Coding Project Final Report

CS 440
at the
University of Illinois Chicago

December 2024
Ryder Douglas, Apoorv Lodhi, Ragini Kalvade and Niyati Malik

Table of Contents

I Project Description

1 Project Overview

"Attack the Virus" is an educational web application designed to enhance public understanding of the human immune system, the role of antibodies, and the effectiveness of vaccines in combating infections, plus basic math skills. The intended audiences are students and young adults, educators, and anyone who may not be informed about viruses and vaccines.

The program consists of multiple components:

Quiz: A multiple-choice game designed to teach players about the human immune system, the appropriate vaccines for various diseases, and the history of both illnesses and vaccinations. Players must answer a series of questions, with a time limit for each question. Answering correctly increases the player's score. Answering incorrectly, or letting the timer run out, reduces the player's health. In either case, the player is presented with an "Info Card", containing information relevant to the question, before moving on to the next question.

Other Games: Virus Breaker, Virus Sweeper and Virus Panic are games with an educational and/or virus theme, designed to both educate and entertain players. Virus Breaker is a virus-themed version of Brick Breaker, Virus Sweeper a virus-themed version of Minesweeper, and Virus Panic requires players to complete math problems in order to clear a grid of virus cells.

Clinic Finder: Using the Google Maps API, this tool allows the user to find clinics, doctors and hospitals within a 1-km radius of their current location.

AI Chatbot "Vacciwiz": An artificial intelligence chatbot that can communicate with users to answer questions and provide information about viruses, diseases, vaccines, the immune system, and more.

2 Project Domain

Attack The Virus lies within the domain of educational games and services, specifically those regarding health, diseases and vaccines. The recent pandemic, and our collective responses to it, have shown a lack of awareness and education, and an overabundance of uninformed or misinformed people, in regards to viruses, vaccines, and the relationship between the two. Existing sources of education and information about these subjects tend to be linear and uninteractive, discouraging people from learning about topics of critical importance. Since they also tend to be voluntary and self-guided, many people who would otherwise be willing to engage with the material decide not to do so. Sources of information often target older audiences, meaning that younger children may not be capable of engaging with much of the material, even though children are affected by viruses and vaccines as much as, if not more than, many other groups.

Attack The Virus seeks to address these issues by presenting the same informative, educational material in an entertaining and exciting manner. By gamifying the learning experiences and blurring the line between game mechanics and learning opportunities, the app seeks to be a more accessible, engaging experience for all demographics.

3 Relationship to Other Documents

The program described in this document was based on the development report written by CS 440 Spring 2024 Group 2. The theme and motivation of the project is closely aligned with those specified in the original design document, though the implementation did not.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Design document: The project development report created by CS 440 Spring 2024 Group 2

Attack The Virus: The web application described in this document.

Component: Self-contained Angular UI element/module that binds logic, data and presentation for a specific portion of the DOM.

Route/Routing: Refers to the mapping in Angular between a URL path and the corresponding component.

Service: Angular class designed to encapsulate business logic, data fetching, and shared functionality.

Service Layer: Collection of Angular services.

4b Data Dictionary for Any Included Models

Attack The Virus used many data structures to store, send and manage data:

User Table: Stores personal information and progression data for system users. The `level` field also determines which question from the **Questions Table** is presented to the user next, which enables a progression-based flow of questions.

Agent_Info Table: Stores details about various agents involved in the system, categorized by type (**IMMUNO_AGENT** and **INFECTIOUS_AGENT**) and subtype (**VACCINE, MEDICINE, VIRUS, BACTERIA,** etc.). The agents are referenced in the **Questions Table** to provide additional context for quiz content. For example, a question might reference a virus or vaccine based on its classification.

Questions Table: Stores the quiz questions, answer choices, correct answers, and relevant metadata.

Leaderboard Table: Stores high scores of users in a central repository, mapping username to score.

II Project Deliverables

This semester, our group successfully planned and developed several foundational components for the Attack The Virus web application. Development centered around these components, which provide the core functionality of the application. These include:

Quiz With Leaderboard: A fully functioning quiz game, with the structure, rules, and many questions implemented, allowing for easy extension and expansions. The implementation includes a leaderboard and an integrated backend.

Frontend: The team implemented a frontend using Angular and an interface created with GSAP. It is reactive, communicative, and also allows for easy expansion for future additions.

Additional Games: We designed and developed Virus Breaker, Virus Sweeper and Virus Panic – 3 additional games intended to add entertainment value to the application.

Clinic Finder: Using Google Maps API, the team implemented a clinic finder that locates clinics, doctors and hospitals within a 1km radius of the user's current location.

Vacciwiz: The team also implemented an AI chatbot that can communicate with users, answering questions and providing knowledge about viruses, vaccines, and human health.

5 First Release

Date: 10/11/24

The first release cycle focused on delivering the foundational functionality of a quiz application, combining interactive features with a polished user interface. The team implemented core quiz mechanics, allowing users to engage with questions and receive scores based on their answers. The interface was built using Angular, with a clean, responsive initial design that ensures a user-friendly experience. To enhance visual appeal, GSAP was utilized to incorporate smooth animations and dynamic theming, giving the application a modern, engaging aesthetic.

On the backend, the team set up a robust infrastructure using Spring Boot and MySQL, providing seamless integration between the frontend and backend. APIs were developed to handle question retrieval, answer submission, and score management, ensuring efficient and scalable data flow. With these features in place, the release established a strong technical foundation and a cohesive user experience, paving the way for future enhancements and additional functionality.

6 Second Release

Date: 11/11/24

The second release focused on enhancing the quiz application by implementing new features, improving user engagement, and addressing user feedback. The core quiz functionality was upgraded with additional question types, better feedback mechanisms, and refined scoring logic, making the experience more engaging and educational. To personalize the user journey, the team introduced a system for customizable avatars, allowing users to select and modify avatars that represent their profiles within the application.

A standout addition in this release was the Virus Breaker, a gamified feature designed to test and reinforce user knowledge in a fast-paced, interactive format, further enhancing the application's appeal. Another significant enhancement was the integration of a Map feature, enabling users to visualize progress, unlock new areas or content, and explore the application's structure in an intuitive way.

The UI underwent a comprehensive update based on user feedback, incorporating improvements in design, navigation, and accessibility. These changes focused on simplifying the user interface, making it more visually appealing, and ensuring a smoother overall experience. This release not only elevated the application's functionality but also strengthened user engagement and satisfaction, laying the groundwork for future innovation.

7 Comparison with Original Project Design Document

As development began, it became evident that the original design document was poorly defined and lacked clarity in critical areas. The gameplay was only vaguely described, aside from being some sort of 3D adventure game with action and roleplaying elements. The educational objectives were vaguely described, offering little guidance on how these elements should function within the gameplay, and mathematical calculations were not mentioned at all. Additionally, the ambitious scope of a 3D adventure game presented significant challenges in terms of time, resources, and technical feasibility, not least due to our group generally having little to no experience with game development. This lack of direction and practicality made it impossible to adhere strictly to the original specifications while delivering a functional and polished product within the constraints.

Faced with these challenges, the team decided to pivot while retaining the core motivation and intention of the original concept: to create an engaging, educational experience. Instead of a 3D adventure game, we opted for a more achievable design that focused on a quiz-based format with gamified features, such as customizable avatars and interactive mini-games like Virus Breaker. This approach allowed us to meet the educational goals in a more scalable and resource-conscious manner. While the implementation and design differed significantly from the original vision, the final product stayed true to its purpose of combining learning with entertainment, resulting

in a practical and polished solution that better aligned with the team's capabilities and timeline.

III Testing

This portion of the report documents the testing procedures applied and the results obtained for the Attack the Virus project. The project was developed using Spring Boot with Java and MySQL, and testing was conducted using JUnit and Mockito frameworks. All the controller and service layers were thoroughly tested, yielding passing test cases.

8 Items to be Tested

The **Attack the Virus** project comprises various components, each responsible for specific functionality. Backend testing focused on the following:

- **Controller Layer:**
  - Validates incoming API requests.
  - Processes user input and delegates tasks to the service layer.
  - Ensures proper error handling and response generation.
- **Service Layer:**
  - Implements the business logic of the application.
  - Communicates with the data access layer for CRUD operations.
  - Ensures correct processing of data and interactions with repositories.

Meanwhile, the frontend functionality that we tested consisted of:

- Component Rendering
- Form Validation
- API Integration
- User Interactions

9 Test Specifications

TC-001 – Validate Controller Layer Request Handling

Description: Test if the controller layer handles valid and invalid API requests correctly.

Items covered by this test: Controller Layer

Requirements addressed:

Validate input, handle API routes, and generate the appropriate HTTP responses.

Environmental Needs: Spring Boot, JUnit, Mockito, and a mock database.

Intercase Dependencies: NA

Test Procedures:

1. Mock HTTP requests for each endpoint.
2. Pass both valid and invalid inputs.
3. Verify the returned HTTP status codes and responses.

Input Specification:

- Valid and invalid JSON payloads for each API endpoint.

Output Specifications:

- 200 OK for valid requests.
- 400 Bad Request for invalid inputs.

Pass/Fail Criteria: Test passes if the correct HTTP status codes and responses are returned.

TC-002 – Validate Service Layer Logic

Description: Test the correctness of business logic in service layer methods.

Items Covered: Service Layer.

Requirements Addressed: Ensure proper data processing and interactions with repositories

Environmental Needs: Spring Boot, JUnit, Mockito, and a mock database.

Intercase Dependencies: NA

Test Procedures:

1. Mock dependencies such as repositories and external services.
2. Pass various inputs to service methods.
3. Verify the correctness of returned results and repository interactions.

Input Specification:

- Sample input objects which are mocked

Output Specifications:

- Correctly processed results based on business rules.

Pass/Fail Criteria: Test passes if the returned results match the expected values.

TC-003 – Validate Component Rendering and UI Functionality

Description: Test that Angular components render correctly and handle user interactions seamlessly.

Items Covered: Angular components and services.

Requirements Addressed: Component rendering, form validation, API integration, and user interaction handling.

Environmental Needs: Angular and backend.

Intercase Dependencies: Controller layer integration.

Test Procedures:

1. Render each component with mock data and verify the output.
2. Simulate user interactions like clicks, form submissions, and navigation.
3. Test service methods to ensure correct API calls and data handling.

Input Specification:

- Mock data- names, scores etc.
- Simulated user events (e.g., button clicks, form inputs).

Output Specifications:

- Correct rendering of components with provided data
- Proper handling of user interactions and form validation

Pass/Fail Criteria: Test passes if the components render correctly and user interactions function as expected.

10 Test Results

TC-001 – Validate Controller Layer Request Handling

Date(s) of Execution: December 2, 2024

Staff conducting tests: Apoorv Lodhi

Expected Results:

- Valid requests return 200 OK
- Invalid inputs return 400 Bad Request

Actual Results: As expected.

Test Status: Pass.

TC-002 – Validate Service Layer Logic

Date(s) of Execution: December 2, 2024

Staff conducting tests: Apoorv Lodhi

Expected Results:

- Business logic processes inputs and returns correct results.
- Repository interactions match expectations.

Actual Results: As expected.

Test Status: Pass.

TC-003 – Validate Service Layer Logic

Date(s) of Execution: December 3, 2024

Staff conducting tests: Niyati Malik

Expected Results:

- Components render correctly
- API calls return accurate information.
- User interactions work as intended

Actual Results: As expected.

Test Status: Pass.

11 Regression Testing

Test Scope:

Regression testing was performed on all critical areas of the project, including:

1. Controller layer request handling.
2. Service layer business logic.
3. Angular frontend components and interactions.

Test Procedures:

1. Reran all previously executed test cases using JUnit.
2. Verified integration between the Angular frontend and Spring Boot backend using end-to-end tests.
3. Focused on high-impact areas, such as the leaderboard and quiz features, which underwent modifications.

Regression Test Results:

- **Date(s) of Execution:** November 15, 2024.
- **Staff Conducting Tests:** Ragini Kalvade and Ryder Douglas.
- **Results:** All previous test cases passed successfully. No new issues were identified during regression testing.

This concludes the testing section for the Attack the Virus project.

IV Inspection

12 Items to be Inspected

Ensure that the map functionality takes the user's current location and shows nearby locations.

Ensure that the questions in the quiz are easy to understand.

Ensure that the rules of the other games are not complicated and are understandable.

Ensure that the players' inputs are handled properly and their progress is stored at the end of the game.

Ensure that all images in the game load on time.

13 Inspection Procedures

13a Map Functionality

- **Ensured the map functionality takes the user's current location:**
  - Launched the application on devices with GPS enabled.
  - Tested at different locations to verify the map fetches the current location accurately.
  - Simulate GPS coordinates using developer tools to check functionality in virtual locations.
- **Ensured the map finds the clinics nearby:**
  - Tested with sample searches in locations with clinics and verify the map displays nearby results correctly.
  - Verified the displayed results are accurate by cross-referencing with Google Maps.

13b Quiz Questions

- **Ensured the questions in the quiz are easy to understand:**
  - Reviewed all quiz questions to ensure clarity and simple language.
  - Conducted usability testing with a group of friends from different courses to identify any ambiguities in the questions.
  - Documented any feedback and iterate on unclear questions.

13c Rules of Other Games

- **Ensured the rules of the games are understandable:**
  - Verified the rules are clear in the game description.
  - Confirmed the rules are concise.
  - Tested the rules with a small group of friends from different courses to confirm they can proceed without confusion.

13d Player Inputs and Progress

- **Ensured player inputs are handled properly:**

    - Entered different valid and invalid inputs during the game to test how the system handles each case.
- **Ensured players' progress is stored at the end of the game:**
    - Played the game to a checkpoint and closed the application. Relaunched to confirm progress is saved and restored accurately.
    - Tested progress saving across multiple devices.
    - Confirmed that all the user inputs are being stored in the database.

13e Images Loading

- **Ensured all images in the game load on time:**
    - Ran the application on different devices and networks to test image load times.
    - Simulated slower network conditions to ensure placeholder images or progress indicators appear until loading completes.

14 Inspection Results

14a Map Functionality

Inspection: Ensured the map functionality takes the user's current location and finds nearby clinics.

Procedure: Tested the application on two devices (Windows and Mac) with GPS enabled. Verified the location accuracy by comparing the results with Google Maps. Simulated different GPS coordinates to confirm nearby clinics were displayed correctly.

Inspected by: Niyati Malik (Windows) and Apoorv Lodhi (Mac)

Date: 10-05-24

14b Quiz Questions

Inspection: Ensured that the quiz questions are easy to understand.

Procedure: Reviewed all quiz questions for clarity and simplicity. Conducted a usability test with five participants to gather feedback on any confusing language or phrasing. Revised two questions based on feedback to improve readability.

Inspected by: Friends from other courses.

Date: 09-10-24

14c Rules of Other Games

Inspection: Ensured the rules of other games are not complicated and are understandable.

Procedure: Verified the rules displayed before each game were concise and contained relevant examples. Conducted a test with three players to confirm understanding of the rules. No changes were necessary as all participants were able to proceed without confusion.

Inspected by: Friends from other courses.

Date: 11-09-24

14d Player Inputs and Progress

Inspection: Ensured player inputs are handled properly and progress is stored at the end of the game.

Procedure: Entered valid and invalid inputs across all input fields to confirm proper error handling and validation. Played through the game to multiple checkpoints, exited, and re-entered to verify progress was accurately saved and restored.

Inspected by: Ragini Kalvade

Date: 11-15-24

14e Images Loading

Inspection: Ensured all images in the game load on time.

Procedure: Tested the application on high-speed and low-speed networks. Verified placeholder images appeared while loading on slower connections. Checked the developer console for missing image errors; no issues were found.

Inspected by: Ryder Douglas

Date: 10-10-24

V Recommendations and Conclusions

Most items covered have passed their testing and inspection process successfully. The map functionality, quiz questions, and game rules were all validated as functional and user-friendly. Player input handling and progress saving passed inspection.

All identified issues were addressed during follow-up sprints and retested, ensuring compliance with project requirements and a seamless user experience. The application is now ready, with all critical functionalities inspected and validated.

VI Project Issues

15 Open Issues

Factors that are uncertain and might make significant difference to the product -

Integration of Location-Based Clinic Finder: Uncertainty remains about the accuracy and accessibility of clinic location APIs, particularly in regions with sparse healthcare facilities. These issues may affect the feature's usability and reliability for players.

API Support for Disease and Vaccine Updates: Dependencies on external APIs for accurate and timely information about diseases and vaccines raise questions about service availability and long-term reliability.

Hardware Compatibility for Interactive Features: The performance of advanced features, like the Virus-Breaker mini-game or GSAP animations, across lower-end devices is still under evaluation and may impact the app's accessibility.

Bringing these uncertainties to light allows the team to:

- Assess risks and prioritize solutions during development.
- Adapt the game design to mitigate potential issues proactively.
- Ensure compliance with external systems and regulatory changes.

Considerations to be taken into account moving forward -

Are there potential changes in public health systems or API providers that might affect functionality?

Could upcoming device software updates or performance issues on lower-spec hardware impact compatibility?

16 Waiting Room

Virus Simulation with Infection Rate Calculations:

A dynamic simulation feature to model the spread of viruses based on real-world infection rate calculations. This would provide users with an interactive, educational experience about how viruses spread and the impact of interventions like vaccines or social distancing. The feature would retain and revisit creative ideas that enhance the game's educational and interactive potential. This would requires advanced modeling

algorithms and potentially real-time data updates, which exceeded the current release scope

Advanced Personalization in Avatar Features:

Enhancements to the avatar system to allow greater customization, such as adjusting physical traits, clothing styles, or accessories based on player preferences. This feature could improve user engagement and immersion. Adding this features would need more UI/UX design work and backend adjustments, making it better suited for future versions.

Self-Adjusting Quiz Questions:

A quiz feature that adapts the difficulty of questions based on the user's performance. Players who perform well would face more challenging questions, while others would receive tailored questions to reinforce their understanding of core concepts. While this adds significant educational value, it demands additional testing, analytics integration, and question bank expansions to function seamlessly.

17 Ideas for Solutions

Game Engine for Simulation:

For future virus simulation features, consider integrating Unity or Unreal Engine for advanced visualization and dynamic modeling. Both engines offer strong support for real-time calculations and interactive displays.

When considering integrating a game engine into an existing application built with Angular (frontend) and Spring Boot (backend), several factors need to be evaluated to ensure compatibility, performance, and maintainability. Below are the key considerations:

1. Compatibility and Integration

Ensure the game engine integrates smoothly with Angular. Some engines like Unity or Babylon.js can run as embedded WebGL elements within Angular components.

Use Babylon.js or Three.js for lightweight 3D rendering directly in the browser. These are JavaScript libraries that work natively with Angular.

Ensure seamless data transfer between the Spring Boot backend and the game engine, especially for real-time interactions like loading user profiles, game progress, or quiz results. REST APIs would be ideal for this.

2. Performance

Game engines can be resource-intensive, especially if using advanced 3D rendering or physics simulations. Ensure the engine's rendering performance aligns with the browser's capabilities.

Use lazy loading in Angular to only load the game engine when required, reducing initial load time for the application.

If the game engine requires frequent backend communication (e.g., fetching game states or leaderboard data), ensure the Spring Boot backend can handle the additional traffic. Consider caching mechanisms (e.g., Redis) for high-demand endpoints.

18 Project Retrospective

The "Attack the Virus" project successfully combined education and interactivity, leveraging effective practices to achieve its goals. Jira Kanban boards provided an organized approach to task management, ensuring clarity and visibility into team progress. Feature ownership allowed individual developers to take responsibility for specific components, such as the leaderboard and virus simulation, which reduced dependencies and enhanced accountability. Daily standups on Discord facilitated open communication and early identification of blockers, maintaining alignment across the team. These practices supported an agile workflow that enabled iterative improvements based on user feedback, resulting in engaging features like personalized avatars, interactive quizzes, and a location-based clinic finder.

However, the project faced challenges that impacted overall efficiency. Pair programming, while fostering collaboration, occasionally created bottlenecks when tasks could have been parallelized. The one-week sprint cycles provided insufficient time for thorough development and testing, leading to rushed implementations and limited flexibility for addressing feedback. Additionally, scope creep from mid-sprint feature additions, such as expanding quiz content or gameplay enhancements, stretched the team’s capacity and disrupted prioritization.

To improve future projects, the team could adopt two-week sprint cycles, allowing more time for feature refinement and comprehensive testing. Introducing automated testing earlier in the development cycle would help identify issues quickly, freeing up time for manual testing of new features. Stricter scope management practices, such as maintaining a “waiting room” for future ideas, would prevent scope creep and ensure better focus on sprint goals. These refinements would enhance efficiency and create even more impactful outcomes.

VII Glossary

VIII References / Bibliography

IX Index