Academic Year	Module	Task
2025	5CS022/HJ1: Distributed and Cloud Systems Programming	4

Name: Swoyam Pokharel

ID: 2431342

Group: 26

Tutor: Ms. Jenny Rajak

Table Of Content:

Table Of Content: **2**
Question: **4**
Solution: **5**
Understanding The Question: 5
Initial Setup: 5
Screenshot of firebase console 5
Screenshot of creating a new project; filling information. 6
Screenshot of continuing project creation 6
Screenshot of Confirmation Of Project Creation 7
Screenshot of firebase console to register a new we app 8
Screenshot of Registering a new web app 8
Screenshot of the SDK Config 9
Navigating to FIrestore For Our Newly Registered App 9
Successfully created 11
Creating A New Collection 12
Document Structure 13
Project Structure: 14
Minimal Viable Code: 15
Initial boilerplate 15
Screenshot of all the pages using split view from my browser 15
User Authentication & Authorization: 16
Enabling Google And Email-Pass Authentication 18
I. User Signup: 19
Setting up Signup With Email & Pass: 19
Setting up Signing Up With Google: 20
Signing up with google: 22
Note: cross the emails **23**
Signing up with email: 24
II. User Login: 25
Setting Up Logging In With Email-Password: 26
Setting Up Logging In With Google: 26
Logging In WIth Google: 28
Logging In With Email-Password: 29
III. User Dashboard 30
Handling Unauthorized User 32
READ: Fetching User's Movies: 33
CREATE: Creating A New Movie: 34
Handling Image Uploads: 34
UPDATE: Updating an existing movie: 37
DELETE: Deleting a record: 38
Updated Code: 38
Index.html: 39
Updated Fetch Function: 40
Testing Create: 41
Testing Update: 43
Testing Delete: 46
Implementing Search & FIlters: 47
Testing Filters: 48
Creating Navigation Bar: 49
Finalized Code: 51
Updated Landing Page: 51
Updated Login Page: 53
Updated Signup Page 53
Updated Dashboard: 54

Question:

1. In a similar approach to Workshop on Google Firestore, create a “movies” review HTML/JavaScript web app, based on a Google Firestore NoSQL database. The web app should access the movies review data store in a Google Firestore NoSQL database via JavaScript and present the information via HTML on the web browser. The web app should have the ability to add the movie name, a rating score from 0 to 5 (integers only), the director’s name and the release date. It should have the ability to be sorted in order on any of the fields. It should have the ability to edit and delete individual reviews. There is no requirement for any user authentication for the web app. Your submission should be a zip of your HTML and JavaScript files and a well documented MS Word file containing your name, student ID, explanation and screenshots of your web browser showing the reviews, and screenshot of the Firestore database.

This task will contribute 30% of the marks to the Portfolio.

Solution:

Understanding The Question:

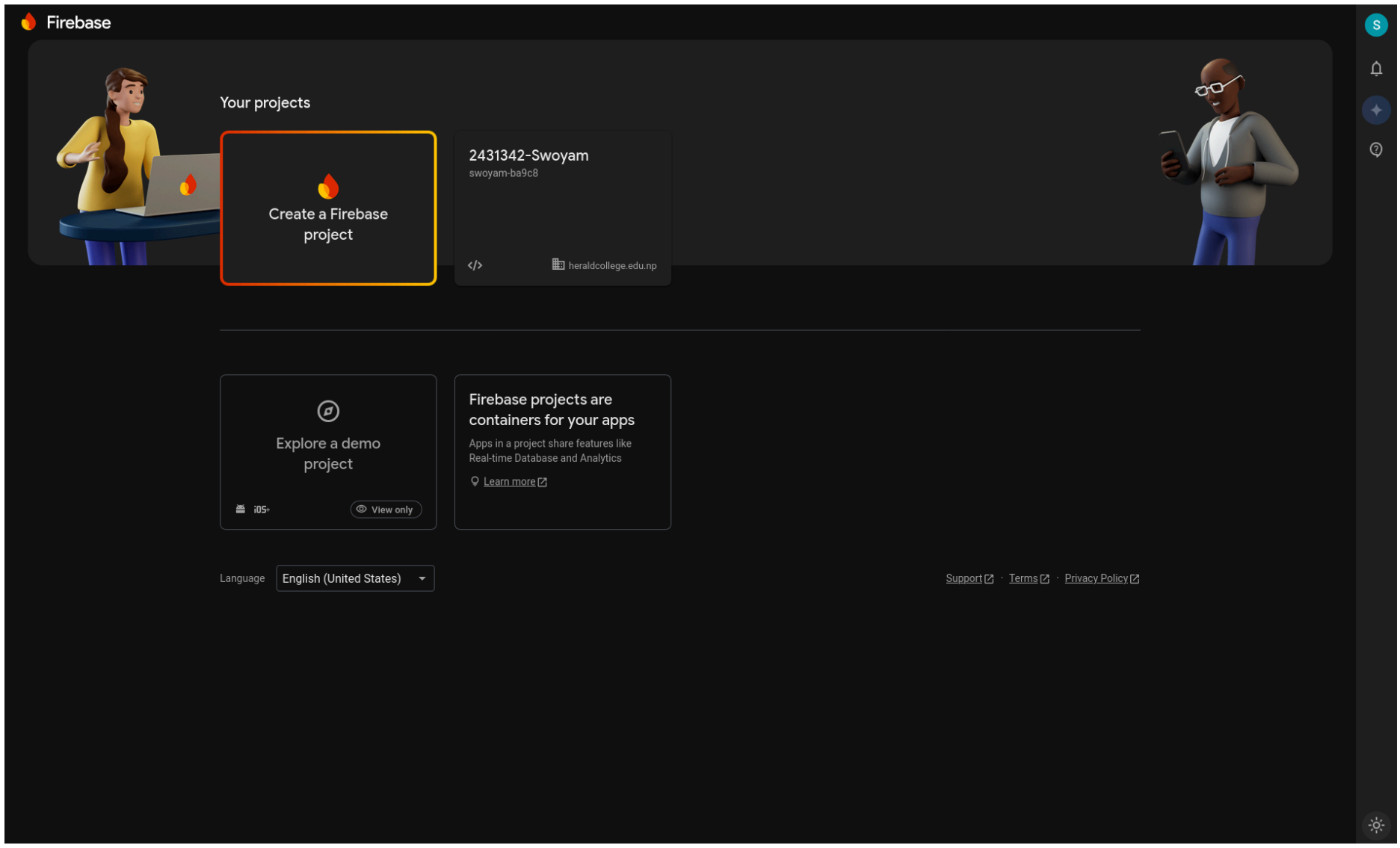
So, the question wants us to create a movie web app allowing the user to store movie details such as:

- Name
- Release Date
- Rating
- Director's Name

The task wants us to have full CRUD operations available and add filtering.Although the task doesn't have any requirement for user authentication, I have implemented authentication as it seemed like a perfect opportunity to explore firebase auth too.

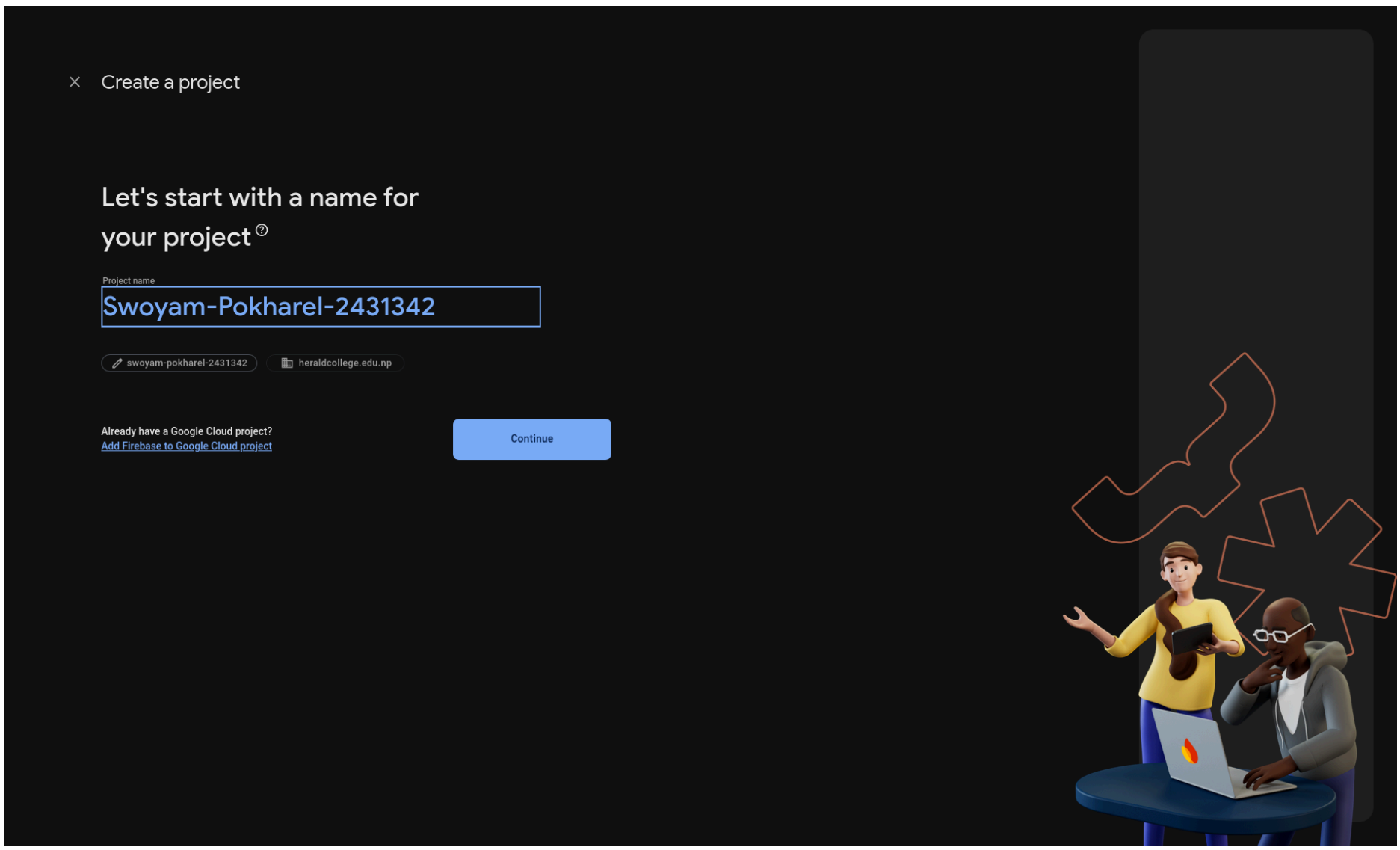
Initial Setup:

Lets first kick things off creating a new firebase project



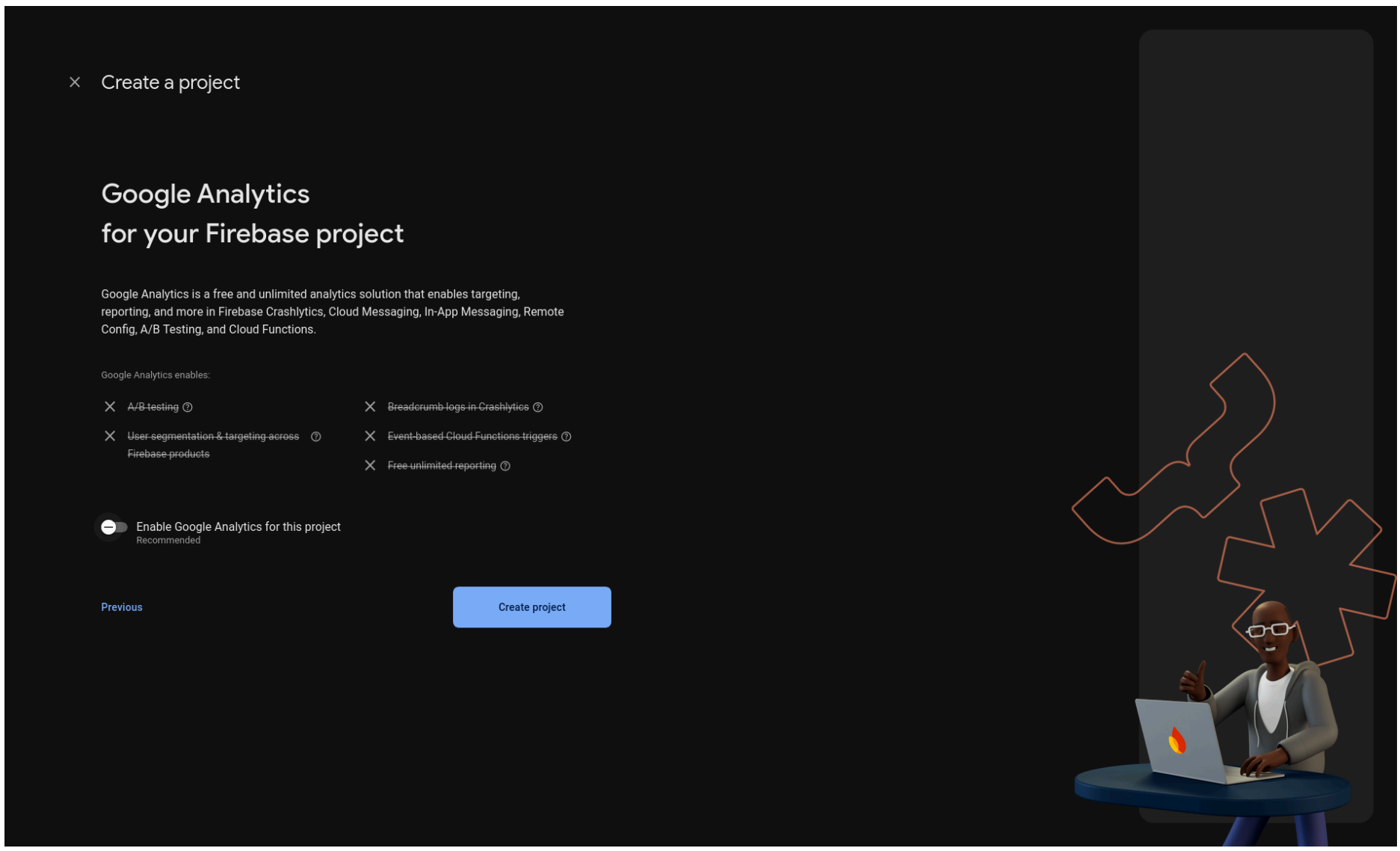
Screenshot of firebase console

Here, I already have an existing projects, which I will leave as is and create a new project:



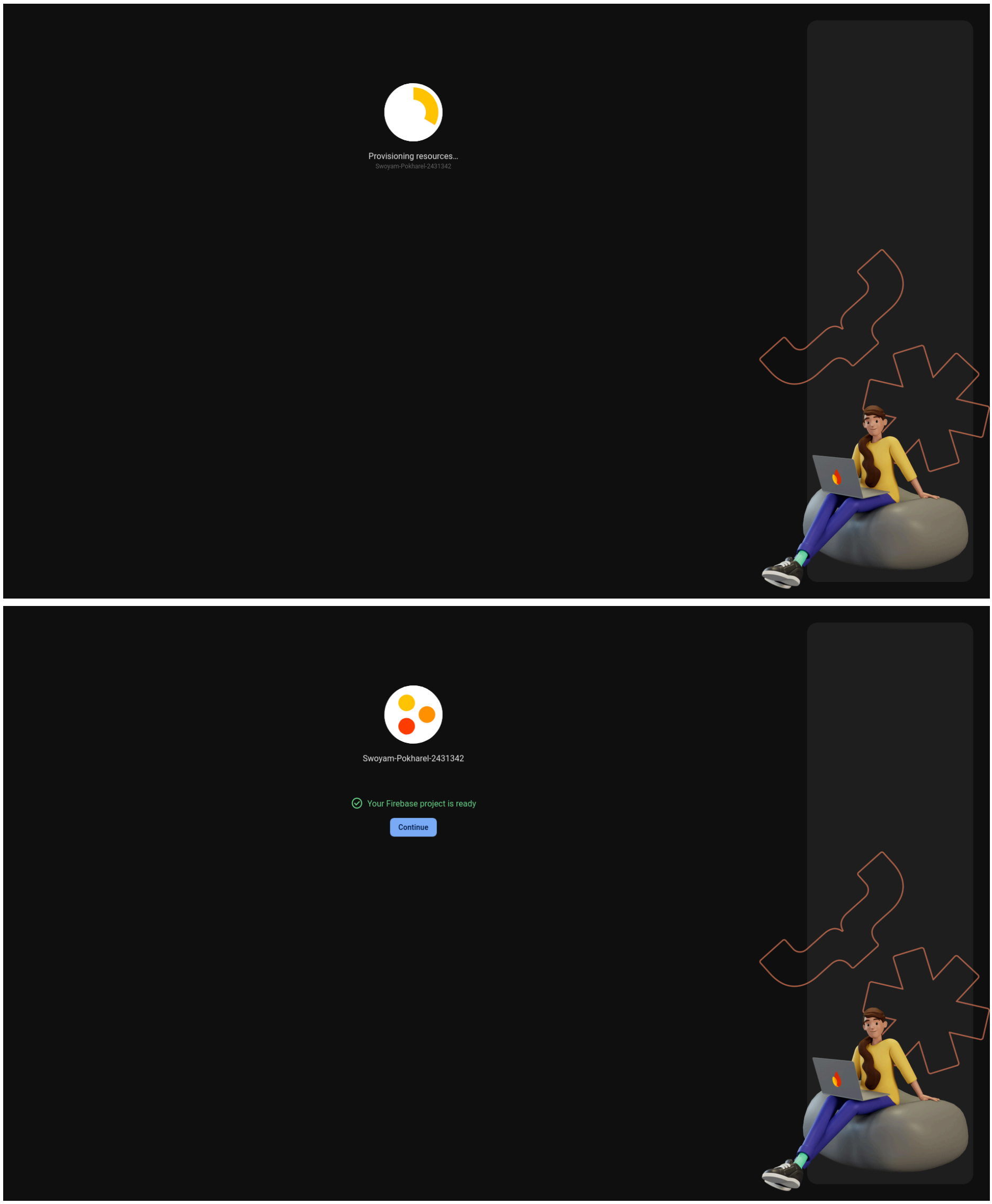
Screenshot of creating a new project; filling information.

I've named my project based on my studentID. I selected the parent group to be that of `heraldcollege.edu.np` as our emails are registered there. We now proceed with the creation:



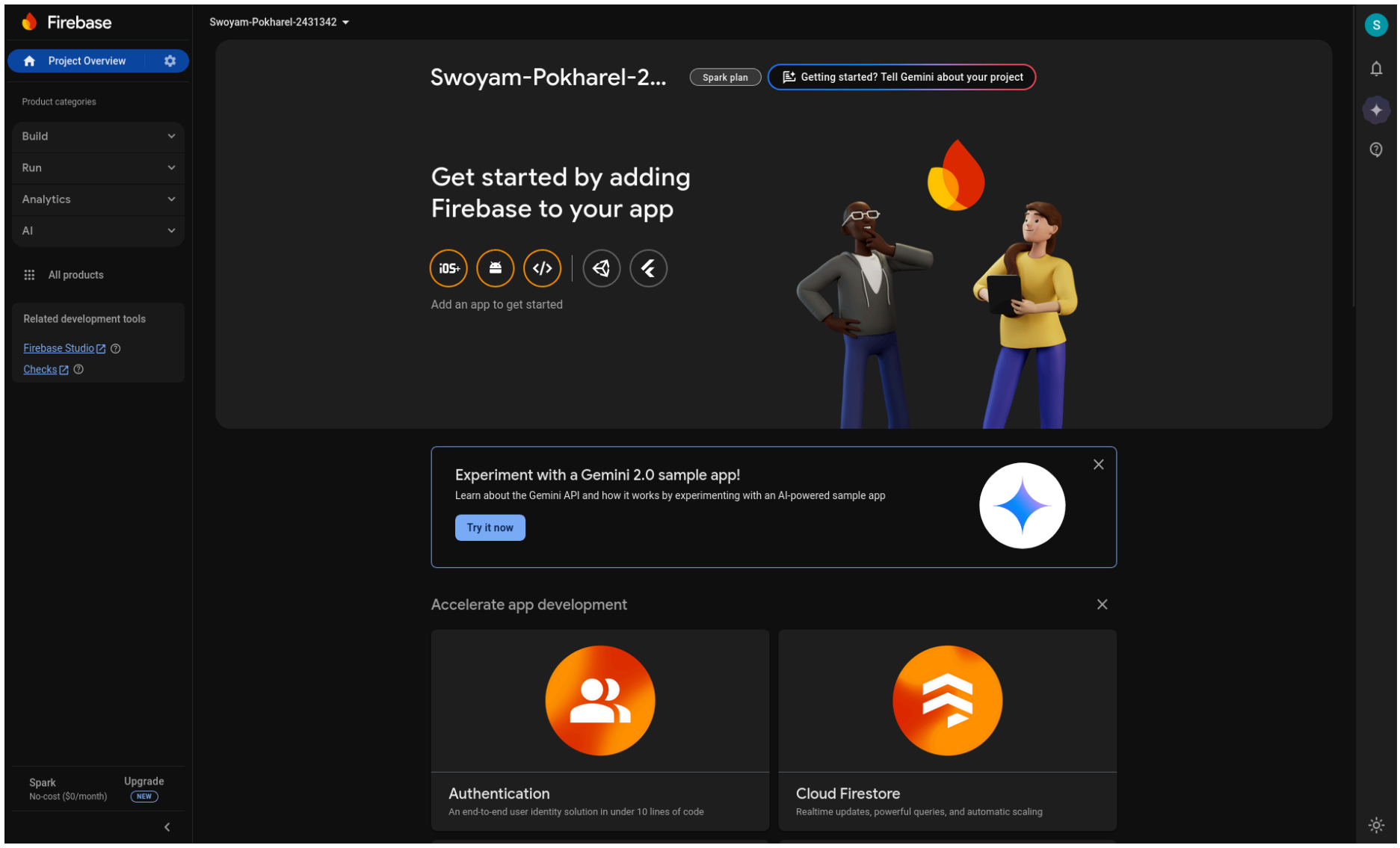
Screenshot of continuing project creation

Here, we uncheck the Google Analytics as we won't be needing since this is not a project aimed at the public. Now, we finish the creation of this new project:

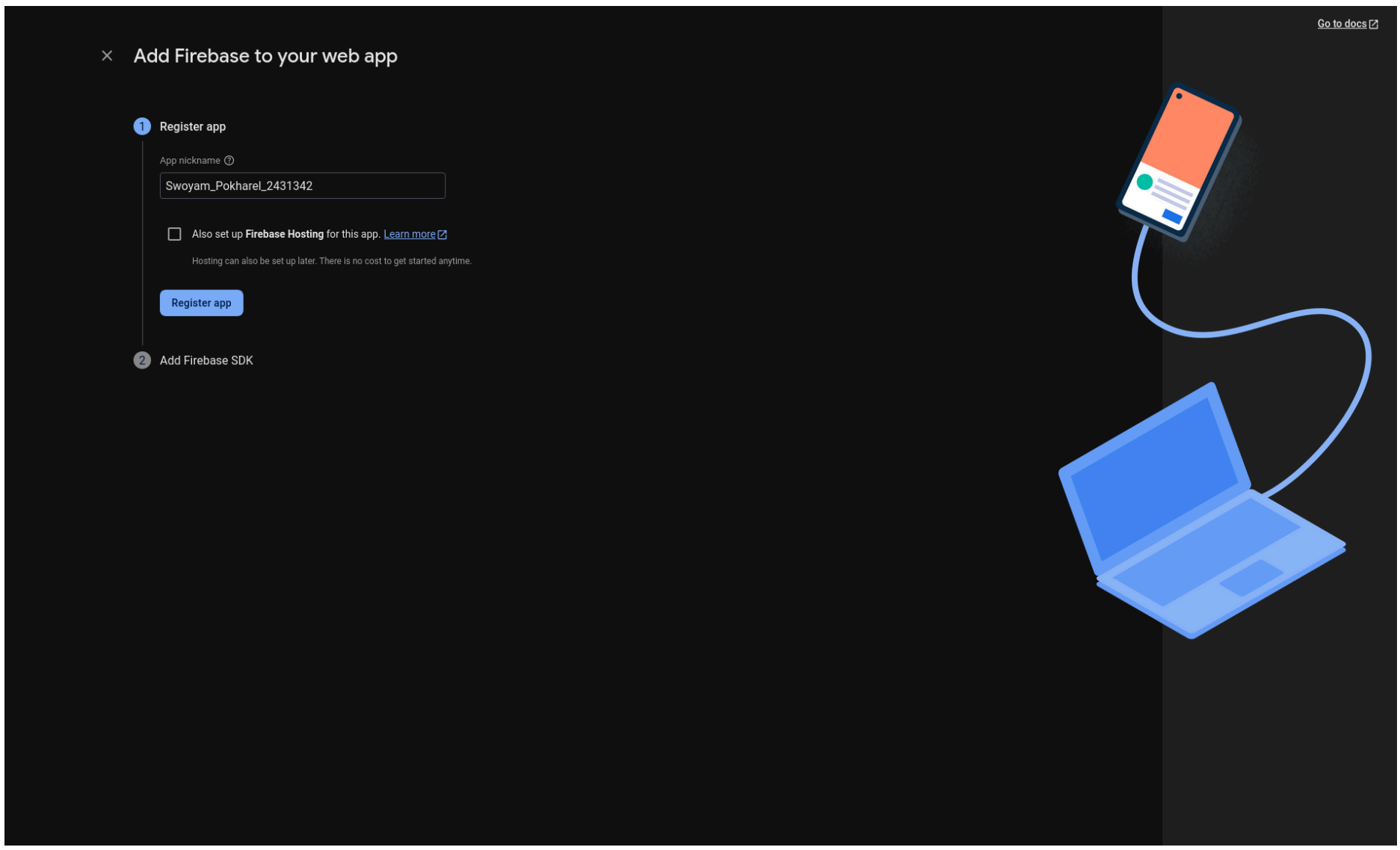


Screenshot of Confirmation Of Project Creation

Now that our project has been created, let’s register our web app and get the keys for firebase. To do this, we navigate to the firebase console again and start with a new webapp.

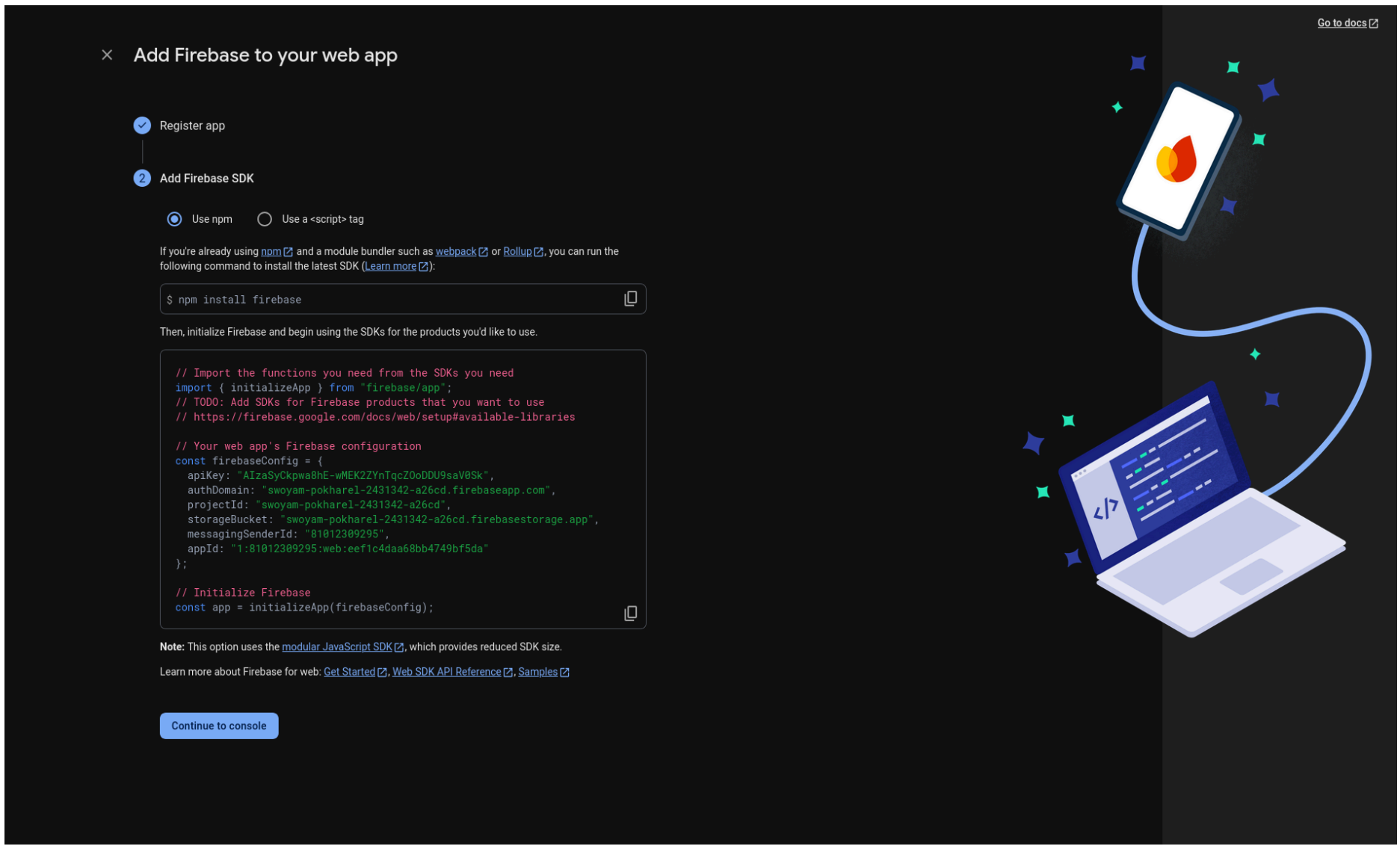


Screenshot of firebase console to register a new we app



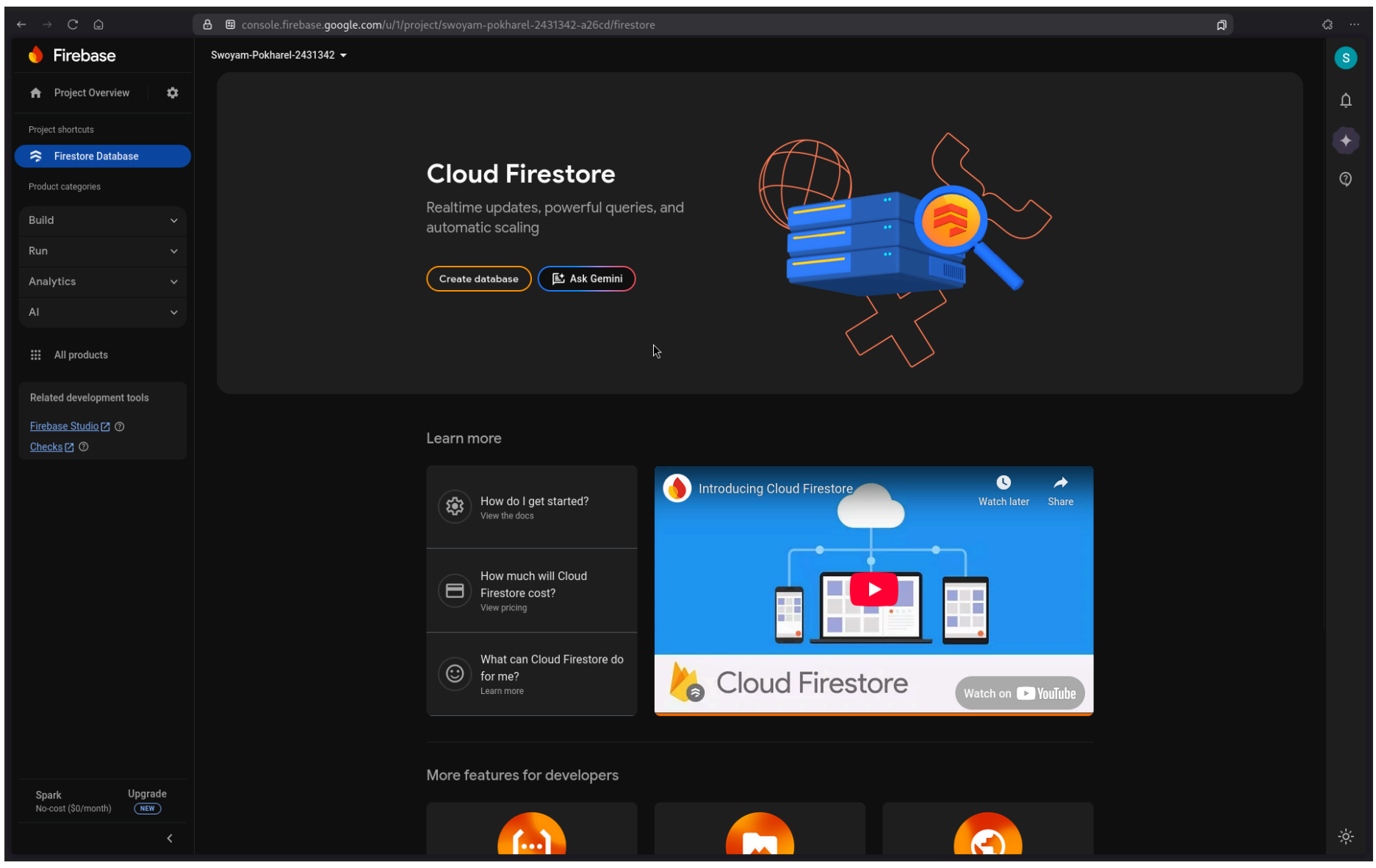
Screenshot of Registering a new web app

Here, I specify the name to be `Swoyam_Pokharel_2431342` on the basis of my studentID and I've opted out of Firebase for now as the task doesn't require hosting it.



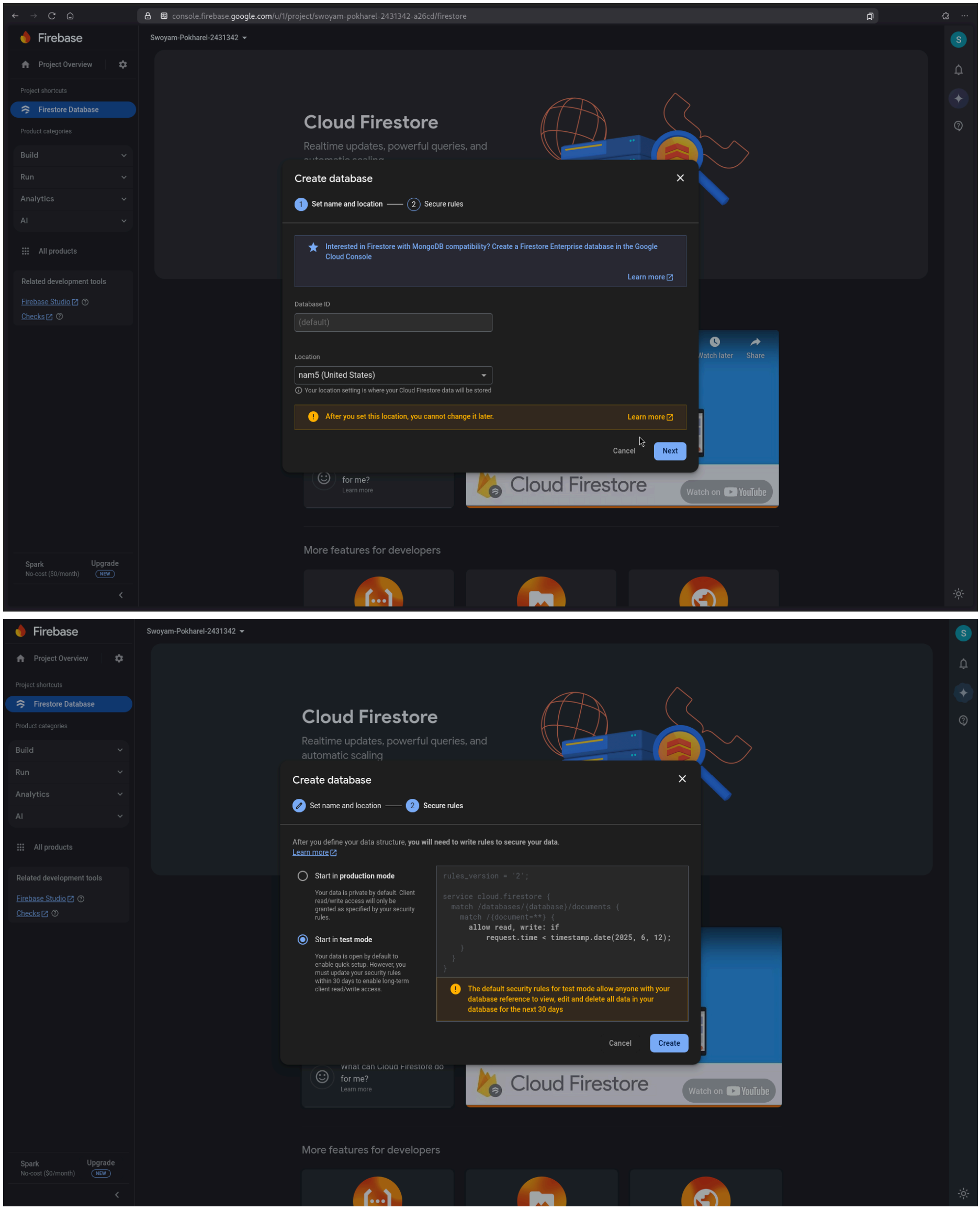
Screenshot of the SDK Config

Now firebase gives us our config for the `Firebase SDK`, which I will be saving in a file for future use:
Now, I will create a new firestore for this particular app that we just registered.

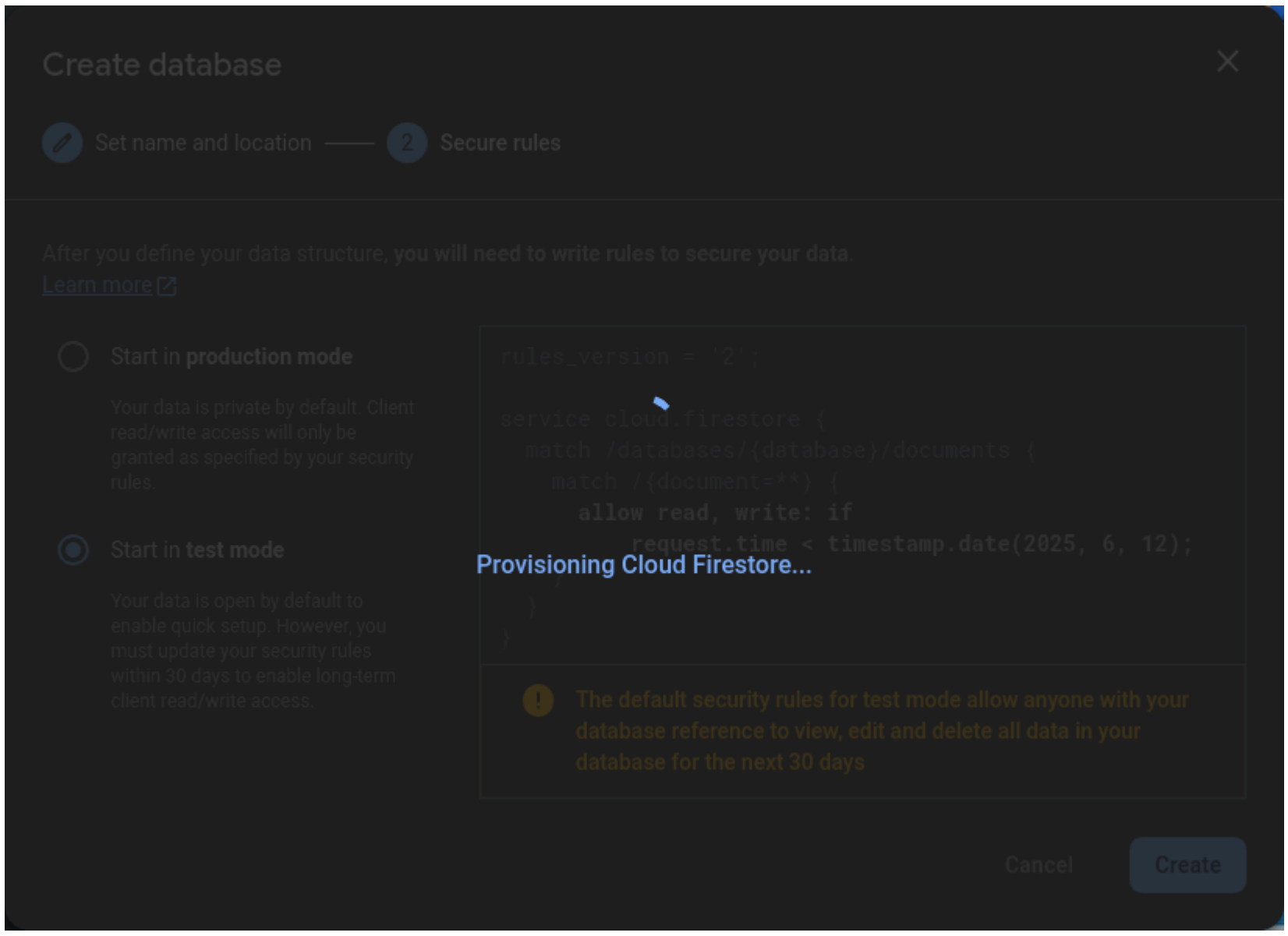


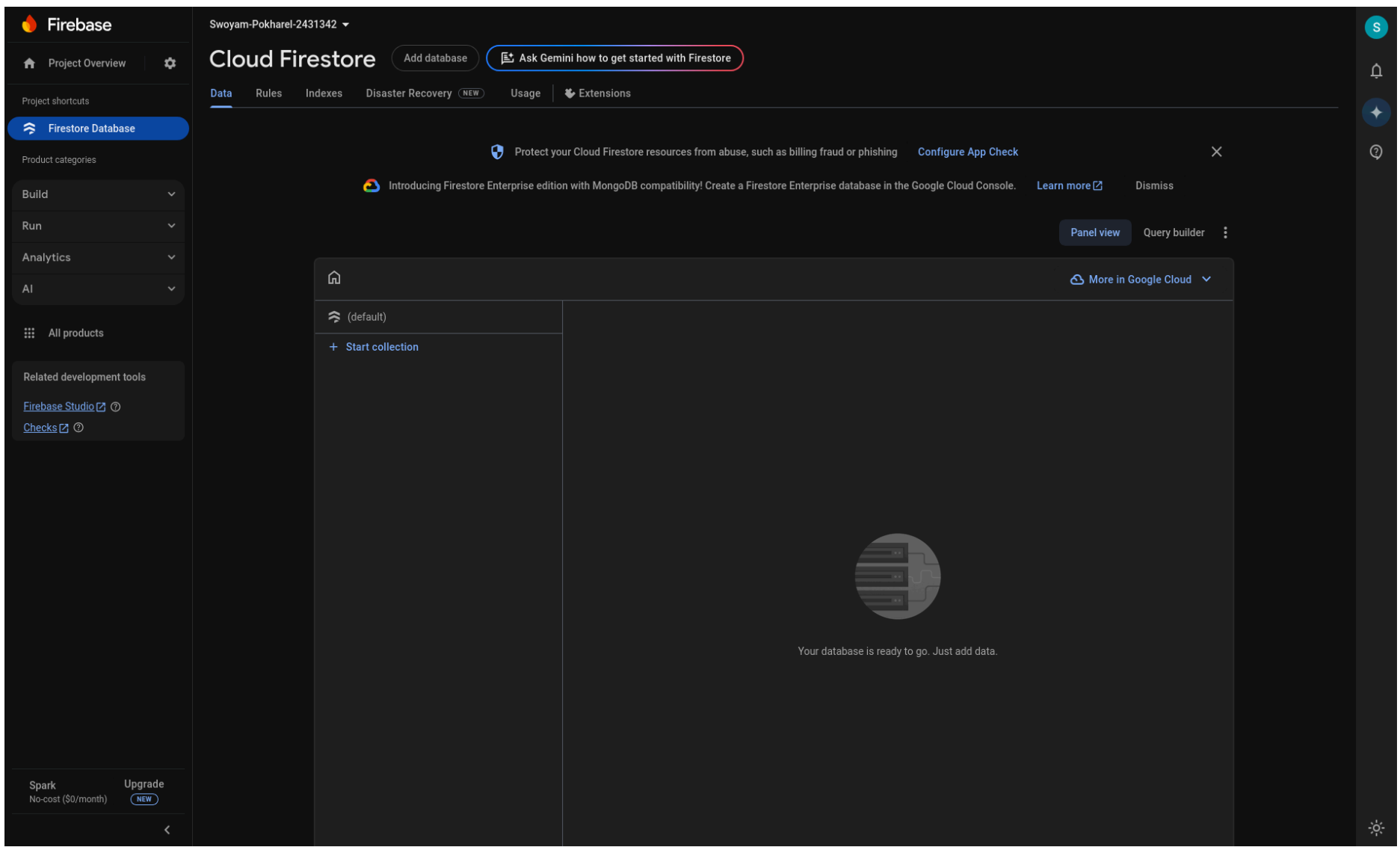
Navigating to FIrestore For Our Newly Registered App

Here, I click on `Create A Database` upon which i get prompted with:


console.firebase.google.com/u/1/project/swoyam-pokharel-2431342-a26cd/firestore
Firebase
Swoyam-Pokharel-2431342
Project Overview
Project shortcuts
Firestore Database
Product categories
Build
Run
Analytics
AI
All products
Related development tools
Firebase Studio
Checks
Spark
No-cost ($0/month)
Upgrade
NEW
Cloud Firestore
Realtime updates, powerful queries, and automatic scaling
Create database
1 Set name and location
2 Secure rules
Interested in Firestore with MongoDB compatibility? Create a Firestore Enterprise database in the Google Cloud Console
Learn more
Database ID
(default)
Location
nam5 (United States)
Your location setting is where your Cloud Firestore data will be stored
After you set this location, you cannot change it later.
Learn more
Cancel
Next
for me?
Learn more
Watch later
Share
Cloud Firestore
Watch on YouTube
More features for developers
Create database
Set name and location
2 Secure rules
After you define your data structure, you will need to write rules to secure your data.
Learn more
Start in production mode
Your data is private by default. Client read/write access will only be granted as specified by your security rules.
Start in test mode
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.
rules_version = '2';
service cloud.firestore {
match /databases/{database}/documents {
match /{document=**} {
allow read, write: if
request.time < timestamp.date(2025, 6, 12);
}
}
}
The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days
Cancel
Create
What can Cloud Firestore do for me?
Learn more

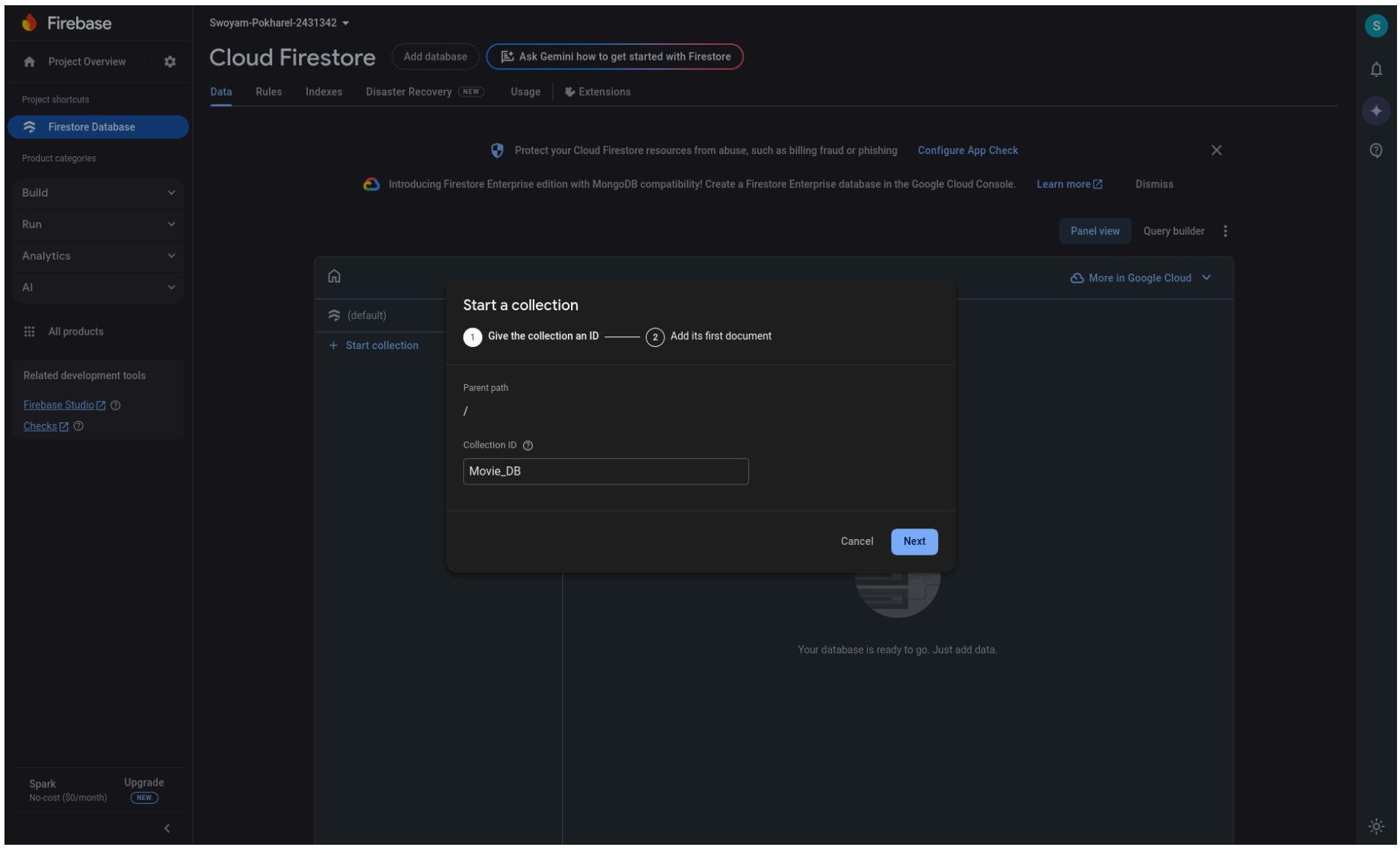
Here firebase is essentially asking what mode we want to start firebase with. For now, I set it to test mode as its easier local development and testing.





Successfully created

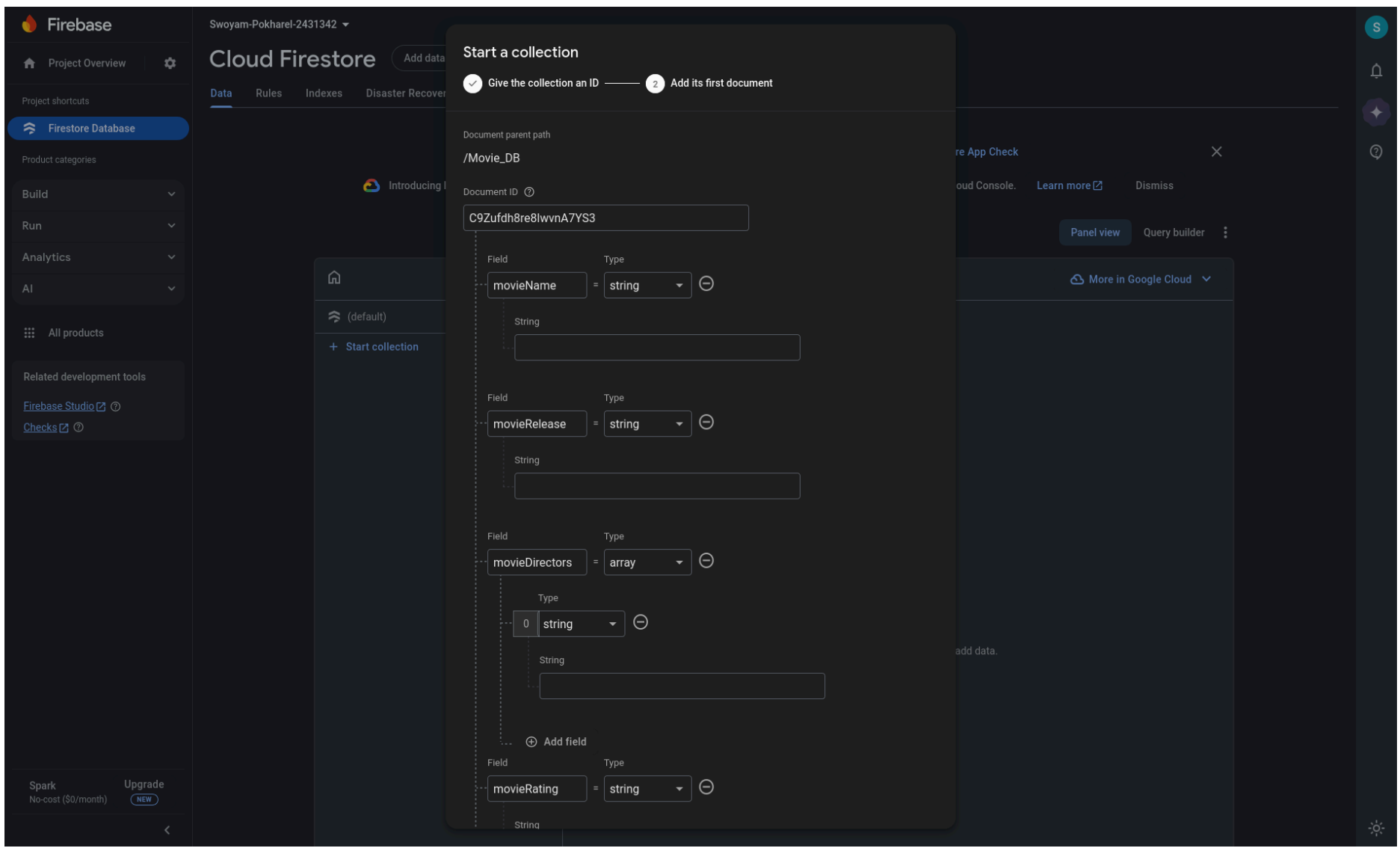
Now that our database is successfully created, I proceed to create a new collection:

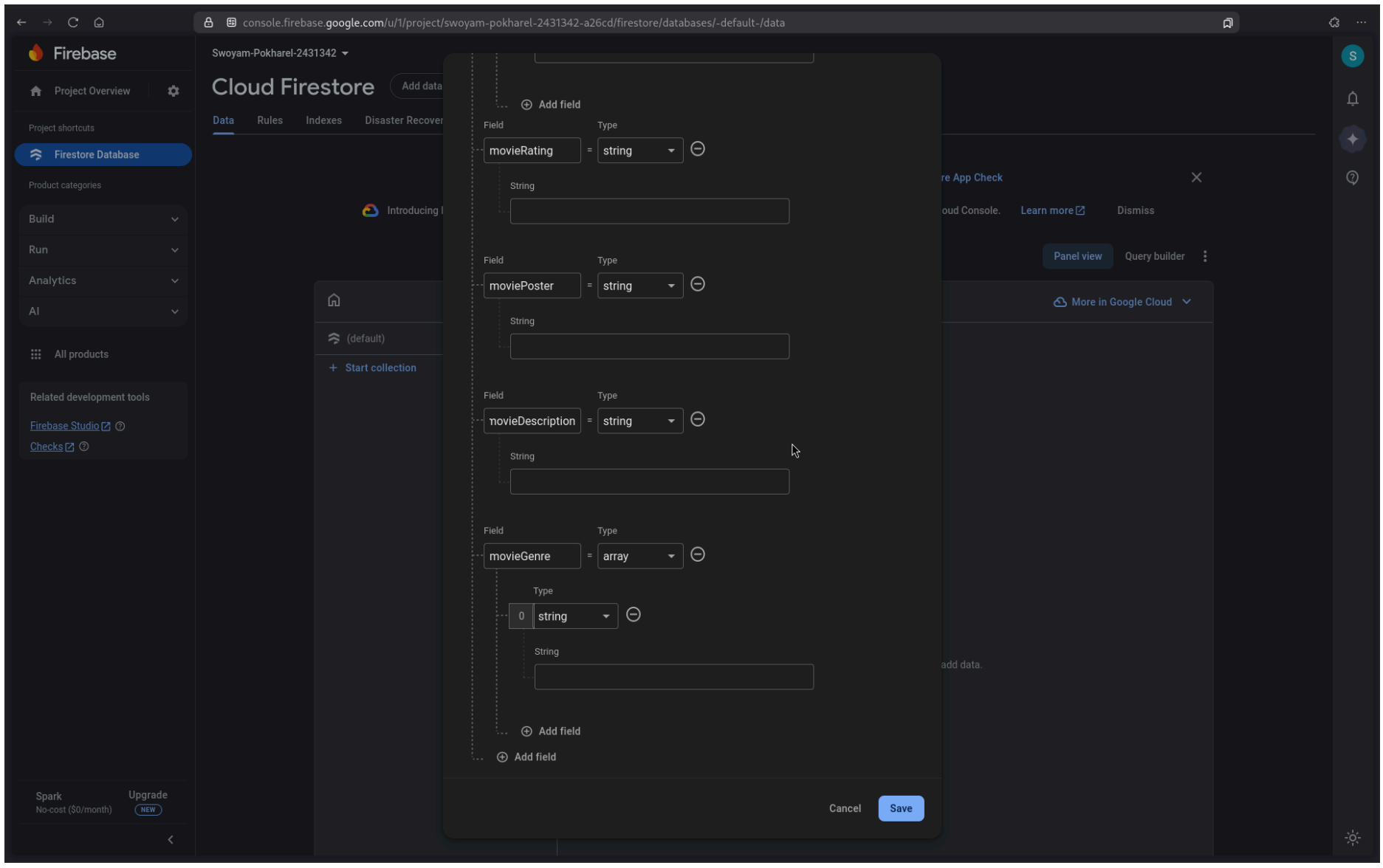


Creating A New Collection

I've named the collection `Movie_DB` and for ease of testing; I'll manually insert a document in this collection so that I have data that shows up when I initially fetch in my javascript.

Although the task doesn't specify images, I want a way where a user can have a poster and perhaps a description and genre too. So, my final structure for a document will look like this:





Document Structure

I've finalized the movie schema with simplicity in mind:

- `movieName` : a plain string since it's just a title.
- `movieRating` : a number (which will be limited to 1 to 5 from the frontend)
- `movieRelease`: stored as a string to avoid Firebase timestamp-to-JavaScript conversion hassles as precision for a movie release date isn't critical
- `movieDirectors`: an array, to handle cases with multiple directors.
- `moviePoster`: a public link to the image uploaded by the user, stored in Firebase Storage.
- `movieDescription`: just a string for a short summary of the movie.
- `movieGenre`: An array.

With the foundational structure in place, I manually created a document so that I have data to work with when I fetch from javascript.

This concludes the initial setup. Now I will proceed to move on to the code, but we will again come back to more setup for file uploads and setting up firebase authentication.

Project Structure:

```
[wizard@archlinux task4]$ tree
.
├── dashboard.html
├── index.html
├── login.html
├── signup.html
└── static
    ├── input.css
    └── js
        ├── dashboard.js
        ├── login.js
        └── signup.js

3 directories, 8 files
[wizard@archlinux task4]$
```

```
[0] 0:bash*
```

Since I will be needing a landing page, a login , a signup and a dashboard for the logged in user, I've made 4 files. The static folder will just hold the css for now which will be the input to `tailwindcss` binary just like in task 2. The `static/js` will hold the javascript files relevant to each of the pages. Later when I setup hosting, all of these files will be cloned in a `public/` directory.

```
[wizard@archlinux task4]$ ls
dashboard.html  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ bunx serve

  Serving!

  - Local:    http://localhost:3000
  - Network:  http://192.168.1.8:3000

  Copied local address to clipboard!
```

```
[wizard@archlinux task4]$ ls
dashboard.html  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ cd static/
[wizard@archlinux static]$ ls
input.css  js/
[wizard@archlinux static]$ cd ..
[wizard@archlinux task4]$ ls
dashboard.html  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ tailwindcss -i static/input.css -o static/output.css --watch
/*! 🌼 daisyUI 5.0.28 */
sh: line 1: watchman: command not found
≈ tailwindcss v4.1.4

Done in 152ms
Done in 404µs
Done in 398µs
```

```
[0] 0:nvim- 1:tailwindcss*
```

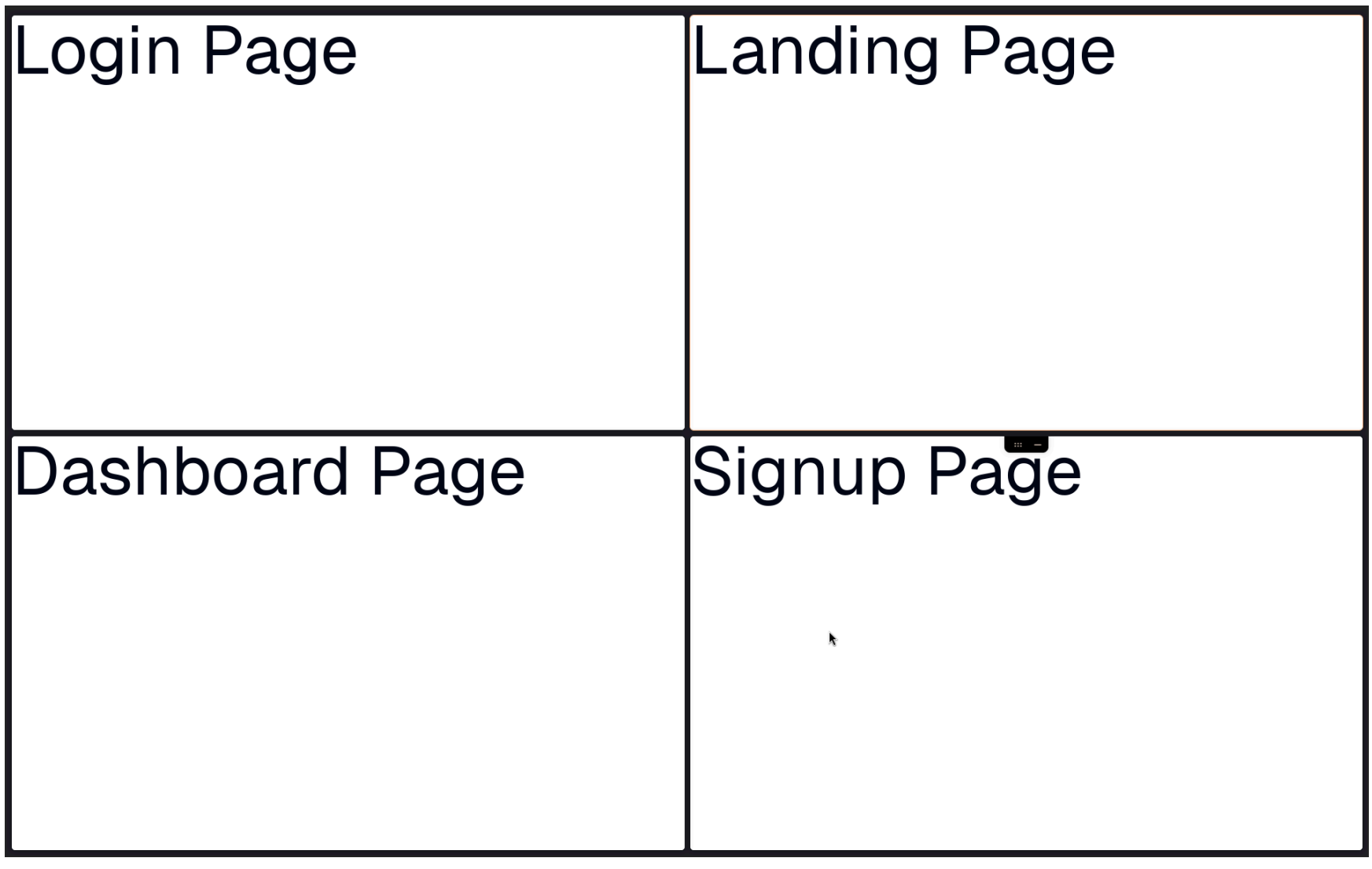
For development, I'm using `bun` as my server and `tailwindcss` on watch mode for convenient styling

Minimal Viable Code:

I will first create a minimal viable code that has all the functionality, and won't concern myself too much with styling and animations. Initially, I've filled all the relevant with basic boilerplate html code shown below:

Initial boilerplate

This allows me to visually distinguish between the pages and provides me with a clear structure to work with. I've also linked the relevant javascript for each of the pages. Here is how each one looks as of now:

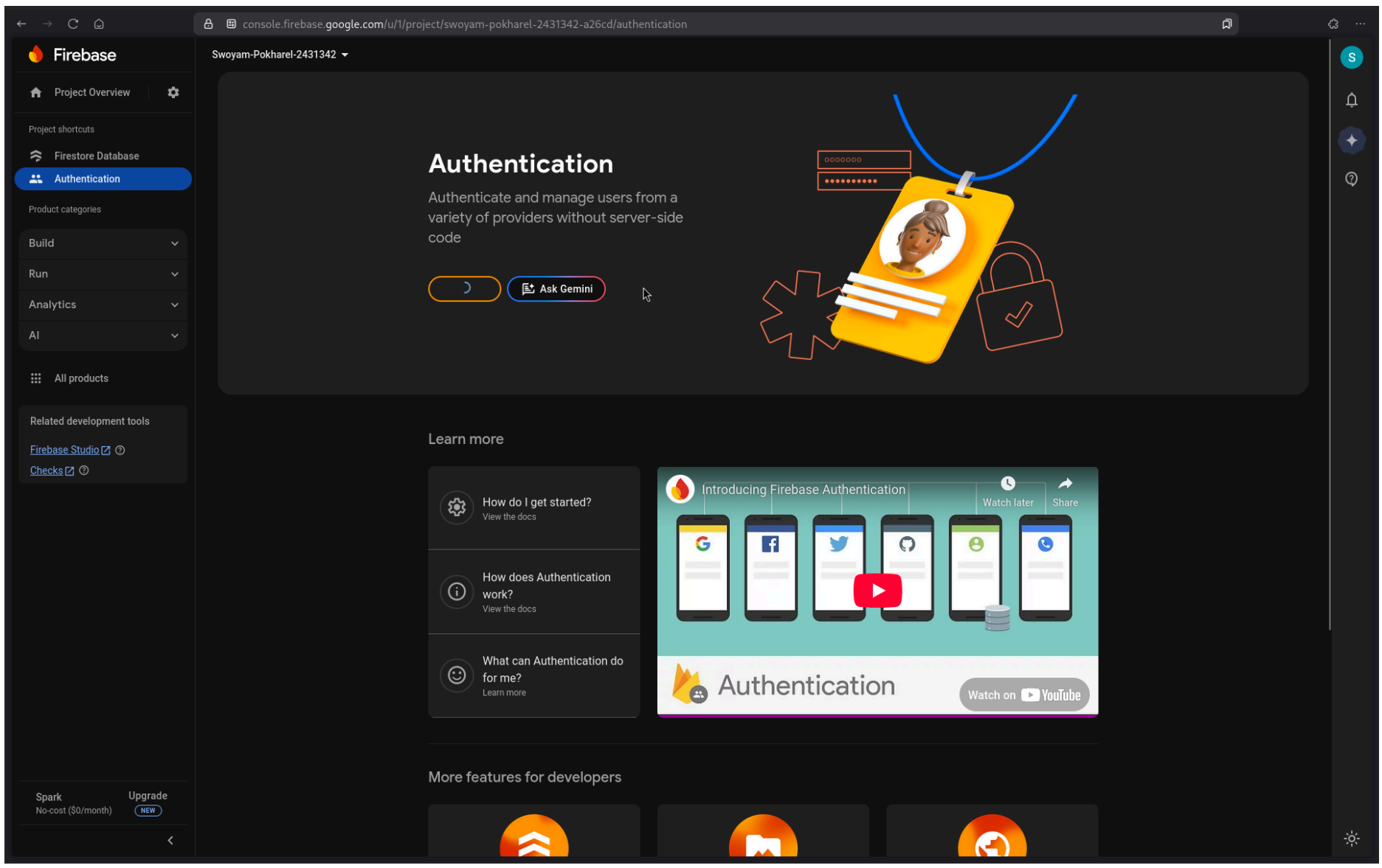


Screenshot of all the pages using split view from my browser

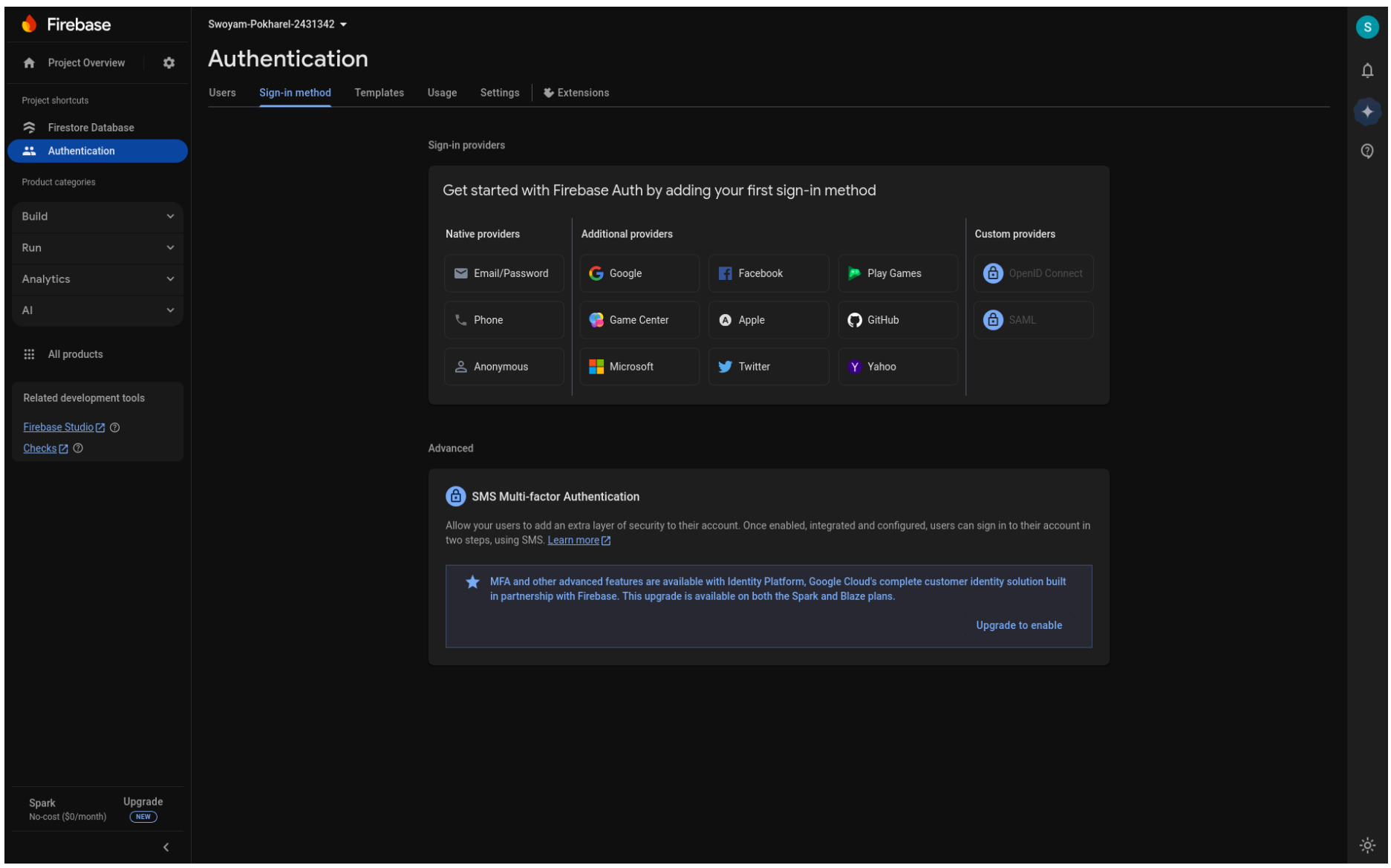
For now, I'll leave the landing page as is as it's not essential for the functionality of the app. I will first be tackling authentication.

User Authentication & Authorization:

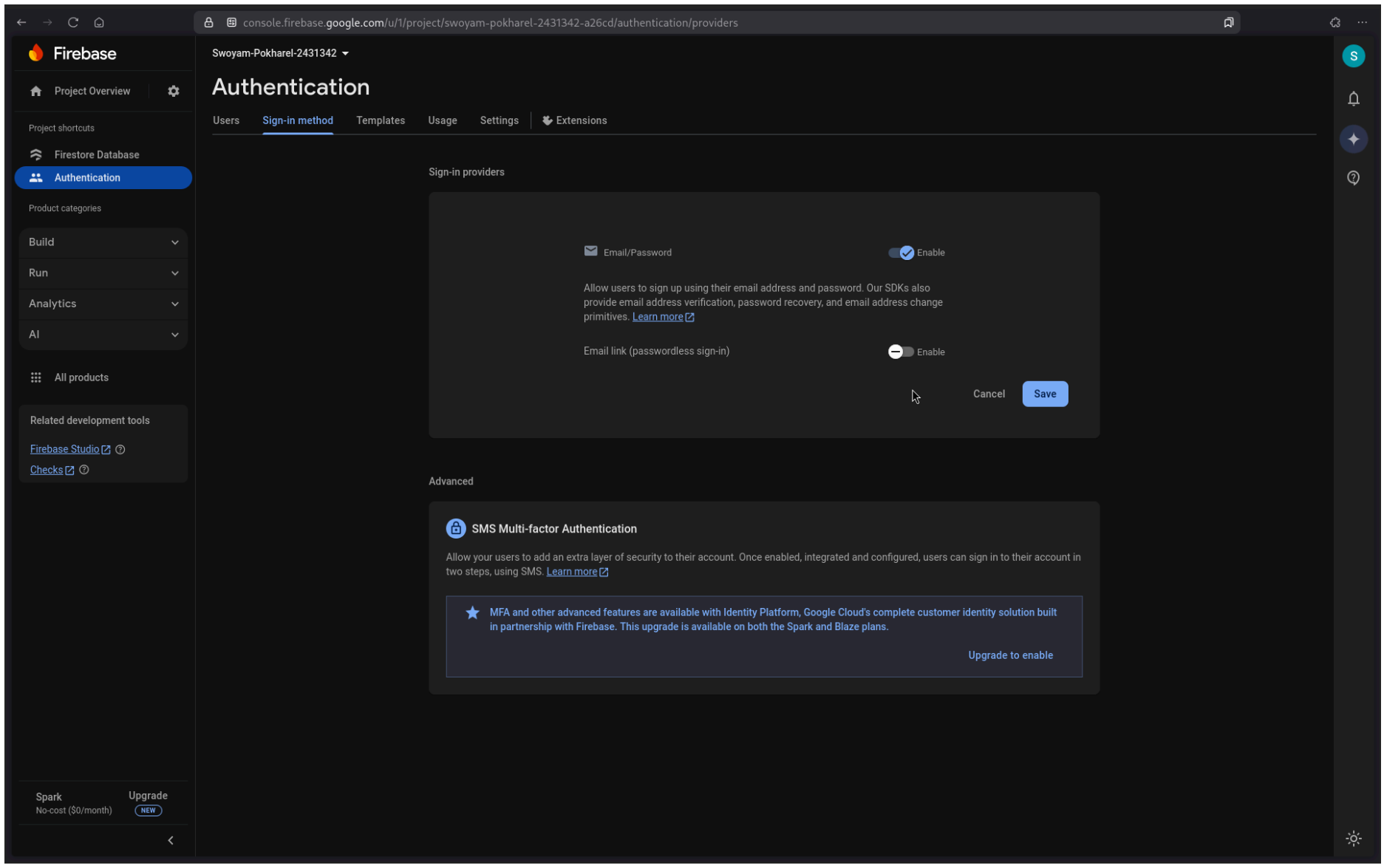
I first set up Authentication in firebase, I navigate to the Authentication section and click on get started

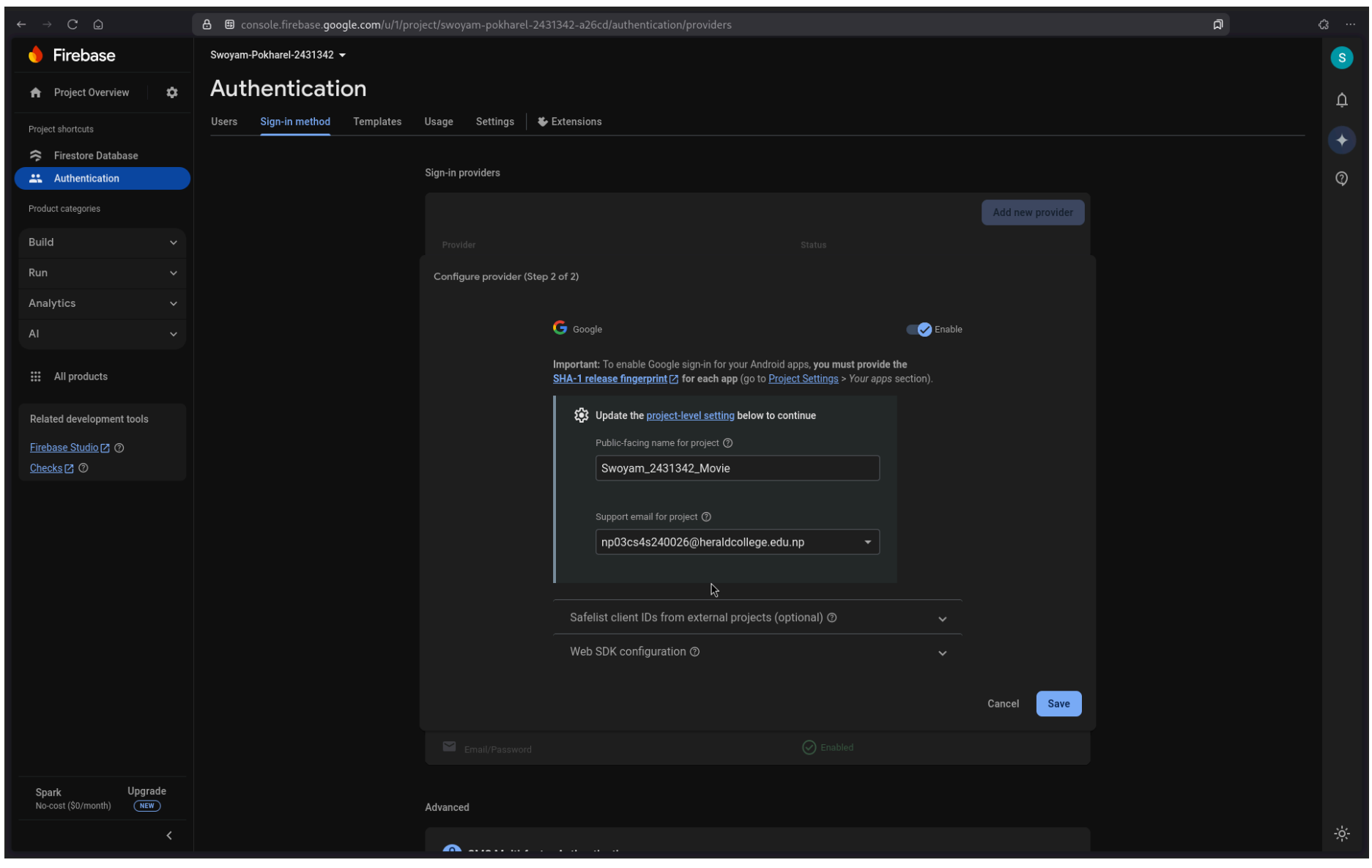


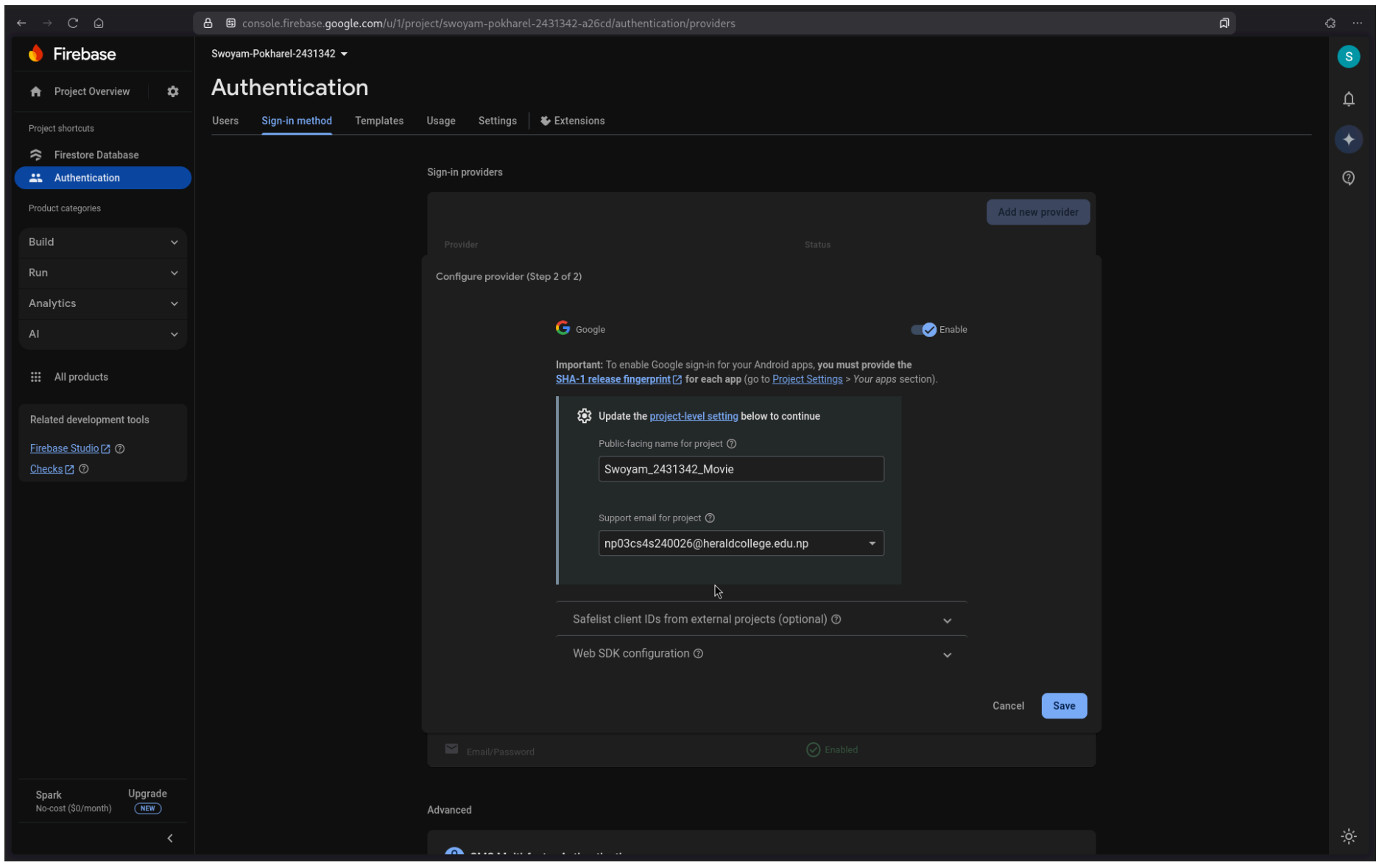
after which I'm redirected to the following page:



From this page, I enable `email/password` authentication along with `Google` authentication.







Enabling Google And Email-Pass Authentication

That concludes the setup needed for google authentication. Now I proceed to the code.

I. User Signup:

Setting up Signup With Email & Pass:

For a user to login, they must first be able to sign up. Google's Firebase SDK makes this super simple, we first need to our firebaseConfiguration:

```
1  import { initializeApp } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-app.js";
 1
 2 const firebaseConfig = {
 3     apiKey: "AIzaSyCkpwa8hE-wMEK2ZYnTqcZOoDDU9saV0Sk",
 4     authDomain: "swoyam-pokharel-2431342-a26cd.firebaseapp.com",
 5     projectId: "swoyam-pokharel-2431342-a26cd",
 6     storageBucket: "swoyam-pokharel-2431342-a26cd.firebasestorage.app",
 7     messagingSenderId: "81012309295",
 8     appId: "1:81012309295:web:eef1c4daa68bb4749bf5da"
 9 };
10
11 const app = initializeApp(firebaseConfig);
```

```
firebaseconfig.js
"firebaseconfig.js" 12L, 496B
[0] 0:nvim* 1:tailwindcss-
```

Then, I pull in the relevant functions `getAuth`, `createUserWithEmailAndPassword`, `GoogleAuthProvider` and `signInWithPopup` from firebase's auth sdk.

```
 6 import { initializeApp } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-app.js";
 5 import {
 4 | getAuth,
 3 | createUserWithEmailAndPassword,
 2 | GoogleAuthProvider,
 1 | signInWithPopup,
 7 } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-auth.js";
 1
 2 const firebaseConfig = {
 3     apiKey: "AIzaSyArjiO-_4w_X7HGdNyhPilYWd0qeLjbbzU",
 4     authDomain: "swoyam-pokharel-2431342.firebaseapp.com",
 5     projectId: "swoyam-pokharel-2431342",
 6     storageBucket: "swoyam-pokharel-2431342.firebasestorage.app",
 7     messagingSenderId: "600290624989",
 8     appId: "1:600290624989:web:7cade8f54135d672d72234"
 9
10 };
11
12
13 const app = initializeApp(firebaseConfig);
14
```

```
static/js/signup.js [+]
1 change; after #26  09:36:52
[0] 0:nvim* 1:tailwindcss-
```

After which, I create a function signup using email and password:

```
 9
 8 const auth = getAuth();
 7
 6 // Email/Password Signup
 5 window.signupWithEmail = async function () {
 4 |   const email = document.getElementById("email").value;
 3 |   const password = document.getElementById("password").value;
 2 |
 1 |   try {
30 |       const userCred = await createUserWithEmailAndPassword(auth, email, password);
 1 |       alert("Signup successful");
 2 |       console.log("User:", userCred.user);
 3 |   } catch (err) {
 4 |       alert("Signup failed: " + err.message);
 5 |   }
 6 };
 7
```

Below is the same code, presented with carbon for readability:

```
const auth = getAuth();

// Email/Password Signup
window.signupWithEmail = async function () {
    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value;

    try {
        const userCred = await createUserWithEmailAndPassword(auth, email, password);
        alert("Signup successful");
        console.log("User:", userCred.user);
    } catch (err) {
        alert("Signup failed: " + err.message);
    }
};
```

The code simply uses the `DOM` to get values for `email` and `password` and calls the `createUserWithEmailAndPassword` function provided by the firebase SDK. That's all for signing up with email and password, now I proceed to allow signing up with google:

Setting up Signing Up With Google:

To signup with google, its equally as simple:

```
10 const googleProvider = new GoogleAuthProvider();
 9 window.signupWithGoogle = async function () {
 8 |   try {
 7 |       const result = await signInWithPopup(auth, googleProvider);
 6 |       const user = result.user;
 5 |       alert("Signed in with Google");
 4 |       console.log(user);
 3 |   } catch (err) {
 2 |       alert("Google signup failed: " + err.message);
 1 |   }
48 };
static/js/signup.js
"static/js/signup.js" 48L, 1502B
[0] 0:nvim* 1:tailwindcss-
```

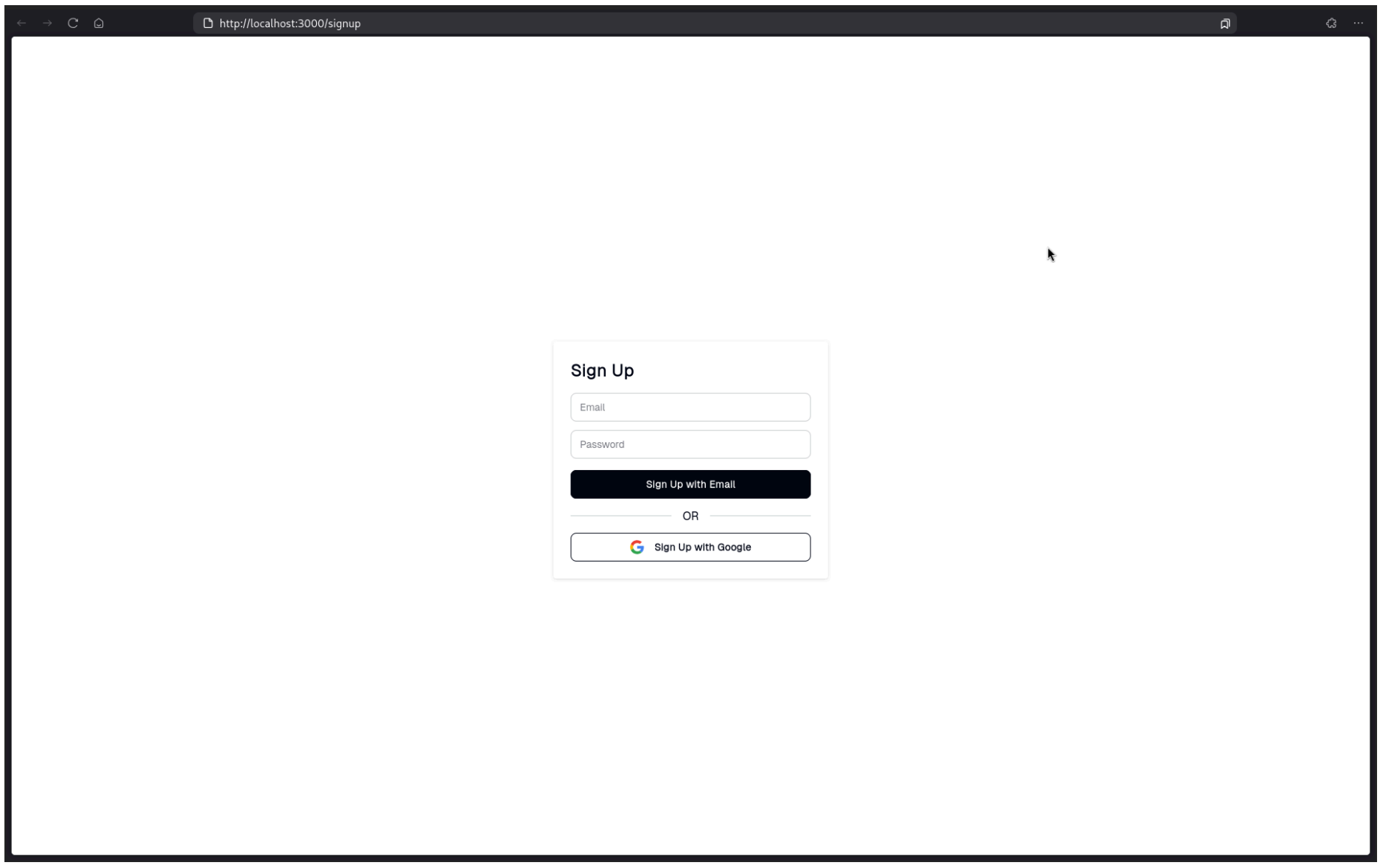
Below is the same code, presented with carbon for readability:

```
const googleProvider = new GoogleAuthProvider();
window.signupWithGoogle = async function () {
    try {
        const result = await signInWithPopup(auth, googleProvider);
        const user = result.user;
        alert("Signed in with Google");
        console.log(user);
    } catch (err) {
        alert("Google signup failed: " + err.message);
    }
};
```

Here, we simply call the `signInWithPopup` that opens a new window and allows the user to select an account to sign up with, and that's it for the javascript side of things – Firebase SDK makes it super convenient. Now I've updated the `signup.html` to include inputs for email and password and and option to continue with google:

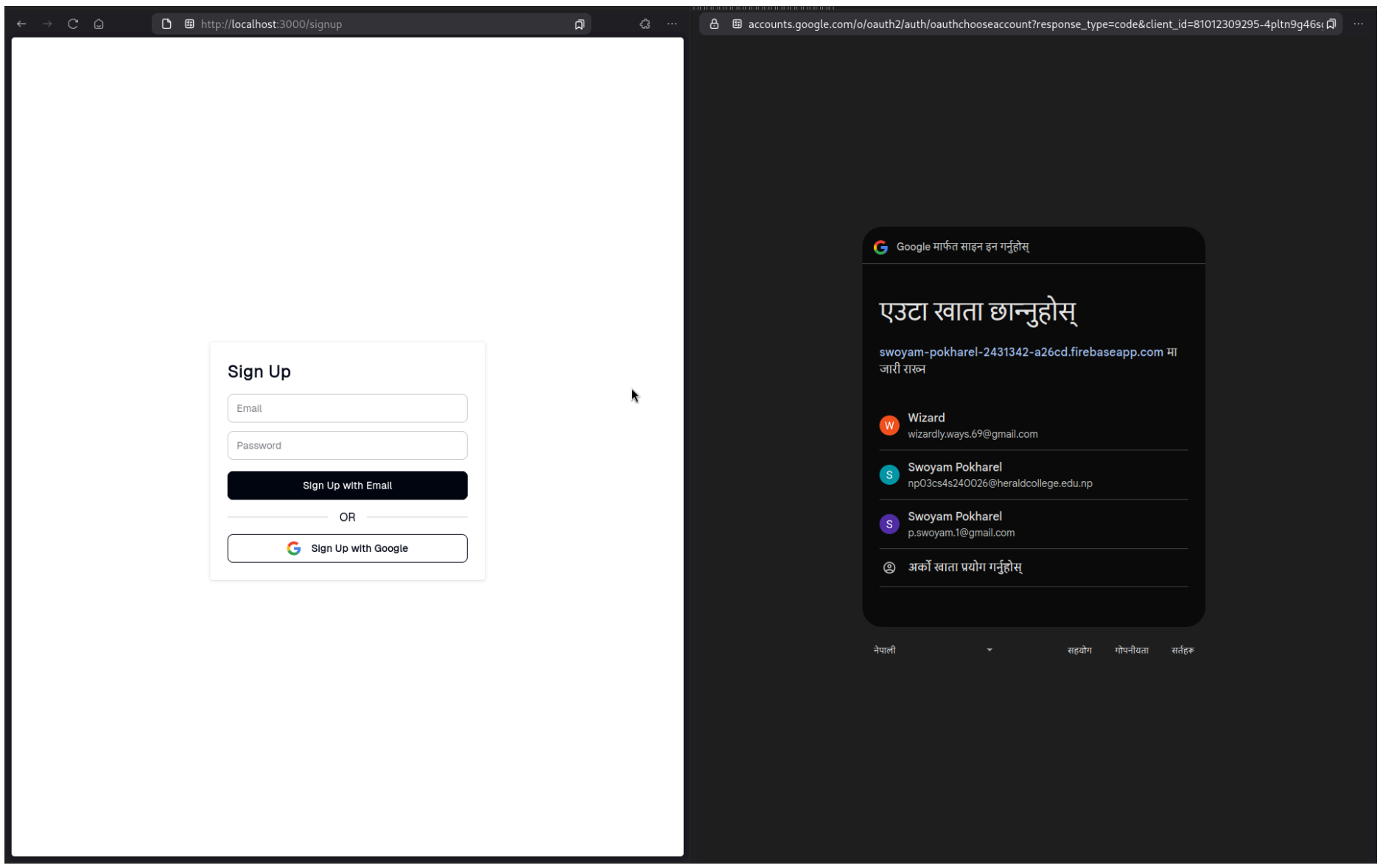
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="/static/output.css" />
        <title>2431342 Signup</title>
    </head>
    <body class="h-screen flex justify-center items-center flex-col">
    |   <div class="max-w-sm mt-10 p-6 bg-white shadow rounded">
    |       <h2 class="text-2xl font-bold mb-4">Sign Up</h2>
    |
    |       <!-- Email and Password Form -->
    |       <input id="email" type="email" placeholder="Email" class="input input-bordered w-full mb-3" />
    |       <input id="password" type="password" placeholder="Password" class="input input-bordered w-full mb-4" />
    |       <button onclick="signupWithEmail()" class="btn btn-primary w-full">Sign Up with Email</button>
    |
    |       <!-- Google Sign Up -->
    |       <div class="divider">OR</div>
    |       <button onclick="signupWithGoogle()" class="btn btn-outline w-full">
    |           <img src="https://www.svgrepo.com/show/475656/google-color.svg" class="w-5 h-5 mr-2" />
    |           Sign Up with Google
    |       </button>
    |   </div>
    |
    |   <script type="module" src="/static/js/signup.js"></script>
    </body>
</html>

signup.html
"signup.html" 28L, 1261B
[0] 0:nvim* 1:tailwindcss-
```

We've made sure to correctly set the ids for the relevant input fields so that the document queries from our javascript don't fail. This is a simple page with just inputs for credentials and a button paired with an optional sign in with google button. `DaisUI` handles the basic styling, so below is how the page looks as of now:

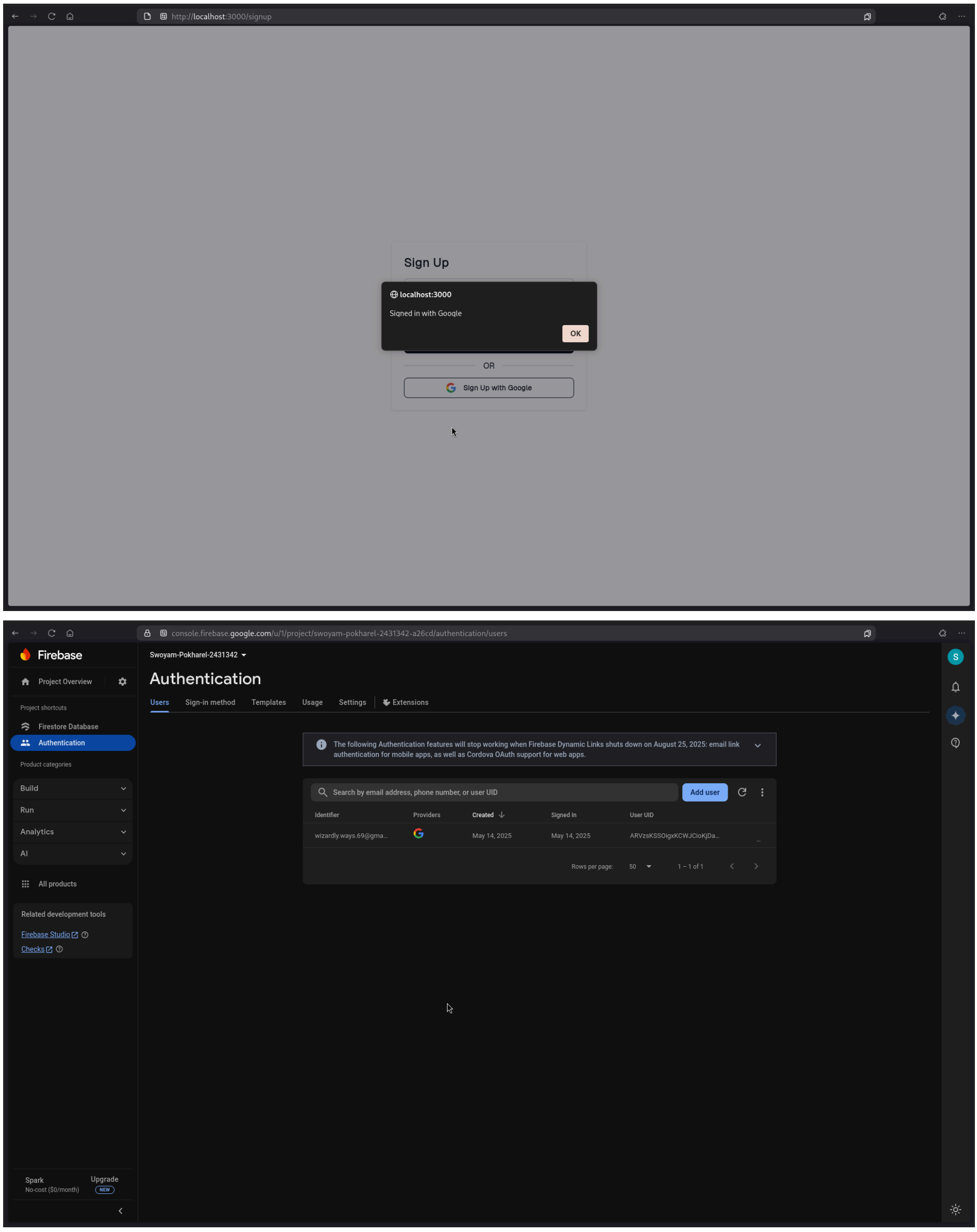


And more importantly, the page works:

Signing up with google:

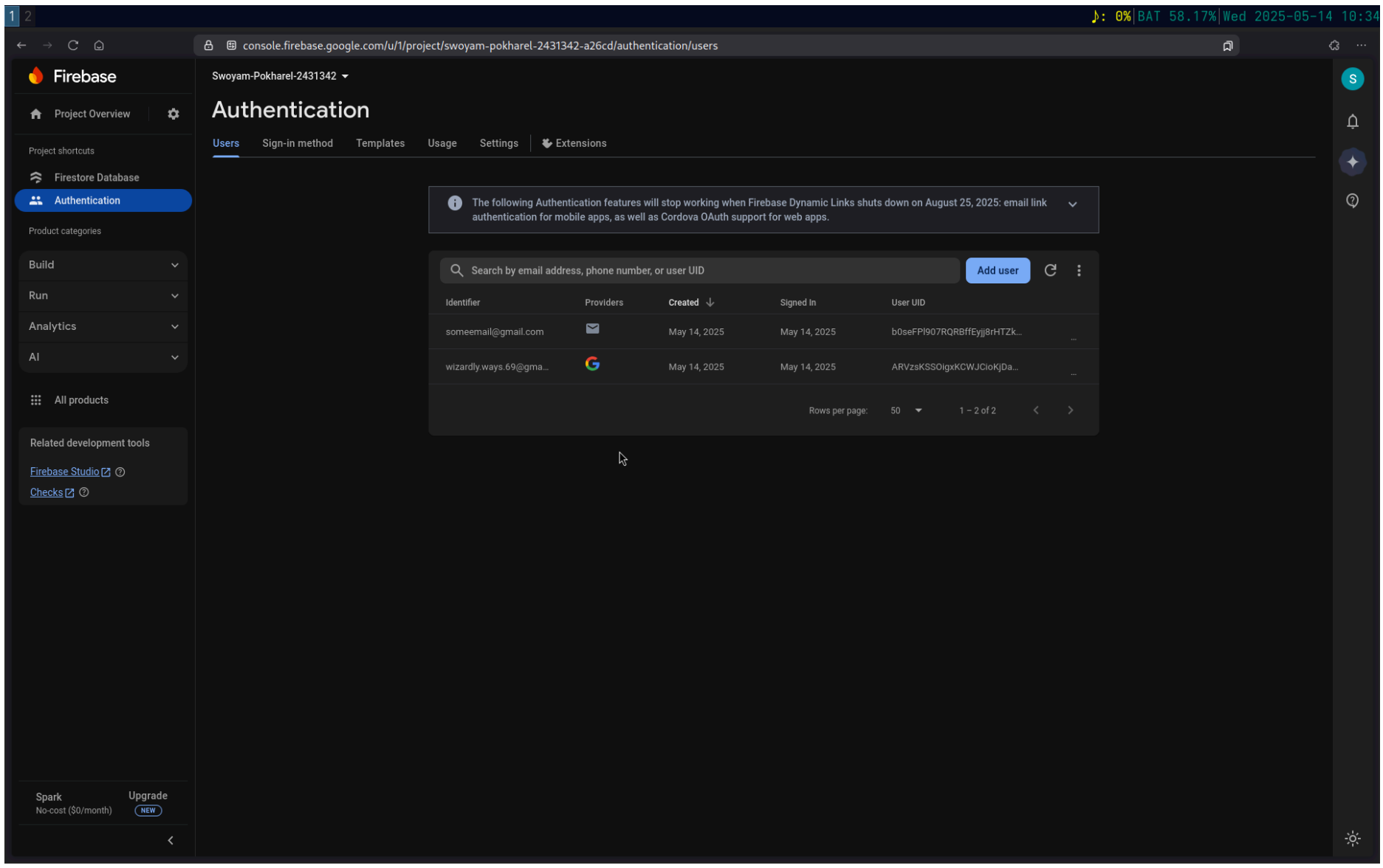


Note: cross the emails



So when I press sign up with google, a new window pops up because of `signInWithPopup` function that we called and it allows me to select a google account, after which a new account is created in firebase as evident by the above screenshot.

Signing up with email:



So signing up with email works too as evident by the above screenshot, although we are accepting any email in this case, there is no validation for emails so one may use something like temporary email, I didn't add validation as it goes beyond the scope of this task.

So, now that signing up with both `email-password` and google works, let's move on to tackling the login situation.

II. User Login:

Logging In is also super convenient because of the functions the SDK provides, all we need to do is:

```
16 import { initializeApp } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-app.js";
15 import {
14     getAuth,
13     signInWithEmailAndPassword,
12     GoogleAuthProvider,
11     signInWithPopup,
10 } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-auth.js";
 9
 8 const firebaseConfig = {
 7     apiKey: "AIzaSyCkpwa8hE-wMEK2ZYnTqcZOoDDU9saV0Sk",
 6     authDomain: "swoyam-pokharel-2431342-a26cd.firebaseapp.com",
 5     projectId: "swoyam-pokharel-2431342-a26cd",
 4     storageBucket: "swoyam-pokharel-2431342-a26cd.firebasestorage.app",
 3     messagingSenderId: "81012309295",
 2     appId: "1:81012309295:web:eef1c4daa68bb4749bf5da"
 1 };
17
static/js/login.js [+]
29 fewer lines
[0] 0:nvim* 1:tailwindcss-
```

Firstly, we need to important all relevant functions that we are going to use,

Setting Up Logging In With Email-Password:

Then logging in with Email-Password is as simple as calling the function `signInWithEmailAndPassword`:

```
// Email/Password Login
window.loginWithEmail = async function () {
    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value;

    try {
        const userCred = await signInWithEmailAndPassword(auth, email, password);
        alert("Login successful");
        console.log("User:", userCred.user);
    } catch (err) {
        alert("Login failed: " + err.message);
    }
};
```

Below is the same code, presented with carbon for readability:

This is a very simple function, all we do here is query the DOM to get the email and password and call the function with it. The rest is automatically handled by the function itself.

Setting Up Logging In With Google:

Logging in with google is also very easy, it's just:

```
// Google Login
window.loginWithGoogle = async function () {
    try {
        const result = await signInWithPopup(auth, googleProvider);
        const user = result.user;
        alert("Signed in with Google");
        console.log(user);
    } catch (err) {
        alert("Google login failed: " + err.message);
    }
};
static/js/login.js
"static/js/login.js" 46L, 1501B                                                    11
[0] 0:nvim* 1:tailwindcss-
```

Below is the same code, presented with carbon for readability:

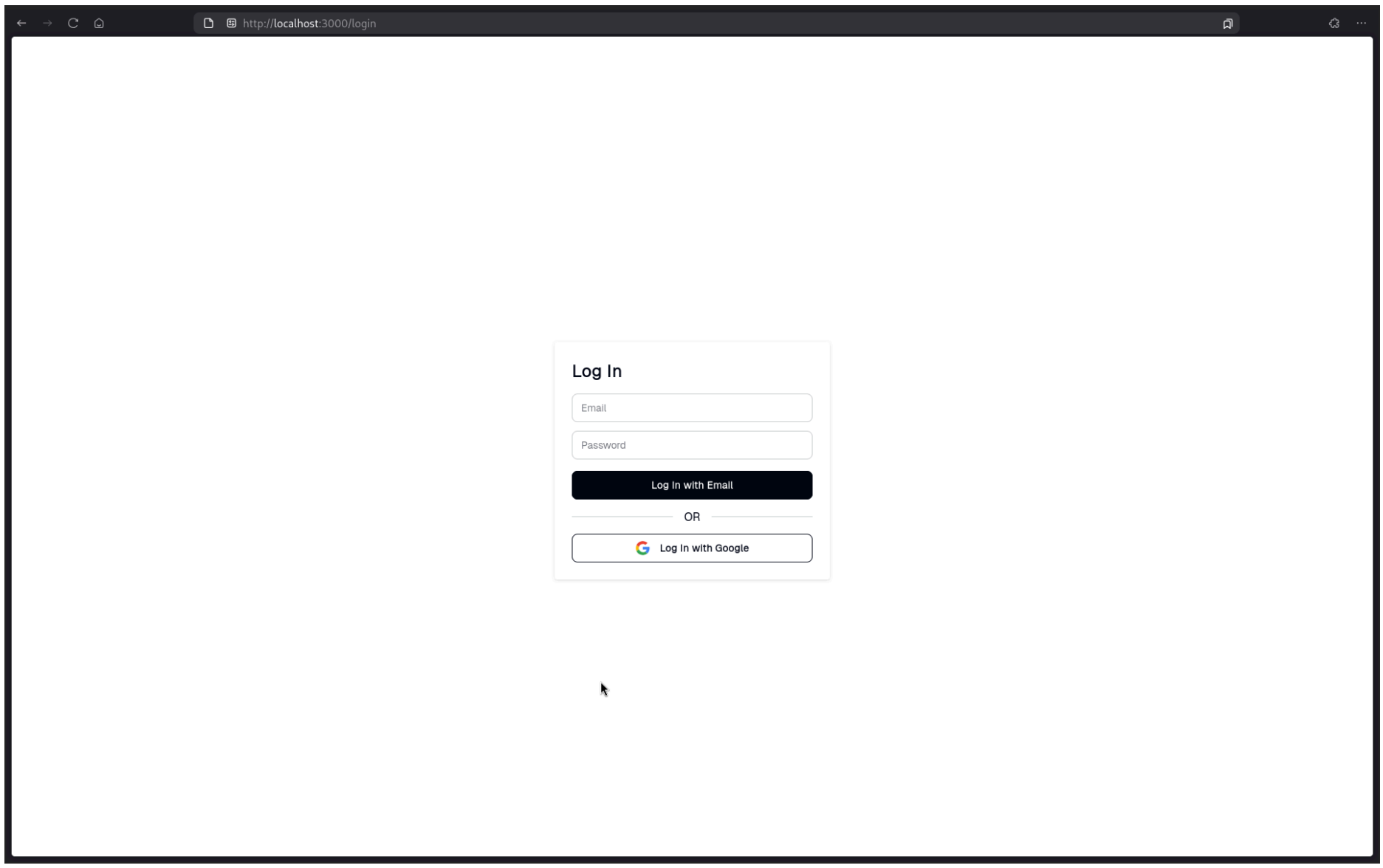
```
// Google Login
window.loginWithGoogle = async function () {
    try {
        const result = await signInWithPopup(auth, googleProvider);
        const user = result.user;
        alert("Signed in with Google");
        console.log(user);
    } catch (err) {
        alert("Google login failed: " + err.message);
    }
};
```

This too is a super simple function, and almost identical to it's signup equivalent. We simply call the `signInWithPopup` function and the rest is handled by firebase itself.

That's all that we need in terms of javascript, now let's setup the HTML and tie it with the javascript:

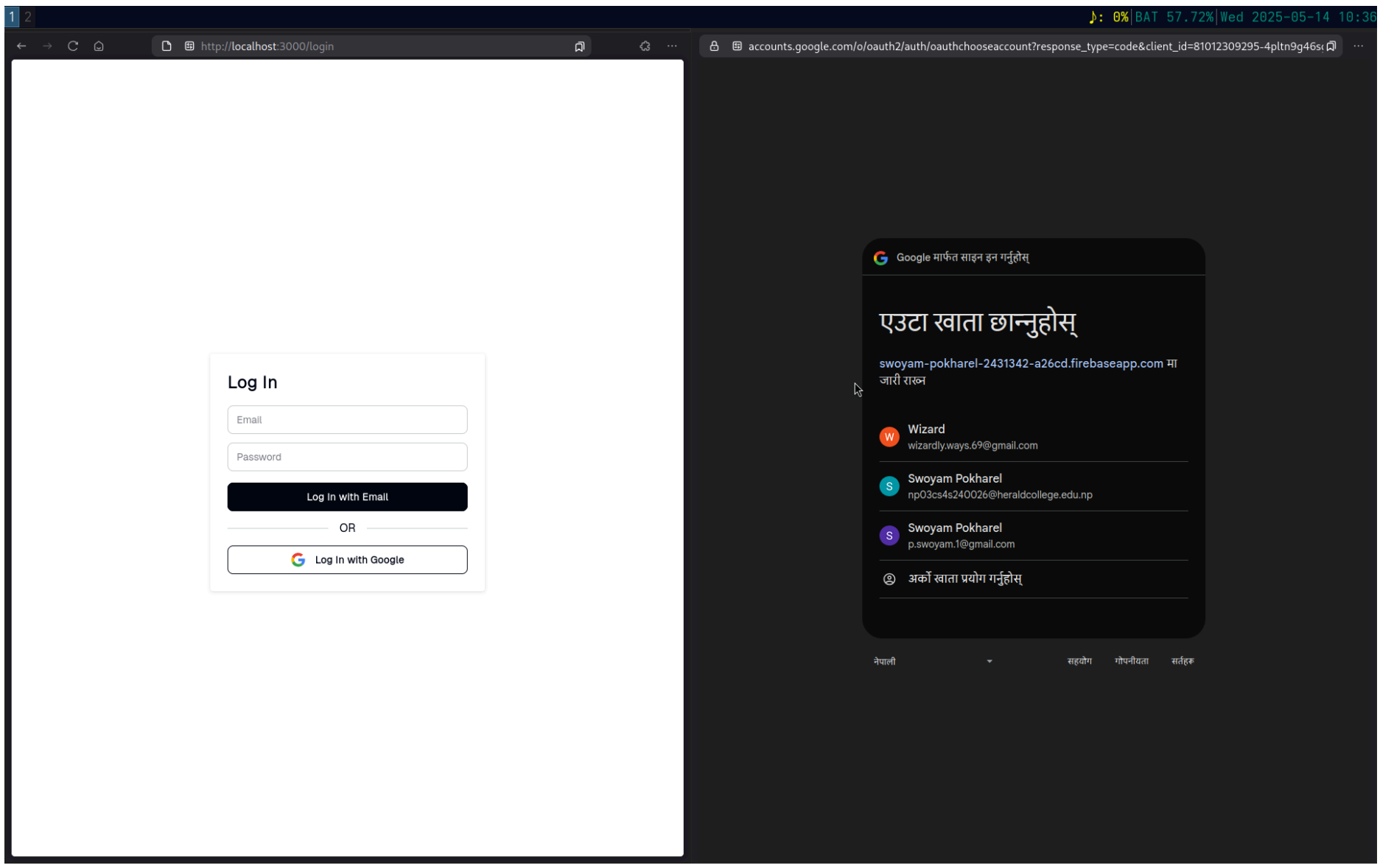
```
<!DOCTYPE html>
<html lang="en">
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <link rel="stylesheet" href="/static/output.css" />
      <title>2431342 Login</title>
  </head>
  <body class="h-screen flex justify-center items-center">
      <div class="max-w-sm mt-10 p-6 bg-white shadow rounded">
          <h2 class="text-2xl font-bold mb-4">Log In</h2>

          <!-- Email and Password Form -->
          <input id="email" type="email" placeholder="Email" class="input input-bordered w-full mb-3" />
          <input id="password" type="password" placeholder="Password" class="input input-bordered w-full mb-4" />
          <button onclick="loginWithEmail()" class="btn btn-primary w-full">Log In with Email</button>

          <!-- Google Login -->
          <div class="divider">OR</div>
          <button onclick="loginWithGoogle()" class="btn btn-outline w-full">
              <img src="https://www.svgrepo.com/show/475656/google-color.svg" class="w-5 h-5 mr-2" />
              Log In with Google
          </button>
      </div>
      <script type="module" src="/static/js/login.js"></script>
  </body>
</html>

login.html
"login.html" 27L, 1242B
[0] 0:nvim* 1:tailwindcss-
```

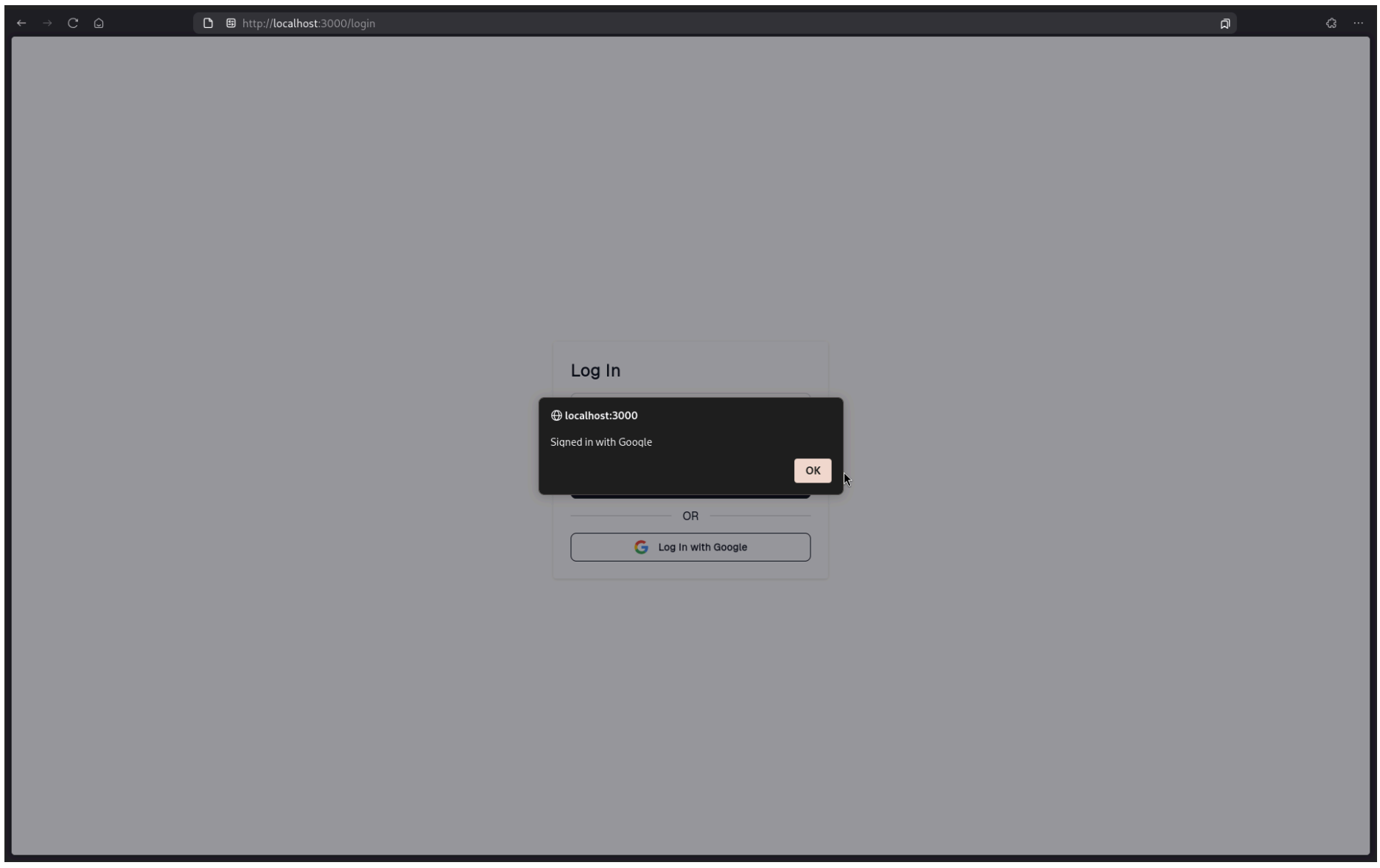
This page is identical to the signup page as well, and we have the same id's so that the DOM query from javascript doesn't fail. We simply call the functions with an inline handler. This is how the page looks:



Logging In WIth Google:

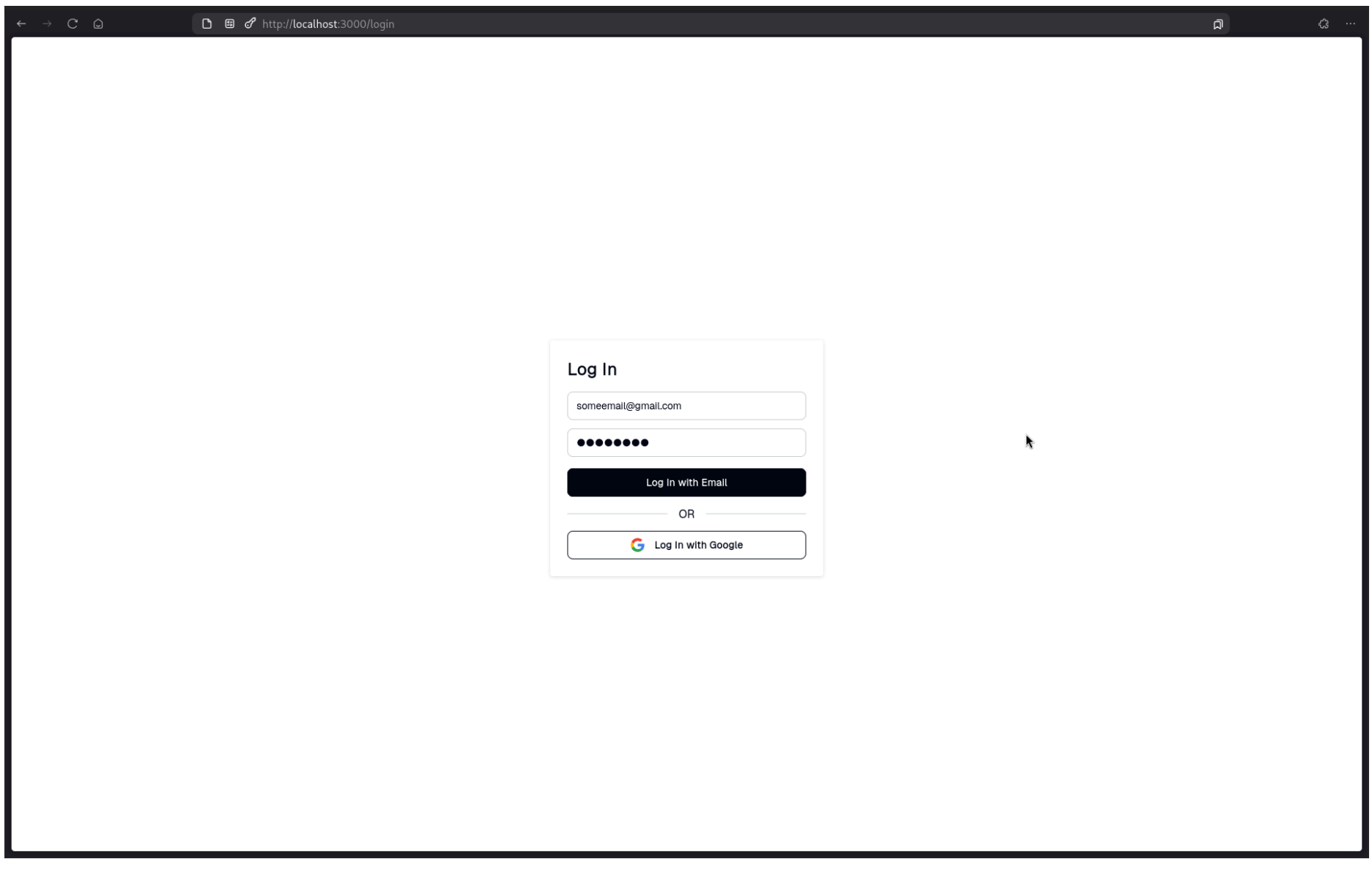
And more importantly, logging in with google works:

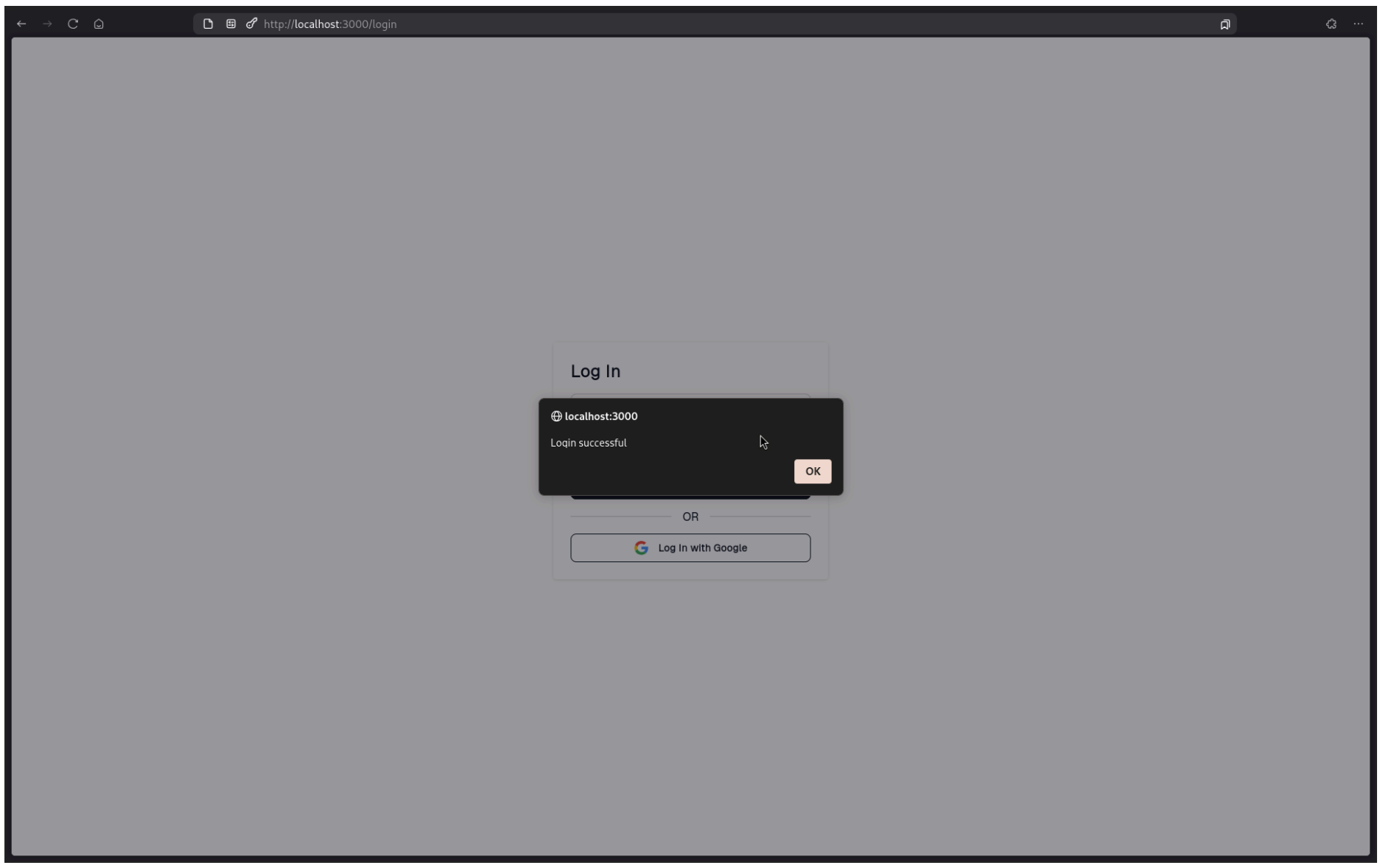




Logging In With Email-Password:

And so does logging in with email and password:





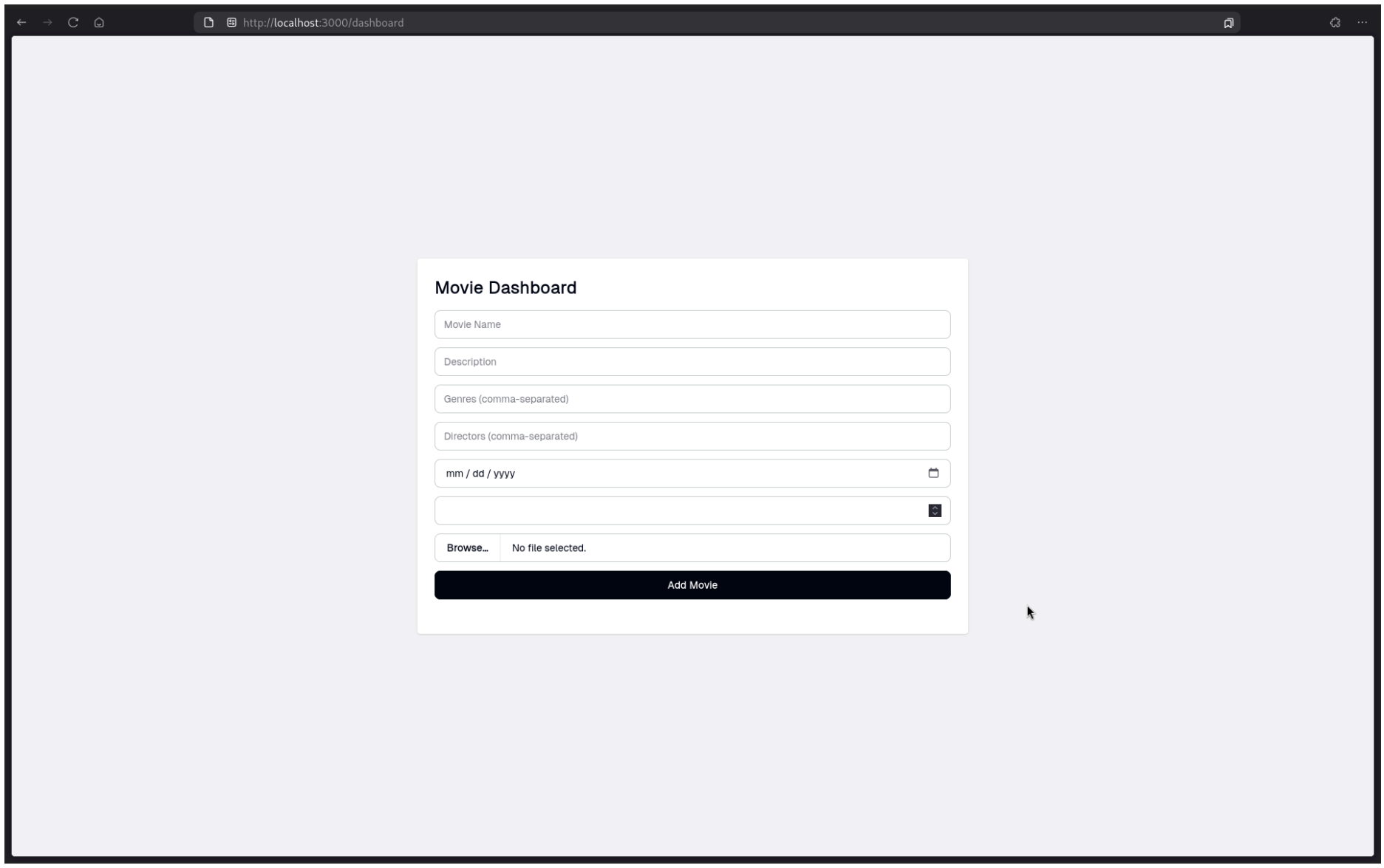
Now that we've covered both log in and sign up let's move to the user dashboard.

III. User Dashboard

User Dashboard is also pretty straight forward, the only major challenge is hooking up image uploads to some sort of a storage. For this, I'll first start with the frontend side of things first and then connect the backend:

```
1  <!DOCTYPE html>
1  <html lang="en">
2      <head>
3          <meta charset="UTF-8">
4          <meta name="viewport" content="width=device-width, initial-scale=1.0">
5          <link rel="stylesheet" href="/static/output.css" />
6          <title>2431342 Dashboard</title>
7      </head>
8      <body class="p-6 bg-gray-100 flex justify-center items-center min-h-screen">
9
10         <div class="max-w-3xl mx-auto bg-white p-6 shadow rounded">
11             <h1 class="text-2xl font-bold mb-4">Movie Dashboard</h1>
12
13             <!-- Movie Form -->
14             <div id="movie-form" class="space-y-3 mb-6">
15                 <input id="movieName" placeholder="Movie Name" class="input input-bordered w-full"/>
16                 <input id="description" placeholder="Description" class="input input-bordered w-full"/>
17                 <input id="genre" placeholder="Genres (comma-separated)" class="input input-bordered w-full"/>
18                 <input id="movieDirectors" placeholder="Directors (comma-separated)" class="input input-bordered w-full"/>
19                 <input id="releaseDate" type="date" class="input input-bordered w-full"/>
20                 <input id="rating" type="number" min="0" max="5" class="input input-bordered w-full"/>
21                 <input id="poster" type="file" accept="image/*" class="file-input file-input-bordered w-full"/>
22                 <button onclick="createMovie()" class="btn btn-primary w-full">Add Movie</button>
23             </div>
24
25             <!-- Movies List -->
26             <div id="movies" class="space-y-4"></div>
27         </div>
28
29         <script type="module" src="/static/js/dashboard.js"></script>
30     </body>
31 </html>


dashboard.html
"dashboard.html" 32L, 1631B
[0] 0:nvim* 1:tailwindcss-
```

This is the page for now, it's just for proof of concept, all we did here is have a bunch of inputs that correspond to the structure here. And because I'm using `DaisyUI` these come pre-styled and look like:



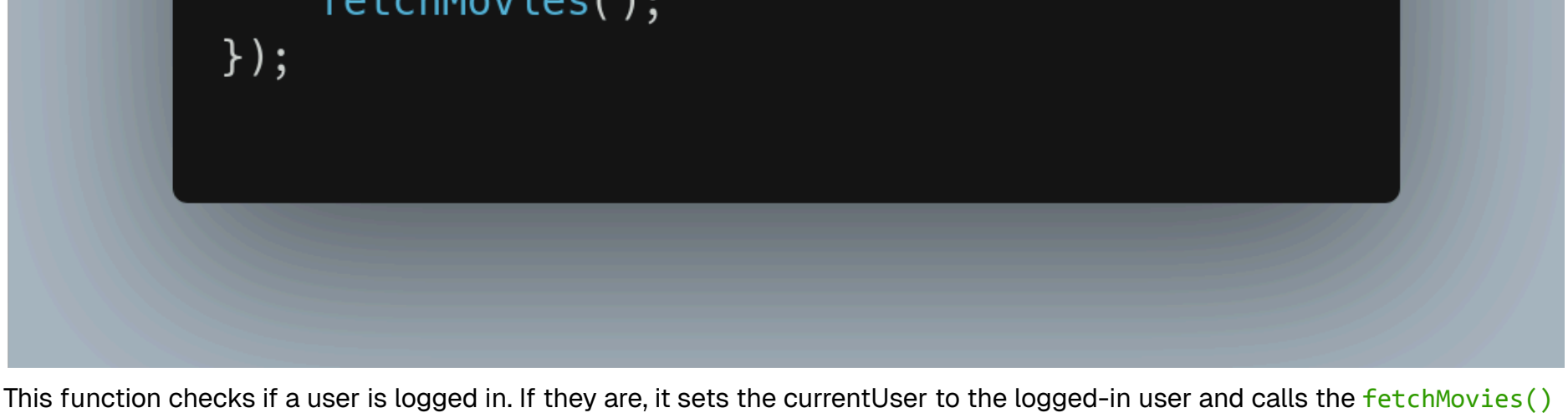
Now that I have a way to upload images from the frontend, let's move on to the javascript:
First things first, we import the necessary functions:

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.7.1/firebase-app.js";

import {
    getAuth,
    onAuthStateChanged
} from "https://www.gstatic.com/firebasejs/11.7.1/firebase-auth.js";

import {
    getFirestore,
    collection,
    addDoc,
    query,
    where,
    getDocs,
    serverTimestamp
} from "https://www.gstatic.com/firebasejs/11.7.1/firebase-firestore.js";

const firebaseConfig = {
    apiKey: "AIzaSyCkpwa8hE-wMEK2ZYnTqcZOoDDU9saV0Sk",
    authDomain: "swoyam-pokharel-2431342-a26cd.firebaseapp.com",
    projectId: "swoyam-pokharel-2431342-a26cd",
    storageBucket: "swoyam-pokharel-2431342-a26cd.firebasestorage.app",
    messagingSenderId: "81012309295",
    appId: "1:81012309295:web:eef1c4daa68bb4749bf5da"
};

const app = initializeApp(firebaseConfig);
const auth = getAuth();
const db = getFirestore();
```

Handling Unauthorized User

Now, the first order of business is not allowing access to users who aren't logged in. To do this, all we have to do is:

```
let currentUser = null;

// Redirect if not logged in
onAuthStateChanged(auth, async (user) => {
    if (!user) {
        window.location.href = "/login.html";
        return;
    }
    currentUser = user;
    fetchMovies();
});
```


let currentUser = null;

// Redirect if not logged in
onAuthStateChanged(auth, async (user) => {
    if (!user) {
        window.location.href = "/login.html";
        return;
    }
    currentUser = user;
    fetchMovies();
});

This function checks if a user is logged in. If they are, it sets the currentUser to the logged-in user and calls the `fetchMovies()` function (which we'll write later). If the user isn't logged in, the function redirects them to `/login.html`. Since this is tied to `onAuthStateChanged`, the function is reactive and updates accordingly.

READ: Fetching User's Movies:

Now that we are sure that only logged in users can view the dashboard, we need to now populate the dashboard with all the documents that the logged in user has created. To do this, we will be querying all the documents where the `ownerID` is equal to the `currentUser.ID` this way, we can be sure that the code will fetch only the documents created by that particular user. Hence, this implies that when creating a new movie document, we also now need to add on to the structure here and add a ownerID field too.

```
1 2                                                                    ♪: 0% BAT 54.71% Wed 2025-05-14 10:45

10 // Redirect if not logged in
 9 onAuthStateChanged(auth, async (user) => {
 8     if (!user) {
 7         window.location.href = "/login.html";
 6         return;
 5     }
 4     currentUser = user;
 3     fetchMovies();
 2 });
 1
43 async function fetchMovies() {
 1     const q = query(collection(db, "Movie_DB"), where("ownerId", "==", currentUser.uid));
 2     const snapshot = await getDocs(q);
 3
 4     const container = document.getElementById("movies");
 5     container.innerHTML = "";
 6
 7     snapshot.forEach(doc => {
 8         const data = doc.data();
 9         container.innerHTML += `
10       <div class="border p-4 rounded bg-gray-50">
11         <h2 class="text-xl font-bold">${data.movieName}</h2>
12         <p>${data.description}</p>
13         <p>Genres: ${data.genre.join(", ")}</p>
14         <p>Directors: ${data.movieDirectors.join(", ")}</p>
15         <p>Release: ${data.releaseDate}</p>
16         <p>Rating: ${data.rating}</p>
17         <img src="${data.posterURL}" alt="poster" class="w-32 mt-2"/>
18       </div>
19     `;
20     });
21 }
22
static/js/dashboard.js
22 lines yanked into "+                                                                              22
[0] 0:nvim* 1:tailwindcss-
```

Here we simply query the collection `Movie_DB` using the query function provided by the SDK and we select those records where the `ownerId` is equal to the `currentUser.ID`. After we fetch the records, we loop over each of those records and manually append the elements in the DOM by adding onto the `innerHTML` of the `movies` div that we defined above.

CREATE: Creating A New Movie:

Moving on to creating a new movie, for this the function will be:

```
24 }
23
22 window.createMovie = async function () {
21 |   const name = document.getElementById("newMovieName").value;
20 |   const desc = document.getElementById("newDescription").value;
19 |   const genre = document.getElementById("newGenre").value.split(",").map(s => s.trim());
18 |   const directors = document.getElementById("newDirectors").value.split(",").map(s => s.trim());
17 |   const release = document.getElementById("newReleaseDate").value;
16 |   const rating = parseFloat(document.getElementById("newRating").value);
15 |   const posterFile = document.getElementById("newPoster").files[0];
14 |
13 |   if (!posterFile) return alert("Please upload a poster!");
12 |   if (rating < 1 || rating > 5) return alert("Please Enter Rating Between 1 and 5");
11 |
10 |   try {
 9 |       const posterURL = await uploadToS3(posterFile);
 8 |       await addDoc(collection(db, "Movie_DB"), {
 7 |           movieName: name,
 6 |           movieDescription: desc,
 5 |           movieGenre: genre,
 4 |           movieDirectors: directors,
 3 |           movieRelease: release,
 2 |           movieRating: rating,
 1 |           moviePoster: posterURL,
108 |           ownerId: currentUser.uid,
  1 |           createdAt: serverTimestamp()
  2 |       });
  3 |
  4 |       document.getElementById("new_movie_modal").close();
  5 |       fetchMovies();
  6 |   } catch (err) {
  7 |       alert("Error creating movie: " + err.message);
  8 |   }
  9 };
 10
static/js/dashboard.js
/create                                                    [1/2]
[0] 0:bunx- 1:nvim*
```

Here all we are doing is querying the dom to extract all the fields from our HTML and we simply call the `addDoc` function that will handle the creation of the document. THe function here `uploadToS3` is not yet defined; the plan is to create a s3 bucket with public reads and writes to store images.

Handling Image Uploads:

Now, let's create a new S3 Bucket to handle our image uploads,

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type Info

- (●) **General purpose**
  Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.
- ( ) **Directory**
  Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

swoyam-pokharel-2431342-movie

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). Learn More

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

- ( ) **ACLs disabled (recommended)**
  All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
- (●) **ACLs enabled**
  Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

⚠ We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

Object Ownership

- (●) **Bucket owner preferred**
  If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.
- ( ) **Object writer**
  The object writer remains the object owner.

ⓘ If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. Learn more

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

- [ ] **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
  - [ ] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
  - [ ] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
    S3 will ignore all ACLs that grant public access to buckets and objects.
  - [ ] **Block public access to buckets and objects granted through *new* public bucket or access point policies**
    S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
  - [ ] **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
    S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

- [ ] I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning

- (●) Disable
- ( ) Enable

Tags - *optional* (0)

You can use bucket tags to track storage costs and organize buckets. Learn more

No tags associated with this bucket.

Add tag

Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type Info

- (●) Server-side encryption with Amazon S3 managed keys (SSE-S3)
- ( ) Server-side encryption with AWS Key Management Service keys (SSE-KMS)
- ( ) Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)
  Secure your objects with two separate layers of encryption. For details on pricing, see **DSSE-KMS pricing** on the **Storage** tab of the Amazon S3 pricing page.

Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. Learn more

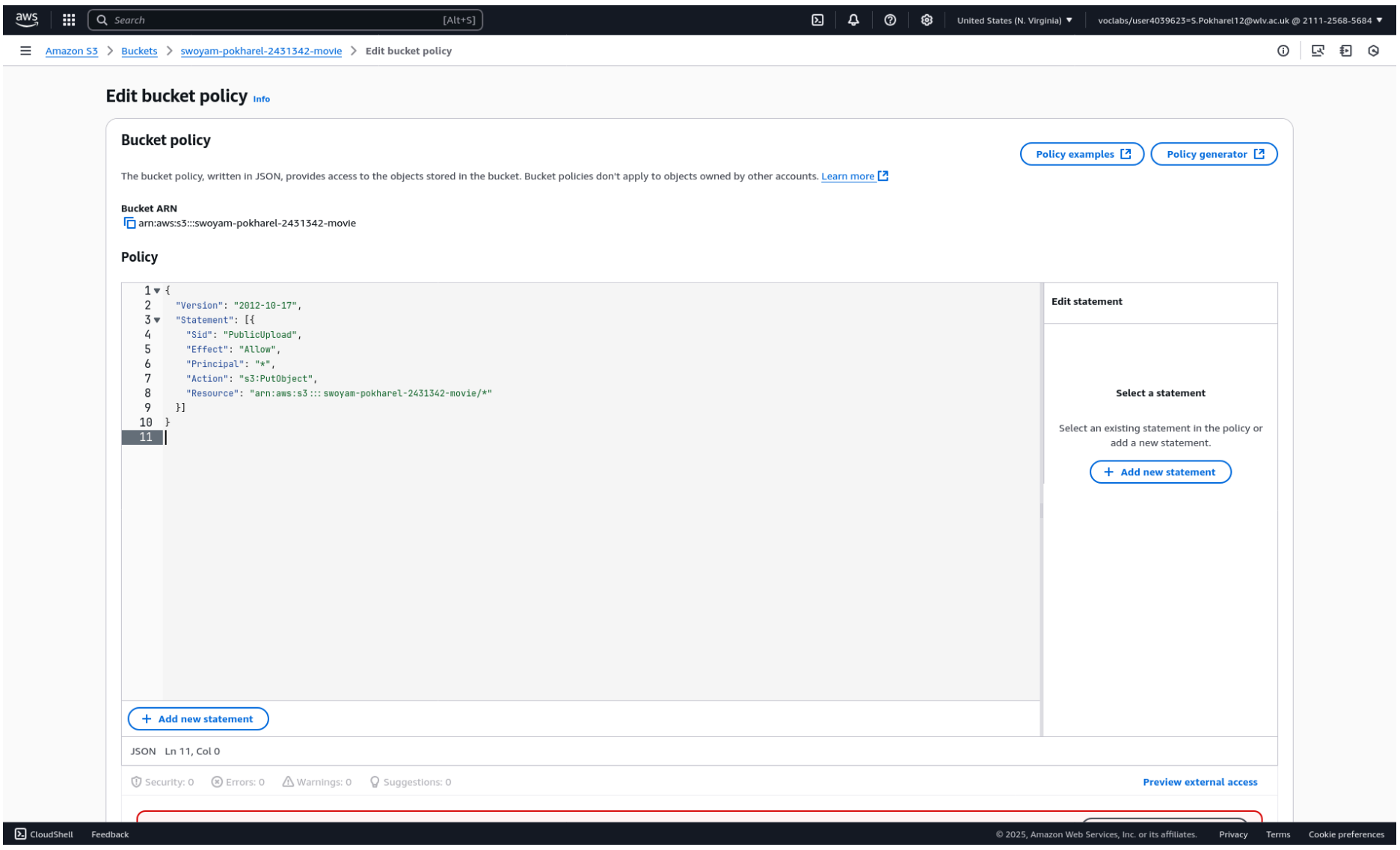
- ( ) Disable
- (●) Enable

▶ **Advanced settings**

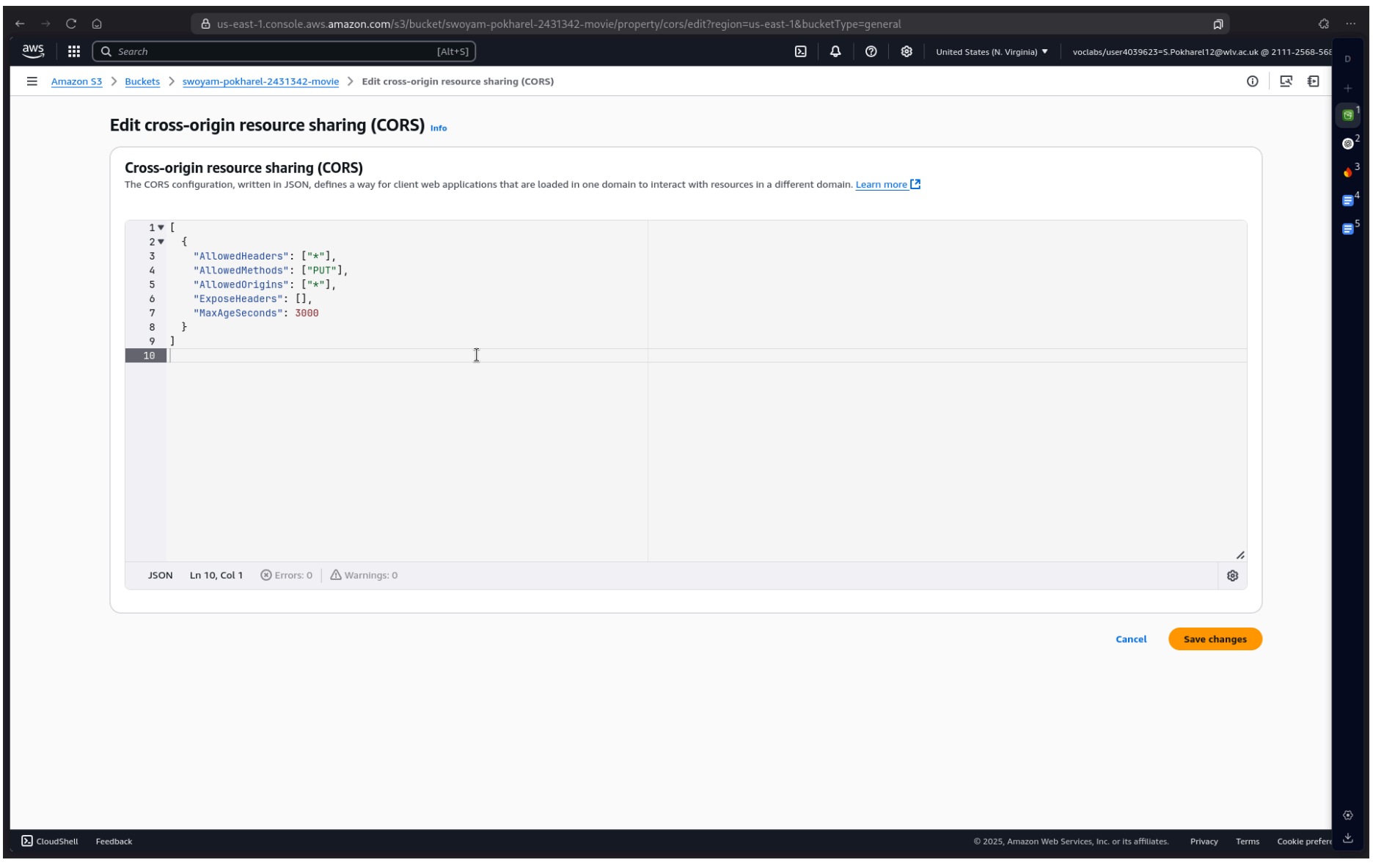
ⓘ After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

Cancel Create bucket

Here, I'm simply creating a new bucket under `swoyam-pokharel-2431342-movie` with public access. After that's done, we need to update the Bucket Policy To Allow Public Writes



I acknowledge that this is not safe as anyone will be able to upload in this bucket, but since this is just a demo I reckon that it won't matter much. Finally, we also need to update the CORS so that the browser doesn't block our request.



And that's all the setup we need in the s3 bucket, now let's move on to the `uploadToS3` function:

```
            return dir === 'asc'
                ? (a.movieRating || 0) - (b.movieRating || 0)
                : (b.movieRating || 0) - (a.movieRating || 0);
        }
        if (field === 'release') {
            return dir === 'asc'
                ? new Date(a.movieRelease) - new Date(b.movieRelease)
                : new Date(b.movieRelease) - new Date(a.movieRelease);
        }
        return 0;
    });
}

renderMovies(filtered);
};

async function uploadToS3(file) {
    const timestamp = Date.now();
    const fileName = encodeURIComponent(`${timestamp}_${file.name}`);
    const uploadUrl = `https://swoyam-pokharel-2431342-movie.s3.amazonaws.com/${fileName}`;

    const res = await fetch(uploadUrl, {
        method: "PUT",
        headers: {
            "Content-Type": file.type
        },
        body: file
    });

    if (!res.ok) throw new Error("Upload failed");

    return uploadUrl;
}

static/js/dashboard.js
"static/js/dashboard.js" 239L, 8140B                                    17
[0] 0:bunx- 1:nvim*
```

Below is the same code

```
async function uploadToS3(file) {
    const timestamp = Date.now();
    const fileName = encodeURIComponent(`${timestamp}_${file.name}`);
    const uploadUrl = `https://swoyam-pokharel-2431342-movie.s3.amazonaws.com/${fileName}`;

    const res = await fetch(uploadUrl, {
        method: "PUT",
        headers: {
            "Content-Type": file.type
        },
        body: file
    });

    if (!res.ok) throw new Error("Upload failed");

    return uploadUrl;
}
```

This is a simple function as well, now that we have public writes in our bucket, the actual uploading aspect is just a simple `fetch` call. I've opted for a timestamp based file upload, so that incase there happen to be 2 files of the same name, those 2 don't overwrite, instead the filenames are based on the time they are uploaded so that there is almost no chance of 2 files having the same name.

UPDATE: Updating an existing movie:

To update a movie, its quite simple too

```
window.updateMovie = async function (docId) {
    const name = document.getElementById("editMovieName").value;
    const desc = document.getElementById("editDescription").value;
    const genre = document.getElementById("editGenre").value.split(",").map(s => s.trim());
    const directors = document.getElementById("editDirectors").value.split(",").map(s => s.trim());
    const release = document.getElementById("editReleaseDate").value;
    const rating = parseFloat(document.getElementById("editRating").value);
    const posterFile = document.getElementById("editPoster").files[0];

    if (rating < 1 || rating > 5) return alert("Please Enter Rating Between 1 and 5");

    const updateData = {
        movieName: name,
        movieDescription: desc,
        movieGenre: genre,
        movieDirectors: directors,
        movieRelease: release,
        movieRating: rating
    };

    try {
        if (posterFile) {
            const posterURL = await uploadToS3(posterFile);
            updateData.moviePoster = posterURL;
        }

        await updateDoc(doc(db, "Movie_DB", docId), updateData);
        document.getElementById("edit_movie_modal").close();
        fetchMovies();
    } catch (err) {
        alert("Error updating movie: " + err.message);
        console.log(err);
    }
};
static/js/dashboard.js
/updating                                                                    [1/1]
[0] 0:bunx- 1:nvim*
```

Here we simply use the `updateDoc` provided by the SDK and that takes care of the updating. The only logic worth mentioning is that if the image is left null, so no image is included in the HTML it won't update. This makes it so that a user doesn't always have to keep re-uploading the image even if they don't want to. Now, when I design the update modal, I will make the fields pre-populate so that the user can only update what they want.

DELETE: Deleting a record:

To delete, it is also super simple, it's just:

```
async function deleteMovie(movieId) {
    const confirmed = confirm("Are you sure you want to delete this movie?");
    if (!confirmed) return;

    try {
        await deleteDoc(doc(db, "Movie_DB", movieId));
        alert("Movie deleted successfully!");
        fetchMovies();
    } catch (err) {
        alert("Error deleting movie: " + err.message);
    }
}
```

A simple call to the `deleteDoc` function handles deletion. Here we make sure to confirm the deletion of a record by using the `confirm` function so that a user doesn't accidentally delete anything.

Now that the fundamental CRUD operations are done, let's move on to updating the ui

Updated Code:

Index.html:

Now that we have our foundation in place we can work on the UI. Let's first start by adding search and filter controls:

```
<section class="flex w-full justify-between">
    <div class="form-control w-full max-w-lg flex">
        <input type="text" class="input border-black rounded-lg w-full nav-link" placeholder="Search Movies"/>
        <select id="sort-options" class="select select-bordered ml-4 bg-white rounded-lg nav-link">
            <option value="">No Filter</option>
            <option value="rating_asc">Rating Ascending</option>
            <option value="rating_desc">Rating Descending</option>
            <option value="release_asc">Release Ascending</option>
            <option value="release_desc">Release Descending</option>
        </select>
        <button class="btn btn-outline ml-4 bg-white rounded-lg filter-button nav-link">Filter</button>
    </div>
    <div class="flex">
        <button class="btn btn-primary ml-4 rounded-lg nav-link" onclick="new_movie_modal.showModal()"> + New Movie</button>
    </div>
</section>
```

Here we define 4 filters and a search bar along with a new movie button on the far right. Now, let's make it so that the movies we fetch are displayed in a tabular form:

```
<div class="overflow-x-auto">
    <table class="table w-full">
        <thead>
            <tr>
                <th>Movie ID</th>
                <th>Movie Name</th>
                <th>Rating</th>
                <th>Director/s</th>
                <th>Release</th>
                <th>Genre</th>
                <th>Description</th>
                <th>Poster</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody id="movies-body"></tbody>
    </table>
</div>
```

The final Table Heading for Actions will contain Edit and Delete Controls For That Particular Movie. I've structured it this way so that it is intuitive and it also makes pre-populating the update form easier as i can simply use the `closest` function and get the closest table row.

Now Let's Create a modal to create new movie record:

```
<!-- new movie dialog -->
<dialog id="new_movie_modal" class="modal">
    <div class="modal-box">
        <h3 class="font-bold text-lg">New Movie</h3>
        <div class="py-2 flex flex-col gap-2">
            <input type="text" id="newMovieName" class="input input-bordered w-full" placeholder="Movie Name" required />
            <input type="number" id="newRating" class="input input-bordered w-full" placeholder="Movie Rating" min="1" max="5" required />
            <input type="text" id="newDirectors" class="input input-bordered w-full" placeholder="Director/s" required />
            <input type="date" id="newReleaseDate" class="input input-bordered w-full" placeholder="Release Date" required />
            <input type="file" id="newPoster" accept="image/*" class="file-input file-input-bordered w-full" required />
            <input type="text" id="newGenre" class="input input-bordered w-full" placeholder="Genre" />
            <textarea id="newDescription" class="textarea textarea-bordered w-full" placeholder="Movie Description"></textarea>
        </div>
        <div class="modal-action">
            <form method="dialog" class="flex gap-2">
                <button type="button" class="btn btn-primary rounded-lg" onclick="createMovie()">Create</button>
                <button class="btn btn-outline rounded-lg">Cancel</button>
            </form>
        </div>
    </div>
</dialog>
```

This modal simply contains all the fields we need to create a new movie as defined in the structure here. The fields that were an array structure will be populated by splitting based on the commas.

Likewise, let's add in an edit modal:

```html
<!-- edit movie dialog -->
<dialog id="edit_movie_modal" class="modal">
    <div class="modal-box">
        <h3 class="font-bold text-lg">Edit Movie</h3>
        <div class="py-2 flex flex-col gap-2">
            <input type="text" id="editMovieName" class="input input-bordered w-full" placeholder="Movie Name" required />
            <input type="number" id="editRating" class="input input-bordered w-full" placeholder="Movie Rating" min="1" max="5" required />
            <input type="text" id="editDirectors" class="input input-bordered w-full" placeholder="Director/s" required />
            <input type="date" id="editReleaseDate" class="input input-bordered w-full" placeholder="Release Date" required />
            <input type="file" id="editPoster" accept="image/*" class="file-input file-input-bordered w-full" />
            <input type="text" id="editGenre" class="input input-bordered w-full" placeholder="Genre" />
            <textarea id="editDescription" class="textarea textarea-bordered w-full" placeholder="Movie Description"></textarea>
        </div>
        <div class="modal-action">
            <form method="dialog" class="flex gap-2">
                <button type="button" class="btn btn-primary rounded-lg" id="editConfirmBtn">Update</button>
                <button class="btn btn-outline rounded-lg">Cancel</button>
            </form>
        </div>
    </div>
</dialog>
```

This is pretty much identical to the above create new modal with the only difference being the IDs so that javascript can effectively query it.

Updated Fetch Function:

Now that we have new HTML structure, we need to update the fetch function too defined here.

```
♪: 0%|BAT 38.87%|Wed 2025-05-14 15:44
```

```javascript
async function fetchMovies() {
    const q = query(collection(db, "Movie_DB"), where("ownerId", "==", currentUser.uid));
    const snapshot = await getDocs(q);

    movies = [];
    snapshot.forEach(doc => {
        movies.push({ id: doc.id, ...doc.data() });
    });
    renderMovies(movies);
}

function renderMovies(list) {
    const tbody = document.getElementById("movies-body");
    if (!tbody) return;

    tbody.innerHTML = list.map(movie => `
        <tr data-id="${movie.id}" class="row-animation">
            <td id="table-row-animation">${movie.id}</td>
            <td id="table-row-animation">${movie.movieName}</td>
            <td id="table-row-animation">${movie.movieRating || ""}</td>
            <td id="table-row-animation">${movie.movieDirectors?.join(", ") || ""}</td>
            <td id="table-row-animation">${movie.movieRelease || ""}</td>
            <td id="table-row-animation">${movie.movieGenre?.join(", ") || ""}</td>
            <td id="table-row-animation" class="max-w-xs whitespace-normal">${movie.movieDescription || ""}</td>
            <td id="table-row-animation"><img src="${movie.moviePoster}" class="w-16" /></td>
            <td id="table-row-animation">
                <button class="btn btn-sm btn-primary edit-btn">Edit</button>
                <button class="btn btn-ghost delete-btn">Delete</button>
            </td>
        </tr>
    `).join("");
```

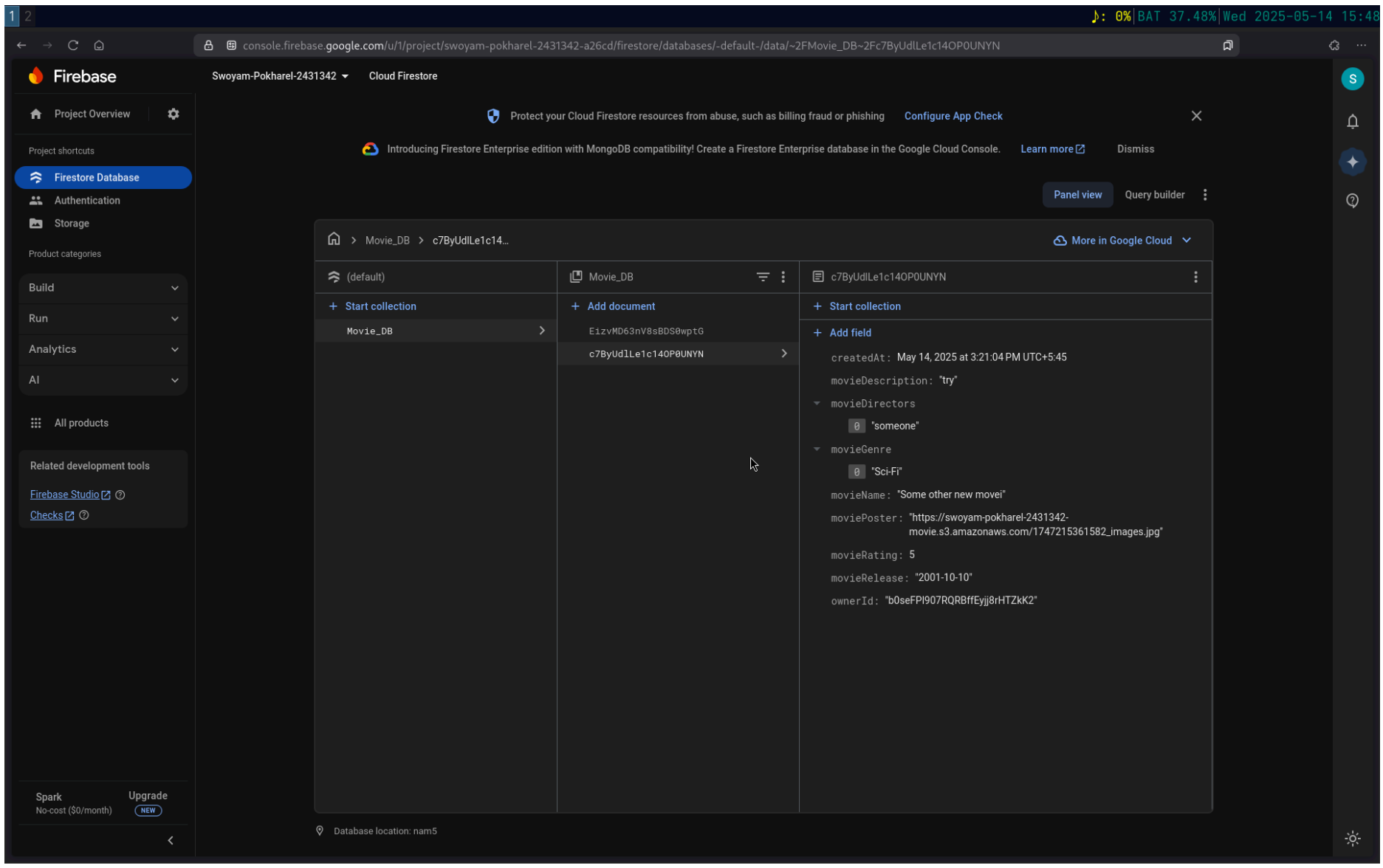
```
static/js/dashboard.js
Type  :qa  and press <Enter> to exit Nvim
[0] 0:bunx- 1:nvim*
```

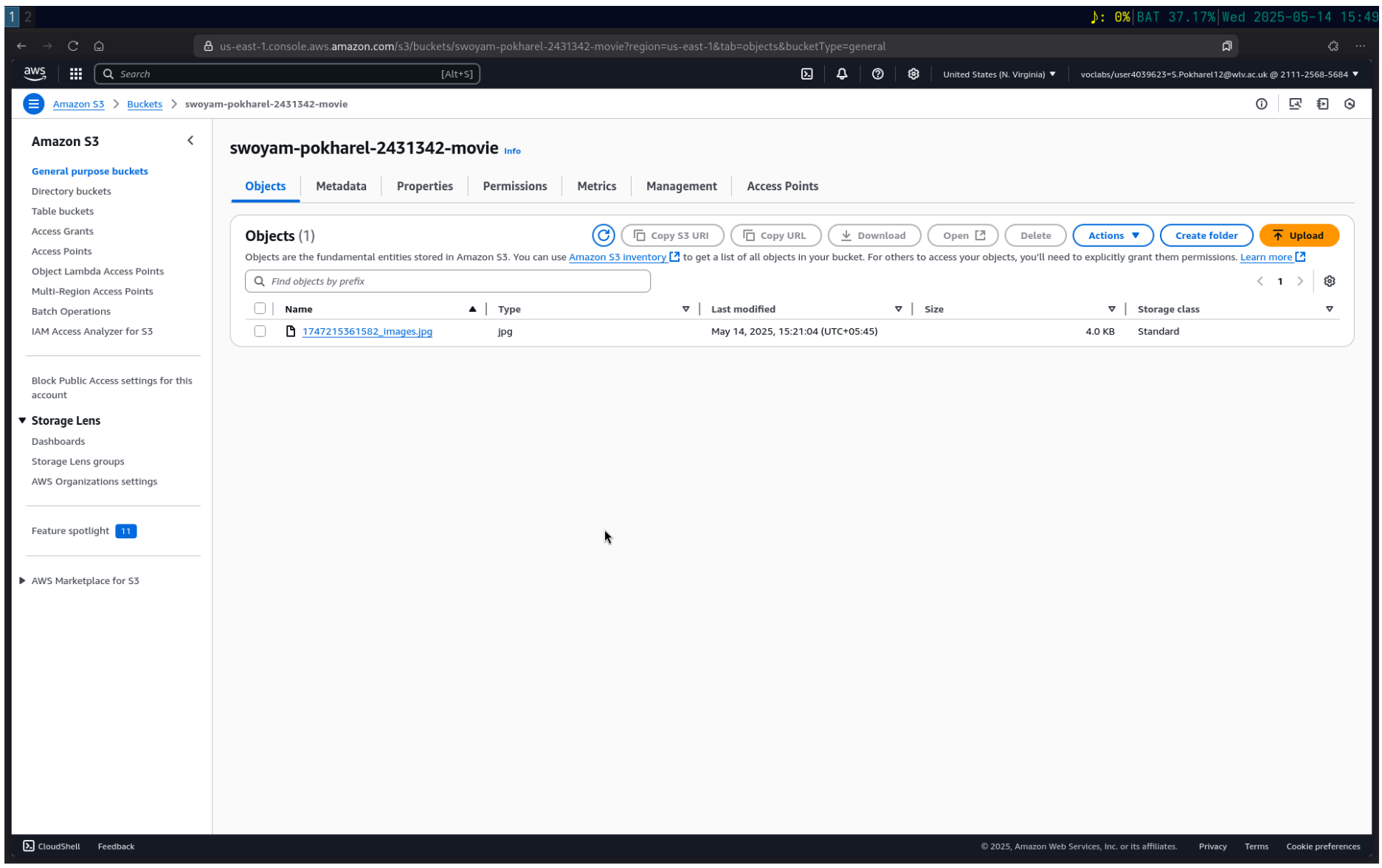
Now the fetch function only fetches and separates the creation of dom objects to the `renderMovies` function. It still follows the same logic as last time but I modularized it because we need to call the `renderMovies` function a lot when we have filters in place and it wouldn't make sense to always keep refetching the data from the firestore when we do so.

Now that these changes are made, lets try it out:

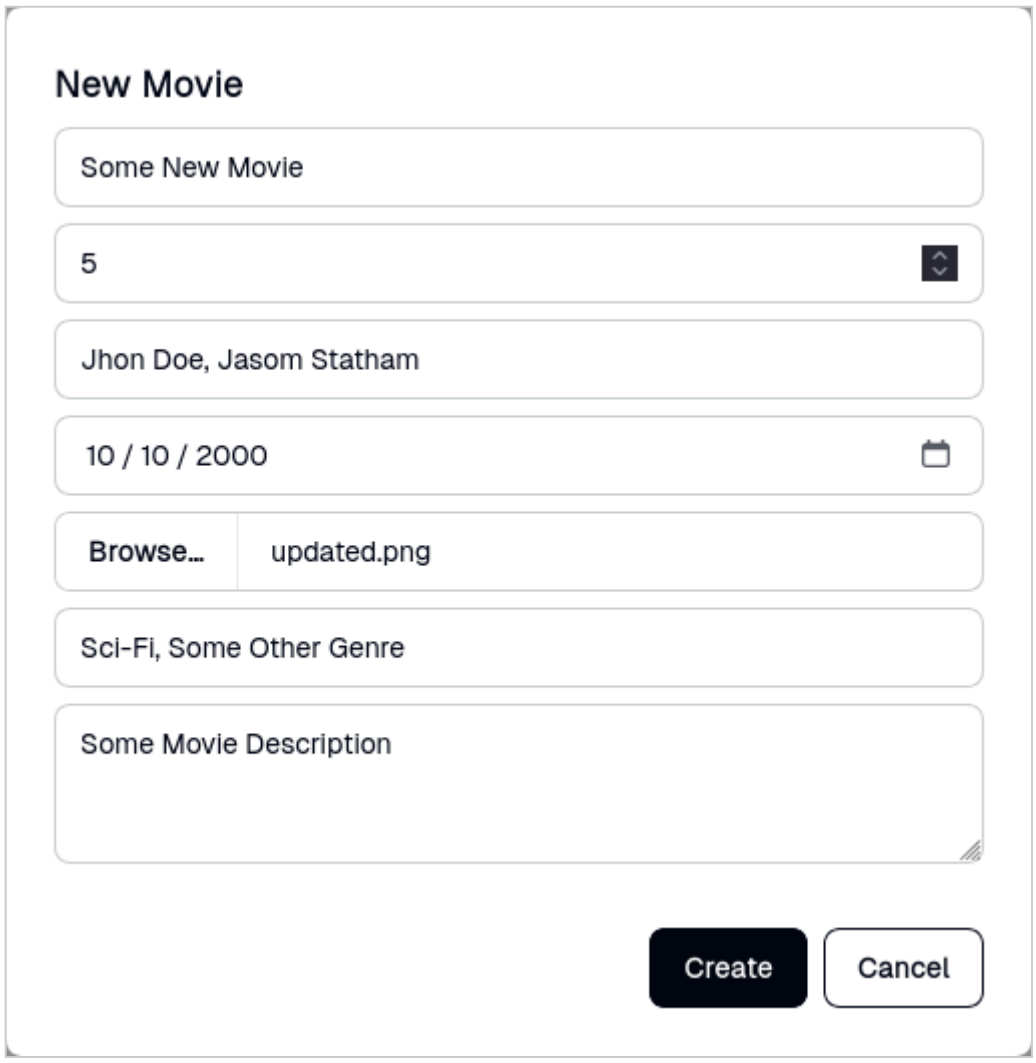
Testing Create:

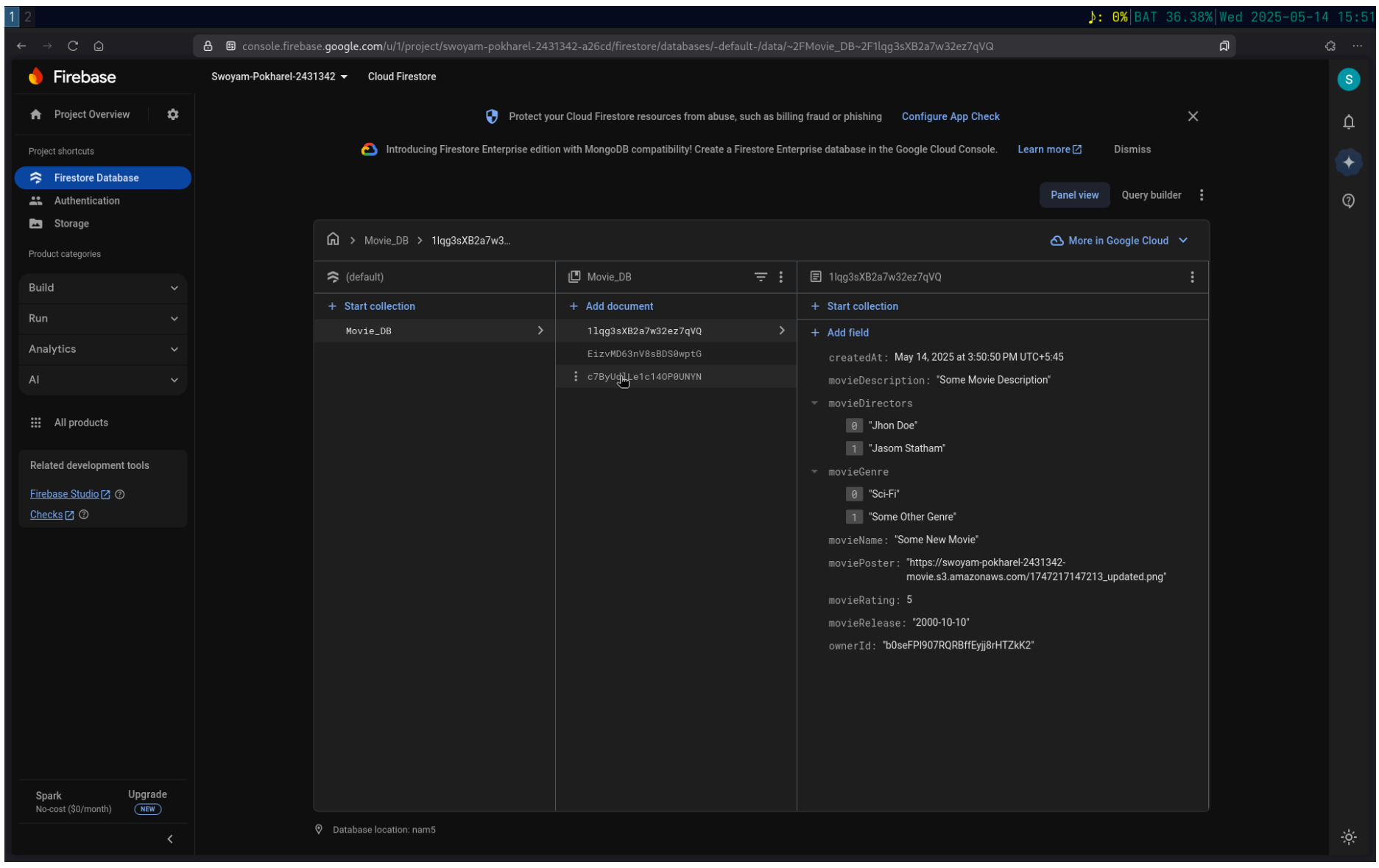
Firestore before creation:



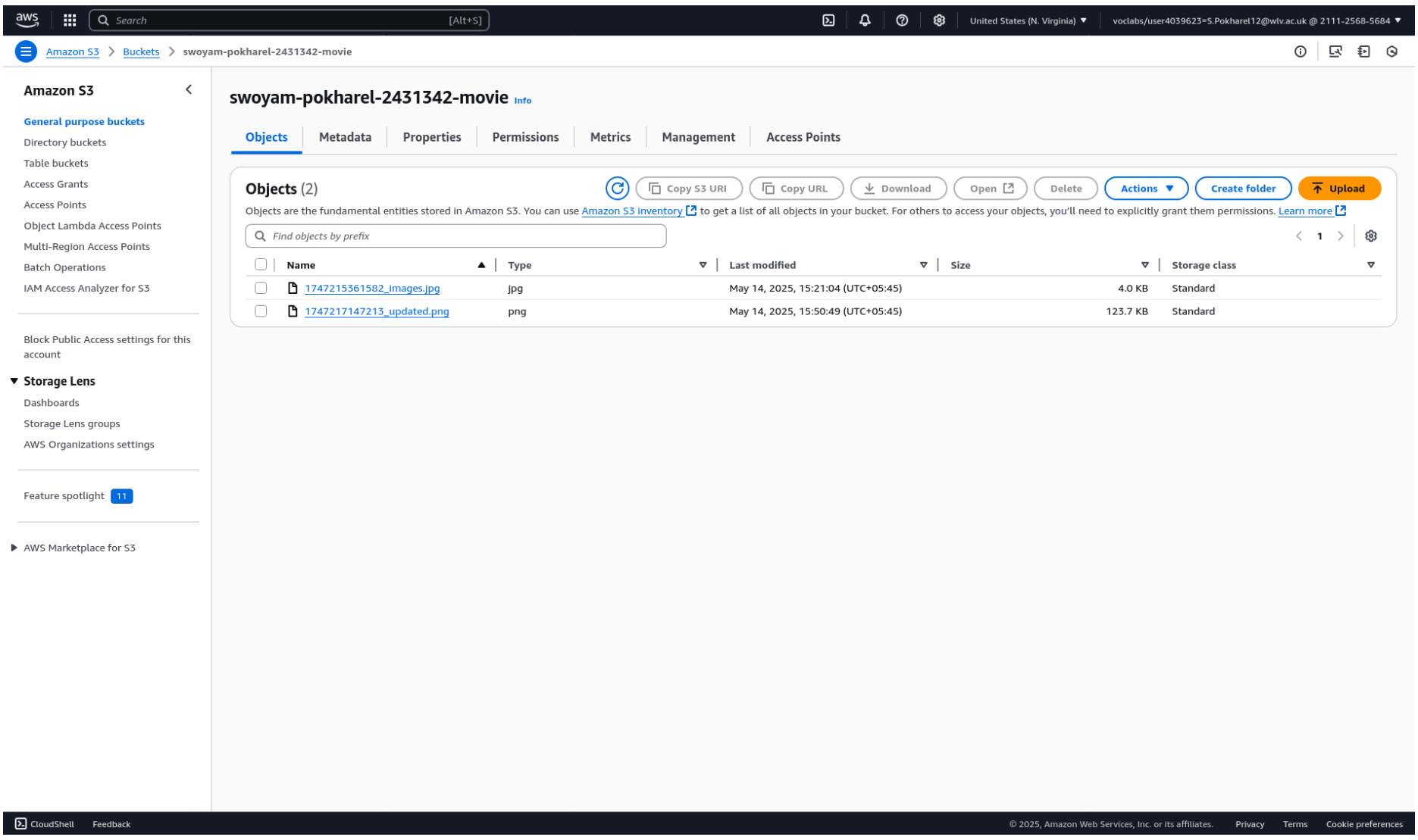


As we can see we have 1 image in the S3 bucket, the image I had previously used to test the upload, and we have 2 documents in firestore, after creating a new movie record,





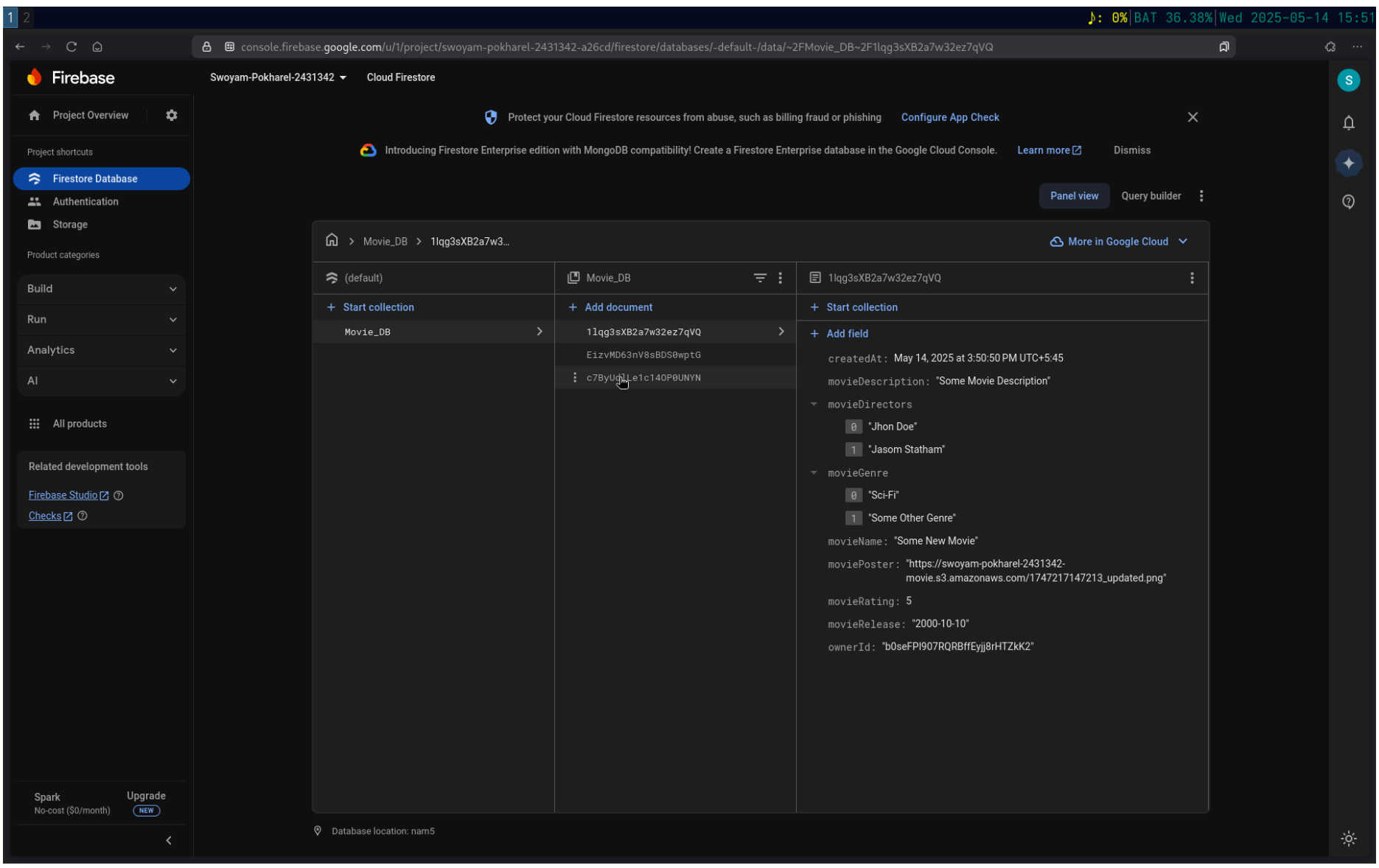
We can see that after we create, a new record is created in the firestore too; which is evidence that our create works correctly. Likewise, the s3 Bucket has a new image too



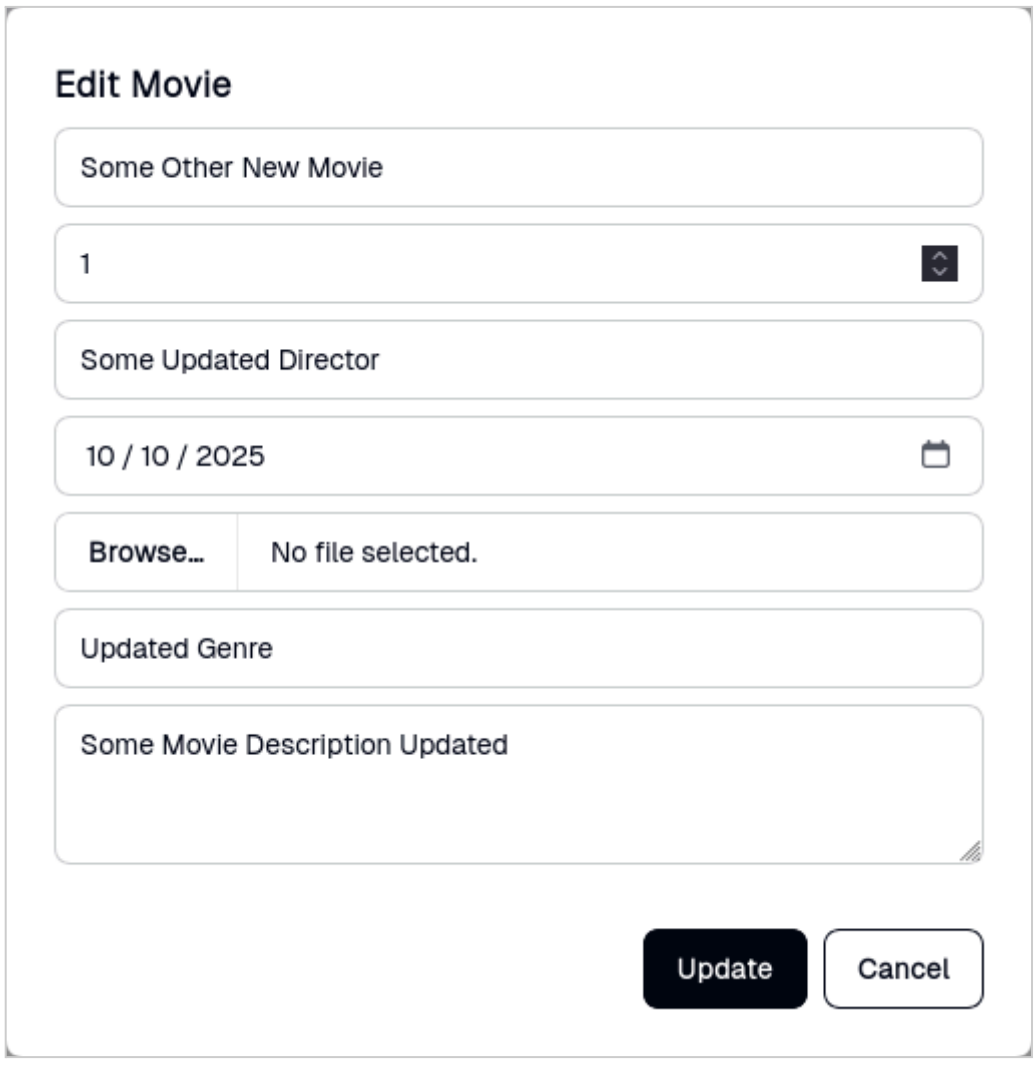
Now that we have established creation works, let's test out the update:

Testing Update:

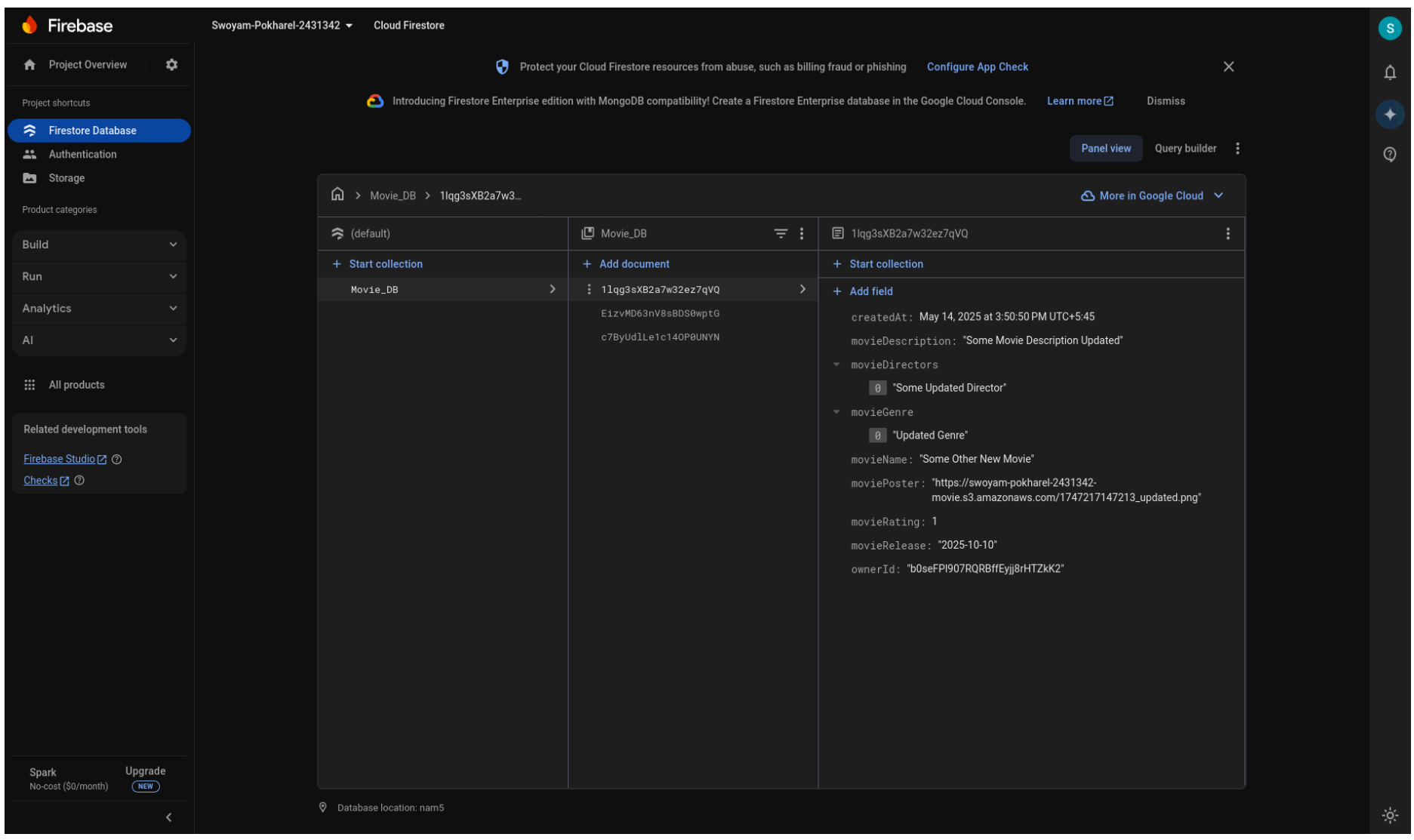
We will update the document we just created. Before updating we have,



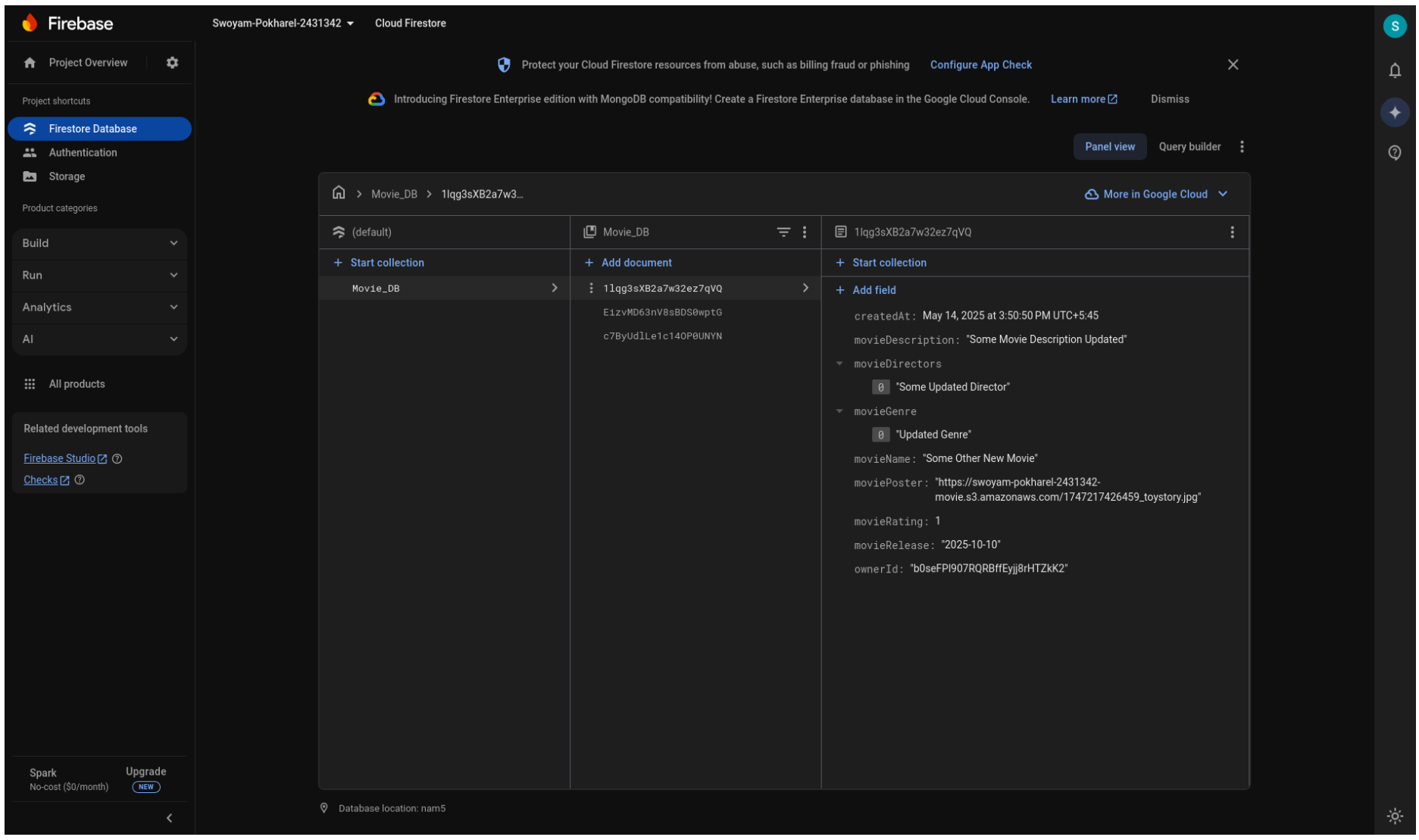
After updating,



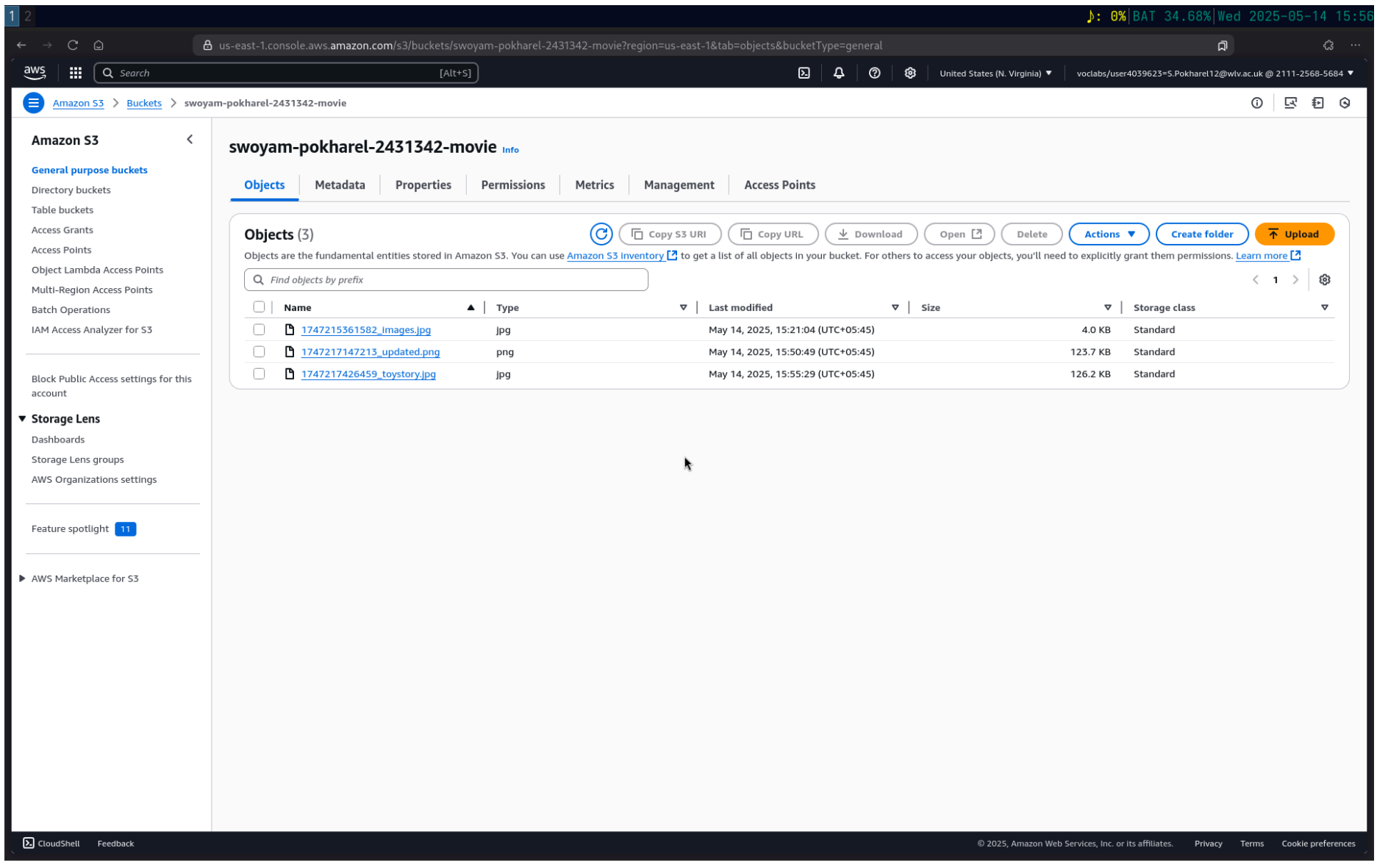
We have,



Note the difference in `movieDirectors`, `movieGenre`, `movieName`, `movieRating` etc. Since we didn't specify any file during the update, it kept the link to the previous file, if this time we change the file too we get:



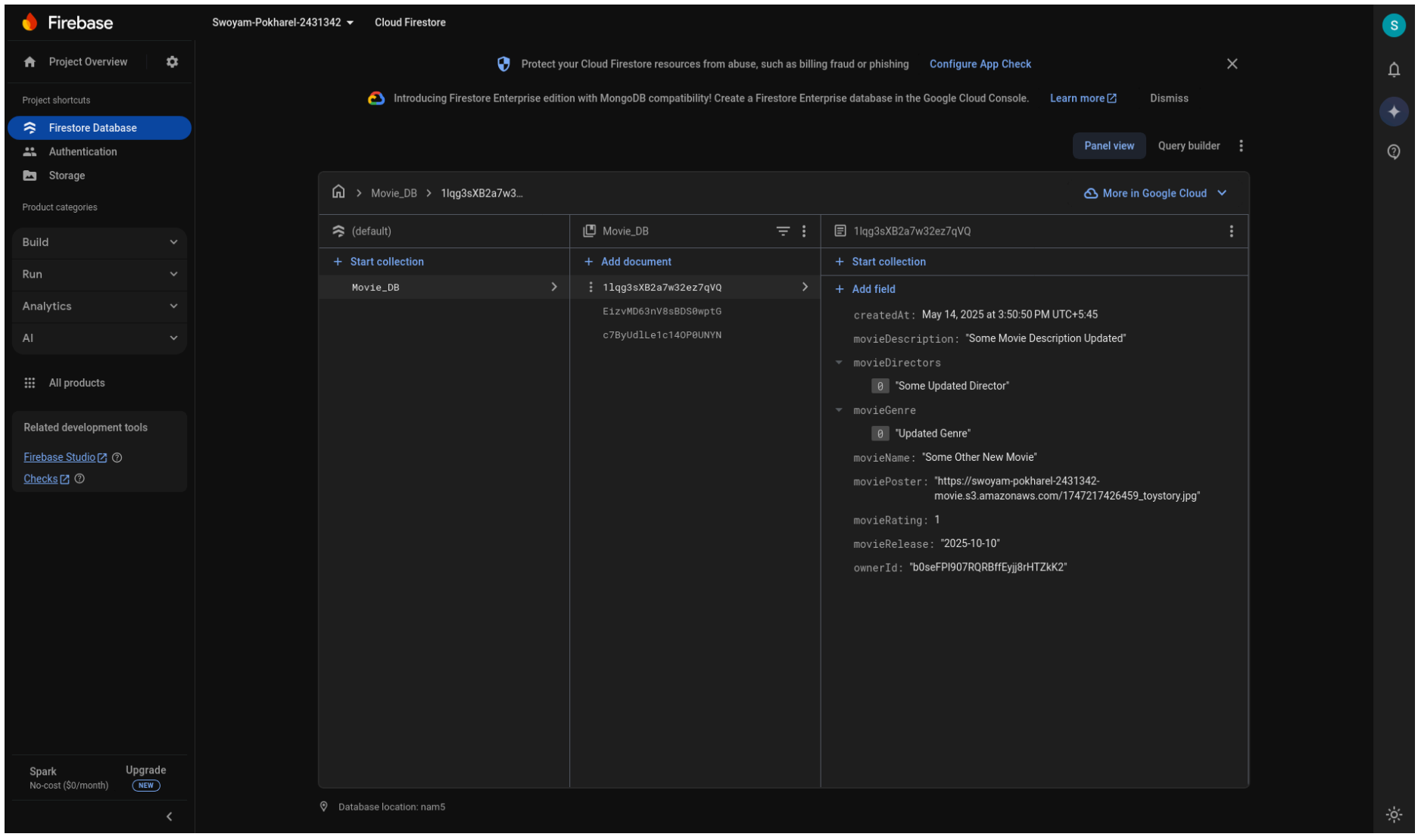
Note the change in the link to `moviePoster`. Furthermore, the updated image is also uploaded in our S3 bucket



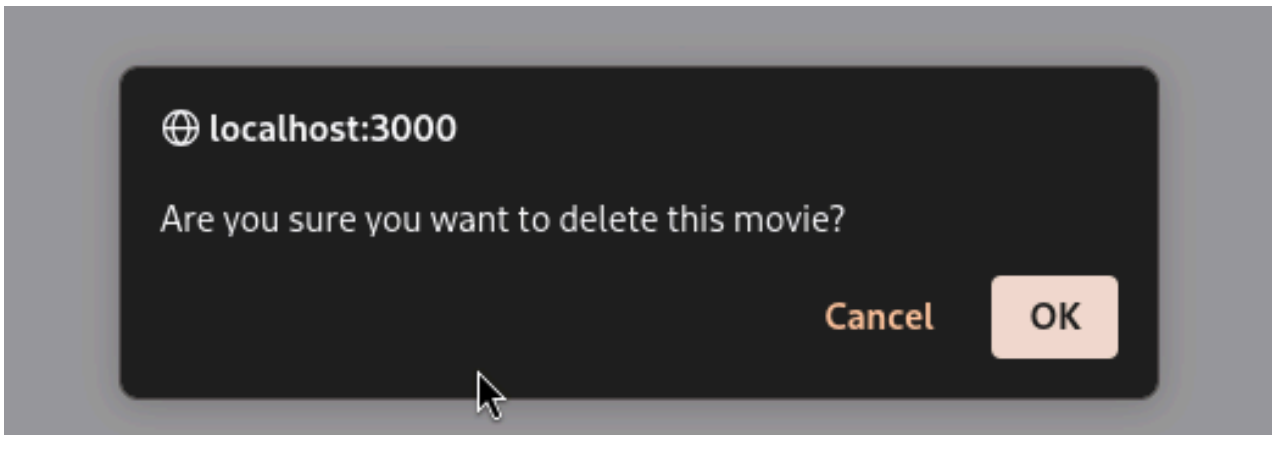
Now that Creates and Updates work, let’s test the Delete

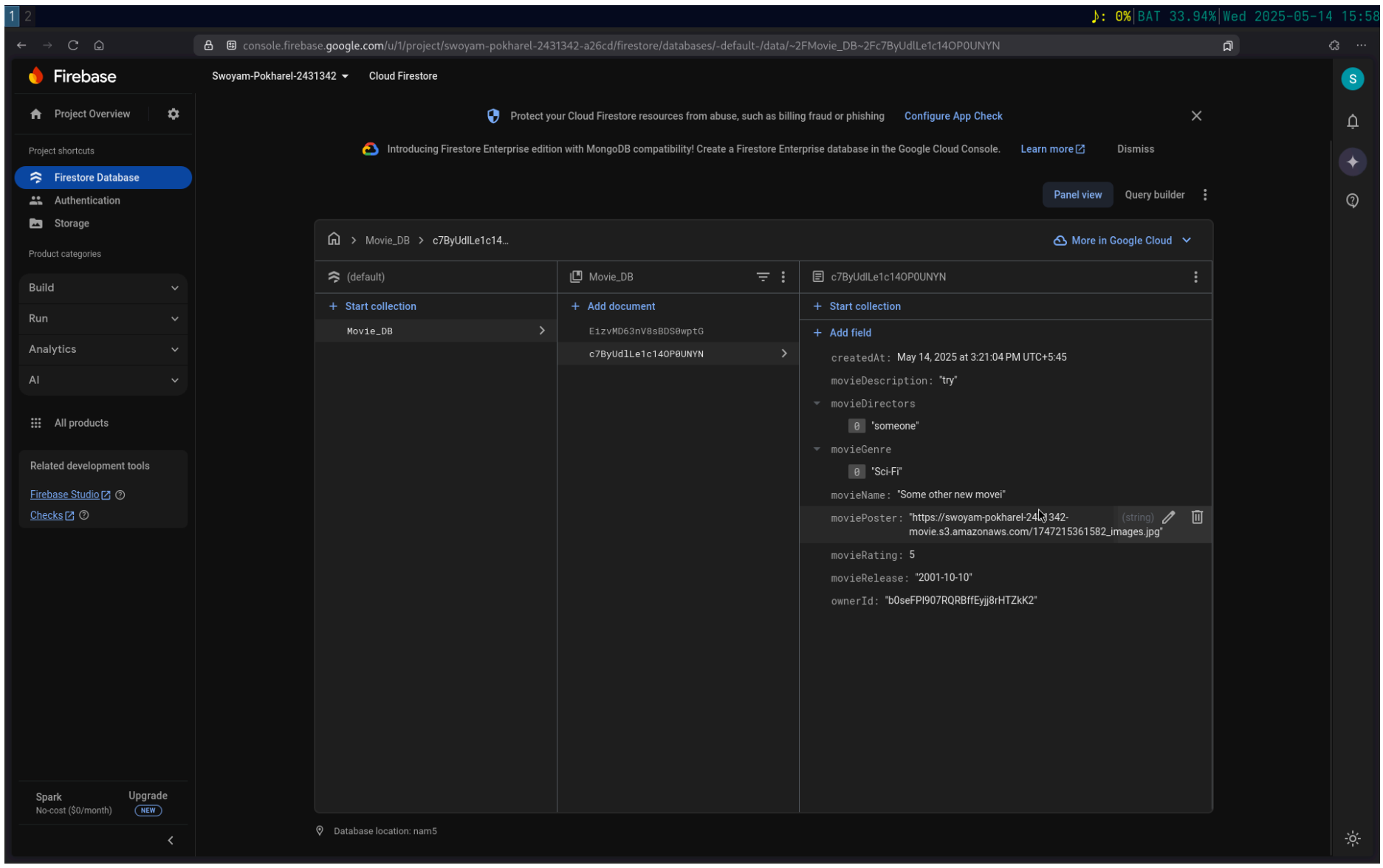
Testing Delete:

Before Deleting:



After Deleting:





After deleting, we can see that the record we had just updated is no longer there. This is evident that deletion works too.

Now that all Crud works, let's move on to making the search and filters work.

Implementing Search & FIlters:

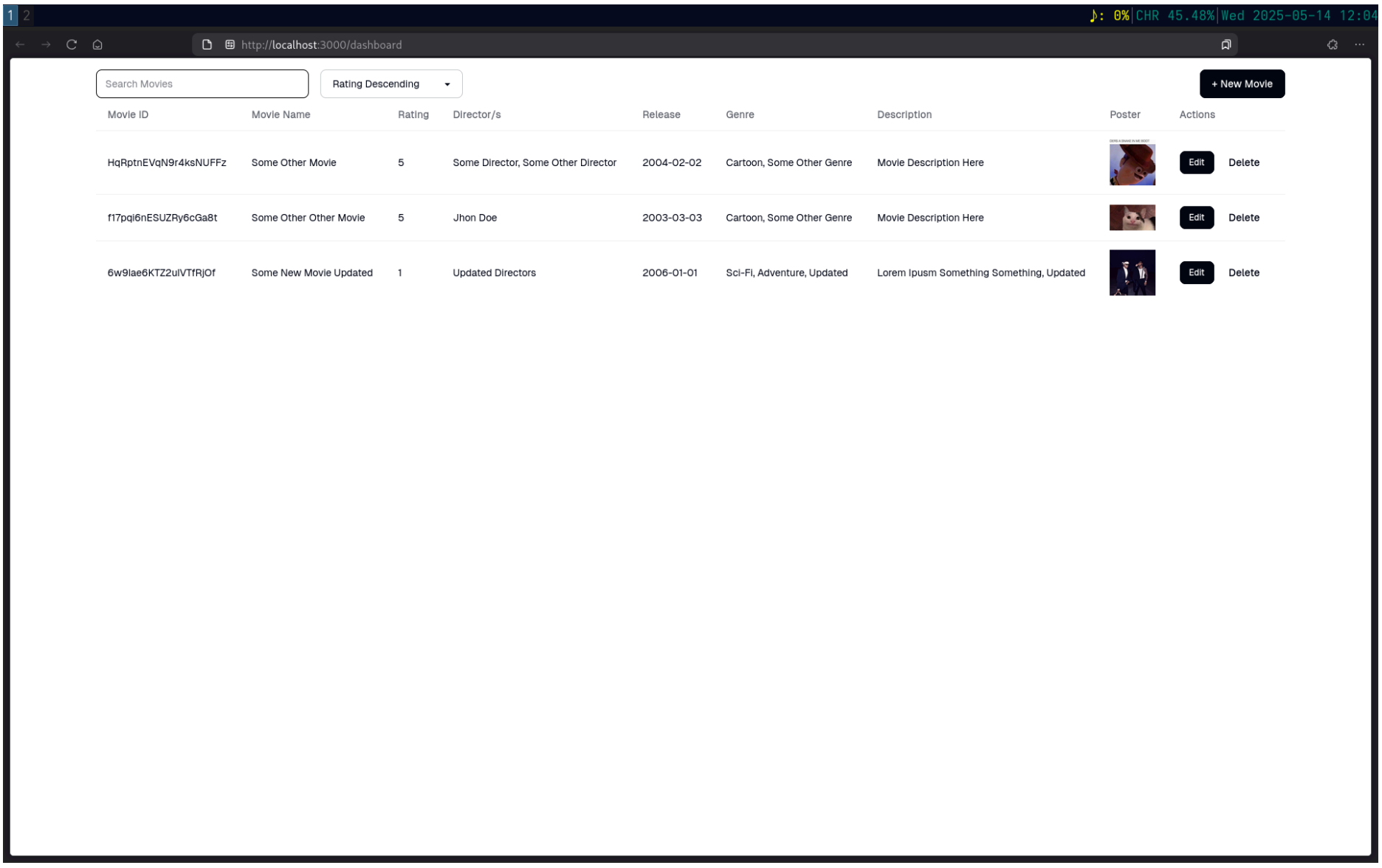
To implement Searching and Filters, we will be making use of the `renderMovies()` function that we modularized above.

```
});

window.filterMovies = () => {
    const term = document
        .querySelector('input[placeholder="Search Movies"]')
        .value
        .toLowerCase();
    const sort = document.getElementById('sort-options').value;

    let filtered = movies.filter(m =>
        m.movieName.toLowerCase().includes(term)
    );

    if (sort) {
        const [field, dir] = sort.split('_');
        filtered.sort((a, b) => {
            if (field === 'rating') {
                return dir === 'asc'
                    ? (a.movieRating || 0) - (b.movieRating || 0)
                    : (b.movieRating || 0) - (a.movieRating || 0);
            }
            if (field === 'release') {
                return dir === 'asc'
                    ? new Date(a.movieRelease) - new Date(b.movieRelease)
                    : new Date(b.movieRelease) - new Date(a.movieRelease);
            }
            return 0;
        });
    }

    renderMovies(filtered);
};
```

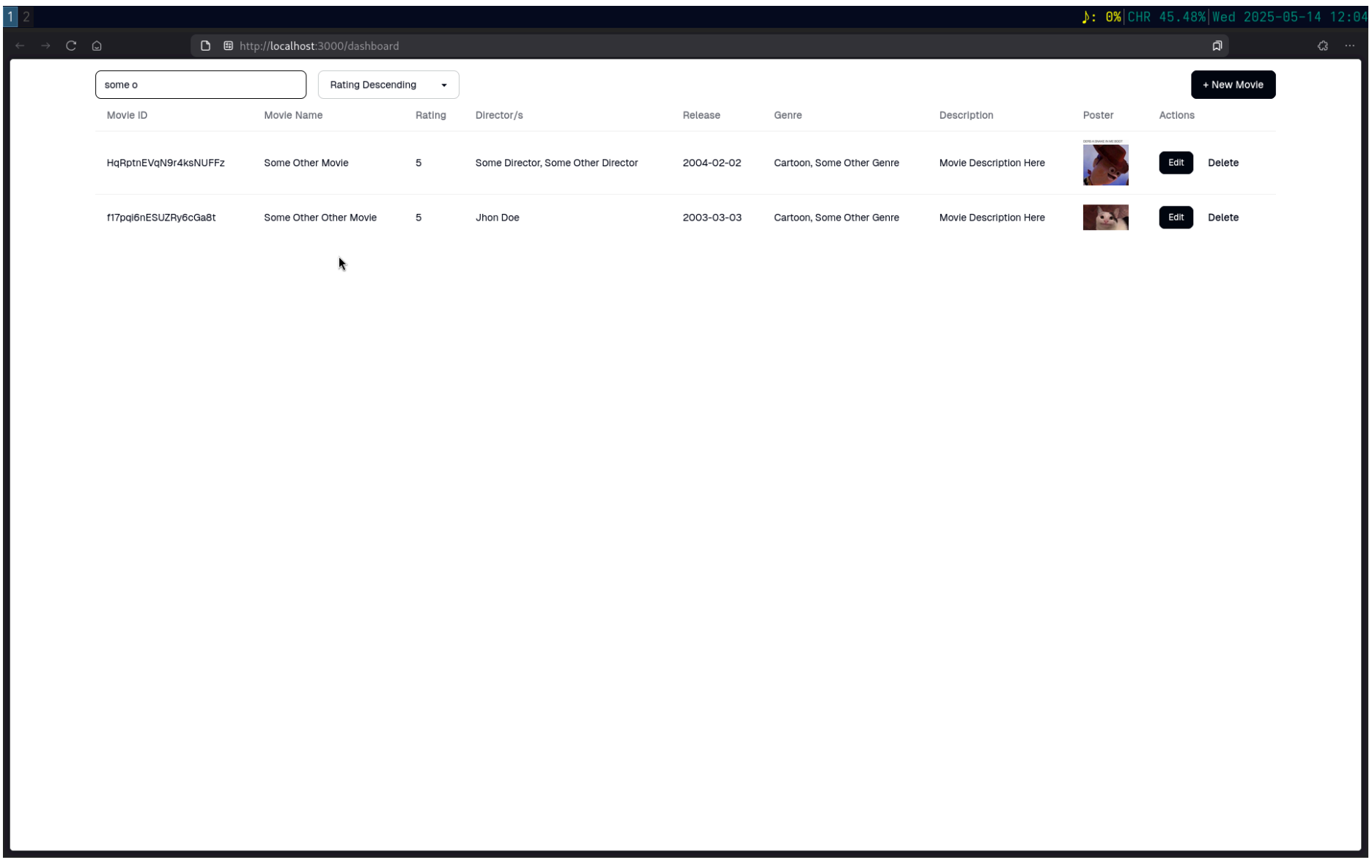
THats the entirety of the `filterMovies` function, this single function handles both search and sort. The `filterMovies` function filters the movies array based on the search term and sorts it by either rating or release date, depending on the selected option. It then updates the UI by calling `renderMovies` with the filtered and sorted results.

Testing Filters:

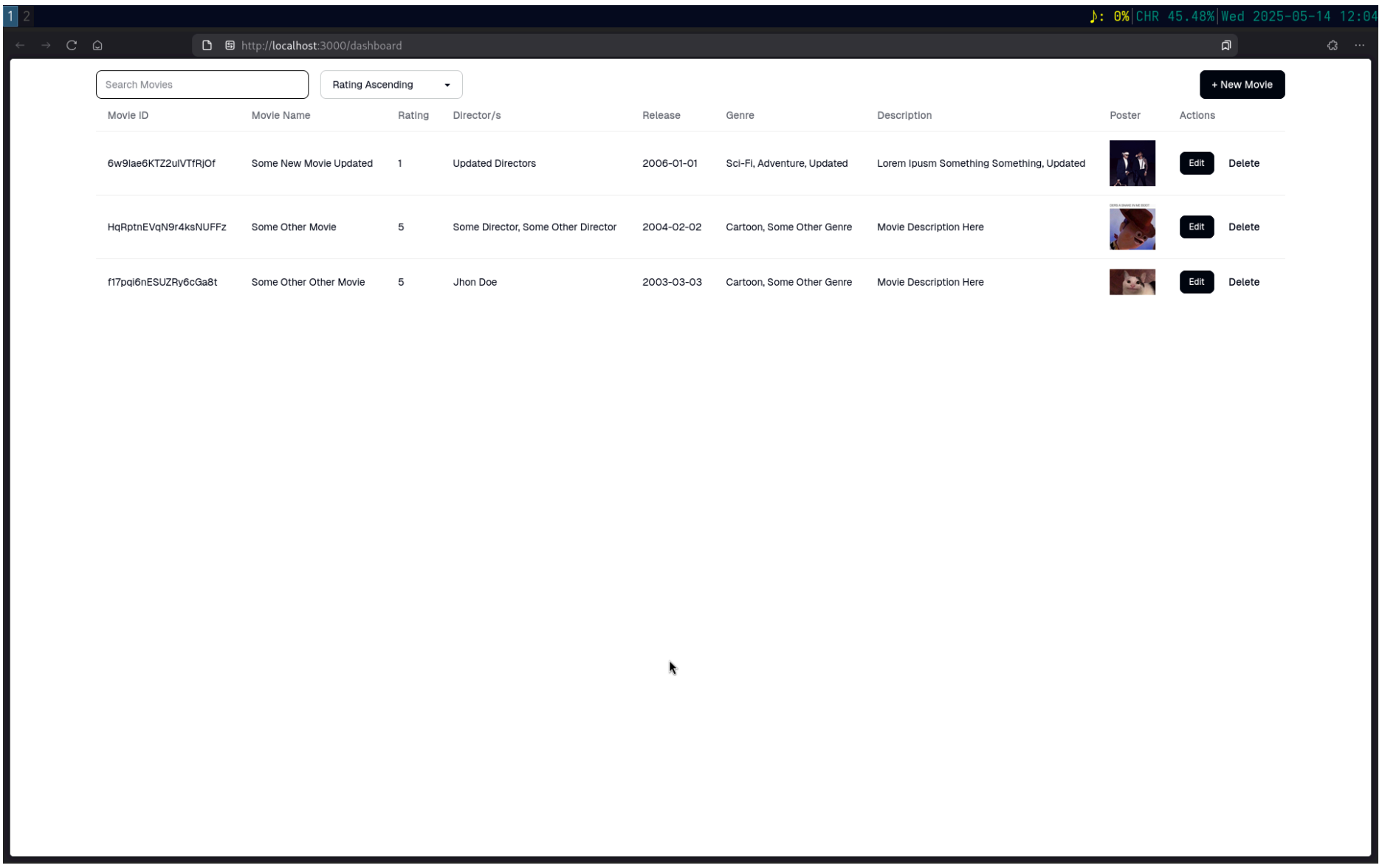
Initial State:



Searching Something:



Sort By Rating Ascending:



Likewise all other sort options work. Now let's create a navigation bar for easy navigation, throw in some better styling and animations and end the task.

Creating Navigation Bar:

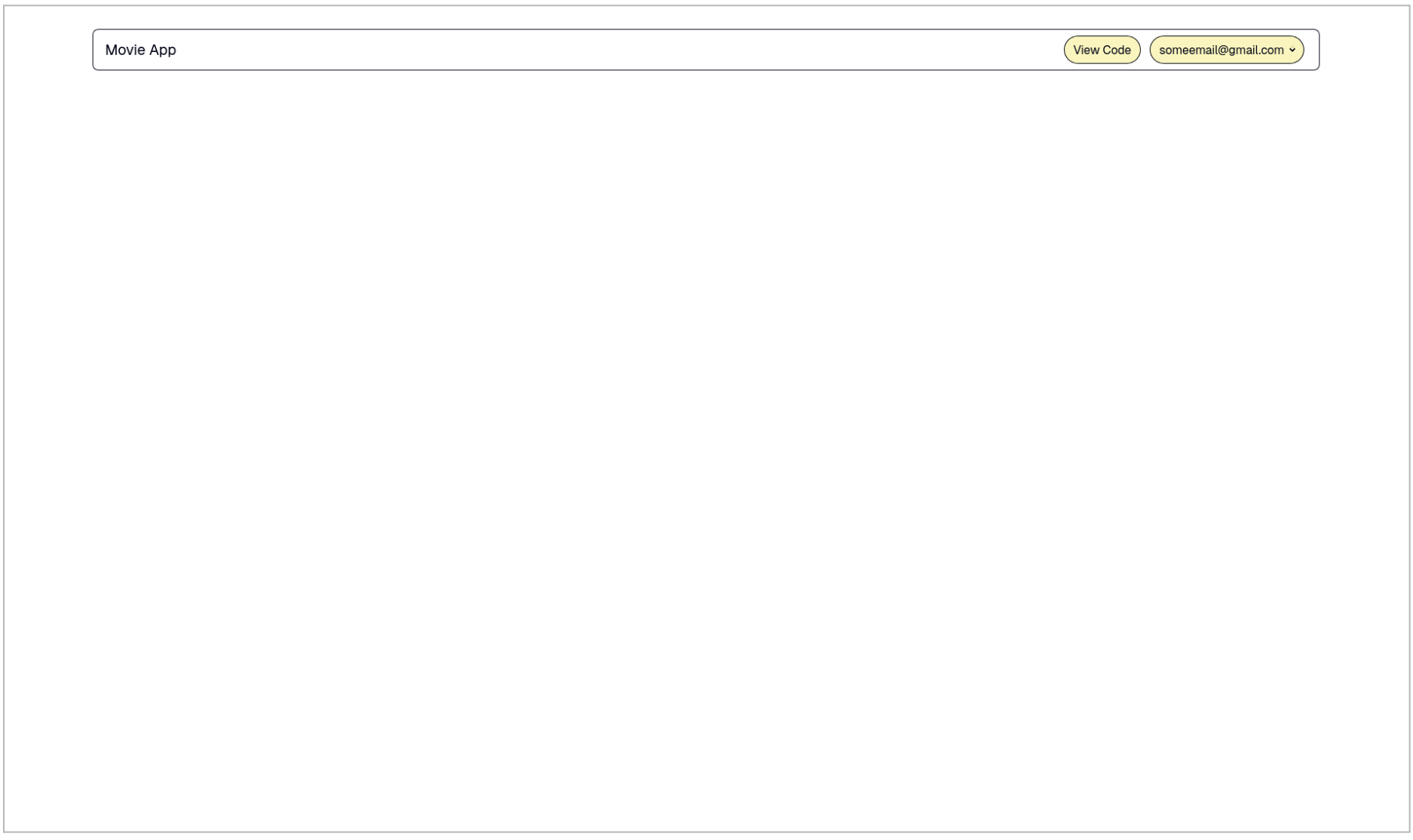
The navigation bar is pretty simple, one caveat we have is that it needs to be reactive, so there needs to be 2 states of the navigation bar, one state where the user is logged in and another where the user isn't. To do this, I have created a function:

```
function updateNavbar(user) {
    const navbarAuth = document.getElementById("navbar-auth");
    if (!navbarAuth) return;

    if (user) {
        navbarAuth.innerHTML = `
            <details class="dropdown dropdown-end">
                <summary class="btn btn-sm bg-yellow-100 text-black font-medium">${user.email}</summary>
                <ul class="p-2 shadow menu dropdown-content bg-base-100 rounded-box w-52 z-[999]">
                    <li><a id="logout-btn">Logout</a></li>
                </ul>
            </details>
        `;

        document.getElementById("logout-btn").onclick = () => {
            auth.signOut().then(() => location.reload());
        };
    } else {
        navbarAuth.innerHTML = `
        <div class="flex gap-2">
            <a class="btn btn-sm bg-yellow-100 text-black font-medium" href="/login.html">Login</a>
            <a class="btn btn-sm btn-outline" href="/signup.html">Signup</a>
        </div>
        `;
    }
}

onAuthStateChanged(auth, async (user) => {
    updateNavbar(user);
});

updateNavbar();
```

That simply makes use of the `onAuthStateChanged` function that we previously looked at to either dynamically insert the user's email with a drop down to log out, or simply links to login and signup. The HTML for the navigation looks like:

```
<div class="px-30 py-8 navbar sticky top-0 z-50 bg-transparent backdrop-blur-md">
    <div class="flex justify-between items-center w-full bg-white rounded-lg px-4 border">

        <!-- Left: Movie App -->
        <div class="flex-1">
            <a class="text-xl nav-link tooltip tooltip-bottom cursor-pointer" data-tip="Built By Swoyam_Pokharel_2431342" href="/index.html">
                Movie App
            </a>
        </div>

        <!-- Right: View Code + Auth Stuff -->
        <div class="flex-1 justify-end flex">
            <ul class="menu menu-horizontal px-1 flex gap-3 items-center">
                <!-- Always visible -->
                <li>
                    <a class="rounded-full border bg-yellow-100 text-base font-medium nav-link tooltip tooltip-bottom"
                       data-tip="Find The Code At My Github"
                       href="https://github.com/PS-Wizard/School/tree/main/Distributed_Systems/Swoyam_Pokharel_2431342">
                        View Code
                    </a>
                </li>

                <!-- Auth buttons here -->
                <li id="navbar-auth" class="flex gap-2"></li>
            </ul>
        </div>
    </div>
</div>
```

It's almost identical to the navigation bar I used in the task 2. So the final navigation bar looks like this when the user is logged in:



And this when the user isn't

Now let’s update all of the pages with the navigation, improve the stylings a bit and throw in some animations.

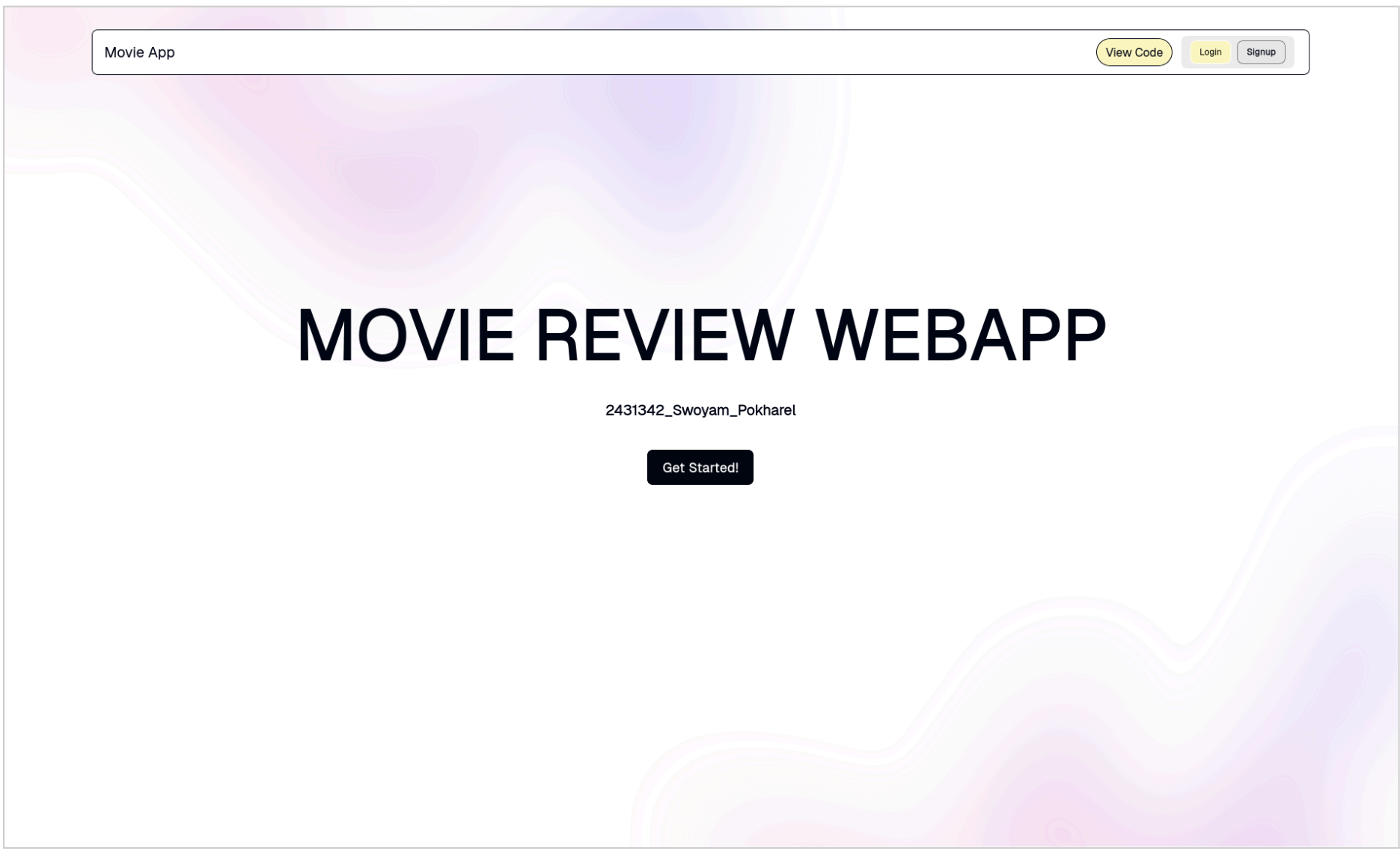
Finalized Code:

Updated Landing Page:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="/static/output.css" />
        <title>2431342 Movie App</title>
    </head>
    <body class="h-screen">
        <div class="absolute inset-x-0 -top-40 -z-10 transform-gpu overflow-hidden blur-3xl sm:-top-80" aria-hidden="true">
            <div class="relative left-[calc(50%-11rem)] aspect-1155/678 w-[36.125rem] -translate-x-1/2 rotate-[30deg] bg-linear-to-tr from-[#ff80b5] to-[#9089fc] opacity-30 sm:left-[
        </div>
        <!-- Navbar -->
        <div class="px-30 py-8 navbar sticky top-0 z-50 bg-transparent backdrop-blur-md">
    <div class="flex justify-between items-center w-full bg-white rounded-lg px-4 border">

        <!-- Left: Movie App -->
        <div class="flex-1">
            <a class="text-xl nav-link tooltip tooltip-bottom cursor-pointer" data-tip="Swoyam_Pokharel_2431342" href="/index.html">
                Movie App
            </a>
        </div>

        <!-- Right: View Code + Auth Stuff -->
        <div class="flex-1 justify-end flex">
            <ul class="menu menu-horizontal px-1 flex gap-3 items-center">
                <!-- Always visible -->
                <li>
                    <a class="rounded-full border bg-yellow-100 text-base font-medium nav-link tooltip tooltip-bottom"
                       data-tip="Find The Code At My Github"
                       href="https://github.com/PS-Wizard/School/tree/main/Distributed_Systems/Swoyam_Pokharel_2431342">
                        View Code
                    </a>
index.html
E492: Not an editor command: W
[0] 0:bunx- 1:nvim*
```

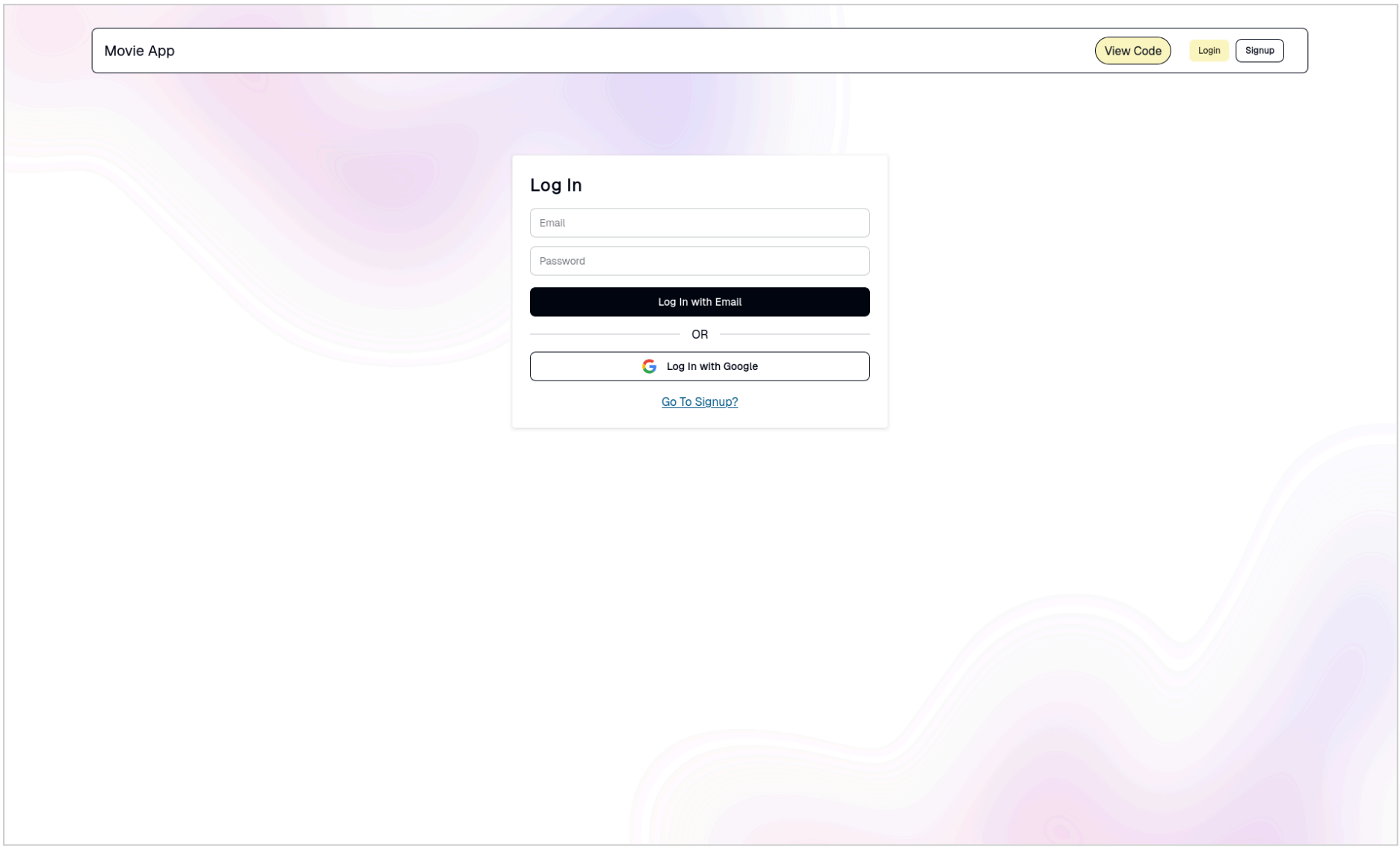
```
1 2                                                                                    ♪: 0% BAT 28.94% Wed 2025-05-14 16:13

32 |           <a class="btn btn-primary btn-lg nav-link" href="/signup.html">Get Started!</a>
31 |       </section>
30 |       <div class="absolute inset-x-0 top-[calc(100%-13rem)] -z-10 transform-gpu overflow-hidden blur-3xl sm:top-[calc(100%-30rem)]" aria-hidden="true">
29 |           <div class="relative left-[calc(50%+3rem)] aspect-1155/678 w-[36.125rem] -translate-x-1/2 bg-linear-to-tr from-[#ff80b5] to-[#9089fc] opacity-30 sm:left-[calc(50%+36rem)]
28 |       </div>
27 |       <script src="/static/js/navbar.js" type="module"></script>
26 |       <script src="https://cdn.jsdelivr.net/npm/gsap@3.13.0/dist/gsap.min.js"></script>
25 |       <script src="https://cdn.jsdelivr.net/npm/gsap@3.13.0/dist/SplitText.min.js"></script>
24 |
23 |       <script>
22 |           document.addEventListener("DOMContentLoaded", () => {
21 |
20 |               // navigation links fly in
19 |               gsap.from(gsap.utils.toArray(".nav-link"), {
18 |                   y: -20,
17 |                   opacity: 0,
16 |                   stagger: 0.1,
15 |                   delay: 0.3,
14 |                   duration: 0.6,
13 |                   ease: "power2.out"
12 |               });
11 |               const split = new SplitText("#hero-text", { type: "chars, words" });
10 |               gsap.from(split.chars, {
 9 |                   opacity: 0,
 8 |                   y: 50,
 7 |                   stagger: 0.05,
 6 |                   ease: "power4.out",
 5 |                   duration: 1.2,
 4 |               });
 3 |           });
 2 |       </script>
 1 |   </body>
78  </html>
index.html
E492: Not an editor command: W
[0] 0:bunx- 1:nvim*
```

Now the landing page includes the navigation bar, the gradient effect as the one I used in task 2 and has a basic hero text. This is how the final landing page looks:



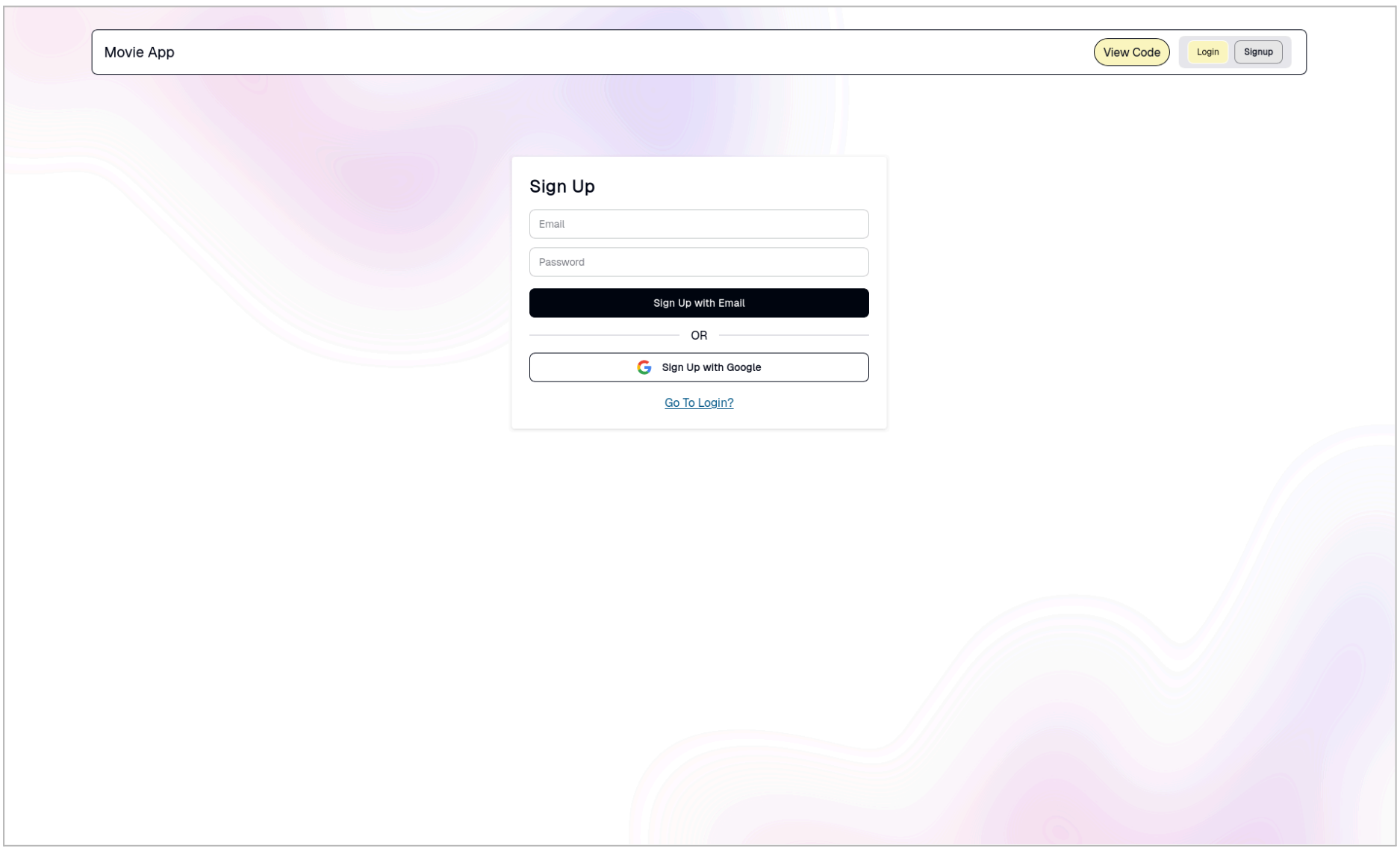
Updated Login Page:

The login page now has the same gradient effect as the landing page and includes the navigation bar too. Apart from that, there is now a link to the signup page. Those are the only changes in the login page: This is how the updated login page looks:



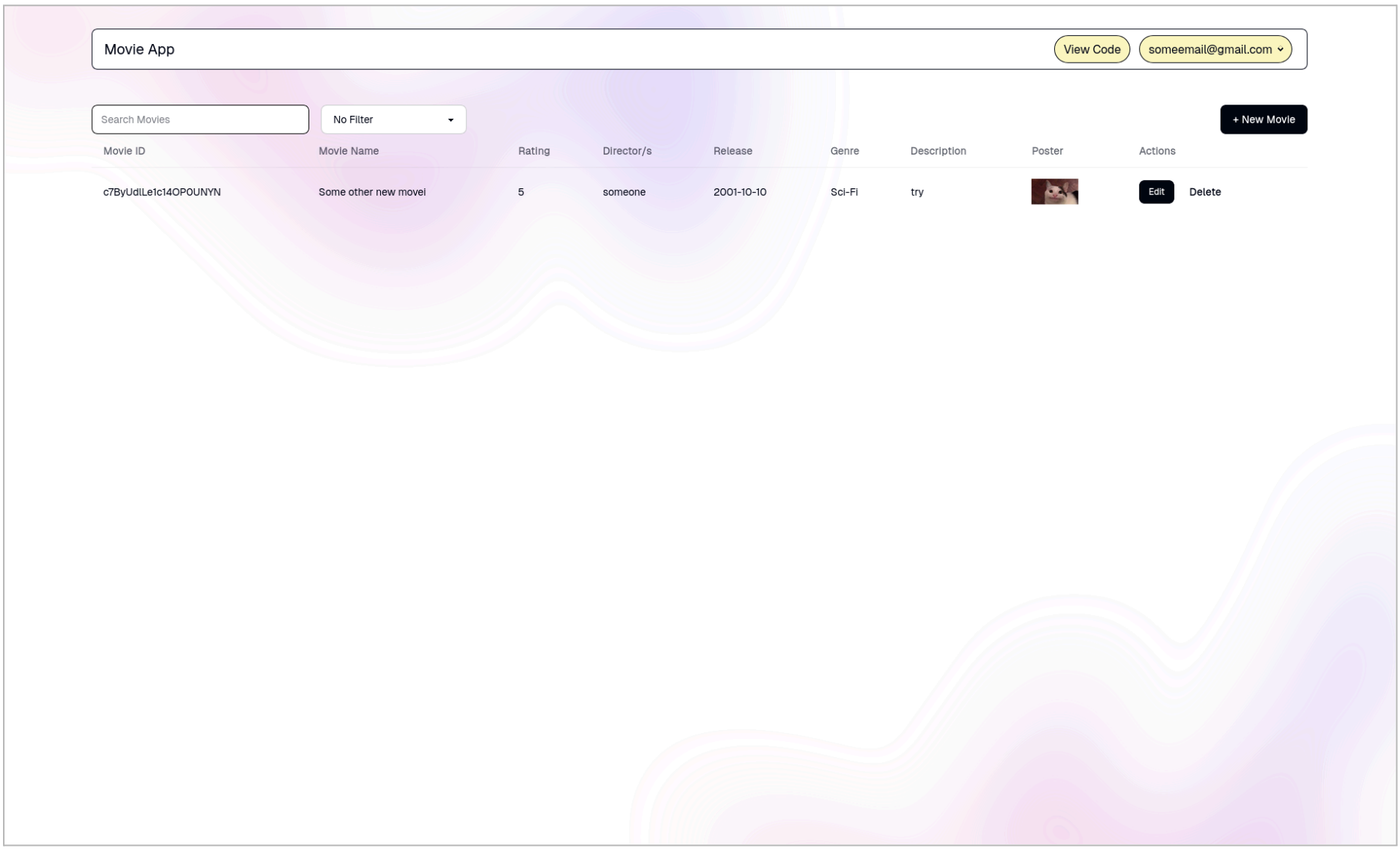
Updated Signup Page

Similar to the login page, the signup page has the same gradient effected, the navigation bar and a link to the login page



Updated Dashboard:

Finally, the dashboard now looks like:



It features the same background gradient and the navigation bar.

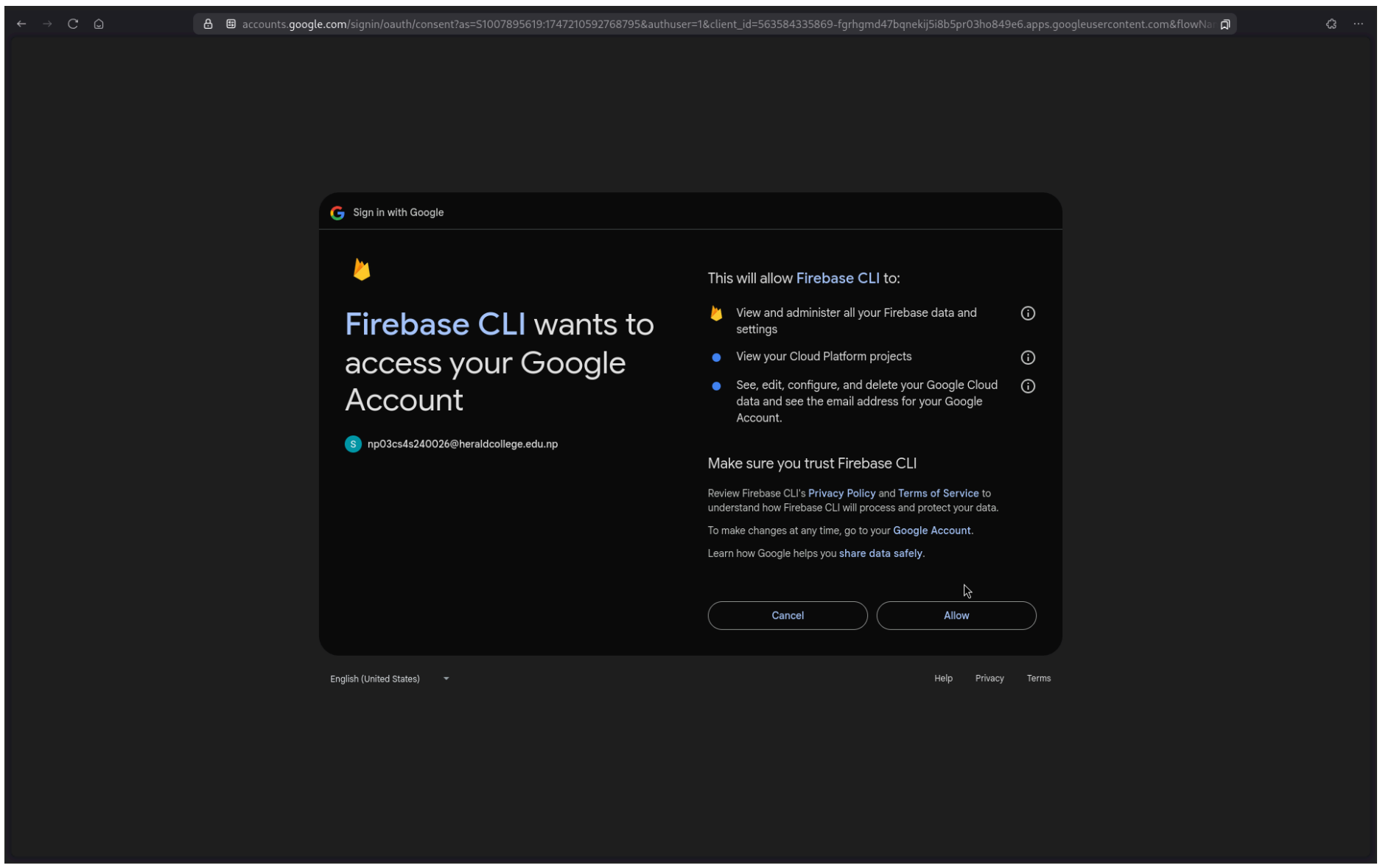
Hosting:

Now that everything is done, let's host this in firebase:

```
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ bun install -g firebase-tools
bun add v1.2.11 (cb6abd21)

installed firebase-tools@14.3.1 with binaries:
 - firebase

614 packages installed [599.00ms]

Blocked 1 postinstall. Run `bun pm -g untrusted` for details.
[wizard@archlinux task4]$

[0] 0:bun*
```

Firstly we need the `firebase-tools` package; I used `bun` to install it

```
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ bun install -g firebase-tools
bun add v1.2.11 (cb6abd21)

installed firebase-tools@14.3.1 with binaries:
 - firebase

614 packages installed [599.00ms]

Blocked 1 postinstall. Run `bun pm -g untrusted` for details.
[wizard@archlinux task4]$ firebase login
i  Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our products. Data is collected in accordance with Google's privacy policy (h
ttps://policies.google.com/privacy) and is not used to identify you.

✓ Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? No

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=425794084&redi
rect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

✓  Success! Logged in as np03cs4s240026@heraldcollege.edu.np
[wizard@archlinux task4]$

[0] 0:node*
```



Then we use the cli to login to our account. Now we can proceed to initialize a new firebase project:

```
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##      ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  /home/wizard/School/Distributed_Systems/Swoyam_Pokharel_2431342/task4

? Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices.
 ○ Genkit: Setup a new Genkit project with Firebase
 ○ Functions: Configure a Cloud Functions directory and its files
 ○ App Hosting: Configure an apphosting.yaml file for App Hosting
❯◉ Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
 ○ Storage: Configure a security rules file for Cloud Storage
 ○ Emulators: Set up local emulators for Firebase products
 ○ Remote Config: Configure a template file for Remote Config

[0] 0:node*
```

Here we we select `hosting` because we are using this to host the application that we already made.

```
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##      ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  /home/wizard/School/Distributed_Systems/Swoyam_Pokharel_2431342/task4

✓ Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices. Hosting: Configure files for Firebase Hosting and
(optionally) set up GitHub Action deploys

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: (Use arrow keys)
❯ Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project

[0] 0:node*
```

And here as mentioned above, we have already made the application so we choose ‘Use an existing project’

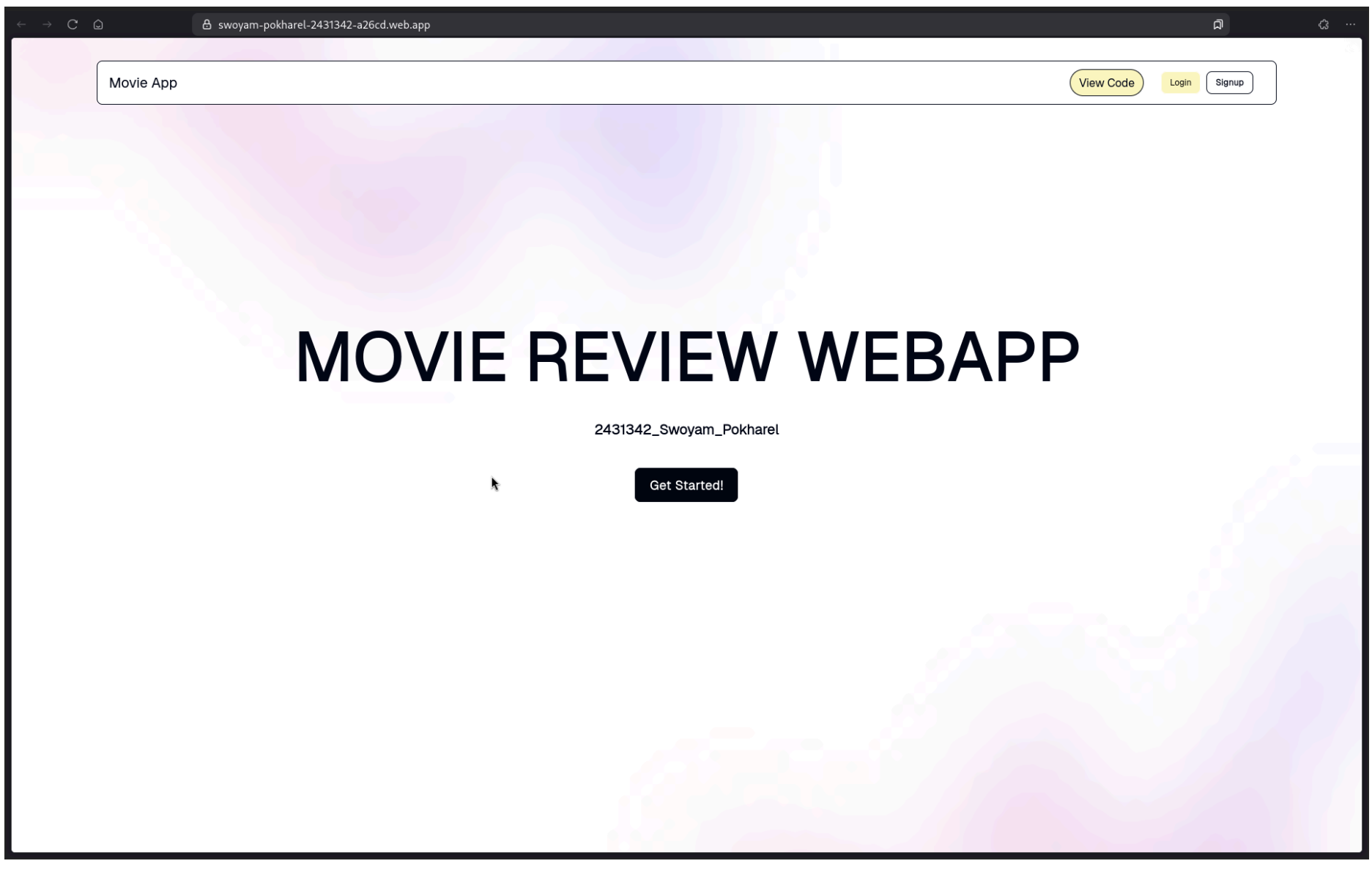
```
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  index.html  login.html  signup.html  static/
[wizard@archlinux task4]$ firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##      ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  /home/wizard/School/Distributed_Systems/Swoyam_Pokharel_2431342/task4

✓ Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices. Hosting: Configure files for Firebase Hosting and
(optionally) set up GitHub Action deploys

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

✓ Please select an option: Use an existing project
✓ Select a default Firebase project for this directory: swoyam-pokharel-2431342-a26cd (Swoyam-Pokharel-2431342)
i  Using project swoyam-pokharel-2431342-a26cd (Swoyam-Pokharel-2431342)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? (public)



[0] 0:node*
```

Here we select our existing project and specify that we want everything inside the public directory to be hosted. So i simply copied all the files that i have into the public directory:

```
[wizard@archlinux public]$ cd ..
[wizard@archlinux task4]$ ls
dashboard.html  firebaseconfig.js  firebase-debug.log  index.html  login.html  public/  signup.html  static/
[wizard@archlinux task4]$ tree
.
├── dashboard.html
├── firebaseconfig.js
├── firebase-debug.log
├── index.html
├── login.html
├── public
│   ├── dashboard.html
│   ├── index.html
│   ├── login.html
│   ├── signup.html
│   └── static
│       ├── input.css
│       ├── js
│       │   ├── dashboard.js
│       │   ├── login.js
│       │   ├── navbar.js
│       │   └── signup.js
│       └── output.css
├── signup.html
└── static
    ├── input.css
    ├── js
    │   ├── dashboard.js
    │   ├── login.js
    │   ├── navbar.js
    │   └── signup.js
    └── output.css

6 directories, 22 files
[wizard@archlinux task4]$


[0] 0:node- 1:bash*
```

```
You're about to initialize a Firebase project in this directory:

  /home/wizard/School/Distributed_Systems/Swoyam_Pokharel_2431342/task4

✓ Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices. Hosting: Configure files for Firebase Hosting and
(optionally) set up GitHub Action deploys

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

✓ Please select an option: Use an existing project
✓ Select a default Firebase project for this directory: swoyam-pokharel-2431342-a26cd (Swoyam-Pokharel-2431342)
i  Using project swoyam-pokharel-2431342-a26cd (Swoyam-Pokharel-2431342)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

✓ What do you want to use as your public directory? public
✓ Configure as a single-page app (rewrite all urls to /index.html)? No
✓ Set up automatic builds and deploys with GitHub? No
✓  Wrote public/404.html
✓ File public/index.html already exists. Overwrite? No
i  Skipping write of public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...
i  Writing gitignore file to .gitignore...

✓  Firebase initialization complete!
[wizard@archlinux task4]$
[0] 0:node* 1:bash-
```

And after the setups are done, we now have a link to our hosted app!



The task is now complete and can be accessed at: https://swoyam-pokharel-2431342-a26cd.web.app

In the case where creating a new account is not desirable, the following credentials may be used to login:

Email: someemail@gmail.com
Password: somepass

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU