Pune Institute of Computer Technology Dhankawadi, Pune

A MINI PROJECT REPORT ON

Create a small application by selecting relevant system environment / plat- form and programming languages. Narrate concise Test Plan consisting features to be tested and bug taxonomy. Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. Prepare Test

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE IN THE PARTIAL FULFILLMENT OF THE LP IV

BE (COMPUTER ENGINEERING) SUBMITTED BY

Roll No.	**Name**
41471	Prachi Said

Under the Guidance of

Prof. P. J. Jambhulkar

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year 2024-25

PUNE INSTITUTE OF COMPUTER TECHNOLOGY DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the Mini Project report of LP IV (STQA) entitled

”Create a small application by selecting relevant system environment / platform and programming languages. Narrate concise Test Plan consist- ing features to be tested and bug taxonomy. Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black- box and White-box testing covering Unit and Integration test by using suitable Testing tools. Prepare Test”

Submitted by

Roll No.	**Name**
41471	Prachi Said

has satisfactorily completed a micro project report under the guidance of Prof. P. J. Jambhulkar towards the partial fulfillment of BE Computer Engineering, Academic Year 2024-25 of Savitribai Phule Pune University.

Prof. P. J. Jambhulkar
(Lab Guide)

Dr. Geetanjali Kale
(H. O. D)

Date: / / Place: Pune

Contents

Project Overview: CodeSync Application

Application Description

Name: CodeSync

Purpose: A collaborative coding platform that allows users to sync their code in real-time, share projects, and manage tasks related to coding assignments.

Platform:

- Frontend: HTML, CSS, JavaScript (React)
- Backend: Node.js with Express.js
- Database: MongoDB

Key Features:

1. Real-time code synchronization.
2. Project sharing among users.
3. Task management related to coding assignments.
4. Version control for project files.
5. User authentication and authorization.

Test Plan

Objective

To ensure that all functionalities of the CodeSync application work as intended, meet the specified requirements, and are free from defects.

Testing Scope

- **Functional Testing**: Verify that all user actions (synchronizing code, sharing projects, managing tasks, etc.) work correctly.
- **Usability Testing**: Confirm that the UI is intuitive and user-friendly.
- **Performance Testing**: Ensure that the application responds quickly to user actions.
- **Compatibility Testing**: Verify compatibility across different browsers (Chrome, Firefox, Safari).
- **Security Testing**: Check for any common vulnerabilities (e.g., SQL injection, XSS).

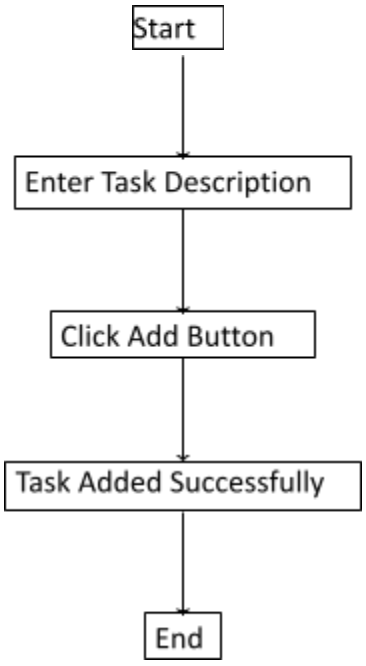
Testing Strategy

1. **Black-Box Testing**: Focuses on testing the functionality of the application without considering internal code structure.
2. **White-Box Testing**: Focuses on testing the internal structure of the applica- tion.

Bug Taxonomy

- **Critical Bugs**: Bugs that cause the application to crash or fail to function completely.
- **Major Bugs**: Bugs that affect major functionalities but do not cause crashes.
- **Minor Bugs**: Minor usability or UI issues.
- **Cosmetic Bugs**: Aesthetic issues that do not affect functionality.

Flowchart for Adding a Task



Test Cases and Procedures

Test Scenarios and Test Cases

Test ID	Test Scenario	Test Case Description	Expected Result	Status
TC-001	Room Creation	Verify that a user can create a room for collaboration	A unique room ID is generated, and the user can share it with others	
TC-002	Room Joining	Verify that multiple users can join the same room using a room ID	Users can successfully join the room and collaborate in real-time	
TC-003	Real-time Code Editing	Verify real-time code synchronization between users in a room	Code changes are instantly broadcasted and updated across all users' editors	
TC-004	File Management	Verify that users can create, edit, and delete files in the editor	Files are successfully created, edited, and deleted; changes are reflected in the file structure for all users	
TC-005	Code Execution	Verify that users can execute code and receive output in the editor	Code is executed via the Piston API, and the output is displayed without errors	
TC-006	Group Chat	Verify that users can communicate via the real-time chat feature	Messages are sent and received instantly; chat updates correctly for all users	
TC-007	User Presence	Verify that the user presence (online/offline) is updated correctly	User status is accurately displayed; notifications are sent when users join/leave	
TC-008	Code Suggestions	Verify that code autocompletion and suggestions work as expected	Autocomplete suggestions appear based on the code, and they can be selected and applied	

TC-009	Syntax Highlighting	Verify that syntax highlighting is applied correctly for various programming languages	The editor detects the language and applies syntax highlighting accordingly	

TC-010	Room Notifications	Verify that users receive notifications when someone joins or leaves a room	Notifications for user presence are displayed in real-time	
TC-011	File Synchronization	Verify that file changes are synchronized in real-time across all users in the room	Any changes to the file structure are reflected for all users instantly	
TC-012	API Error Handling	Verify that the system handles Piston API errors during code execution	Appropriate error messages are shown when API fails or returns an error	
TC-013	Code Editor Performance	Test the performance of the code editor under load	The editor should remain responsive and synchronize changes under heavy usage	
TC-014	Security Testing	Test for security vulnerabilities (e.g., XSS, unauthorized access)	Application should not allow any unauthorized access or script injections	

Test Procedures

1. **Procedure for Creating a Room (TC-001)**:
   (a) Navigate to the ”Create Room” option.
   (b) Click on ”Create Room” to generate a unique room ID.
   (c) Verify that the room ID is displayed, and the user is able to share it with others.
2. **Procedure for Joining a Room (TC-002)**:
   (a) Navigate to the ”Join Room” option.
   (b) Enter a valid room ID provided by another user.
   (c) Click the ”Join” button.
   (d) Verify that the user successfully joins the room and can collaborate in real-time.
3. **Procedure for Real-time Code Editing (TC-003)**:
   (a) Open the code editor after joining a room.
   (b) Make changes to the code (e.g., add or modify a line of code).
   (c) Verify that the changes are reflected instantly across all users’ editors in the room.
4. **Procedure for File Management (TC-004)**:
   (a) Open the file manager.
   (b) Create a new file or folder.
   (c) Edit or delete an existing file.
   (d) Verify that the file creation, editing, and deletion are synchronized across all users in the room
5. **Procedure for Code Execution (TC-005)**:
   (a) Write or paste code into the editor.
   (b) Click on the ”Run” button to execute the code.
   (c) Verify that the code is sent to the Piston API for execution and the output is displayed in the results section.
6. **Procedure for Using Group Chat (TC-006)**:
   (a) Open the chat window.
   (b) Type a message into the chat input box.
   (c) Click the ”Send” button.
   (d) Verify that the message is sent and displayed for all users in the room in real-time.

7. **Procedure for Checking User Presence (TC-007)**:
    (a) Join a room.
    (b) Check the list of users in the room.
    (c) Verify that the online status of users is accurately displayed, and notifica- tions appear when users join or leave the room.
8. **Procedure for Using Code Suggestions (TC-008)**:
    (a) Start typing code in the editor.
    (b) Check for autocompletion and code suggestions.
    (c) Verify that suggestions appear correctly, and they can be selected and applied to the code.
9. **Procedure for Syntax Highlighting (TC-009)**:
    (a) Open the code editor.
    (b) Write or paste code in a supported language.
    (c) Verify that syntax highlighting is applied correctly for the chosen language.
10. **Procedure for Room Notifications (TC-010)**:
    (a) Join or leave a room.
    (b) Verify that a notification is displayed to all users when someone joins or leaves the room.
11. **Procedure for File Synchronization (TC-011)**:
    (a) Create, edit, or delete a file in the file manager.
    (b) Verify that the changes are instantly reflected for all users in the room.
12. **Procedure for API Error Handling (TC-012)**:
    (a) Simulate an API failure (e.g., disconnect the internet during code execu- tion).
    (b) Verify that an appropriate error message is displayed when the Piston API fails or returns an error.
13. **Procedure for Testing Code Editor Performance (TC-013)**:
    (a) Open the code editor with multiple users in the room.
    (b) Simulate high usage by editing code rapidly across multiple users.
    (c) Verify that the editor remains responsive, and code changes are synchro- nized without noticeable delays.

Black-Box Testing

Test Tool: Manual testing using a browser and developer tools.

- **Functional Testing**:
  - Test all user-facing features, including room creation, real-time code synchronization, and chat functionality in CodeSync.
  - Example: Verify that users can create rooms, join rooms using a room ID, and that real-time code changes are reflected instantly.
- **Usability Testing**:
  - Ensure the platform is user-friendly, and the layout of the collaborative editor, chat, and file management are intuitive.
  - Example: Verify the ease of sharing room IDs and navigating between files, editor, and chat.
- **Compatibility Testing**:
  - Check CodeSync's functionality across various browsers like Chrome, Fire- fox, Safari, and ensure the real-time updates and chat work seamlessly.
  - Example: Ensure no synchronization issues arise across different browser platforms.
- **Security Testing**:
  - Test input fields and chat for security vulnerabilities like SQL Injection and XSS.
  - Example: Verify that the application sanitizes all inputs to prevent any form of malicious code execution.

White-Box Testing

Test Tool: Jest (JavaScript testing framework) for unit and integration testing.

- **Unit Testing**:
  - Test individual functions and components like room creation, real-time code broadcasting, and user presence updates in CodeSync.
  - Example: Test that the createRoom() function generates a unique room ID and that the broadcastCodeChange() function sends updates to all users in the room.
- **Integration Testing**:
  - Test the integration of CodeSync's frontend with its backend (Node.js and Express.js), and real-time communication (Socket.io) for real-time synchronization and notifications.

  - Example: Verify that the frontend sends user actions like code changes to the backend and that these actions are synchronized across all users in real-time.

- **API Testing**:
  - Test the integration with the Piston API to ensure code execution requests are handled correctly and results are returned without errors.
  - Example: Verify that the backend handles failed API responses gracefully and displays appropriate error messages to the users.

Conclusion

In conclusion, the testing of the CodeSync application is vital to ensure its relia- bility, functionality, and user satisfaction. This test plan outlines the comprehensive strategies to evaluate both functional and non-functional aspects of the application. By implementing thorough black-box and white-box testing methodologies, we aim to identify and rectify any potential issues before the official launch.

Through rigorous testing, we will ensure that users can effectively add, edit, and manage their coding tasks without encountering significant errors or usability issues. The focus on security testing also addresses the importance of safeguarding user data and preventing vulnerabilities that could compromise the application's integrity.

Ultimately, this test plan serves as a roadmap to achieving a robust CodeSync application that meets the needs of its users and fosters a collaborative coding en- vironment. By adhering to this plan, we can deliver a high-quality product that enhances productivity and facilitates seamless teamwork in coding projects.