DIT 637 Smart and Secure Systems
TT03 Experiencing MERN - A Full-Stack Mobile App
06/20/2024 Developed by Clark Ngo
07/03/2024 Reviewed by Sam Chung
07/31/2025 Rajdeep Singh Golan

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Key Concepts and Tools for Full-Stack – MERN (MongoDB, Express, React Native, Node)

Full-Stack – MERN (MongoDB, Express, React Native, Node)
Developing a full-stack mobile app involves creating an application from beginning to end. This includes designing the user interface (frontend) and implementing the server-side logic (backend) that powers all the functionality behind the scenes.

Frontend
To create the visual part of a mobile app that users interact with
- **React Native**: To efficiently build dynamic and responsive user interfaces for mobile apps.

Backend
The backend of a web application is responsible for the server-side logic, handling database interactions, user authentication, and application logic. It processes incoming requests, applies business rules, accesses databases, and sends responses back to the client.
- **Express or ExpressJS**: A minimal and flexible Node.js web application framework that provides robust features for web and mobile applications. It is used to build the server-side logic of web applications.

Database
A database is an organized collection of data generally stored and accessed electronically from a computer system. It is essential for storing and managing large amounts of data efficiently. There are two types: SQL and NoSQL DB.
- **MongoDB Atlas**: A cloud-based NoSQL database service for modern applications, providing a fully managed and global cloud database with built-in automation for resource and workload management.

Development Environment
Codespaces: To provide a cloud-based development environment using containerization that is easy to set up and use from anywhere, similar to Google Docs, but for coding.

Version Control and Collaboration
GitHub as Repo: To store and manage code with version control, making collaboration more manageable, just like using Google Drive for sharing documents.

Example: Movie Search Application
To practice and demonstrate the capabilities of React Native and Express in creating real-world applications, like searching for movies on Netflix.

TASK: Uploading three image files to your GitHub Repository generated from GitHub Classroom

1. The screenshot of your 'Network Access' as '*first_last_*network_access.png' by using your first and last name.
2. The screenshot of your backend Express web application as '*first_last_*backend.png' by using your first and last name.
3. The screenshot of your frontend mobile app as '*first_last_*frontend.jpg' by using your first and last name.

1) User Case

As a movie enthusiast using a **mobile device**, I want to search and browse a list of movies with details such as title, genre, and year so that I can easily find information about movies I am interested in while on the go.

Note: The user story didn't change from the previous TT01 and TT02. The question then is how a full-stack mobile app works with DevOps Tools (GitHub and GitHub Codespaces).

Why Do We Need a Backend and Database?

1. **Data Management:**
   - **Database:** Storing and managing large amounts of movie data (titles, genres, years, etc.) efficiently. A database like MongoDB Atlas is ideal for handling such documents and providing quick access to them.
   - **Backend:** Facilitating the retrieval, manipulation, and updating of data from the database. The backend (using Express) acts as an intermediary that requests from the frontend, queries the database, and sends the relevant data back to the frontend.
2. **Business Logic:**
   - **Backend:** Encapsulating the business logic required to process data and perform operations such as filtering movies by genre, searching for specific titles, and handling user preferences. It separates the logic from the frontend, making the application more modular and easier to maintain.

Why Do We Need Full-Stack Development?

1. **Limited Data Handling:**
   - React Native alone cannot efficiently handle large datasets or complex queries. It is primarily a frontend framework meant for building user interfaces, not for managing or processing data.
2. **No Data Persistence:**
   - Without a backend and database, data would be stored locally on the device, making it inaccessible if the user switches devices or uninstalls the app. A cloud database ensures that data is persistent and accessible from anywhere.
3. **Security Concerns:**
   - Storing and processing data directly on the client-side exposes it to security risks. A backend server ensures data is processed securely and can implement necessary security protocols.
4. **Separation of Concerns:**
   - Maintaining a clear separation between frontend and backend simplifies development and maintenance. It allows each part to be developed, tested, and scaled independently.

2) Project Setup

Creating our database in the cloud

MongoDB Atlas and Load Sample Data

4. Create an account in [MongoDB Atlas](#).
5. Click on "Create" under Cluster (If you already have a Project)
6. Create a New Project and follow along with the screenshot. Use any values when required and use any default values if given.
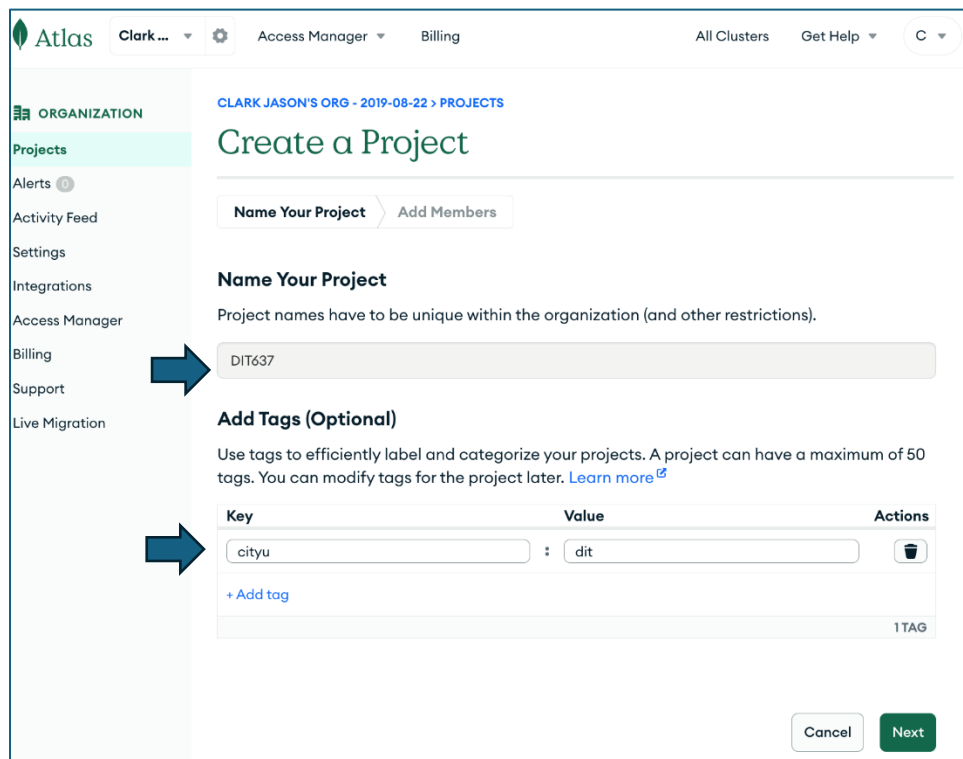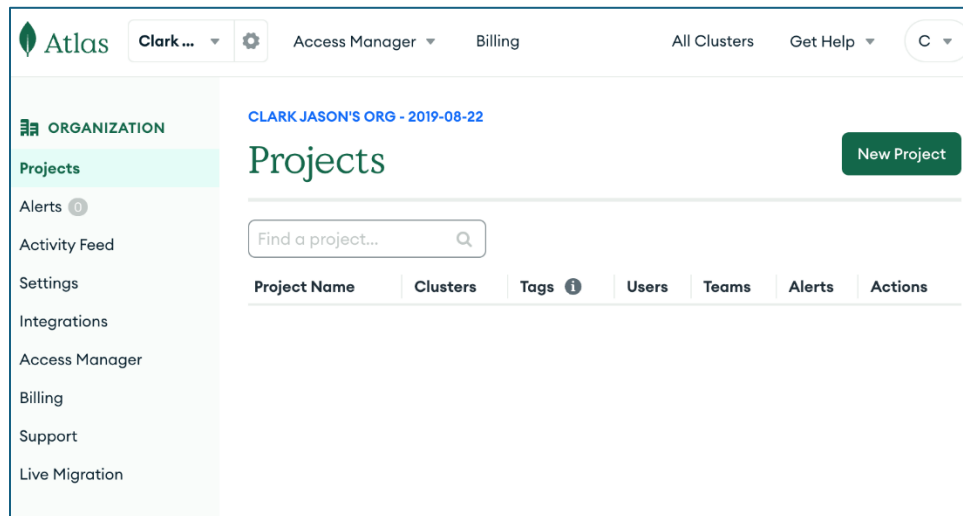
Use your email.
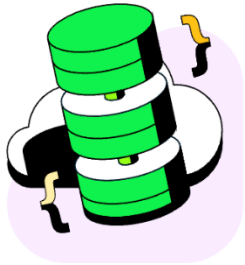
7. Choose M0 Free. Select the provider (AWS) and the region according to your location.

Select the AWS as provider and your region.

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

○ **M10**	$0.08/hour
For production applications with sophisticated workload requirements.	

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

○ **Serverless**	$0.10/1M reads
For application development and testing, or workloads with variable traffic.	

STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

● **M0**	Free
For learning and exploring MongoDB in a cloud environment.	

STORAGE	RAM	vCPU
512 MB	Shared	Shared

✓ **Free forever!** Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Name
You cannot change the name once the cluster is created.

Cluster0

☑ Automate security setup ⓘ

☑ Preload sample dataset ⓘ

Provider

aws	Google Cloud	Azure

Region

🇺🇸 **N. Virginia (us-east-1)** ★ 🌿 ▾

★ Recommended ⓘ   🌿 Low carbon emissions ⓘ

Tag (optional)
Create your first tag to categorize and label your resources; more tags can be added later. Learn more.

Select or enter key : Select or enter value

I'll do this later

Go to Advanced Configuration   Create Deployment

8. **Copy and save your password** (if you lose it, you can regenerate a new one in left pane side menu 'Database Access' under 'SECURITY').

Connect to Cluster0

① Set up connection security — ② Choose a connection method — ③ Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more ↗

1. **Add a connection IP address**

   ✓ Your current IP address (152.44.208.138) has been added to enable local connectivity. Add another later in Network Access ↗ .

2. **Create a database user**

   This first user will have atlasAdmin ↗ permissions for this project.

   We autogenerated a username and password. You can use this or create your own.

   ⓘ **You'll need your database user's credentials in the next step. Copy the database user password.**

   Username

   clarkngo

   Password

   HIDE   📋 Copy

   Create Database User

   Close   Choose a connection method

9. Click "Create Database User"

Connect to Cluster0

① Set up connection security — ② Choose a connection method — ③ Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more ↗

1. **Add a connection IP address**

   ✓ Your current IP address (152.44.208.138) has been added to enable local connectivity. Add another later in Network Access ↗ .

2. **Create a database user**

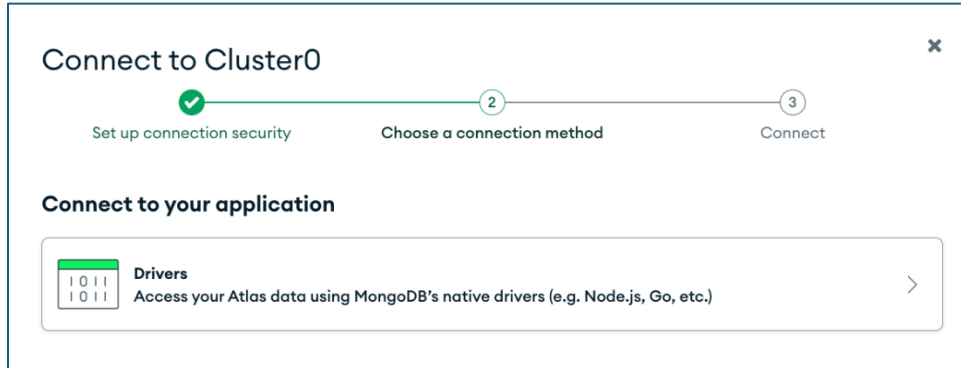   ✓ A database user has been added to this project. Create another user later in Database Access ↗ .

   You'll need your database user's credentials in the next step.

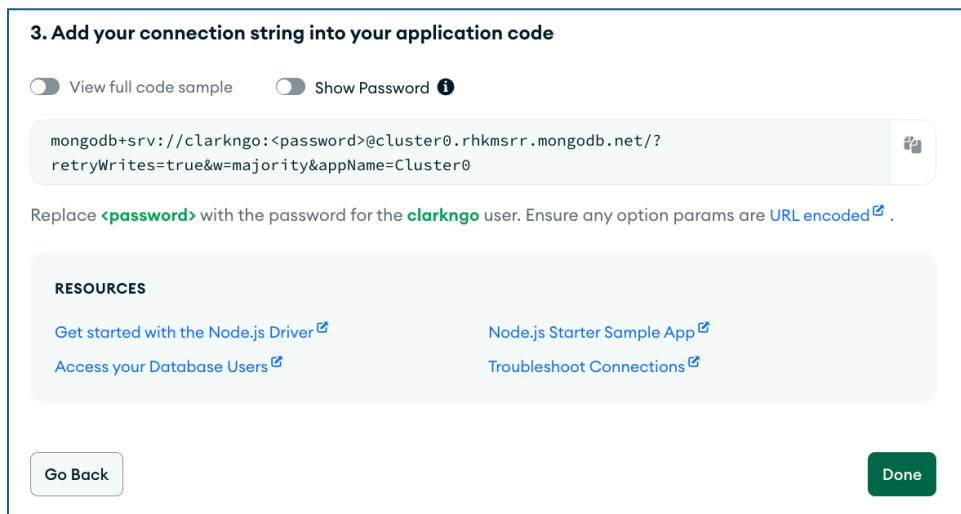   Close   Choose a connection method

10. Click "Choose a connection method", Select "Drivers"

9. Replace the <password> (including <>) with the password you copied when you created a user. Copy the 'mongodb' connection string in a notepad for now.
   Example:
   mongodb+srv://clarkngo:<password>@cluster0.pcbqf6b.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

10. Click the 'Browse Collection.' Or, please wait till Sample data is loaded (till you can see '**Browse Collection**')
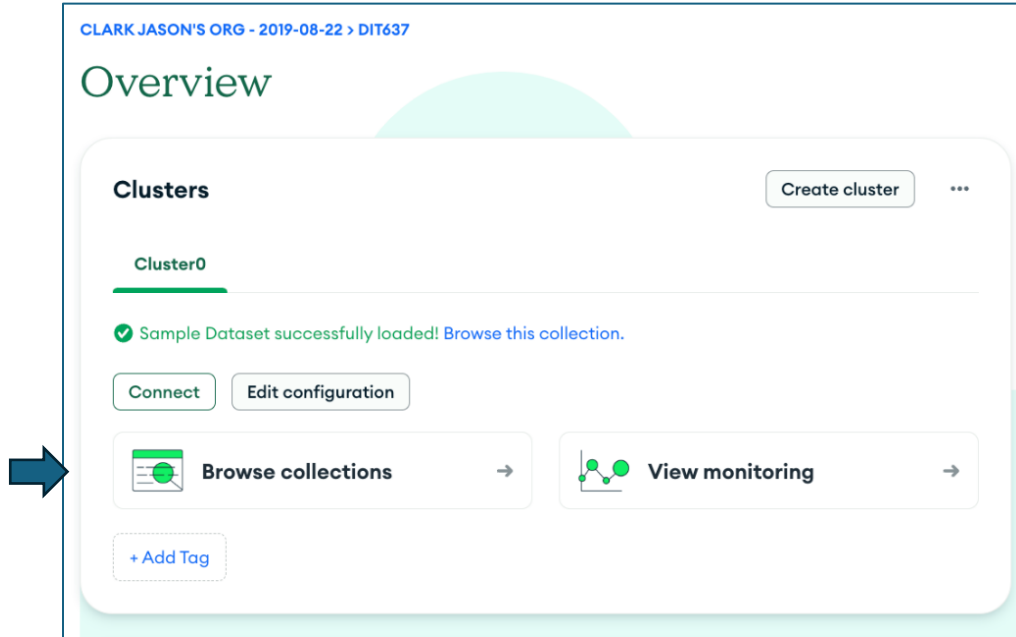
It should have a preloaded sample database: 'sample_mflix'

CLARK JASON'S ORG - 2019-08-22 › DIT637 › DATABASES

ClusterO

VERSION
7.0.11

REGION
AWS N. Virginia (us-east-1)

Overview   Real Time   Metrics   **Collections**   Atlas Search   Performance Advisor   Online A

DATABASES: 1 COLLECTIONS: 6

+ Create Database

Q Search Namespaces

▾ **sample_mflix**

| **comments**

embedded_movies

movies

sessions

theaters

users

VISUALIZE YOUR DATA   **REFRESH**

sample_mflix.comments

STORAGE SIZE: 5.7MB   LOGICAL DATA SIZE: TOTAL DOCUMENTS: INDEXES TOTAL SIZE:

11.14MB   41079   1.12MB

Find   Indexes   Schema Anti-Patterns ⓪   Aggregation   Search Indexes

Generate queries from natural language in Compass ⧉   **INSERT DOCUMENT**

Filter ⧉   Type a query: { field: 'v   Reset   **Apply**   Options ▸

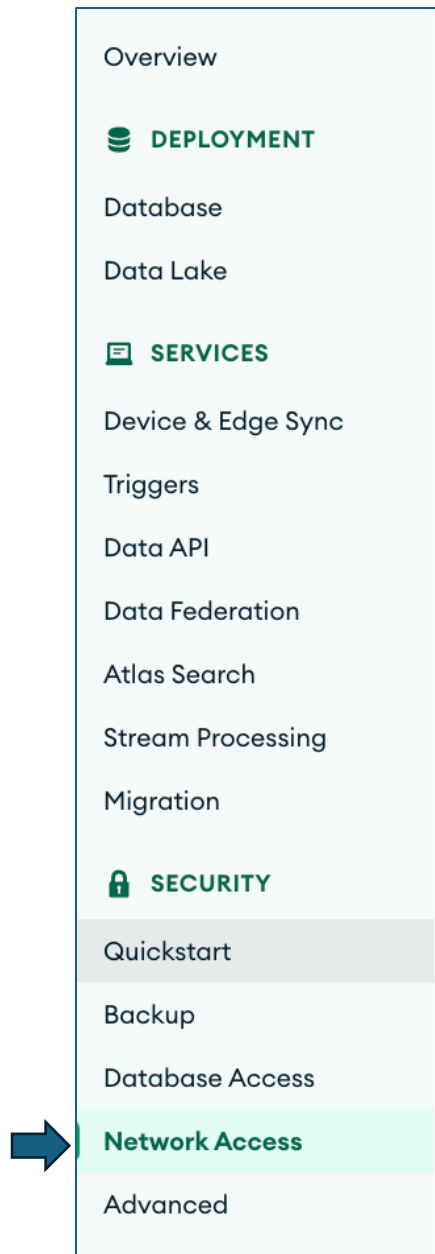QUERY RESULTS: **1-20 OF MANY**

```
_id: ObjectId('5a9427648b0beebeb69579e7')
name : "Mercedes Tyler"
email : "mercedes_tyler@fakegmail.com"
movie_id : ObjectId('573a1390f29313caabcd4323')
text : "Eius veritatis vero facilis quaerat fuga temporibus.
        Praesentium exped…"
date : 2002-08-18T04:56:07.000+00:00
```

```
_id: ObjectId('5a9427648b0beebeb69579f5')
name : "John Bishop"
email : "john_bishop@fakegmail.com"
movie_id : ObjectId('573a1390f29313caabcd446f')
text : "Id error ab at molestias dolorum incidunt. Non deserunt
        praesentium do…"
date : 1975-01-21T00:31:22.000+00:00
```
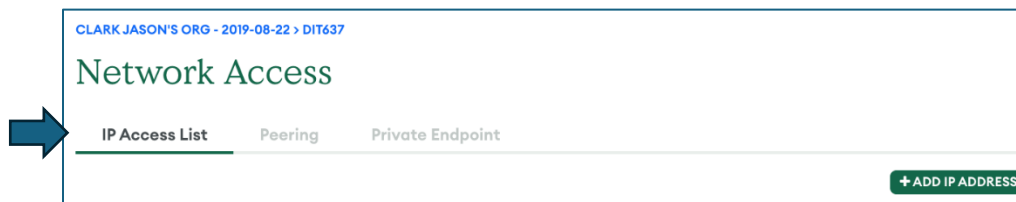
```
_id: ObjectId('5a9427648b0beebeb6957a21')
name : "Jaqen H'ghar"
email : "tom_wlaschiha@gameofthron.es"
movie_id : ObjectId('573a1390f29313caabcd516c')
text : "Minima odit officiis minima nam. Aspernatur id
        reprehenderit eius inve…"
date : 1981-11-08T04:32:25.000+00:00
```

11. Allow Network access. Go to Network Access under Security.

Overview

DEPLOYMENT

Database

Data Lake

SERVICES

Device & Edge Sync

Triggers

Data API

Data Federation

Atlas Search

Stream Processing

Migration

SECURITY

Quickstart

Backup

Database Access

Network Access

Advanced

12. Click the 'ADD IP ADDRESS.'

CLARK JASON'S ORG - 2019-08-22 › DIT637

Network Access

IP Access List   Peering   Private Endpoint

+ ADD IP ADDRESS

13. Add '0.0.0.0/0'

Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. Learn more.

ALLOW ACCESS FROM ANYWHERE

Access List Entry: 0.0.0.0/0

Comment: allow access

This entry is temporary and will be deleted in 6 hours   Cancel   Confirm

14. Have the screenshot of your 'Network Access' as 'sam_chung_network_access.png' by using your first and last name.
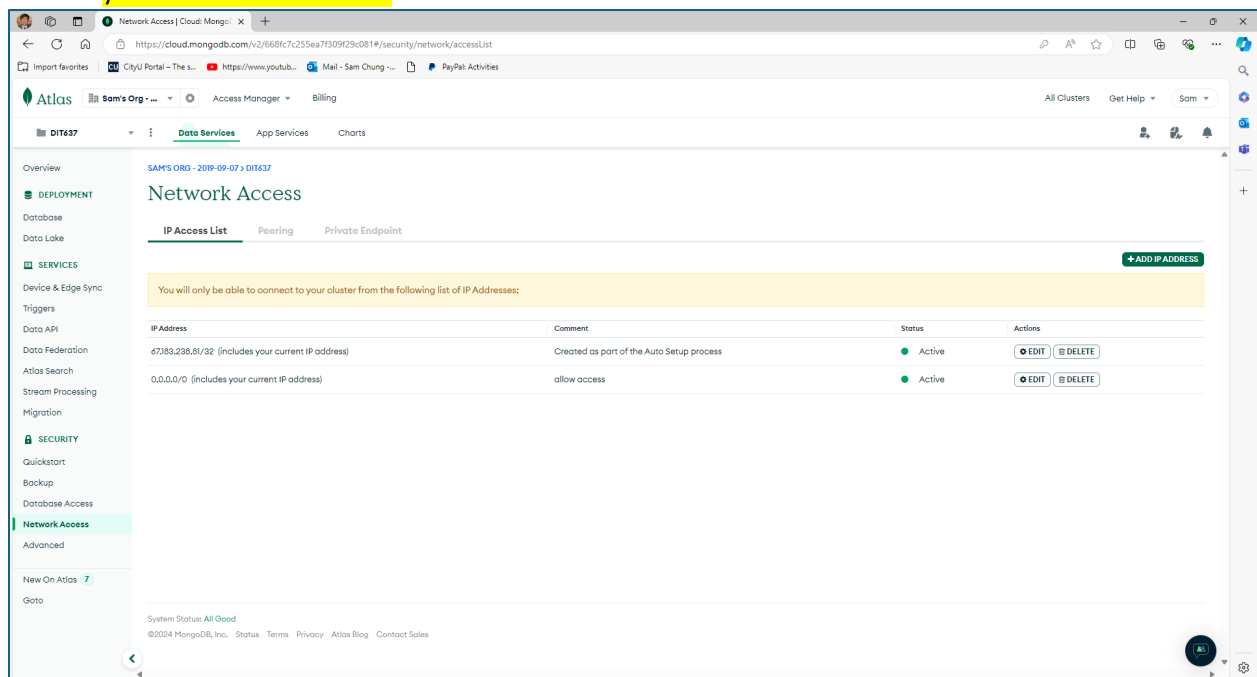
3) Accessing our development environment
1. Open the project in GitHub.
2. Create a cloud-based development project with GitHub Codespaces:

   a.

   b.

   c.
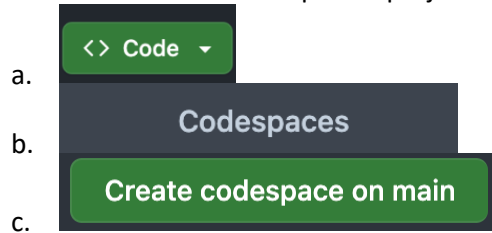
*Connecting our database to our backend and running the backend server*

ExpressJS Backend
1. Connect MongoDB:
   a. cd backend-expressjs
   b. Inside 'backend-expressjs' folder, **create '.env'** file (use the 'example.env' as a reference) and type the following:
      i. MONGODB_URI=mongodb+srv://<username>:<password>@<cluster-url>/<database-name>
   *Note: you should have copy pasted the mongodb connection string. If not, go back to previous steps.*
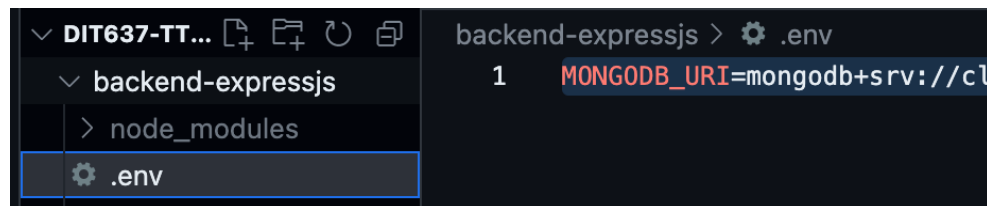
2. In the terminal type:
   a. npm install
   b. npm start
   c. Click 'Make Public'
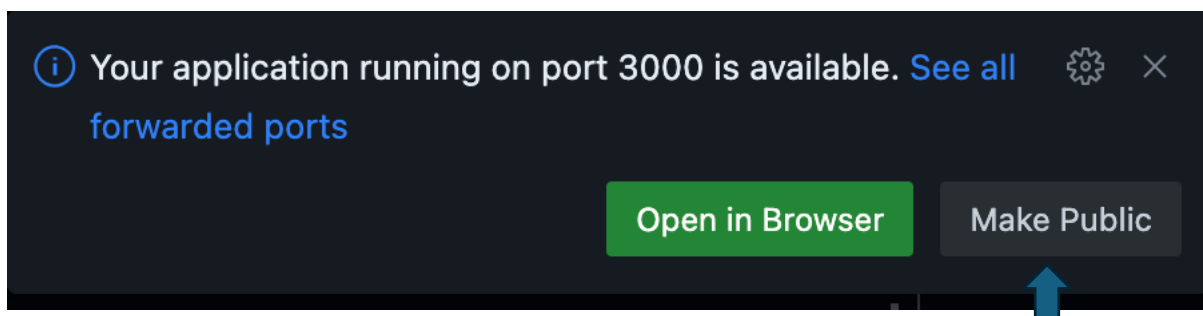
3. Go to the port and visit the website. If the visibility is not 'Public,' change it by right-clicking. If your server port's visibility is private, your mobile app will show the following error message: *"Error fetching movies: [SyntaxError: JSON Parse error: Unexpected character: <]"*
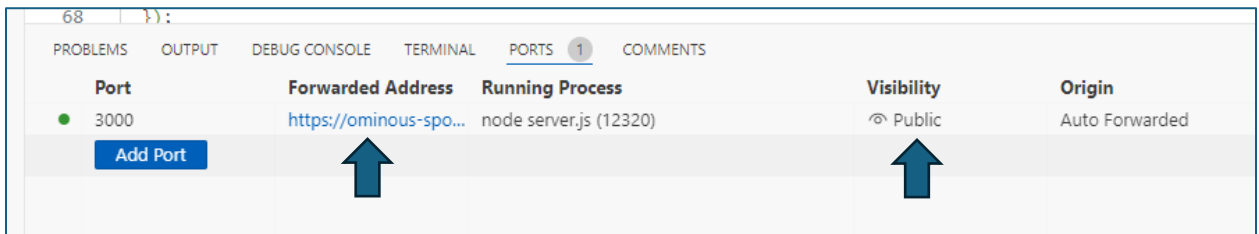
4. You may see the screen below. But your URL may be different.

5. Open the server-side web page in your browser.
   If you see the following screen, press the 'Continue' button.
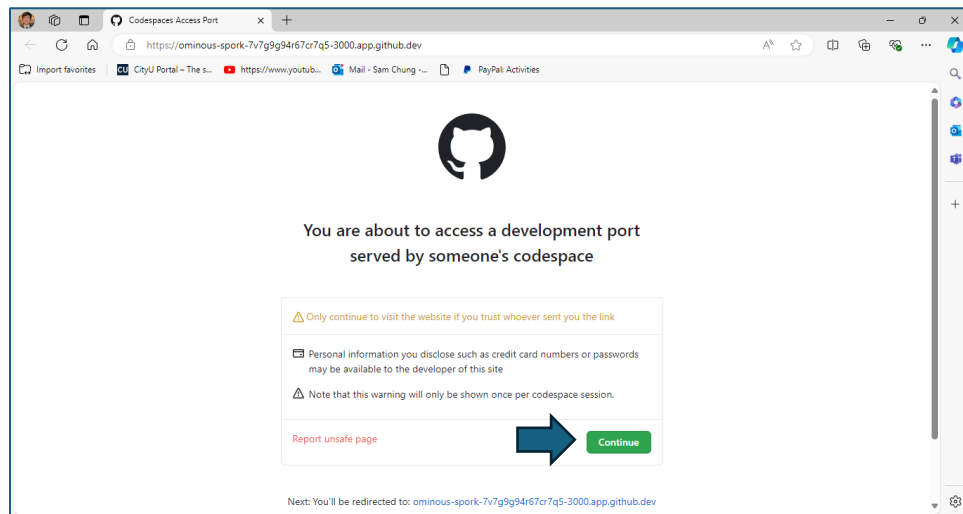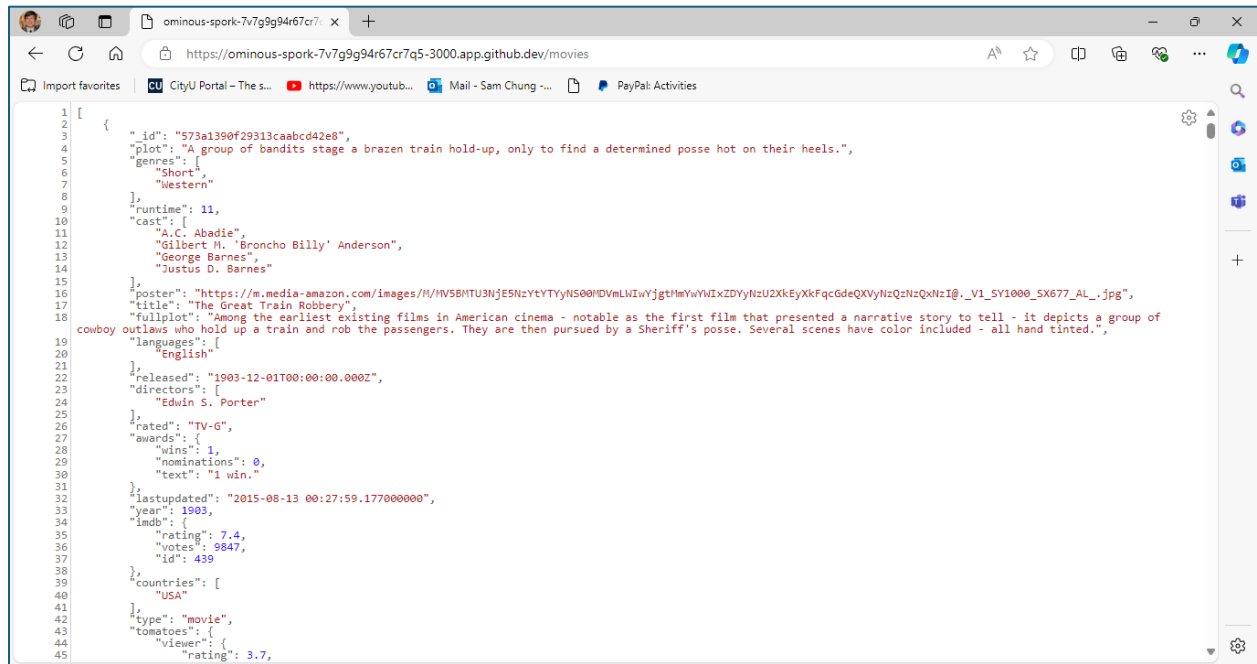

Have the screenshot of your backend Express web application as 'sam_chung_backend.png' by using your first and last name.
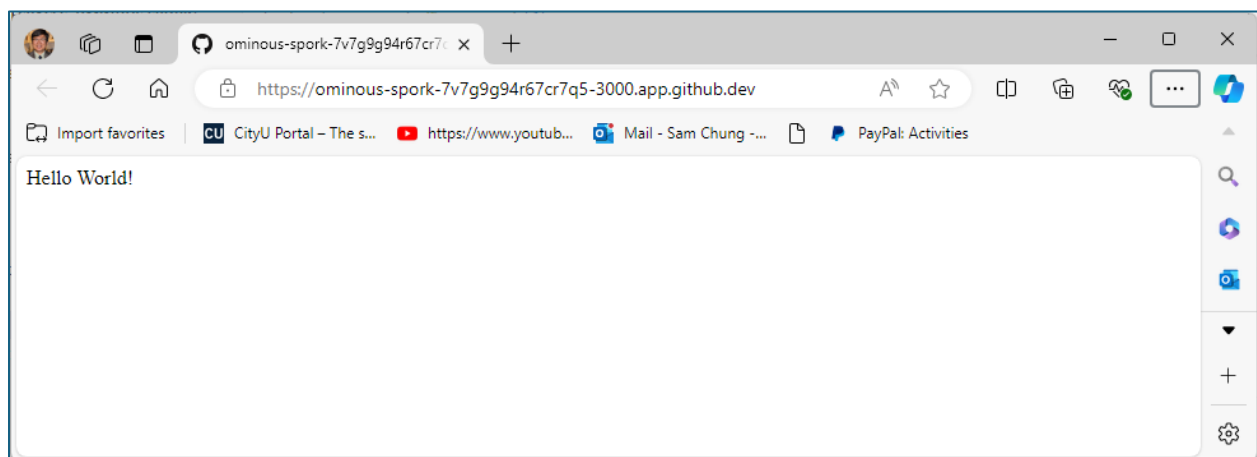
6. Let's test the list of movies. After the URL, type '/movies' and watch the displayed list.
   Example: https://ominous-spork-7v7g9g94r67cr7q5-3000.app.github.dev/**movies**

Connecting our backend to our frontend and running the frontend server

React Native Frontend

1. Open a new terminal.
2. cd frontend-reactive
3. **Create** your own **.env** inside the 'frontend-reactive' folder. You can follow the example in **example.env** file.

4. Copy the backend URL in the Ports tab.

	Port	Forwarded Address	Running Process
PROBLEMS	OUTPUT	DEBUG CONSOLE TERMINAL PORTS ③ COMMENTS	
●	3000	https://probable-fishstick-77q6jrvvpxrfwr7r-3000.app.github.dev/	node server.js (66526)

5. In your **.env** file, you have **API_URL** and your own 'backend-expressjs' address from GitHub codespaces.

```
## update this with your own backend–expressjs forwarded address
API_URL='https://probable–fishstick–77q6jrvvpxrfwr7r–3000.app.github.dev'
```

Note: the URL endpoint should **not** have a "**/**" at the end.

6. In the terminal, type the following:
   a. npm install
   b. npx expo login
      i. Note: you need to create an account in https://expo.dev/go
   c. npx expo start --tunnel

7. Download Expo Go in App Store or Google Play Store
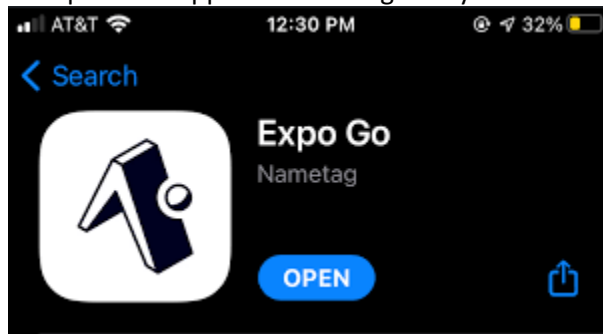
   a.
   b. Open then log in

Screen showing no local development server yet.
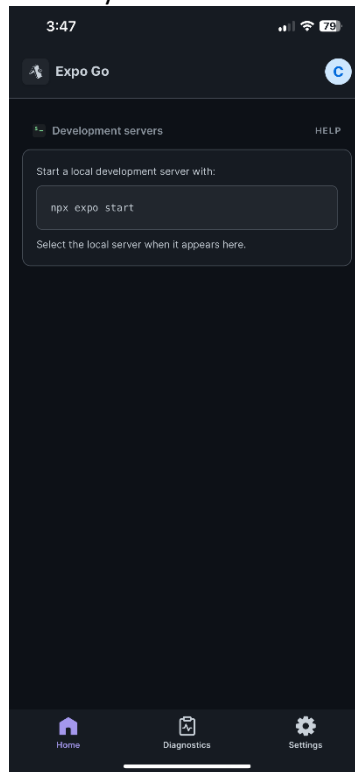
You can touch and drag down to refresh. You should see a development server like below:
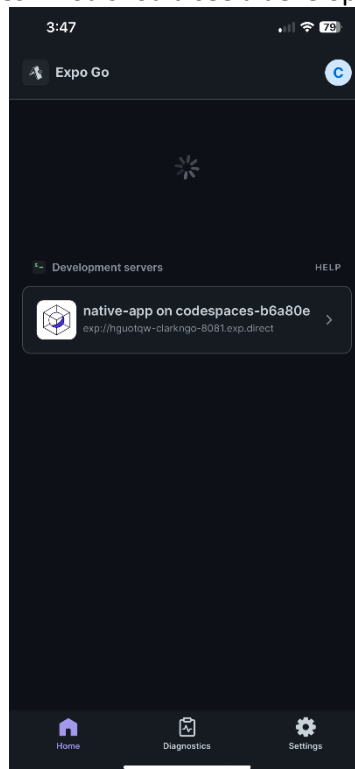
Movie List

Search movies...

The Ace of Hearts
Year: 1921
Genres: Crime, Drama, Mystery
Cast: Lon Chaney, Leatrice Joy, John Bowers, Hardee Kirkland

Peter Pan
Year: 1924
Genres: Adventure, Fantasy, Family
Cast: George Ali, Esther Ralston, Cyril Chadwick, Mary Brian

The Thief of Bagdad
Year: 1924
Genres: Adventure, Family, Fantasy
Cast: Douglas Fairbanks, Snitz Edwards, Charles Belcher, Julanne Johnston

Lady Windermere's Fan
Year: 1925
Genres: Comedy
Cast: Ronald Colman, May McAvoy, Bert Lytell, Irene Rich

Disraeli
Year: 1929
Genres: Biography, Drama, History
Cast: George Arliss, Doris Lloyd, David Torrence, Joan Bennett

The Divorcee
Year: 1930
Genres: Romance, Drama
Cast: Norma Shearer, Chester Morris, Conrad Nagel, Robert Montgomery

King of Jazz
Year: 1930
Genres: Animation, Music
Cast: Paul Whiteman, John Boles, Laura La Plante, Jeanette Loff

Morocco
Year: 1930

8. Each smartphone provides a feature to take a screenshot. The captured screenshot is shown below. ==Have the screenshot of your frontend mobile app as 'sam_chung_frontend.jpg' by using your first and last name.== The captured screenshot is shown below.
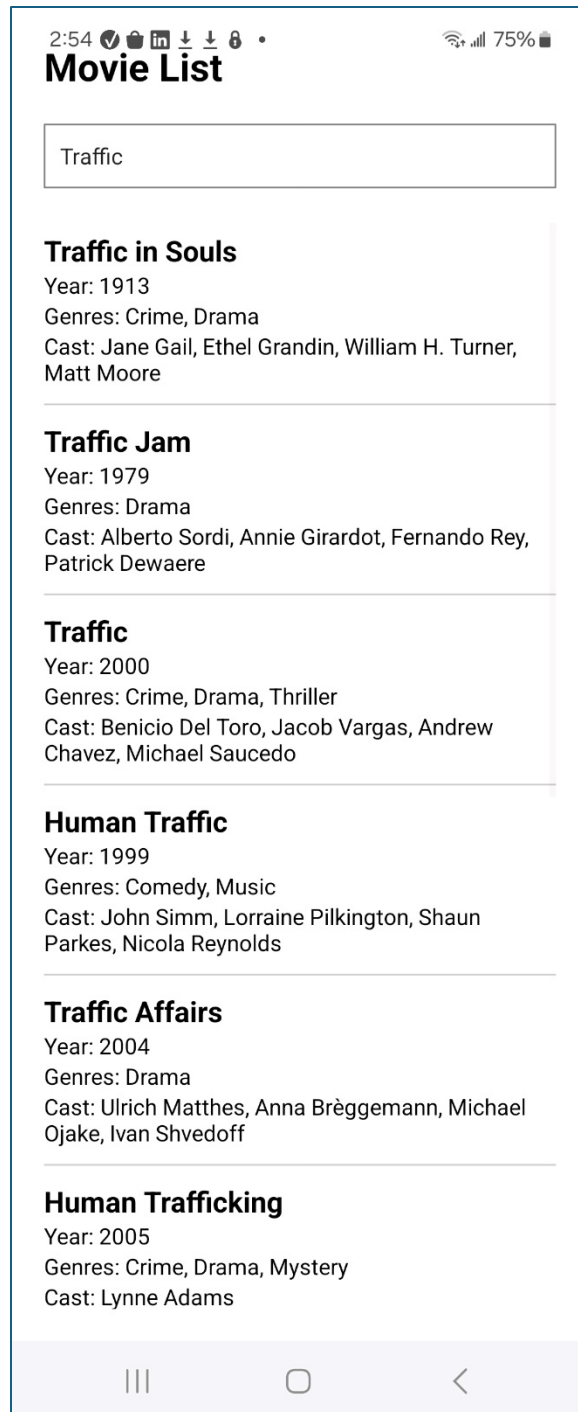
4) **Pushing your work to GitHub**
1) Go to Source Control on your GitHub codespaces and observe the pending changes.
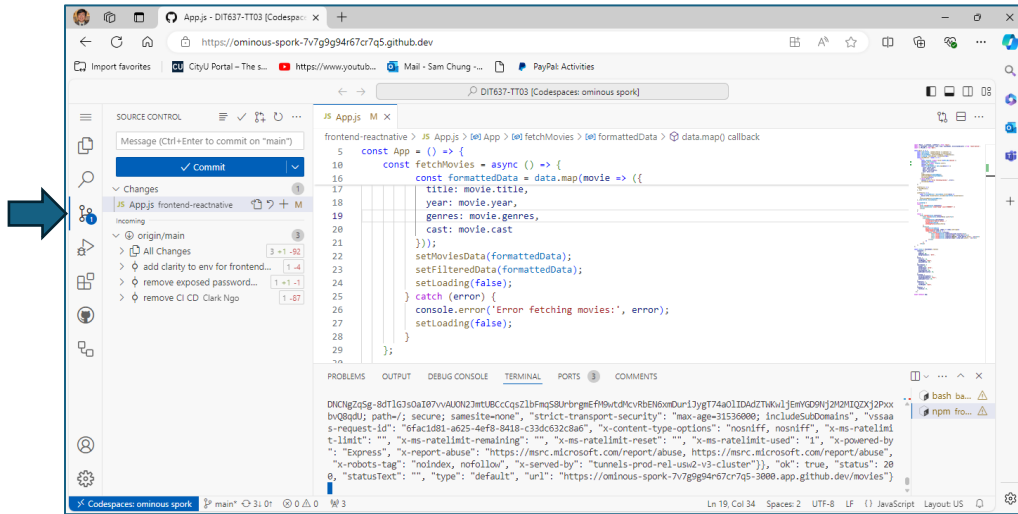
2) Type the Message for your changes in the Message box on the top. For example," **Submission for TT03 – Your Name"**

3) Click on the dropdown beside the commit button and select **Commit & Push** to update the changes to your repository main branch.

4) Select **Yes** when prompted.
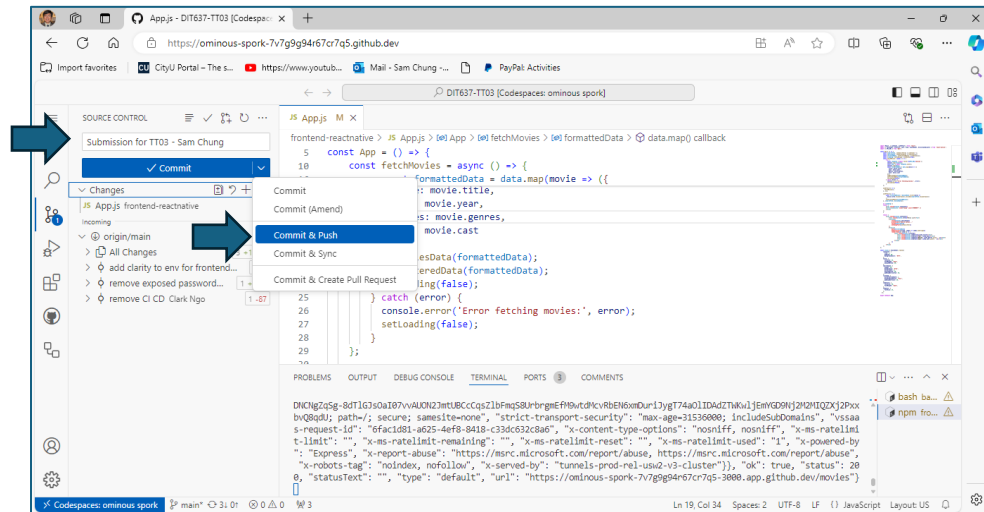
5) Screenshot Summary

In the provided screenshot, we see:

1. **Movie List Title**: Displayed using a Text component with a large, bold font style to indicate the screen's main purpose.
2. **Search Bar**: A 'TextInput' component at the top allows users to type in their search queries, dynamically filtering the movie list as they type.
3. **Movie Items**: Each movie is displayed within a 'View' styled as a list item, showing the title in bold, along with the genre and year in smaller text beneath it.

Sample Data for Movies: Provides a predefined list of movies with their titles, casts, genres, and release years, used to populate the initial movie list, and demonstrate search functionality.
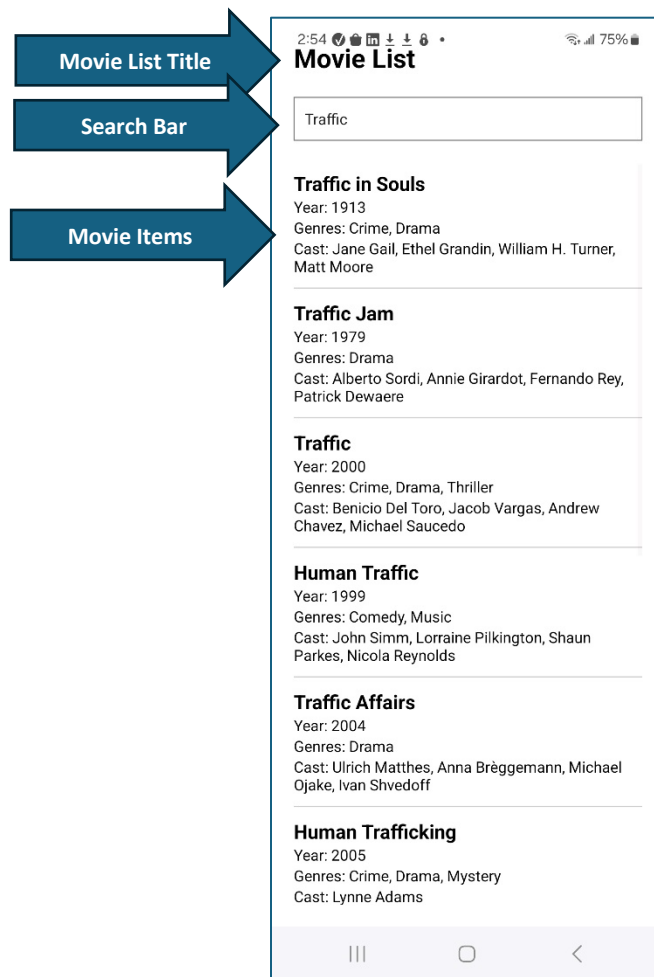
6) Breakdown of Frontend code

React Native code sets up a simple mobile application to search and browse a list of movies. It fetches movie data from an API, filters the data based on user input, and displays the movies in a list. The application includes a loading indicator for data fetching and uses hooks for managing state and side effects. The styles are defined using 'StyleSheet.create' to ensure a consistent look and feel.

Importing Modules

```
import React, { useState, useEffect } from 'react';
import { TextInput, FlatList, Text, View, StyleSheet, ActivityIndicator } from 'react-native';
import { API_URL } from '@env';
```

- **React**: A JavaScript library for building user interfaces.
- **useState**: A hook that lets you add React state to function components.
- **useEffect**: A hook that lets you perform side effects in function components (e.g., data fetching).
- **TextInput, FlatList, Text, View, StyleSheet, ActivityIndicator**: React Native components for building mobile UIs.
- **API_URL**: An environment variable for the API endpoint, imported using the @env module.

Component and State Initialization

```
const App = () => {
    const [searchTerm, setSearchTerm] = useState('');
    const [moviesData, setMoviesData] = useState([]);
    const [filteredData, setFilteredData] = useState([]);
    const [loading, setLoading] = useState(true);
```

- **App**: A functional component representing the main application.
- **searchTerm**: State to hold the search query input by the user.
- **moviesData**: State to store the raw movie data fetched from the API.
- **filteredData**: State to store the filtered list of movies based on the search term.
- **loading**: State to manage the loading state of the application.

Fetching Movies Data

```javascript
const fetchMovies = async () => {
    try {
        const response = await fetch(`${API_URL}/movies`);
        const data = await response.json();
        const formattedData = data.map(movie => ({
            title: movie.title,
            year: movie.year,
            genres: movie.genres,
            cast: movie.cast
        }));
        setMoviesData(formattedData);
        setFilteredData(formattedData);
        setLoading(false);
    } catch (error) {
        console.error('Error fetching movies:', error);
        setLoading(false);
    }
};
```

- **fetchMovies**: An asynchronous function that fetches movie data from the API.
- **response**: The response object from the fetch call.
- **data**: The JSON data parsed from the response.
- **formattedData**: An array of formatted movie objects to store only the required fields.
- **setMoviesData**: Updates the moviesData state with the fetched data.
- **setFilteredData**: Initializes the filteredData state with the fetched data.
- **setLoading(false)**: Sets the loading state to false after data is fetched.

Fetching Data on Component Mount

```javascript
useEffect(() => {
    fetchMovies();
}, []);
```

- **useEffect(() => { fetchMovies(); }, [])**: Calls the fetchMovies function once when the component mounts.

Filtering Data Based on Search Term

```
useEffect(() => {
    const filteredMovies = moviesData.filter(movie =>
        movie.title.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setFilteredData(filteredMovies);
}, [searchTerm, moviesData]);
```

- **useEffect(() => { ... }, [searchTerm, moviesData])**: Filters the movie data whenever searchTerm or moviesData changes.
- **filteredMovies**: An array of movies that match the search term.

Loading Indicator

```
if (loading) {
    return (
        <View style={styles.container}>
            <ActivityIndicator size="large" color="#0000ff" />
        </View>
    );
}
```

- **if (loading)**: Shows a loading spinner if the data is still being fetched.

Rendering the App

```
return (
    <View style={styles.container}>
        <Text style={styles.title}>Movie List</Text>
        <TextInput
            style={styles.searchInput}
            placeholder="Search movies..."
            value={searchTerm}
            onChangeText={setSearchTerm}
        />
        <FlatList
            data={filteredData}
            keyExtractor={(item, index) => index.toString()}
            renderItem={({ item }) => (
                <View style={styles.listItem}>
                    <Text style={styles.itemTitle}>{item.title}</Text>
                    <Text style={styles.itemText}>Year: {item.year}</Text>
                    <Text style={styles.itemText}>Genres: {item.genres.join(', ')}</Text>
                    <Text style={styles.itemText}>Cast: {item.cast.join(', ')}</Text>
                </View>
            )}
        />
    </View>
);
```

- **View**: Container component for laying out child components.
- **Text**: Component for displaying text.
- **TextInput**: Component for user input.
- **FlatList**: Component for rendering a list of items.
- **styles.container**: Style for the main container.
- **styles.title**: Style for the title text.
- **styles.searchInput**: Style for the search input.
- **styles.listItem**: Style for each list item.
- **styles.itemTitle**: Style for the title of each movie.
- **styles.itemText**: Style for the text displaying year, genres, and cast.

Styles

```
const styles = StyleSheet.create({
    container: {
        flex: 1,
        padding: 20,
        backgroundColor: '#fff',
    },
    title: {
        fontSize: 24,
        fontWeight: 'bold',
        marginBottom: 20,
    },
    searchInput: {
        height: 40,
        borderColor: 'gray',
        borderWidth: 1,
        marginBottom: 20,
        paddingHorizontal: 10,
    },
    listItem: {
        borderBottomWidth: 1,
        borderBottomColor: '#ccc',
        paddingVertical: 10,
    },
    itemTitle: {
        fontSize: 18,
        fontWeight: 'bold',
    },
    itemText: {
        fontSize: 14,
    },
});
```

- **StyleSheet.create**: Method for creating styles in React Native.

Exporting the App Component

```
export default App;
```

- **export default App**: Exports the App component as the default export of the module.

7) Breakdown of Backend code

The backend code sets up an Express.js server with MongoDB integration using Mongoose. It includes environment variable management with dotenv, CORS handling, and a middleware that connects to MongoDB on demand. The server has routes to fetch movie data from MongoDB and responds with a simple message for the root route. The server is conditionally started based on the environment, and the app instance is exported for testing.

Importing Modules

```
import express from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import cors from 'cors';
```

- **express**: A web framework for Node.js, used to build web applications and APIs.
- **mongoose**: An ODM (Object Data Modeling) library for MongoDB and Node.js. It provides a schema-based solution to model application data.
- **dotenv**: A module that loads environment variables from a .env file into process.env. This is useful for keeping sensitive information, such as database connection strings, outside the source code.
- **cors**: A middleware that allows your server to handle requests from different origins, enabling Cross-Origin Resource Sharing (CORS).

Configuration

```
dotenv.config();
```

- **dotenv.config()**: Loads environment variables from a .env file into process.env.

Creating Express Application

```
const app = express();
const port = 3000;
```

- **app**: An instance of an Express application.
- **port**: The port number where the application will run.

Enabling Cross-Origin Resource Sharing (CORS)

```
app.use(cors());
```

- **app.use(cors())**: Enables CORS for all routes, allowing your application to accept requests from different origins.

MongoDB URI

```
const uri = process.env.MONGODB_URI;
if (!uri) {
    console.error('MongoDB URI is not defined');
    process.exit(1);
}
```

- **uri**: Retrieves the MongoDB URI from environment variables.
- **if (!uri)**: Checks if the URI is not defined and exits the application if it's missing.

MongoDB Connection Middleware

```javascript
let isConnected = false;

const connectToMongoDB = async (req, res, next) => {
  if (!isConnected) {
    try {
      await mongoose.connect(uri, {
        dbName: 'sample_mflix' // Set the database name
      });
      isConnected = true;
      console.log('Connected to MongoDB Atlas');
    } catch (error) {
      console.error('Error connecting to MongoDB Atlas:', error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }
  }
  next();
};
```

- **isConnected**: A flag to track the connection status to MongoDB.
- **connectToMongoDB**: Middleware that connects to MongoDB if not already connected. It sets the isConnected flag to true once the connection is established. If the connection fails, it responds with a 500 status code.

Movie Schema and Model

```javascript
const movieSchema = new mongoose.Schema({
  title: String,
  genres: [String],
  cast: [String],
  year: Number
}, { collection: 'movies' });


const Movie = mongoose.model('Movie', movieSchema);
```

- **movieSchema**: Defines the structure of the movie documents in MongoDB.
- **Movie**: A model based on the movieSchema, representing the movies collection in MongoDB.

Routes

```javascript
app.get('/movies', connectToMongoDB, async (req, res) => {
  try {
    const movies = await Movie.find();
    res.json(movies);
  } catch (error) {
    console.error('Error retrieving movies:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});


app.get('/', (req, res) => {
  res.send('Hello World!');
});
```

- **/movies**: A route that retrieves all movies from the database. It uses the connectToMongoDB middleware to ensure a connection to MongoDB before querying.
- **/**: A simple route that responds with 'Hello World!'.

Starting the Server

```
if (process.env.NODE_ENV !== 'test') {
  app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
  });
}
```

- **app.listen(port, ...)**: Starts the server on the specified port, but only if the environment is not set to 'test'. This allows for conditional server starting, useful for testing purposes.

Exporting the App

```
export default app;
```

- **export default app**: Exports the app instance for use in testing or other modules.

FAQs

What environment variables should I have for backend?
Sample:

```
MONGODB_URI=mongodb+srv://<username>:<password>@<cluster-url>/<database-name>
```

What environment variables should I have for frontend?
Sample:

```
## update this with your own backend-expressjs forwarded address
API_URL=https://probable-fishstick-77q6jrvvpxrfwr7r-3000.app.github.dev
```

How to run my frontend code?
Inside the frontend-reactnative folder
- npx expo login
- npm expo start --tunnel

How to run my backend code?
Inside the backend-expressjs folder
- npm start