Bangladesh University of Professionals

Faculty of Science and Technology
Department of Information & Communication Technology (ICT)

Course Title: Web Technologies Lab

Course Code: ICE-3206

A Report on

iFlix– Movie Server

Submitted to

Atika Akter

Lecturer

ICT, BUP

Submitted by

Section - A

Md. Taufik Hasan	2254901001
Raian Alif	2254901035
Samia Maliha	2254901043

Date: 09-07-2025

Project Title: iFlix– A Responsive Movie Streaming Platform

Project Overview: The goal of this project was to build a fully functional movie web application that offers users the ability to create and manage personal accounts, maintain a profile with preferences, and interact with movie-related features like saving favorites for future viewing. This platform simulates a real-world streaming service where users can browse content, and administrators can manage user-related data. The system ensures secure login and session handling, file uploads for profile images, and dynamic user interaction. It supports both user-facing pages (e.g., profile management, content interaction) and an admin dashboard for overseeing users and activity. By incorporating both client-side and server-side technologies, this project demonstrates how modern web applications work with persistent data, authentication, and dynamic user interfaces. The application emphasizes best practices in security, modular backend logic, and frontend responsiveness. It also focuses on offering seamless user experience with features like real-time profile updates, image previewing before upload, and informative dashboard elements.

Technologies Used

This project integrates several modern web technologies to deliver dynamic, interactive, and secure full-stack movie streaming and user management platforms.

Frontend Technologies:

HTML: Used to structure all web pages, including forms, buttons, tables, and layout elements. HTML ensures semantic organization of content and supports multimedia features essential for a streaming platform.

CSS: Used to style the web pages, ensuring responsiveness and aesthetic appeal across different devices. Features like Flexbox and Grid were used to organize layouts, while transitions and hover effects enhanced interactivity.

JavaScript: JavaScript was used to handle dynamic behavior on the client side, including form submission, real-time DOM manipulation, image preview for profile pictures, and event-driven updates. Fetch API is used to make asynchronous requests to the backend.

Backend Technologies:

Node.js: Node.js serves as the runtime environment for executing server-side JavaScript. It allows efficient handling of requests and non-blocking I/O operations.

Express.js: A minimalist web application framework for Node.js, Express was used to manage routing, middleware, and API endpoints. It simplifies the development of RESTful APIs and supports modular application architecture.

Database:

MySQL: MySQL is used as the relational database management system to store user information, movie details, and user-specific actions like profile data and streaming history. Queries are executed using both mysql2 with callback and promise-based APIs to ensure performance and flexibility.

Other Tools and Libraries:

EJS (Embedded JavaScript Templates): While most frontend pages are static .html files, EJS can be optionally used to render dynamic HTML on the server side using embedded JavaScript, particularly useful when templating data directly from the backend.

Fetch API: Utilized in frontend scripts to asynchronously communicate with backend routes for user login, profile updates, subscription management, and dynamic page rendering, enhancing user interactivity without full page reloads.

Chart.js: A powerful JavaScript library for data visualization. It can be integrated into the admin dashboard to show analytics such as user activity, subscription trends, or popular genres through interactive charts.

Font Awesome: A rich icon toolkit used throughout the site for visual icons like buttons, menus, user profile links, and navigation items. It helps improve the UI's clarity and aesthetics.

Express Middleware & File Upload: Middleware like express-file upload is used for handling uploads of profile pictures and movie posters. Other middleware includes cookie parsing, body parsing, and serving static assets from directories like public/ and protected/.

Multer / express-file upload: These middleware libraries are used to handle file uploads securely and efficiently. In the project, they enable users to upload profile pictures and allow admins to upload movie posters. Uploaded files are stored in designated directories like uploads/ or public/img, ensuring organized media management.

Crypto (Node.js built-in module): Used for securely hashing user passwords using SHA-256 to protect against plain-text password storage.

Cookies / Sessions (In-Memory)
The project uses session-based authentication by leveraging cookies (req.cookies.sessionId) and an in-memory session store to track user and admin login states. This mechanism ensures secure access to protected routes and personalized user experiences without relying on persistent storage for session data.

Responsive Design Techniques
The website employs responsive design principles using custom CSS, Flexbox, and media queries. These techniques ensure that the layout adjusts gracefully across various screen sizes and devices, delivering a seamless and consistent user experience on desktops, tablets, and mobile phones.

Together, these technologies work in harmony to build a robust, responsive, and secure movie streaming platform. They ensure smooth user interaction, efficient data handling, and scalability for future enhancements.

Project Folder Structure:

```
iflix-project/
├── .vscode/            # VS Code editor-specific settings
├── backend/            # (Optional: Can be used for backend scripts or modularization)
├── node_modules/       # Installed npm dependencies
│
├── protected/          # Pages accessible only after login (user/admin dashboards)
│   ├── add-movie.html              # Admin Add Movie Feature
│   ├── admin-charts.html           # Admin Show Graph/charts Feature
│   ├── admin-dashboard.html            # Admin Home Page
│   ├── admin-profile.html              # Admin Profile Page
│   ├── admin-subscriptions.html        # Admin See Subscription Feature
│   ├── delete-movie.html               # Admin Delete Movie Feature
│   ├── movie-template.html             # Admin Add Movie Template Page
│   ├── view-users.html                 # Admin View User Page
│   ├── details.css                     # Movie Details Page CSS
│   ├── details1-titanic.html           # Movie Details Page
│   ├── details2-angrybirds.html        # Movie Details Page
│   ├── details3-hobbit.html            # Movie Details Page
│   ├── details4-1917.html              # Movie Details Page
│   ├── details5-3idiots.html           # Movie Details Page
│   ├── details6-joker.html             # Movie Details Page
│   ├── details7-her.html               # Movie Details Page
│   ├── details8-oblivion.html          # Movie Details Page
│   ├── details10-duedate.html          # Movie Details Page
│   ├── details11-kickass2.html         # Movie Details Page
│   ├── details12-emoji-movie.html      # Movie Details Page
│   ├── details13-wednesday.html          # Series Details Page
│   ├── details14-kotafactory.html        # Series Details Page
│   ├── details15-mirzapur.html           # Series Details Page
│   ├── filter.html                       # Movie & Series Filter Page
│   ├── index.html                        # Movie Server Home Page
│   ├── SB1-search.html                   # Movie Server Search Page
│   ├── SB3-users.html                    # Movie Server User Review & Discussion
│   ├── SB4-watch_later.html                # User Watch Later Movie Page
│   ├── SB7-subscription.html               # User Buy Subscription Page
│   ├── series.html                         # Web Series Page
│   ├── user-profile.html                   # User Profile Page
│
├── public/                                 # Publicly Accessible Static Pages
│   ├── img/                                # User Profile, Movie Pictures
│   ├── movies/                             # Movie Videos
│   ├── admin-style.css                     # Styles for Admin Pages
│   ├── app.js                              # JS for General Use
│   ├── login.css                           # Styling for Login Page
│   ├── login.html                          # Login Page
│   ├── profile.css                         # Styling for User Profile
│   ├── profile.js                          # JavaScript for Profile Page
│   ├── SB7-subscription.css                # Styling for Subscription Page
│   ├── sign-up.html                        # Signup Page
│   ├── signup.css                          # Styling for Signup
│   ├── signup.js                           # JavaScript for Signup Page
│   └── style.css                           # General/global styles
├── uploads/                        # Uploaded profile pictures (via multer)
├── .env                            # Environment variables
├── db.js                           # Contain DB connection setup
├── server.js                       # Main Backend Page
├── package.json                    # Project metadata and dependencies
└── package-lock.json               # Exact versions of installed packages
```

Main Features:

This web-based movie streaming and management application supports both regular users and administrators. It offers a complete end-to-end platform for browsing, filtering, and managing movie content with user subscriptions and profile management. Below are the core features of the system, divided by functionality:

1. Movie Server Features

These are the core features that power the frontend movie experience:

- **Dynamic Movie Listing**

The home page (index.html) dynamically showcases movie and series listings pulled from the database. Each movie is represented with a high-resolution thumbnail, title, genre, and a short description. Grouping mechanisms (e.g., Latest, Popular, Trending) provide categorized displays that enhance user discovery.

- **Detailed Movie Pages**

Individual movie pages such as details1-titanic.html or details6-joker.html contain extended descriptions, cast info, year, genre, trailer, related movie suggestions, movie play and download option. These pages are designed to mirror real movie streaming platforms.

- **Movie Filtering System**

Users can explore content based on genres (Action, Comedy, Drama, etc.), popularity, and trend categories. The filter.html page facilitates quick filtering, narrowing search results based on user-selected parameters. This improves content discoverability and reduces browsing time.

- **Search Feature**

. A functional search bar allows keyword-based searches across movie titles, genres, and tags. JavaScript is used on the client side to fetch matching results in real time from local or database-derived movie data.

- **TV & Web Series Section**

A dedicated page is provided for TV shows and web series. Users can view serialized content separately from movies. This page organizes series and offers a browsing experience for episodic content.

- **Responsive UI and Clean Design**

The interface uses custom CSS, flexbox, and media queries to ensure responsiveness across devices. Whether on mobile, tablet, or desktop, the layout adjusts fluidly with consistent visuals.

2. User Features

These features enable registered users to have a personalized experience, manage their preferences, and interact with the platform securely.

- **User Sign-up and Login**

Users can create accounts via sign-up.html and log in securely through login.html. Credentials are verified against stored records in the MySQL database, and sessions are established using cookie-based authentication.

- **User Profile Page**

Users can view and update their personal information such as name, email, gender, bio, age and movie preferences. They can also upload a profile picture using profile.js and see their current subscription and past subscription plans.

- **Subscription Plans**

Users can subscribe to plans (1-month, 3-month, 6-month, 1-year and 2-year plan). Subscription data (start date, end date, plan type) is stored in the MySQL database. Subscriptions are shown under "Current Subscription" or "Past Subscriptions" depending on their status.

- **Session-Based Security**

User sessions are managed using cookies and in-memory session tracking (req.cookies.sessionId). Unauthorized users are redirected to the login page when attempting to access protected content.

- **Watch Later Feature**

Logged-in users can click a "Watch Later" button on movies, saving them to their personal list. This list is stored in the database and accessed via SB4-watch_later.html, making it easy for users to return to content later.

- **Profile Picture Upload**

Profile pictures are uploaded via the profile page, with preview functionality handled in profile.js. Uploaded images are stored on the server, and the updated picture reflects instantly upon refresh

- **User Review & Feedback**

The platform supports a chat and review feature where logged-in users can post feedback on movies or series. Reviews are saved in the database and displayed dynamically, allowing real-time interaction and enhancing user engagement.

3. Admin Features (Administrative Control Panel)

Admins are provided with comprehensive tools for managing content, subscriptions, users, and platform analytics.

- **Admin Dashboard**

The main hub (admin-dashboard.html) presents management options through interactive cards: Add Movie, View Users, View Subscriptions, Delete Movie, Charts, and more. Each card routes to the relevant admin page.

- **Add Movies**

Admins can add new movies by filling out a form that includes details like the movie title, genre, description, movie poster, movie video file and related more information. Once submitted, the data is inserted into the database, and the new movie becomes visible to users in the movie listings.

- **Delete Movies**

Admins can remove existing movies from the server by selecting a movie from the list and deleting it. This operation deletes the corresponding record from the database, ensuring the movie is no longer shown to users.

- **View All Users**

Admins can view all registered users with their profile details in table format, if admin want, he/she can delete and user from the movie server. This allows monitoring of user activity and growth.

- **Subscription Management**

Admins can view which user is subscribed to which plan, including start and end dates. Data is fetched directly from the subscriptions table and presented in an organized table.

- **Admin Profile Management**

Admins can update their own credentials and contact details, similar to regular user profile features.

- **Data Visualization**

This page allows admins to view key metrics like user count, subscriptions, and content preferences through interactive charts. Using libraries like Chart.js, it presents data in visual formats (bar, pie, line) to help monitor trends and platform performance effectively.

- **Protected Routes for Admin Access Only**

All admin pages are stored in the /protected/ directory. Middleware logic ensures that only authenticated admins can access these routes. Unauthorized users are redirected to login or denied access.

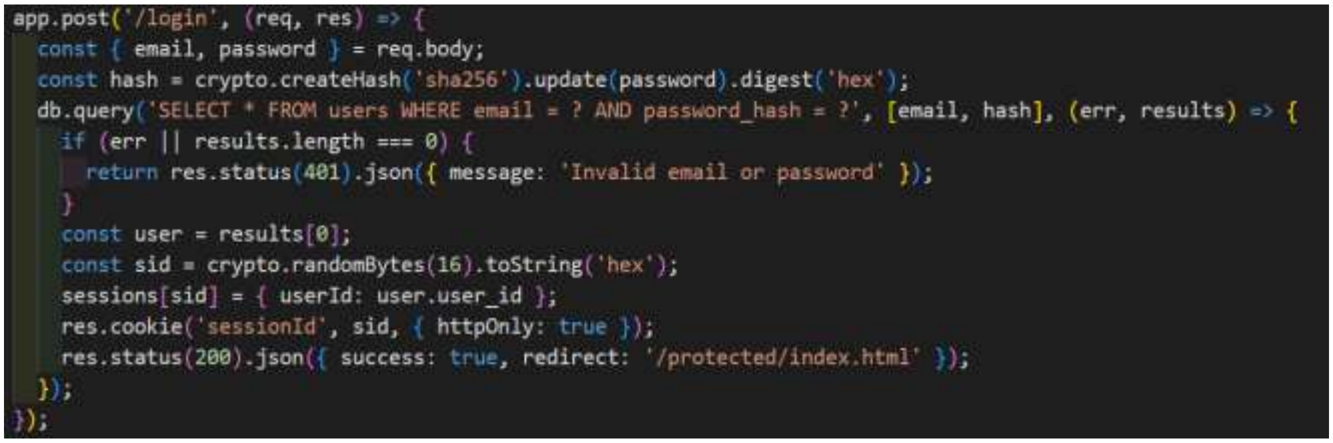
These comprehensive features make the movie streaming platform both user-friendly and powerful. Users enjoy personalized experiences with smooth navigation, while admins manage content and subscriptions efficiently. The system ensures secure, dynamic interactions backed by a robust backend. Overall, the platform delivers a complete entertainment and management solution.

Code Snippets with Explanations

1. Core Backend Logic

User Login & Session Handling (server.js)

Code:



```
app.post('/login', (req, res) => {
  const { email, password } = req.body;
  const hash = crypto.createHash('sha256').update(password).digest('hex');
  db.query('SELECT * FROM users WHERE email = ? AND password_hash = ?', [email, hash], (err, results) => {
    if (err || results.length === 0) {
      return res.status(401).json({ message: 'Invalid email or password' });
    }
    const user = results[0];
    const sid = crypto.randomBytes(16).toString('hex');
    sessions[sid] = { userId: user.user_id };
    res.cookie('sessionId', sid, { httpOnly: true });
    res.status(200).json({ success: true, redirect: '/protected/index.html' });
  });
});
```

Explanation:

1. Request Parsing: The POST route /login extracts email and password from the incoming JSON request body using req.body.

2. Password Hashing: The input password is securely hashed using the SHA-256 algorithm via Node.js's crypto module to match it with the hashed password stored in the database.

3. Database Validation: A SQL query is run to find a matching user record by email and hashed password. If no match is found or an error occurs, it returns a 401 Unauthorized response with an error message.

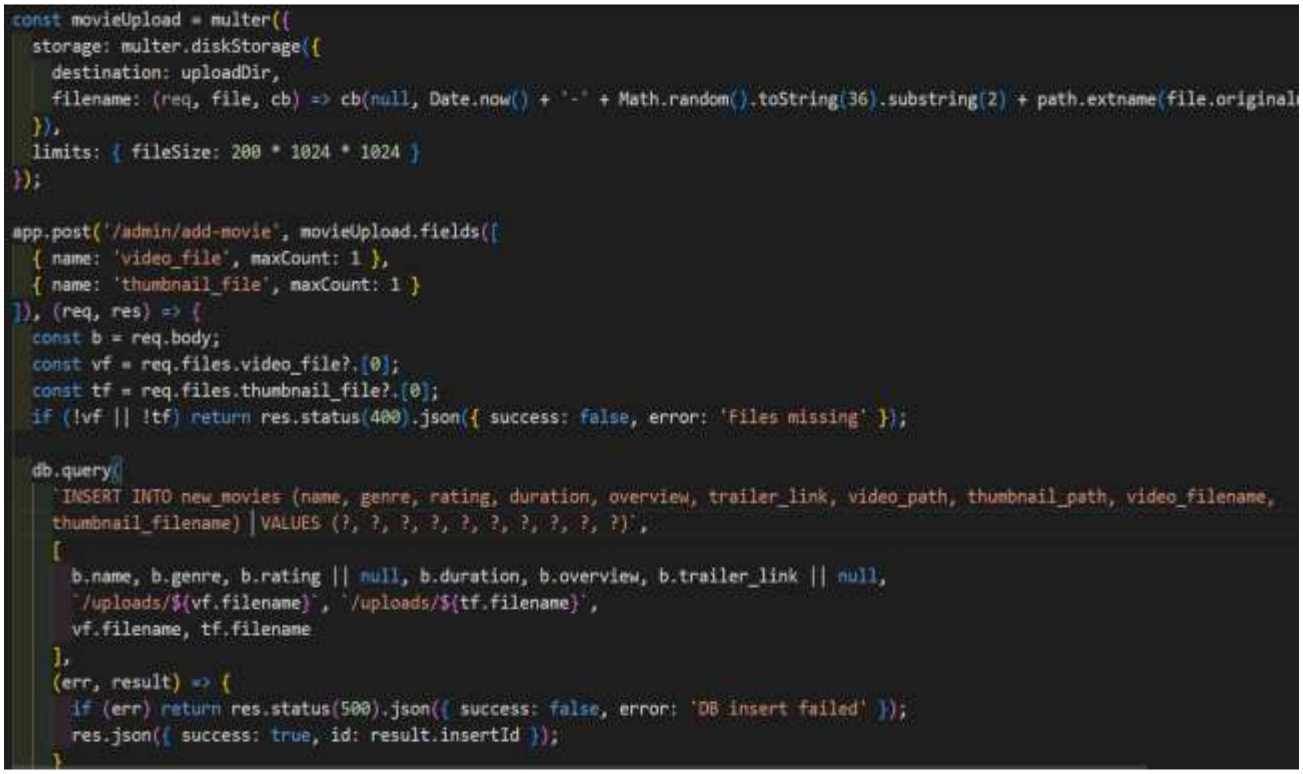
4. Session Creation: When login is successful, a unique session ID (sid) is generated using crypto.randomBytes. This session ID is stored in an in-memory sessions object along with the user's ID (user.user_id), allowing the server to recognize the user in future requests.

5. Cookie Setup: The session ID is sent to the client and saved in a cookie named sessionId, marked httpOnly so it can't be accessed from client-side scripts.

6. Successful Response: The user is redirected to the protected home page (/protected/index.html) upon successful login.

Admin – Add Movie Functionality (server.js)

Code:



```
const movieUpload = multer({
  storage: multer.diskStorage({
    destination: uploadDir,
    filename: (req, file, cb) => cb(null, Date.now() + '-' + Math.random().toString(36).substring(2) + path.extname(file.originall
  }),
  limits: { fileSize: 200 * 1024 * 1024 }
});

app.post('/admin/add-movie', movieUpload.fields([
  { name: 'video_file', maxCount: 1 },
  { name: 'thumbnail_file', maxCount: 1 }
]), (req, res) => {
  const b = req.body;
  const vf = req.files.video_file?.[0];
  const tf = req.files.thumbnail_file?.[0];
  if (!vf || !tf) return res.status(400).json({ success: false, error: 'Files missing' });

  db.query(
    `INSERT INTO new_movies (name, genre, rating, duration, overview, trailer_link, video_path, thumbnail_path, video_filename,
    thumbnail_filename) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
    [
      b.name, b.genre, b.rating || null, b.duration, b.overview, b.trailer_link || null,
      `/uploads/${vf.filename}`, `/uploads/${tf.filename}`,
      vf.filename, tf.filename
    ],
    (err, result) => {
      if (err) return res.status(500).json({ success: false, error: 'DB insert failed' });
      res.json({ success: true, id: result.insertId });
    }
```

Explanation:

1. File Upload Setup (multer): The multer middleware is configured to store uploaded video and thumbnail files in a specified directory (uploadDir). Filenames are randomized using Date.now() and a unique string to avoid overwriting.

2. Request Handling: This route handles POST requests sent to /admin/add-movie when the admin submits a form to upload a movie. It expects two files: video_file and thumbnail_file.

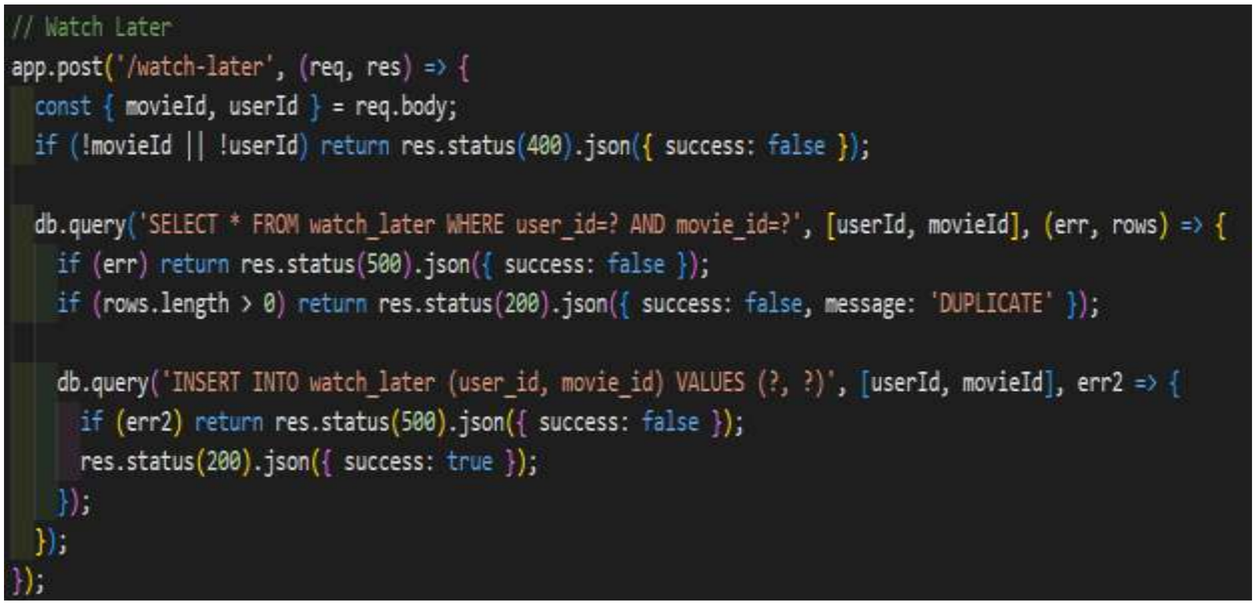
3. Validation: If either the video or thumbnail is missing, it returns a 400 error response indicating required files were not uploaded.

4. Database Insertion: Movie details from the form (e.g., name, genre, duration) are inserted into the new_movies table. The uploaded file paths (/uploads/...) and actual file names are also saved in the database for future access and streaming.

5. Success Response: If the database insertion is successful, the server responds with a success message and the ID of the newly inserted movie.

Watch Later Feature (server.js)

Code:



```
// Watch Later
app.post('/watch-later', (req, res) => {
  const { movieId, userId } = req.body;
  if (!movieId || !userId) return res.status(400).json({ success: false });

  db.query('SELECT * FROM watch_later WHERE user_id=? AND movie_id=?', [userId, movieId], (err, rows) => {
    if (err) return res.status(500).json({ success: false });
    if (rows.length > 0) return res.status(200).json({ success: false, message: 'DUPLICATE' });

    db.query('INSERT INTO watch_later (user_id, movie_id) VALUES (?, ?)', [userId, movieId], err2 => {
      if (err2) return res.status(500).json({ success: false });
      res.status(200).json({ success: true });
    });
  });
});
```

Explanation:

1. Route Purpose: This route handles POST requests to save a movie to a user's Watch Later list. It allows users to bookmark movies they want to watch in the future.

2. Extracting Data: The route receives movieId and userId from the request body. If either value is missing, it sends a 400 Bad Request response.

3. Duplicate Check: It checks the watch_later table in the MySQL database to see if the user has already added the same movie: If a matching row exists, it returns a 200 OK with a message indicating it's a duplicate, and no new entry is added.

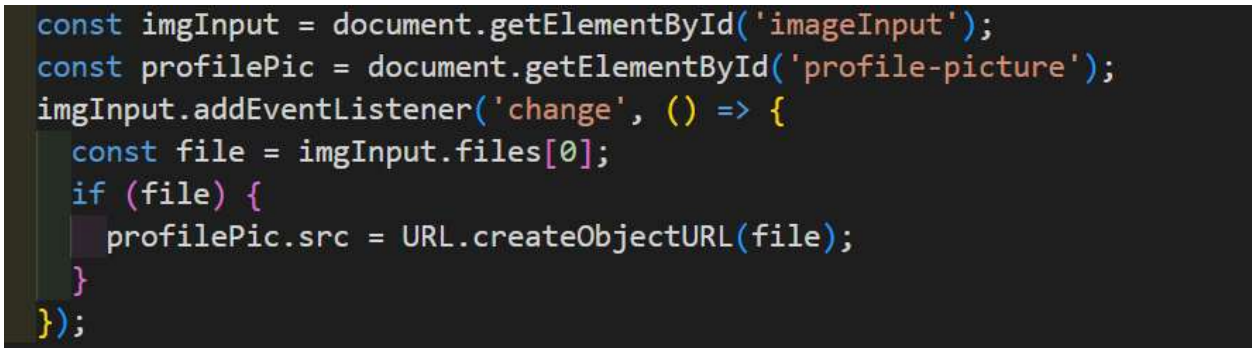
4. Database Insert: If there is no duplicate, it inserts a new record with the user_id and movie_id into the watch_later table.

5. Response: If insertion is successful: responds with { success: true }. If a database error occurs: responds with a 500 Internal Server Error.

2. Dynamic Frontend Behavior

Profile Picture Upload & Preview (server.js)

Code:

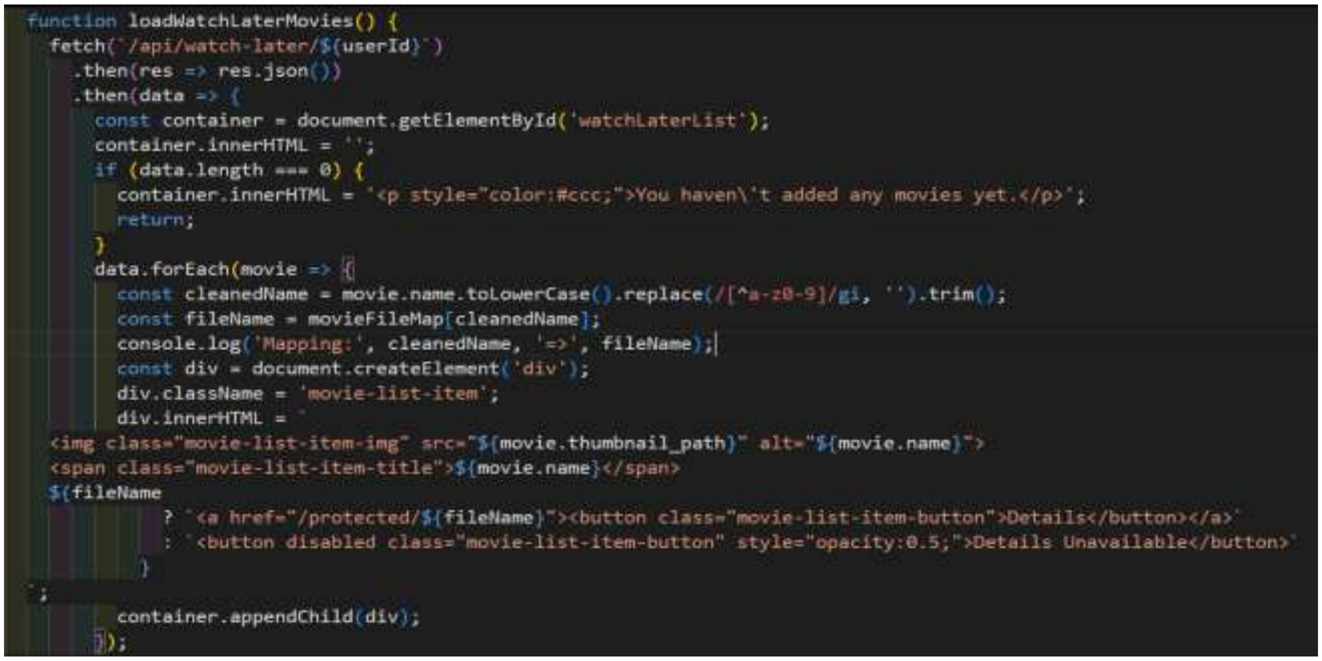


```javascript
const imgInput = document.getElementById('imageInput');
const profilePic = document.getElementById('profile-picture');
imgInput.addEventListener('change', () => {
  const file = imgInput.files[0];
  if (file) {
    profilePic.src = URL.createObjectURL(file);
  }
});
```

Explanation: This feature allows users to upload and preview their profile pictures. When a user selects an image, it's instantly displayed using URL.createObjectURL() for live feedback. The image is submitted with the profile form and uploaded to the server using Multer, which stores it in the uploads/ folder. The image path is saved in the database and later shown in the user's profile page. This improves personalization and user experience.

User Watch Later Movie

Code:

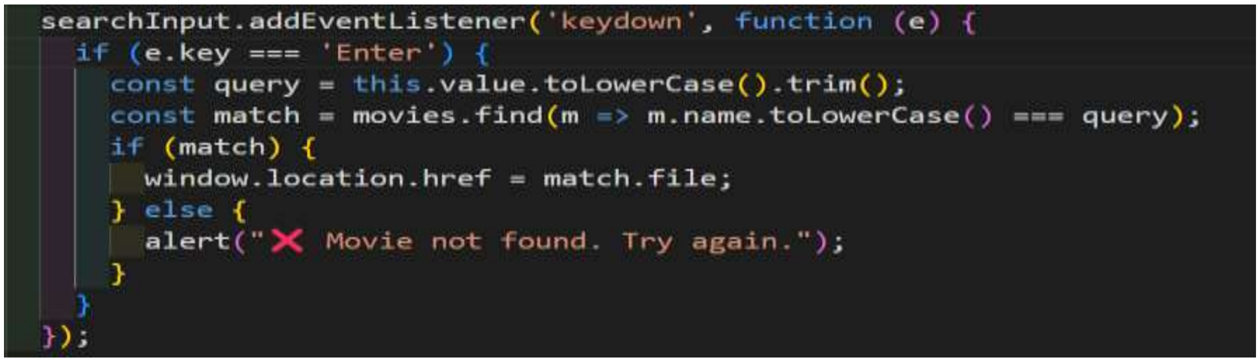


```javascript
function loadWatchLaterMovies() {
  fetch(`/api/watch-later/${userId}`)
    .then(res => res.json())
    .then(data => {
      const container = document.getElementById('watchLaterList');
      container.innerHTML = '';
      if (data.length === 0) {
        container.innerHTML = '<p style="color:#ccc;">You haven\'t added any movies yet.</p>';
        return;
      }
      data.forEach(movie => {
        const cleanedName = movie.name.toLowerCase().replace(/[^a-z0-9]/gi, '').trim();
        const fileName = movieFileMap[cleanedName];
        console.log('Mapping:', cleanedName, '=>', fileName);
        const div = document.createElement('div');
        div.className = 'movie-list-item';
        div.innerHTML = `
  <img class="movie-list-item-img" src="${movie.thumbnail_path}" alt="${movie.name}">
  <span class="movie-list-item-title">${movie.name}</span>
  ${fileName
          ? `<a href="/protected/${fileName}"><button class="movie-list-item-button">Details</button></a>`
          : `<button disabled class="movie-list-item-button" style="opacity:0.5;">Details Unavailable</button>`
        }
`;
        container.appendChild(div);
      });
```

Explanation: This function dynamically loads and displays the list of movies the user has marked as "Watch Later." It fetches movie data from the backend using the user's ID, then creates and appends movie cards to the page. Each card includes a thumbnail, title, and a "Details" button. If a valid detail page exists, the button links to it; otherwise, the button is disabled.

Search Movie Functionality

Code:

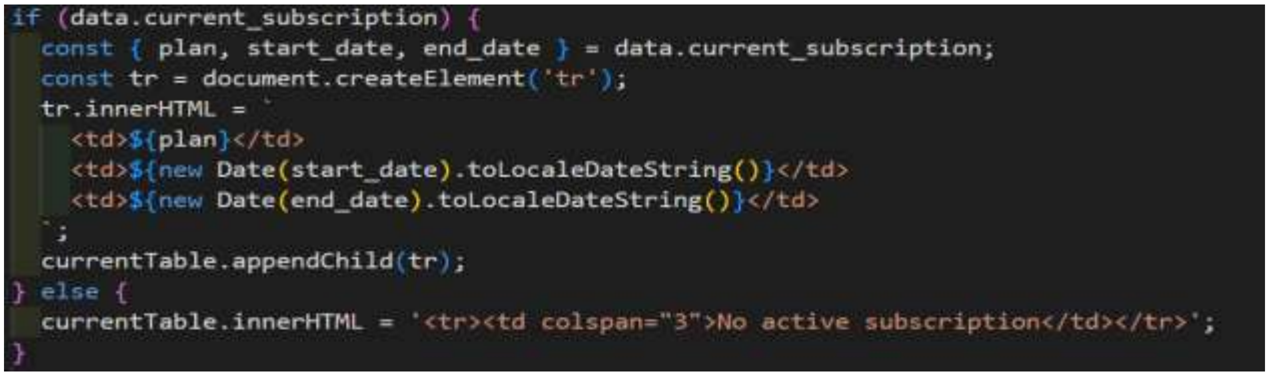


```
searchInput.addEventListener('keydown', function (e) {
  if (e.key === 'Enter') {
    const query = this.value.toLowerCase().trim();
    const match = movies.find(m => m.name.toLowerCase() === query);
    if (match) {
      window.location.href = match.file;
    } else {
      alert("❌ Movie not found. Try again.");
    }
  }
});
```

Explanation: This code listens for the Enter key event in the search input field. When pressed, it takes the user's input, converts it to lowercase, and searches the movies array for a match by comparing movie names. If a matching movie is found, it redirects to the corresponding detail page (match.file). If not found, it shows an alert saying “Movie not found.” This provides a simple client-side search feature for direct movie access.

Dynamic Subscription Display

Code:

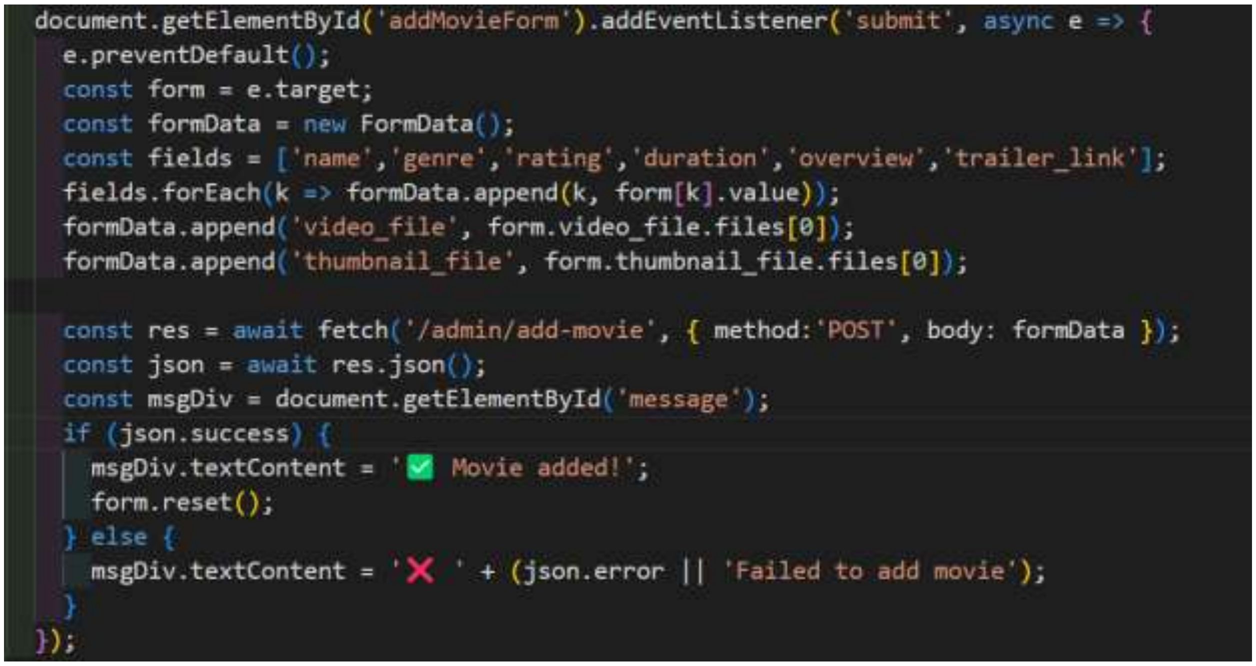


```
if (data.current_subscription) {
  const { plan, start_date, end_date } = data.current_subscription;
  const tr = document.createElement('tr');
  tr.innerHTML = `
    <td>${plan}</td>
    <td>${new Date(start_date).toLocaleDateString()}</td>
    <td>${new Date(end_date).toLocaleDateString()}</td>
  `;
  currentTable.appendChild(tr);
} else {
  currentTable.innerHTML = '<tr><td colspan="3">No active subscription</td></tr>';
}
```

Explanation: This JavaScript snippet dynamically displays a user's current subscription on their profile page: It checks if data.current_subscription exists. If present, it extracts the plan name, start date, and end date, formats the dates using toLocaleDateString(), and creates a new table row (<tr>) to show these details. If no subscription is found, it shows a message: "No active subscription". This ensures the subscription section updates in real-time based on actual database records, providing a personalized and up-to-date experience for the user.

Admin Dynamically Add New Movie

Code:



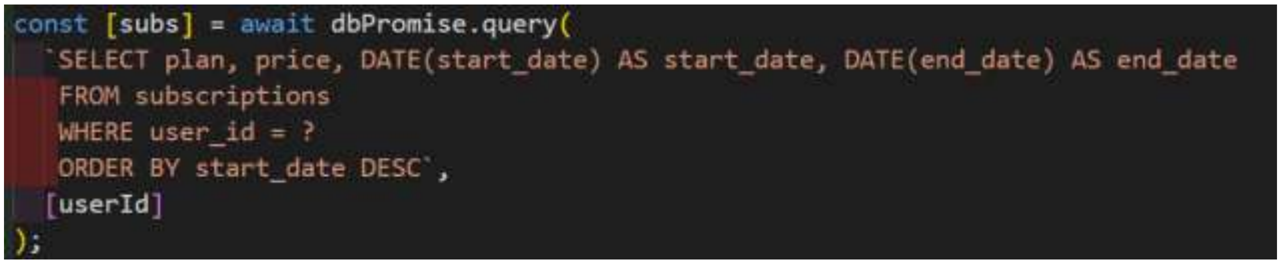
```javascript
document.getElementById('addMovieForm').addEventListener('submit', async e => {
  e.preventDefault();
  const form = e.target;
  const formData = new FormData();
  const fields = ['name','genre','rating','duration','overview','trailer_link'];
  fields.forEach(k => formData.append(k, form[k].value));
  formData.append('video_file', form.video_file.files[0]);
  formData.append('thumbnail_file', form.thumbnail_file.files[0]);

  const res = await fetch('/admin/add-movie', { method:'POST', body: formData });
  const json = await res.json();
  const msgDiv = document.getElementById('message');
  if (json.success) {
    msgDiv.textContent = '✅ Movie added!';
    form.reset();
  } else {
    msgDiv.textContent = '❌ ' + (json.error || 'Failed to add movie');
  }
});
```

Explanation: This script enables admin users to upload and add new movies dynamically via a form submission. Here's how it works: It listens for the submit event on the form with ID addMovieForm. It collects form inputs such as movie name, genre, rating, duration, overview, and trailer link. It also uploads two files: a video file and a thumbnail image using the FormData API. The form data is sent via a POST request to /admin/add-movie, which stores the info in the database. If the operation is successful, a success message is displayed and the form is reset. Otherwise, an error message appears.

3. Database Queries

Subscription Fetching Query

Code:

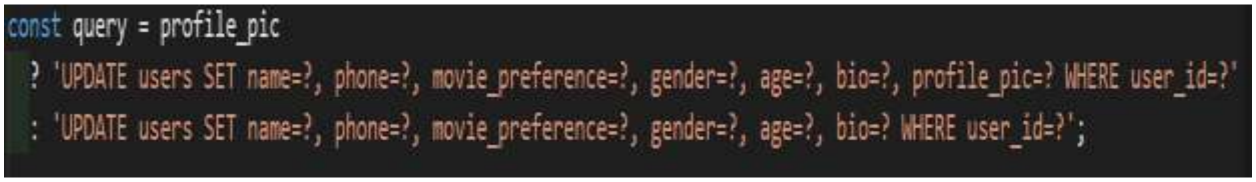


```
const [subs] = await dbPromise.query(
  `SELECT plan, price, DATE(start_date) AS start_date, DATE(end_date) AS end_date
   FROM subscriptions
   WHERE user_id = ?
   ORDER BY start_date DESC`,
  [userId]
);
```

Explanation: This SQL query retrieves a user's subscription history, including the plan name, price, and formatted start/end dates (without time). It filters by user_id and orders the results by most recent subscriptions. The DATE() function ensures clean date display, making it easier to show current and past plans on the user profile.

Update Profile Pic Query

Code:

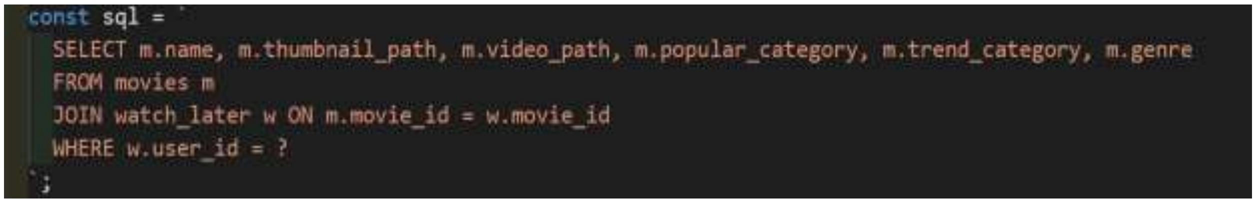


```
const query = profile_pic
  ? 'UPDATE users SET name=?, phone=?, movie_preference=?, gender=?, age=?, bio=?, profile_pic=? WHERE user_id=?'
  : 'UPDATE users SET name=?, phone=?, movie_preference=?, gender=?, age=?, bio=? WHERE user_id=?';
```

Explanation: This code updates a user's profile in the database. If a new profile picture is uploaded (profile_pic exists), it includes that in the UPDATE query. Otherwise, it updates only the text fields like name, phone, gender, etc., without changing the profile picture.

User Watch Later Query

Code:

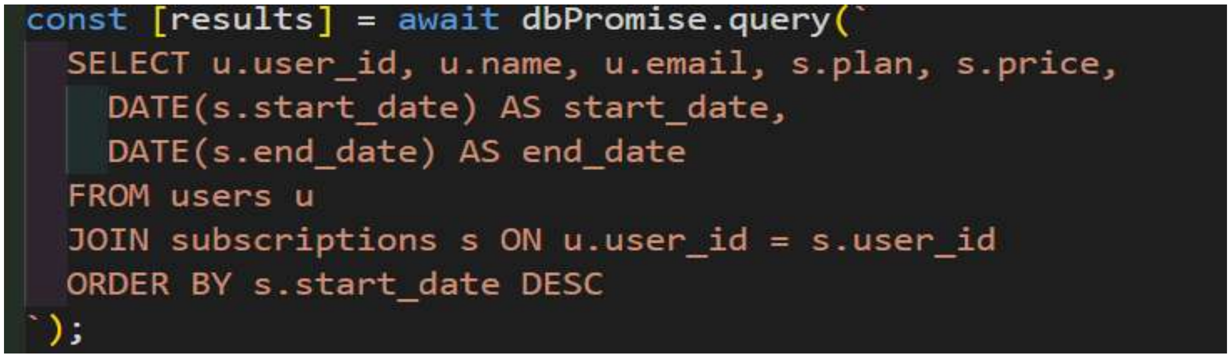


```
const sql = `
  SELECT m.name, m.thumbnail_path, m.video_path, m.popular_category, m.trend_category, m.genre
  FROM movies m
  JOIN watch_later w ON m.movie_id = w.movie_id
  WHERE w.user_id = ?
`;
```

Explanation: This route fetches all "Watch Later" movies for a user by joining the movies and watch_later tables using the userId. It returns movie details like name and thumbnail to show the user's saved list.

Admin View All Users And Their Subscriptions Query

Code:

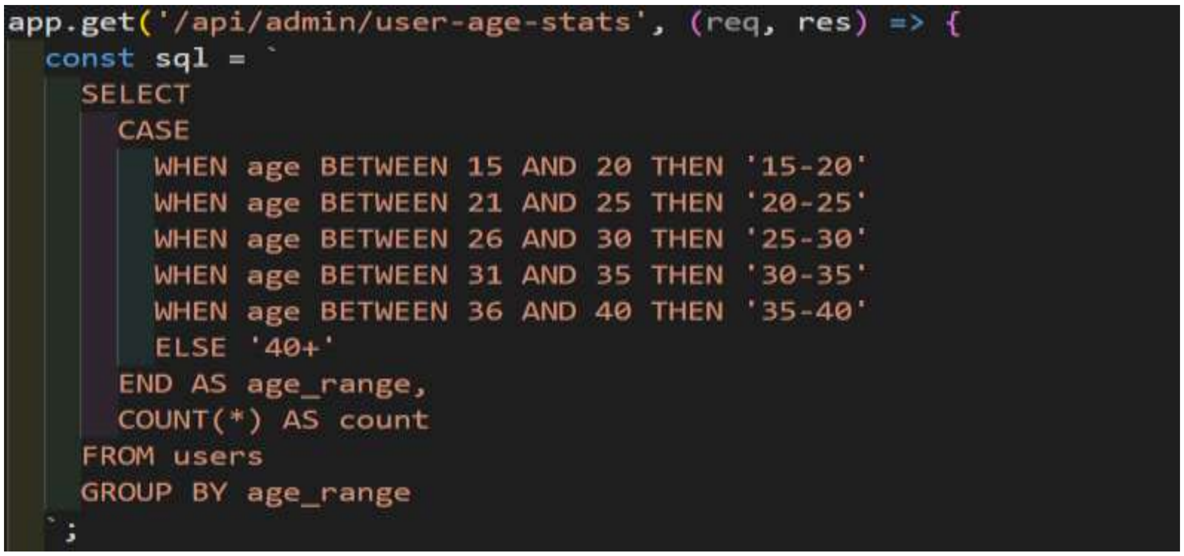


```
const [results] = await dbPromise.query(`
  SELECT u.user_id, u.name, u.email, s.plan, s.price,
    DATE(s.start_date) AS start_date,
    DATE(s.end_date) AS end_date
  FROM users u
  JOIN subscriptions s ON u.user_id = s.user_id
  ORDER BY s.start_date DESC
`);
```

Explanation: This query retrieves a list of users along with their subscription details by joining the users and subscriptions tables. It shows each user's name, email, plan, price, and the subscription's start and end dates, sorted by the latest subscriptions first.

Admin Overview Graph Query

Code:

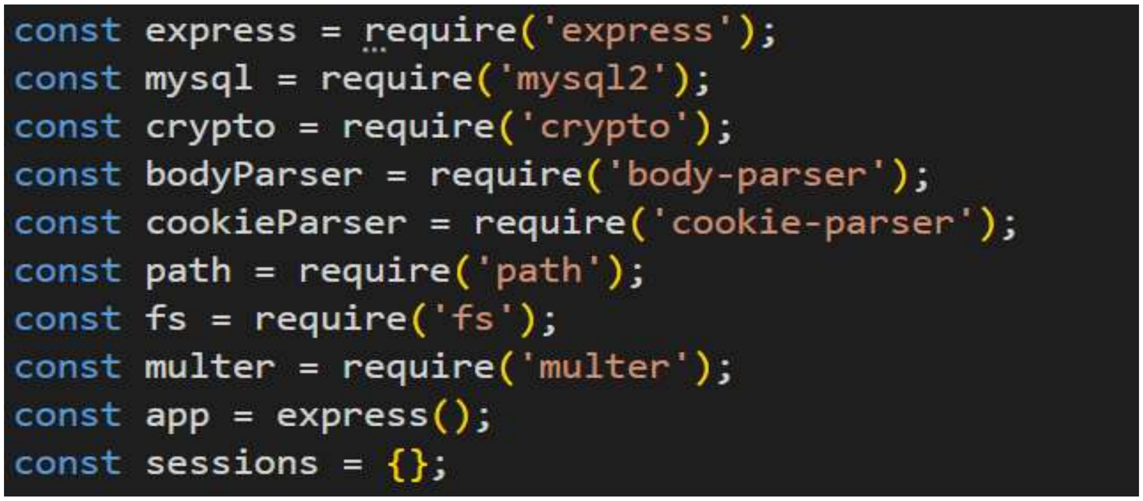


```
app.get('/api/admin/user-age-stats', (req, res) => {
  const sql = `
    SELECT
      CASE
        WHEN age BETWEEN 15 AND 20 THEN '15-20'
        WHEN age BETWEEN 21 AND 25 THEN '20-25'
        WHEN age BETWEEN 26 AND 30 THEN '25-30'
        WHEN age BETWEEN 31 AND 35 THEN '30-35'
        WHEN age BETWEEN 36 AND 40 THEN '35-40'
        ELSE '40+'
      END AS age_range,
      COUNT(*) AS count
    FROM users
    GROUP BY age_range
  `;
```

Explanation: This SQL query groups users into age ranges (e.g., 15–20, 21–25, etc.) and counts how many users fall into each range. The result helps the admin visualize age distribution of users on a graph (like a bar chart or pie chart) in the dashboard.

4. Server Initialization & Module Imports

Code:



```
const express = require('express');
const mysql = require('mysql2');
const crypto = require('crypto');
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const path = require('path');
const fs = require('fs');
const multer = require('multer');
const app = express();
const sessions = {};
```

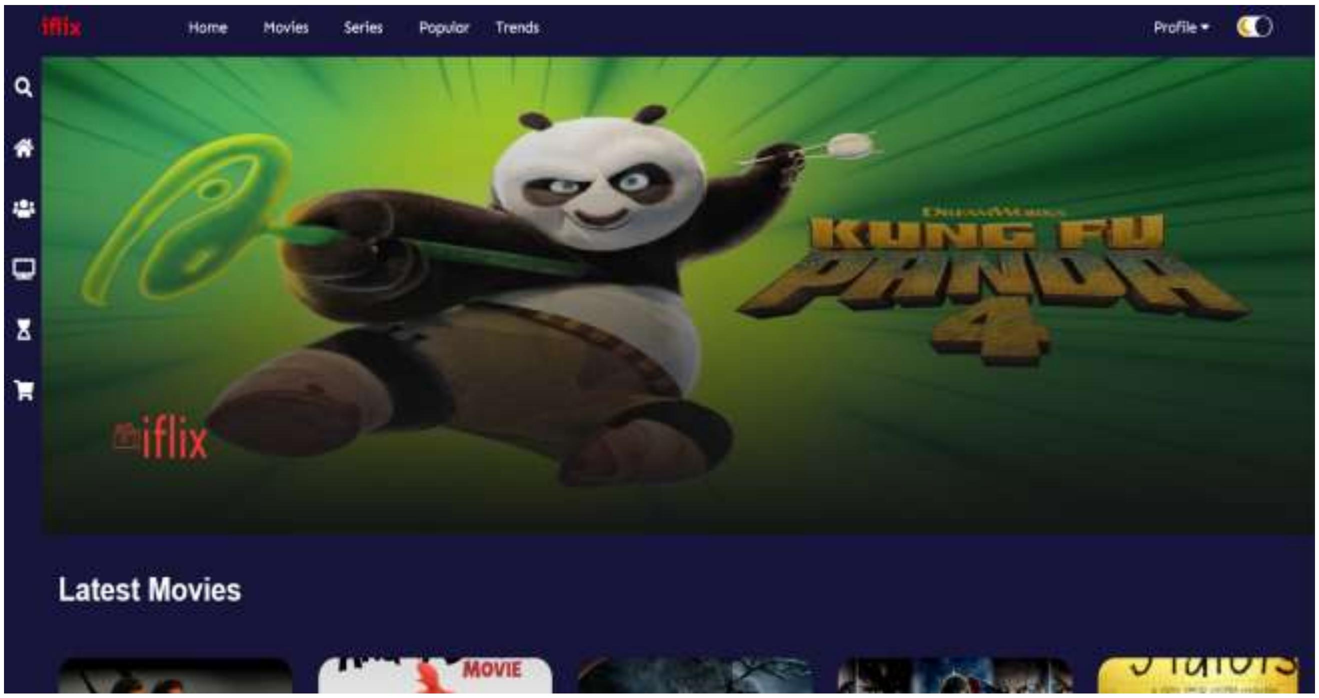
Explanation: This section sets up the essential tools and libraries required to run the movie server. It includes:

- **express** – The core web framework used to create the backend server and handle HTTP routes.
- **mysql2** – A MySQL client that enables database interaction, such as storing and retrieving user, movie, and subscription data.
- **crypto** – Provides cryptographic functionality (e.g., secure IDs or hashes).
- **body-parser** – Parses incoming JSON or form data from requests for easier handling.
- **cookie-parser** – Parses cookies in HTTP requests, allowing for session management.
- **path & fs** – Used for file and directory operations like saving movie files or reading static assets.
- **multer** – Middleware for handling multipart/form-data, mainly used for file uploads (e.g., thumbnails, videos, profile pictures).
- **sessions** – An in-memory object used to track user/admin sessions manually across different requests.

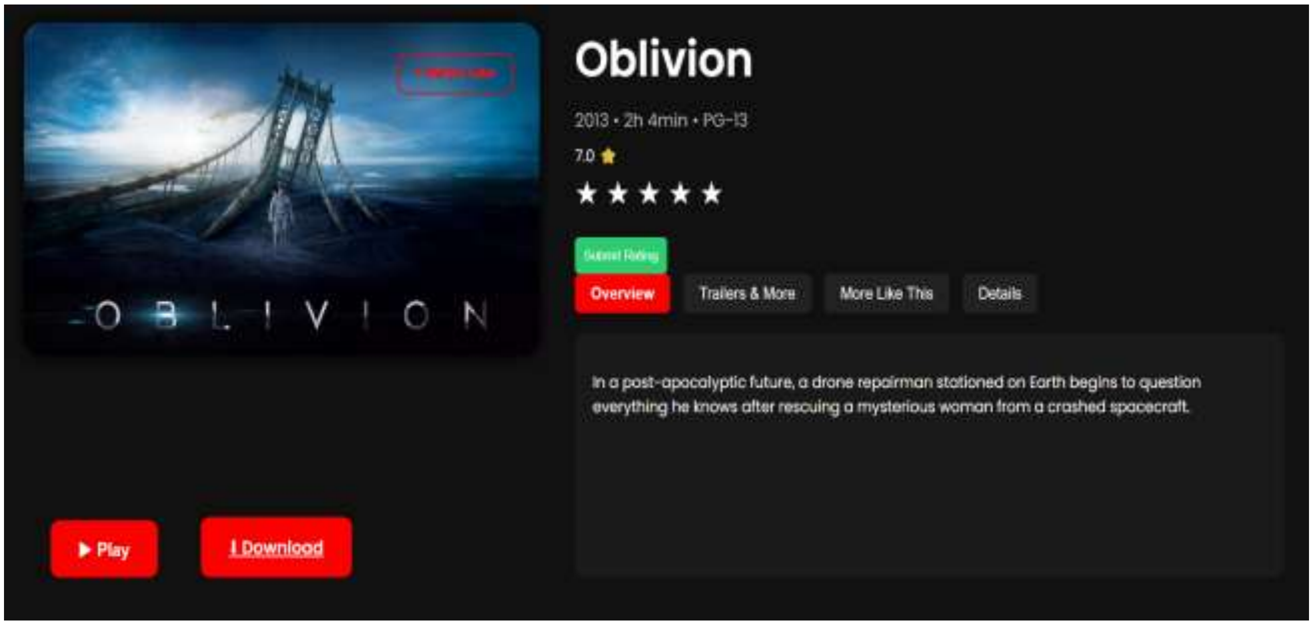
This setup is the foundation of the entire backend logic and ensures all essential tools are available before defining routes.

Screenshots of Web Pages:

Movie Server Home Page:

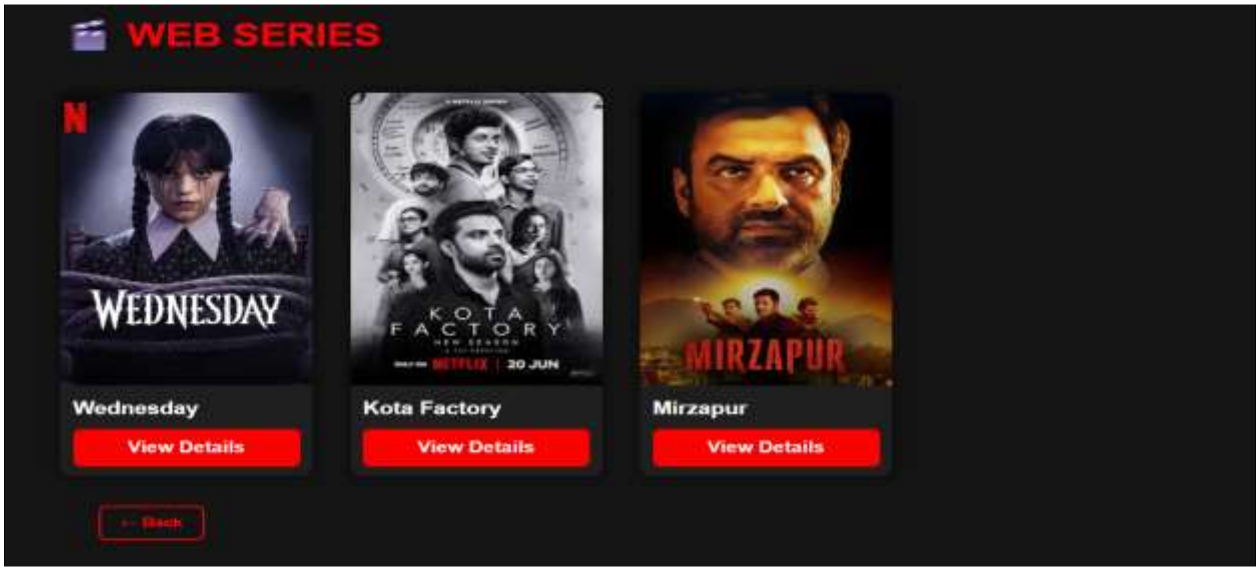


Movie Details Page:

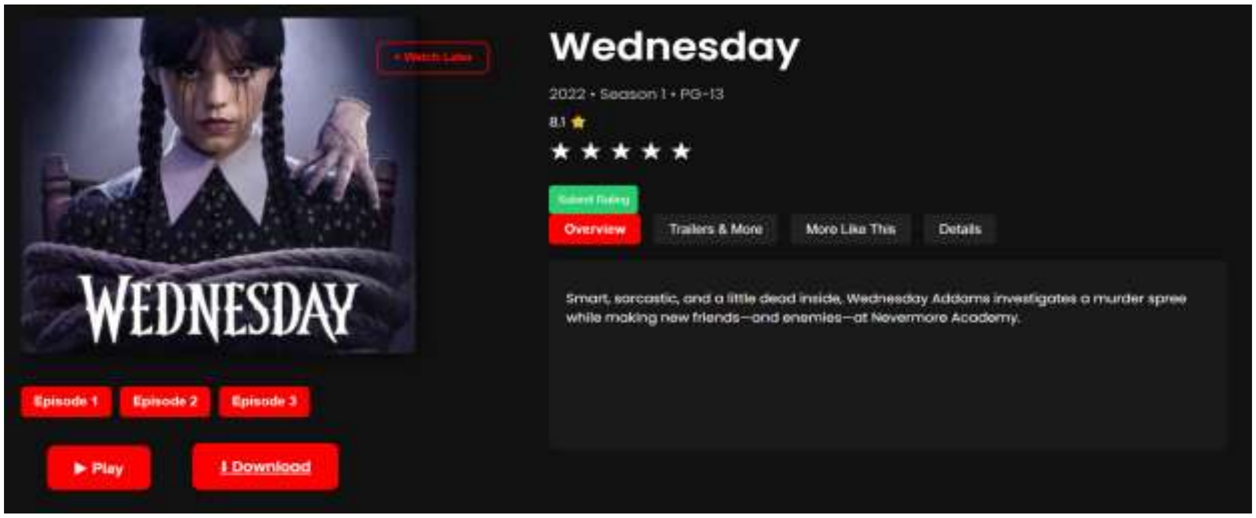


In a post-apocalyptic future, a drone repairman stationed on Earth begins to question everything he knows after rescuing a mysterious woman from a crashed spacecraft.

Series Home Page:



Series Details Page:



Wednesday

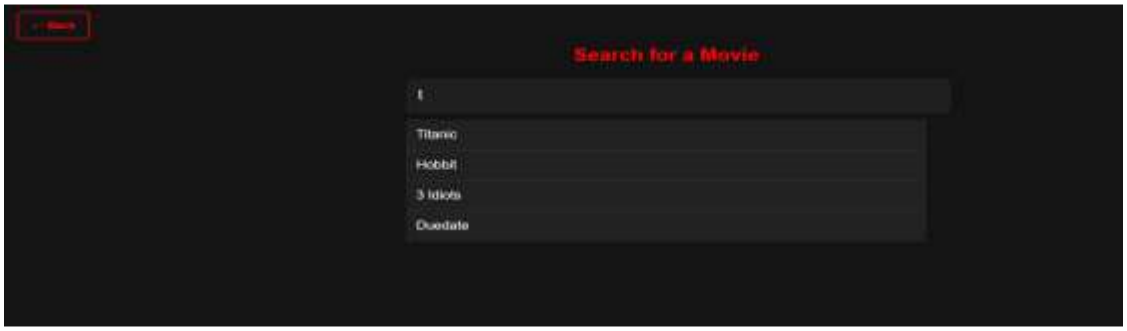
2022 • Season 1 • PG-13

8.1 ⭐

★ ★ ★ ★ ★

Smart, sarcastic, and a little dead inside, Wednesday Addams investigates a murder spree while making new friends—and enemies—at Nevermore Academy.

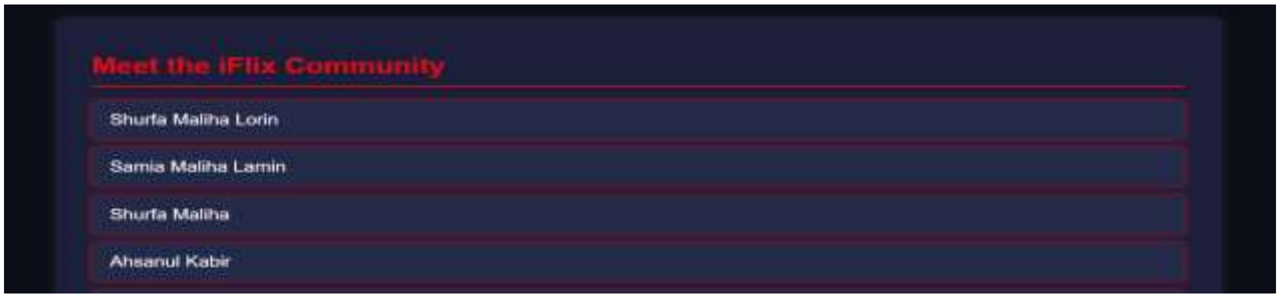
Search Movie Page:



t

Titanic
Hobbit
3 Idiots
Duedate

User Community & Review, Discussion Page:



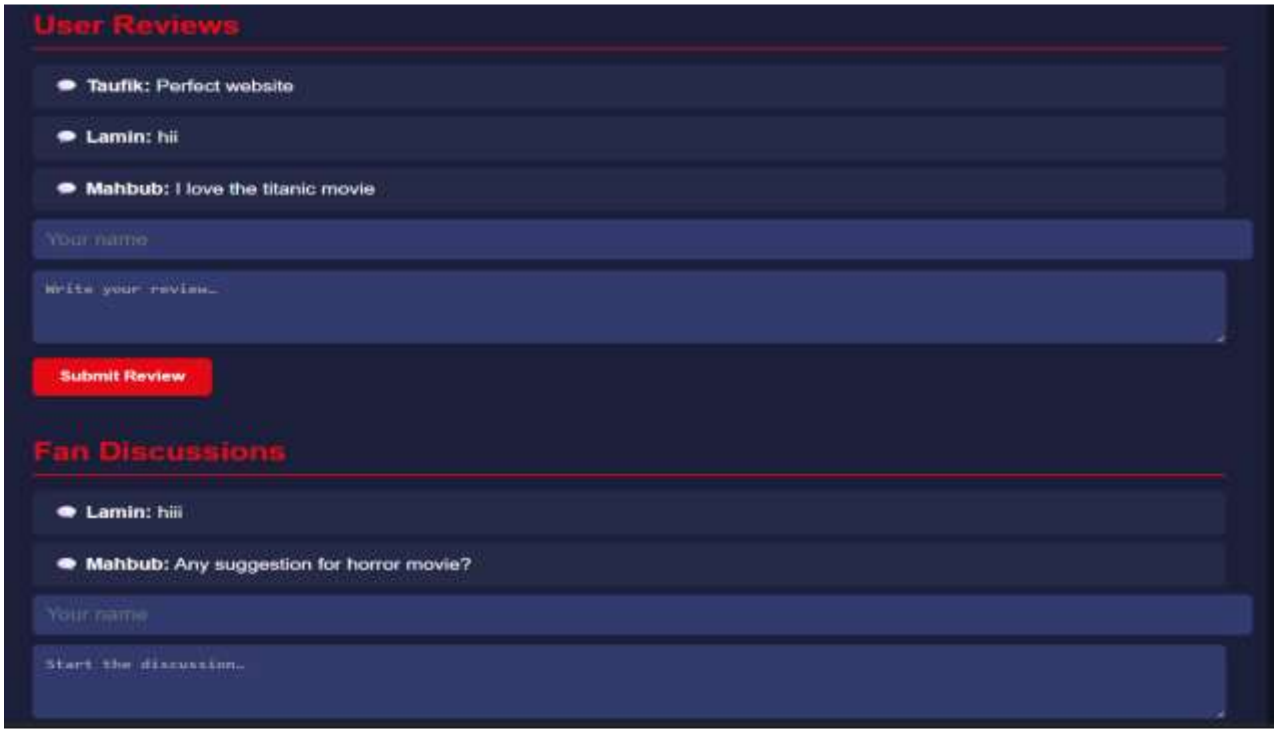
Meet the iFlix Community

Shurfa Maliha Lorin

Samia Maliha Lamin

Shurfa Maliha

Ahsanul Kabir



User Reviews

Taufik: Perfect website

Lamin: hii

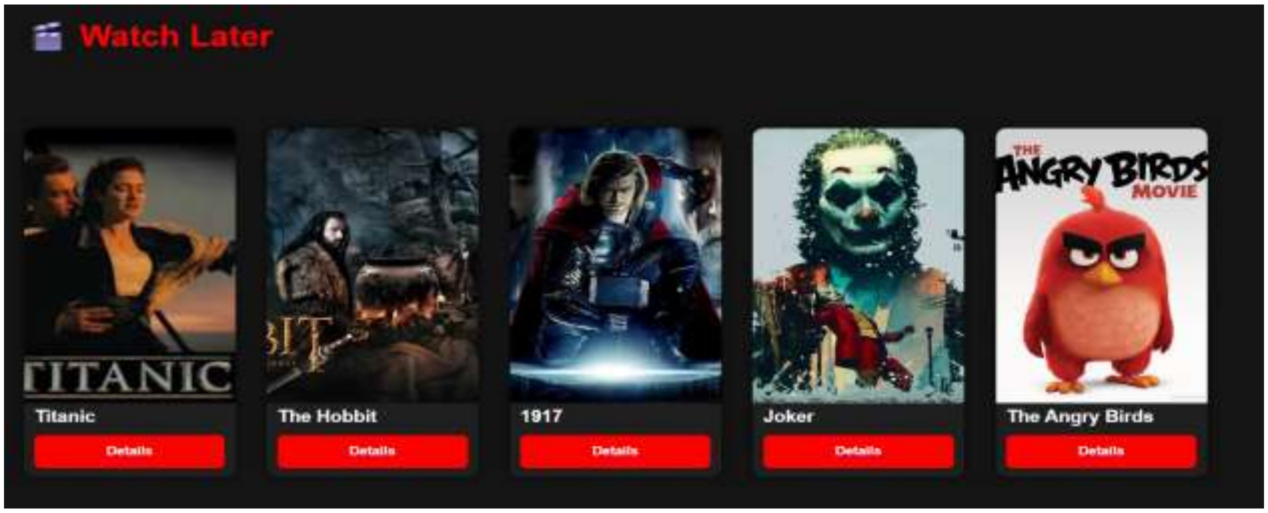
Mahbub: I love the titanic movie

Fan Discussions

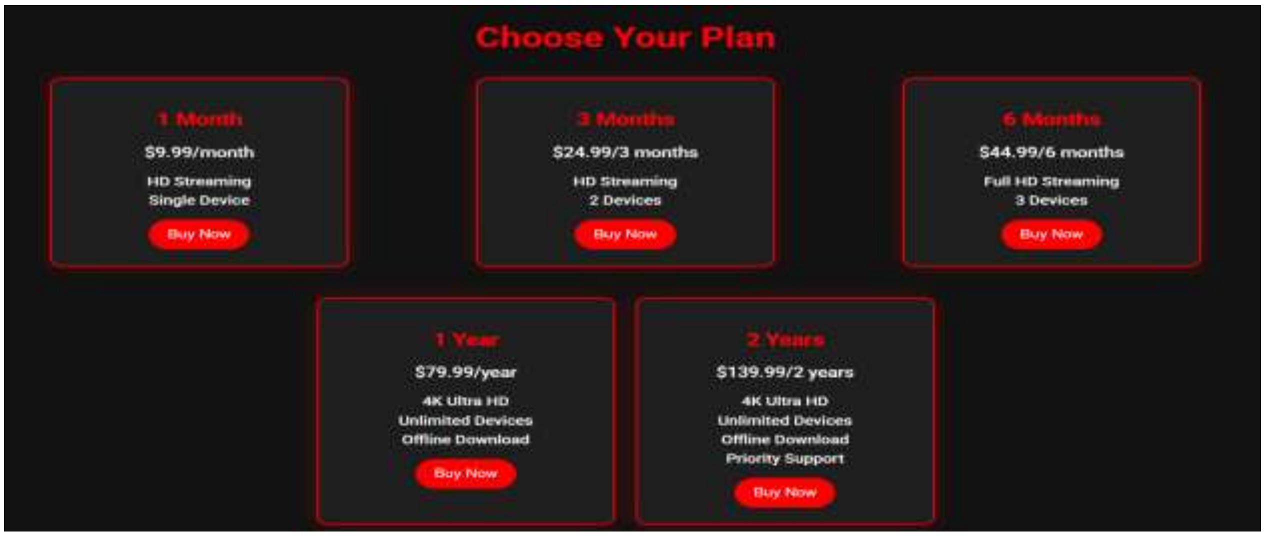
Lamin: hiii

Mahbub: Any suggestion for horror movie?

Watch Later Movie Page:



Buy Subscription Page:



Choose Your Plan

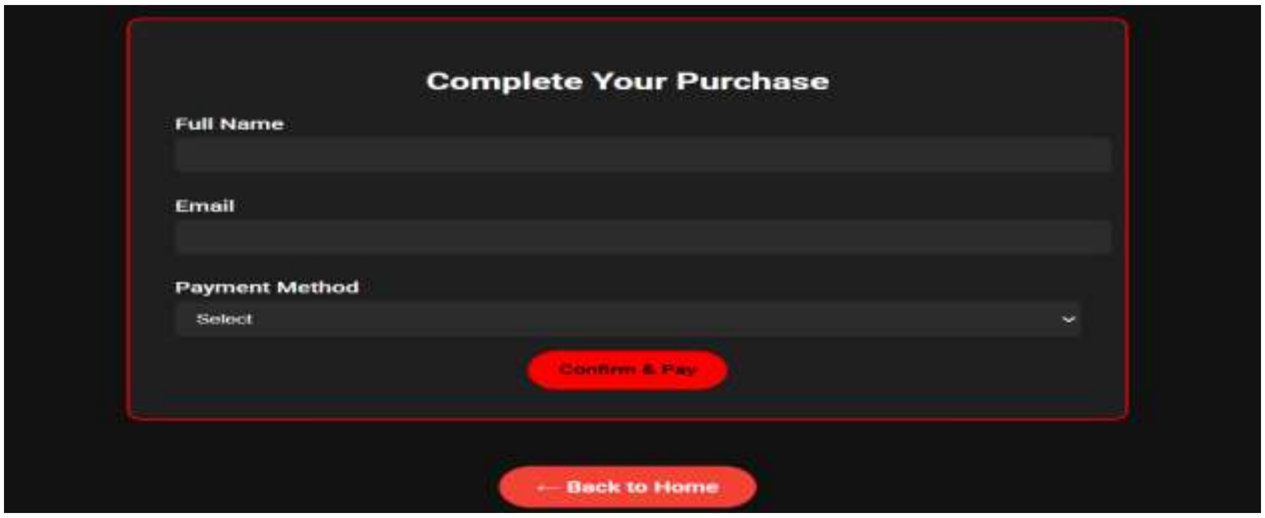
1 Month
$9.99/month
HD Streaming
Single Device
Buy Now

3 Months
$24.99/3 months
HD Streaming
2 Devices
Buy Now

6 Months
$44.99/6 months
Full HD Streaming
3 Devices
Buy Now

1 Year
$79.99/year
4K Ultra HD
Unlimited Devices
Offline Download
Buy Now

2 Years
$139.99/2 years
4K Ultra HD
Unlimited Devices
Offline Download
Priority Support
Buy Now



Complete Your Purchase

Full Name

Email

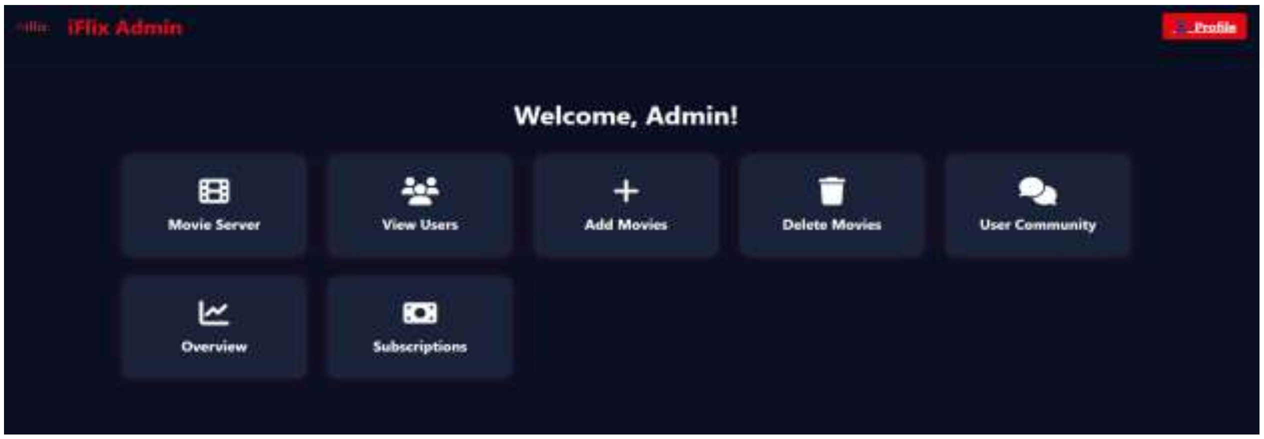
Payment Method

Select

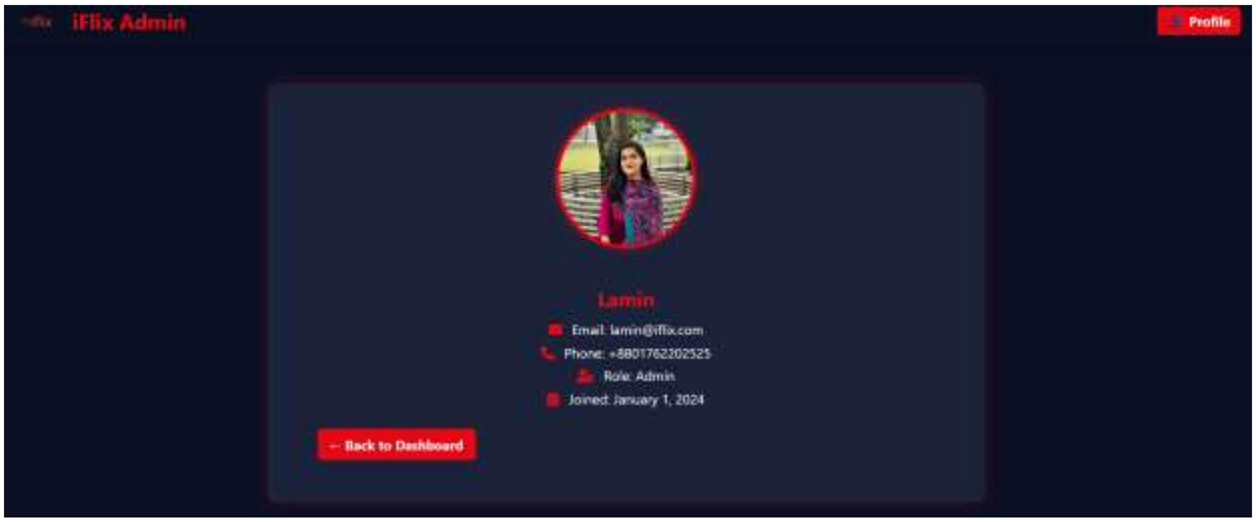
Confirm & Pay

← Back to Home

Admin Dashboard Page:



Admin Profile Page:



Lamin

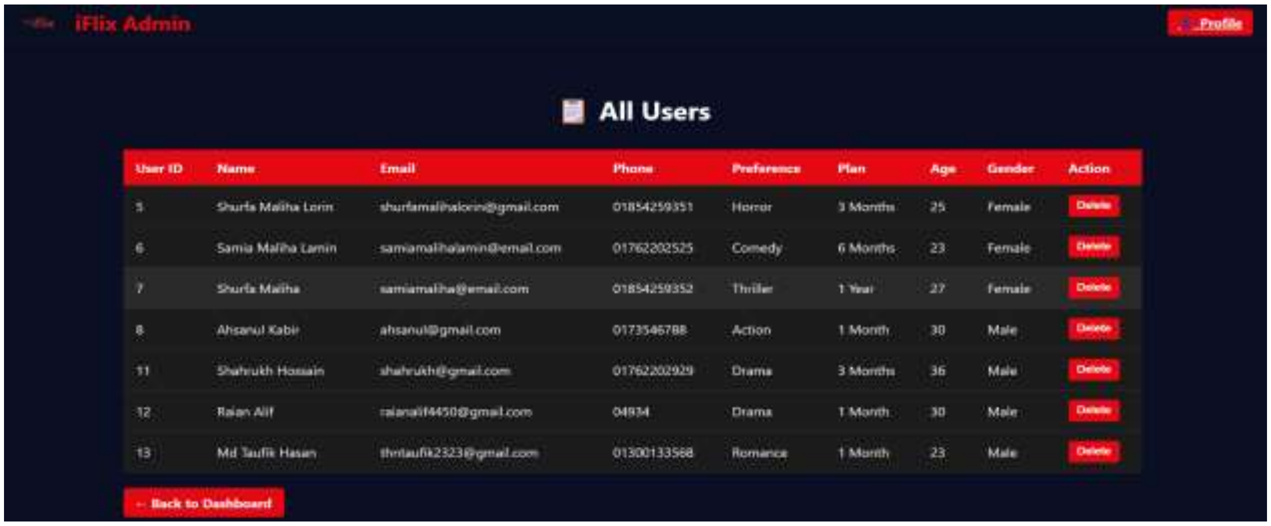
Email: lamin@iflix.com

Phone: +8801762202525

Role: Admin

Joined: January 1, 2024

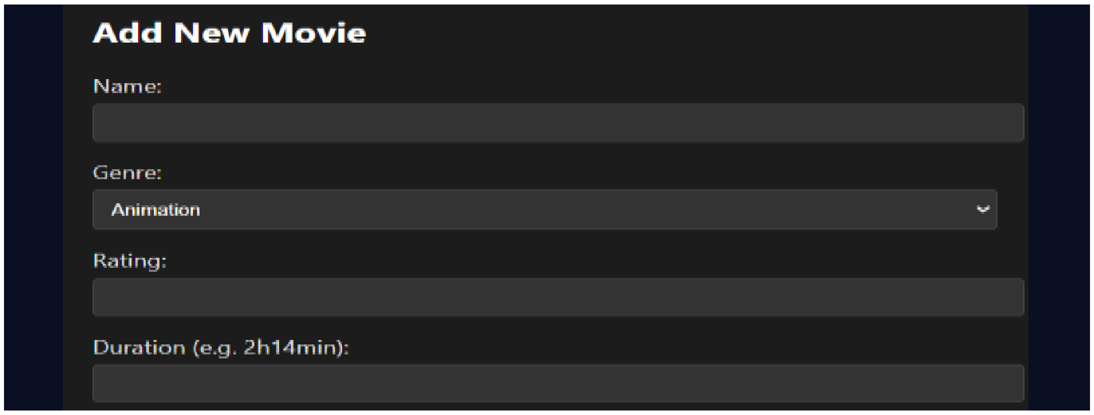
Admin View Users Page:



All Users

User ID	Name	Email	Phone	Preference	Plan	Age	Gender	Action
5	Shurfa Maliha Lorin	shurfamalihalorin@gmail.com	01854259351	Horror	3 Months	25	Female	Delete
6	Samia Maliha Lamin	samiamalihalamin@email.com	01762202525	Comedy	6 Months	23	Female	Delete
7	Shurfa Maliha	samiamaliha@email.com	01854259352	Thriller	1 Year	27	Female	Delete
8	Ahsanul Kabir	ahsanul@gmail.com	0173546788	Action	1 Month	30	Male	Delete
11	Shahrukh Hossain	shahrukh@gmail.com	01762202929	Drama	3 Months	36	Male	Delete
12	Raian Alif	raianalif4450@gmail.com	04934	Drama	1 Month	30	Male	Delete
13	Md Taufik Hasan	thmtaufik2323@gmail.com	01300133568	Romance	1 Month	23	Male	Delete

Admin Add New Movie Page:



Add New Movie

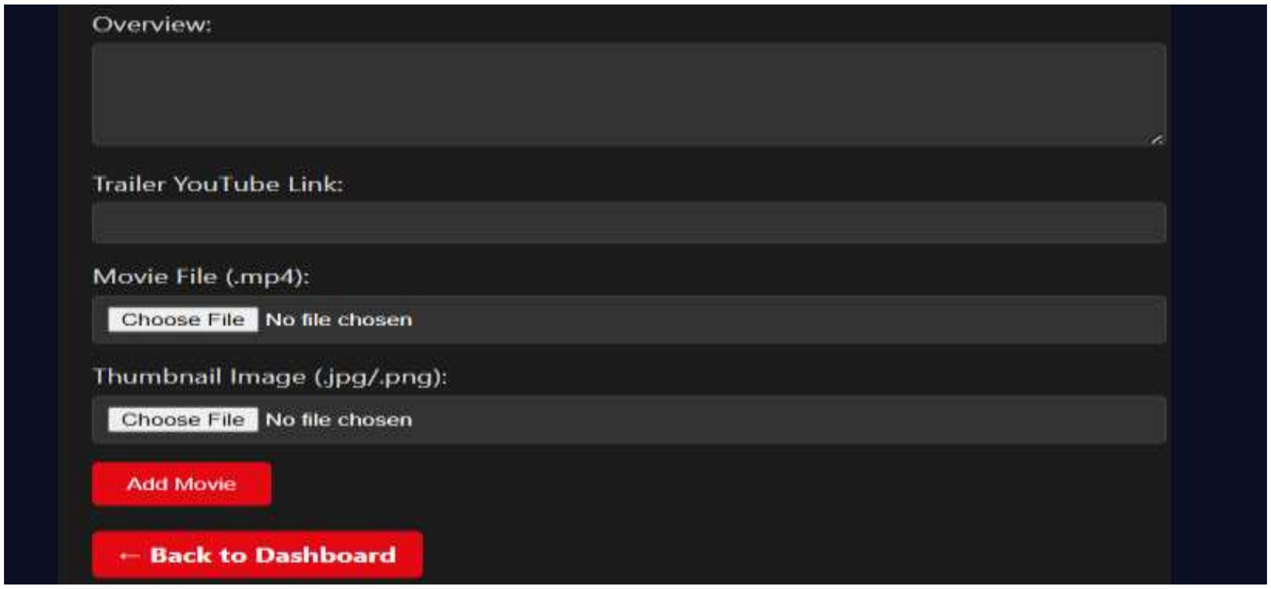
Name:

Genre:

Animation

Rating:

Duration (e.g. 2h14min):



Overview:

Trailer YouTube Link:

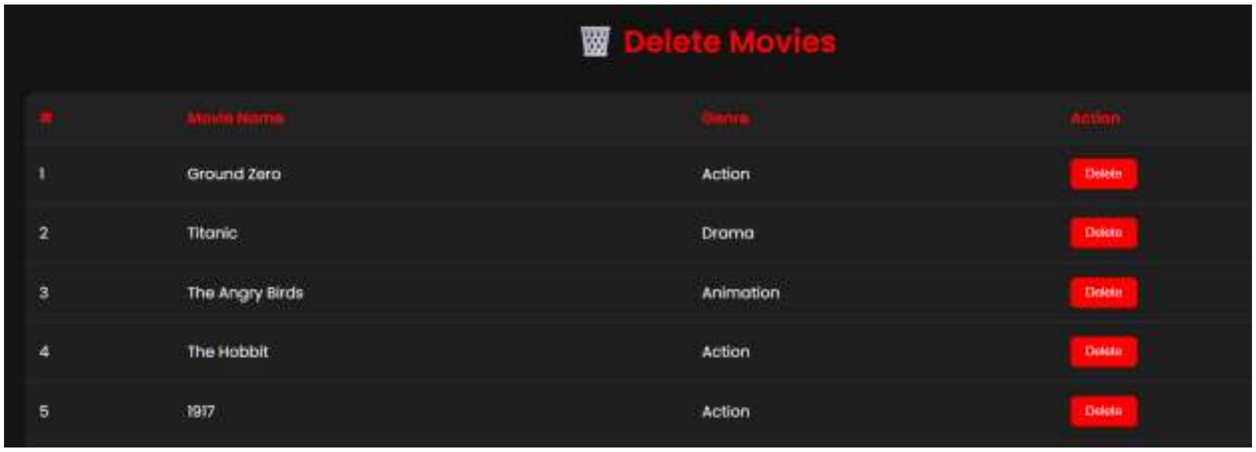
Movie File (.mp4):

Choose File No file chosen

Thumbnail Image (.jpg/.png):

Choose File No file chosen

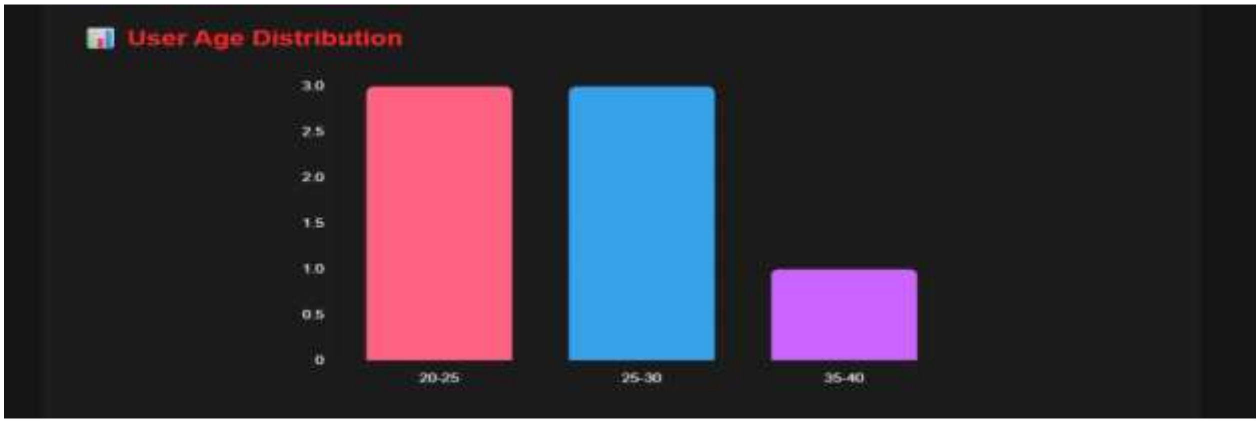
Admin Delete Page:

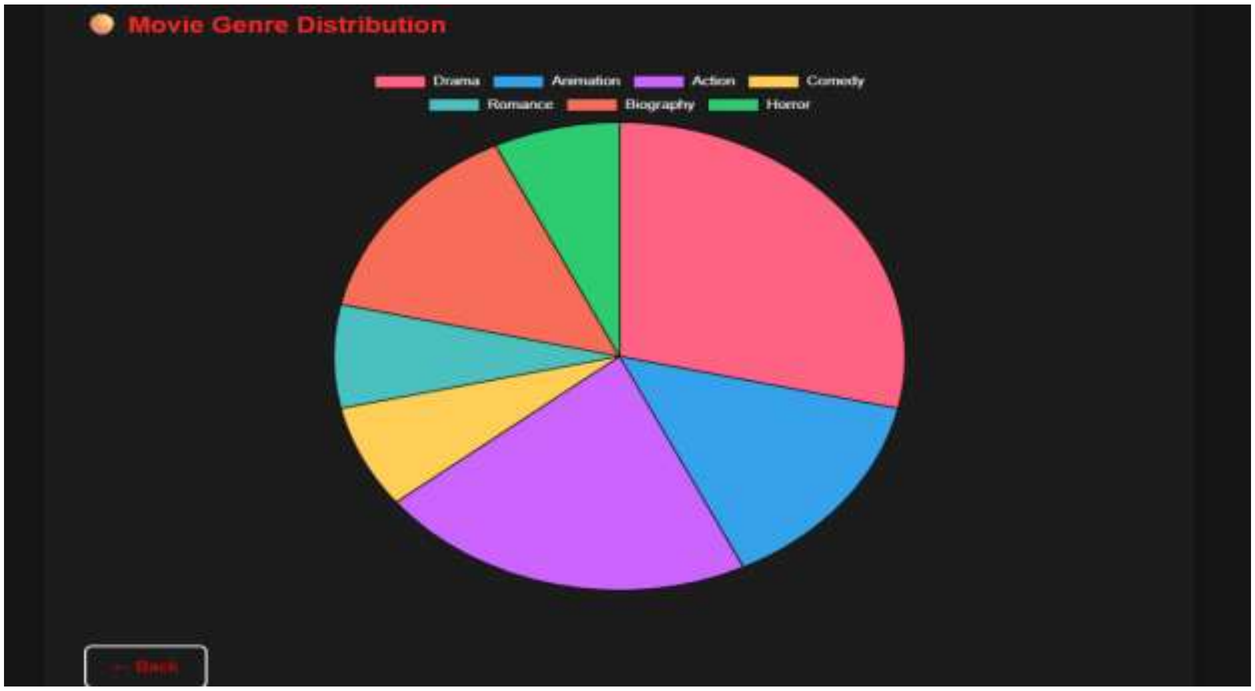


Delete Movies

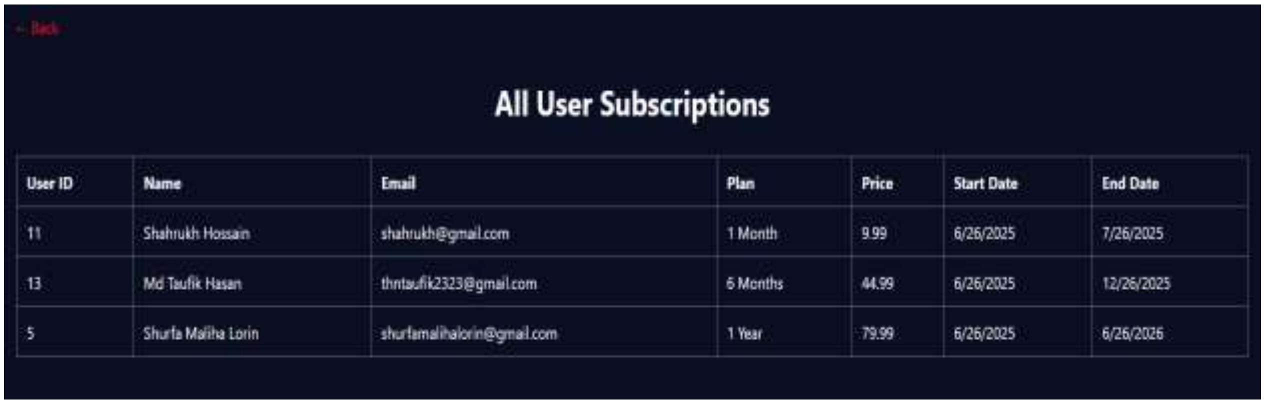
#	Movie Name	Genre	Action
1	Ground Zero	Action	Delete
2	Titanic	Drama	Delete
3	The Angry Birds	Animation	Delete
4	The Hobbit	Action	Delete
5	1917	Action	Delete

Admin See Overview Page:





Admin View User Subscription Page:



All User Subscriptions

User ID	Name	Email	Plan	Price	Start Date	End Date
11	Shahrukh Hossain	shahrukh@gmail.com	1 Month	9.99	6/26/2025	7/26/2025
13	Md Taufik Hasan	thntaufik2323@gmail.com	6 Months	44.99	6/26/2025	12/26/2025
5	Shurfa Maliha Lorin	shurfamalihalorin@gmail.com	1 Year	79.99	6/26/2025	6/26/2026

Challenges Faced and How We Solved Them

While developing this full-stack movie streaming and management platform, several technical and logical challenges arose. Below are the key obstacles encountered during the development process and how each was addressed:

1. Backend Logic and API Integration

Challenge: Building the backend using Node.js and Express was one of the initial hurdles, especially managing multiple routes for both users and admins. Ensuring proper separation between public and protected routes while maintaining clean, modular logic was difficult.

Solution: We organized backend routes systematically and implemented middleware for session validation. This ensured only authenticated users or admins could access sensitive pages. We also used express.json() and body-parser to handle incoming data properly in API requests.

2. Saving Watch Later Data in the Database

Challenge: Implementing a “Watch Later” feature required linking movie selections with the correct user, preventing duplicates, and fetching the list correctly.

Solution: We used session-based user identification (req.cookies.sessionId) and ensured the backend inserts the movie ID along with the user ID into a dedicated watch_later table. Then, we fetched the list using a JOIN query to match movie details dynamically.

3. Movie Download Functionality

Challenge: Enabling users to download movie videos securely was tricky, especially when dealing with file paths and MIME types.

Solution: We created a /download route that verifies the file’s existence and streams it with correct headers for video downloads. We also added basic access protection to avoid unauthorized file downloads.

4. User Profile Picture Upload

Challenge: Allowing users to upload profile pictures required handling multipart form data, storing the file, and then referencing it in the database.

Solution: We used the multer middleware to handle file uploads. Uploaded images were stored in a dedicated /uploads folder and saved with unique names. The image path was then stored in the user's profile in the MySQL database and retrieved when rendering the profile.

5. Admin Dynamic Movie Insertion

Challenge: One of the more complex tasks was creating an interface where the admin could dynamically add movies (title, genre, description, image, video) and make them appear immediately in the movie list.

Solution: We built a movie form (add-movie.html) that accepted inputs and file uploads. The backend processed the form, saved images/videos to public/movies/, and inserted movie details into the database. The front-end movie listing was dynamically populated by querying the database.

6. Filtering and Search Mechanism

Challenge: Implementing an efficient filter and search experience without using a framework like React was tough. Matching genre, type, and categories like "Trending Now" needed dynamic behavior.

Solution: We used URL query strings to filter content (filter.html?genre=Drama) and implemented JavaScript that filtered the movie list based on genre, type, or popularity. We also handled server-side filtering for more accurate search results.

7. Subscription Status Logic

Challenge: Displaying the current and past subscription based on the current date required time-based logic and conditional rendering.

Solution: In the backend /api/profile route, We compared start_date and end_date of subscriptions with today's date. If the subscription was active, it showed under "Current Subscription"; otherwise, it appeared under "Past Subscriptions."

8. Managing Responsive Design

Challenge: Ensuring that all pages looked clean and functional on both mobile and desktop screens was time-consuming. The layout often broke on smaller viewports.

Solution: We used Flexbox, media queries, and percentage-based widths in the custom CSS. We tested each major view on various screen sizes and adjusted spacing, font sizes, and grid behaviors accordingly.

9. Handling Session Authentication for Both Admin and User

Challenge:
Managing two separate login flows (admin vs. user) while avoiding cross-session issues was a tricky task.

Solution:
I added a checkbox in the login form to determine the role (admin/user) and routed the login request accordingly. Separate session variables were maintained in the backend to distinguish roles and ensure protected routes were role-specific.

10. File Path Issues and Static Serving

Challenge: Incorrect paths or broken images/videos were frequent problems, especially due to improper directory references in HTML and JS.

Solution: We structured all static files properly in /public and /uploads, then used express.static() to serve these files. All references were double-checked to use correct relative paths or absolute URLs.

11. Ensuring Smooth Admin/User UI Separation

Challenge: Since both user and admin interfaces existed under one codebase, maintaining clear UI separation was crucial to avoid confusion and access errors.

Solution: We organized files under the /protected directory, added different dashboards (admin-dashboard.html, user-profile.html) and used conditional routes and session checks to ensure proper redirection after login.

12. Live Preview of Uploaded Images

Challenge: Adding a preview of the selected profile picture before upload was requested for better user experience.

Solution: We used the FileReader API and URL.createObjectURL() in JavaScript to display a temporary preview when a user selected an image before submitting.

These challenges, while initially roadblocks, became learning opportunities. Solving them improved my understanding of full-stack development, file handling, authentication, session management, and database interactions—skills that are essential for real-world web application development.

Learning Outcomes:

1. Full-Stack Application Development:
I gained hands-on experience in building a complete web application that connects frontend and backend logic. This included creating routes, working with middleware, handling file uploads, and rendering dynamic content.

2. Database Integration with MySQL:
Working with MySQL helped me understand how to design relational schemas, write optimized queries, and manage data integrity for users, subscriptions, movies, and reviews.

3. Session-Based Authentication:
I implemented secure login systems for users and admins using cookies and sessions. This taught me the importance of protecting routes, maintaining sessions, and handling user identity in real time.

4. File Handling with Multer/Express-FileUpload:
Uploading and storing images and videos was a key feature. Using Multer middleware gave me practical experience in handling multipart form data and managing static file access.

5. Frontend Design and Responsiveness:
Through custom CSS, Flexbox, and media queries, I built an attractive and responsive interface compatible with mobile and desktop devices. This enhanced my understanding of user interface (UI) design and UX optimization.

6. Dynamic Rendering and JavaScript Interactions:
Using JavaScript to dynamically populate movies, handle reviews, and update UI components helped me learn about DOM manipulation and asynchronous programming using fetch.

7. Admin Control & Content Management:
I built an admin dashboard that enabled content addition, deletion, and user monitoring. This simulated real-world administrative features of content platforms and gave insight into role-based access control.

8. Debugging and Problem Solving:
Throughout the project, I encountered and resolved several backend and frontend bugs. These challenges significantly improved my problem-solving skills and debugging strategies.

9. Real-Time Features (Chat/Review):
Implementing features like user comments and reviews helped me understand how to build interactive elements that reflect changes immediately and are stored permanently in the backend.

Conclusion

Building this web-based movie streaming and management system has been a rewarding and insightful experience. The project successfully integrates both frontend and backend technologies to deliver a feature-rich platform for users and administrators. Users can browse and watch movies, manage their profiles and subscriptions, leave reviews, and save favorites — while administrators can dynamically manage content, users, and view analytics. Throughout the development process, I applied real-world web development practices, such as session-based authentication, RESTful routing, database integration, and file handling. The challenges I faced — from implementing secure login, to handling dynamic content and file uploads — enhanced my problem-solving skills and deepened my understanding of full-stack development. This project not only strengthened my technical abilities but also improved my logical thinking, attention to detail, and project management skills. It has prepared me for future development work by providing hands-on experience with technologies like Node.js, Express, MySQL, JavaScript, and modern CSS — and most importantly, it taught me how to turn ideas into fully functional, user-friendly web applications.