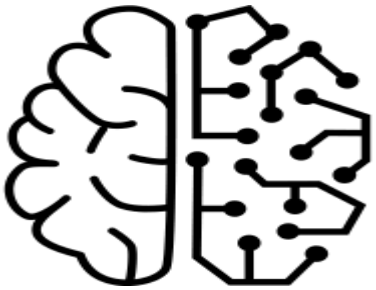


Code Editor & Similarity Scoring System Website

Functional Specification

Sam Cullen

Table of Contents

Brief Description

This documentation outlines the functional requirements for the development of a web-based platform for automated coding exam-correcting

Technologies

Front-End Technologies

- HTML
- CSS
- JavaScript
- CodeMirror / Manoco Edition
- Node / Express
- API Integration

Back-End Technologies

- Java
- Python
- PHP
- Firebase Authentication
- Netlify
- GitHub
- MongoDB Compass

Project Overview

CodeChallenges is a user-friendly platform that aims to provide a platform for automated coding exam correcting. With CodeChallenges, you can log in and register an account and take various coding exams and where you can compile and send to be automatically compared to a sample output, where the user will be given a score back based on similarity; enhancing the quality of coding assessments and promote effective learning and evaluation.

Key Features

User Registration and Authentication

Users can create accounts, log in securely, and access their profiles, ensuring data privacy and security

Code Submission

Users can submit their code snippets through the web interface, with support from multiple programming languages

Scoring and feedback

Users receive similarity scores and feedback highlighting where they went right or wrong in their results

User Dashboard

Users have access to the ability to review their coding submissions and previous score

Intended Audience

The website CodeChallenges serves a diverse audience, catering to various educational, professional, and personal use cases. Its versatile features make it a valuable tool for various user groups:

Students and Learners: Coding Bootcamp students, programmers and learners looking to improve their coding skills can use CodeChallenges to practise, submit their code for evaluation and receive feedback on their coding work.

Technical Recruiters and Employers: Technical recruiters and employers seeking to evaluate the code and abilities of job applicants can rely on CodeChallenges to effectively assess coding skills during technical interviews it streamlines the tactical hiring process and provides A structured approach for evaluating candidates

Self-Learners: Individual developers and coding enthusiasts can use CodeChallenges for self-assessment practice and skill enhancement it's a convenient tool for those who wish to refine their coding abilities

Overall CodeChallenges's broad appeal stems from its user-friendly interface efficient code comparison and detailed feedback capabilities it empowers users to enhance their coding skills streamline assessments and harness the benefits of technology for code and evaluations making it an invaluable tool for a diverse range of users

Functional Requirements

Use case 1: User Registration

- **User Interaction:** User creates an account
- **System Response:** Account is created successfully

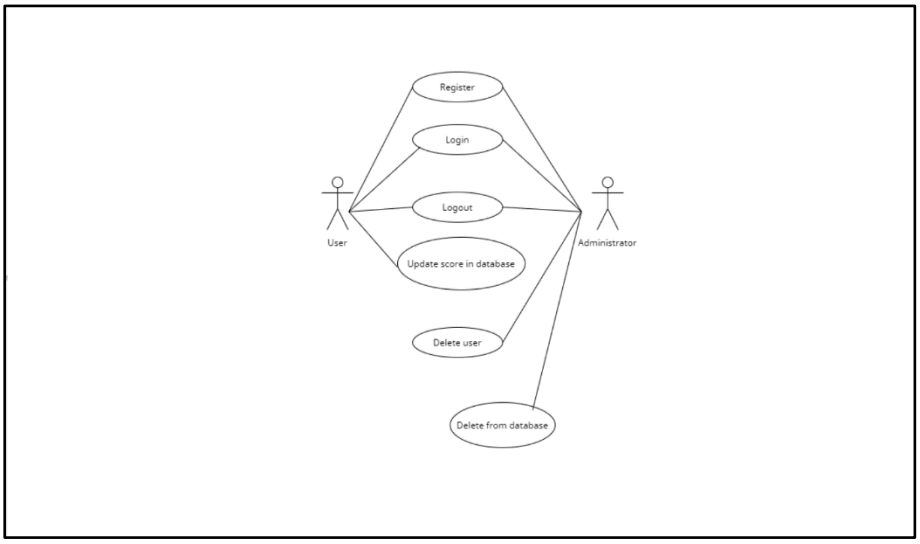


Figure 0.1 Diagram for login/signup

Use case 2: Code Submission

- **User Interaction:** User writes and submits code
- **System Response:** Outputs result, increment user score in the database if the output is correct

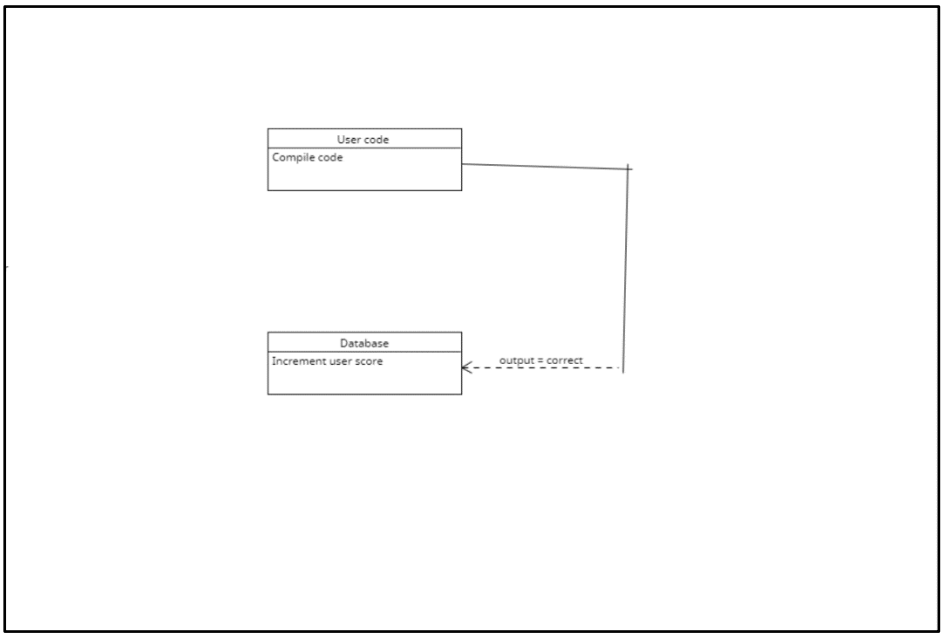


Figure 0.2 Diagram for code submission

Functionalities

- User registration and login
- Take code challenges with various languages
- Code submission for multiple programming languages
- Automatic compiler for instant result
- Update user database score If the output is correct to the corresponding challenge
- User leaderboard for reviewing scores

External APIs

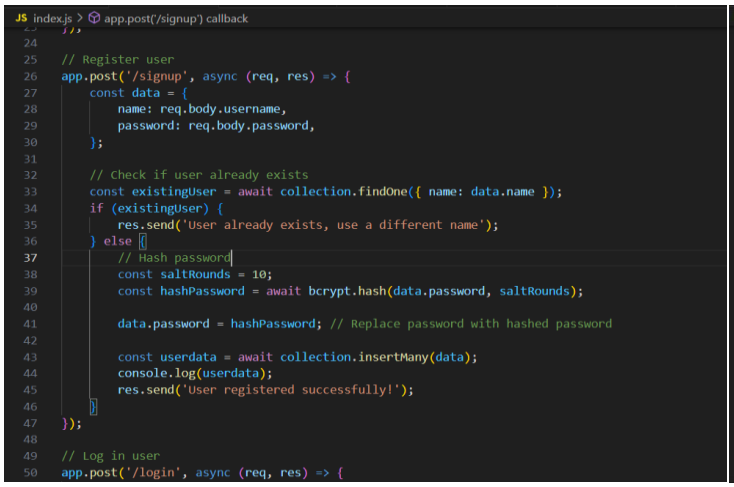
The Express.js application is used as the backend API for code compilation and execution

Integration Points

Express handles integration points by redefining routes and endpoints that communicate between frontend and backend components, in this case, the /compile endpoint acts as an integration point for code compilation. By receiving requests with the code, the input and language information, the Express application integrates with the 'compilex' library to compile and display the result of the code

Security

To enhance security for our users, especially when dealing with passwords, it is crucial to use secure password hashing and salt techniques. In the provided Express.js code below, the 'bcrypt' library is used to achieve this.



```
JS index.js > app.post('/signup') callback
24
25  // Register user
26  app.post('/signup', async (req, res) => {
27      const data = {
28          name: req.body.username,
29          password: req.body.password,
30      };
31
32      // Check if user already exists
33      const existingUser = await collection.findOne({ name: data.name });
34      if (existingUser) {
35          res.send('User already exists, use a different name');
36      } else {
37          // Hash password
38          const saltRounds = 10;
39          const hashPassword = await bcrypt.hash(data.password, saltRounds);
40
41          data.password = hashPassword; // Replace password with hashed password
42
43          const userdata = await collection.insertMany(data);
44          console.log(userdata);
45          res.send('User registered successfully!');
46      }
47  });
48
49  // Log in user
50  app.post('/login', async (req, res) => {
```

```
JS index.js > app.post('/signup') callback
45          res.send('User registered successfully!');
46      }
47  });
48
49  // Log in user
50  app.post('/login', async (req, res) => {
51      try {
52          const check = await collection.findOne({ name: req.body.username });
53          if (!check) {
54              res.send('User name not found');
55          }
56
57          // Compare hashed password with plain text
58          const isPasswordMatch = await bcrypt.compare(req.body.password, check.password);
59          if (isPasswordMatch) {
60              res.render('home');
61          } else {
62              res.send('Wrong password');
63          }
64      } catch {
65          res.send('Wrong details added');
66      }
67  });
68
69  const port = 2000;
70  app.listen(port, () => {
71      console.log(`Server running on Port: ${port}`);
```

During the user login (/login) the stored hash password is retrieved from the database based on the username provided. The user's password is then compared with the stored hashed password using 'bycrypt. compare'. If the passwords match, the login is successful

By incorporating 'bycrypt' for password hashing and salting, it significantly improves and security of the website user's credentials, protecting sensitive information in case of a data breach.

Challenges and Solutions

During the development of this website, several challenges were encountered, each requiring thoughtful solutions to ensure the project's success. This section outlines the challenges faced with developing the website, from technical, integration, scope-related and resource problems faced, along with the strategies employed to overcome them.

Technical Challenges

Design of the website

- **Challenges:** Defining the visual aesthetics and user interface design was the first challenge faced with creating the website. Deciding on a theme, colours and other front-end features proved a challenging first step in trying to align with user expectations
- **Impact:** Delays in the decision phase affected the overall project timeline
- **Solution:** Took inspiration from various similar websites

Database Linking and Password Encryption

- **Challenge:** Establishing a secure and efficient connection between the application and the database, including implementation and robust password encryption
- **Impact:** Initial database issues affected user authentication and data security

- **Solution:** Implemented database connection pooling for improved performance, employed bcrypt for hashing and salting passwords

Code Editor Implementation

- **Challenge:** Integrating a configuring CodeMirror library onto my webpage
- **Impact:** Difficulties in achieving the desired outcome, affected the overall project timeline
- **Solution:** Reviewed CodeMirror documentation, video tutorials, reviewed community support and interactively adjusted configurations to optimise the code editor's performance

Localhost and 127.0.0.1 Discrepancy

- **Challenge:** Inconsistencies in accessing the application via 'localhost:5500' and '127.0.0.1:5500'
- **Impact:** Confusion during development – trying to link the localhost code editor with the 127.0.0.1 address of the code editor version
- **Solution:** Reached out to lecturers and supervisors – received helpful feedback. Resolved this by checking the following:
    1. Disable firewall – blocking connections to local ports.
    2. Switched off proxy to allow local connections to browser

Integration Issues

External API Integration

- **Challenge:** General difficulties in integrating APIs into a website
- **Impact:** Feature dependencies on external services slowed down development – affected project timeline
- **Solution:** Reviewing community support and research for solutions and help

Scope Issues

Altered Project Scope

- **Challenge:** Changes in project requirements and scope during the development process
- **Impact:** Disruption in the original project plan and timeline -
- **Solution:** Implemented an agile development approach, regular updates published to GitHub meant I could keep track of my work and retrace my steps if needed

Testing Complications

Testing Environment Changes

- **Challenge:** Constraints in terms of development time, personal availability and technology resources
- **Impact:** Delays in feature implementation
- **Solution:** Prioritised certain features over others, and explored alternative technologies to increase production speed.

Lessons Learned

This section provides insights into the lessons learned during the development of the project, the lessons learned can be applied to future projects and demonstrate the adaptability and commitment to continuous improvement

Reflection on Challenges

Lessons Learned: Emphasized the importance of early, detailed project planning, including user interface design, API integration, and dealing with changes throughout the project

Improvement Strategies

Improvement Strategy: Incorporate prototyping for user interface design at the beginning of the development, and have a general idea made at the beginning of the project to work with to reduce iterations and setbacks.

Conclusion

In summary, the CodeChallenges platform has been thoughtfully designed to offer a user-friendly experience for automated coding exam correction. The document outlines key features, technologies, security measures, and intended audience. Challenges in design, database linking, and scope changes were addressed with proactive solutions, and finally, Lessons learned emphasize early project planning and continuous improvement.

Collaboration Form

Student Name: Sam Cullen

Student No: C00250093

Supervisor: Joseph Kehoe

The following is a total list of all students I collaborated with whilst conducting research for this assignment:

Student Name: Sam Cullen

Student Number: C00250093

% of collaboration within submission: 100%