5. Testing

Testing in SmartClass will mostly be divided into frontend and backend to test the functionality for each of their respective components. There will be tests that combine the frontend and backend to ensure the connection between the server and the database is working as intended. These tests will be used to ensure each of these components function how they are supposed to.

Testing begins in the development phase and will continue until after the project is complete. No testing methodology is perfect and tiny bugs or errors can be hard to detect in a test environment. Continued testing on critical components of the project will ensure the application continues to function as long as it can be efficiently maintained.

Writing tests using available APIs and public libraries will be sufficient for this project. The tests will be tailored to this application and focus on the continued functionality of the website. Each time the project is updated, automated tests will be run using CI/CD controls to ensure no functionality gets broken by the update. These tests are performed by Docker. Other APIs used for testing include JUnit, Mockito, and SpringBootTests.

Frontend

- Unit Tests (Simulate user workflows)
- Usability tests (Testing buttons)
- Security Testing
- UX Testing

Backend

- Use Mockito for testing REST APIs in the backend to create accounts and retrieve user information
- Integration Tests w/ frontend (Ensure the backend is functioning correctly with frontend)

5.1 Unit Testing

Unit testing will be done on an individual method basis. Each method must be tested to check for functionality and proper implementation. Unit testing consists of providing an input to a method and comparing the output with an expected output which is calculated beforehand.

JUnit will be used for conducting unit tests. JUnit is a library that provides many of the features wanted while testing. One useful set of methods includes assertions that allow for many types of comparisons, including equality, less than, greater than, and inequality. JUnit also allows automatic testing by allowing inputs to be read from an input file to test many cases at the same time without requiring knowledge of coding.

Key Interfaces:

- JUnit integration with Eclipse IDE
    - Allow for a single click to perform all tests
    - Shows detailed analysis of passing and failed tests

Testing Approach:

- Write multiple tests for each method
    - Test for edge cases
    - Test a variety of common values with randomly generated input
    - Test boundaries
- Use input files for test cases
    - Keep a structured file for writing inputs and expected outputs

Testing Tools:

- Decorations:
    - @Test : Indicates test methods
    - @BeforeEach : Performs this operation before every test
    - @AfterEach : Performs this operation after every test
    - @BeforeAll : Performes this operation once before any tests
    - @AfterAll : Performs this operation once after all tests are complete
- Methods:
    - AssertEquals(expected, actual) : Compares for equality between expected and actual
    - AssertNotEqual(expected, actual) : Compares for inequality between expected and actual
    - AssertTrue(condition) : Checks condition for true

Testing Process:

1) Write tests for methods as they are created
2) Run tests and fix bugs
3) Save tests to rerun in the future for regression testing

5.2 Interface Testing

Key Interfaces:

- User Interface to Backend
    - REST API endpoints for user authentication (ie. login and registration for different roles)
    - WebSocket connections (polls, live chat, question posting)
    - File upload of lecture materials
- Backend to Database
    - Data persistence for accounts, attendance and course data
    - Storage of quiz responses and participation statistics
- Role-Based Access
    - Student, Teacher, TA permissions
- Frontend Components
    - Navigation between course pages, lecture content, quizzes, etc.
    - Quiz display and submission

Testing Approach:

- API Interface testing
    - Test course creation and joining/adding courses
    - Test quiz creation and collecting responses
    - Test anonymous question submissions
- Websocket Testing
    - Test real-time chat function
    - Test participation in quizzes with more than one user
    - Test storage of chat
- Files & downloads
    - Test lecture material uploads
    - Verification of file types
    - Test download of files and materials

Testing Tools:

- API testing

    - Postman for endpoints
    - Automated API tests in CI/CD pipeline
- Fronted Interface
    - React testing library and Jest for testing the components
- Backend Interface Testing
    - Spring Boot
    - JUnit for testing backend service interfaces

Testing Process:

1. Test every interface independently
2. Integrated tests
3. Verify consistency across different interfaces
4. Test role-based permissions
5. Error handling
6. Real-time features maintain consistent and synced data.

5.3 Integration Testing

Critical Integration Paths:

- User Authentication and Authorization
    - Student, TA, and Teacher registration
    - Role-based access
    - Class creation and joining a class

  Justification: This path is important because it controls access to the overall use and functionality of SmartClass. It must enforce security and accessibility for different user types.

- Live Interaction
    - Real-time chat function
    - In-class quiz delivery and response
    - Submissions and answering

  Justification: This path is important because is the main feature and use for SmartClass (enhancing student participation)

- Content
    - Lecture uploads and downloads
    - Course organization
    - Previous lecture and quiz access
    - Attendance tracking
    - Performance statistics

Justification: Storage and retrieval of materials is important for the platform to be useful outside of live classroom activities. The statistics are also very necessary for teachers to evaluate engagement.

Testing Tools:

- Automated Tests
    - Supertest for API
    - CI/CD
- API testing
    - Postman for API workthrough
- Real-Time Testing
    - Websocket libraries to test functionality of chat and quiz

Testing Process:

1. Integration tests will run automatically in the CI/CD pipeline for changes in code
2. Tests will simulate interaction of different user types
3. Each integration path will be tested independently
4. Failed tests will block a merge to the main branch

Testing Approach:

- Full Test Scenario:
    - Full simulation of the web application being used, with multiple user types
    - Creating classes, joining, posting, responding
- Components
    - Testing interactions between different components
    - Verify that data and responses are tracked correctly with the frontend and backend
    - Test the websockets
- Data Integrity Testing
    - Verify that the user inputs are correctly stored
    - Test that the statistics and analytics work
    - Quiz results

5.4 System Testing

Will validate that the application functions as a whole, ensuring that all the components work together to provide the "easy to use" application we hope for.

Testing Strategy:

The system will combine multiple testing methods to make sure that SmartClass meets the functional and non-functional requirements outlined in section 2.1.

Key Test Scenarios:

- Full Classroom Test:
    - Scenario: Login multiple users of different roles, this being teacher, students, TA
    - Actions: Teacher starts a lecture session, uploads materials, creates polls and quizzes
    - Student Actions: Attendance, answering polls, submitting anonymous questions
    - Tools: Selenium for UI testing
- Course Management
    - Scenario: Teacher creates course, generates join codes, students join
    - Actions: Manage attendance
    - Verification: Role-based permissions are enforced, class materials are available
    - Tools: Automated Scripts

End to end Process:

- Live lecture activities
    - Teacher starts a lecture and attendance
    - Real-time quiz
    - Student question submission
    - Teacher/TA response
    - Verification all real-time features function

Performance:

1. Load Testing
    a. Testing response times
2. UI load Testing
    a. Determine amount of users can be supported
    b. Website remains fast and responsive

Tools:

- Selenium
- Test scripts
- CI/CD pipeline

5.5 Regression Testing

To make sure that new feature implementations do not break existing functionality, we have a combination of tools that we use.

- Automated Testing
    - Unit tests for both frontend and backend are being implemented. For background logic, we will use Mockito.
    - The tests are expected to grow over time whenever new features are introduced.
- Integration Testing
    - Integration tests between the frontend and backend are used to verify that APIs behave as expected with UI components.
- Usability Testing
    - Manual and observational testing are part of the ongoing development, where buttons layouts, and flow are tested to make sure other features such as quizzes, live chat, and Q&A continue to work

Critical Features to Maintain

- Authentication and Role-based Views
    - Login and dashboard logic for students, TAs, and teachers must not break as they control feature access.
- Anonymous Q&A and Live Chat
    - These core engagement features directly align with the project's goal to support shy or remote students and must remain functional.
- Quizzes and Polling System
    - Any failure in these systems would disrupt classroom interactivity and grading processes.
- Document Uploading and Archiving
    - Teacher's ability to upload notes and student's access to past class content are important.

Is it Requirements-Driven?

Yes, the testing is driven heavily by:

- User Needs & Functional Requirements
    - Anonymous interaction, quiz handling, participation tracking, and accessibility support are outlined as needs and requirements.
- Engineering Standards
    - Compliance with ISO 9000 and 12207 makes sure of software quality and adherence to a good development lifecycle, including testing phases.

Tools Involved

- Mockito for backend REST API testing
- React Testing Library
- Manual usability testing

5.6 Acceptance Testing

To show that both functional and non-functional requirements are satisfied, we will use a structured acceptance testing that includes:

Functional Requirements Validation

- Scenario-based testing: Each core feature will be tested with use cases based on the requirements outlined in our documentation
- End-to-end workflow tests: An example can be a teacher creating a class, adding a quiz, students joining and responding, teacher viewing participation results. This demonstrates that multiple components are functioning together properly

Non-Functional Requirements Validation

- Usability: Conduct usability testing with diverse users to ensure the UI is intuitive and responsive
- Performance: Stimulate high user loads to validate the scalability of chats, quiz submissions, and database queries
- Accessibility: Ensure features like text-to-speech, keyboard navigation, and clean design work for users with different needs
- Security: Validate secure login, data encryption, and access control through test accounts and role-switching

Involving Clients in Acceptance Testing

- Client Walkthroughs: Live demos where the client performs as a teacher to validate functionality and gather feedback
- Feedback-Driven Iterations: Weekly client meetings allow for iterative feedback and verification of components as they are developed

5.7 Security Testing

Security testing is important for the SmartClass learning platform due to the nature of the data shared. To be more specific it holds user credentials, education data and resources, and an overall goal to maintain academic integrity. Due to SmartClass being an interactive learning platform that connects students, teachers, and TAs, SmartClass requires in-depth security testing to protect the privacy of users and prevent unauthorized access.

Security Testing Areas:

- Authentication mechanisms
- Protect personal information
- Session management
- Integrity within the anonymous posting features
- Protection against common web vulnerabilities:
    - XSS
    - SQL injection
    - CSRF
- Security within file upload

Testing Methods:

1. Burp Suite
    a. Web vulnerability scanning
    b. Intercepting proxy to analyze the requests and responses

    c. Active scanning for web vulnerabilities (referenced above)
    d. Session analysis for weaknesses
    e. Testing the anonymous posting features for integrity.
2. Manual Review
    a. Reviewing code for the authentication methods
    b. Review of password storage
    c. Check guidelines in comparison with ISO 27000 (commonly used for web applications)
    d. Testing privilege escalation
3. Data Protection Testing
    a. Verification of data encryption (HTTPS)

Testing Plan:

- Initial scan before “completion”
- After major refinements in code base there will be security scans
- Light “pen-testing” before release, hands on approach.

Documentation:

- Findings documented alongside severity ratings
- Vulnerabilities tracked and resolved
- Security testing results logged to be reviewed before release.

As mentioned in section 2.2 the ISO 27000 is centered around security and privacy protection and will be used to ensure that SmartClass provides a secure environment and interface for users. The goal is to protect user data using good cybersecurity practices.

5.8 Results

The current results of testing are light. Unit tests are unimplemented, so all testing has been done by hand up to this point. This tedious process consists of writing code, restarting the server to test, fix code, restart the server to test again, and repeat. The goal is to begin automating tests using the strategies mentioned in this section to alleviate some of the workload.

However, the results of the manual testing are good. Bugs have been fixed by trial and error. Updates to the webpage are seen with each refresh allowing for quicker testing on the frontend. The user experience isn’t anything fancy as of yet on account of the focus being on adding features to the website. That being said, the navbar is functional and accessible at all times from all pages. All currently in-progress and finished pages are accessible from the navbar for usability.

No user testing has been done yet, so it’s to be determined as to how others would interact with the website. More testing also needs to be done with color and accessibility for people of different capabilities. Decisions on font styles and sizes are reliant on this testing getting done.

The plan is to begin testing as soon as this week since the website is finally in a usable state. Enough features have been implemented to allow for user, system, and acceptance testing to begin. Soon, unit and regression tests will also be integrated using JUnit.