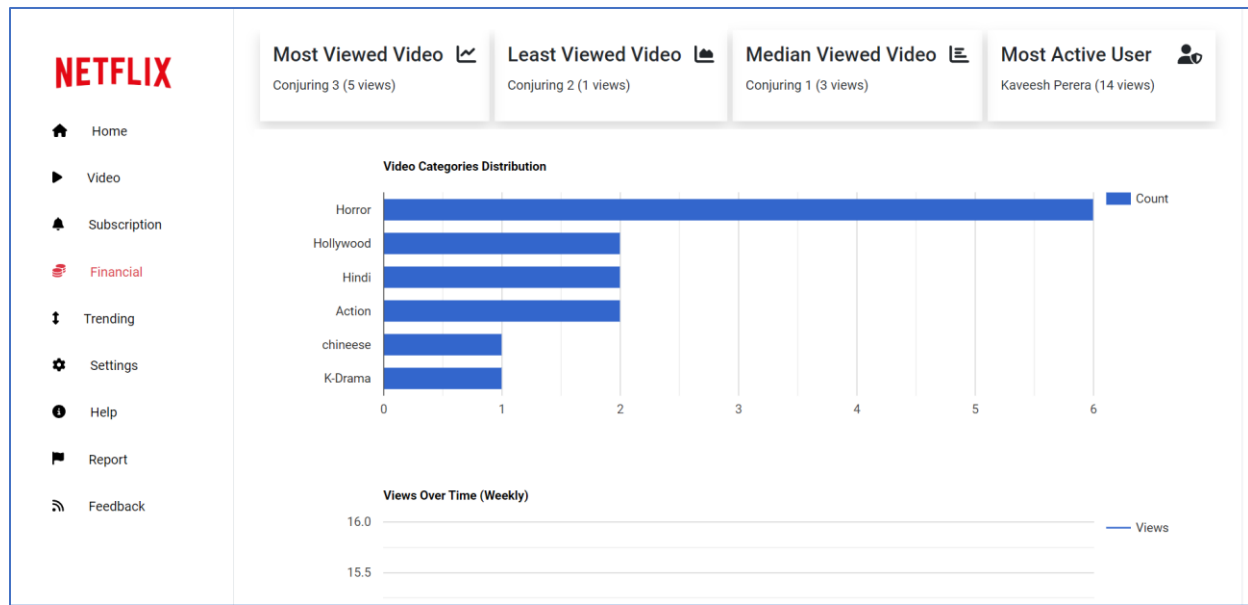


Figure 21: User interface design 03

## Enterprise application development report for Netflix web service

### Scalability

- Horizontal scaling

When deploying the system as two major components separately for frontend and backend it pretty much acquires some feature of microservice architecture because both are working separately and communicate with suitable communication protocol of HTTP/REST and service to service integration to achieve one goal to ensure that the developed web service is ready to scale up maximum ability of handle data when increasing the customers of developed web service.

- Load balancing network traffic balancing

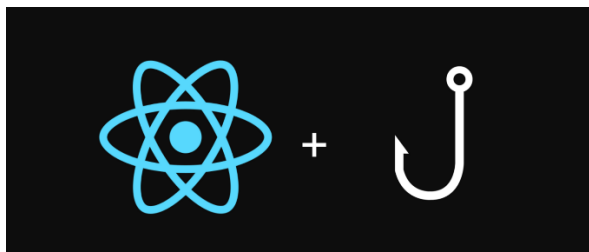
The system is ready to ensure optimal utilization of data, avoid bottlenecks as well as mitigate the network traffic by providing streamlined data flow to handle thousands of users at a time, with ensuring maximum throughput to the user and minimal jitter for provide maximum experience for quality video streaming experience to the user without caring whatever the device user is being used.

- Caching

The system has ensured that the system is ready to process real time data without latencies and maximum efficiency of data retrieval efficiency from database for both data insertions as well as data retrievals.

- Asynchronized processing

To reduce redundancies and time lacking for synchronization for the functionalities of dataflow of the system is going on. UseEffects are the example react hook for example for asynchronized data processing and used async function to streamline the data manipulation except using pure Ajax and jQuery.



Optimization suggestions for further improvements of Netflix web app.



*Figure 22: Kubernetes*



*Figure 23: Nginx*

Use Kubernetes for container orchestration of separate components for modules deployed as independent components, enabling automated scaling based on load and the Nginx to distribute incoming traffic across multiple instances, ensuring optimal resource utilization.



*Figure 24: Redis*



*Figure 25: Rabbit HQ*

Integrated caching mechanisms to reduce database load and improve response times for frequently accessed data retrieval in both read and write operations. Rabbit HQ is handling background tasks and processing heavy computations asynchronously to keep the application responsive. This will help reduce time and stop waiting for another task until the current one.

## **Security**

- Authentication and authorization

Has implemented appropriate login, signup and logout sessions to ensure user survival inside the webservice, the system guarantees that only the registered and paid users are only allowed to watch videos in the web service. Also, these sessions are protected timely routing to protect user details as well as system ethics.

## Frontend

```
5 AuthCheck.js > useAuthCheck
import { useEffect } from 'react';
import { useNavigate } from 'react-router-dom';

function useAuthCheck() {
  const navigate = useNavigate();

  useEffect(() => {
    const user = sessionStorage.getItem('user');
    if (!user) {
      navigate('/login');
    }
  }, [navigate]);
}

export default useAuthCheck;

useEffect(() => {
  const user = sessionStorage.getItem('user');
  if (!user) {
    navigate('/login');
  }
}, [navigate]);

const handleLogout = async () => {
  try {
    const response = await axios.post('http://localhost:8080/user/logout');
    setMessage(response.data);
    sessionStorage.removeItem('user');
    navigate('/login');
  } catch (error) {
    setMessage('Error logging out');
  }
};

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post('http://localhost:8080/user/login', { userEmail, password });
    setMessage(response.data);
    if (response.data === 'Login successful') {
      sessionStorage.setItem('user', userEmail);
      navigate(`/gallery/${userEmail}`);
    } else {
      setMessage({ text: 'Invalid Email or Password, Try again', class: 'text-white text-saml1' });
    }
  } catch (error) {
    setMessage({ text: 'Error Login, Try again', class: 'text-danger' });
  }
};
```

## Backend

```
@PostMapping("/login")
public String login(@RequestBody User user, HttpSession session) {
  User existingUser = userRepository.findByUserEmail(user.getUserEmail());
  if (existingUser != null && existingUser.getPassword().equals(user.getPassword())) {
    session.setAttribute("user", user);
    return "Login successful";
  } else {
    return "Invalid username or password";
  }
}

@PostMapping("/logout")
public String logout(HttpSession session) {
  session.invalidate();
  return "Logged out successfully";
}
```

- Data encryption

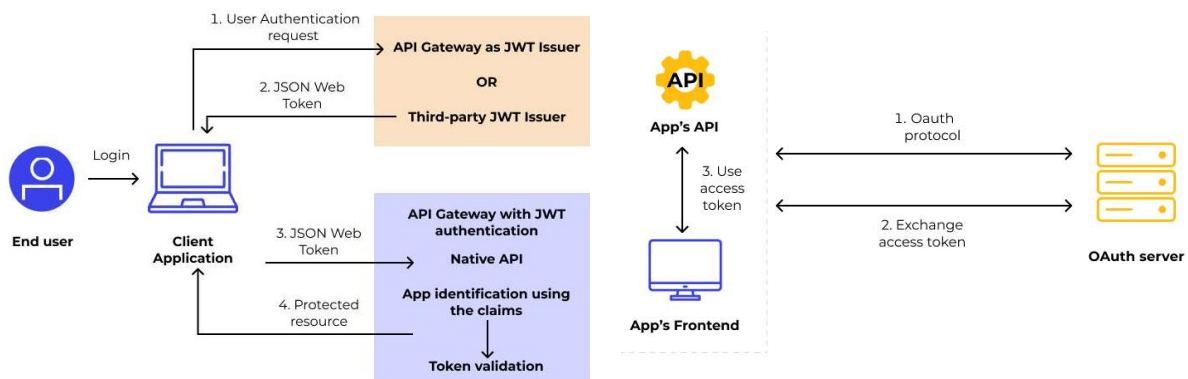
When storing and retrieving data from databases the data is encrypted and all data in models has been encapsulated with appropriate mechanisms.

<pre>private Long userId; private String userName; private String userEmail; private String password; @Column(name = "userRole") private String userRole = "User"; @Column(name = "dateViewed") private LocalDate dateViewed = LocalDate.now(); @Column(name = "timeViewed") private LocalTime timeViewed = LocalTime.now();</pre>	<pre>public Long getUserId() {     return userId; }  public String getUserRole() {     return userRole; }  public void setUserRole(String userRole) {     this.userRole = userRole; }</pre>
--	---

- Vulnerability management and security coding practices

Robust testing, investigation and monitoring is focused on risk mitigation and vulnerability reduction for the developed system. All the introductory level mechanisms of risk mitigation have been established to ensure the security of the system.

### Further extensions of security on Netflix web service



### Advanced Encryption Standard (AES)



Figure 26: JWT Authentication and ASE



Figure 27: OAuth protocol and HTTPS

## Integration

### RESTful API


- Design principles

The team followed RESTful principles to design APIs that are stateless, scalable, and easy to use for any kind of population in community. Each resource is represented by a unique URL with format (), and standard HTTP methods (GET, POST, PUT, DELETE) are used to perform operations on these resources.

```
const [titleResponse, categoryResponse, typeResponse] = await Promise.all([
  axios.get(`http://localhost:8080/video/searchByTitle/${query}`),
  axios.get(`http://localhost:8080/video/searchByCategory/${query}`),
  axios.get(`http://localhost:8080/video/searchByVideoType/${query}`)
]);
```

- Resource modeling

The entire application was modeled resources based on business entities and their relationships such as videos, users, views and resource include attributes like title, description, genre, and release date, while a "User" resource includes attributes like username, email, and subscription details.



```
@Entity
public class Video {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long videoId;
    private String youtubeId;
    private String videoTitle;
    private String category;
    private String videoType;
    @Column(name = "dateAdded")
    private LocalDate dateAdded = LocalDate.now();
    @Column(name = "timeAdded")
    private LocalTime timeAdded = LocalTime.now();
}
```

- Versioning on different controls

Implemented API versioning to ensure backward compatibility and allow for seamless updates and enhancements without disrupting existing clients. Each crud operation is having unique URL to communicate with frontend and backend.



```
'http://localhost:8080/view/add',
'http://localhost:8080/user/register'
'http://localhost:8080/view/add',
```

Figure 28: Frontend connection with backend

```
@RestController
@CrossOrigin("http://localhost:3000")
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserRepository userRepository;
```

Figure 29: Backend connection with frontend

- Error and exception handling

The team worked with standard status error responses with appropriate HTTP status codes and meaningful error and success messages to help clients understand and resolve issues. Also, the team used custom exception to give maximum user experience to customer to enhance the useability and stop unexpected termination of project.

```
if (response.status === 200) {
    console.log('View added successfully');
```

Figure 30: Status code usages

```
public class NotFoundException extends RuntimeException{
    public NotFoundException(Long id){
        super("Could not found the entity with id "+id);
    }
    public NotFoundException(String character){
        super("Could not found the entity with id "+character);
    }
}
```

Figure 31: Exception handling

## Customization and flexibility

- Modular design

The system has developed as sperate component, even the main project is developed as two separate components for frontend and backend. Also, the entire frontend was developed with reusable react.js functional components. Due to this feature any update can be done on simple components without affecting the other components of the system.

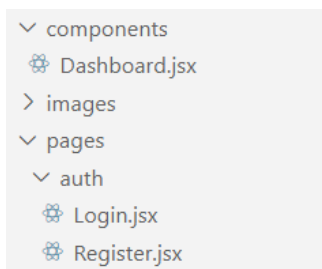


Figure 32: Components

```
<YouTube videoId={video.youtubeId}
    opts={opts} onReady={onReady}/>
<div className='p-4 border rounded'>
    <Dashboard />
</div>
```

Figure 33: Usages of components

## **Maintainability**

- Code quality

The team followed appropriate coding standards and best practices when developing the system to ensure that the code is clean, highly readable and encourage easy maintenance with flexibility of manipulation for continuous code quality checks with continuous development, deployment and integration.

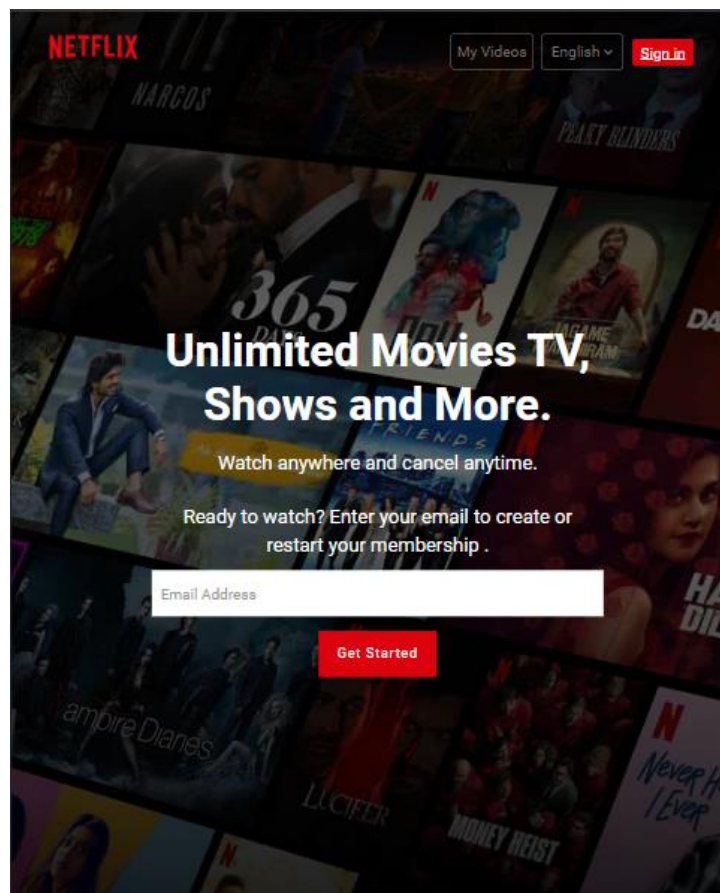
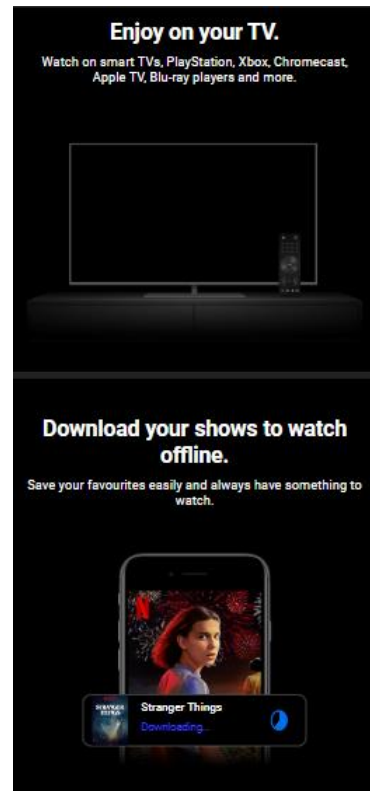
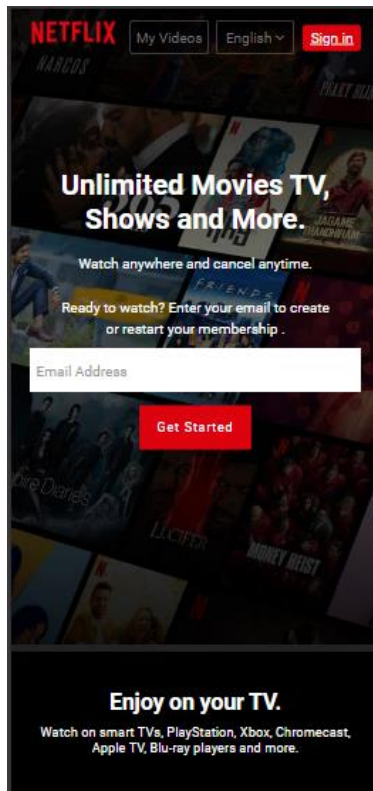
- Automated and manual testing

The system has contributed to all of the test steps, because due to this being a collaborative work each member of the group was responsible to develop separate components and they are also responsible for unit testing as well as integration testing. After successfully completing the project the system testing was completed and still doing acceptance testing as well as regression testing parallel to the continuous development and integrations.

## **User experience**

- Responsive and comfortable design

The web application was designed using html, CSS, JS and bootstrap for better user experience and developed attractive user interfaces. All the user interfaces have been styled using bootstrap as well as customized CSS. Also chose colors with convenient with the user's eyes because the web service is being used by the customers for more than hours. The web service is responsive for any device and any screen without losing performance and uniquely acts their role by each component.



- User feedback

The system and developers are open to hear to the customers and make changes appropriately. The feedback portal is available with the system, and it is continuously monitored to provide better user experiences to the user and F&Q is also available. Feedback tracking has automated with suitable methodologies and the system has potential to prioritize the user needs based on the number of requests and its severity and also the team is ready to rapid troubleshooting and recovery for sudden emergencies without login customer satisfaction about the web service.

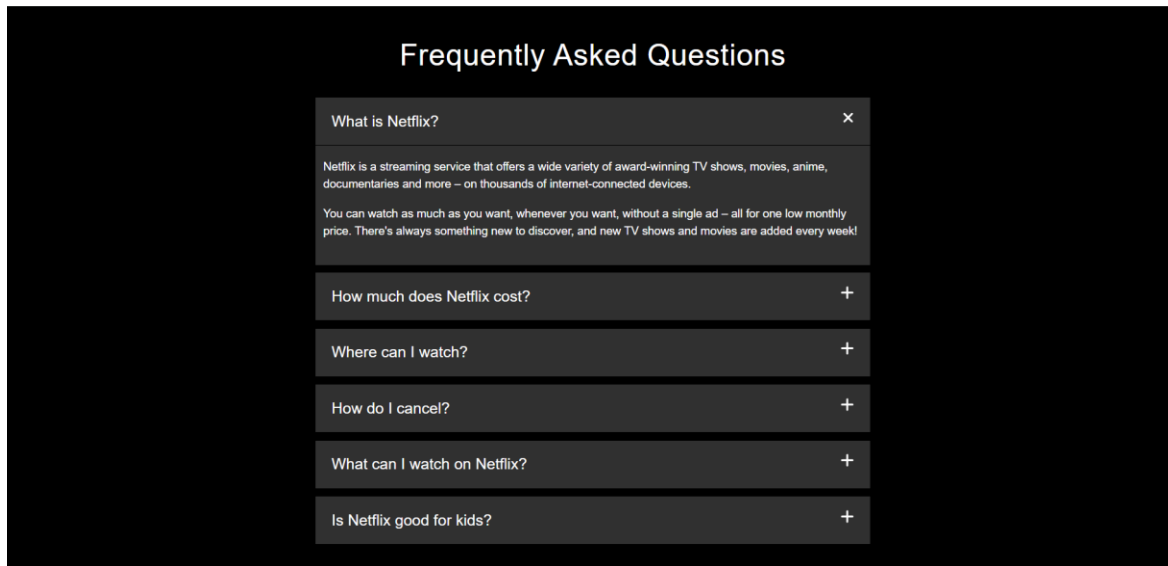


Figure 34: F&Q in the web service



Figure 35: Desktop vie

- Accessibility

The system is accessible for any user who has access to the web service with an account and selected prepaid subscription plan and can be accessed the system using any device which has an active internet connection.

## **Data management**

- Database management

The team was selected MySQL as database for Netflix web service and it is relational database with sufficient performances for dealing with huge amount of data and it is used by thousands of companies very large organizations over world to store and handle their data. In this case the entire system is depending on the relational SQL data which has ability to interact with the system properly and the features available in the spring boot, that was helped to easy data manipulation without querying. Also, there were very few steps of integration for the database and the system.

- Data backup and recovery

The system is possible to implemented automated data backup and recovery processes to prevent data loss and the system can easily upgrade to NoSQL data.

- Data consistency

The system has ensured data consistency and integrity using database manipulations with spring data JPA and distributed data management techniques.

## **Compliance and regulations**

- Data privacy

The system has guaranteed and ensured both customer and content's data protection based on the British data protection regulations (1998).

## **Monitoring and analytics**

- Application Performance Monitoring (APM)

The system performance and other nonfunctional features are measured and continuously evaluated through APM system. This will help to maintain the quality of service and consistency.

Also, this ensures the timely maintenance and troubleshooting for smooth and robust operations on the developed system.

- User analytics

Implemented user behavior analytics to understand and improve user engagement. The system consists of different data visualization methods charts, graphs and quantitative data visualization.

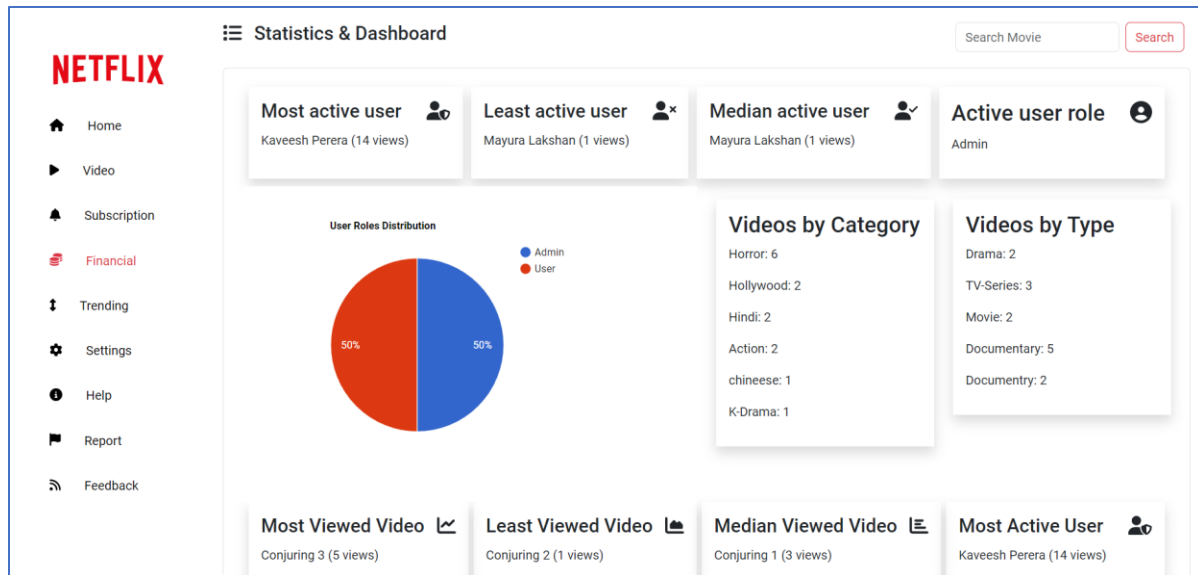


Figure 36: Dashboard 01



Figure 37: Dashboard 02



*Figure 38: Dashboard 03*

### **Collaboration and workflow**

- Collaboration tool (GitHub)

The team used GitHub for collaboration, collaboration and continuous development of Netflix web service, also this GitHub helps and enables team to collaboratively develop the system working remotely and merge all for final output.

- Project management (MS Project)

When a project is going on, the team was used Microsoft project tool to monitor progress and manage development time.

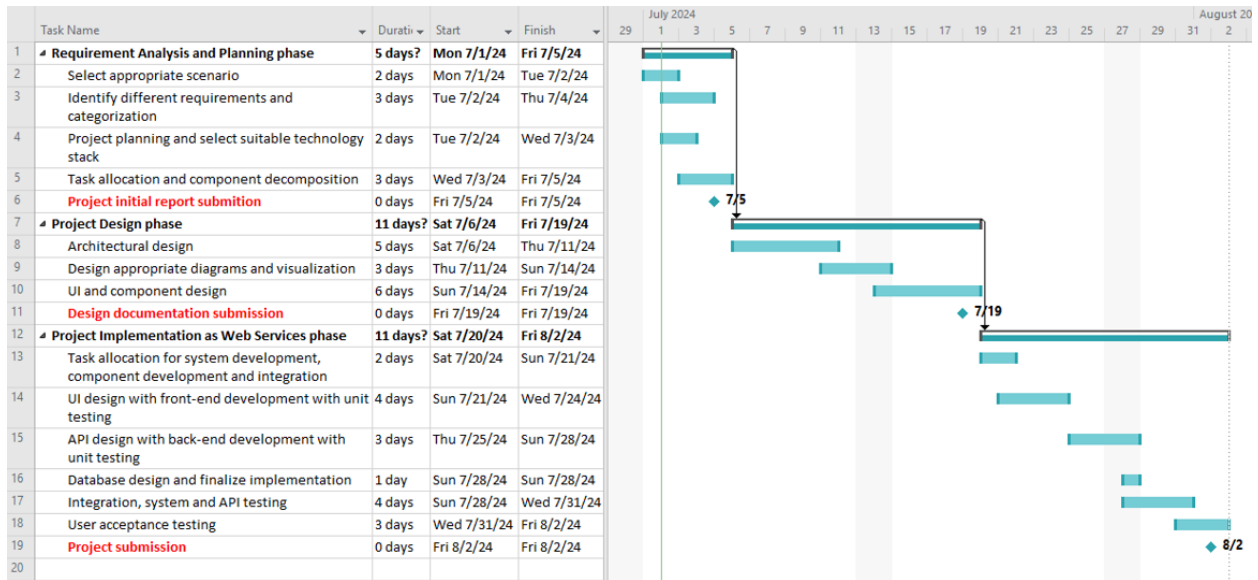


Figure 39: Project management with MS Project)

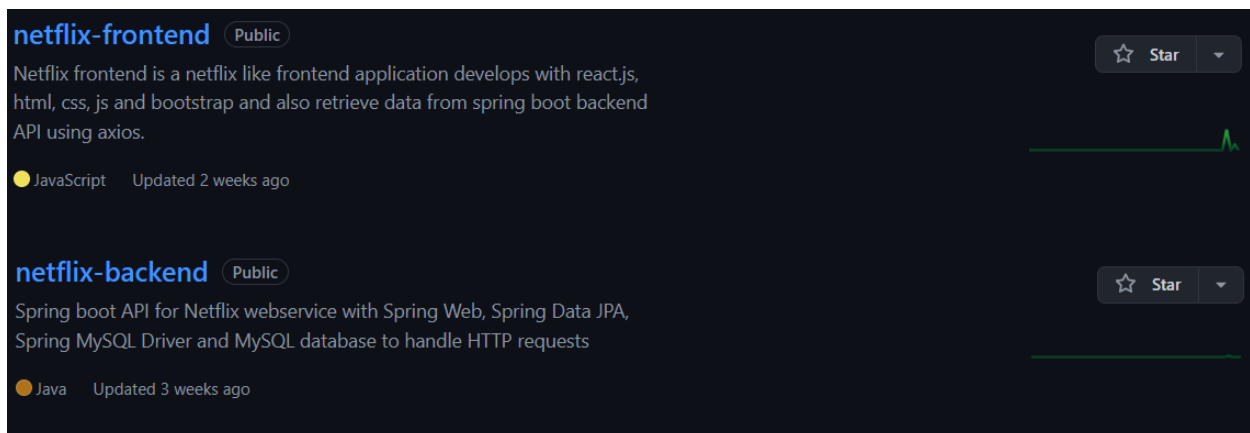


Figure 40: Collaborative development with Git & GitHub



### **Quality and system testing**

URL	Method	Status
http://localhost:8080/user/add	POST	200 OK
http://localhost:8080/view/add		200 OK
http://localhost:8080/video/add		200 OK
http://localhost:8080/payment/add		200 OK
http://localhost:8080/user/register		200 OK
http://localhost:8080/user/login		200 OK
http://localhost:8080/view/searchById/2	GET	200 OK
http://localhost:8080/view/searchAll		200 OK
http://localhost:8080/video/searchByCategory/action		200 OK
http://localhost:8080/video/searchByVideoType/Movie		200 OK
http://localhost:8080/video/searchAll		200 OK
http://localhost:8080/video/searchById/7		200 OK
http://localhost:8080/user/searchById/7		200 OK
http://localhost:8080/user/searchByUsername		200 OK
http://localhost:8080/user/searchByEmail		200 OK
http://localhost:8080/user/searchByUserRole		200 OK
http://localhost:8080/video/delete/1	DELETE	200 OK
http://localhost:8080/view/delete/1		200 OK
http://localhost:8080/user/delete/4		200 OK
http://localhost:8080/payment/delete/1		200 OK
http://localhost:8080/video/update/1	PUT	200 OK
http://localhost:8080/view/update/1		200 OK
http://localhost:8080/user/update/1		200 OK
http://localhost:8080/payment/update/1		200 OK

*Table 3: API Testing*

### **Acceptance criteria**

The system acceptance requirement is minimum viable product (MVP) without user interfaces but in this project the team has provided the completed product with proposed functionalities.

## **Conclusion**

This project, developed by a team of four students, demonstrates the implementation of key enterprise application development aspects in a Netflix web service. Utilizing React.js for the frontend and Spring Boot for the backend, the team ensured the application is scalable, secure, and maintainable. RESTful APIs were meticulously designed for seamless integration with external systems, enhancing the application's flexibility and modularity.

The project's focus on performance through caching, load balancing, and asynchronous processing ensures it can handle increasing user loads efficiently. Security measures, including HTTPS and secure coding practices, safeguard user data and maintain integrity. GitHub and Git were used for collaborative development, while MS Project facilitated effective project and time management. The responsive design and accessibility standards ensure a positive user experience across devices.

In summary, this project showcases the practical implementation of enterprise application development principles, resulting in a robust, flexible, and user-friendly application that can meet the demands of modern enterprises. The collaboration and dedication of the team members were pivotal in achieving these objectives.