Final Project

COP 4808 Full stack web development

Professor : David Jaramillio

Date: 01 May 2023

group # 9

Team github link to project:
https://github.com/cop4808-spring-2023-fullstack-web/final-project-Team-9

Project website link (deployed through firebase):
https://movie-app-full-stack-1.web.app/trending.html

Youtube Demo:
https://youtu.be/3570_EPu4EI

Team members :

Anastasija Bulatovic: https://github.com/bulatovic-anastasija
Marven Cesar : https://github.com/MarvenCesar
Carlo Leiva : https://github.com/Vidacelinda
Thomas Wozniak :https://github.com/Wozn14

Tools

Firebase (database and authentication) ,javascript,html,css,node js,bootstrap

Firebase
Firebase is a Backend-as-a-Service (BaaS) platform developed by Google. It provides various services, such as real-time databases, authentication, cloud storage, and hosting, that help developers build and scale web and mobile applications without managing backend infrastructure.

JavaScript
JavaScript is a widely-used, high-level programming language primarily used for client-side web development. It enables developers to create interactive and dynamic web applications by manipulating HTML elements, handling user events, and making AJAX requests to servers.

HTML
HTML (HyperText Markup Language) is the standard markup language for creating web pages and web applications. It uses a system of tags and attributes to structure and organize content, such as text, images, and multimedia, in a web browser.

CSS
CSS (Cascading Style Sheets) is a stylesheet language used to describe the look and formatting of a document written in HTML. It allows developers to apply styles (such as colors, fonts, and spacing) to HTML elements, control their layout, and create responsive designs.

Node.js
Node.js is an open-source, cross-platform runtime environment that allows developers to execute JavaScript code on the server-side. Built on Chrome's V8 JavaScript engine, Node.js enables the creation of scalable and high-performance server-side applications, utilizing an event-driven, non-blocking I/O model.

Bootstrap
Bootstrap is a popular open-source CSS framework designed for responsive, mobile-first web development. Developed by Twitter, it offers a collection of pre-built CSS and JavaScript components that developers can easily integrate into their projects. These components include responsive grids, typography, forms, buttons, navigation, modals, and more. By using Bootstrap, developers can quickly create professional-looking and responsive web applications, ensuring a consistent design and improved user experience across various devices and screen sizes.

Project design

In order To fulfill API requests from MovieDB, we will need an API key that allows us to access their resources. The API KEY: ***2f1775072b783326efa2a8d064dfeb34***
With this, our app will be able to send and receive requests using their resources to allow us to present a variety of movies with different genres and popularity. In addition to this, each Movie should have an image, description, and rating.
The primary methods being used for the Movie app will be: Get, Post, and Delete.
Some endpoint examples include but are not limited to:

GET - Creating Guest Session
- https://api.themoviedb.org/3/authentication/token/new?api_key=2f1775072b783326efa2a8d064dfeb34

GET - Searching Movie By Name
- https://api.themoviedb.org/3/search/movie?api_key=2f1775072b783326efa2a8d064dfeb34&language=en-US&query={Keyword}&page=1&include_adult=false

GET - Searching Movie By Genre
- https://api.themoviedb.org/3/genre/movie/list?api_key=2f1775072b783326efa2a8d064dfeb34&language=en-US

GET - Trending Movies
- https://api.themoviedb.org/3/trending/all/day?api_key=2f1775072b783326efa2a8d064dfeb34

POST - Rate a movie
- https://api.themoviedb.org/3/movie/500/rating?api_key=2f1775072b783326efa2a8d064dfeb34&session_id=df40596ef4e082ab7231ccf247e6699bb1c063ee

DELETE - Delete Rating from Movie

- https://api.themoviedb.org/3/discover/movie?api_key=2f1775072b783326efa2a8d064dfeb34&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate

GET - Discover Movie by different parameters

- https://api.themoviedb.org/3/discover/movie?api_key=2f1775072b783326efa2a8d064dfeb34&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate

GET /movie/top_rated

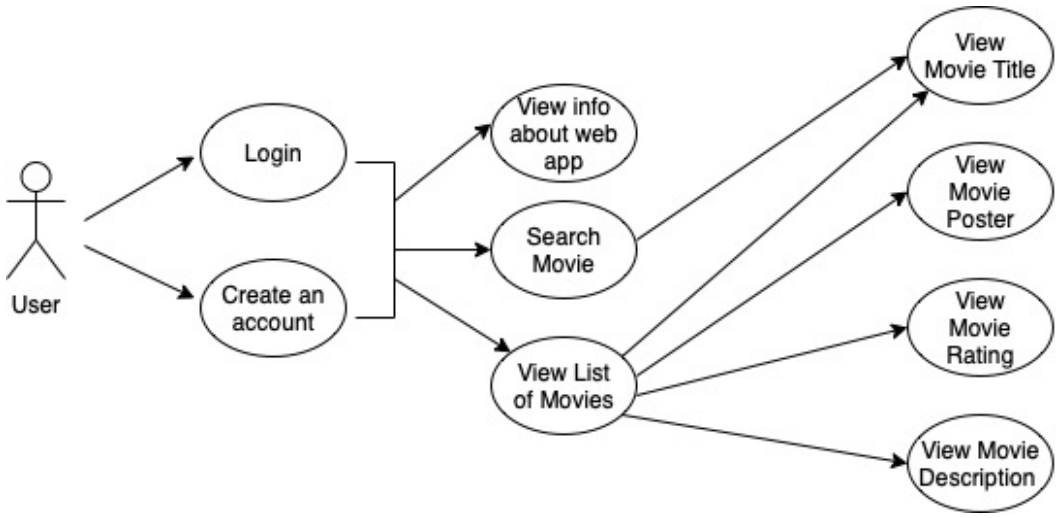
- https://api.themoviedb.org/3/movie/top_rated?api_key=2f1775072b783326efa2a8d064dfeb34&language=en-US&page=1

POST Favorite

- https://api.themoviedb.org/3/account/{account_id}/favorite?api_key=%202f1775072b783326efa2a8d064dfeb34%20&session_id=3075e9a40bc7ad67241126ef6eb11df6

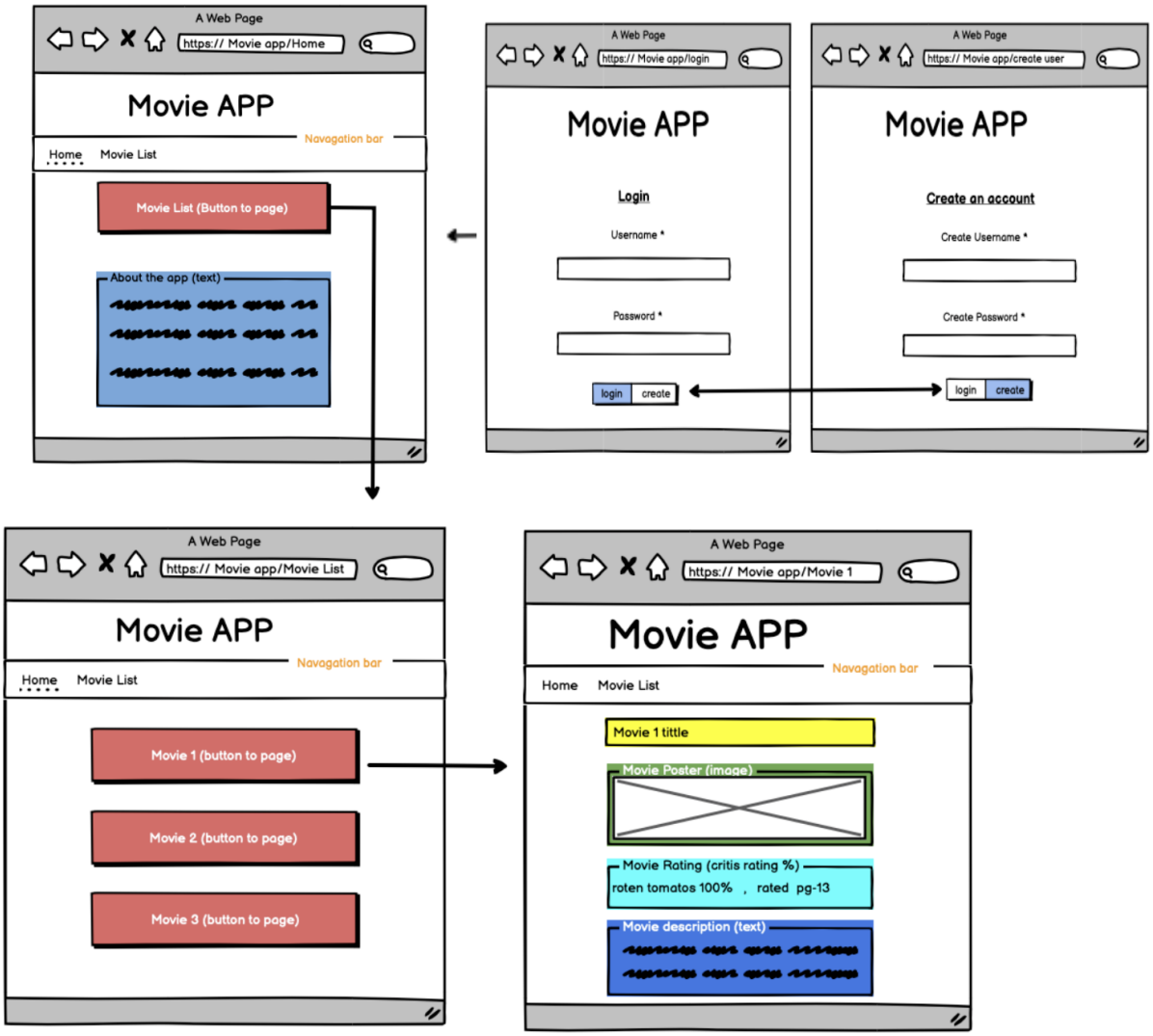
There will be no PUT method because it is not necessary or required for this app.

initial Use Case Diagram of the App:

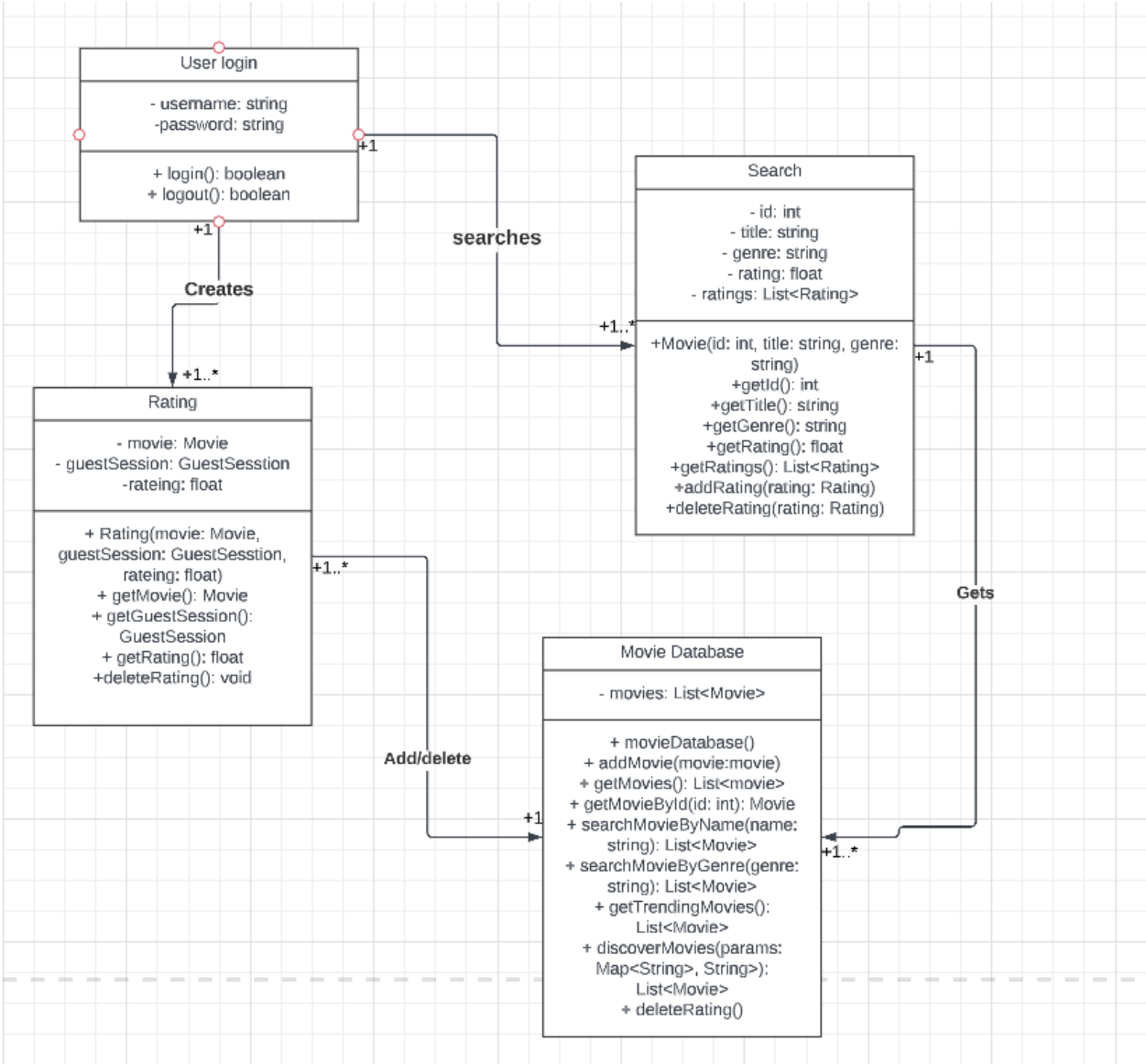


Initial URL link to UI design

NOTE: Start at login and also url for each page are also done properly



UML class digram design.



Code review

For the project we used a firebase database to store the user selected favorite movies.
We will describe the key functionalities of our documents and illustrate how they operate in conjunction with Firebase, including user authentication, the favorites page, the search page, the trending page, and the top-rated page.We will first explain the usage quickly how each page works then we will go in depth with the code.

User Authentication

Movie App uses Firebase Authentication for managing user accounts. The app supports both Google account login and email/password login. If the user does not have an account they may also sign up. The user is able to sign in to view contents of application & also sign out when done. Each user has a validation token generated once signed up and used again to validated every time they log back in.

Home Page

After logging in, you will be taken to the home page, where you can see a list of the top-rated movies. You can hover over the movie posters to see the movie title and a brief overview.

Trending Movies

To view the trending movies, navigate to the "Trending" page using the link in the navigation bar. A list of currently trending movies will be displayed in a scrollable container. Hover over the movie posters to see the movie title and a brief overview.

Searching for Movies

To search for a movie, enter a keyword or movie title in the search bar located at the top of the home page. Click the "Search" button to see the search results. A list of movies matching your query will be displayed.

Adding Movies to Favorites

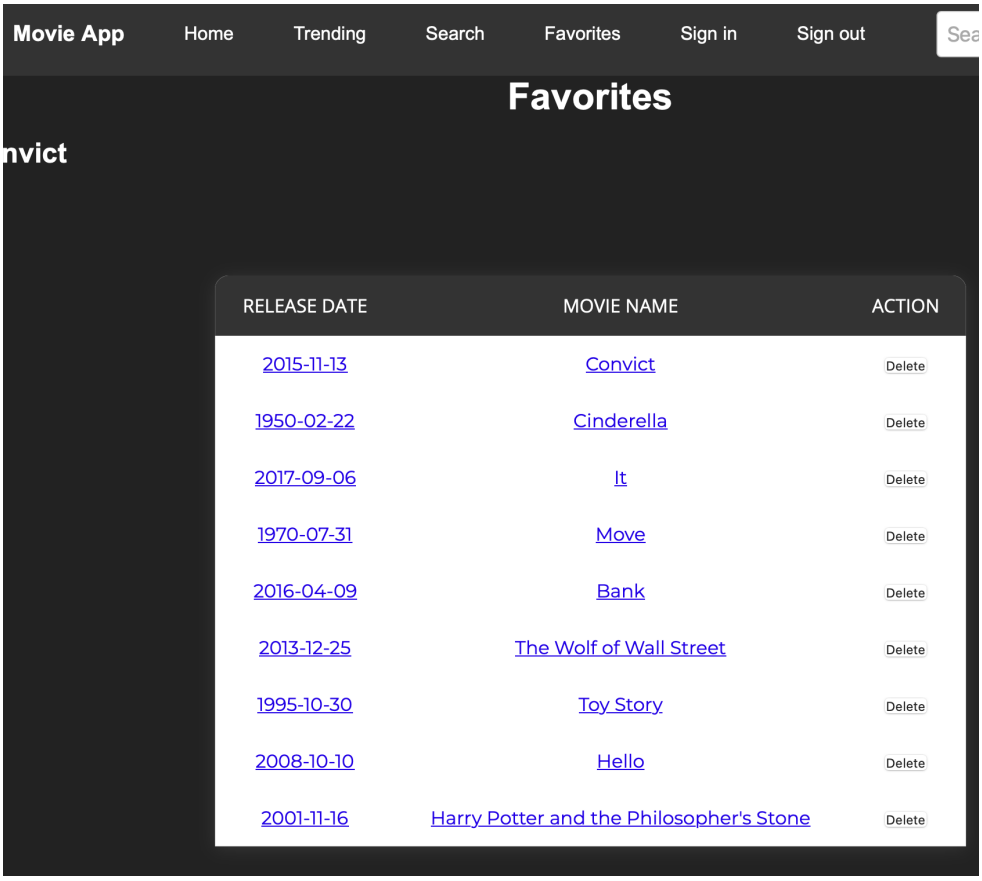
To add a movie to your favorites list, Once you have made a search to the desired movie Click on the "Add to Favorites" button to save add the movie to your favorites list.

Viewing and Managing Favorites

To view your saved favorites, click on the "Favorites" link in the navigation bar at the top of the page. This will take you to the favorites page, where you can see a list of your saved movies, along with their release dates.

To remove a movie from your favorites list, click on the "Delete" button next to the movie title in the favorites table. The movie will be removed from your list, and the table will update to reflect the change.

Favorites page (in depth code : favorites.html)



We mainly used the database to create a favorite list for the movie app. Let's break down the code line by line:

```
<link rel="stylesheet" href="navbar.css">
<link rel="stylesheet" href="homestyle.css">
```

```
<link rel="stylesheet" href="fave_table.css">
```

The first two lines link to external stylesheets that define the look and feel of the webpage. They are `navbar.css` and `homestyle.css`.The “fave_table.css” links to an external stylesheet `fave_table.css`, which is used to style the table of the favorite movies page.

```
<script type="module" src="https://www.gstatic.com/firebasejs/9.0.0/firebase-app.js"></script>
<script type="module" src="https://www.gstatic.com/firebasejs/9.0.0/firebase-firestore.js"></script>

<!-- added for db -->
<script src="/__/firebase/9.19.1/firebase-app-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-auth-compat.js"></script>
<!-- database  -->
<script src="/__/firebase/9.19.1/firebase-database-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-firestore-compat.js"></script>
<!--  DB-->
<script src="/__/firebase/9.19.1/firebase-functions-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-messaging-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-storage-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-analytics-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-remote-config-compat.js"></script>
<script src="/__/firebase/9.19.1/firebase-performance-compat.js"></script>
```

These lines of code link to various scripts that are used to initialize Firebase and interact with the Firestore database.

```
<body>
        <div id="navbar"></div>
```

Within the HTML body a `div` element was created the id `navbar` where the navbar will be loaded dynamically using JavaScript.

```
        <div class="container">
        <h1>Favorites</h1>
        <!-- DO NOT REMOVE THIS H2 MOVIE NAME!! Will cause bug otherwise! -->
        <h2 id="movieName"></h2>
        <div id="favorites-container"> </div>
```

This creates a container with a `h1` header that says "Favorites". There is also an empty `div` with the id `favorites-container`, which will be used to display the list of favorite movies. There is an `h2` element with the id `movieName`as wel.

```
        <div>
            <table id="favTable">
                <thead>
                  <tr>
                    <th>Release Date</th>
                    <th>Movie Name</th>
                    <th>Action</th>
                  </tr>
                </thead>
                <tbody>
                </tbody>
              </table>
          </div>
      </div>
    <script src="loadnavbar.js"></script>
```

This creates a table with three columns: "Release Date", "Movie Name", and "Action". The `thead` element contains the column headers, while the `tbody` element is currently empty. At the end a script line loads an external JavaScript file called `loadnavbar.js`, which is used to load the navbar dynamically.

```
<script>
  // Initialize Firebase
  const firebaseConfig = {…
  firebase.initializeApp(firebaseConfig);
```

We start a script block that contains JavaScript code that connects with fire base and works with the favorite list database.We initializes Firebase with the given configuration object `firebaseConfig`.

```
// Retrieve movieName from Firestore
const db = firebase.firestore();
const myFav = db.collection('favList').doc('fav');
myFav.get().then(doc => {
  if (doc.exists) {
    const data = doc.data();
    const movieName = data.movieName;

    // Update h1 element with first movie name
    const a = document.getElementById('movieName');
    a.textContent = movieName[0].title;

    // Update table with movie names
    const table = document.getElementById('favTable');
    for (let i = 0; i < movieName.length; i++) {
      const row = table.insertRow(-1);
      const cell1 = row.insertCell(0);
      const cell2 = row.insertCell(1);
      const cell3 = row.insertCell(2);
      cell1.innerHTML = '<a href="https://www.example.com/movie">' + movieName[i].release_date + '</a>';
      cell2.innerHTML = '<a href="https://www.example.com/movie">' + movieName[i].title + '</a>';
      cell3.innerHTML = '<button onclick="deleteMovie(' + i + ')">Delete</button>';

      // if (i === 0) {
      //   const a = document.getElementById('movieName');
      //   a.textContent = movieName[i].title;
      // }
    }
  } else {
    console.log("No such document!");
  }
}).catch(error => {
  console.log("Error getting document:", error);
});
```

The code above retrieves the `movieName` data from the Firestore database and updates the HTML elements with the corresponding movie data. The first movie name is displayed in the `h2` element with the id `movieName`. The table is populated with the movie names and release dates, and a "Delete" button is added to each row.

```
// Delete a movie from the array and update Firestore
function deleteMovie(index) {
    myFav.get().then((doc) => {
      if (doc.exists) {
        const data = doc.data();
        const movieName = data.movieName;

        // Delete the movie at the given index
        movieName.splice(index, 1);

        // Save the updated array back to Firestore
        return myFav.update({ movieName: movieName });
      } else {
        console.log('No such document!');
        return null;
      }
    }).then((result) => {
      if (result !== null) {
        console.log('Element successfully deleted from array!');
        location.reload(); // Reload the page to update the table
      }
      }).catch((error) => {
          console.error('Error deleting element from array:', error);
      });
}
```

This is a function called `deleteMovie` that takes an index as an argument. It is called when a "Delete" button is clicked on a row of the table.

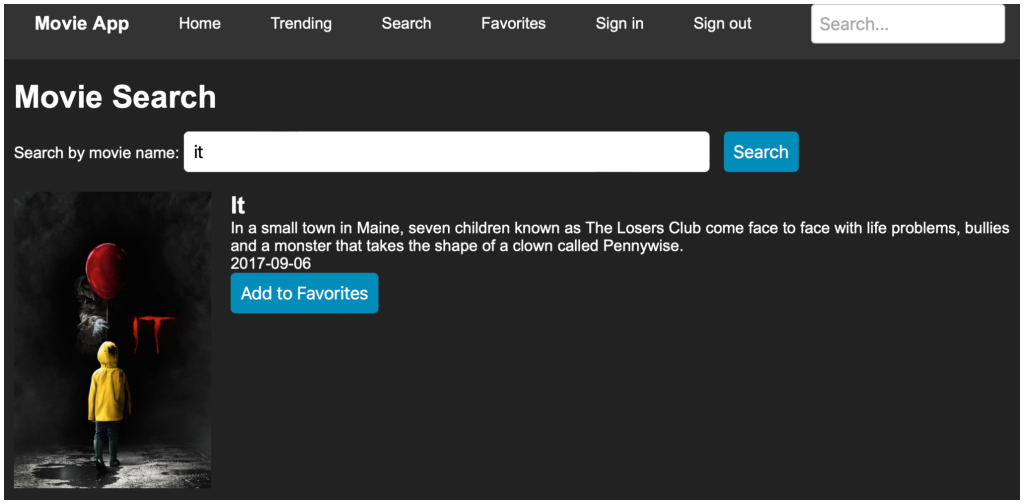
The function first retrieves the `movieName` data from the Firestore database using the `myFav` reference. It then gets the `movieName` array from the retrieved data and deletes the movie at the given index using the `splice` method.

Next, the function updates the `movieName` array in the Firestore database using the `update` method on the `myFav` reference. The updated array is passed as an object with the key `movieName`.

If the update is successful, the page is reloaded using `location.reload()` to update the table with the new data. If there is an error, an error message is logged to the console.

Overall, this function updates the Firestore database by deleting a movie from the `movieName` array at the given index, and then updates the table to reflect the new data.

Search Page (in depth code:search.html)



The following code snippets showcase various aspects of the search page, including required HTML and scripts. As the HTML at the beginning is similar to that of the favorites page discussed earlier, we will skip it for the sake of efficiency. The code below provides further details on the necessary forms, request endpoints, and how the "Add to Favorites" button functions.
explanations:

1. HTML form for movie search:

```
<body>
    <div id="navbar"></div>
    <div class="container">
        <h1>Movie Search</h1>
        <form>
            <label for="search">Search by movie name:</label>
            <input type="text" id="search" name="search">
            <button type="button" onclick="searchMovies()">Search</button>
        </form>
        <div class="poster">
            <img id="poster" src="">
```

This snippet creates a form with a label, an input field (with the id "search"), and a search button. When the search button is clicked, the `searchMovies()` function is called to perform the movie search.

2. Initializing Firebase:

```
// Initialize Firebase
const firebaseConfig = {…
firebase.initializeApp(firebaseConfig);
```

This snippet initializes the Firebase app with the given configuration details. The configuration object contains necessary information such as API key, authDomain, databaseURL, projectId, etc.

3. Firestore database reference:

```
// Get a reference to the Firestore database
const db = firebase.firestore();

// Get a reference to the 'fav' document in the 'favList' collection
const myFav = db.collection('favList').doc('fav');
```

This snippet creates a reference to the Firestore database and to the 'fav' document in the 'favList' collection. This reference will be used to interact with the Firestore database, specifically to store and update favorite movies.

4. searchMovies function:

```
function searchMovies() {
    const url = `http://localhost:3000/movies?query=${document.getElementById("search").value}`
    $.get(url, function(data) {
        if (data.error) {
            alert(data.error);
        } else {
            document.getElementById("poster").src = data.poster_path;
            document.getElementById("title").innerHTML = data.title;
            document.getElementById("overview").innerHTML = data.overview;
            document.getElementById("release").innerHTML = data.release_date;

            // Remove any existing favorite button before appending a new one
            const existingButton = document.getElementById("favorite-button");
            if (existingButton) {
                existingButton.remove();
            }

            // Create a new button element
            const favoriteButton = document.createElement("button");
            favoriteButton.innerHTML = "Add to Favorites";
            favoriteButton.id = "favorite-button";

            // Add an event listener to the button that stores the name of the movie
            favoriteButton.addEventListener("click", function() {
                const movieName = {
                title: data.title,
                release_date: data.release_date
            };
                storeFavoriteMovie(movieName);
            });

            // Pass the name of the movie to the function using a data attribute
            favoriteButton.setAttribute("data-movie-name", data.title);

            // Add the button element to the page
            document.querySelector(".poster").appendChild(favoriteButton);
        }
    });
}
```

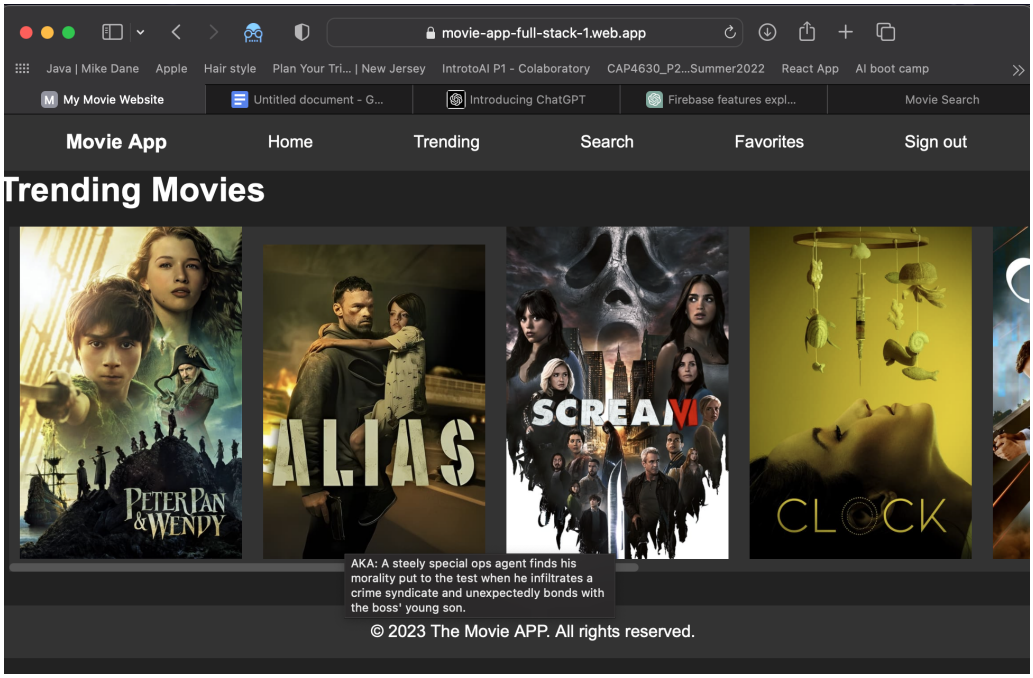
This snippet defines the `searchMovies()` function, which sends an HTTP GET request to the specified URL with the search query. Upon receiving the data, the function updates the movie poster, title, overview, and release date in the HTML structure.

5. Add to Favorites button and event listener:

```
// Create a new button element
const favoriteButton = document.createElement("button");
favoriteButton.innerHTML = "Add to Favorites";
favoriteButton.id = "favorite-button";

// Add an event listener to the button that stores the name of the movie
favoriteButton.addEventListener("click", function() {
    const movieName = {
    title: data.title,
    release_date: data.release_date
};
    storeFavoriteMovie(movieName);
});

// Pass the name of the movie to the function using a data attribute
favoriteButton.setAttribute("data-movie-name", data.title);

// Add the button element to the page
document.querySelector(".poster").appendChild(favoriteButton);
```

This snippet creates a new button element for "Add to Favorites," assigns it an id, and adds an event listener. When the button is clicked, it calls the `storeFavoriteMovie()` function to store the movie in the favorites list. The button is then appended to the ".poster" div in the HTML structure.

Trending page (in depth code: trending.html)



This code is an HTML document that displays the top trending movies using a local server API. It contains HTML, CSS (not shown in the provided code because of simplicity sake and does not really affect the functionality), and JavaScript.

Here's a description of each section and their functionality:

1. **Navigation**

```
<div id="navbar"></div>

<script src="loadnavbar.js"></script>
```

Once again this snippet creates a `div` with an id of "navbar" and includes the `loadnavbar.js` script, which is responsible for loading the navigation bar in the "navbar" div.

2. **Trending Movies heading and container**

```
<!-- TOP RATED -->
<h1>Trending Movies</h1>
<div class="movie-scroll-container">
    <div id="movie-posters"></div>
</div>
```

This snippet creates a heading "Trending Movies" and a container with a class of "movie-scroll-container." Inside the container, there's another `div` with an id of "movie-posters" to display the movie posters.

3. **API call using jQuery**

```
$.get('http://localhost:3000/movietrending', function(data) {
```

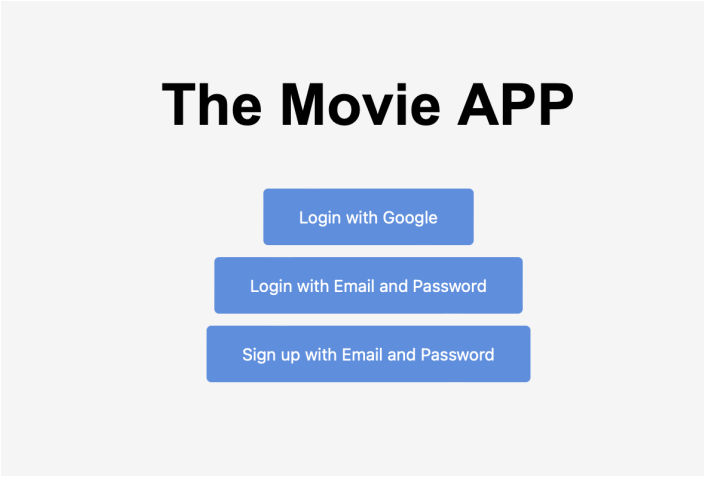
This snippet sends an HTTP GET request to the specified local server API endpoint with the help of jQuery. Upon receiving the data, the function processes it to display movie posters.

4. **Rendering movie posters**

```
for (var i = 0; i < 5; i++) {
    // Get the movie poster path and construct the full image URL
    var posterPath = data.results[i].poster_path;
    var imageUrl = "https://image.tmdb.org/t/p/w500" + posterPath;
    // Create an image element for the movie poster and set its source
    var img = document.createElement("img");
    img.src = imageUrl;
    img.alt = data.results[i].title;
    img.className = "movie-poster";
    // Add the image element to the container
    document.getElementById("movie-posters").appendChild(img);
}
```

This snippet is part of the callback function for the API call. It iterates through the top 5 movies in the received data, creates an image element for each movie poster, and appends it to the "movie-posters" div. The image source, alternate text, and class are set using the movie details from the data.

FireBase Authentication page in depth code



This project uses Firebase Authentication, a comprehensive user authentication solution provided by Firebase, a platform developed by Google for creating web and mobile apps.

- **Sign in with Google**: This feature allows users to log in to the application using their Google account. By integrating Google Sign-In, users don't have to remember another username and password.

```
async function googleLogin() {
  const provider = new firebase.auth.GoogleAuthProvider();
  firebase.auth().signInWithPopup(provider)
  .then(async (result) => {
    const user = result.user;
    window.location.href = `home.html`;
    console.log(user);
    const idToken = await user.getIdToken();
    await validateToken(idToken);
  })
  .catch(console.log);
}
```

- **Sign up and sign in with email and password**: This feature enables users to create a new account using their email address and a unique password. After registering, users can sign in to the app using the same credentials. Firebase Authentication securely stores user credentials and handles the entire authentication process, including password hashing and secure token management.

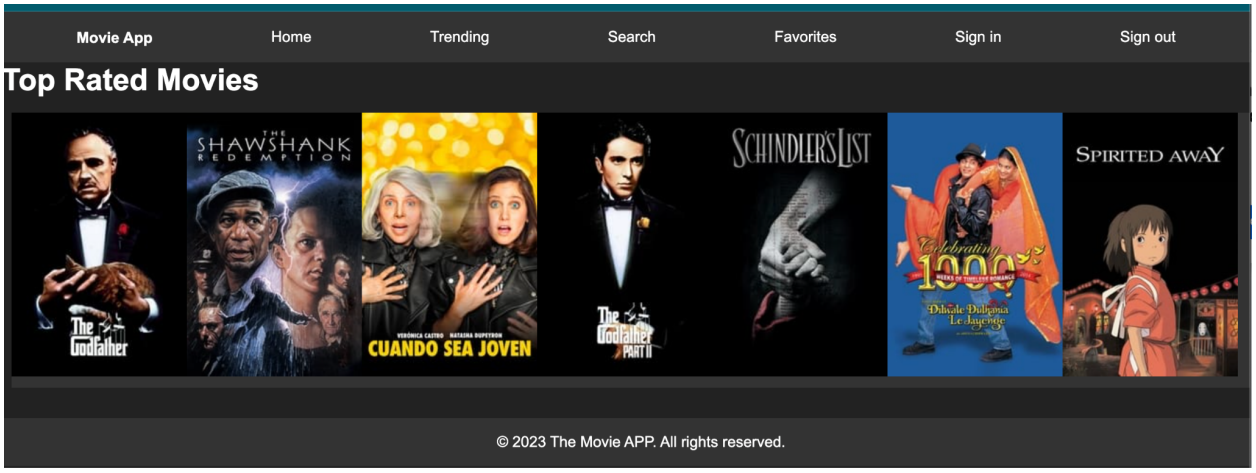
```
async function emailSignUp() {
  const fullName = document.getElementById("fullname").value;
  const email = document.getElementById("email").value;
  const password = document.getElementById("password").value;
  firebase.auth().createUserWithEmailAndPassword(email, password)
  .then(async (result) => {
    const user = result.user;
    const idToken = await user.getIdToken();
    await validateToken(idToken);
    await user.updateProfile({
      displayName: fullName,

    });
    window.location.href = `home.html`;
})
.catch(console.log);
}

async function emailLogin() {
  const email = document.getElementById("login-email").value;
  const password = document.getElementById("login-password").value;
  firebase.auth().signInWithEmailAndPassword(email, password)
  .then(async (result) => {

      const user = result.user;
      const idToken = await user.getIdToken();
      await validateToken(idToken);
      const fullName = user.displayName;
      window.location.href = `home.html`;
    })
```

- **Retrieve, display, and validate user ID token**: Once a user is authenticated, Firebase generates a unique ID token for that user. This ID token contains information about the user, such as their email address, display name, and profile picture. The project retrieves and displays this information in the app. Additionally, the app can validate the ID token to ensure the user is properly authenticated before accessing protected resources or performing actions that require authentication. This token validation process helps maintain a secure environment within the application.

```
async function getIdToken() {
  const user = firebase.auth().currentUser;
  if (user) {
    user.getIdToken(true).then((idToken) => {
      console.log('ID token:', idToken);
    }).catch((error) => {
      console.error('Error getting ID token:', error);
    });
  } else {
    console.log('No user is currently signed in.');
  }
}

async function validateToken(token) {
  console.log("Token in validateToken function:", token);
  try {
    const response = await fetch("http://localhost:3000/protected", {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + token
      },
      credentials: 'include'
    });

    const data = await response.json();
    if (data.message === "Access granted to protected route") {
      console.log("Token validated successfully");
    } else {
      console.log("Token validation failed");
    }
  } catch (error) {
```

Top Rated page (in depth code: home.html)

We decided to implement a home.html page for our web app. Upon logging in the user is redirected to the homepage that displays the top rated movies.
This is an image of our home page and the movies that are displayed there:



This home page is accessed upon user login.
Implementation:

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <title>My Movie Website</title>
    <link rel="stylesheet" href="navbar.css">
    <link rel="stylesheet" href="homestyle.css">
```

At the beginning of the home.html file the style CSS sheets are loaded, they set up the look and stylization of out web-app.
A JQuery script is also included. This allows us to traverse and manipulate the HTML Document. In this case we will use it to allow us to access the results of our backend api call which returns the top rated movies.

```
<div id="navbar"></div>

<script src="loadnavbar.js"></script>
```

Within our body the nav bar is implemented and dynamically loaded through a JavaScript file.

Within our home.html file we have a script that is implemented through JQuery code that retrieves movie data from the backend api and renders the poster images of the top 7 rated

movies. We chose to implement it this way so that the user cannot access our API keys, which intern bolsters the security of or application.

This is the code where this is implemented.

```
<script>
    // API endpoint to retrieve top 7 rated movies
    // Function to retrieve data from API and render poster images
    $.getJSON('http://localhost:3000/movie', function(data) {
        var movies = data.results;
        //Render posters of first 7 movies
        for (var i = 0; i < 7; i++) {
            var poster_url = "https://image.tmdb.org/t/p/w185" + movies[i].poster_path;
            var poster = "<img src='" + poster_url + "'>";
            $("#movie-posters").append(poster);
        }
        });
```

This script also communicates with our backend api endpoint. (which can be seen here below):

```
//API endpoint that retrieves the top rated movies
app.get('/movie', (req, res) => {
  const Url = `https://api.themoviedb.org/3/movie/top_rated?api_key=${API_KEY}`;
    request(Url, (error, response, body) => {
        if (error) {
            console.error(error);
            res.status(500).send('An error occurred');
        } else {
            res.send(body);
        }
    });
});
```

This api call retrieves the top rated movies which are then rendered to the html page through the html included script.
We chose not to directly include the api keys in the backend either instead they have been declared in a “config.js” file and are then imported/passed through a require statement into the API_KEY variable.

FireBase Deploy

Firebase Deploy is a command used in this app to upload and deploy the web app's resources to Firebase Hosting. It allows the app to host on a fast, secure, and reliable hosting service with global CDN support, ensuring optimal performance and availability for users.

```
marvencesar@Marvens-MacBook-Pro final-project-Team-9 % firebase deploy

=== Deploying to 'movie-app-full-stack-1'...

i  deploying hosting
i  hosting[movie-app-full-stack-1]: beginning deploy...
i  hosting[movie-app-full-stack-1]: found 16 files in public
✔  hosting[movie-app-full-stack-1]: file upload complete
i  hosting[movie-app-full-stack-1]: finalizing version...
✔  hosting[movie-app-full-stack-1]: version finalized
i  hosting[movie-app-full-stack-1]: releasing new version...
✔  hosting[movie-app-full-stack-1]: release complete

✔  Deploy complete!

Project Console: https://console.firebase.google.com/project/movie-app-full-stack-1/overview
Hosting URL: https://movie-app-full-stack-1.web.app
marvencesar@Marvens-MacBook-Pro final-project-Team-9 %
```

One important note: Make sure Hosting URL matches origin CORS policy in server.js or else backend will not be able to properly connect to frontend due to connection refusal.

```
const corsOptions = {
  origin: "https://movie-app-full-stack-1.web.app",
  methods: "GET,HEAD,PUT,PATCH,POST,DELETE",
  allowedHeaders: [
    "Content-Type",
    "Authorization",
    "Origin",
    "X-Requested-With",
    "Accept"
  ],
  credentials: true
};
```

Hurdles

Each team member faced the challenge of learning Firebase and its implementation in the project. Firebase, although slightly different from MongoDB, which the team learned in class, shares similar concepts and query structures. This allowed the team to leverage their existing knowledge while adapting to Firebase's unique features.

By learning Firebase, the team members gained experience with a popular and powerful platform that offers real-time database capabilities, user authentication, and storage services.

This new skill set will not only enhance their ability to develop robust web applications but also make them more versatile as developers.

Working together on this project, the team members collaborated effectively to overcome the learning curve associated with Firebase. As a result, they successfully built the Movie App, a practical and efficient solution for movie enthusiasts, showcasing their adaptability and commitment to learning new technologies.

Limitations

After reviewing our movie app before the presentation, we have identified several limitations that we would like to improve. Firstly, we want to implement a feature that allows the user's login credentials to be remembered for a certain period, providing them with a more convenient experience.

Secondly, we would like to enhance the app's design to make it more visually appealing and user-friendly. Although we have made significant progress in this area, we still want to explore more design options to create a unique and attractive interface that stands out compared to the other group.

Thirdly, we are currently working on improving the way the app displays favorites. Instead of showing all favorites across the entire database, we are planning to assign favorites to individual users to ensure that only their favorites are displayed, making it easier for users to manage their preferences and improve the app's overall functionality.

Lastly, we have considered displaying the user's name on the navbar to indicate that they are logged in. However, it turned out to be more difficult than either of us thought with the login section as a whole and the idea was scrapped to get key functions done in time.

Overall, our movie app has several limitations that we planned to have to show in the presentation such as having it visually appealing, having the favorites functioning correctly, and having the user’s name shown at the top in the navbar.

Whats next

Ideally we would like to improve the look and stylization of our application. This would improve the UI and usability of our application, while also appealing to a larger user base. Things like adding an image for the movies saved to the favorites would be a nice touch that would make our app look more polished.

We are also interested in making our database more customizable allowing users to not only save favorites but maybe make their own custom lists.

We are also interested in continuously improving the security of our web application.

conclusion

In conclusion, the Movie App is a user-friendly and convenient web application that provides a seamless experience for movie enthusiasts. By integrating Firebase for authentication and data storage, as well as The Movie Database API for movie information, the app offers a secure and reliable platform for users to discover, search, and manage their favorite movies. With features like user authentication, top-rated movies, trending movies, search functionality, and favorites management, the Movie App is a comprehensive solution catering to the needs of movie lovers, making it an enjoyable and useful tool for exploring the world of cinema.