FULL STACK PROJECT REPORT

(2020-21)

ON

"MOVIE HUB WEBSITE"

Institute of Engineering and Technology

Submitted By:

Ashutosh Kumar (181500150)
Mayank Kumar (181500377)

Under the Supervision of

Mr. Pankaj Kapoor

Technical Trainer

(Department of Computer Engineering & Application)

ACKNOWLEDGEMENT

We thank the almighty for giving us the courage and perseverance in completing the project.

This project itself is acknowledgements for all those people who have gave us their heartfelt co-operation in making this project a grand success.

We extend our sincere thanks to ***Mr. Pankaj Kapoor*** Full Stack Trainer at GLA University, Mathura for providing valuable guidance at every stage of this project work. We are profoundly grateful towards the unmatched services rendered by him.

Last but not least, we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

ABSTRACT

Movie website is a beautiful fully integrated entertainment website. The website covers all the description or can say information regarding all the latest and past movies. The project is developed to provide an online platform for all those user who want to joy their leisure time by entertaining themselves by watching a movie that full-fills their requirements. The main module in this project are home page, description pages of all movies along with user registration who want to connect with us. Movie website gives you a online platform where you can decide and watch your favorites movies and view its description before watching it. In present era where all the contents are available on plenty amount that it might difficult for someone to find the exact thing which suits their requirement. Therefore, in order to make them free from entertaining and enjoying we made this website so that users can access and get all the movie related stuffs from this site without being moving anywhere else or accepting any premium membership for watching and entertaining them.

Contents

1. INTRODUCTION

1.1 Introduction

As we all know that in present day all things are available on the internet with just one click but instead of having all these, it is difficult to get all the contents and updated information as it is present in bulk amount and scatter throughout the internet which sometimes results in wasting most of the time of an individual. And the similar problems have been faced when we are looking for an entertainment like searching for a latest or the most popular movie or the oldest one that has thrill, romance, comedy, action, or all etc. So this project is used to provide one platform for all the movies with detailed information about it along with beautiful frontend and easy navigation for our users so that they can browse or search a movie that well fits its watching requirements.

1.2 Motivation and Objective

The main aim of building this project is to make a website where we can center all the movies of different categories to one place which includes all the latest as well as the oldest one so that our customer can view and compare different movies by visiting to each page of a particular movie which describes its popularity, released date and revenue generated and etc., which will help the users to choose the film which suits their entertainment requirements.

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 Software Required:

2.1.1 Web Browser

A web browser, or simply "browser," is an application used to access and view websites. Common web browsers include Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, and Apple Safari.

The primary function of a web browser is to render HTML, the code used to design or "mark-up" web pages. Each time a browser loads a web page, it processes the HTML, which may include text, links, and references to images and other items, such as cascading style sheets and JavaScript functions. The browser processes these items, then, renders them in the browser window. The primary function of a web browser is to render HTML, the code used to design or "mark-up" web pages. Each time a browser loads a web page, it processes the HTML, which may include text, links, and references to images and other items, such as cascading style sheets and JavaScript functions. The browser processes these items, then, renders them in the browser window.

2.1.2 Visual Code Studio

Visual Studio Code is a free source-code editor made
by for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

Visual Studio Code's source code comes from Microsoft's free and open- source software VSCode project released under the permissive Expat License, but the compiled binaries are freeware for any use. The main purpose of brackets is its live HTML, CSS and JavaScript editing functionality.

2.1.3 MongoDB

MongoDB’s document model is simple for developers to learn and use, while still providing all the capabilities needed to meet the most complex requirements at any scale.

2.1.4 Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

2.1.5 React

React JS is an open-source JavaScript library that is **used** for building user interfaces specifically for single-page applications. ... **React** allows developers to create large web applications that can change data, without reloading the page. The main purpose of **React** is to be fast, scalable, and simple.

2.1.6 Node JS

Node. js is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services, but was designed with real-time, push-based architectures in mind.

2.2 Modules and their Functionality

2.2.1 Home Page:

The home page is the default page which will be loaded first for each person visiting the website. It contains navigation bar on the top with brand name, one hyperlinks named as favorites and sign in button and signup button. Below the navbar, there is a banner and some of the top listing latest movies with on load more button which load the next latest movie list.

2.2.2 Movie Description Pages:

Here a user's can get all the required information about a movie that is movie released date, its popularity and revenue generated by it. This page will provide a detailed information regarding a particular movie and according to that user can compare with other movie description pages and decide which one is best to watch.

2.2.3 Sign-up Page:

This page is for all those user who want to purchase or know more about some projects and are the new users for our website.

2.2.4 Sign-in Page:

This page is designed for creating login in the website. It consist a form of some credentials like email and password etc.

3. WEBSITE DESIGNNING

In this section, all about the modules of websites. The navigation of user, how navigate from one page to another page.

3.1. Data Flow Diagram:

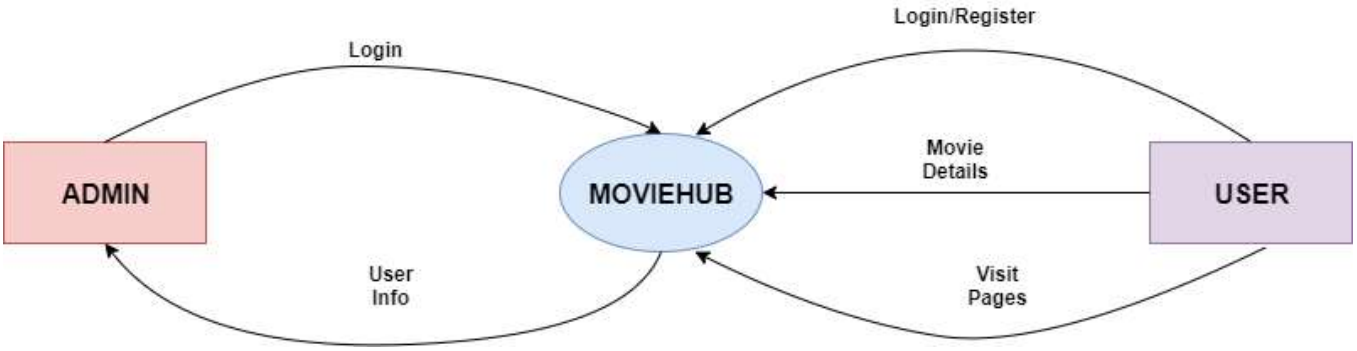


Fig 1: Fig 1: 0-Level DFD

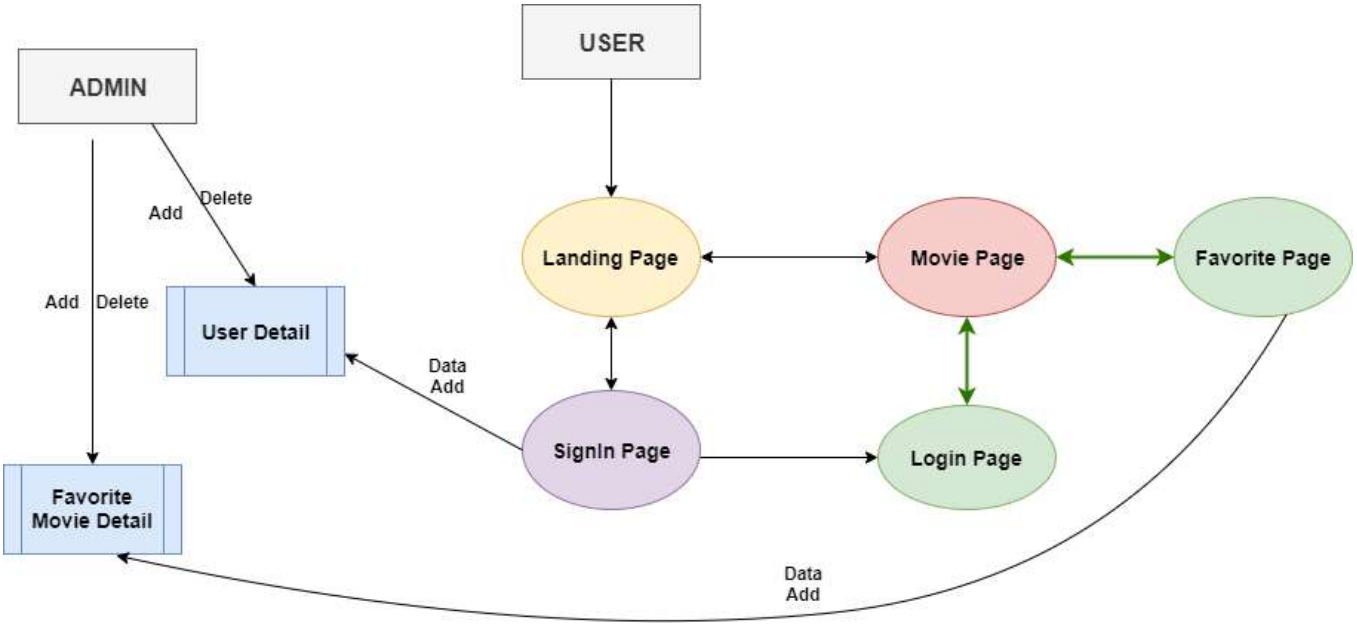


Fig 2: 1-Level DFD

User are those who visit the website. They navigate to any module to website.

3.2 UML(Use case diagram):

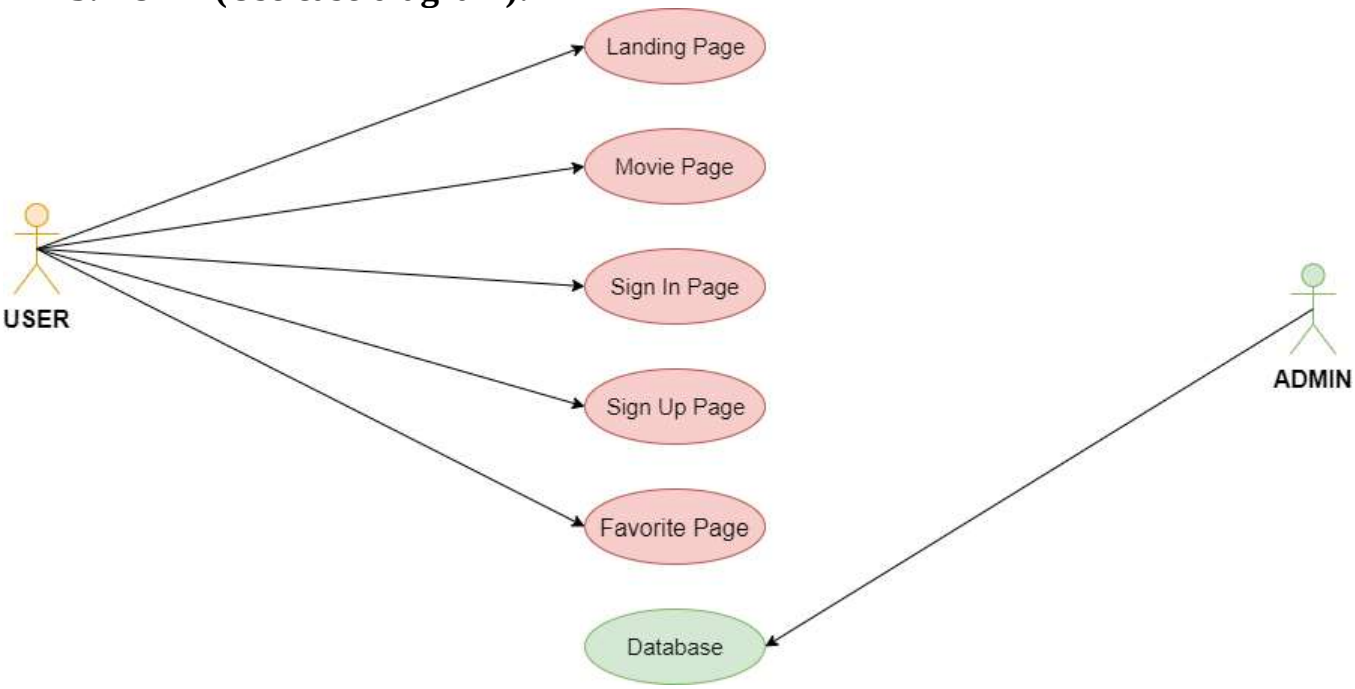


Fig 2: Use Case Diagram of Solar System

Use case diagram shows how user will navigate. This is also show the interaction between user and website. They will visit the different pages to reach the destination.

4.3. Database Design:

- ER Diagram:

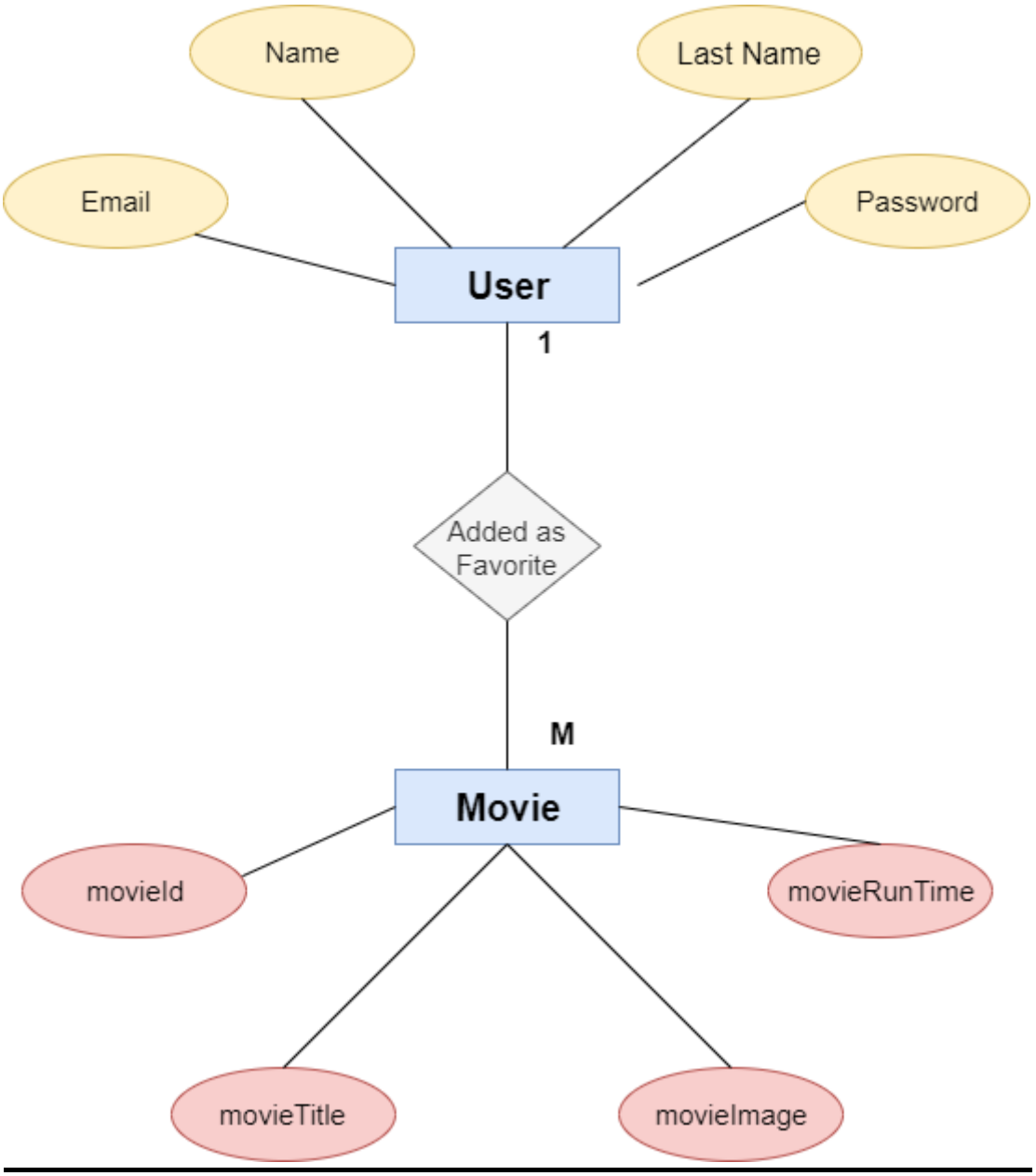


Fig 4: ER-Diagram

- Table for Fields and their Data types:
- User and Project Database Schema

Fields	Datatype
name	String
email	String
password	String
lastname	String
role	Number
image	String

movieImage	String
movieId	String
movieTitle	String
movieRunTime	String

- Code for storing the data:

Code For connectivity with Database:

```
module.exports = {

  mongoURI:
'mongodb+srv://ashutosh7889:ashutosh@movie.zdvk8.mongodb.net/myFirstDatabase?retryWrites=true&w=majority'

}
```

// User.js

```
const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const saltRounds = 10;

const jwt = require('jsonwebtoken');

const moment = require("moment");


const userSchema = mongoose.Schema({

  name: {

    type: String,

    maxlength: 50

  },

  email: {

    type: String,

    trim: true,

    unique: 1

  },
```

```
  password: {
    type: String,
    minglength: 5
  },
  lastname: {
    type: String,
    maxlength: 50
  },
  role: {
    type: Number,
    default: 0
  },
  image: String,
  token: {
    type: String,
  },
  tokenExp: {
    type: Number
  }
})


userSchema.pre('save', function(next) {
  var user = this;

  if (user.isModified('password')) {
    // console.log('password changed')
```

```
        bcrypt.genSalt(saltRounds, function(err, salt) {
            if (err) return next(err);

            bcrypt.hash(user.password, salt, function(err, hash) {
                if (err) return next(err);
                user.password = hash
                next()
            })
        })
    } else {
        next()
    }
});

userSchema.methods.comparePassword = function(plainPassword, cb) {
    bcrypt.compare(plainPassword, this.password, function(err, isMatch) {
        if (err) return cb(err);
        cb(null, isMatch)
    })
}

userSchema.methods.generateToken = function(cb) {
    var user = this;
    console.log('user', user)
    console.log('userSchema', userSchema)
    var token = jwt.sign(user._id.toHexString(), 'secret')
    var oneHour = moment().add(1, 'hour').valueOf();
```

```
    user.tokenExp = oneHour;
    user.token = token;
    user.save(function(err, user) {
        if (err) return cb(err)
        cb(null, user);
    })
}

userSchema.statics.findByToken = function(token, cb) {
    var user = this;

    jwt.verify(token, 'secret', function(err, decode) {
        user.findOne({ "_id": decode, "token": token }, function(err, user) {
            if (err) return cb(err);
            cb(null, user);
        })
    })
}

const User = mongoose.model('User', userSchema);

module.exports = { User }
```

//Favorite.js

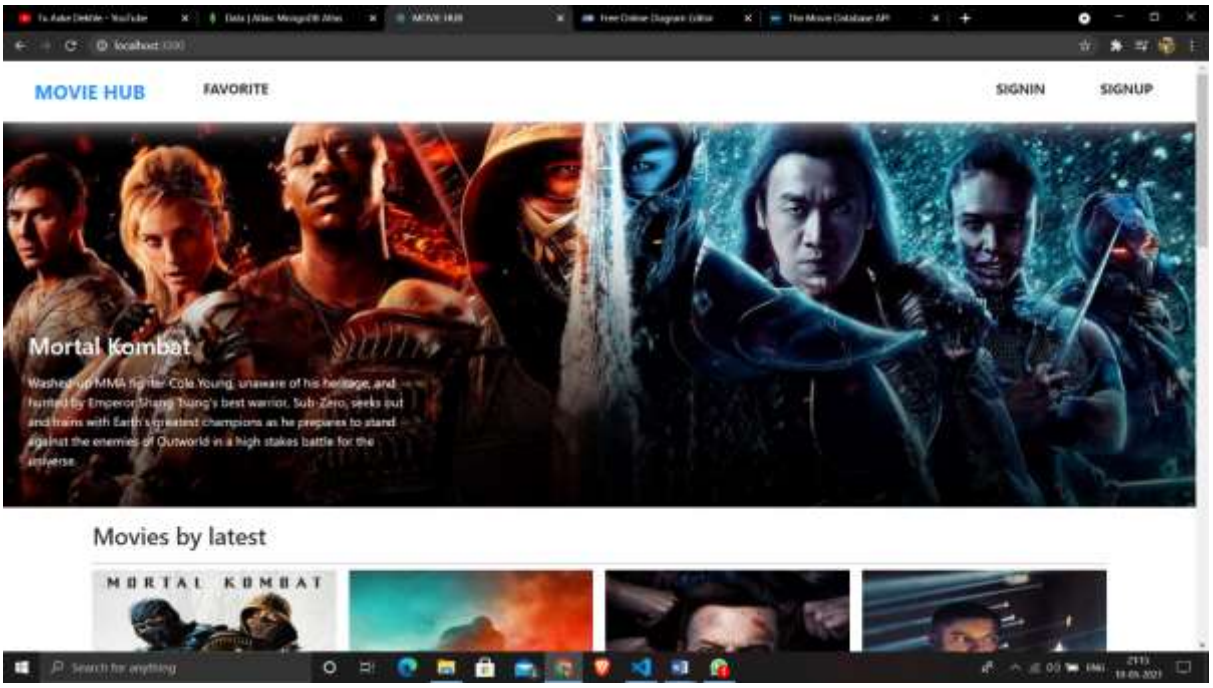
```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const favoriteSchema = mongoose.Schema({
```

```
    userFrom: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    },
    movieId: {
        type: String
    },
    movieTitle: {
        type: String
    },
    movieImage: {
        type: String
    },
    movieRunTime: {
        type: String
    }
})


const Favorite = mongoose.model('Favorite', favoriteSchema);

module.exports = { Favorite }
```

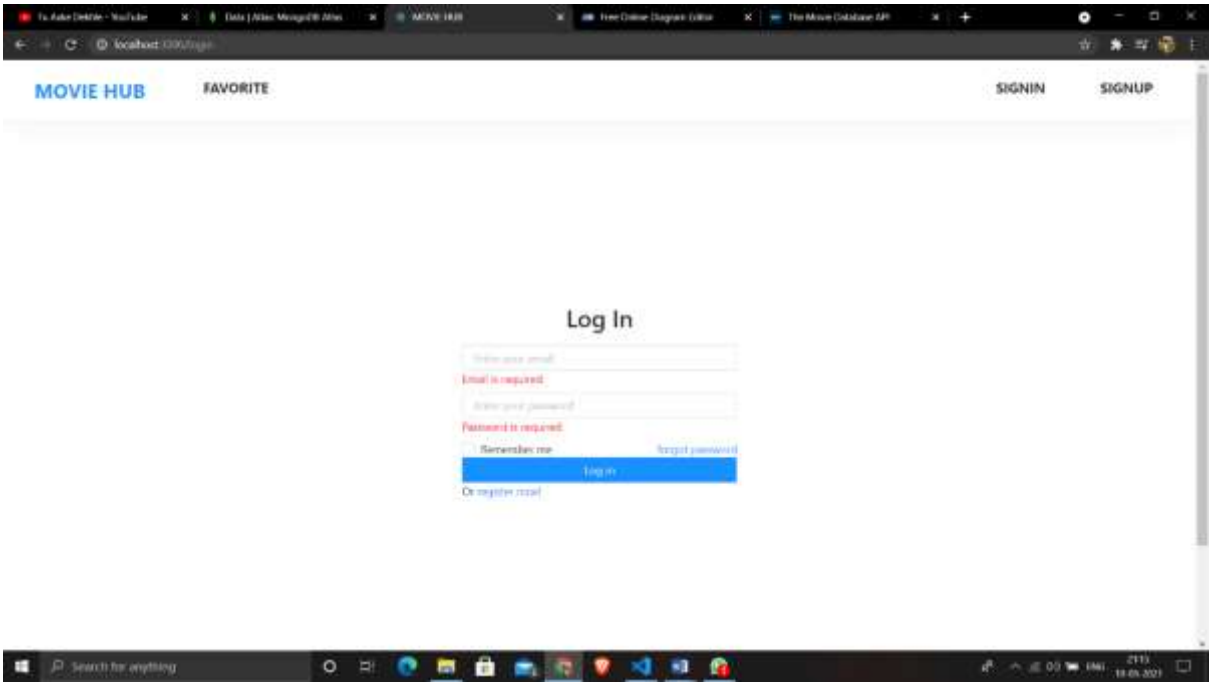
4. TESTING

4.1. Testing for Login:

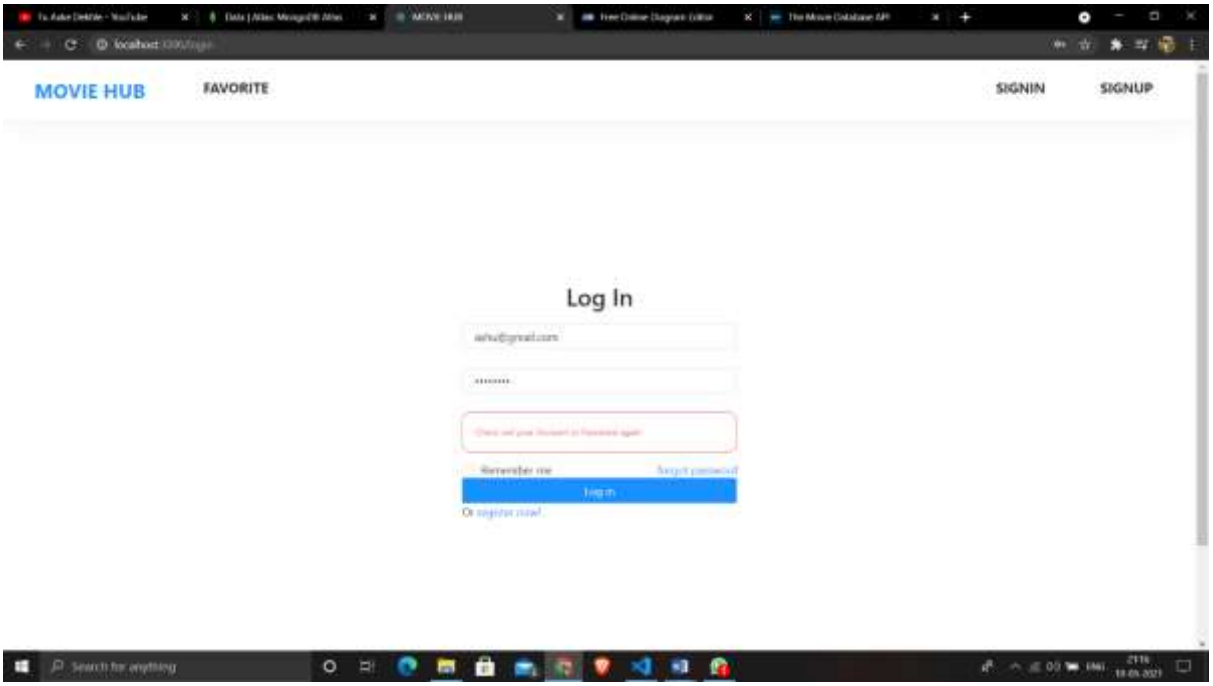
When user not logged in:



When user login without enter any credentials:



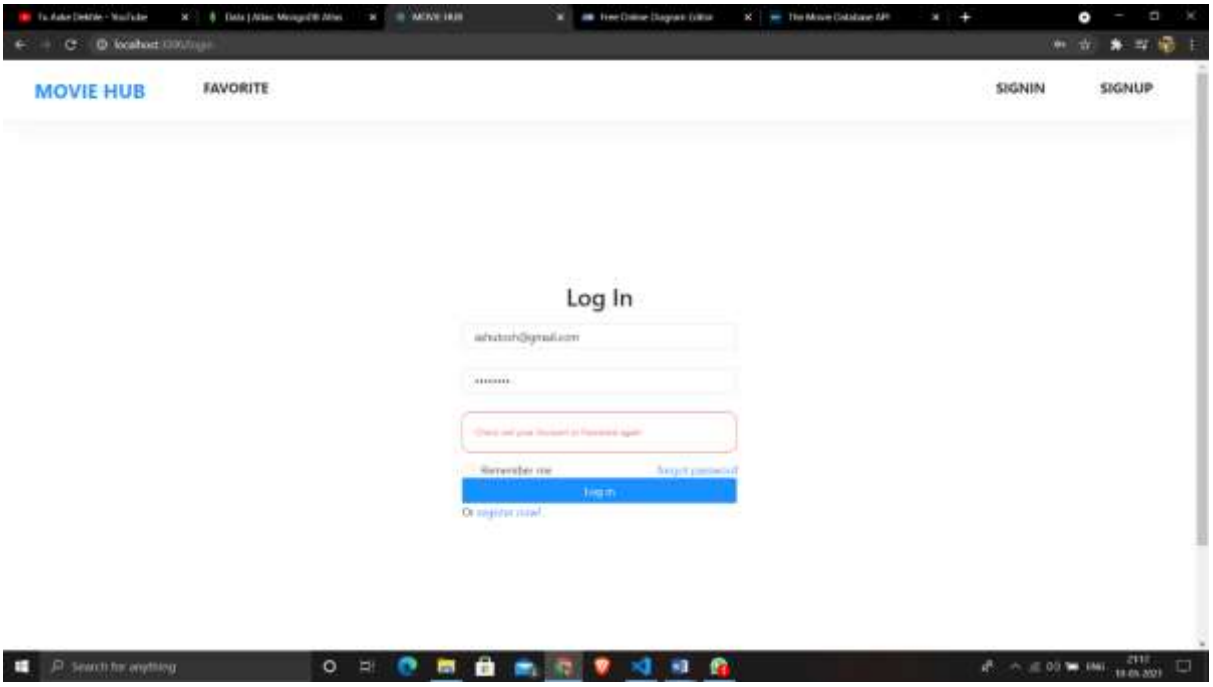
When user does not exists:



When user login with wrong password:

Email: ak@gmail.com

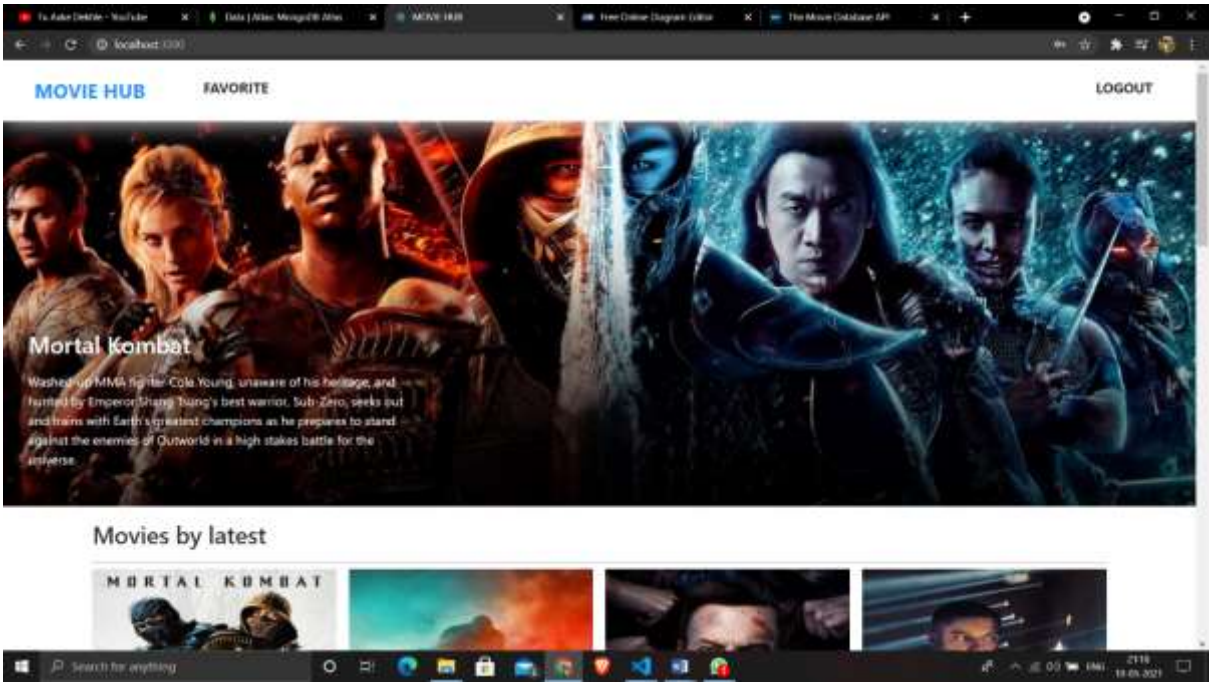
Password: ashutosh



When user login with correct credentials:

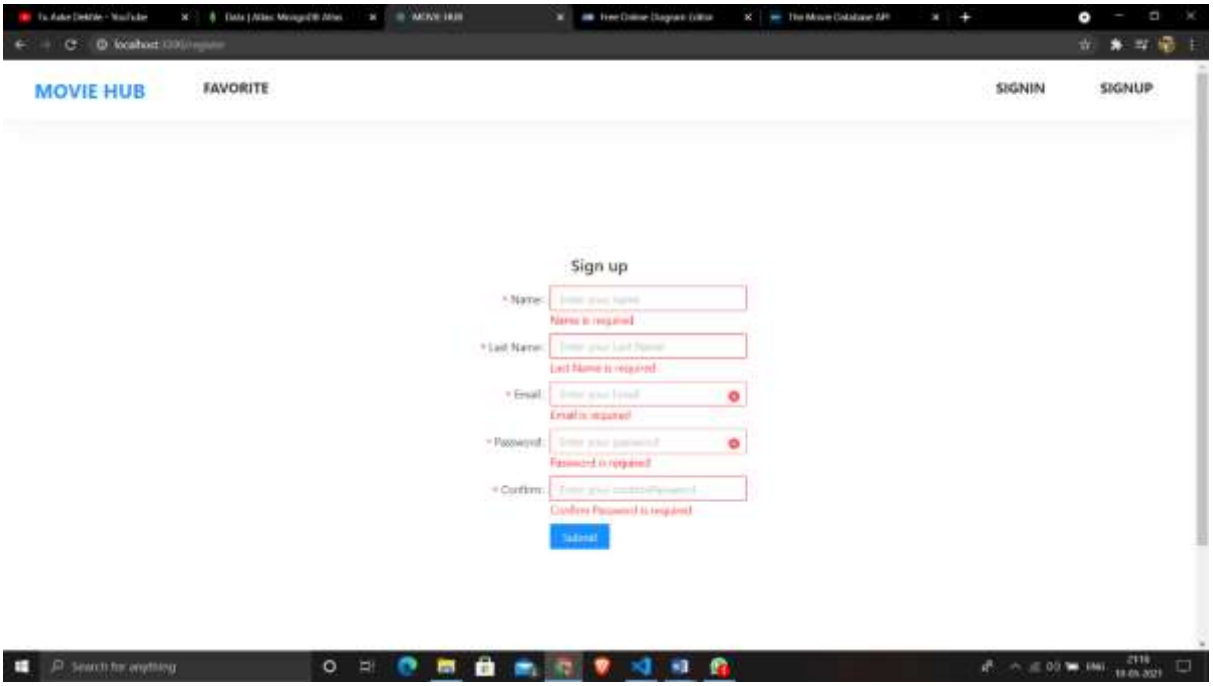
Email: ashutosh@gmail.com

Password: ashutosh



4.2. Testing for new user registration:

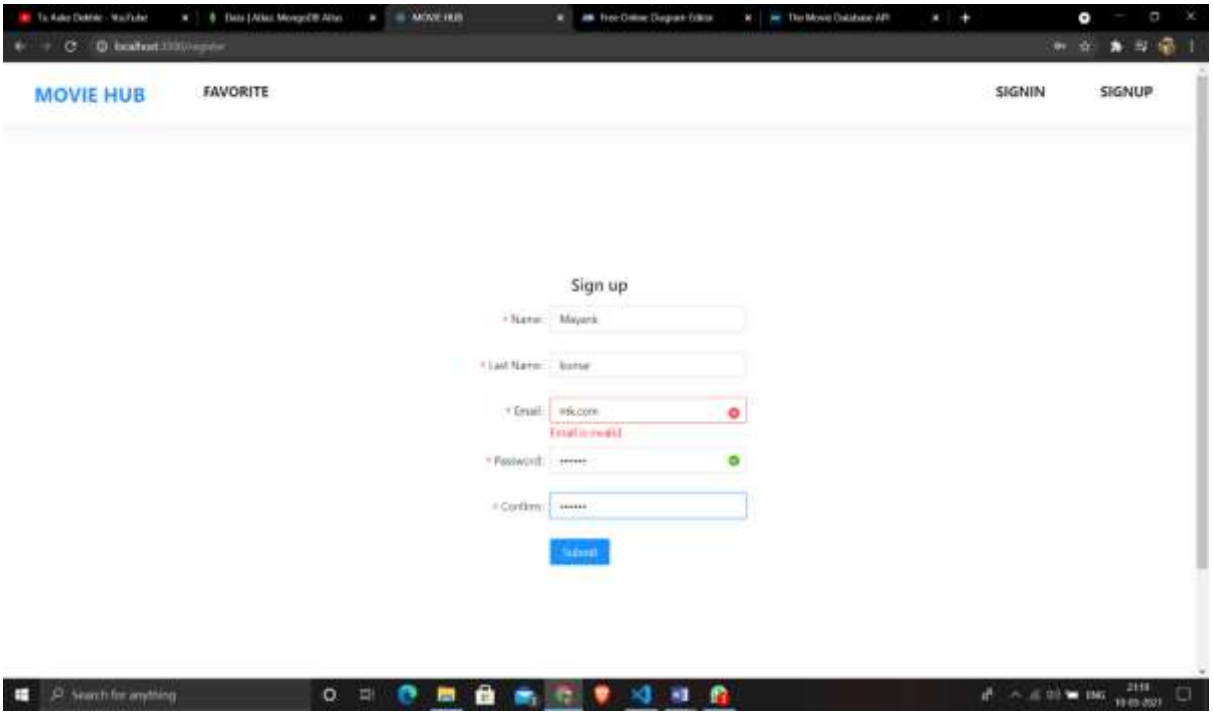
When user signup without enter any credential:



When user entered invalid email:

Name: Mayank, Last Name: kumar, Email: mk.com

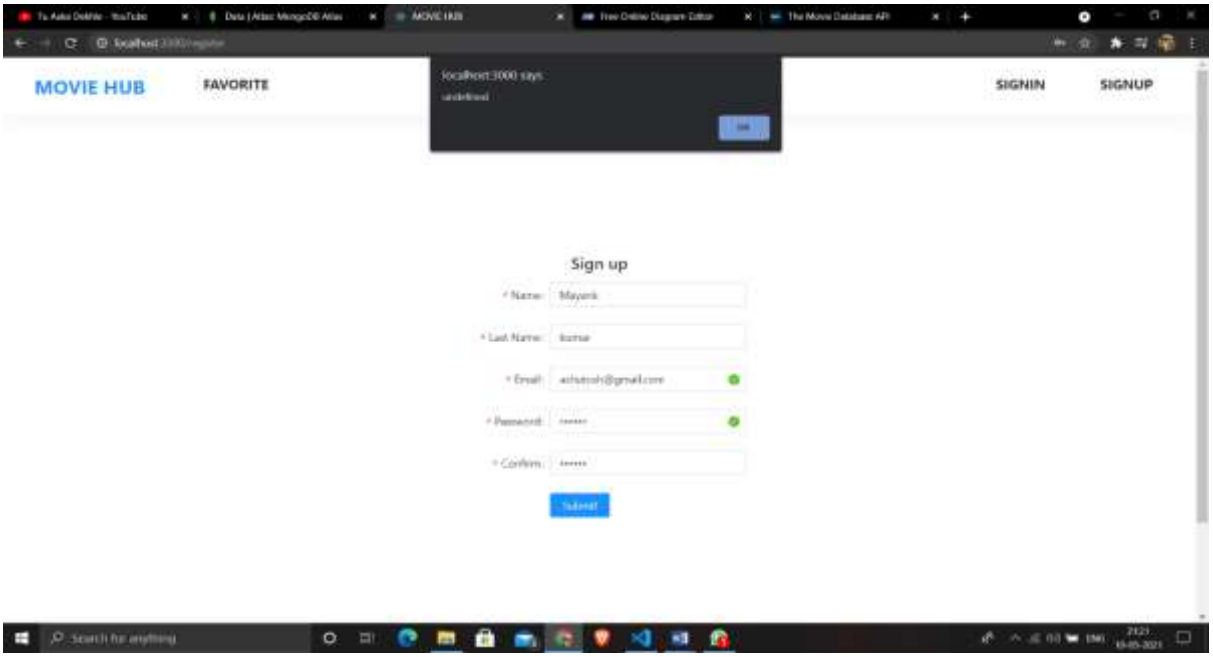
Password: 123456, Confirm: 123456



When user entered exists email:

Name: Mayank, Last Name: kumar, Email: ashutosh@gmail.com

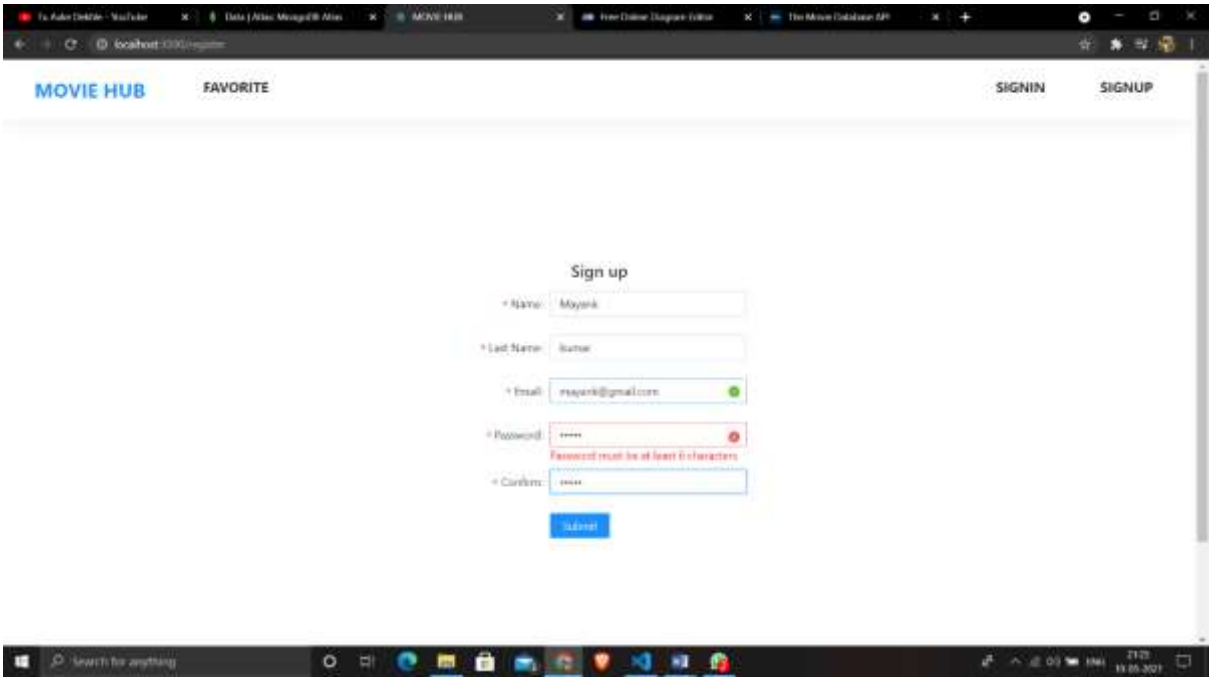
Password: 123456, Confirm: 123456



When user choose the password of length less than 6:

Name: Mayank, Last Name: kumar, Email: mayank@gmail.com

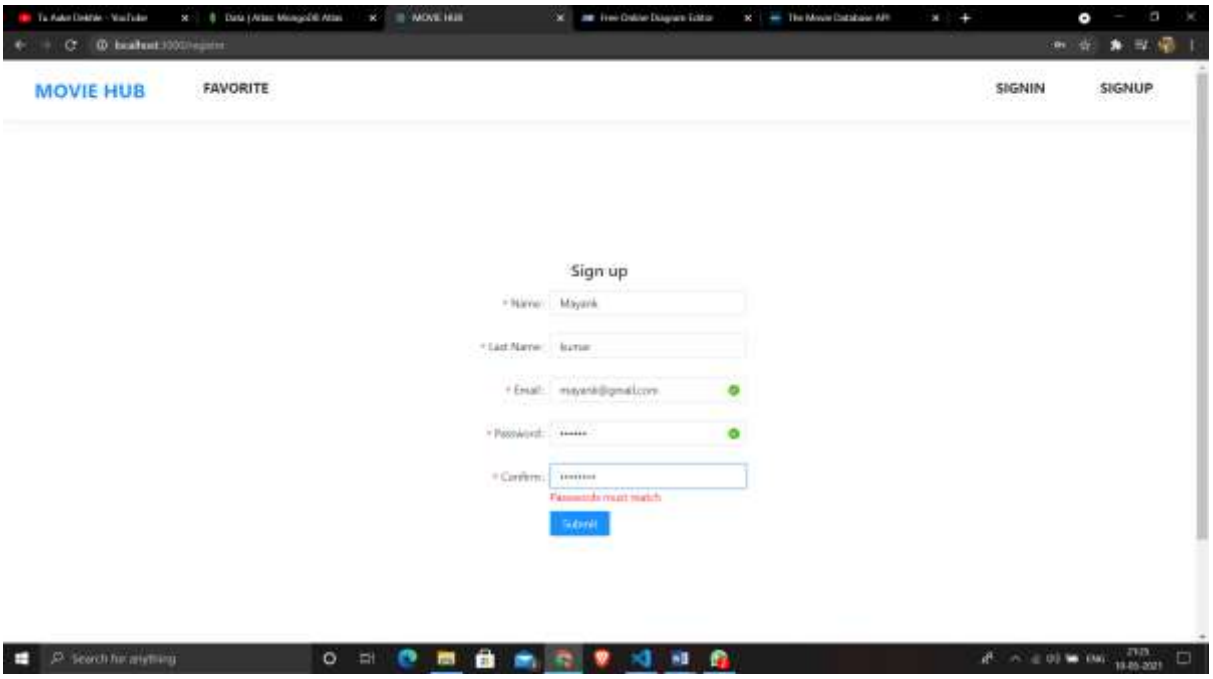
Password: 12345, Confirm: 12345



When password do not match:

Name: Mayank, Last Name: kumar, Email: mayank@gmail.com

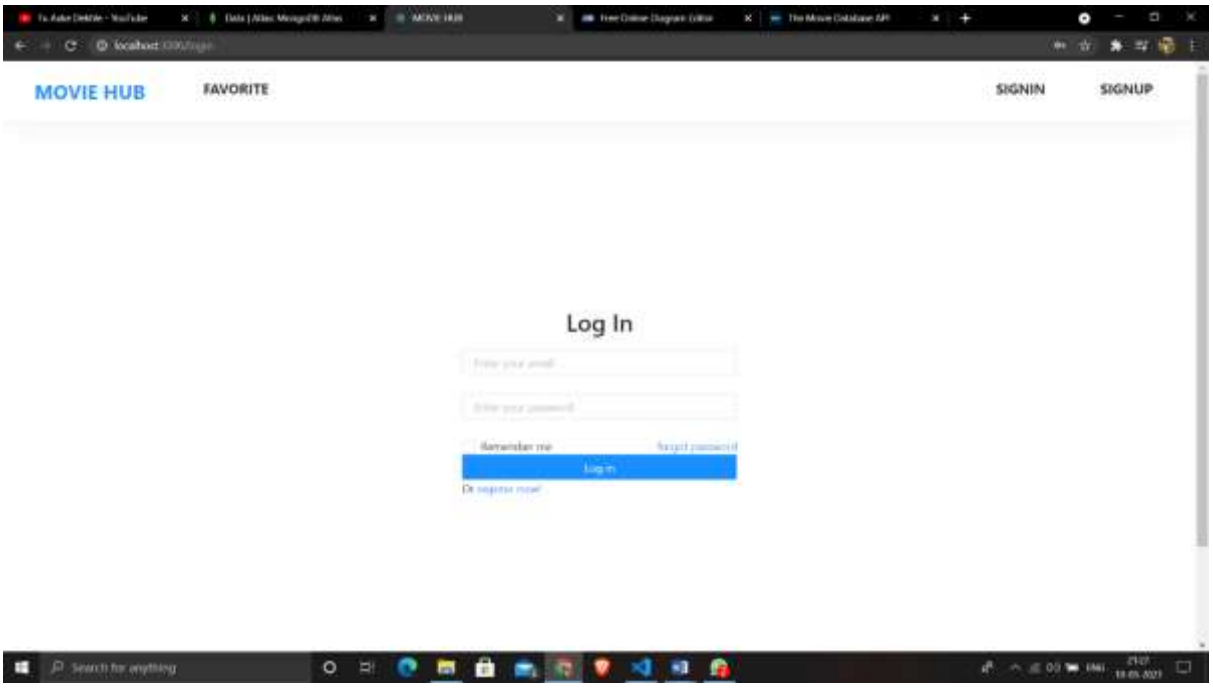
Password: mayank, Confirm: ashutosh



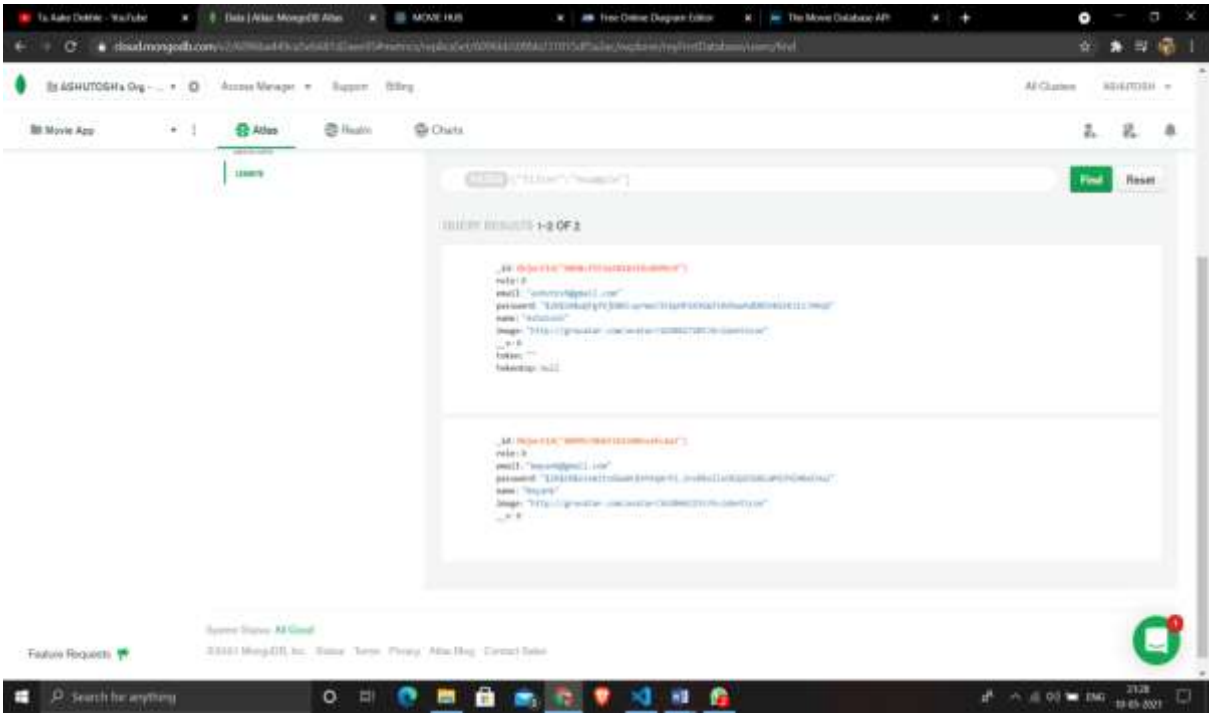
When user signin with enter all valid credentials:

Name: Mayank, Last Name: kumar, Email: mayank@gmail.com

Password: mayank, Confirm: mayank



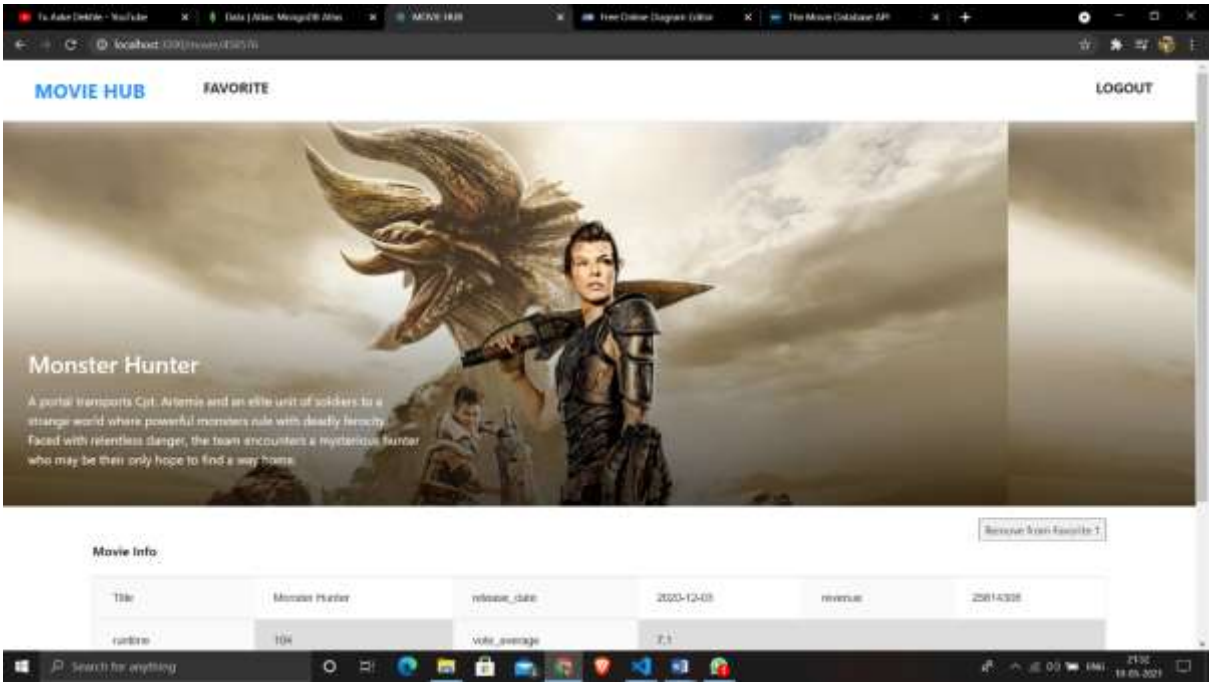
User data stored in database:



4.3. Testing for adding movie in favorite page:

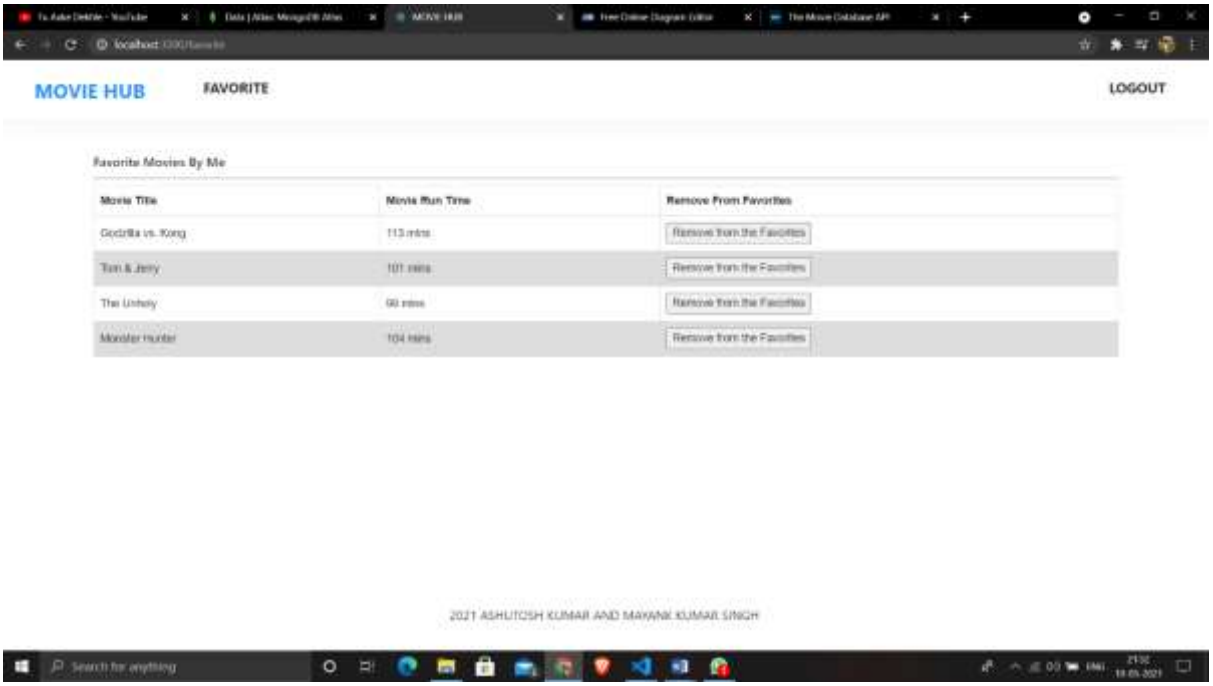
When user add to as favorite the add to favorite button:

Movie Name: Monster Hunter



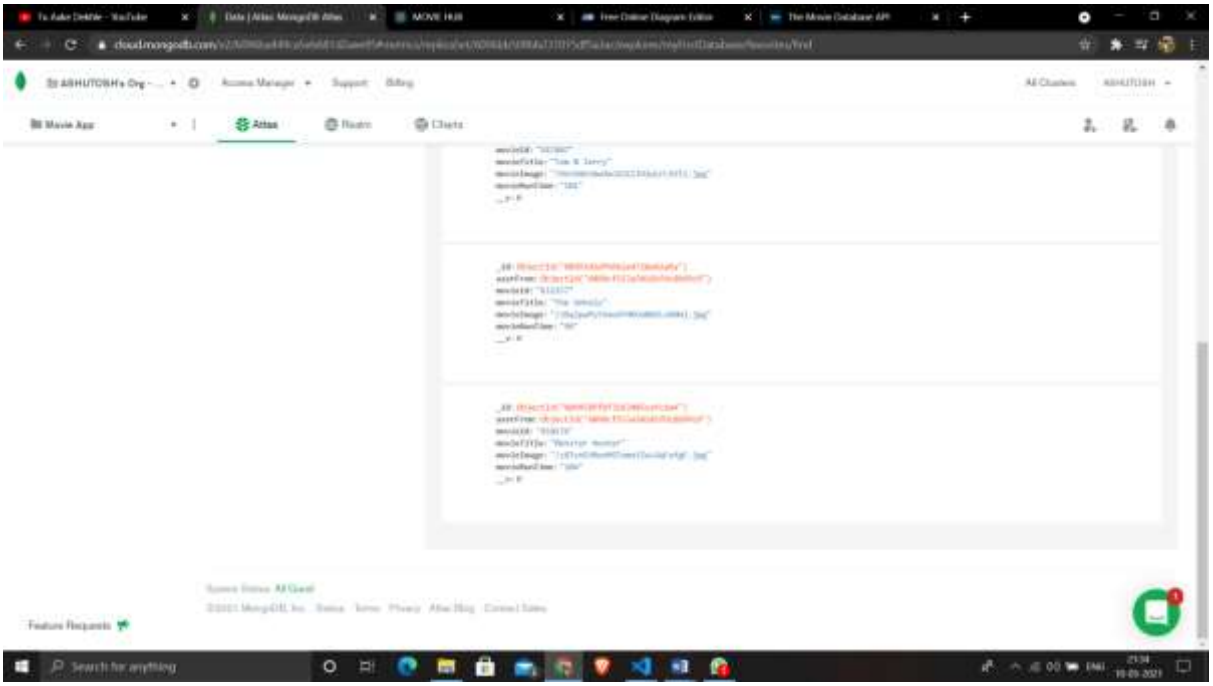
When movie detail added in favorite page:

Movie Name: Monster Hunter



Data of movie store in database which added as favorite:

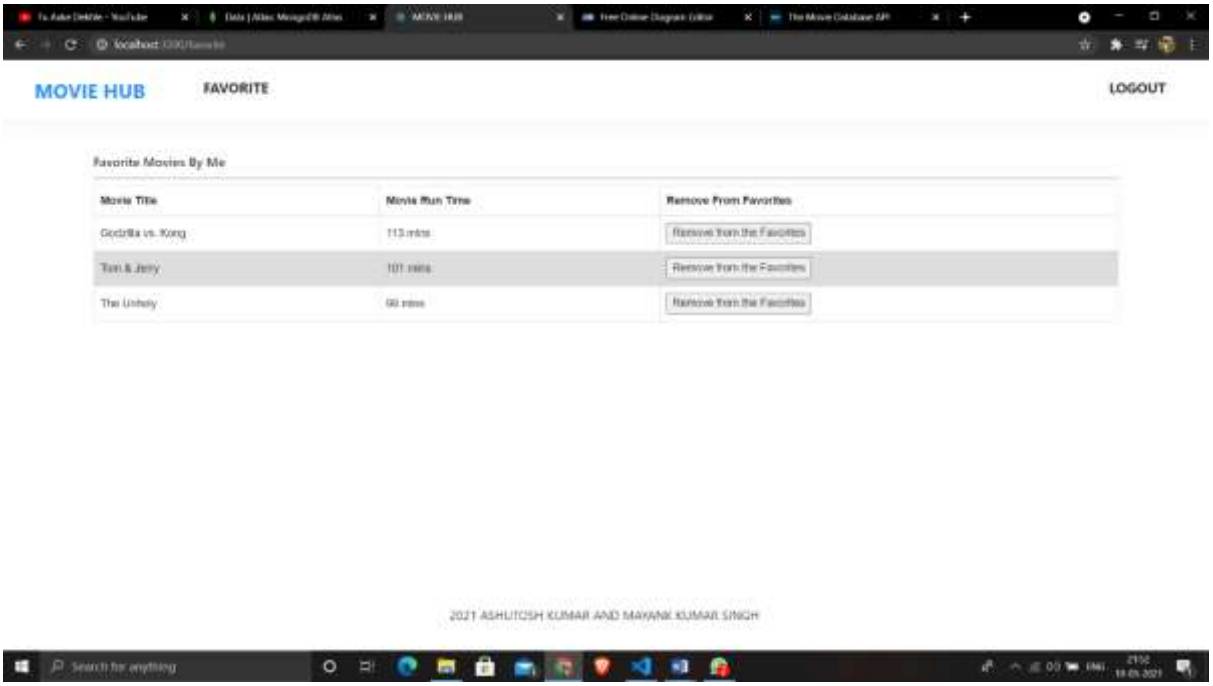
Movie Name: Monster Hunter



4.4. Testing for removing movie in favorite page:

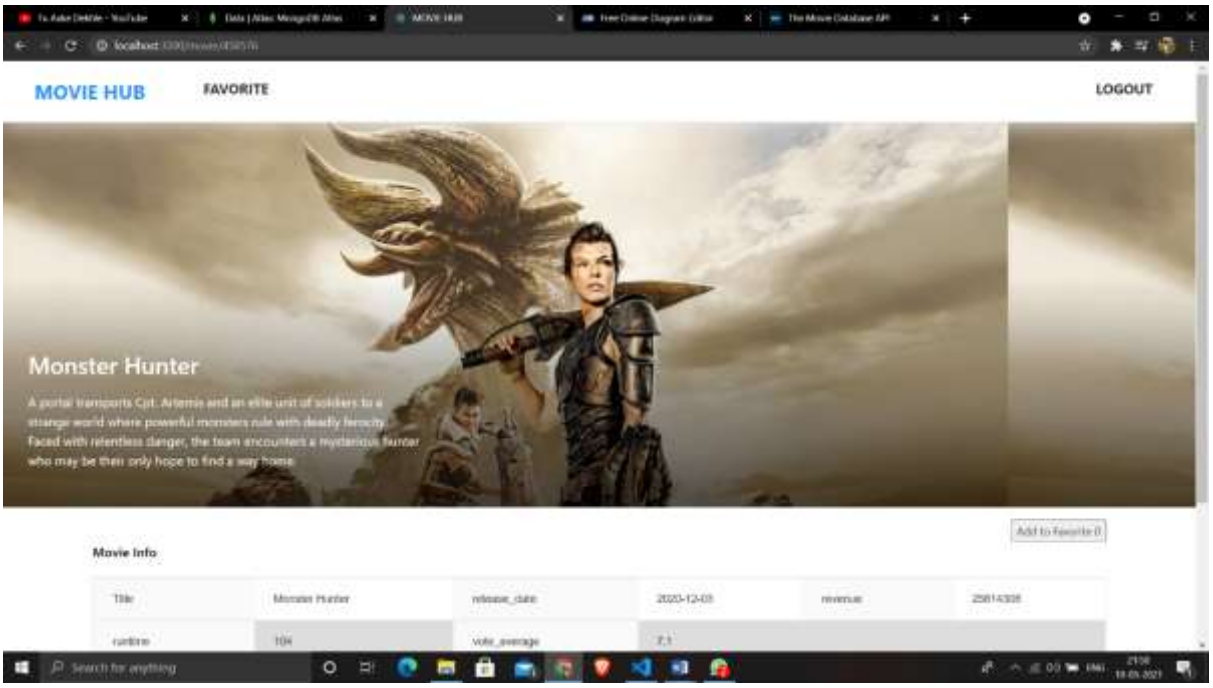
When user remove the movie from the favorite page:

Movie Name: Monster Hunter



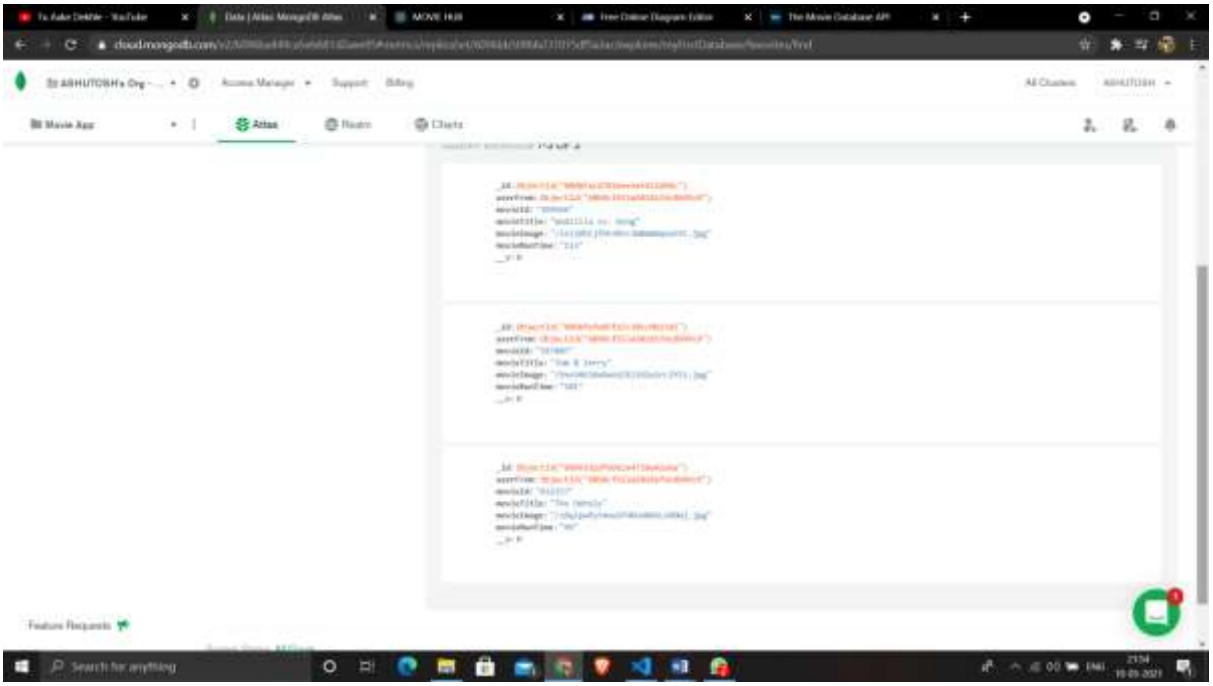
When user remove the movie, add as favorite button:

Movie Name: Monster Hunter



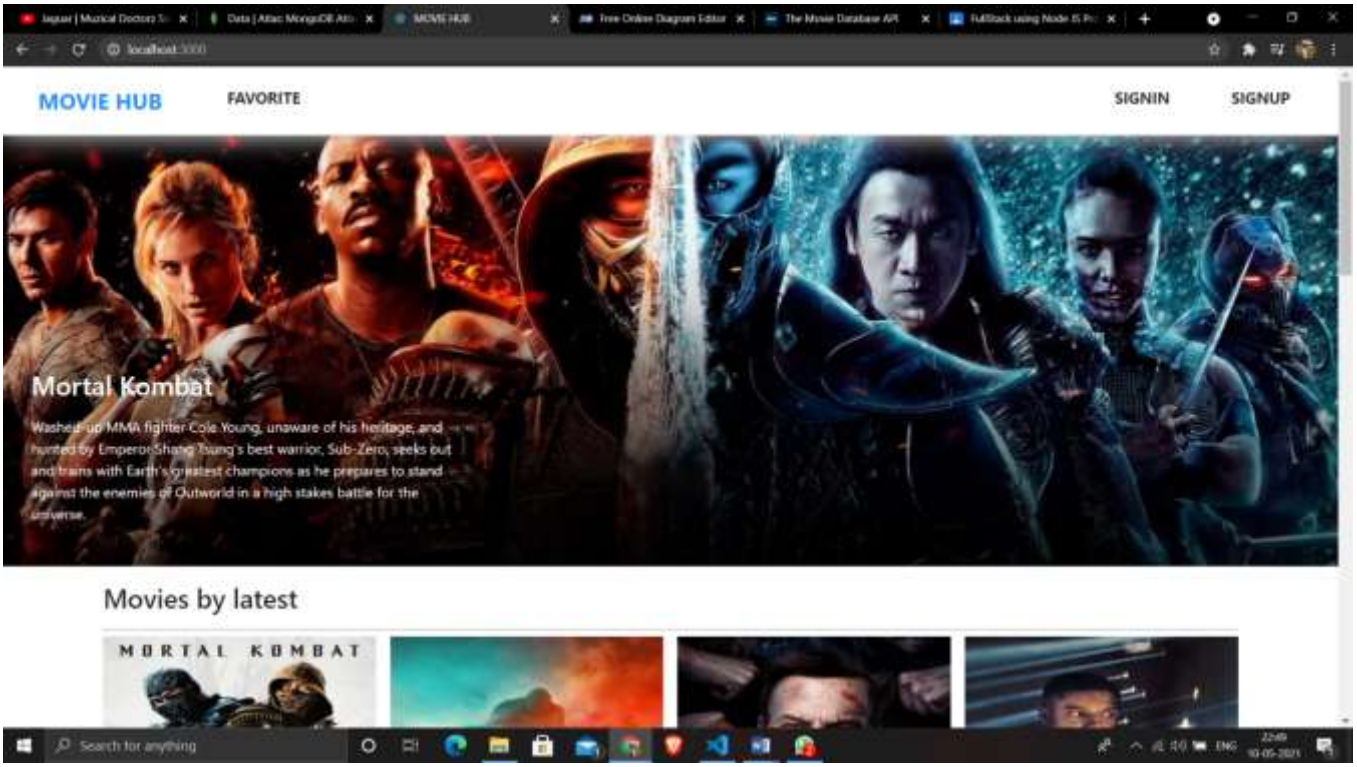
Movie Detail deleted from the database:

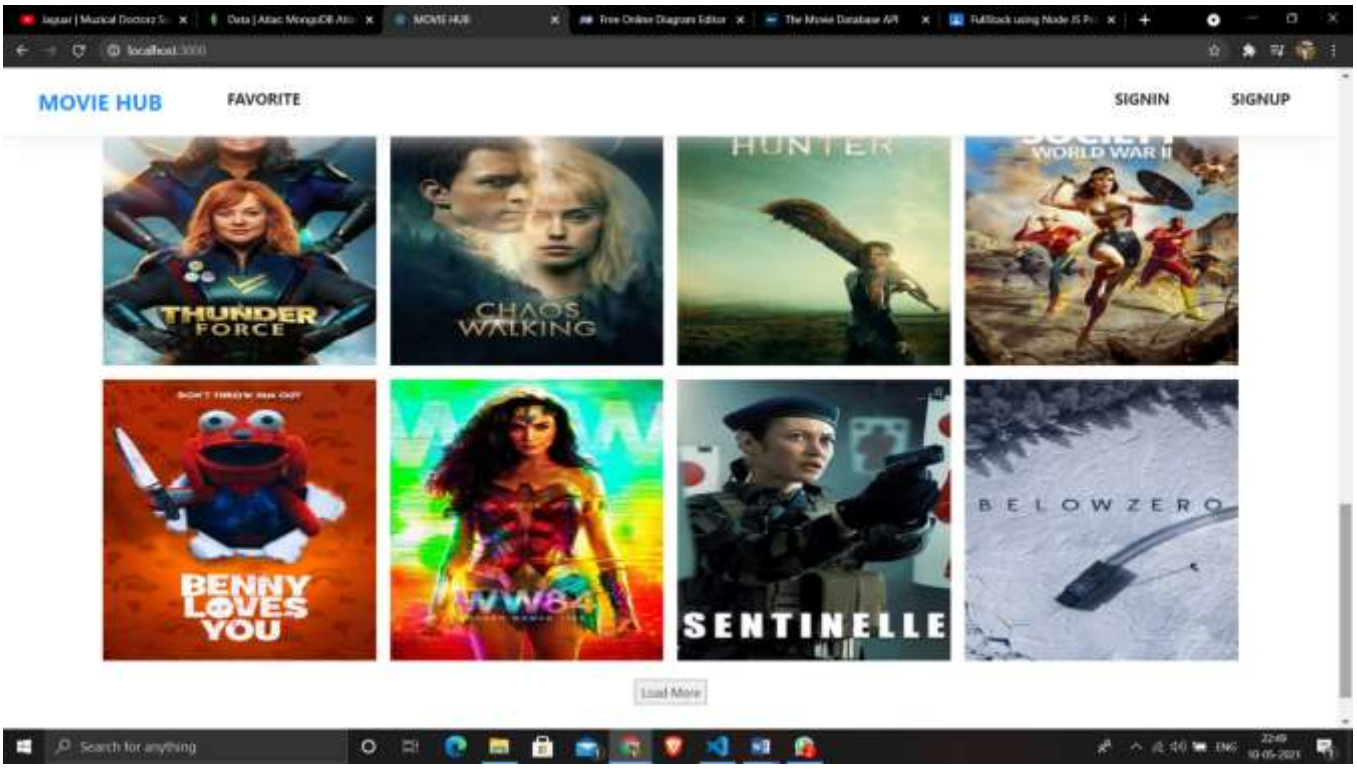
Movie Name: Monster Hunt



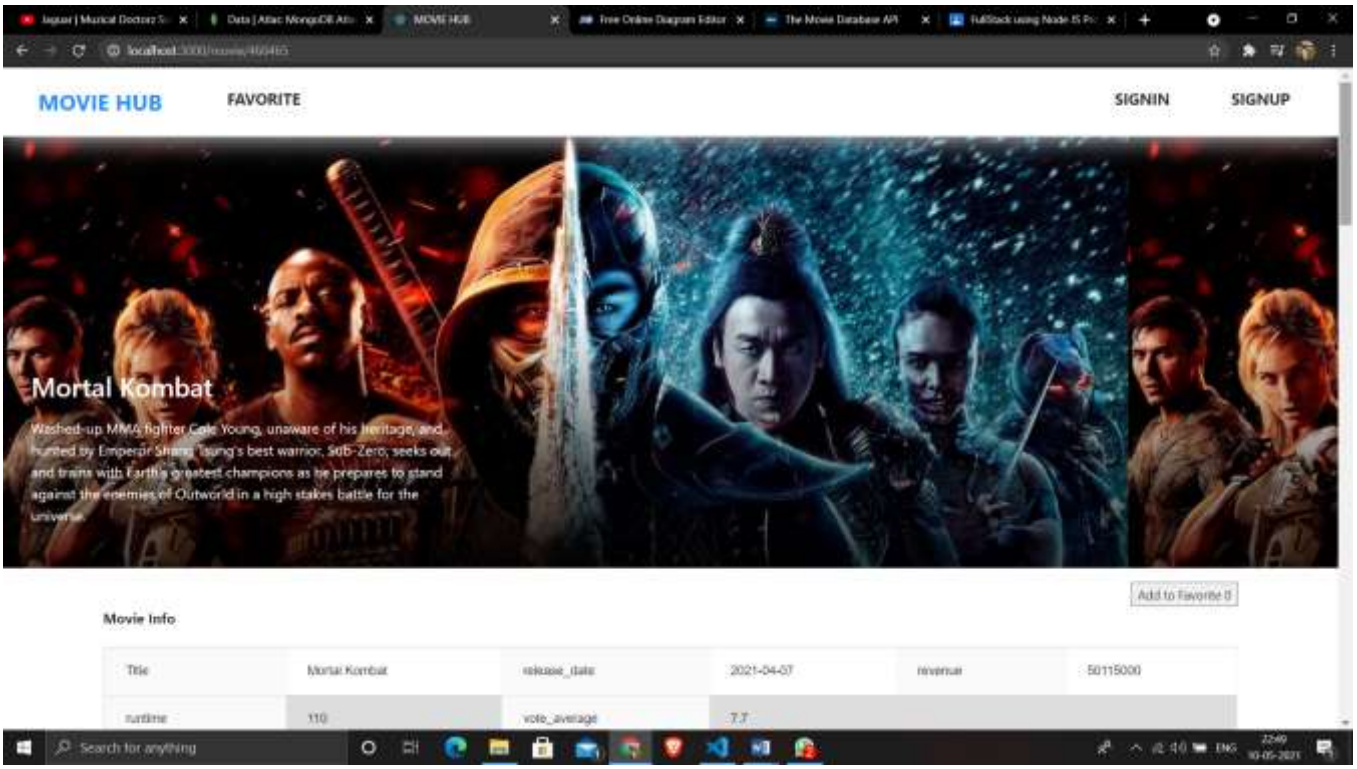
5. USER INTERFACE

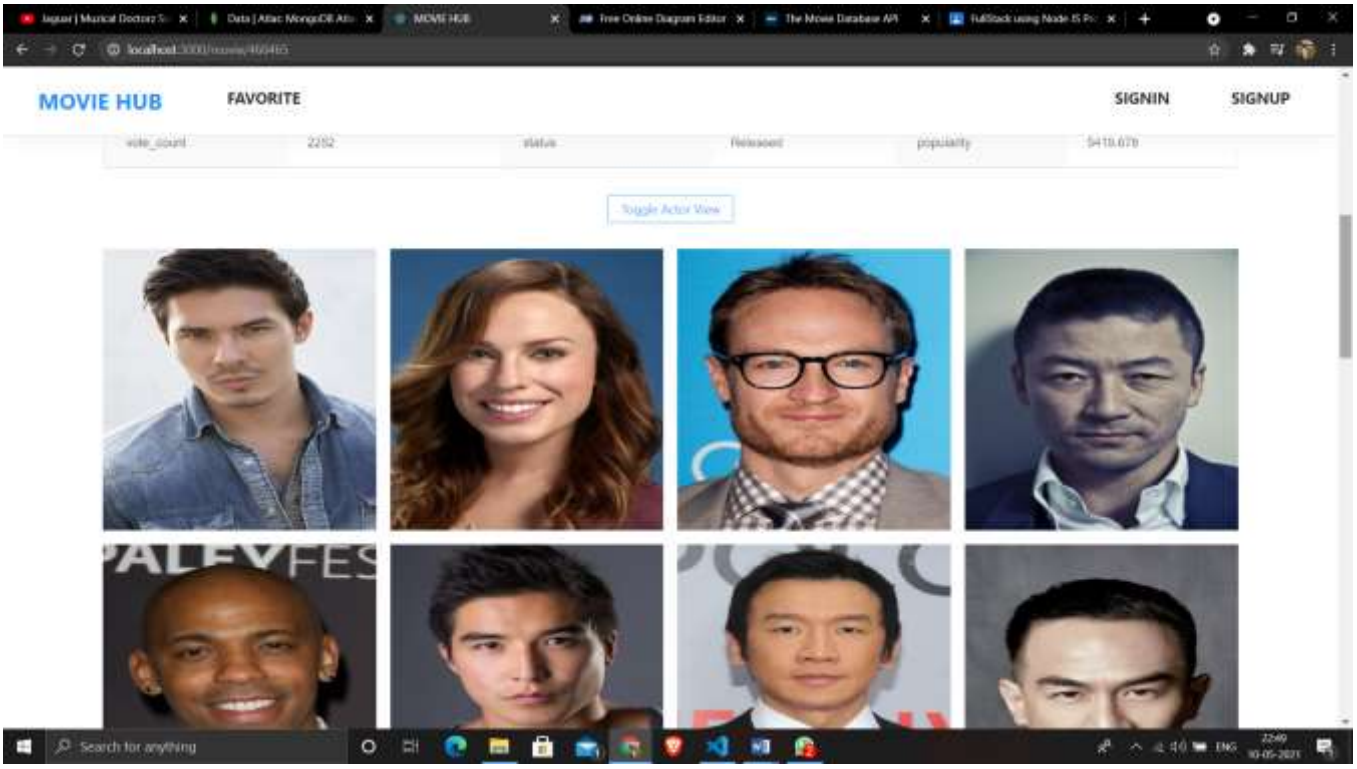
5.1 Home Page:



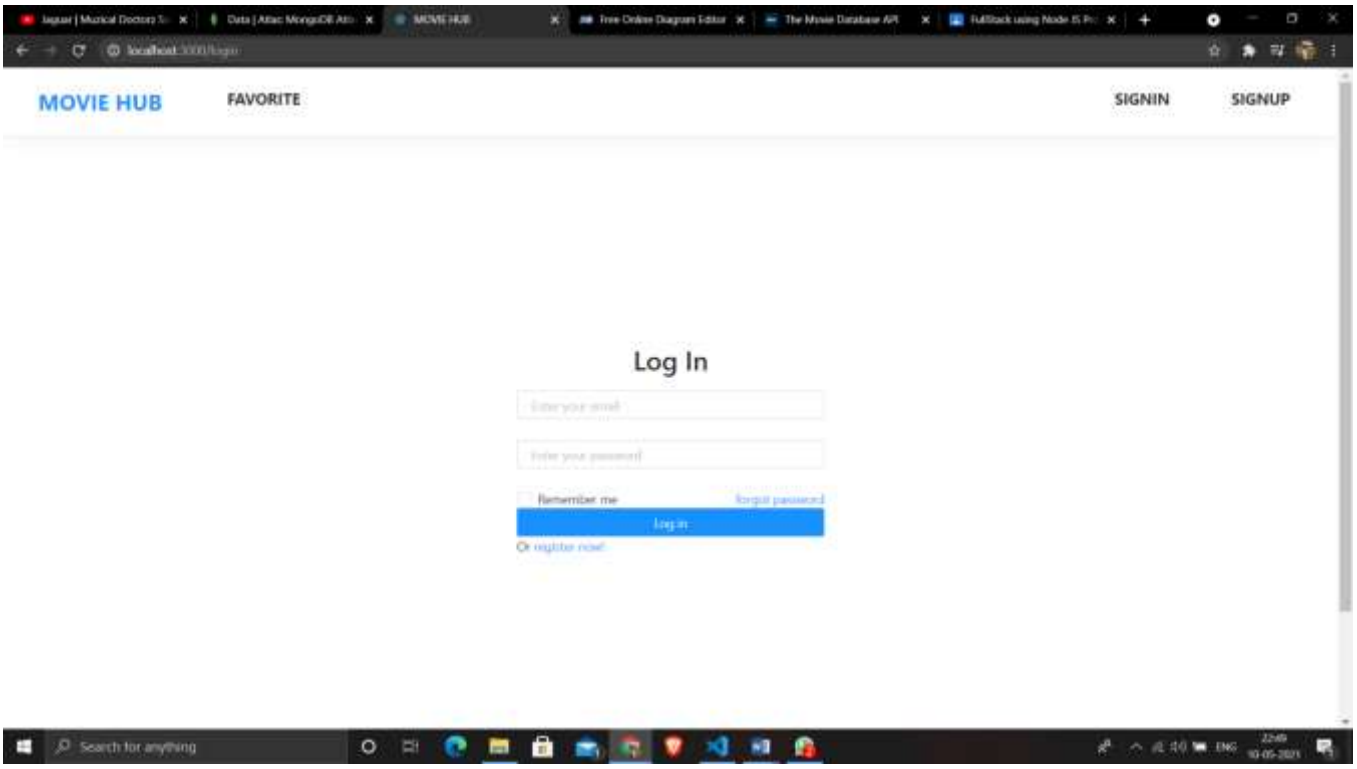


5.2 Movie Detail Page:

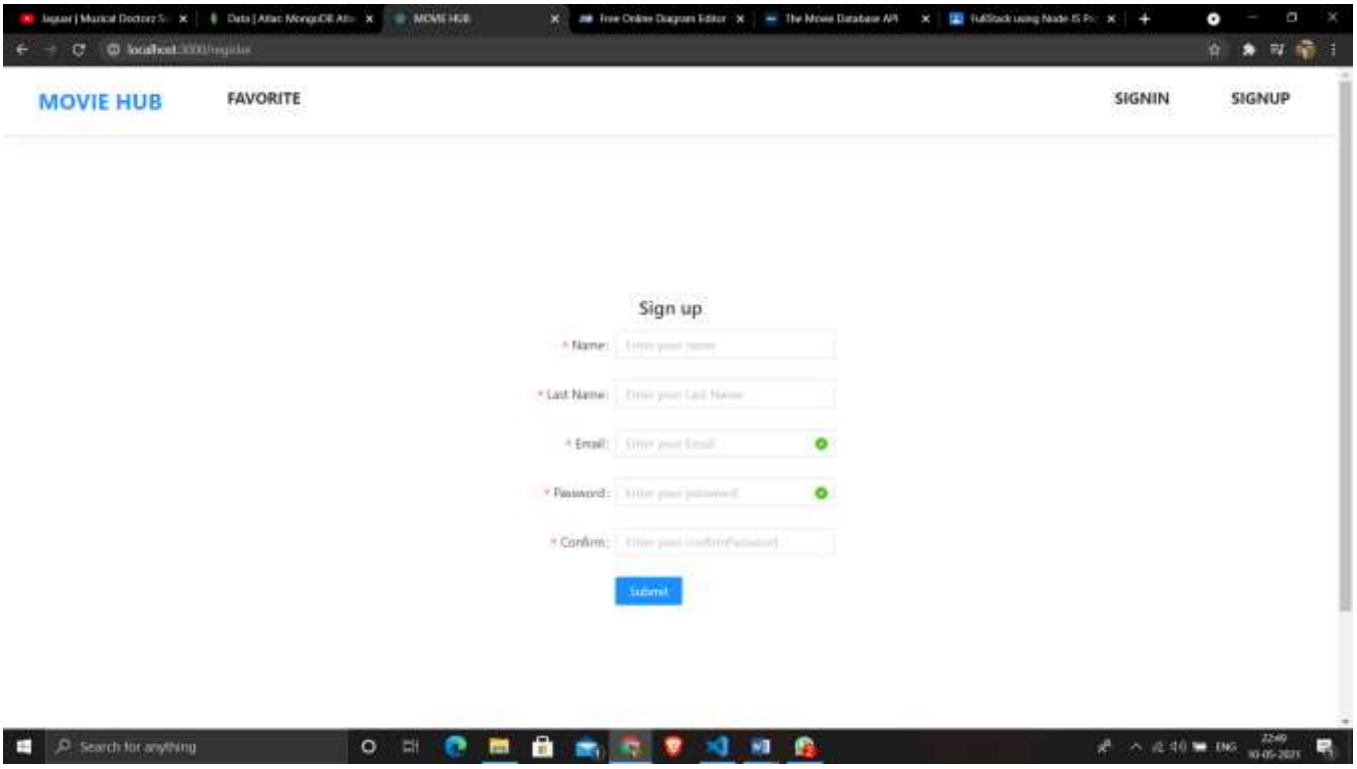




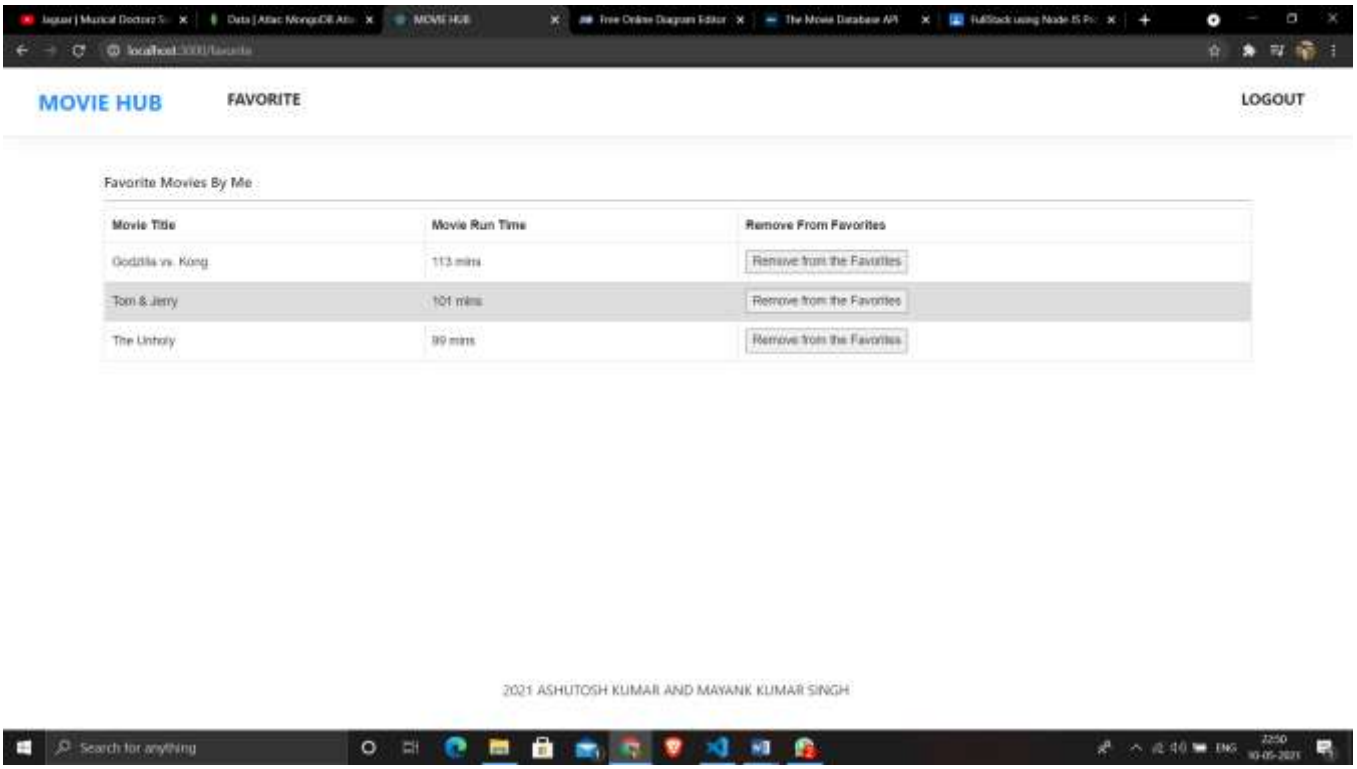
5.3 Sign In Page:



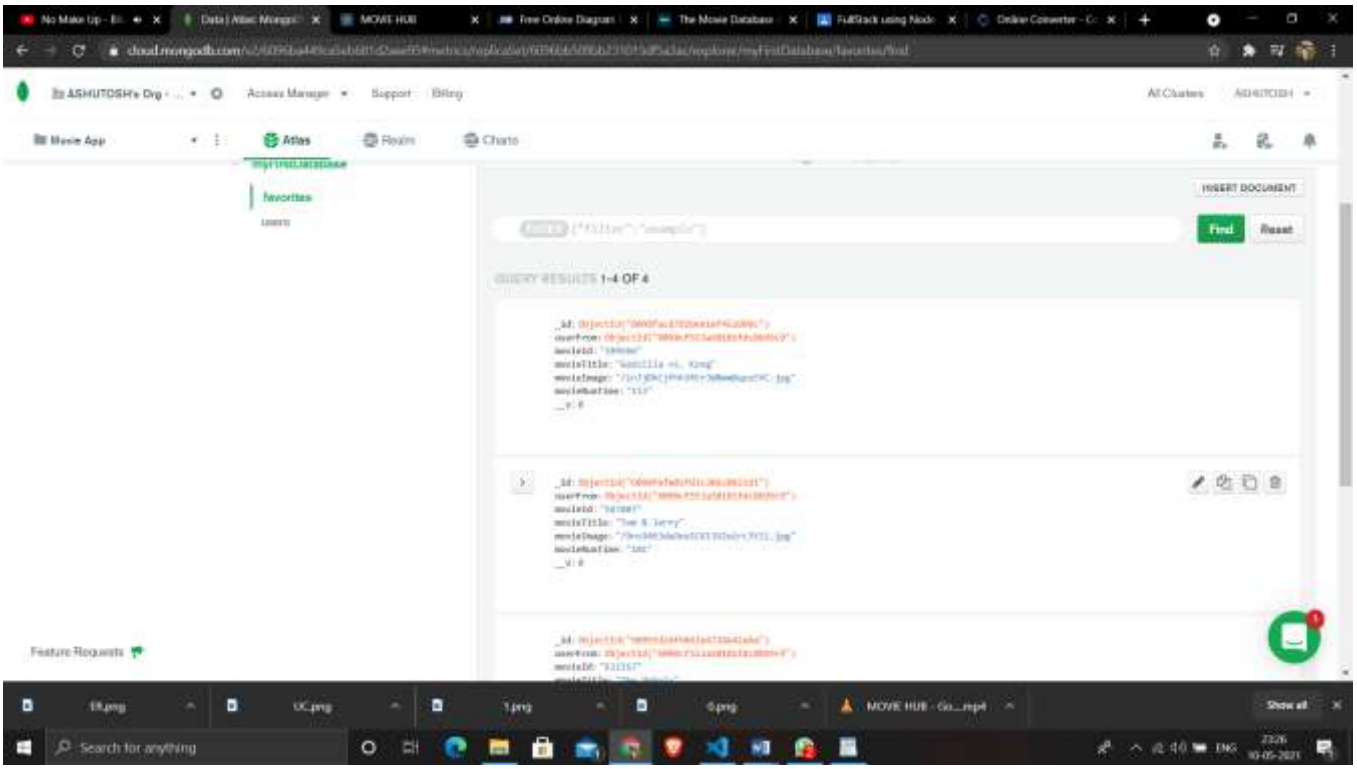
5.4 Sign Up Page:

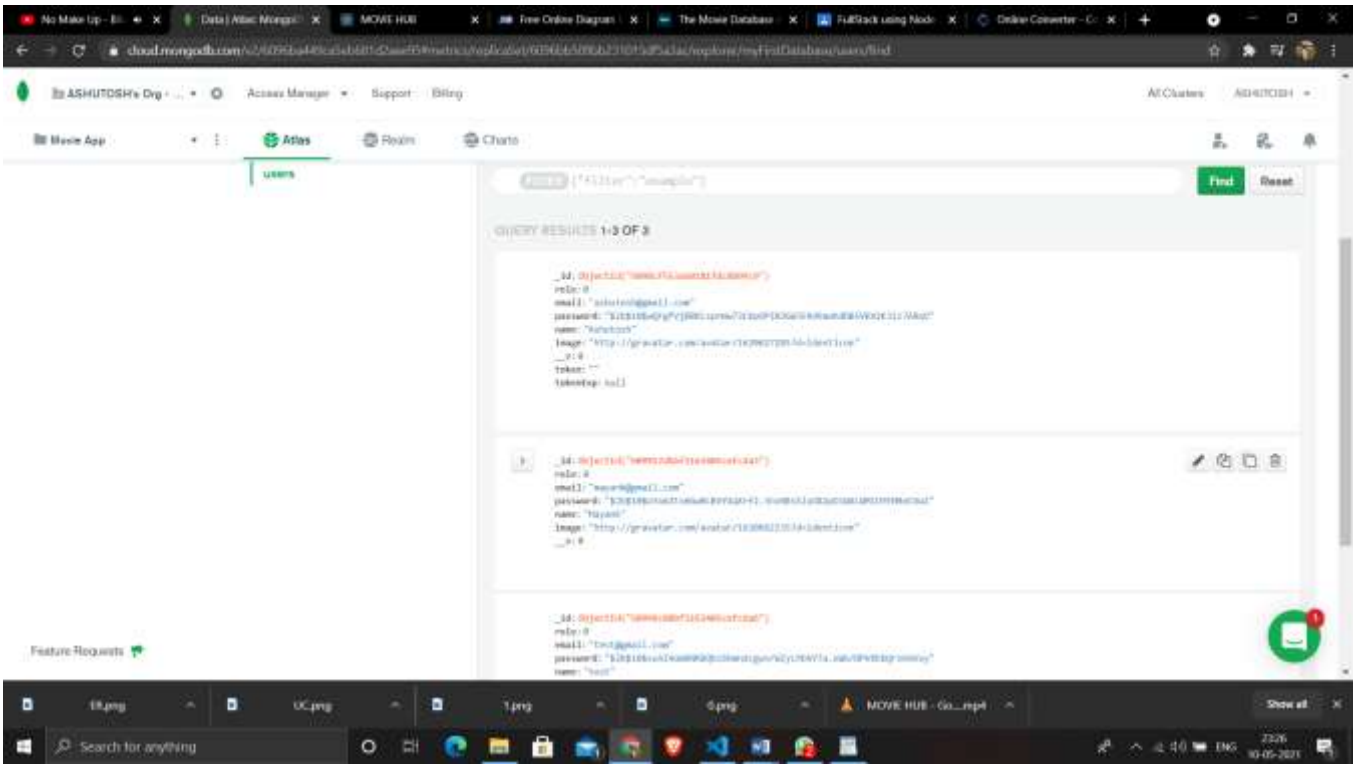


5.5 Favorite Page:



5.6. Database (User Collections and Favorites Collections):




Movie App
Atlas
Realm
Charts
users
Find
Reset
Feature Requests

6. CONCLUSION

This Project will be made by considering the needs of an online platform to all those customers who are looking to view and in search of good movies that entertain them most which includes thriller, action, suspense etc. which best fits their requirements, as this project is available online with latest film contents and videos, with easy navigation throughout the website will provide them a best place from where they can search and view their favorites movies or download it rather than surfing or accepting any premium membership for getting a good categories of movies throughout the internet.

7. REFERENCES

[1] https://stackoverflow.com/

[2] https://www.w3schools.com/bootstrap/

[3] https://space-facts.com/solar-system//

[4] https://getbootstrap.com/

[5] https://youtube.com/

[6] https://www.beta-labs.in/2020/06/html-basics_7.html

[7] https://www.nasa.gov/

[8] https://theplanets.org/solar-system/

8. APPENDIX

```
const express = require("express");

const app = express();

const path = require("path");

const cors = require('cors')


const bodyParser = require("body-parser");

const cookieParser = require("cookie-parser");


const config = require("./config/key");


// const mongoose = require("mongoose");

// mongoose

//   .connect(config.mongoURI, { useNewUrlParser: true })

//   .then(() => console.log("DB connected"))

//   .catch(err => console.error(err));


const mongoose = require("mongoose");

const connect = mongoose.connect(config.mongoURI, {

      useNewUrlParser: true,

      useUnifiedTopology: true,

      useCreateIndex: true,

      useFindAndModify: false

  })

  .then(() => console.log('MongoDB Connected...'))

  .catch(err => console.log(err));
```

```
app.use(cors())

//to not get any deprecation warning or error
//support parsing of application/x-www-form-urlencoded post data
app.use(bodyParser.urlencoded({ extended: true }));
//to get json data
// support parsing of application/json type post data
app.use(bodyParser.json());
app.use(cookieParser());

app.use('/api/users', require('./routes/users'));
app.use('/api/favorite', require('./routes/favorite'));

//use this to show the image you have in node js server to client (react js)
//https://stackoverflow.com/questions/48914987/send-image-path-from-node-js-express-server-to-react-client
app.use('/uploads', express.static('uploads'));

// Serve static assets if in production
if (process.env.NODE_ENV === "production") {

  // Set static folder
  // All the javascript and css files will be read and served from this folder
  app.use(express.static("client/build"));

  // index.html for all page routes    html or routing and naviagtion
```

```
  app.get("*", (req, res) => {
      res.sendFile(path.resolve(__dirname, "../client", "build", "index.html"));
  });
}

const port = process.env.PORT || 5000

app.listen(port, () => {
  console.log(`Server Listening on ${port}`)
});
```

```
import React from "react";
import moment from "moment";
import { Formik } from 'formik';
import * as Yup from 'yup';
import { registerUser } from "../../../_actions/user_actions";
import { useDispatch } from "react-redux";

import {
  Form,
  Input,
  Button,
} from 'antd';

const formItemLayout = {
  labelCol: {
```

```
  xs: { span: 24 },
  sm: { span: 8 },
 },
 wrapperCol: {
  xs: { span: 24 },
  sm: { span: 16 },
 },
};
const tailFormItemLayout = {
 wrapperCol: {
  xs: {
   span: 24,
   offset: 0,
  },
  sm: {
   span: 16,
   offset: 8,
  },
 },
};

function RegisterPage(props) {
 const dispatch = useDispatch();
 return (

  <Formik
   initialValues={{
```

```
  email: '',
  lastName: '',
  name: '',
  password: '',
  confirmPassword: ''
}}
validationSchema={Yup.object().shape({
  name: Yup.string()
    .required('Name is required'),
  lastName: Yup.string()
    .required('Last Name is required'),
  email: Yup.string()
    .email('Email is invalid')
    .required('Email is required'),
  password: Yup.string()
    .min(6, 'Password must be at least 6 characters')
    .required('Password is required'),
  confirmPassword: Yup.string()
    .oneOf([Yup.ref('password'), null], 'Passwords must match')
    .required('Confirm Password is required')
})}
onSubmit={(values, { setSubmitting }) => {
  setTimeout(() => {

    let dataToSubmit = {
      email: values.email,
      password: values.password,
```

```
      name: values.name,
      lastname: values.lastname,
      image: `http://gravatar.com/avatar/${moment().unix()}?d=identicon`
    };

    dispatch(registerUser(dataToSubmit)).then(response => {
      if (response.payload.success) {
        props.history.push("/login");
      } else {
        alert(response.payload.err.errmsg)
      }
    })

    setSubmitting(false);
  }, 500);
 }}
>
 {props => {
  const {
    values,
    touched,
    errors,
    dirty,
    isSubmitting,
    handleChange,
    handleBlur,
    handleSubmit,
```

```
  handleReset,
 } = props;
 return (
  <div className="app">
   <h2>Sign up</h2>
   <Form style={{ minWidth: '375px' }} {...formItemLayout} onSubmit={handleSubmit} >

    <Form.Item required label="Name">
     <Input
      id="name"
      placeholder="Enter your name"
      type="text"
      value={values.name}
      onChange={handleChange}
      onBlur={handleBlur}
      className={
       errors.name && touched.name ? 'text-input error' : 'text-input'
      }
     />
     {errors.name && touched.name && (
      <div className="input-feedback" style={{ marginTop:'0px' }}>{errors.name}</div>
     )}
    </Form.Item>

    <Form.Item required label="Last Name">
     <Input
      id="lastName"
```

```
            placeholder="Enter your Last Name"
            type="text"
            value={values.lastName}
            onChange={handleChange}
            onBlur={handleBlur}
            className={
              errors.lastName && touched.lastName ? 'text-input error' : 'text-input'
            }
          />
          {errors.lastName && touched.lastName && (
            <div className="input-feedback" style={{ marginTop:'0px' }}>{errors.lastName}</div>
          )}
        </Form.Item>

        <Form.Item required label="Email" hasFeedback validateStatus={errors.email && touched.email ? "error" : 'success'}>
          <Input
            id="email"
            placeholder="Enter your Email"
            type="email"
            value={values.email}
            onChange={handleChange}
            onBlur={handleBlur}
            className={
              errors.email && touched.email ? 'text-input error' : 'text-input'
            }
          />
          {errors.email && touched.email && (
```

```
            <div className="input-feedback" style={{ marginTop:'0px' }}>{errors.email}</div>
          )}
        </Form.Item>

        <Form.Item required label="Password" hasFeedback validateStatus={errors.password &&
touched.password ? "error" : 'success'}>
          <Input
            id="password"
            placeholder="Enter your password"
            type="password"
            value={values.password}
            onChange={handleChange}
            onBlur={handleBlur}
            className={
              errors.password && touched.password ? 'text-input error' : 'text-input'
            }
          />
          {errors.password && touched.password && (
            <div className="input-feedback" style={{ marginTop:'0px' }}>{errors.password}</div>
          )}
        </Form.Item>

        <Form.Item required label="Confirm" hasFeedback>
          <Input
            id="confirmPassword"
            placeholder="Enter your confirmPassword"
            type="password"
            value={values.confirmPassword}
```

```
            onChange={handleChange}
            onBlur={handleBlur}
            className={
              errors.confirmPassword && touched.confirmPassword ? 'text-input error' : 'text-input'
            }
           />
           {errors.confirmPassword && touched.confirmPassword && (
            <div className="input-feedback" style={{ marginTop:'0px' }}>{errors.confirmPassword}</div>
           )}
          </Form.Item>

          <Form.Item {...tailFormItemLayout}>
           <Button onClick={handleSubmit} type="primary" disabled={isSubmitting}>
            Submit
           </Button>
          </Form.Item>
         </Form>
        </div>
      );
    }}
  </Formik>
 );
};


export default RegisterPage
```