Capstone Project

23CSP-339 - Full Stack-I

Capstone Recommendation Engine

A PROJECT REPORT

Submitted by

Yashasvi

in partial fulfilment for the award of the degree of

Bachelor of Technology in Computer Science

IN

Big Data Analytics

October 2025

BONAFIDE CERTIFICATE

Certified that this project report **"Capstone Recommendation Engine"** is the Bonafide work of **Yashasvi** carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

Submitted for the project viva-voice examination held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

Project Title

Capstone Recommendation Engine

Project Description

The Capstone Recommendation Engine is a full-stack web application that provides personalized movie recommendations to users. The application is built using React.js for the user interface, Node.js Express for the backend API, and MongoDB as the database. A Python FastAPI service implements the recommendation logic, dynamically loading movies based on real-time data in MongoDB.

Objectives

- Develop a functional recommendation engine supporting add/view features for users and movies.
- Integrate collaborative filtering for better user recommendations.
- Demonstrate scalable full-stack architecture (React, Express, FastAPI, MongoDB).

Key Features

- Users and movies can be added live from the frontend.
- Recommendations update automatically for all users as movies/interactions are added.
- Backend APIs for CRUD operations.
- Recommendation logic handled by Python FastAPI, results sent to frontend via Express.js.

Hardware/Software Requirements

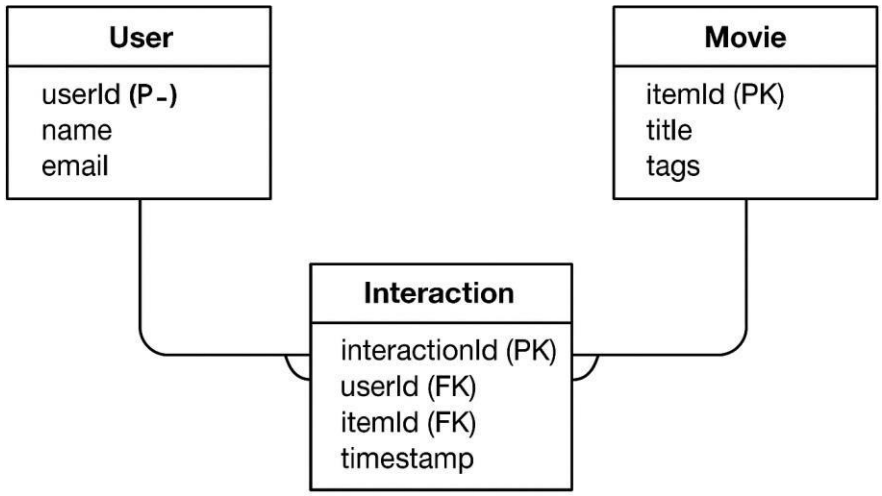
- **Hardware:** Intel i3+, 8GB RAM, 100GB HDD/SSD
- **Software:** Node.js v24.x, MongoDB v7.x, React.js, FastAPI (Python 3.14), VSCode or similar IDE, Google Chrome

ER Diagram

Entities:

- **User** – Stores user information
- **Movie** – Stores movie data with tags
- **Recommendation** – Stores movie scores

[User] ---< [Recommendation] >--- [Movie]



Database Schema

Movie Collection:

Field	Type	Description
ItemId	String	Unique movie identifier
title	String	Movie title
tags	Array	Tags associated with the movie

User Collection:

Field	Type	Description
userId	ObjectId	Unique user identifier

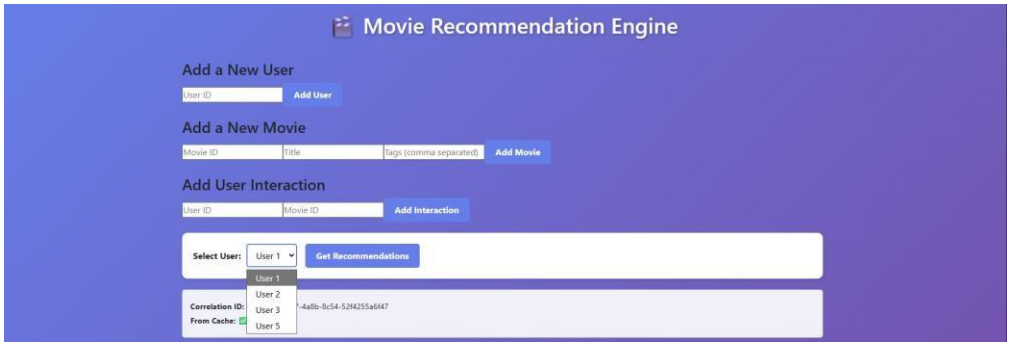
Recommendation Collection:

Field	Type	Description
userId	String	Linked user
ItemId	String	Linked movie

Front-End Screens

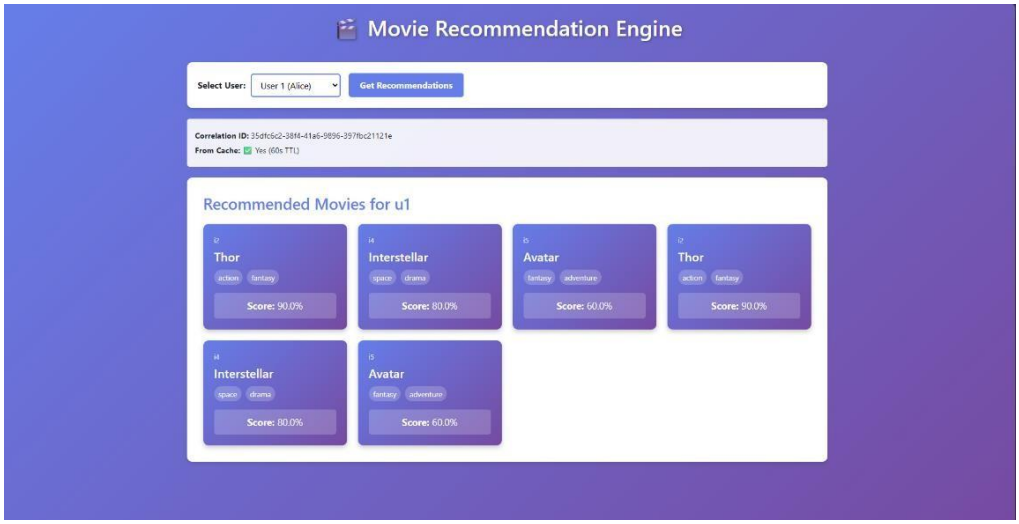
Home Screen:

- The Home Screen includes forms to add users and movies, and the dropdown to select users



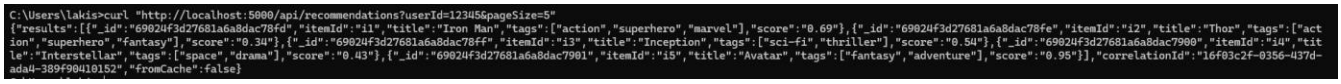
Recommendation Screen:

- The recommended movies display (carousel or card list).
- Movie titles, tags, and recommendation scores visible for each recommended item.

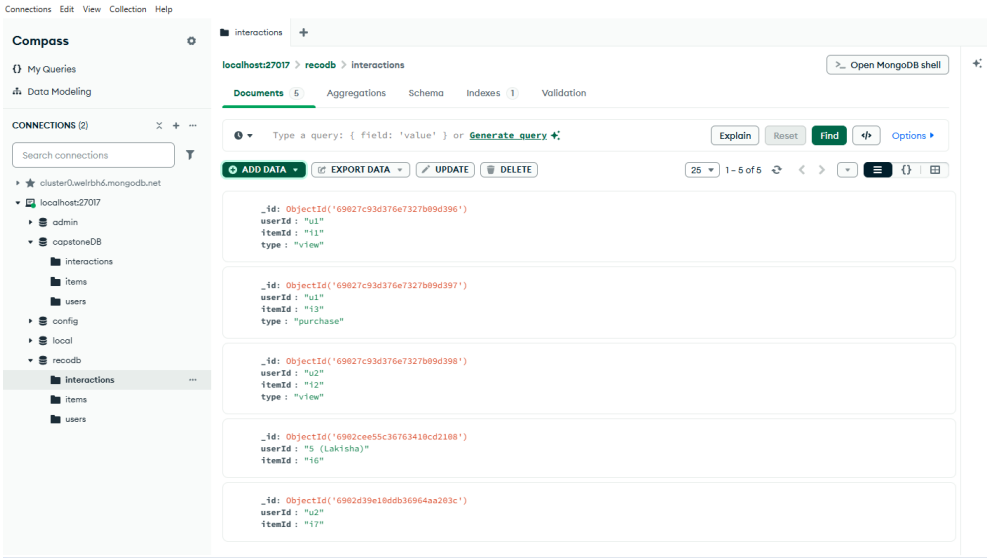


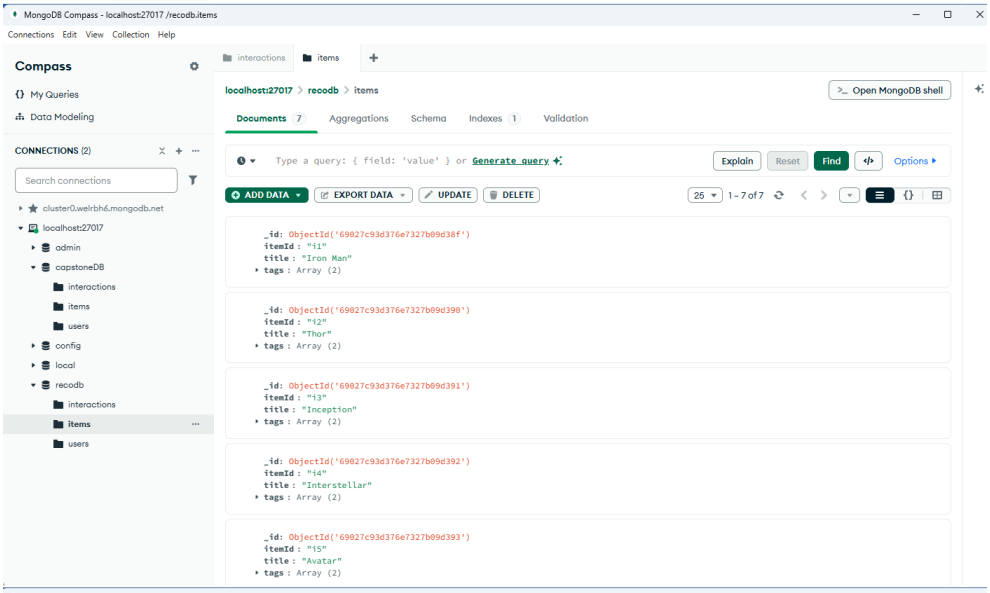
Output Screens and Reports

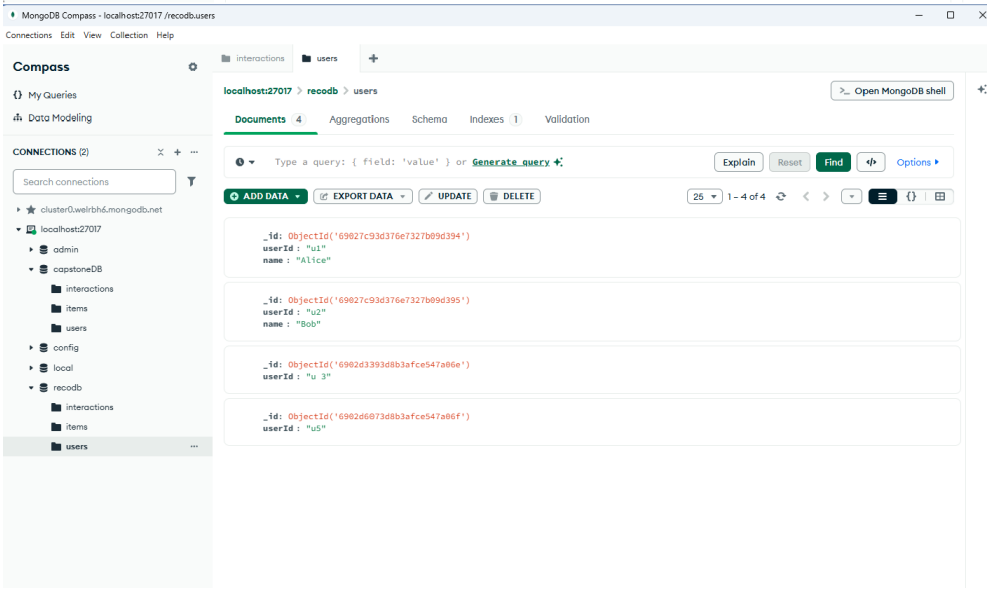
Backend calculates recommendation scores and returns **JSON responses**.



MongoDB stores the real time data for the recommendation engine







Limitation & Future Scope

Limitations

- Movie-only Recommendations:
  The current system is limited to recommending movies. Other types of media are not supported yet.

- Basic Scoring Logic:
  Recommendation scores are generated using simple rules or basic similarity; deep personalization based on advanced user behavior or machine learning is not yet implemented.

- No Authentication:
  The platform does not require users to log in or register securely. All user management is open and does not restrict access.

- No User Profiles:
  User-specific preferences and histories beyond basic interactions are not tracked or used for recommendation refinement.

- Limited User Interface:
  UI is functional but lacks advanced design, accessibility features, and mobile optimization.

Future Scope

- User Authentication & Profiles:
  Implement secure login and individual user account features to enable saving preferences and history.

- Machine Learning-Based Recommendations:
  Upgrade the backend to use machine learning models for smarter, context-aware recommendations based on more complex user data.

- Multi-Category Expansion:
  Extend the engine to recommend other content types such as books, music, or games for a richer user experience.

- Real-Time Analytics Dashboard:
  Add dashboards for administrators or users to see usage statistics, popular items, and system performance metrics.

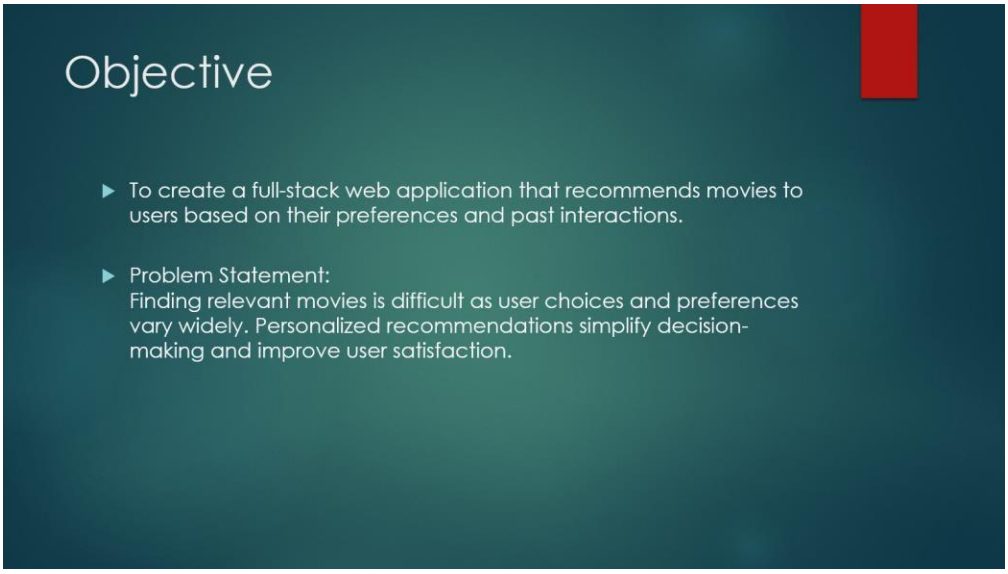
- User Ratings & Feedback:
  Enable users to rate movies and provide feedback to further personalize future recommendations.

- Improved UI/UX:
  Enhance visual appeal, usability, and responsiveness to deliver a professional-grade user experience.

GitHub URL

https://github.com/LakishaJaiswal/Recommendation-Engine/tree/main

Presentation Slides



System Architecture / Workflow

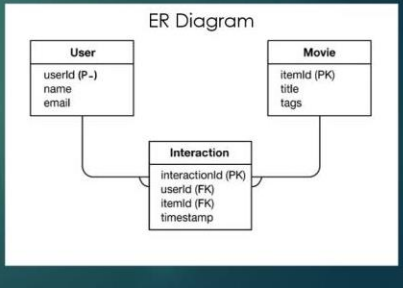
- Architecture Overview:
  - Frontend sends/receives user data, interacts with backend.
  - Backend processes requests, connects with MongoDB and Python model service.
  - Model Service dynamically generates recommendations using data from MongoDB.
- Workflow:
  - User interacts with frontend forms (add/view user/movie, get recommendations).
  - Backend APIs trigger data storage and retrieval.
  - Model service fetches movies, generates top suggestions for requested user.

Tech Stack

- Frontend: React.js for fast, interactive UI
- Backend: Node.js with Express for scalable REST APIs
- Database: MongoDB for flexible, real-time storage
- Model Microservice: Python FastAPI for dynamic recommendation logic
- Other Tools: Axios (API requests), NodeCache (caching), UUID (unique identifiers)

Database Design

- ER Diagram:
  - Three collections: Users, Items (Movies), Interactions
  - Users and Movies are connected via Interactions collection (mapping userId and itemId).
- Database Schema:
  - Users: userId
  - Items: itemId, title, tags
  - Interactions: userId, itemId

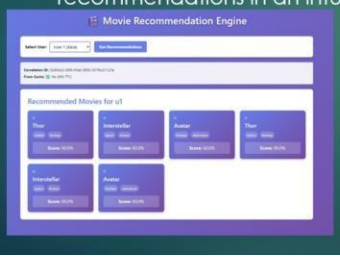


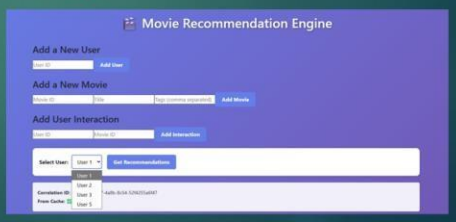
Key Features and Working

- Key Features:
- Add new users, movies, and interactions through easy-to-use forms.
- Recommendations update instantly with each database change.
- Clean, responsive UI allows real-time testing.
- Data persists in MongoDB and is always up-to-date.
- Working:
- Model Service (Python FastAPI) fetches all movies from MongoDB.
- Randomly selects/topK movies for recommendations (demo logic).
- Scores are assigned to each recommended movie.
- Results are sent as JSON to the backend and displayed in the frontend.

Front-End Screens

- Home: User selection, add movie/user/interaction forms.
- Recommendations: Display of recommended movies as cards/carousel.
- Description:
  Users can add themselves and movies, then immediately view personalized recommendations in an intuitive interface.

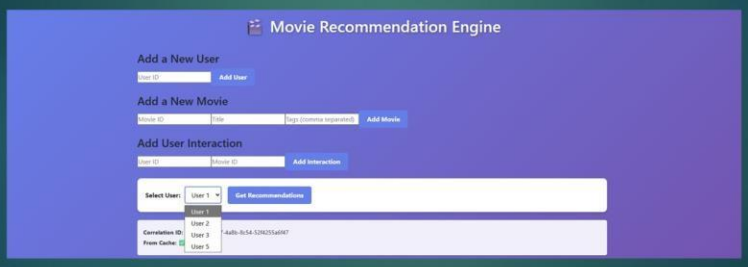


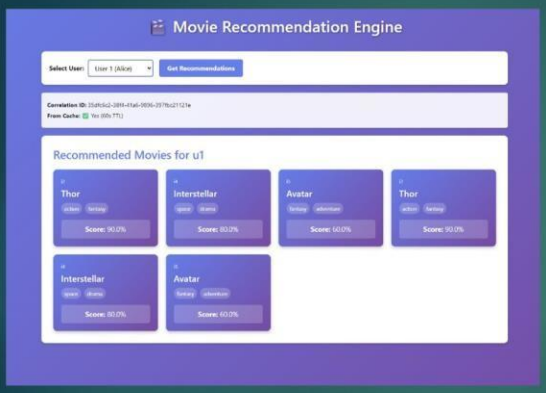


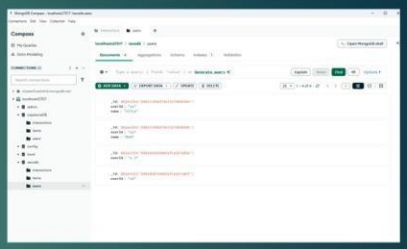
Limitations and Future Scope

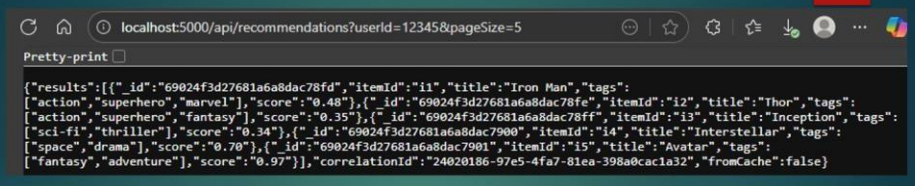
- Limitations
- Only supports movie recommendations (no other media types yet).
- Recommendation scores use basic, non-personalised logic.
- No password/authentication; open user management.
- UI and features are focused on core functionality.
- Future Scope
- Integrate user authentication and secure logins.
- Add machine learning models for smarter recommendations.
- Expand engine to support books, music, and other content.
- Add user ratings, reviews, and preference tracking.
- Develop advanced analytics dashboards for usage and results.

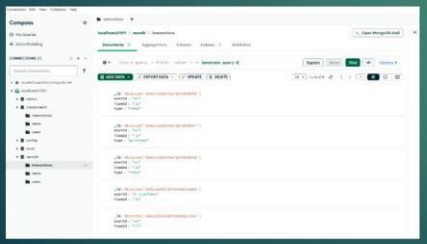
Outputs:

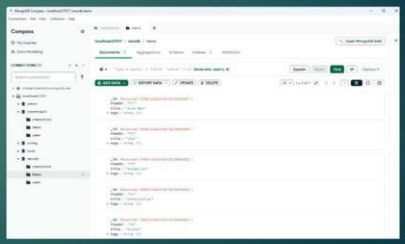

Movie Recommendation Engine
Add a New User
Add User
Add a New Movie
Add Movie
Add User Interaction
Add Interaction
Select User:
Get Recommendations
User 1
User 2
User 3
User 5


Movie Recommendation Engine
Select User:
Get Recommendations
Recommended Movies for u1
Thor
Interstellar
Avatar
Thor
Interstellar
Avatar




localhost:5000/api/recommendations?userId=12345&pageSize=5
Pretty-print
{"results":[{"_id":"69024f3d27681a6a8dac78fd","itemId":"i1","title":"Iron Man","tags":
["action","superhero","marvel"],"score":"0.48"},{"_id":"69024f3d27681a6a8dac78fe","itemId":"i2","title":"Thor","tags":
["action","superhero","fantasy"],"score":"0.35"},{"_id":"69024f3d27681a6a8dac78ff","itemId":"i3","title":"Inception","tags":
["sci-fi","thriller"],"score":"0.34"},{"_id":"69024f3d27681a6a8dac7900","itemId":"i4","title":"Interstellar","tags":
["space","drama"],"score":"0.70"},{"_id":"69024f3d27681a6a8dac7901","itemId":"i5","title":"Avatar","tags":
["fantasy","adventure"],"score":"0.97"}],"correlationId":"24020186-97e5-4fa7-81ea-398a0cac1a32","fromCache":false}





Project Code

Backend

- **server.js** (main Express server, MongoDB + APIs)

javascript

```
const express = require('express');
const cors = require('cors');
const axios = require('axios');
const { MongoClient } = require('mongodb');
const NodeCache = require('node-cache');
const { v4: uuidv4 } = require('uuid');
```

```
const app = express();
const cache = new NodeCache({ stdTTL: 60 });

app.use(cors({
   origin: ['http://localhost:3000'],
   credentials: true
}));
app.use(express.json());

const MONGO_URL = 'mongodb://localhost:27017';
const DB_NAME = 'recodb';
const MODEL_URL = 'http://localhost:8000';

app.get('/health', (req, res) => {
   res.json({ status: 'Backend running on port 5000' });
});

// Add User
app.post('/api/adduser', async (req, res) => {
  const { userId } = req.body;
  try {
    const client = new MongoClient(MONGO_URL);
    await client.connect();
    const db = client.db(DB_NAME);
    await db.collection('users').updateOne({ userId }, { $set: { userId } }, { upsert:
true });
    await client.close();
    res.json({ status: 'User added!', userId });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Get All Users
app.get('/api/users', async (req, res) => {
  try {
    const client = new MongoClient(MONGO_URL);
    await client.connect();
```

```
    const db = client.db(DB_NAME);
    const users = await db.collection('users').find({}).toArray();
    await client.close();
    res.json(users.map(u => u.userId));
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Add Movie
app.post('/api/addmovie', async (req, res) => {
    const { itemId, title, tags } = req.body;
    try {
        const client = new MongoClient(MONGO_URL);
        await client.connect();
        const db = client.db(DB_NAME);
        await db.collection('items').insertOne({ itemId, title, tags });
        await client.close();
        res.json({ status: 'Movie added!', itemId });
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
 });

// Add Interaction
app.post('/api/interact', async (req, res) => {
    const { userId, itemId } = req.body;
    try {
        const client = new MongoClient(MONGO_URL);
        await client.connect();
        const db = client.db(DB_NAME);
        await db.collection('interactions').insertOne({ userId, itemId });
        await client.close();
        res.json({ status: 'Interaction saved!', userId, itemId });
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
```

```
});

// Get Recommendations
app.get('/api/recommendations', async (req, res) => {
  const userId = req.query.userId;
  const cacheKey = `reco_${userId}`;
  const correlationId = uuidv4();

  const cached = cache.get(cacheKey);
  if (cached) {
    return res.json({
      results: cached,
      fromCache: true,
      correlationId: correlationId
    });
  }

  try {
    const modelRes = await axios.get(`${MODEL_URL}/recommend`, {
      params: { userId, topK: 5 }
    });

    const itemIds = modelRes.data.recommendations.map(r => r.itemId);
    const client = new MongoClient(MONGO_URL);
    await client.connect();
    const db = client.db(DB_NAME);
    const results = await db.collection('items').find({ itemId: { $in: itemIds }
}).toArray();
    await client.close();

    const movieMap = {};
    results.forEach(movie => { movieMap[movie.itemId] = movie; });

    const uniqueResults = [];
    const seenIds = new Set();
    for (const rec of modelRes.data.recommendations) {
      if (!seenIds.has(rec.itemId) && movieMap[rec.itemId]) {
```

```
            uniqueResults.push({ ...movieMap[rec.itemId], score:
parseFloat(rec.score) });
            seenIds.add(rec.itemId);
          }
        }

        cache.set(cacheKey, uniqueResults, 60);
        res.json({
          results: uniqueResults,
          fromCache: false,
          correlationId: correlationId
        });
    } catch (err) {
        res.status(500).json({ error: 'Recommendation error', details: err.message });
    }
});

app.listen(5000, () => {
    console.log('✓ Backend running on port 5000');
});
```

- **db/seed.js** (for demo data)

```
const { MongoClient } = require('mongodb');

async function seed() {
    const client = new MongoClient('mongodb://localhost:27017');
    await client.connect();
    const db = client.db('recodb');

    // Clear existing
    await db.collection('items').deleteMany({});
    await db.collection('users').deleteMany({});
```

```
  await db.collection('interactions').deleteMany({});

  // Insert movies
  await db.collection('items').insertMany([
    { itemId: 'i1', title: 'Iron Man', tags: ['action', 'superhero'] },
    { itemId: 'i2', title: 'Thor', tags: ['action', 'fantasy'] },
    { itemId: 'i3', title: 'Inception', tags: ['sci-fi', 'thriller'] },
    { itemId: 'i4', title: 'Interstellar', tags: ['space', 'drama'] },
    { itemId: 'i5', title: 'Avatar', tags: ['fantasy', 'adventure'] }
  ]);
  // Insert users
  await db.collection('users').insertMany([
    { userId: 'u1', name: 'Alice' },
    { userId: 'u2', name: 'Bob' }
  ]);
  // Insert interactions
  await db.collection('interactions').insertMany([
    { userId: 'u1', itemId: 'i1', type: 'view' },
    { userId: 'u1', itemId: 'i3', type: 'purchase' },
    { userId: 'u2', itemId: 'i2', type: 'view' }
  ]);
  console.log('✓ Database seeded successfully');
  await client.close();}

seed().catch(console.error);
```

Modelservice

app.py (Python FastAPI microservice, dynamic recommendations)

```
from fastapi import FastAPI, Query
from fastapi.middleware.cors import CORSMiddleware
from pymongo import MongoClient
import random

app = FastAPI()
app.add_middleware(
```

```
    CORSMiddleware, allow_origins=["*"], allow_credentials=True,
allow_methods=["*"], allow_headers=["*"])

MONGO_URL = "mongodb://localhost:27017"
DB_NAME = "recodb"

@app.get("/health")
def health():
    return {"status": "Model service running"}

@app.get("/recommend")
def recommend(userId: str = Query(...), topK: int = Query(5)):
    client = MongoClient(MONGO_URL)
    db = client[DB_NAME]
    all_movies = list(db.items.find({}))
    client.close()

    if len(all_movies) == 0:
        return {"recommendations": []}

    sampled = random.sample(all_movies, min(topK, len(all_movies)))
    recs = [
        {"itemId": movie["itemId"], "score": round(random.uniform(0.7, 1.0), 2),
"title": movie["title"]}
        for movie in sampled
    ]
    return {"recommendations": recs}
```

Frontend

- **src/App.js** (React main file, dynamic UI)

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [userId, setUserId] = useState('');
  const [userList, setUserList] = useState([]);
```

```
const [recommendations, setRecommendations] = useState([]);
const [loading, setLoading] = useState(false);
const [fromCache, setFromCache] = useState(false);
const [correlationId, setCorrelationId] = useState('');

const [newMovie, setNewMovie] = useState({ itemId: '', title: '', tags: '' });
const [newUser, setNewUser] = useState({ userId: '' });
const [newInteraction, setNewInteraction] = useState({ userId: '', itemId: '' });

// Fetch users from users collection
const fetchUsers = async () => {
try {
    const res = await axios.get('http://localhost:5000/api/users');
    setUserList(res.data);
    if (res.data.length > 0 && userId === '') {
      setUserId(res.data[0]);
    }
  } catch {
    setUserList([]);
  }
};

// Fetch recommendations
const fetchRecommendations = async () => {
  if (!userId) return;
  setLoading(true);
  try {
    const response = await axios.get('http://localhost:5000/api/recommendations', {
      params: { userId }
    });
    setRecommendations(response.data.results);
    setFromCache(response.data.fromCache);
    setCorrelationId(response.data.correlationId);
  } catch (error) {
    console.error('Error fetching recommendations:', error);
    alert('Failed to fetch recommendations');
  }
```

```
  setLoading(false);
};

useEffect(() => {
  fetchUsers();
}, []);

useEffect(() => {
  fetchRecommendations();
  // eslint-disable-next-line
}, [userId]);

return (
  <div className="App">
    <header  className="App-header">
      <h1>🎬 Movie Recommendation Engine</h1>
    </header>

    <div className="container">

      {/* Add a New User */}
      <form
        style={{ marginBottom: 20, marginTop: 20 }}
        onSubmit={async (e) => {
          e.preventDefault();
          await axios.post('http://localhost:5000/api/adduser', {
            userId: newUser.userId,
          });
          alert('User added!');
          setNewUser({ userId: '' });
          fetchUsers();
        }}
      >
        <h3>Add a New User</h3>
        <input
          placeholder="User ID"
          value={newUser.userId}
```

```
    onChange={e => setNewUser({ userId: e.target.value })}
  />
  <button type="submit">Add User</button>
</form>

{/* Add a New Movie */}
<form
  style={{ marginBottom: 20 }}
  onSubmit={async (e) => {
  e.preventDefault();
    await axios.post('http://localhost:5000/api/addmovie', {
      itemId: newMovie.itemId,
      title: newMovie.title,
      tags: newMovie.tags.split(',').map(tag => tag.trim()),
    });
    alert('Movie added!');
    setNewMovie({ itemId: '', title: '', tags: '' });
  }}
>
  <h3>Add a New Movie</h3>
  <input
    placeholder="Movie ID"
    value={newMovie.itemId}
    onChange={e => setNewMovie({ ...newMovie, itemId: e.target.value })}
  />
  <input
    placeholder="Title"
    value={newMovie.title}
    onChange={e => setNewMovie({ ...newMovie, title: e.target.value })}
  />
  <input
    placeholder="Tags (comma separated)"
    value={newMovie.tags}
    onChange={e => setNewMovie({ ...newMovie, tags: e.target.value })}
  />
  <button type="submit">Add Movie</button>
</form>
```

```
      {/* Add User Interaction */}
      <form
        style={{ marginBottom: 20 }}
        onSubmit={async (e) => {
        e.preventDefault();
          await axios.post('http://localhost:5000/api/interact', {
            userId: newInteraction.userId,
            itemId: newInteraction.itemId,
          });
          alert('User interaction saved!');
          setNewInteraction({ userId: '', itemId: '' });
          fetchUsers(); // update available users
        }}
      >
        <h3>Add User Interaction</h3>
        <input
          placeholder="User ID"
          value={newInteraction.userId}
          onChange={e => setNewInteraction({ ...newInteraction, userId:
e.target.value })}
        />
        <input
          placeholder="Movie ID"
          value={newInteraction.itemId}
          onChange={e => setNewInteraction({ ...newInteraction, itemId:
e.target.value })}
        />
        <button type="submit">Add Interaction</button>
      </form>

      <div className="control-panel">
        <div className="user-selector">
          <label htmlFor="userId">Select User: </label>
          <select
            id="userId"
            value={userId}
            onChange={e => setUserId(e.target.value)}
```

```
          >
            {userList.map(uid => (
              <option key={uid} value={uid}>
                {uid.startsWith('u') ? `User ${uid.substring(1)}` : uid}
              </option>
            ))}
          </select>
        </div>
        <button onClick={fetchRecommendations} disabled={loading}>
          {loading ? 'Loading...' : 'Get Recommendations'}
        </button>
      </div>

      {correlationId && (
        <div className="info-box">
          <p><strong>Correlation ID:</strong> {correlationId}</p>
          <p><strong>From Cache:</strong> {fromCache ? '✅ Yes (60s TTL)' : '❌
Fresh from DB'}</p>
        </div>
      )}

      <div className="recommendations-container">
        <h2>Recommended Movies for {userId}</h2>
        {loading ? (
          <p>Loading recommendations...</p>
        ) : recommendations.length > 0 ? (
          <div className="carousel">
            {[...new Map(recommendations.map(m => [m.itemId,
m])).values()].map((movie) => (
              <div key={movie.itemId} className="movie-card">
                <div className="movie-id">{movie.itemId}</div>
                <h3>{movie.title}</h3>
                <div className="tags">
                  {movie.tags.map(tag => (
                    <span key={tag} className="tag">{tag}</span>
                  ))}
                </div>
                <div className="score">
```

```
                <strong>Score:</strong> {(parseFloat(movie.score) *
100).toFixed(1)}%
              </div>
            </div>
          ))}
        </div>
      ) : (
        <p>No recommendations found</p>
      )}
    </div>
  </div>
 );
}

export default App;
```

- **src/index.js**

```
import 'bootstrap/dist/css/bootstrap.min.css';
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
```

```
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

- **src/App.css**

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
}

.App {
  min-height: 100vh;
  padding: 20px;
}

.App-header {
  text-align: center;
  color: white;
  margin-bottom: 40px;
}

.App-header h1 {
  font-size: 2.5em;
  text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
}

.container {
```

```
  max-width: 1200px;
  margin: 0 auto;
}

.control-panel {
  background: white;
  padding: 20px;
  border-radius: 10px;
  margin-bottom: 20px;
  display: flex;
  gap: 15px;
  align-items: center;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}

.user-selector {
  display: flex;
  gap: 10px;
  align-items: center;
}

.user-selector label {
  font-weight: bold;
}

.user-selector select {
  padding: 8px 12px;
  border: 2px solid #667eea;
  border-radius: 5px;
  font-size: 1em;
}

button {
  padding: 10px 20px;
  background: #667eea;
  color: white;
  border: none;
```

```
  border-radius: 5px;
  cursor: pointer;
  font-size: 1em;
  font-weight: bold;
  transition: background 0.3s;
}

button:hover:not(:disabled) {
  background: #764ba2;
}

button:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}

.info-box {
  background: rgba(255,255,255,0.9);
  padding: 15px;
  border-radius: 5px;
  margin-bottom: 20px;
  font-size: 0.9em;
}

.info-box p {
  margin: 5px 0;
}

.recommendations-container {
  background: white;
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}

.recommendations-container h2 {
  color: #667eea;
```

```
  margin-bottom: 20px;
  font-size: 1.8em;
}

.carousel {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
}

.movie-card {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  padding: 20px;
  border-radius: 10px;
  transition: transform 0.3s, box-shadow 0.3s;
  box-shadow: 0 4px 8px rgba(0,0,0,0.2);
}

.movie-card:hover {
  transform: translateY(-10px);
  box-shadow: 0 8px 16px rgba(0,0,0,0.3);
}

.movie-id {
  font-size: 0.8em;
  opacity: 0.8;
  margin-bottom: 8px;
}

.movie-card h3 {
  font-size: 1.4em;
  margin-bottom: 10px;
}

.tags {
  display: flex;
```

```
  flex-wrap: wrap;
  gap: 8px;
  margin-bottom: 15px;
}

.tag {
  background: rgba(255,255,255,0.2);
  padding: 4px 8px;
  border-radius: 20px;
  font-size: 0.85em;
}

.score {
  background: rgba(255,255,255,0.2);
  padding: 10px;
  border-radius: 5px;
  text-align: center;
  font-size: 1.1em;
}
```

Folder Structure

```
recommendation_engine/
│
├── backend/
│   ├── server.js
│   ├── package.json
│   ├── package-lock.json
│   ├── db/
│   │   └── seed.js
│
├── modelservice/
```

```
│   ├── app.py
│   ├── requirements.txt
│
├──frontend/
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── App.js
│   │   ├── index.js
│   │   └── App.css
│   ├── package.json
│   └── package-lock.json
└── README.md
```

References

- MongoDB Documentation – https://www.mongodb.com/docs
- Node.js Documentation – https://nodejs.org/en/docs/
- React.js Documentation – https://react.dev/
- FastAPI Documentation - https://fastapi.tiangolo.com/

Bibliography

1. ByteXL Full Stack Development Capstone Project Guidelines
2. Various online tutorials on Node.js, MongoDB, and React.js