Software Implementation and Testing Document

For
Group 10

Version 3.0

Authors:
Claudio Olmeda Florio
Alex Gonzalez
Samuel Anderson
Joel Bustamante
Juancarlos Alguera

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).

PHP:

- **Where:** We use php for backend development, handling server-side logic in Laravel.
- **Reason:** We are using Laravel, which is a PHP framework that provides useful features for building web applications. Laravel's extensive features and documentation made it a reliable choice for the backend development for our project.

TypeScript:

- **Where:** We use TypeScript for frontend development. Specifically for creating components in the resources/js directory. TypeScript powers UI components like NewPost.tsx or PostCircle.tsx.
- **Reason:** TypeScript enhances JavaScript by providing static typing which helps catch errors during development and improves code reliability.

JavaScript:

- **Where:** We use JavaScript for frontend development alongside TypeScript. Used in configuration files like eslint.config.js and tailwind.config.js.
- **Reason:** JavaScript enables interactive features in our frontend. While TypeScript handles most of the typed frontend logic.

SQL:

- **Where:** Database queries, migrations, and schema definitions
- **Reason:** SQL is the standard language for PostgreSQL which we are using, allowing data storage and retrieval.

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).

Platforms:

Web Application:

- The app is deployed as a web-based application accessible through browsers.

Neon (Database Hosting):

- Hosting the PostgreSQL database for the project.
- **Why:** Neon provides reliable free hosting tailored to PostgreSQL.

APIs:

LeafletJS:

- **Where:** Used for the frontend map component.
- **Why:** Displays user posts and their geographic locations with interactive features such as zooming and panning. LeafletJS provides extensive customization options for integrating maps.

Laravel API:

- **Where:** Backend for handling HTTP requests, and talking with the database.
- **Why:** Supports core functionality like creating posts, liking, commenting, retrieving user data, and managing user sessions

Database:

PostgreSQL:

- **Where:** Backend for data persistence.
- **Why:** We chose postgres out of familiarity and compatibility with Laravel.

Other Technologies:

React:

- **Where:** Frontend framework
- **Why:** React facilitates building dynamic UI components ensuring consistent user interactions across the app.

shadcn:

- **Where:** Frontend UI components.
- **Why:** Used for UI elements like buttons, dropdowns and other interactive elements to ensure a consistent look throughout the app.

vite:

- **Where:** Frontend development environment for compiling and bundling assets.
- **Why:** Ensures efficient handling of TypeScript and CSS.

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).

Unit Testing:

- **Backend:** Tested Laravel controllers (PostController, CommentController, LikeController) making sure it behaves as expected for actions like creating posts, liking, commenting, and retrieving data. We also made sure to validate database migrations.
- **Frontend:** Tested React components (like NewPost.tsx, PostCircle.tsx) to confirm they rendered correctly and functioned as intended including user interactions like post creation and circle visualization.
- **Tools**: PHPUnit (backend)
- **Outcome**: Individual units functioned as expected

Interaction Testing:

- **What:** Verified interaction between the Laravel backend and the React frontend.
- **How:** Tested API endpoints for functionality like user authentication, post creation, and comment handling, making sure they returned the correct responses. Simulated user actions in the frontend and validated the corresponding backend.
- **Tools:** Postman for API, and browser testing using the browser console.
- **Outcome:** The interactions between components were confirmed to meet functional requirements.

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).

Performance Testing:

- **What:** Tested the apps speed and responsiveness under different conditions.
- **Backend:** Checked API response times for user login, post creation, and data retrieval.
- **Frontend:** Measured the time it takes to load and render components, such as the map view with user posts and pins.
- **Tools:** Postman

Security Testing:

- **What:** Verified the apps security
- **Backend:** Checked secure authentication mechanisms like hashed passwords.

- **Frontend:** Checked that sensitive user actions, like posting or deleting, required proper authorization.
- **Outcome:** Authentication mechanism and user permissions worked as intended.

Reliability Testing:
- **What:** Ensured the application worked consistently across different environments.
- **How:** Tested the app on different browsers and devices to confirm expected behavior. Ensured compatibility across multiple operating systems.
- **Outcome:** The app performed consistently across all tested platforms.

Usability Testing:
- **What:** Assessed the user interface and overall user experience
- **How:** We conducted testing sessions using different roles. Looked at user interactions with key features such as posting, commenting, and map navigation.
- **Outcome:** Minor changes were made to improve users' experience.

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

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

Code Reviews:
We conducted code reviews to identify potential issues, and to verify functionality. Team members reviewed each other's code during pull requests on the Git repository. We focused on ensuring proper input validation, adherence to coding standards, and ensuring it followed our functional requirements. Code reviews helped catch bugs early, and improved our collaboration.

Walkthroughs:
We did team walkthroughs of certain sections of our app to make sure everyone was caught up and understood the logic and flow of the application. These were mainly conducted when someone has made a major change to the code. For example we had one when the backend was completed to help the people working on the front end understand the vision. This improved out team understanding of the code and enhanced overall design coherence.

Design Inspections:
We inspected the design and database schema for potential flaws. We did a group review session of our database schema where we looked over our tables and models making sure they aligned with our project's goal. We were able to optimize our database and found a duplication of userID.