Software Implementation and Testing Document

For

Group 5

Version 1.0

Authors:

Matthew Christoffel
Alan Castillo

Kirsten Blair

Writer Leto

Alexander Kajada

1. Programming Languages (5 points)

List the programming languages use in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).

Python - Python is used as the primary server-side language for the web application. It powers the Flask framework which handles all the backend functionalities including routing, user authentication, data processing, and interactions with the database. Python is chosen for its simplicity and readability, making it easy to maintain and scale. Its extensive libraries and frameworks, particularly Flask, provide robust tools for web development. Additionally, Python's compatibility with various database systems and its efficiency in handling server-side scripting make it ideal for backend development.

HTML - Utilized to structure the web pages of the application, forming the skeleton of the user interface. We chose this because it is the standard markup language for creating web pages, essential for any web development project.

CSS - Employed to style the web pages, enhancing the user interface with visually appealing designs, layouts, and responsive features.

2. Platforms, APIs, Databases, and other technologies used (5 points)

List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).

Flask:

- *Where Used: Flask is used as the web framework for your entire application. It handles the routing, HTTP requests, and the rendering of HTML templates. Specific examples include:*
    - *Handling routes like '/', '/profile', '/create_post', etc.*
    - *Rendering templates such as home.html, profile.html, create_post.html, etc.*
    - *Processing form data from POST requests to create, edit, like, or dislike posts.*

SQLite:

- *Where Used: SQLite served as the database management system for the project. It was used to store and manage data related to users, posts, comments, likes, and other functionalities of your tech news style social media web application.*

3. Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the **functional requirements** listed in your RD).*

Posting functionality: I manually tested the posting functionality by inputting a title, description, and URL. Checked that the post appears on the site with the same input I put in.

Testing Scenarios:

- Created a post with valid inputs to ensure it is successfully added to the site.
- Attempted to create a post with missing information (e.g., no title or URL) to verify that the system prompts for necessary details.
- Tested updating existing posts to ensure changes were accurately reflected.

The rest of the function requirements have not yet been implemented into the code, but this is how we would test each one:

User Registration and Authentication

Testing Approach:

- Verified user registration and authentication process using a mock Google account. Ensured successful redirection to the login page and checked session management after login.

Testing Scenarios:

- Registered a new account and logged in to verify the process.
- Attempted to log in with incorrect credentials to test error handling.
- Checked session persistence across different site pages.

User Profile Management

Testing Approach:

- Tested profile management functionality by updating user information and checking if changes were reflected on the user's profile page.

Testing Scenarios:

- Manually updated username and email to ensure changes are saved and displayed.
- Reviewed the user post history on the profile to confirm it lists all user posts.

Admin Content Moderation

Testing Approach:

- As an admin user, manually tested the ability to delete posts and user accounts. Verified that actions by admin accounts had the intended effect.

Testing Scenarios:

- Manually deleted a user-created post to ensure it was removed from the site.
- Manually removed a user account and verified that their associated posts and comments were also removed.

Search and Filtering

Testing Approach:

- Tested search functionality by manually entering various keywords and phrases to see if relevant posts were displayed. Also tested filtering functionality by selecting different post categories.

Testing Scenarios:

- Manually searched for posts using specific titles and verified the correct posts were returned.
- Applied different filters and checked if the displayed posts matched the filter criteria.

4. Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the **non-functional requirements** listed in your RD).*

This is how we can test the non-functional requirements when the time comes:

1. Performance Testing:

Methods:

- Load Testing: Simulate multiple users accessing your website simultaneously to ensure that it can handle high traffic without significant performance degradation.
- Speed Testing: Use tools like Google PageSpeed Insights, GTmetrix, or WebPageTest to analyze the load time of your website and identify bottlenecks.

Procedure:

- Run load tests by using tools like JMeter or LoadRunner. Monitor response times and system behavior under different levels of user load.
- Use speed testing tools to measure the load time of your website's pages. Analyze the reports to identify elements that take the longest to load and optimize them (e.g., image size reduction, script minification).

2. Usability Testing:

Methods:

- User Testing Sessions: Conduct sessions where real users interact with the user interface and gather feedback.
- Heuristic Evaluation: Use a set of usability principles (heuristics) to evaluate the user interface.

Procedure:

- Recruit a small group of typical users and give them a set of tasks to complete using the website. Observe how easily they can complete these tasks and get their feedback on the user interface.
- Conduct a heuristic evaluation by checking if the user interface conforms to established usability principles, such as Nielsen’s Heuristics. This includes consistency, error prevention, and the clarity of navigation and labels.

3. Maintainability Testing:

Methods:

- Conduct code reviews and refactorings to ensure code readability, modularity, and adherence to best practices.

Procedure:

- Regularly review the codebase with the development team to identify areas for improvement. Implement automated code quality tools like Pylint (for Python) to enforce coding standards.
- Document any major architectural changes and update the documentation accordingly.

4. Performance Monitoring and Logging Testing:

Methods:

- Implement logging mechanisms in the application and use monitoring tools to track performance metrics.

Procedure:

- Integrate a logging framework (like Python's logging module) to record system events, errors, and performance metrics. Use tools like Grafana or Prometheus to monitor system performance in real-time. Simulate various scenarios (such as high traffic or system errors) and verify that they are accurately logged and reported.

5. Security Testing:

Methods:

- Perform vulnerability assessments and penetration testing to ensure the security of the application.

Procedure:

- Use tools like OWASP ZAP or Burp Suite to scan the application for vulnerabilities (e.g., SQL injection, XSS). Conduct penetration tests to simulate cyber attacks and assess the system’s ability to withstand them. Ensure that user data is encrypted in transit and at rest using protocols like HTTPS and secure storage solutions.

5. Non-Execution-based Testing (10 points)

Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).

This is how we will perform non-execution based testing:

Peer Review: Organize peer review sessions where team members systematically examine each other's code. This can be done using a platform like GitHub, where pull requests are used to facilitate code reviews. Team members can comment on specific lines of code, suggest improvements, and ensure adherence to coding standards.

Checklist-Based Inspection: Develop a checklist based on common errors in Flask applications, Python coding standards,, and project-specific requirements. During code reviews, use this checklist to ensure all important aspects are covered. For example, checking for consistent naming conventions, and verifying that routes are correctly defined and secured.

Automated Code Analysis Tools: Use tools like Pylint, Flake8, or PyCodeStyle to perform static code analysis. These tools can automatically identify issues in the code, such as syntax errors, potential bugs, coding standard violations, and other anomalies.

Walkthroughs:

Team Meetings: Conduct regular team meetings where each member explains their code, focusing on the logic, implementation choices, and how their code integrates with other parts of the project. This can be especially useful for understanding complex parts of the application, like user authentication and database interactions.

Functionality Walkthroughs: Go through the functionality of the application step by step. For example, demonstrate the process of user registration, posting functionality, and admin controls. This can help in understanding the application flow and uncovering any gaps in the requirements.

Best Practices:

Documentation: Keep a record of the findings and suggestions from code reviews and walkthroughs. This documentation is useful for future reference.

Continuous Process: Make code reviews and walkthroughs a regular part of our development cycle. This encourages ongoing quality assurance and knowledge sharing within the team.