DevLab
Innovative Code Evaluation andLearning Experience

VISHAL N -22BCE1166

Objective

The primary goal of the **Ed-Tech Web Development Learning Platform** is to create an interactive, scalable, and engaging platform for individuals learning web development. By integrating cutting-edge technologies and leveraging AWS services, the platform ensures personalized, secure, and efficient delivery of learning resources and experiences. The specific objectives of the platform include:

1. **Interactive Learning Environment**
   Provide a hands-on coding experience with dynamic challenges in HTML, CSS, JavaScript, React, and Tailwind. Users can practice coding directly in the platform and receive real-time feedback.
2. **Personalized Resource Recommendations**
   Use machine learning to analyze user performance and preferences, recommending tailored resources (videos, exercises, and articles) to improve learning outcomes.
3. **Dynamic Content Storage and Retrieval**
   Store all learning resources, tutorials, and user-uploaded materials in a scalable and secure manner using AWS S3 and DynamoDB.
4. **Scalable Architecture for High Availability**
   Design the platform to handle large numbers of users simultaneously without downtime or performance degradation, leveraging AWS Elastic Beanstalk and CloudFront.
5. **Secure User Authentication and Data Protection**
   Ensure robust security for user data, authentication, and authorization using AWS Cognito, IAM, and encryption protocols.
6. **Community Engagement Through Forums**
   Foster collaboration and engagement among learners through an interactive forum for QCA, discussions, and sharing learning tips.
7. **Performance Tracking and Analytics**
   Track user progress through detailed analytics and provide visual dashboards that help users monitor their learning journey.
8. **Cost-Efficient Resource Management**
   Utilize AWS tools such as CloudWatch, CloudTrail, and AWS Budgets to maintain operational efficiency while keeping the platform cost-effective.
9. **Real-Time Challenge Evaluation**
   Implement serverless functions (AWS Lambda) for evaluating user submissions to challenges, ensuring fast and accurate feedback.
10. **Seamless Integration of Frontend and Backend**
    Develop a user-friendly frontend hosted on AWS S3 and powered by a Flask backend deployed using Elastic Beanstalk, with communication managed via AWS API Gateway.

11. **Adaptive Learning Features**
    Use analytics and feedback loops to adapt the platform to user needs, enabling personalized learning paths and suggestions for improvement.
12. **Comprehensive Monitoring and Reporting**
    Enable end-to-end monitoring of platform performance and user activity, ensuring smooth operation and early detection of issues through AWS CloudWatch and CloudTrail.
13. **Future-Ready Design**
    Incorporate flexibility in the platform architecture to allow for easy integration of new features, services, and content as the platform evolves.
14. **Regulatory Compliance and Privacy Protection**
    Adhere to data privacy regulations, ensuring user consent for data usage and secure handling of personal information throughout the platform.

List of Modules

1. User Authentication and Profile Management
2. Interactive Learning Dashboard
3. Challenges and Progress Tracking
4. Resource Recommendation System
5. Discussion Forum
6. Admin Panel for Content Management
7. Real-Time Challenge Evaluation
8. Frontend Search and Content Display
9. Monitoring and Reporting
10. Cost Monitoring and Optimization
11. Secure Data Management
12. Comprehensive Logging and Alerts
13. Regulatory and Data Compliance

Detailed description of the module

1. User Authentication and Profile Management

- Description:
  Handles user registration, login, and profile customization securely. Ensures personalized experiences by storing user-specific settings and progress.
- Key Features:
  - Secure login and registration.
  - Password reset and session management.
  - User-specific data stored in DynamoDB.
  - Role-based access for learners and admins.
- AWS Services Used:
  - AWS IAM: Role-based access.
  - Amazon DynamoDB: Stores user data and preferences.

2. Interactive Learning Dashboard

- Description:
  Displays user-specific learning paths, coding progress, tutorials, and recommended resources.
- Key Features:
  - Tracks user challenges completed and progress percentages.
  - Personalized recommendations based on activity.
  - Dynamic UI with React and AWS Amplify.
- AWS Services Used:
  - AWS DynamoDB: Progress tracking.
  - AWS Amplify: Frontend hosting and integration.

3. Challenges and Progress Tracking

- Description:
  Presents dynamic coding challenges and tracks user performance.
- Key Features:

    - Dynamic generation of coding challenges.
    - Leaderboard tracking for competitive learning.
    - Scoring system for performance analysis.
- AWS Services Used:
    - Amazon DynamoDB: Stores challenge data and scores.
    - AWS Elastic Beanstalk: Hosts the backend for challenge processing.

4. Resource Recommendation System

- Description:
  Recommends personalized learning resources like tutorials and exercises based on user performance.
- Key Features:
    - Resources stored and organized in S3.
    - Recommendations derived from stored user progress data.
- AWS Services Used:
    - Amazon S3: Stores learning resources.
    - Amazon DynamoDB: Metadata storage.

5. Discussion Forum

- Description:
  A collaborative space for learners to engage in discussions.
- Key Features:
    - Users can post questions and comments.
    - Topics categorized for better navigation.
- AWS Services Used:
    - Amazon DynamoDB: Stores forum threads and comments.
    - AWS Amplify: Hosts the frontend forum interface.

6. Admin Panel for Content Management

- Description:
  Provides tools for managing challenges, tutorials, and user data securely.
- Key Features:
  - CRUD operations for challenges and tutorials.
  - Role-specific access for admins.
  - Data analytics dashboard for monitoring platform usage.
- AWS Services Used:
  - Amazon DynamoDB: Stores admin data.
  - AWS IAM: Manages access control.

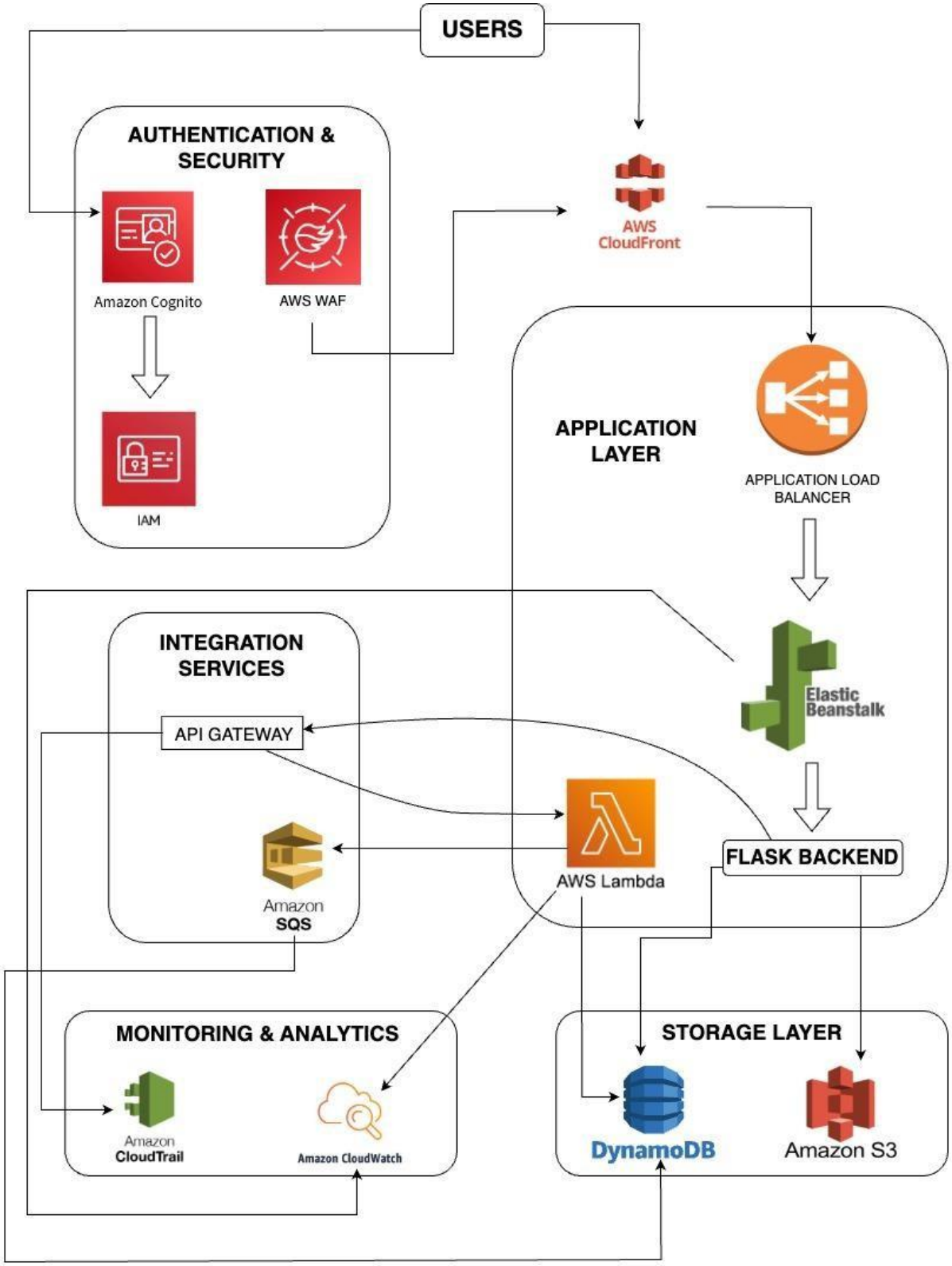
7. Monitoring and Reporting

- Description:
  Tracks platform performance and ensures smooth operation.
- Key Features:
  - Real-time monitoring of user activity and system health.
  - Alerts for unusual activity or performance degradation.
- AWS Services Used:
  - AWS CloudWatch: Monitors performance and generates alerts.
  - AWS CloudTrail: Logs and audits API activity.

8. Frontend Search and Content Display

- Description:
  User-friendly interface for accessing tutorials, challenges, and learning resources.
- Key Features:
  - Search bar with advanced filters.
  - Responsive design powered by React.
- AWS Services Used:
  - AWS Amplify: Frontend hosting.
  - Amazon S3: Stores static content.

Architecture



List of AWS Services Used

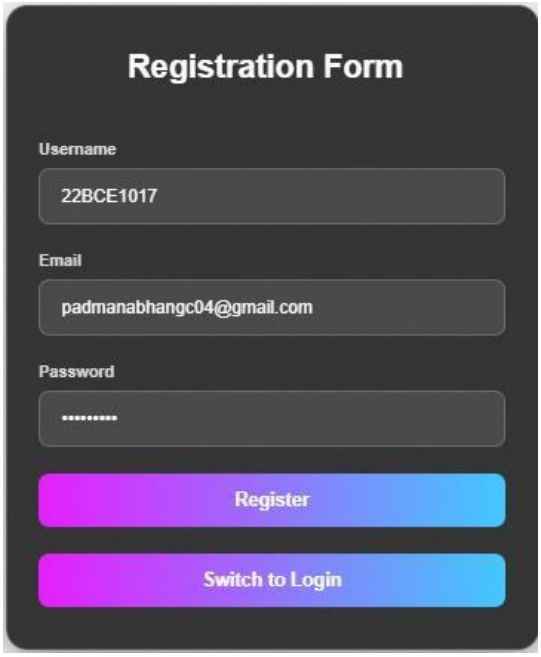
1. **AWS CI/CD Tools**
    - Automates code integration, testing, and deployment processes.
2. **Amazon EC2 (Elastic Compute Cloud)**
    - Provides scalable compute resources to host backend or application logic.
3. **Amazon S3 (Simple Storage Service)**
    - Stores static assets like frontend files, resources, and backups with high availability.
4. **Amazon DynamoDB**
    - A NoSQL database for storing user data, progress tracking, and metadata.
5. **AWS IAM (Identity and Access Management)**
    - Manages secure access to resources with role-based permissions and fine-grained control.
6. **AWS Elastic Beanstalk**
    - Simplifies deployment and scaling of web applications by automating infrastructure management.
7. **AWS Amplify**
    - Provides a simplified workflow for frontend hosting and seamless integration with backend services.

AWS CloudWatch

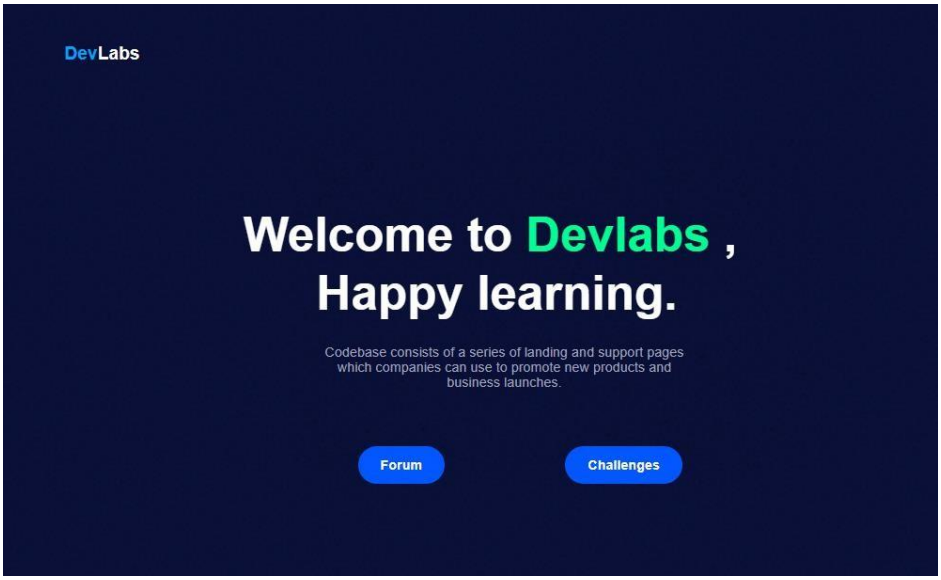
- Monitors system performance, logs application activity, and sets up alerts for potential issues.

User Interface Screenshots

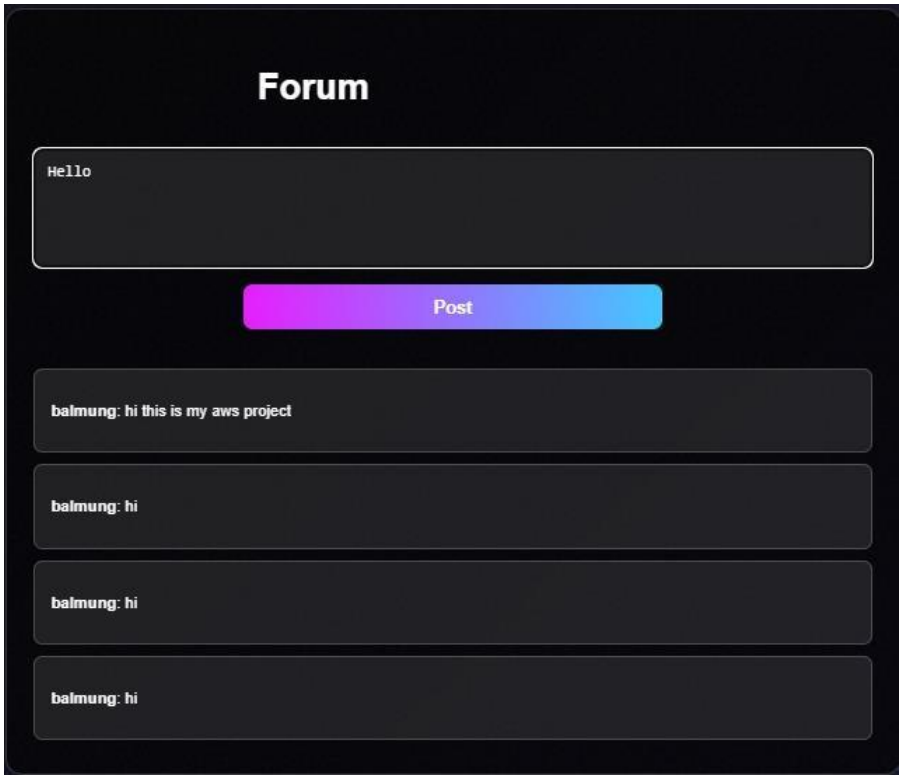
1. Registration / Login Form



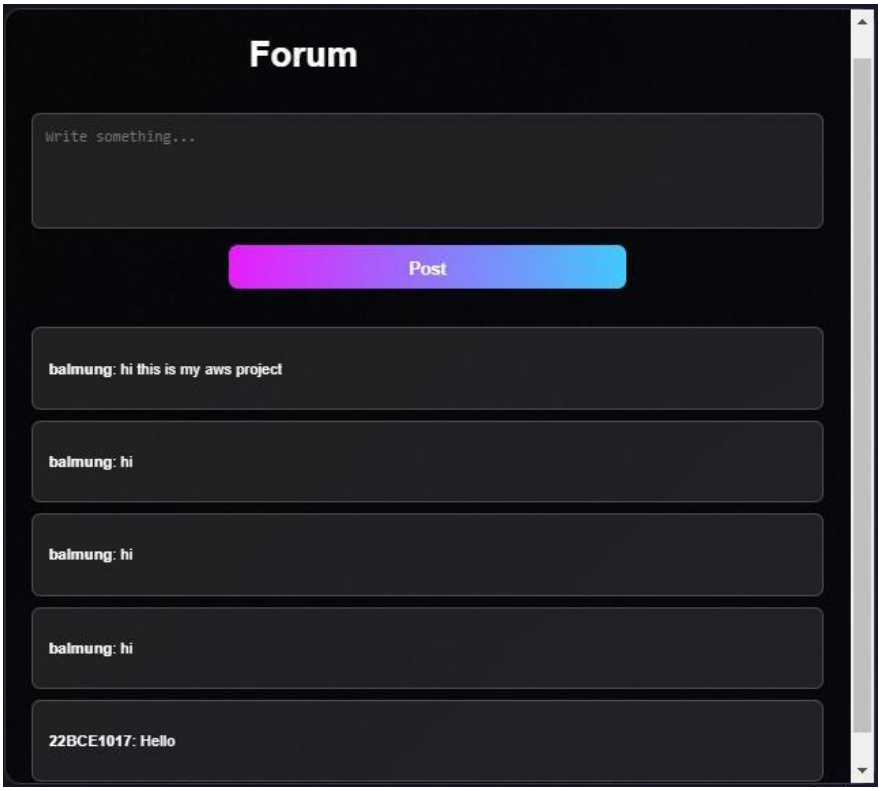
2. Landing Page



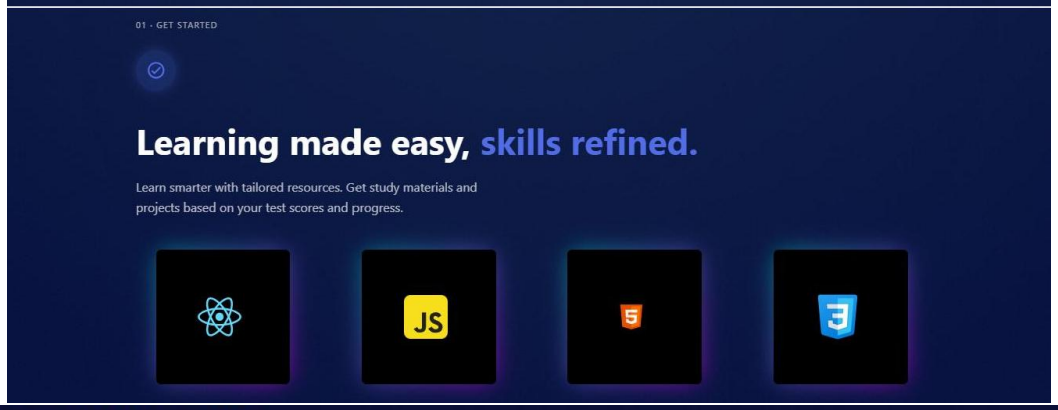
3. Forum

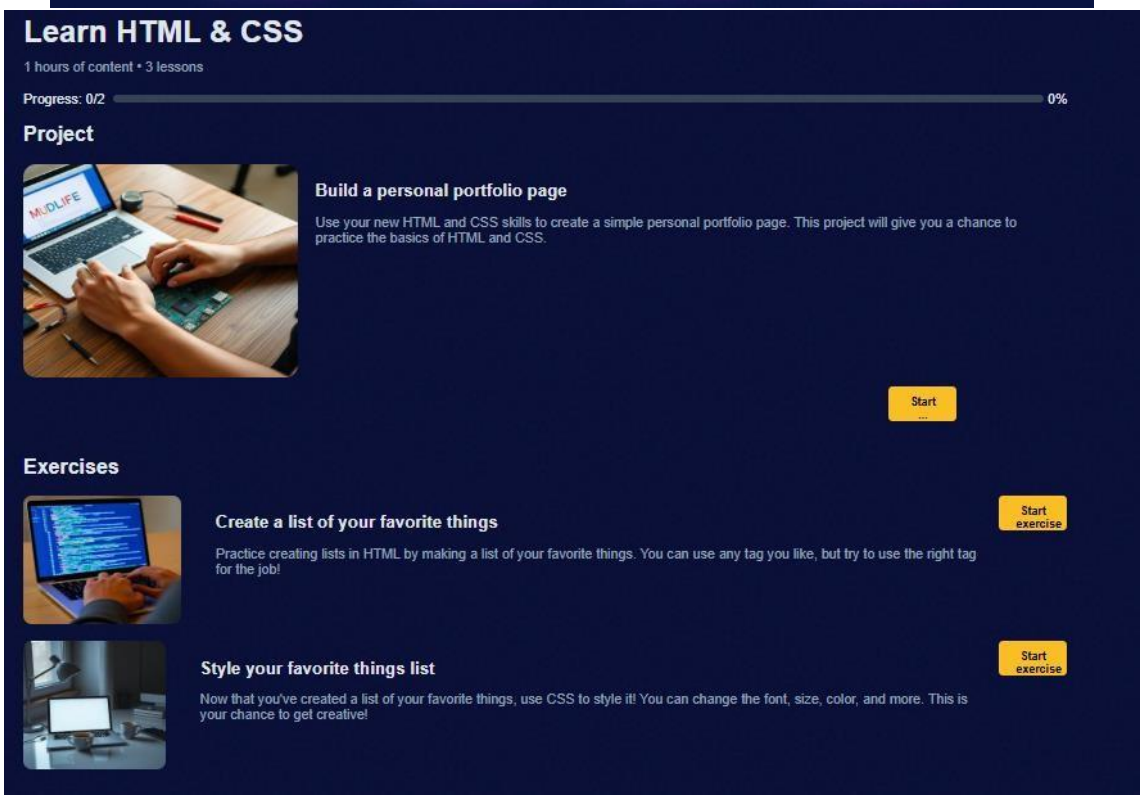


After posting forum

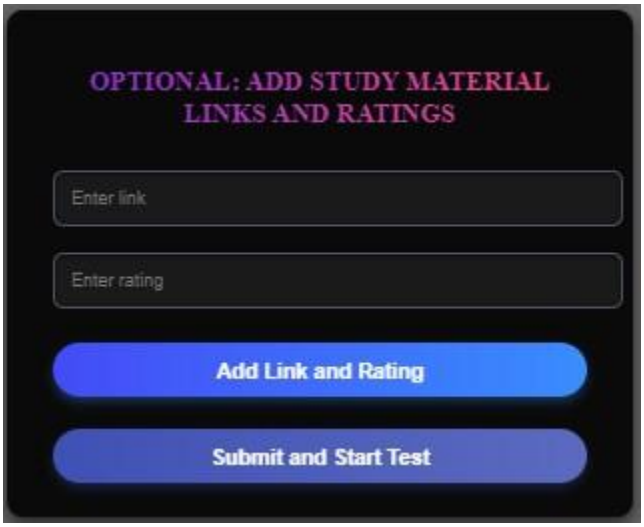


4. Challenges Page

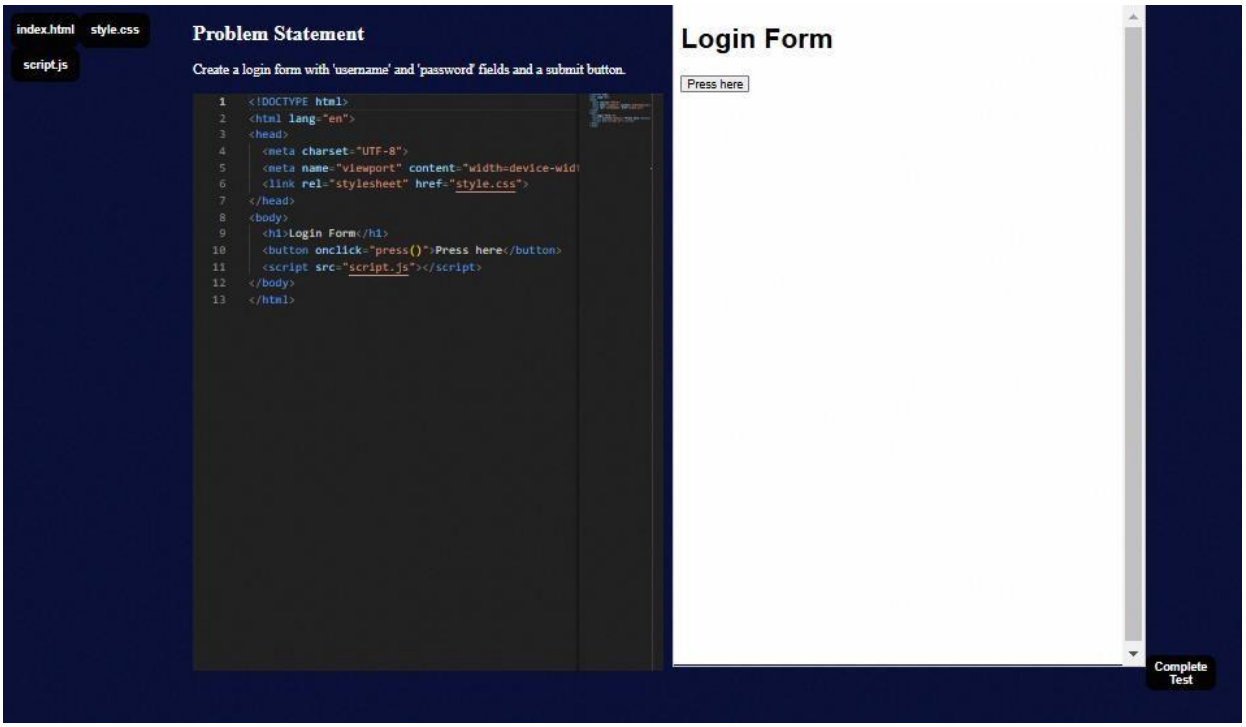


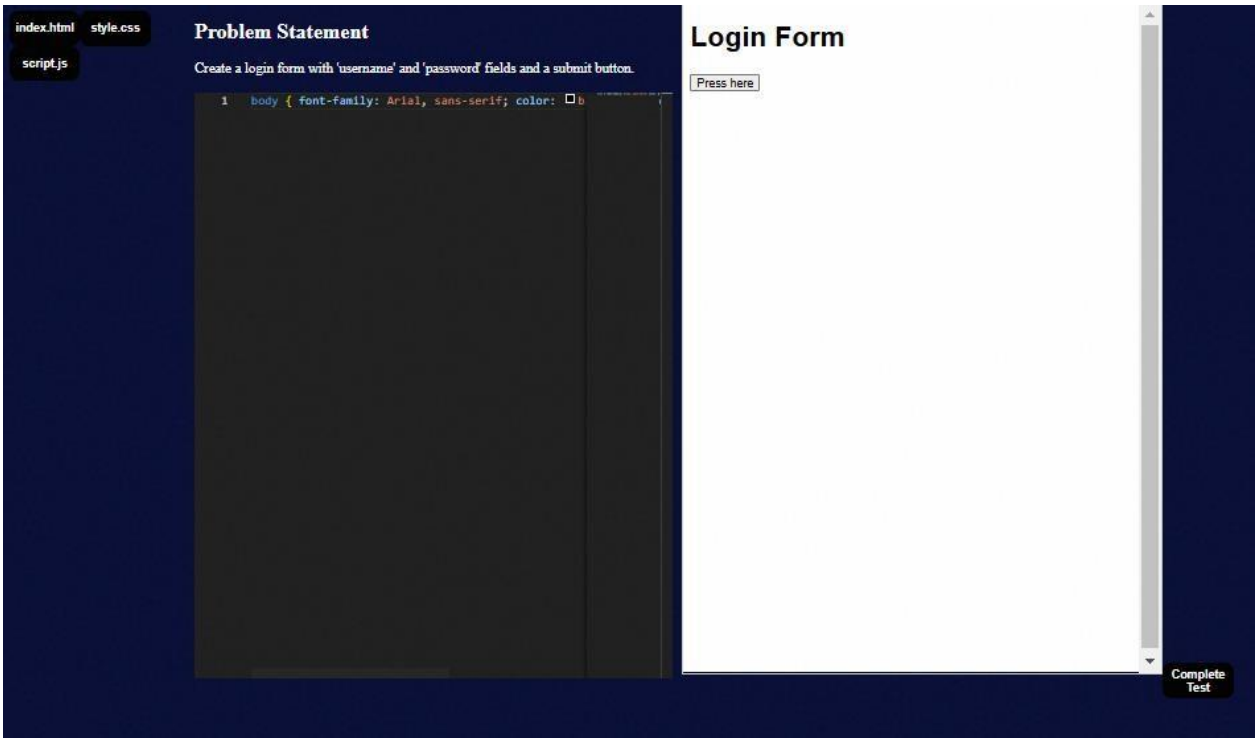


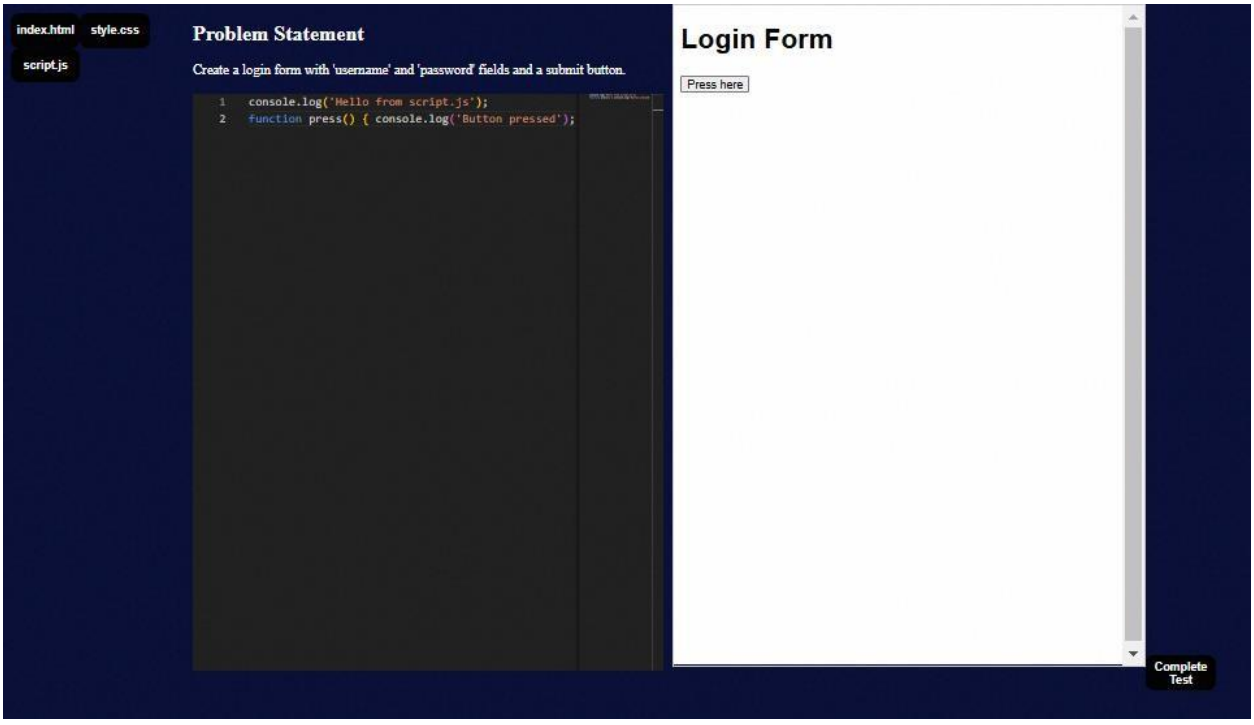
5. Resource Collection Page



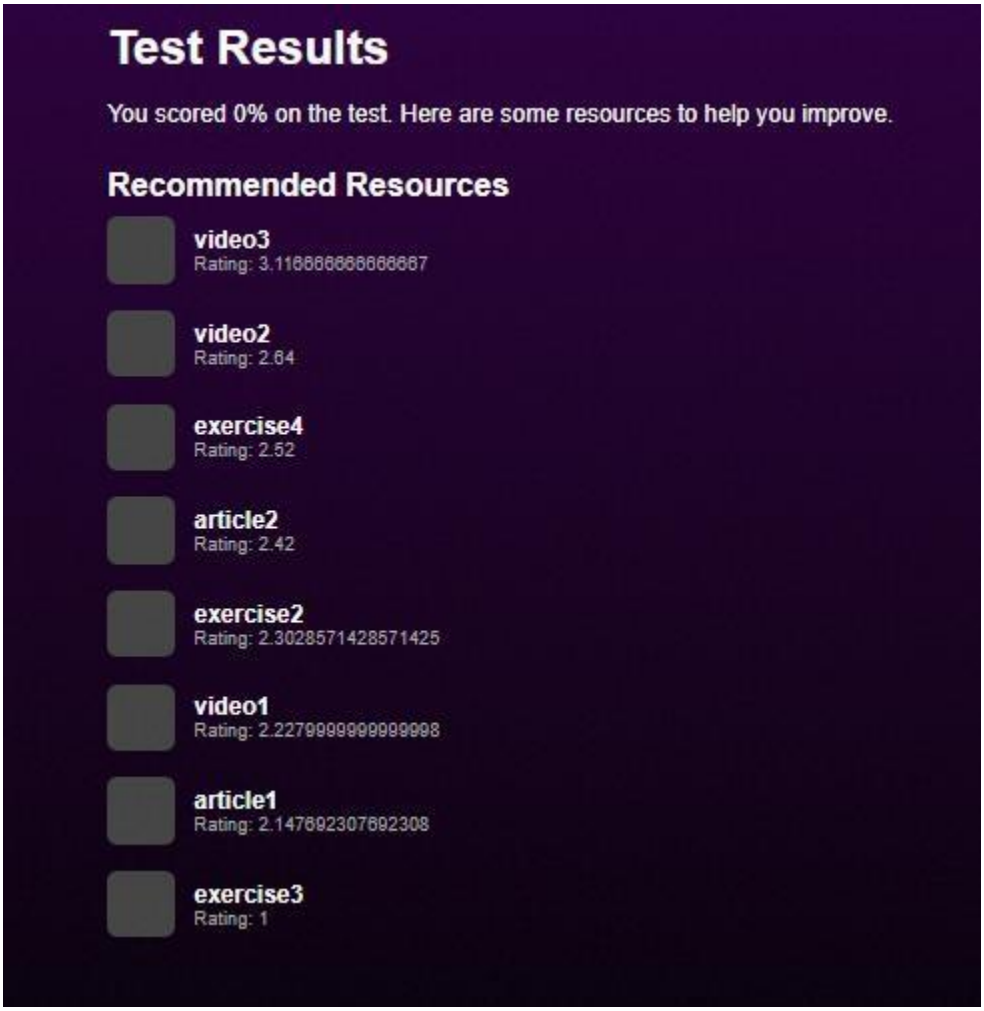
6. Coding Workspace (Editor and preview)




index.html
style.css
script.js
Problem Statement
Create a login form with 'username' and 'password' fields and a submit button.
1 body { font-family: Arial, sans-serif; color:
Login Form
Press here
Complete Test

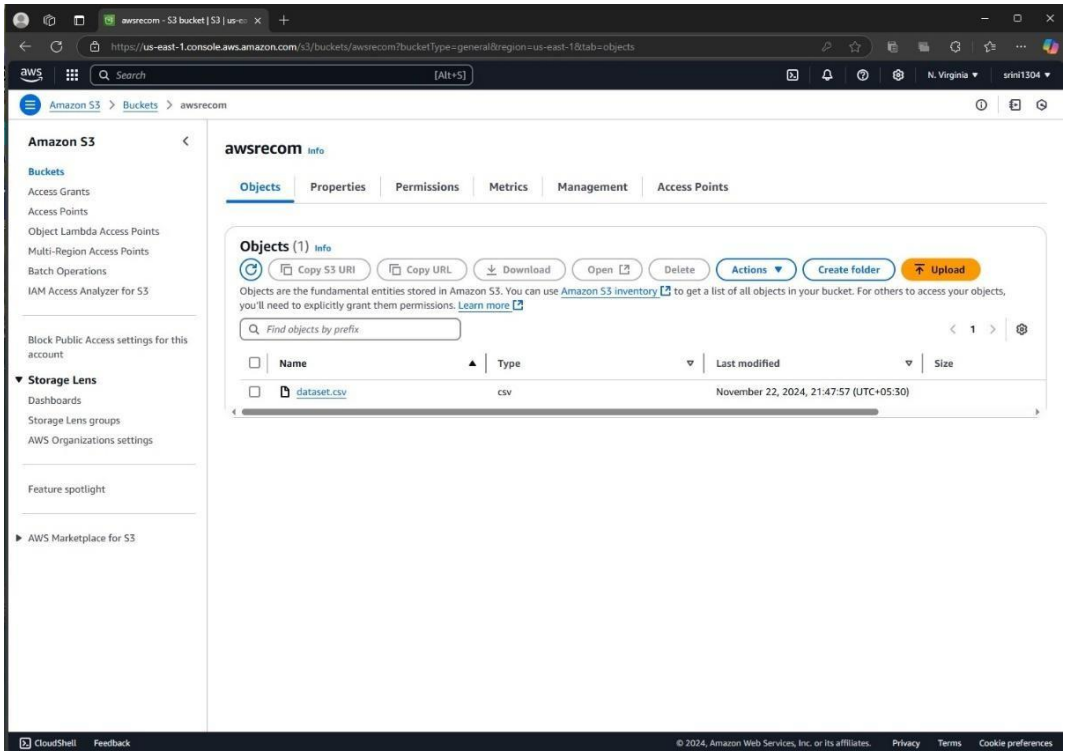

index.html
style.css
script.js
Problem Statement
Create a login form with 'username' and 'password' fields and a submit button.
1 console.log('Hello from script.js');
2 function press() { console.log('Button pressed');
Login Form
Press here
Complete Test

7. Results Page

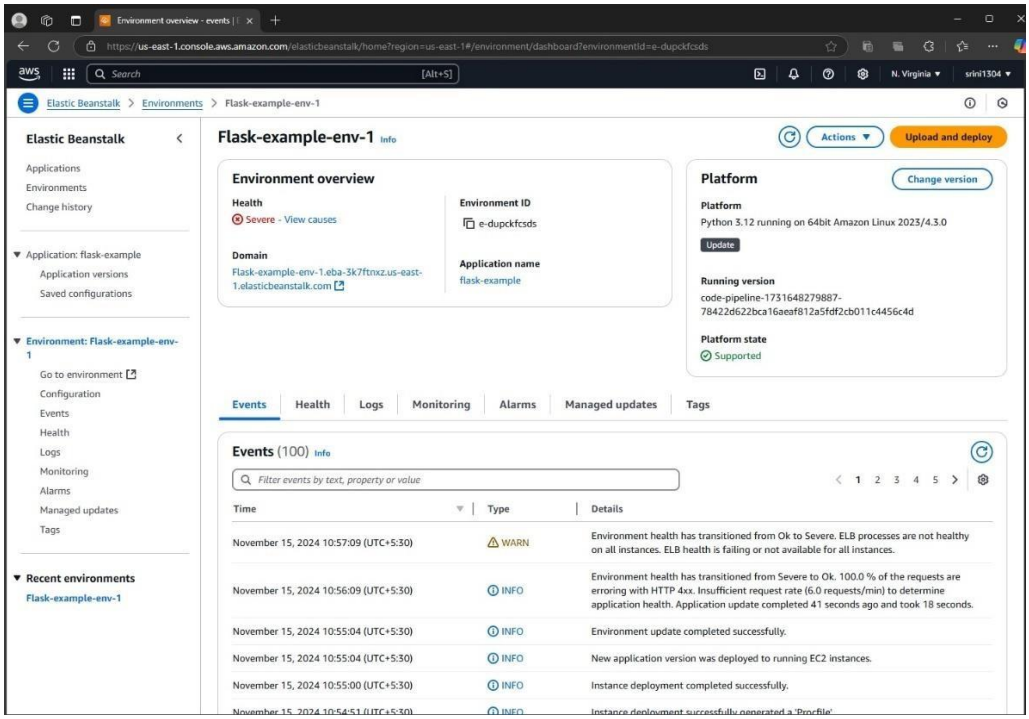


AWS Services Screenshot

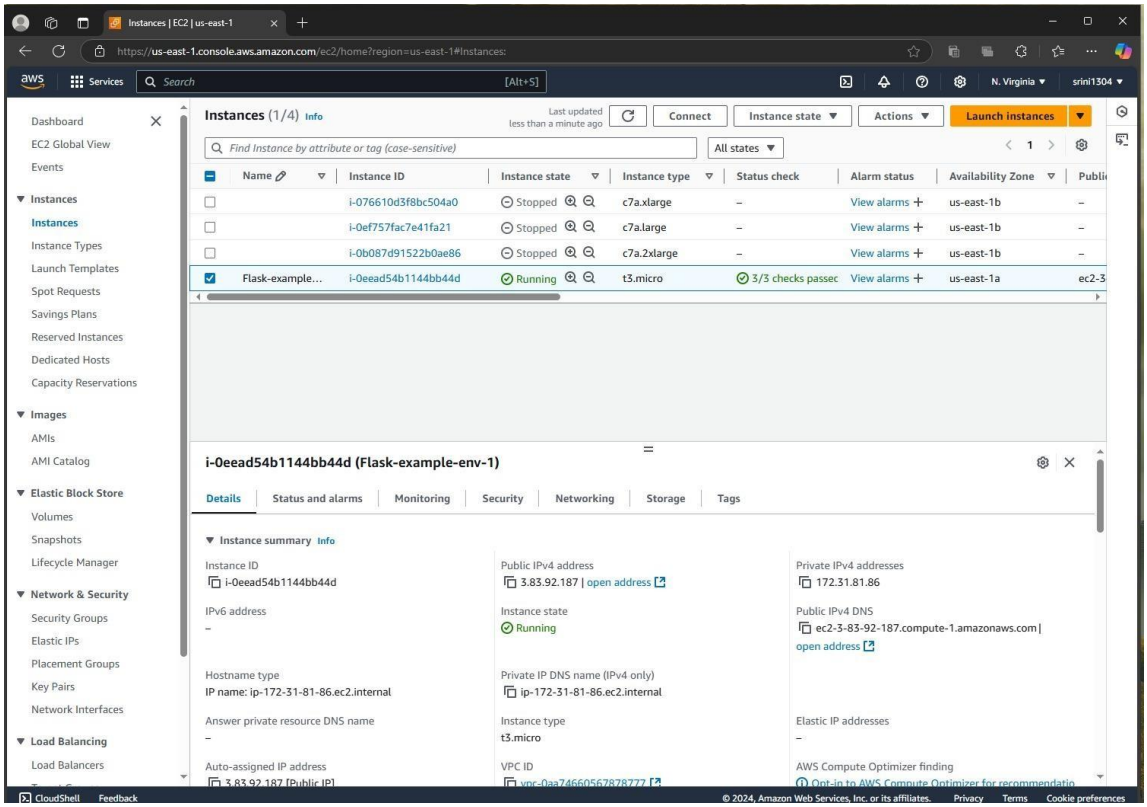
1. AWS S3



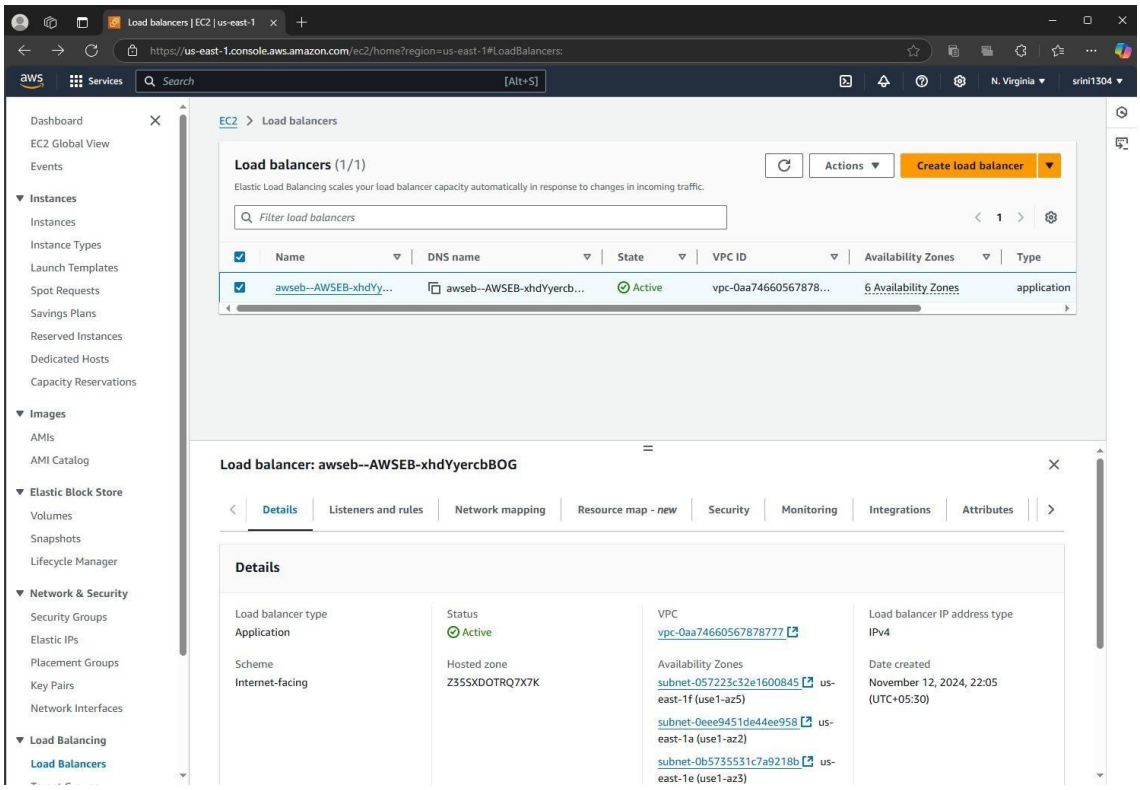
2. Elastic Beanstalk



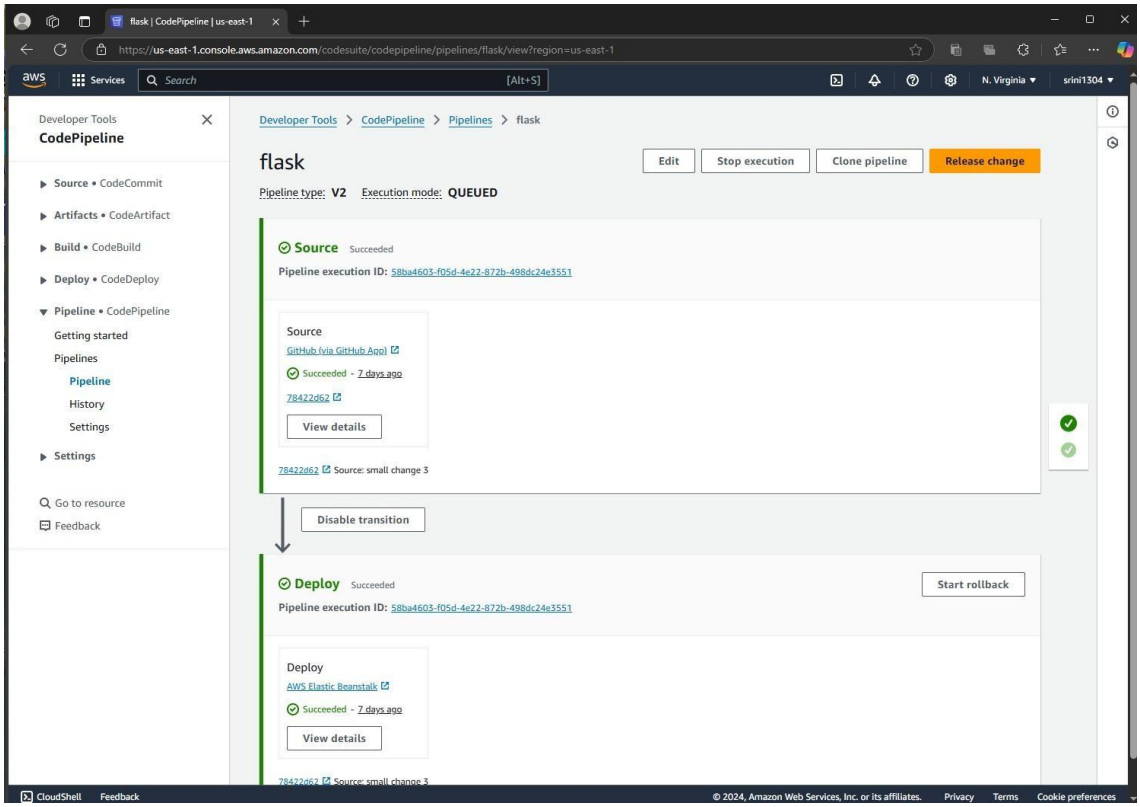
3. EC2 Instance



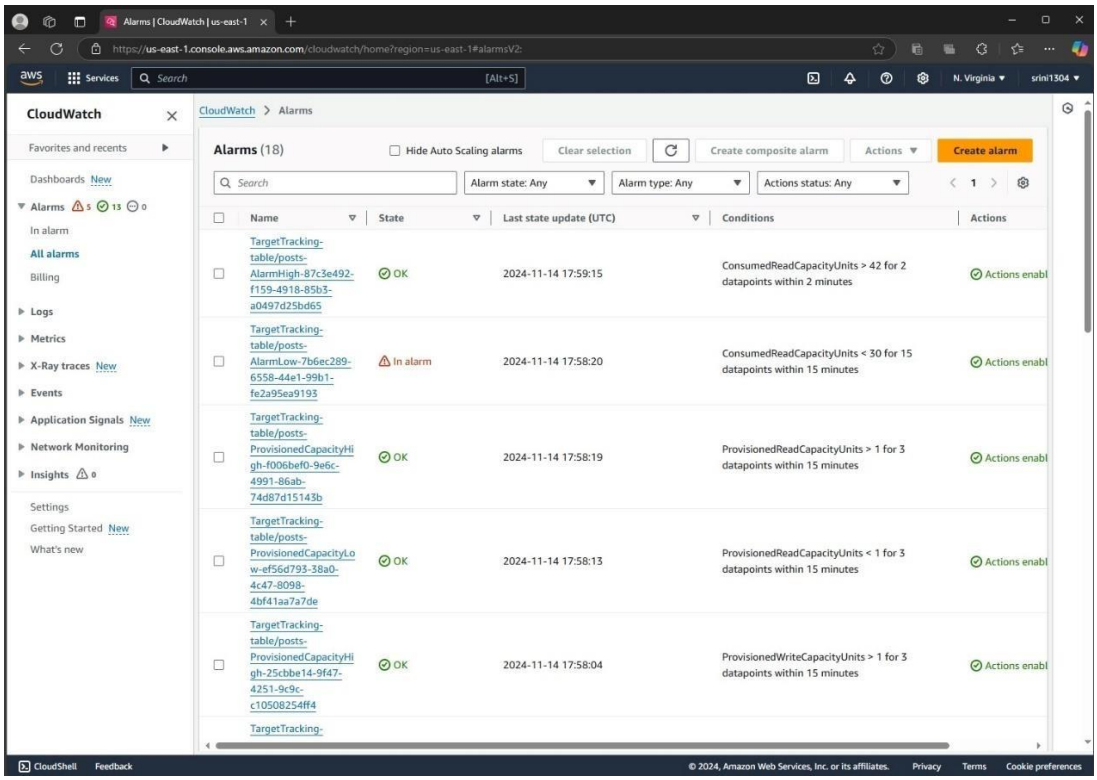
4. Load Balancer



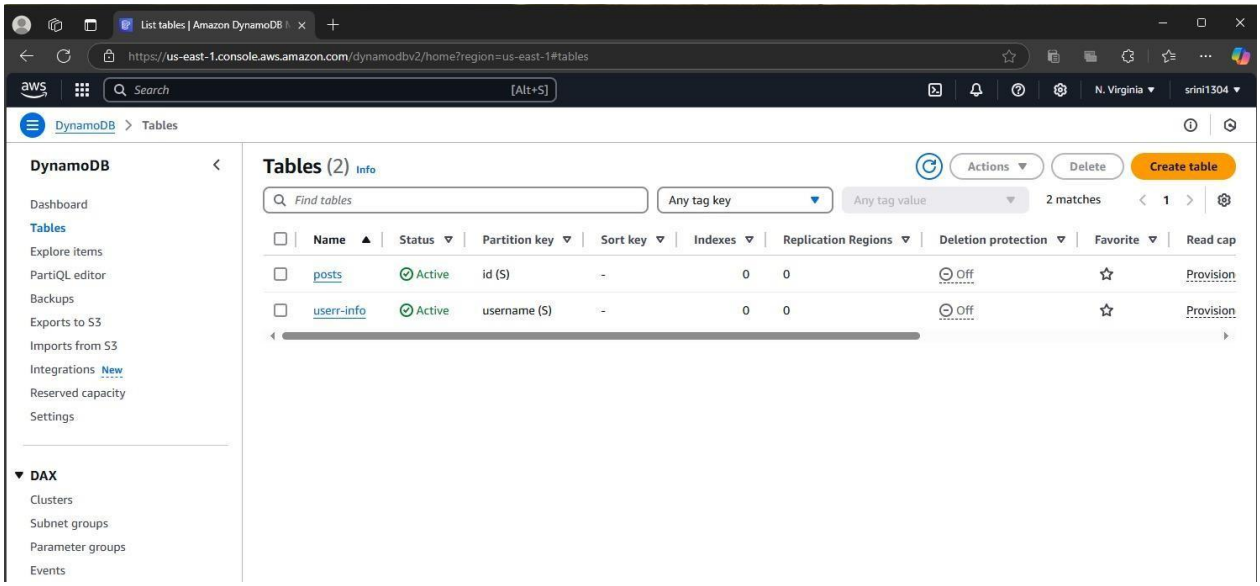
5. CI/CD Pipeline



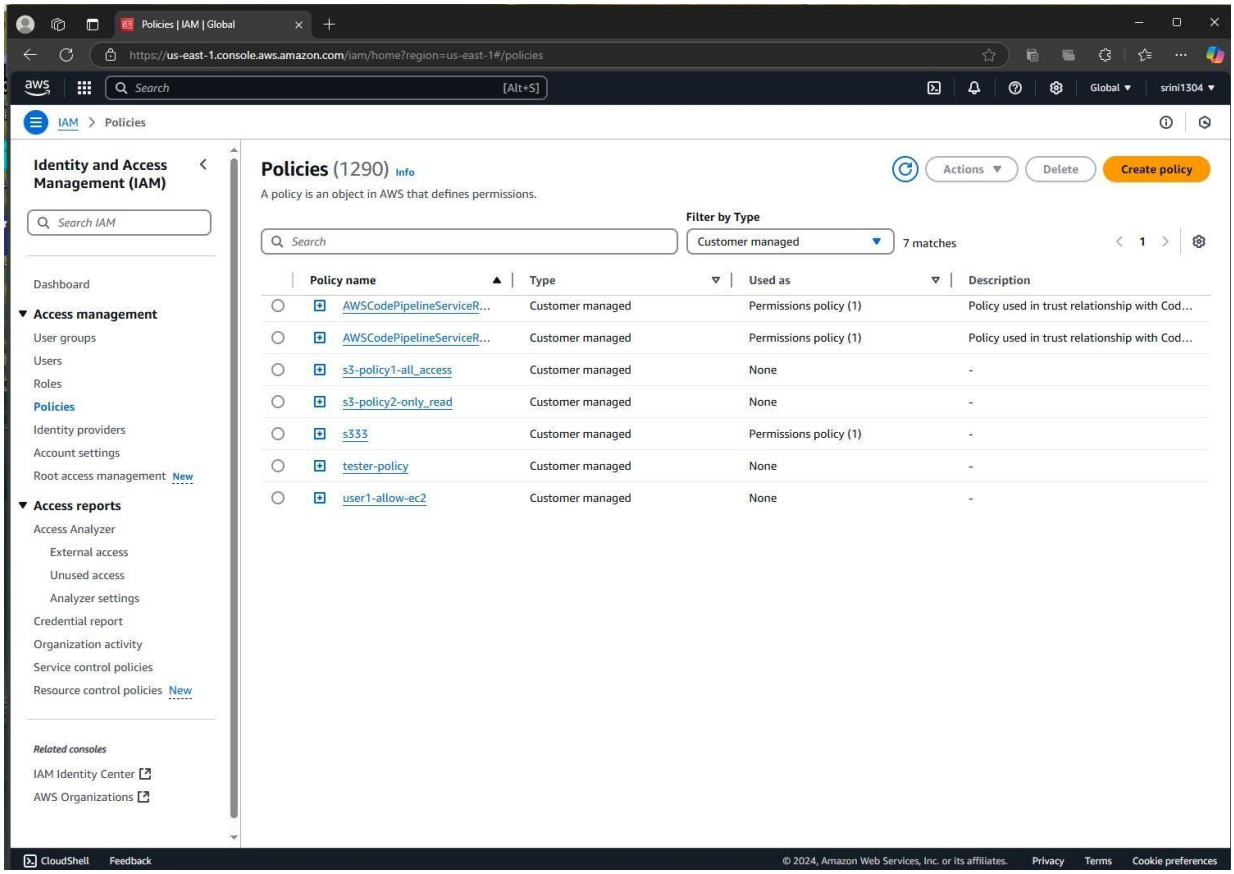
6. CloudWatch



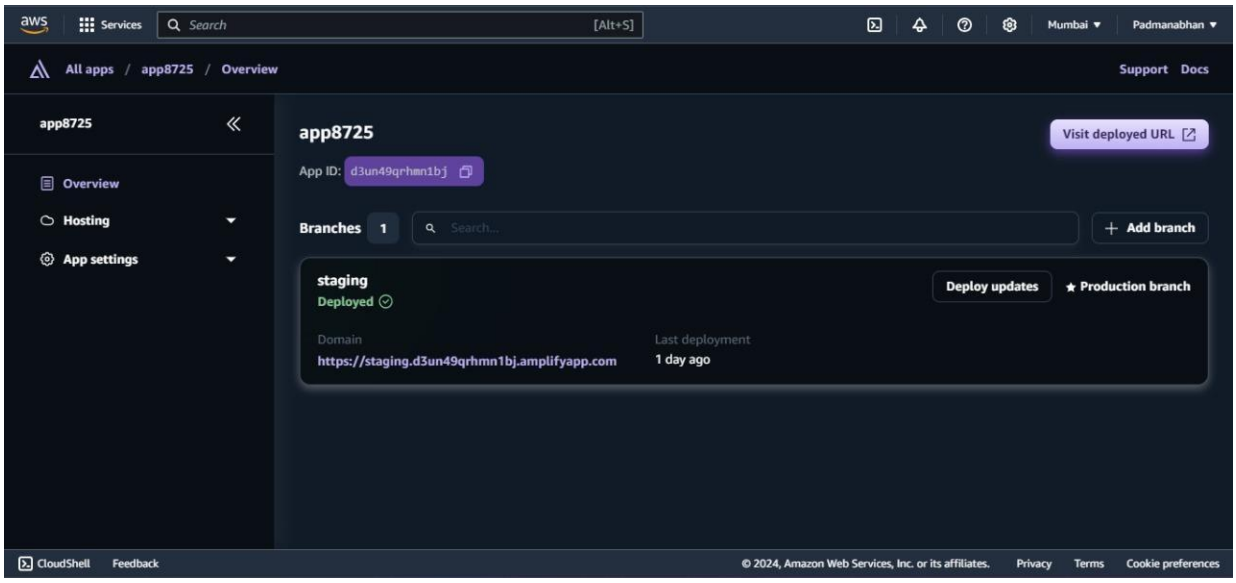
7. DynamoDB



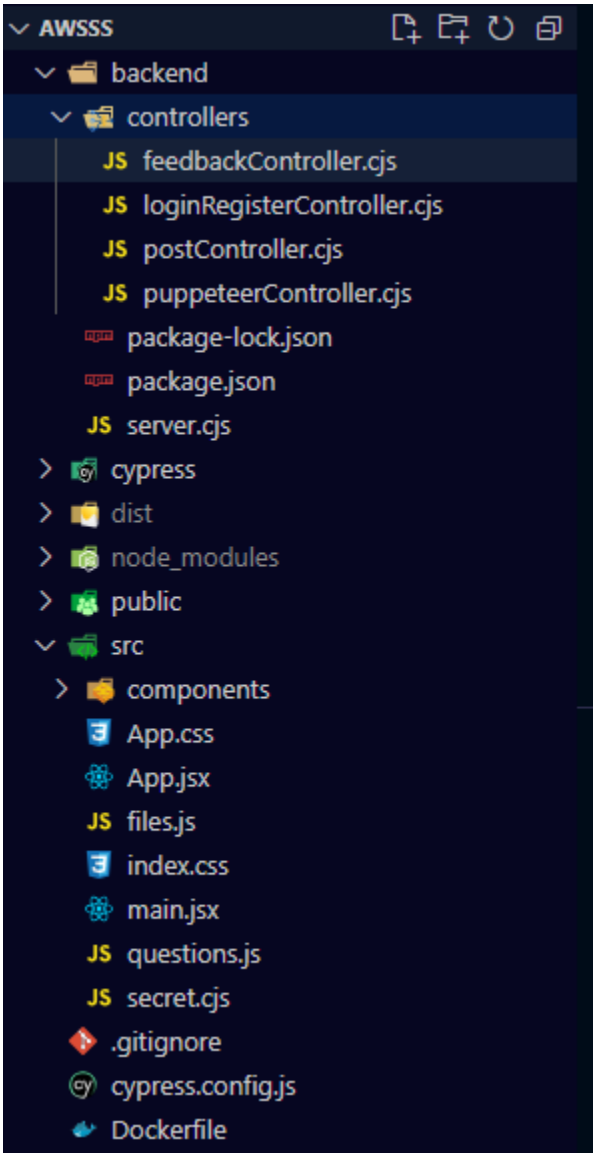
8. AWS IAM

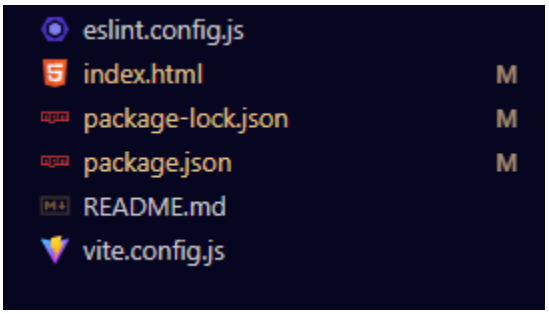


9. AWS Amplify



Codes


AWSSS
backend
controllers
feedbackController.cjs
loginRegisterController.cjs
postController.cjs
puppeteerController.cjs
package-lock.json
package.json
server.cjs
cypress
dist
node_modules
public
src
components
App.css
App.jsx
files.js
index.css
main.jsx
questions.js
secret.cjs
.gitignore
cypress.config.js
Dockerfile


eslint.config.js
index.html M
package-lock.json M
package.json M
README.md
vite.config.js

Backend Codes

feedbackController.cjs

```
const AWS = require('aws-sdk');
const jwt=require('jsonwebtoken');
const { Readable } = require('stream');

// Initialize AWS SDK with hardcoded credentials
AWS.config.update({
 accessKeyId: '',
 secretAccessKey: '',
 region: 'us-east-1' // Example: 'us-west-2'
});

// Initialize S3 client
const s3 = new AWS.S3();

// Define the S3 bucket and file key (CSV file name)
const BUCKET_NAME = 'awsrecom';
const CSV_FILE_KEY = 'dataset.csv';

// Function to append data to CSV in S3
const appendDataToCsv = async (user_id,test_id, resources_used, scores, ratings) => {
 try {
  console.log(test_id,resources_used,scores);
  // Get the existing CSV file from S3
  const response = await s3.getObject({ Bucket: BUCKET_NAME, Key: CSV_FILE_KEY
}).promise();
  const csvContent = response.Body.toString('utf-8');
```

```
    // Convert arrays to comma-separated strings for single-cell entries
    const resourcesUsedString = Array.isArray(resources_used) ? resources_used.join(';') :
resources_used;
    const ratingsString = Array.isArray(ratings) ? ratings.join(';') : ratings;

    // Append new data to the CSV content
    const newRow =
`\n${user_id},${test_id},"${resourcesUsedString}","${scores}","${ratingsString}"`;
    const updatedCsvContent = csvContent + newRow;

    // Create a stream from the updated CSV content
    const updatedCsvStream = new Readable();
    updatedCsvStream.push(updatedCsvContent);
    updatedCsvStream.push(null); // End of stream

    // Upload the updated CSV file back to S3
    await s3
     .upload({
      Bucket: BUCKET_NAME,
      Key: CSV_FILE_KEY,
      Body: updatedCsvStream,
      ContentType: 'text/csv',
     })
     .promise();

    console.log('Data appended to CSV successfully.');
    return { status: 'success', message: 'Data appended to CSV in S3' };

  } catch (error) {
```

```
    console.error('Error appending data to CSV:', error);
    return { status: 'error', message: error.message };
  }
};

// Function to handle feedback submission
const submitFeedback = async (req, res) => {
  const { user_id,test_id, resources_used, scores, ratings } = req.body;

  try {
    // Call the function to append data to S3 CSV
    const result = await appendDataToCsv(user_id,test_id, resources_used, scores, ratings);
    res.json(result);

  } catch (error) {
    console.error('Error submitting feedback:', error);
    res.status(500).json({ status: 'error', message: error.message });
  }
};

module.exports = { submitFeedback };
```

loginRegisterController.cjs

```
const AWS = require('aws-sdk');
const jwt = require('jsonwebtoken');

// Initialize AWS SDK
AWS.config.update({
accessKeyId: '',
```

```
 secretAccessKey: '',
 region: 'us-east-1',
});

const dynamoDB = new AWS.DynamoDB.DocumentClient();
const TABLE_NAME = 'userr-info'; // DynamoDB table name

// Function to handle user registration
const registerUser = async (req, res) => {
 const { username, email, password } = req.body;

 try {
  // Check if the user already exists based on username
  const params = {
   TableName: TABLE_NAME,
   Key: { username }, // Using username as the unique identifier
  };

  const existingUser = await dynamoDB.get(params).promise();

  if (existingUser.Item) {
   return res.status(400).json({ message: 'User already exists' });
  }

  // Store user data in DynamoDB
  const putParams = {
   TableName: TABLE_NAME,
   Item: {
    username, // Partition key
```

```
      email,  // Secondary attribute
      password, // Store plain text password
    },
  };

  await dynamoDB.put(putParams).promise();

  res.status(201).json({ message: 'User registered successfully' });
 } catch (error) {
  console.error('Error registering user:', error);
  res.status(500).json({ message: 'Error registering user' });
 }
};

// Function to handle user login
const loginUser = async (req, res) => {
 const { username, password } = req.body;

 try {
  // Fetch user from DynamoDB based on username
  const params = {
   TableName: TABLE_NAME,
   Key: { username }, // Query by username (partition key)
  };

  const user = await dynamoDB.get(params).promise();

  if (!user.Item) {
   return res.status(400).json({ message: 'User not found' });
```

```
  }

  // Compare the provided password with the stored password (plain text comparison)
  if (password !== user.Item.password) {
   return res.status(400).json({ message: 'Invalid password' });
  }

  // Generate a JWT token for authentication
  const token = jwt.sign({ username }, '123', {
  expiresIn: '1h',
  });
  console.log('Generated Token:',token);
  res.status(200).json({ message: 'Login successful', token });
 } catch (error) {
  console.error('Error logging in user:', error);
  res.status(500).json({ message: 'Error logging in user' });
 }
};

module.exports = { registerUser, loginUser };
```

postController.cjs

```
const AWS = require('aws-sdk');
const jwt = require('jsonwebtoken');
const { v4: uuidv4 } = require('uuid'); // Import the uuid library

AWS.config.update({
```

```
  accessKeyId: '',
  secretAccessKey: '',
  region: 'us-east-1',
});

// DynamoDB Client
const dynamoDB = new AWS.DynamoDB.DocumentClient();
const postsTable = 'posts'; // Your DynamoDB table name

// Middleware to verify JWT token
const verifyToken = (req, res, next) => {
  console.log('Authorization Header:', req.headers['authorization']);

  const token = req.headers['authorization'];
  if (!token) {
    return res.status(403).send('Token is required');
  }

  console.log('Token Received:', token);

  // Remove "Bearer " prefix from the token
  var newToken = token.substring(7); // Removing "Bearer " (7 characters) from the token string

  // Decode the token to check its contents (useful for debugging)
  var decoded = jwt.decode(newToken, { complete: true });
  console.log('Decoded Token:', decoded);

  // Verify the token using the secret key
  jwt.verify(newToken, '123', (err, decoded) => {
```

```
    if (err) {
      return res.status(403).send('Invalid token');
    }
    req.username = decoded.username; // Add username to the request object
    next();
  });
};

// Route to post a message
const postMessage = async (req, res) => {
  const { message } = req.body;
  const username = req.username;

  const params = {
    TableName: postsTable,
    Item: {
      id: uuidv4(),        // Generate a unique ID for each post
      username: username,
      message: message,
      createdAt: new Date().toISOString(),
    },
  };

  try {
    await dynamoDB.put(params).promise();
    res.status(201).send('Post created successfully');
  } catch (error) {
    console.error('Error posting message:', error);
    res.status(500).send('Error posting message');
```

```
 }
};

// Route to get all posts
const getPosts = async (req, res) => {
 const params = {
  TableName: postsTable,
 };

 try {
  const result = await dynamoDB.scan(params).promise();
  res.status(200).json(result.Items);
 } catch (error) {
  console.error('Error fetching posts:', error);
  res.status(500).send('Error fetching posts');
 }
};

// Exporting the functions and middleware
module.exports = { verifyToken, postMessage, getPosts };
```

puppeteerController.cjs

```
// puppeteerController.js
const puppeteer = require('puppeteer');

const runTest = async (req, res) => {
 const { html, css, js, validationScript } = req.body;

 const fullHtml = `
```

```
  <!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>${css}</style>
  </head>
  <body>
    ${html}
    <script>${js}</script>
  </body>
  </html>
`;

try {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.setContent(fullHtml);

  // Convert validation script from string back to function
  const validate = eval(`(${validationScript})`);

  // Run the validation script
  const message = await validate(page);
  res.json({ message });

  await browser.close();
} catch (error) {
  res.status(500).json({ message: 'Error running test', error: error.message });
```

```
 }
};

module.exports = { runTest };
```

server.cjs

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

// Import controller functions
const { registerUser, loginUser } = require('./controllers/loginRegisterController.cjs');
const { verifyToken, postMessage, getPosts } = require('./controllers/postController.cjs');
const { runTest } = require('./controllers/puppeteerController.cjs');
const { submitFeedback } = require('./controllers/feedbackController.cjs')
const app = express();
const port = 5000;

// Middleware
app.use(cors());

app.use(bodyParser.json());

// Routes for registration and login
app.post('/register', registerUser);
app.post('/login', loginUser);

// Routes for posting and getting posts (Forum)
app.post('/post', verifyToken, postMessage); // Posting a message
```

```
app.get('/posts', getPosts); // Getting all posts

// Route for running the Puppeteer test (Task page in second project)
app.post('/run-test', runTest);
app.post('/submit-feedback', submitFeedback);
// Start server
app.listen(port, () => {
 console.log(`Server running on http://localhost:${port}`);
});
```

Frontend code

App.jsx

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import LoginRegister from './components/LoginRegister';
import Forum from './components/Forum';
import HomePage from './components/HomePage';
import ResourceReqPage from './components/ResourceReqPage';
import TestPage from './components/TestPage';
import ResultPage from './components/ResultPage';
import Home from './components/Home';
import Chlghome from './components/chlghome';
const App = () => {
 return (
  <Router>
   <Routes>
```

```
      <Route path="/" element={<LoginRegister />} />
      <Route path="/home" element={<Home />} />
      <Route path="/forum" element={<Forum />} />
      <Route path="/challenges" element={<HomePage />} />
      <Route path="/resource/:taskId" element={<ResourceReqPage />} />
      <Route path="/test/:taskId" element={<TestPage />} />
      <Route path="/results" element={<ResultPage />} />
      <Route path="/chlg" element={<Chlghome/>}/>
    </Routes>
  </Router>
 );
};

export default App;
```

files.js

```
const someJSCodeExample = `
 // The source (has been changed) is
https://github.com/facebook/react/issues/5465#issuecomment-157888325

 const CANCELATION_MESSAGE = {
  type: 'cancelation',
  msg: 'operation is manually canceled',
 };

 function makeCancelable(promise) {
  let hasCanceled_ = false;

  const wrappedPromise = new Promise((resolve, reject) => {
```

```
    promise.then(val => hasCanceled_ ? reject(CANCELATION_MESSAGE) : resolve(val));
    promise.catch(reject);
  });

  return (wrappedPromise.cancel = () => (hasCanceled_ = true), wrappedPromise);
 }

 export default makeCancelable;
`;

const someCSSCodeExample = `
 @import
url('https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0
,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900Cdisplay=swap');

 * {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  outline: none;
  -ms-overflow-style: none;
  scrollbar-width: none;
 }
 *::-webkit-scrollbar {
  display: none;
 }

 body {
  margin: 0;
```

```
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

[type=reset], [type=submit], button, html [type=button] {
    -webkit-appearance: button;
}

[type=button]{
  -webkit-appearance: none;
}

.full-width {
  width: 100%;
}
.full-height {
  height: 100%;
}
.full-size {
  width: 100%;
  height: 100%;
}

.ql-editor a {
  color: rgba(255, 255, 255, 0.20);
  cursor: pointer;
```

```
  padding-left: 8px;
  padding-right: 8px;
  text-decoration:  none;
}
.ql-editor ul, .ql-editor li, .ql-editor ol {
  margin-left: 16px;
}
.ql-editor object {
  color: #d32f2f;
}
.ql-editor blockquote {
  border-left: 3px solid rgba(255, 255, 255, 0.12);
  padding-top: 8px;
  padding-left: 24px;
  padding-right: 16px;
  padding-bottom: 8px;
}
.ql-editor .ql-align-center {
  text-align: center;
}
.ql-editor .ql-align-justify {
  text-align: justify;
}
.ql-editor .ql-align-right {
  text-align: right;
}
.ql-editor a:hover {
  text-decoration:  underline;
}
```

```
`;

const someHTMLCodeExample = `
 <!DOCTYPE html>
 <html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <!-- https://web.dev/uses-rel-preconnect -->
    <link rel="preconnect" href="https://storage.googleapis.com">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#111" />

    <meta
     name="description"
     content="Wlist"
     data-react-helmet="true"
    />
    <meta
     property="og:title"
     content="Wlist"
     data-react-helmet="true"
    >
    <meta
     property="og:description"
     content="Wlist"
     data-react-helmet="true"
    >
    <meta
```

```
      property="og:url"
      content="%PUBLIC_URL%"
      data-react-helmet="true"
    >
    <meta
      property="og:image"
      content="%PUBLIC_URL%/images/cover.png"
      data-react-helmet="true"
    />
    <meta
      name="twitter:card"
      content="summary"
      data-react-helmet="true"
    />
    <meta property="og:type" content="website" />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" crossorigin="use-credentials" />
    <!-- https://web.dev/defer-non-critical-css/ -->
    <link rel="preload"
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700Cdisplay=swap"
as="style" onload="this.onload=null;this.rel='stylesheet'">

    <title>Wlist</title>

    <!-- ie -->
```

```
<script type="text/javascript">
  var ua = navigator.userAgent;
  var is_ie = ua.indexOf('MSIE ') > -1 || ua.indexOf('Trident/') > -1;

  if (is_ie) {
    document.ie = 'true';

    var ie_script = document.createElement('script');
    var ie_styles = document.createElement('link');

    ie_script.src = 'no-ie/init.js';
    ie_styles.rel = 'stylesheet';
    ie_styles.href = 'no-ie/styles.css';

    function injectScripts() {
      document.body.innerHTML = '';
      document.body.appendChild(ie_styles);
      document.body.appendChild(ie_script);
    }

    if (document.addEventListener) {
      document.addEventListener('DOMContentLoaded', injectScripts);
    } else { // before IE 9
      document.attachEvent('DOMContentLoaded', injectScripts);
    }

  }
 </script>
</head>
```

```
  <body>
   <noscript>You need to enable JavaScript to run this app.</noscript>
   <script type="text/javascript">
    // set the body color before app initialization, to avoid blinking
    var themeMode = localStorage.getItem('theme-mode');
    var initialBodyStyles = document.createElement('style');
    var currentThemeColor = themeMode === 'light' ? '#fafafa': '#111';
    initialBodyStyles.innerText = 'body { background-color: ' + currentThemeColor + ' }';
    document.head.appendChild(initialBodyStyles);

    // also set meta[name="theme-color"] content
    var metaTheme = document.querySelector('meta[name="theme-color"]');

    metaTheme.content = currentThemeColor;
   </script>
   <div id="root"></div>
  </body>
 </html>
`;

const files = {
 "script.js": {
  name: "script.js",
  language: "javascript",
  value: someJSCodeExample
 },
 "style.css": {
  name: "style.css",
  language: "css",
```

```
  value: someCSSCodeExample
 },
 "index.html": {
  name: "index.html",
  language: "html",
  value: someHTMLCodeExample
 }
};

export default files;
```

questions.js

```
export const questions = [
 {
  id: 1,
  problem: "Create a login form with 'username' and 'password' fields and a submit button.",
  validationScript: async (page) => {
   const usernameExists = await page.$('#username') !== null;
   const passwordExists = await page.$('#password') !== null;
   const buttonExists = await page.$('#submit') !== null;

   let score = 0;

   if (usernameExists) score += 25;
   if (passwordExists) score += 25;
   if (buttonExists) score += 50;

   if (score === 100) {
    await page.type('#username', 'testuser');
```

```
      await page.type('#password', 'password123');
      await page.click('#submit');
      await page.waitForNavigation({ waitUntil: 'networkidle0' });
      return `Test Passed: Form submitted successfully! You scored ${score} points.`;
    } else {
      return `Test Failed: Missing form elements. You scored ${score} points.`;
    }
  },
 },
 {
  id: 2,
  problem: "Create a To-Do list where the user can add tasks. Each task should have a checkbox to mark it as completed.",
  validationScript: async (page) => {
    const inputExists = await page.$('#task-input') !== null;
    const addButtonExists = await page.$('#add-task') !== null;
    let score = 0;

    if (inputExists) score += 25;
    if (addButtonExists) score += 25;

    const taskExists = await page.evaluate(() => {
      const task = document.querySelector('.task');
      return task CC task.textContent.includes('Buy groceries');
    });

    if (taskExists) score += 50;

    return `Test Completed: You scored ${score} points.`;
```

```
  },
 }
];
```

HomePage.jsx

```
import { Link } from 'react-router-dom';
import React from 'react';
import './hompg.css';

function HomePage() {
 return (
  <div class="bdd">
 <div class="whole">
 <div class="container">
  <div class="content">
   <h1>
   Learn HTML Camp; CSS
   </h1>
   <div class="details">
   1 hours of content • 3 lessons
   </div>
   <div class="progress-bar">
    <div class="progress-text">
    Progress: 0/2
   </div>
   <div class="progress">
    <div  class="progress-fill">
    </div>
   </div>Cnbsp;
```

```
    <div class="progress-percentage">
     0%
    </div>
   </div>
   <div class="section">
    <h2>
     Project
    </h2>
    <div class="project">
     <img class="img1" alt="A person working on a circuit board with a laptop and tools on the table" width="680px" height="750px" src="https://storage.googleapis.com/a1aa/image/M4wxrU5exOxvISMBdcQnnKhxh2BdTVmYR0Ic2xY28xkIH94JA.jpg" />
     <div class="description">
      <h3>
       Build a personal portfolio page
      </h3>
      <p>
       Use your new HTML and CSS skills to create a simple personal portfolio page. This project will give you a chance to practice the basics of HTML and CSS.
      </p>
      <button class="start-button">
      Start ...
      </button>
     </div>

    </div>
   </div>
   <div class="section">
    <h2>
```

Exercises

</h2>

<h3>

Create a list of your favorite things

</h3>

Practice creating lists in HTML by making a list of your favorite things. You can use any tag you like, but try to use the right tag for the job!

<*Link* to="/resource/1">

<button class="start-button">

Start exercise

</button></*Link*>

<h3>

Style your favorite things list

</h3>

Now that you've created a list of your favorite things, use CSS to style it! You can change the font, size, color, and more. This is your chance to get creative!

```
      </p>
     </div>
     <Link to="/resource/2">
     <button class="start-button">
      Start exercise
     </button>
     </Link>
    </div>
   </div>
  </div>
 </div>
 </div>
 );
}

export default HomePage;
```

ResourceReqPage.jsx

```
import React, { useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import './ResourceReqPage.css'; // Import updated CSS for styling

function ResourceReqPage() {
 const { taskId } = useParams();
 const navigate = useNavigate();
 const [links, setLinks] = useState([]);
 const [ratings, setRatings] = useState([]);
 const [inputLink, setInputLink] = useState('');
 const [inputRating, setInputRating] = useState('');

 // Handle adding a new link and its rating
 const addLink = () => {
  if (inputLink.trim() !== '' CC inputRating !== '') {
   setLinks([...links, inputLink.trim()]);
   setRatings([...ratings, parseInt(inputRating)]);
   setInputLink(''); // Clear input fields
   setInputRating('');
  }
 };

 // Handle submission to save links and ratings, then navigate to TestPage
 const handleSubmit = () => {
  localStorage.setItem('studyMaterials', JSON.stringify(links));
  localStorage.setItem('ratings', JSON.stringify(ratings));
```

```
 navigate(`/test/${taskId}`);
};

return (
 <div class="card">
  <div class="card-info">
 <div className="overlay">
  <div className="popup">
   <h2>Optional: Add Study Material Links and Ratings</h2>
   <input
    type="text"
    placeholder="Enter link"
    value={inputLink}
    onChange={(e) => setInputLink(e.target.value)}
   />
   <input
    type="number"
    placeholder="Enter rating"
    value={inputRating}
    onChange={(e) => setInputRating(e.target.value)}
   />
   <button className="add-button" onClick={addLink}>
    Add Link and Rating
   </button>
   <button className="submit-button" onClick={handleSubmit}>
    Submit and Start Test
   </button>
  </div>
 </div>
```

```
  </div>
  </div>
 );
}

export default ResourceReqPage;
```

ResultPage.jsx

```
import React from 'react';
import { useLocation } from 'react-router-dom';
import './ResultPage.css';

function ResultPage() {
 const location = useLocation();
 const { recommendedData, scores } = location.state;

 return (
  <div className="result-container">
   <div className="content-wrapper">


    <h1 className="title">Test Results</h1>


    <p className="score-text">
```

```
        You scored {scores}% on the test. Here are some resources to help you improve.
      </p>

      <h2 className="resources-title">Recommended Resources</h2>

      <div className="resource-list">
       {recommendedData.map(([resource, rating], index) => (
        <div key={index} className="resource-item">
         <div className="resource-image" />

         <div className="resource-details">
          <h3 className="resource-title">{resource}</h3>
          <div className="resource-meta">
           Rating: {rating}
          </div>
         </div>
        </div>
       ))}
      </div>
     </div>
    </div>
  );
}

export default ResultPage;
```

TestPage.jsx

```
import React, { useState, useEffect } from 'react';
import Editor from '@monaco-editor/react';
import axios from 'axios';
import Question from './Question';
import { questions } from '../questions';
import { useNavigate,useParams } from 'react-router-dom';
import { getuser } from '../secret.cjs';
import cors from 'cors';
import './smthn.css';

function TestPage() {
 const { taskId } = useParams();
 const navigate = useNavigate();
 const [files, setFiles] = useState({
 'script.js': {
    name: 'script.js',
    language: 'javascript',
    value: `console.log('Hello from script.js');\nfunction press() { console.log('Button pressed'); }`,
  },
  'style.css': {
    name: 'style.css',
    language: 'css',
    value: `body { font-family: Arial, sans-serif; color: black;background-color: white }`,
  },
  'index.html': {
    name: 'index.html',
    language: 'html',
    value: `<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link rel="stylesheet" href="style.css">
</head>
<body>
 <h1>Login Form</h1>
 <button onclick="press()">Press here</button>
 <script src="script.js"></script>
</body>
</html>`,
  },
 });
 const [fileName, setFileName] = useState('index.html');
 const [html, setHtml] = useState(files['index.html'].value);
 const [css, setCss] = useState(files['style.css'].value);
 const [js, setJs] = useState(files['script.js'].value);
 const [score, setScore] = useState(null);

 const questionIndex = parseInt(taskId, 10) - 1;
 const selectedQuestion = questions[questionIndex];

 const handleEditorChange = (value) => {
  setFiles((prevFiles) => ({
   ...prevFiles,
   [fileName]: {
    ...prevFiles[fileName],
    value,
   },
```

```
  }));
  if (fileName === 'index.html') setHtml(value);
  if (fileName === 'style.css') setCss(value);
  if (fileName === 'script.js') setJs(value);
 };

 const completeTest = async () => {
  try {
   const user_id = getuser();
   const resourcesUsed = JSON.parse(localStorage.getItem('studyMaterials')) || [];
   const ratings = JSON.parse(localStorage.getItem('ratings')) || [];

   // Send test data
   const testResponse = await axios.post('http://localhost:5000/run-test', {
    html,
    css,
    js,
    validationScript: selectedQuestion.validationScript.toString(),
   });

   const testMessage = testResponse.data.message;
   const scoreMatch = testMessage.match(/(\d+)\s+points/);
   const scores = scoreMatch ? parseInt(scoreMatch[1], 10) : 0;
   setScore(scores);

   // Send feedback
   await axios.post('http://localhost:5000/submit-feedback', {
    user_id,
    test_id: taskId,
```

```
      resources_used: resourcesUsed,
      scores,
      ratings,
    });

    // Fetch recommendations
    const url = `http://flask-example-env-1.eba-3k7ftnxz.us-east-
1.elasticbeanstalk.com/?test_id=101`;
    const fetchResponse = await fetch(url);
    if (!fetchResponse.ok) throw new Error('Network response was not ok');
    const recommendedData = await fetchResponse.json();

    // Navigate to the result page with data
    console.log(recommendedData,scores);
    navigate('/results', { state: { recommendedData, scores } });
  } catch (error) {
    console.error('Error:', error);
  }
 };

 const updatePreview = () => {
  const previewDiv = document.getElementById('preview-div');
  previewDiv.innerHTML = '';
  const iframe = document.createElement('iframe');
  iframe.style.width = '100%';
  iframe.style.height = '100%';
  iframe.style.border = 'none';
  previewDiv.appendChild(iframe);
```

```
  const iframeDocument = iframe.contentDocument || iframe.contentWindow.document;
  iframeDocument.open();
  iframeDocument.write(`
   <!DOCTYPE html>
   <html lang="en">
   <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>${css}</style>
   </head>
   <body>
    ${html}
    <script>${js}</script>
   </body>
   </html>`
  );
  iframeDocument.close();
 };
 useEffect(() => {
  updatePreview();
 }, [html, css, js]);
 return (
  <div class="bddd">
  <div style={{ display: 'flex', width: '1500px' }}>
   <div style={{ width: '200px', padding: '10px' }}>
    <button disabled={fileName === 'index.html'} onClick={() => setFileName('index.html')}>
     index.html
    </button>
    <button disabled={fileName === 'style.css'} onClick={() => setFileName('style.css')}>
```

```
        style.css
       </button>
       <button disabled={fileName === 'script.js'} onClick={() => setFileName('script.js')}>
        script.js
       </button>
      </div>
      <div style={{ flex: 1 }}>
       <Question problem={selectedQuestion.problem} class="dd"/>
       <Editor
        height="70vh"
        theme="vs-dark"
        language={files[fileName].language}
        value={files[fileName].value}
        onChange={handleEditorChange}
       />
      </div>
      <div id="preview-div" style={{ flex: 1, border: '1px solid #ccc', marginLeft: '10px', height: '80vh',
overflow: 'auto' }} />
      <br/>
      <div style={{marginTop:'750px',marginRight:'100px'}}>
       <button onClick={completeTest}>Complete Test</button>
      </div>
      {score !== null CC <div>Score: {score}</div>}
     </div>
    </div>
   );
}

export default TestPage;
```

References

- https://www.ijfans.org/uploads/paper/f6838e1e98c7748d6f5c492888c6e2f5.pdf
- https://link.springer.com/chapter/10.1007/978-981-97-1329-5_28
- https://ieeexplore.ieee.org/document/10497289
- https://ieeexplore.ieee.org/abstract/document/10426375
- https://aws.amazon.com/blogs/devops/automated-code-review-on-pull-requests-using-aws-codecommit-and-aws-codebuild/
- https://arxiv.org/abs/2307.08705
- https://link.springer.com/chapter/10.1007/978-3-030-60036-5_3
- https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-019-0134-y

AWS Services Documentation

- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
- https://docs.aws.amazon.com/cloudwatch/
- https://docs.aws.amazon.com/elasticbeanstalk/
- https://docs.aws.amazon.com/dynamodb/
- https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/
- https://docs.aws.amazon.com/s3/
- https://docs.amplify.aws/
- https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html