Ruben Martinez (Team Leader)

Alfonoso Hernandez (Planning Manager)

David Duru (Quality Manager)

Logan Armendariz (Support Manager)

Francisco Jimenez (Design Manager)

Caleb Lopez (Implementation Manager)

Jose Legarreta (Test Manager)

Professor Bhanukiran Gurijala

CS 4374/5374 Software Construction

2 November 2024

Team 3 Sprint 2 Project Report

**Project Explanation**

Muscle Mind is a quiz-based game that lets users create custom quizzes based on any topic they wish. The game allows users to create a maximum of 3 quizzes each with a maximum of 10 questions per quiz which can be useful for studying different categories. The quizzes are meant to test your knowledge and memory with wrong answers penalizing you with a very short, randomized workout. The main aim of the project is to provide a more active form of learning while also promoting general body health. The distribution of the game will be done through the internet and will allow users to create accounts to monitor progress and quizzes they have made. In its current state, it hosts a local session from the user's computer. There are plans to include more interactive graphics and animations to get users to become more engaged with the game itself. A look at how the game looks currently can be seen on pages 4-5.

Our backend is powered by Flask (Python) and MySQL, where we’ve set up a robust structure to manage user accounts, quizzes, and exercise data efficiently. We’ve split the project into two main subsystems: the Quiz Management Subsystem and the User Management Subsystem. This separation allows us to keep the codebase modular and flexible, making it easier to adapt, scale, and add features in future sprints. Each user can create a maximum of three quizzes, with each quiz containing up to ten questions, giving users a good balance between flexibility and focused content creation while maintaining a manageable load on the system.

On the frontend, we’re using HTML and CSS to build an intuitive and visually engaging interface that keeps users immersed. The UI is simple yet effective, making it easy for users of all levels to interact with the quizzes and track their learning progress. Our primary goal here is to ensure a seamless user experience, where users can quickly access quizzes, monitor their scores, and, if needed, take on those fun exercise challenges tied to incorrect answers.

Technically, we’re working with a shared MySQL database and utilizing pymysql to ensure smooth and efficient data transactions. For version control, we’re relying on GitHub, which is crucial for keeping track of changes, rolling back when necessary, and facilitating seamless team collaboration. To keep things organized and clear, we’ve implemented a Software Configuration Item (SCI) directory structure. This structure includes three main folders: /src for source code, /docs for documentation, and /test for our test suite, providing an organized view of the project and keeping each aspect of the system isolated and manageable.

Our team also follows a Configuration Control Board (CCB) Approval process for versioning, ensuring consistency and quality in updates. Major version changes represent significant feature updates, while minor versions handle bug fixes and code refactoring. This approach keeps everyone on the same page and provides a transparent history of changes.

To protect our data and ensure reliability, we’ve set up a full weekly backup system, with incremental nightly backups stored in the cloud. This backup structure safeguards against data loss, and we’ve integrated recovery protocols that allow us to restore the system seamlessly if needed. The backup plan is critical to maintaining user trust and ensuring uninterrupted access, particularly as the project scales.

Overall, this project is part of our CS 4374 Software Construction course, and it’s an opportunity for our team to bring together various skills in software engineering, database management, and team collaboration. We’re proud of the progress we’ve made so far and excited to see Muscle-Mind evolve into a meaningful, impactful educational platform for our users, where learning about fitness and wellness becomes both informative and physically engaging.

Figure 1 - Log In Page

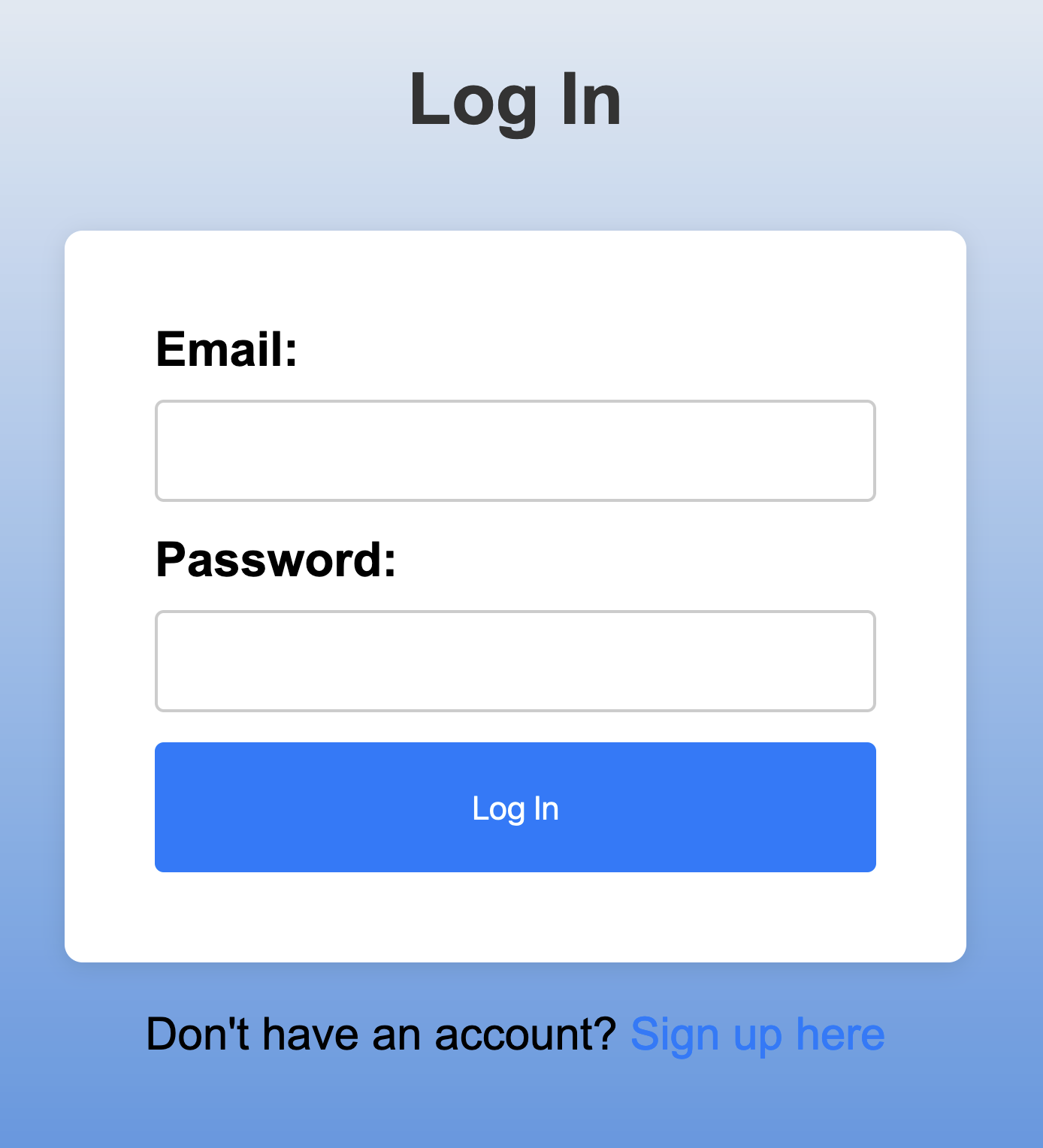


Figure 2 - Main Menu

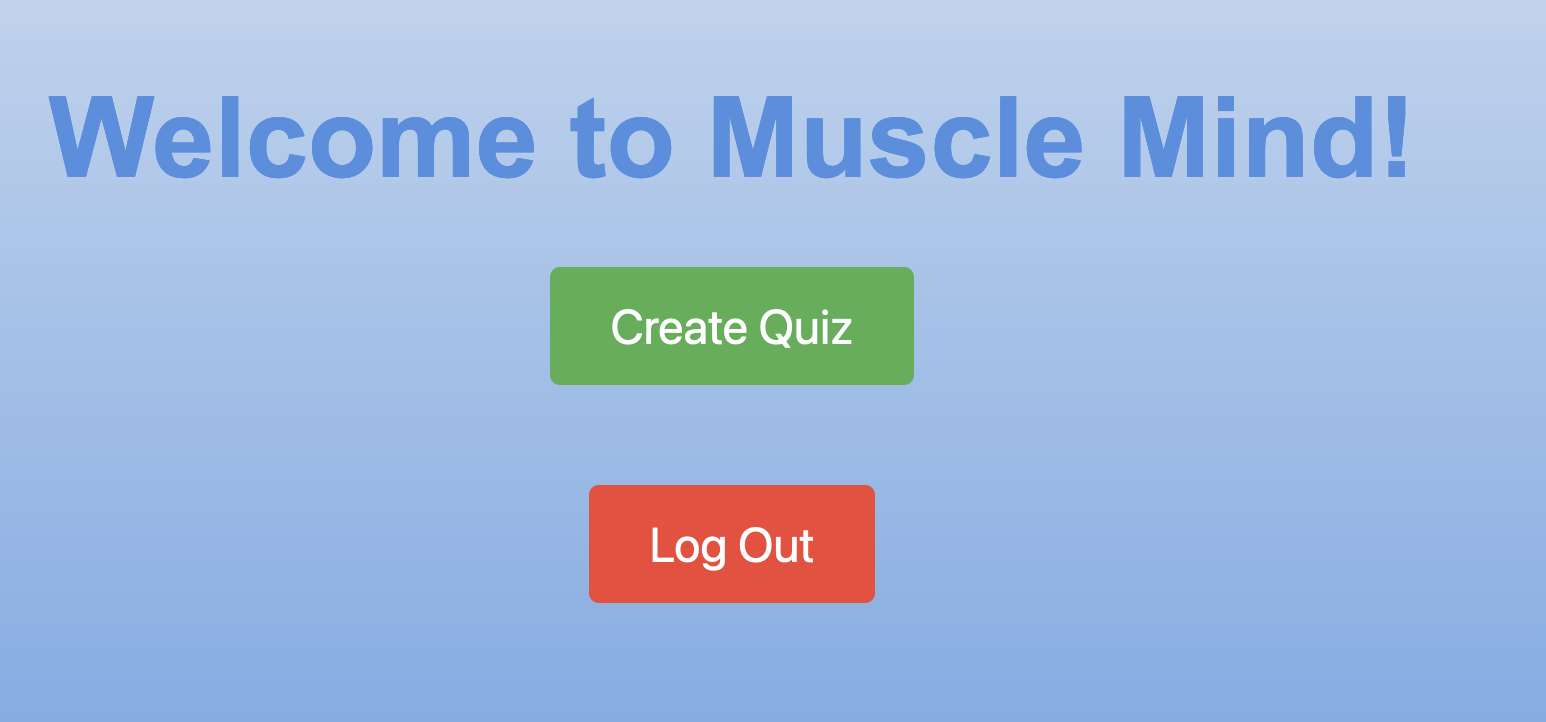


Figure 3 - Quiz Creation Page

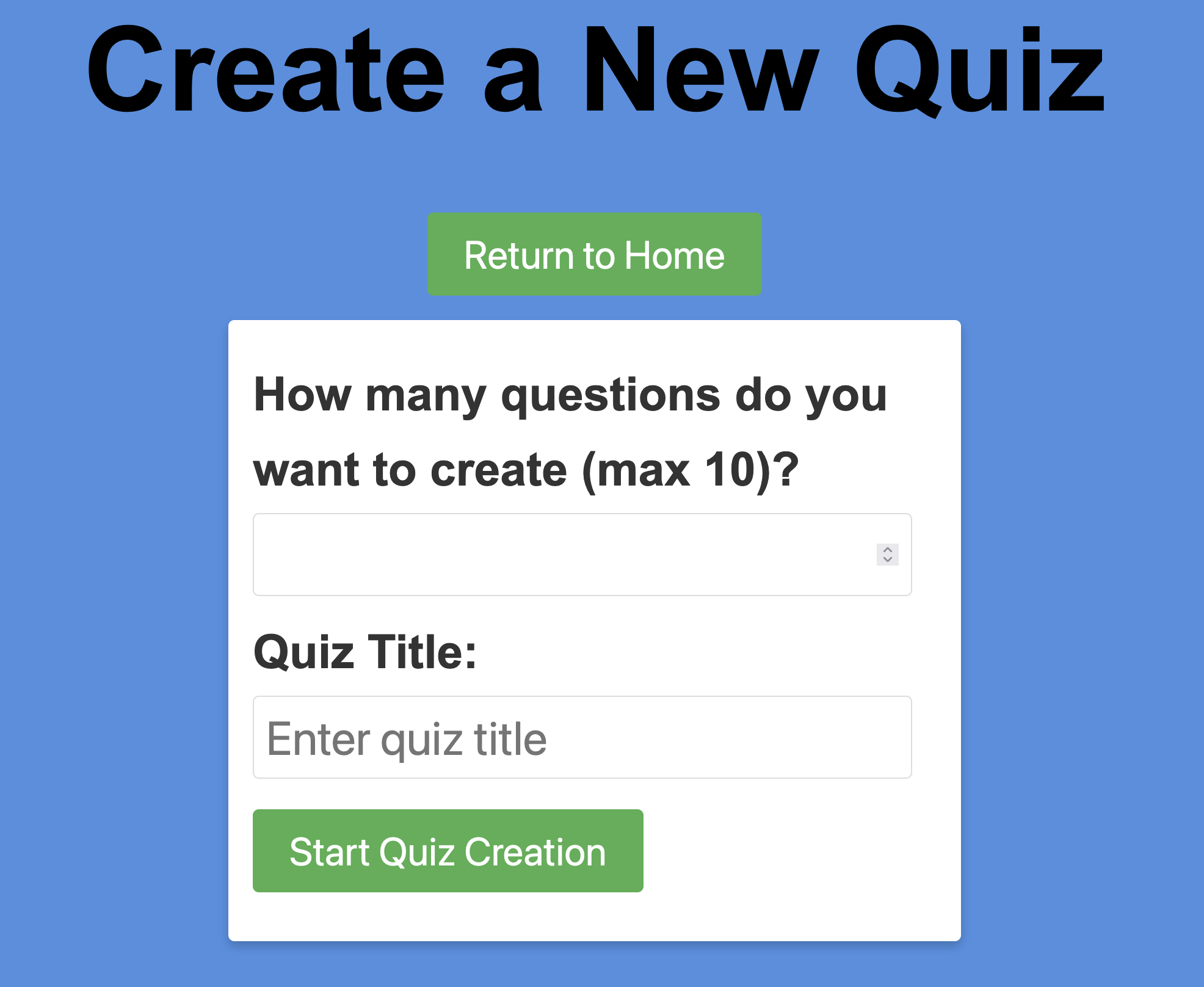


Figure 4 - Main Page with a Quiz Present

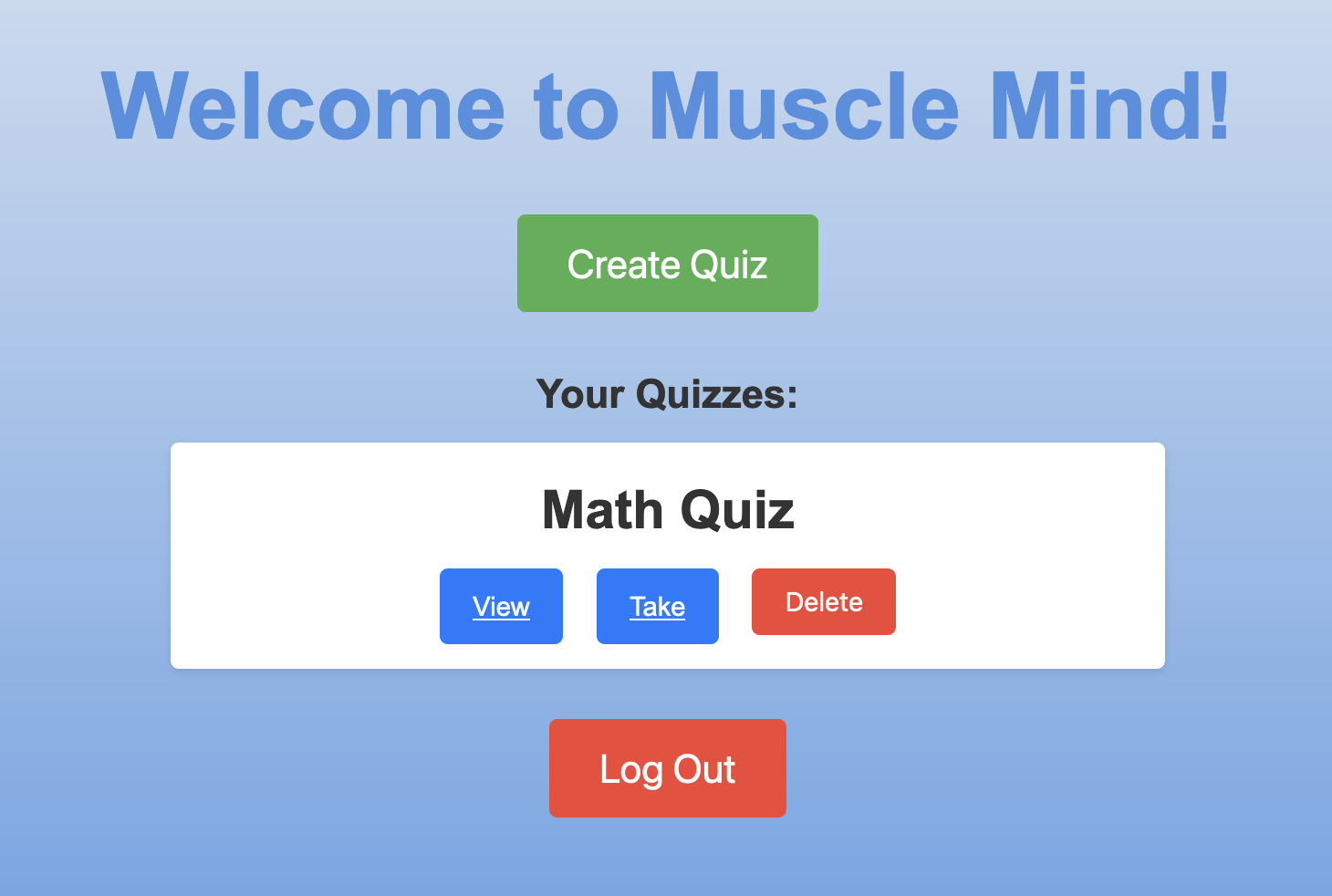


Figure 6 – Score

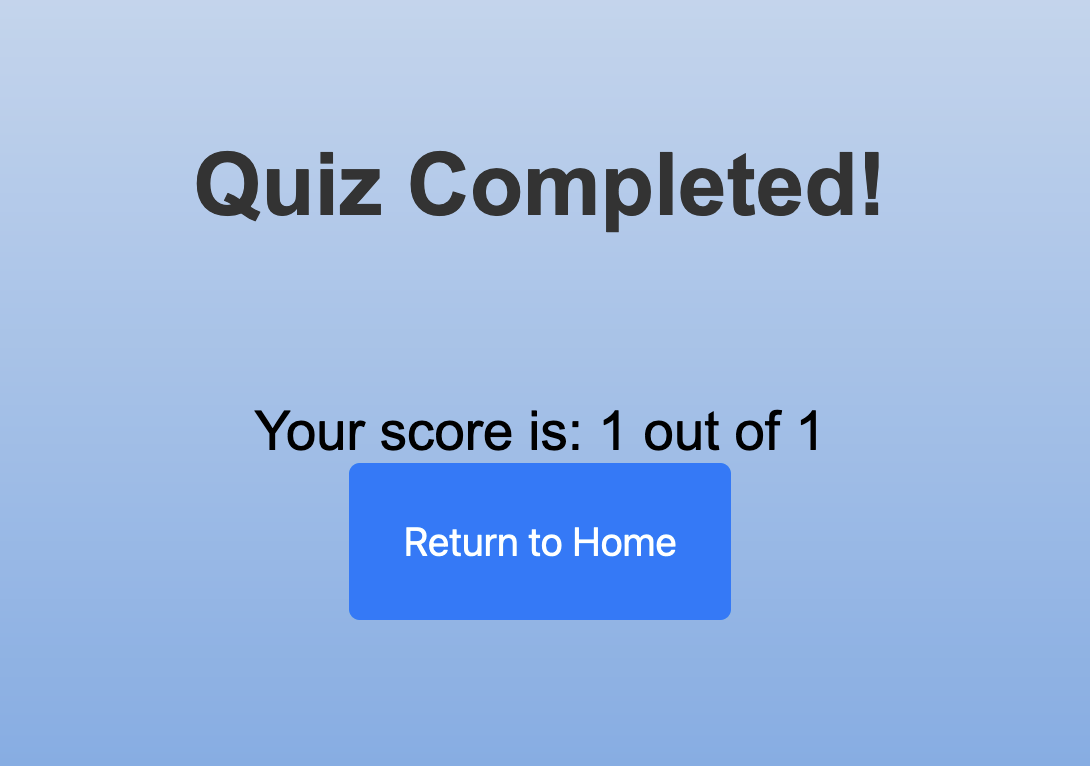


Figure 7.1 & 7.2 – Quiz Penalty Timer

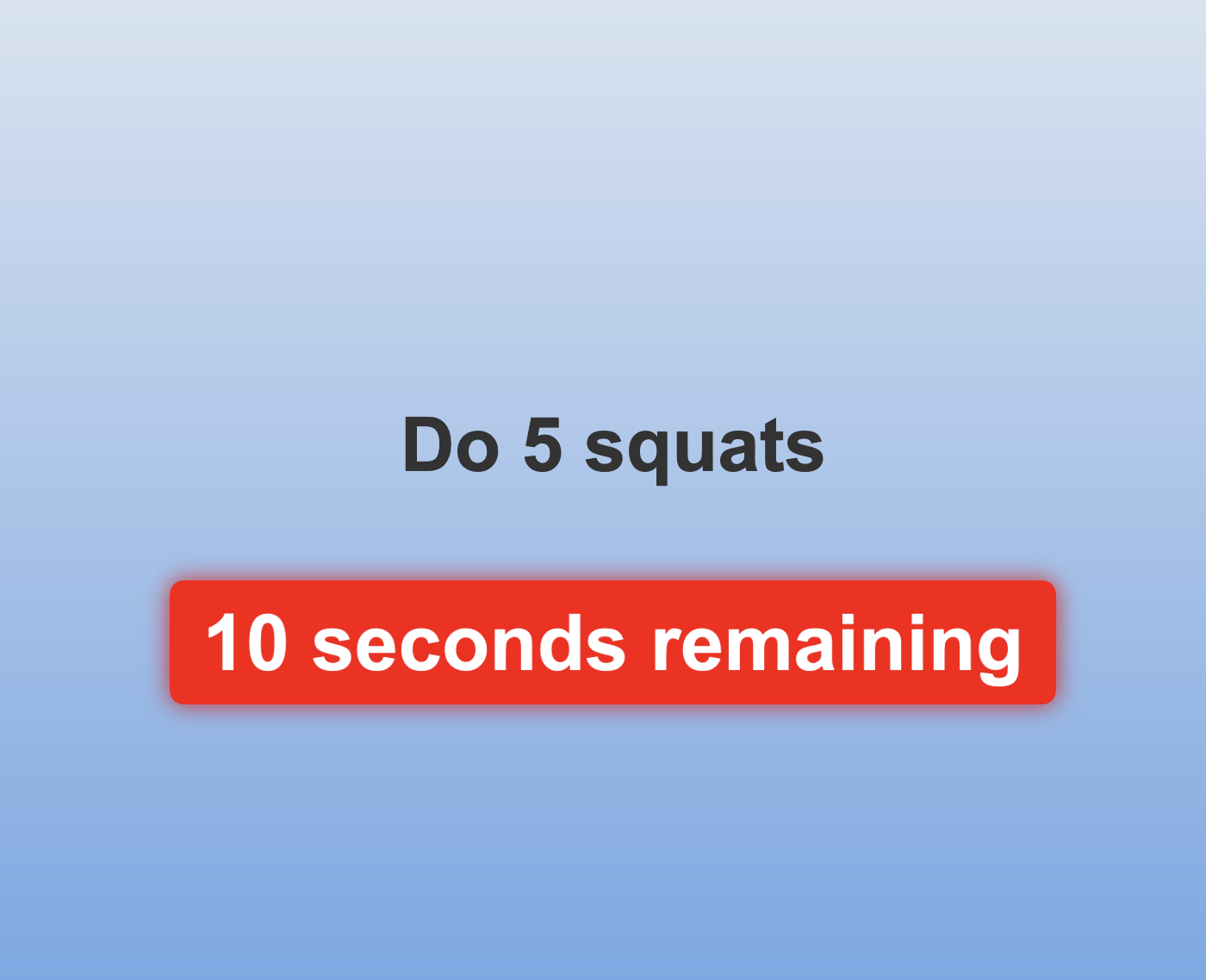
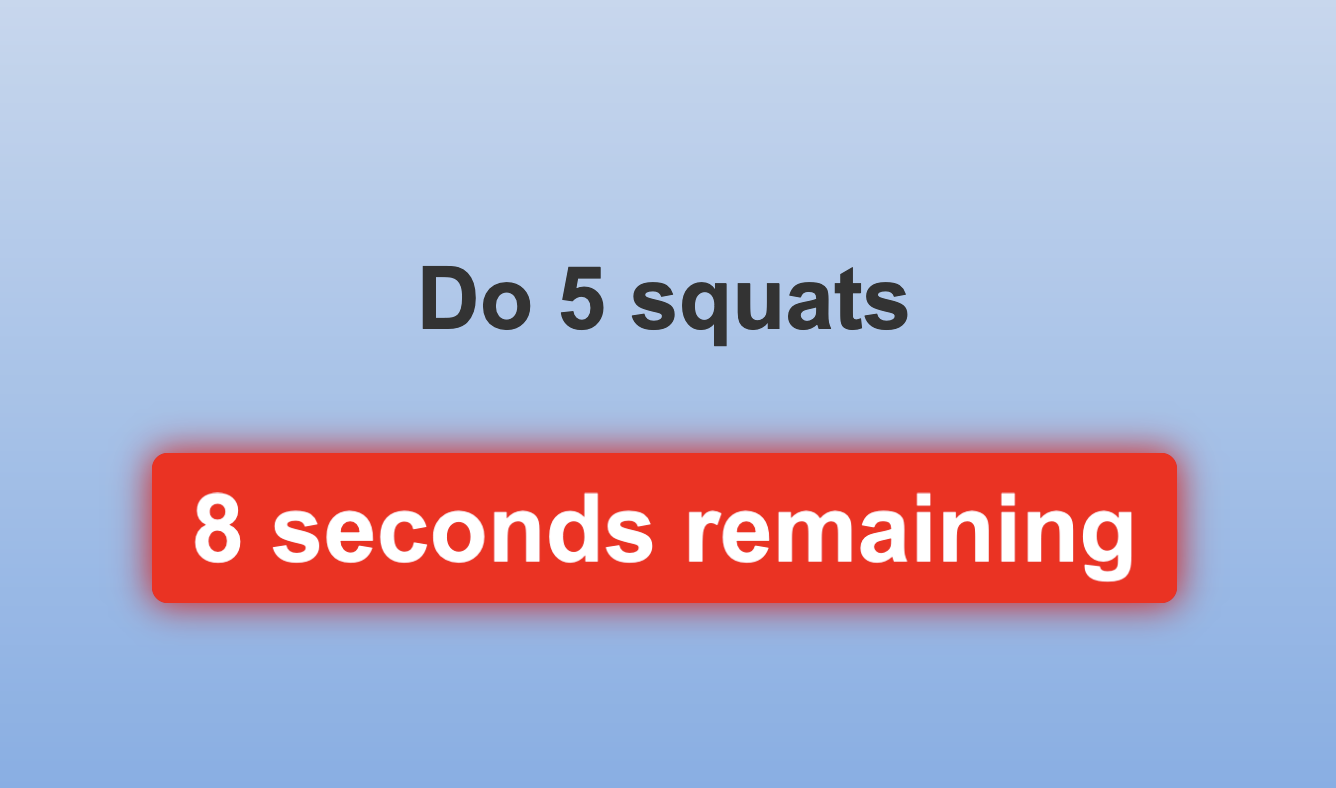
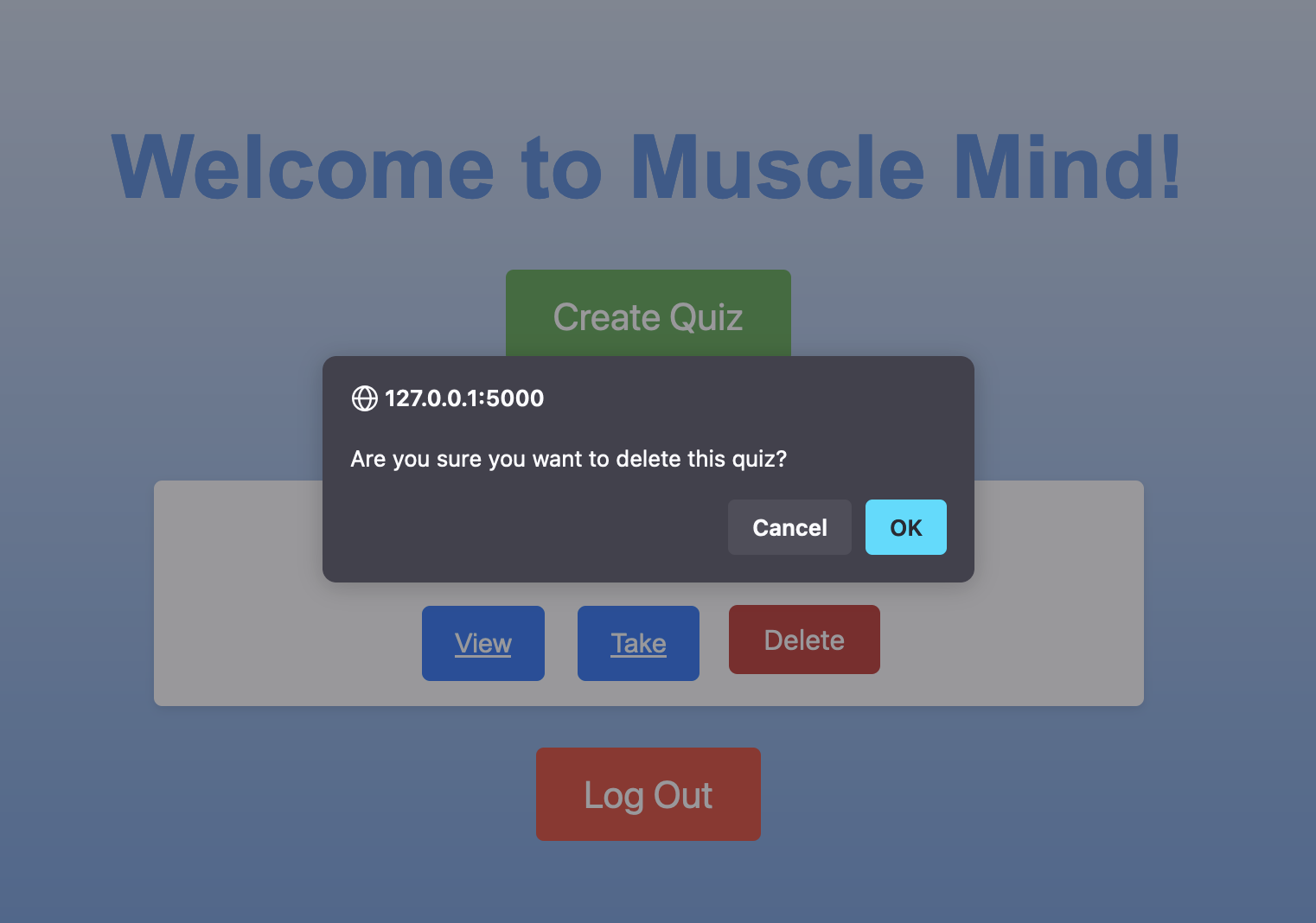
[7.1] [7.2]

Figure 8 - Quiz Deletion Pop up



**Solution Design**

The team approached the Muscle-Mind solution design with a structured, collaborative, and iterative methodology. Ruben Martinez, the team leader, led the team in initial brainstorming sessions to map out the project requirements and define our main objectives. Alfonso Hernandez, as the plan manager, coordinated the project timeline and ensured that every design iteration aligned with our milestones. Francisco Jimenez, the design manager, played a critical role in translating our ideas into a coherent structure, guiding the team’s use of LucidChart to create clear and effective diagrams. Together, we established a strong foundation by breaking down the platform’s functionality into two primary subsystems: the Database Management Subsystem and the Quiz Management Subsystem.

One of our main design choices was to adopt a modular architecture to facilitate maintenance, scalability, and separation of concerns. We assumed that our user base would steadily grow and that the system might need to expand in the future to include additional features. This led us to create distinct subsystems, each responsible for a focused set of operations. The Database Management Subsystem is in charge of securely handling data operations, including user authentication, session management, and retrieving quiz data. Meanwhile, the Quiz Management Subsystem manages the creation, submission, and scoring of quizzes. We chose this separation because it allowed us to streamline each component’s responsibilities, reducing complexity and ensuring that future development could proceed in a modular fashion.

To make the design process more effective, we implemented "Design by Contract" principles, defining each class's responsibilities using informal English descriptions. These contracts set clear requirements for each component, specifying what each class must accomplish and the guarantees it provides upon execution. For instance, the DatabaseManager class has a "ManageDatabaseOperations" contract, which outlines the required actions for establishing and closing connections, executing SQL queries, and committing changes to the MySQL database. This method allowed us to establish a common understanding across the team about the role of each component, enabling a smoother transition from design to code.

During the design process, several assumptions shaped our approach. We assumed that the system needed to handle sensitive user information, such as login credentials and quiz scores, with a high level of security and integrity. This influenced our decision to centralize database interactions within the DatabaseManager class, ensuring a single point of control over data storage and retrieval. Additionally, we assumed that user interactions with quizzes would vary between creation and submission stages, which led us to establish QuizCreationManager and QuizSubmissionManager classes to handle these distinct tasks. By splitting these functionalities, we could address each phase of the quiz lifecycle with targeted methods and validations, ensuring the robustness of the overall system.

The App class serves as the mediator between the two primary subsystems, facilitating communication between the Database Management Subsystem and the Quiz Management Subsystem. This central class routes requests from the user to the appropriate subsystem, ensuring a smooth flow of information across the platform. By designating the App class as the mediator, we simplified interactions between subsystems, reducing dependencies and allowing each subsystem to focus on its responsibilities without needing to handle cross-subsystem interactions directly. This structure also enhances scalability, as new features can be introduced within each subsystem without disrupting the core App functionality.

The completeness of our design is evident in its coverage of all identified features. By specifying contracts for each class, we ensured that every critical function has been accounted for and can operate within well-defined boundaries. The UserManager manages user sessions and authentication, while the QuizRetrievalManager provides users with access to quiz content, ensuring that the Database Management Subsystem can handle all data-related operations efficiently. Similarly, the Quiz Management Subsystem’s components cover the entire quiz lifecycle, from quiz creation with QuizCreationManager to submission and scoring with QuizSubmissionManager. These contracts not only guide our coding efforts but also help us ensure the integrity and reliability of each component, providing a solid foundation for future feature expansions.

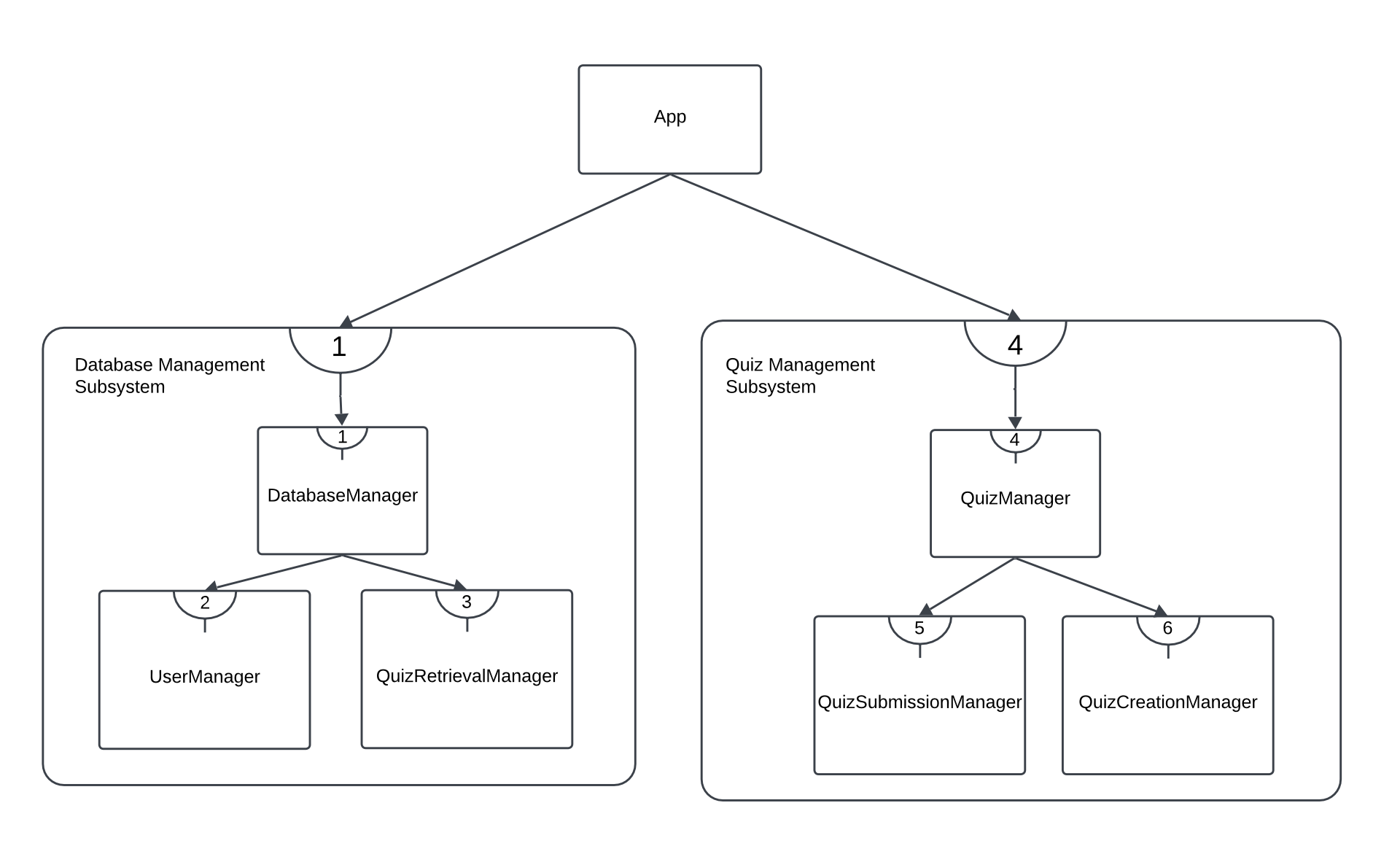


Figure 9: Refactored design after Design Reviews

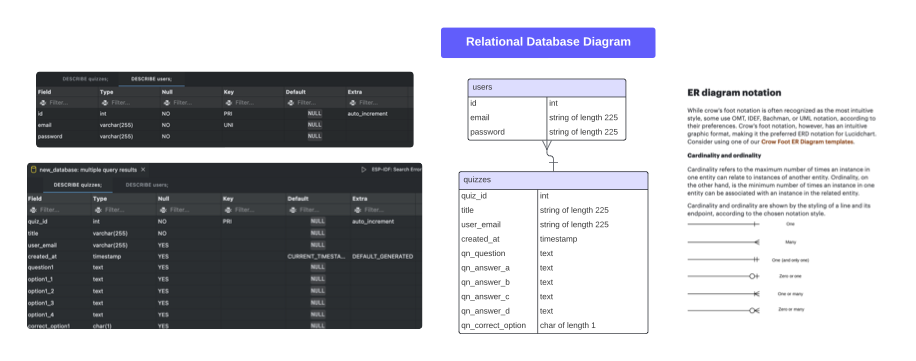


Figure 10: Relational Database Graph

Our team chose to bridge the CRC cards to code by combining Design by Contract principles with modular object-oriented programming. Using contracts helped us define each class's preconditions and postconditions, setting clear expectations for the code implementation. The informal English descriptions were invaluable, as they allowed the entire team, regardless of coding background, to understand what each method must achieve and what guarantees it provides. This approach also aligned with our modular architecture, making it easy to implement, test, and expand each class independently. By focusing on contracts and modularity, Ruben Martinez, Alfonso Hernandez, and Francisco Jimenez could maintain cohesion across the design, ensuring that every part of the project contributes effectively to the system’s overall functionality.

In summary, the Muscle-Mind solution design reflects a carefully structured approach, grounded in assumptions about scalability, security, and modularity. Through an iterative process and the guidance of each team member in their specialized roles, we developed a complete and robust system design that meets all identified requirements while allowing for future growth. This collaborative, contract-driven approach enabled us to transition smoothly from CRC cards to code, ensuring that our implementation will meet the high standards expected of a dynamic fitness and wellness platform.

**Implementation**

The solution design of the Muscle-Mind project was carefully translated into implementation by following a structured and disciplined approach. Utilizing the Design by Contract methodology, we ensured that each function in our code was documented with preconditions and postconditions. At the beginning of every function, we included comments with @requires and @ensures annotations to specify what each function expected and guaranteed upon completion. This approach allowed us to create reliable code that adhered to the expected functionality outlined in our design document, making the process smoother and more predictable. These annotations also served as informal contracts that guided our development process, ensuring that each function met its defined role within the overall system.

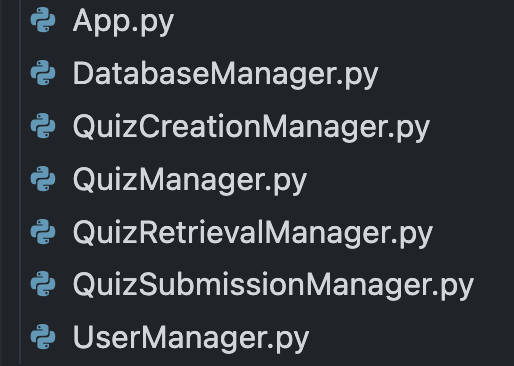


Figure 11: Class files from the collaboration graph in Design Document

The implementation manager, Caleb Lopez, worked closely with Ruben Martinez, the team leader, to bring the features specified in the design document to life. They followed the system collaboration graph, as shown in the images provided, to ensure each class was implemented correctly and interacted as expected. This graph guided the organization of our files, resulting in separate Python modules for each core class: App.py, DatabaseManager.py, QuizCreationManager.py, QuizManager.py, QuizRetrievalManager.py, QuizSubmissionManager.py, and UserManager.py. Each of these modules corresponds to classes and components defined in our design, maintaining the structure we had initially planned. By keeping these components separate, we preserved the modularity and maintainability of the codebase, aligning closely with the design document's architecture.

To manage the development process effectively and backtrack in case of defects, we used GitHub as our version control system. This allowed us to monitor changes across the codebase, making it easy to roll back if any issues arose. GitHub served as an essential tool for collaboration and debugging, ensuring that each team member could contribute and review code changes without disrupting the project’s progress. With GitHub, we could experiment with implementations, validate the results, and make adjustments as necessary, all while maintaining a complete history of changes.

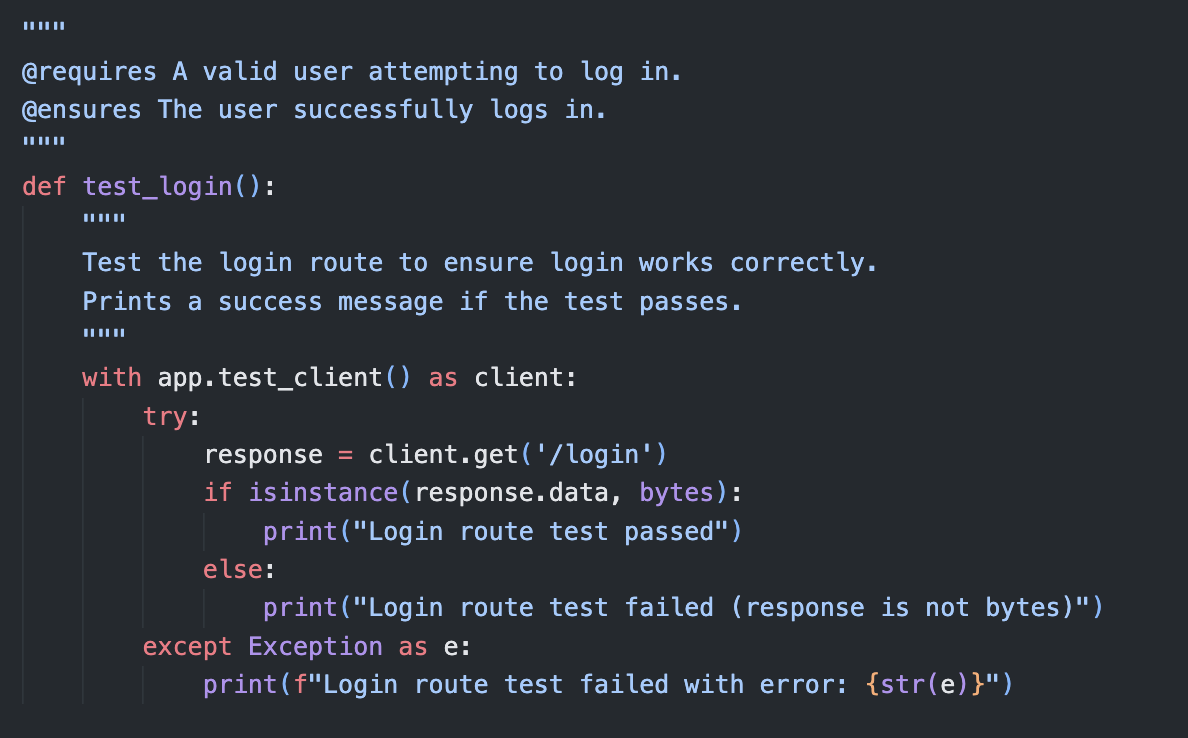


Figure 11: Adding the Design by Contract using informal notation

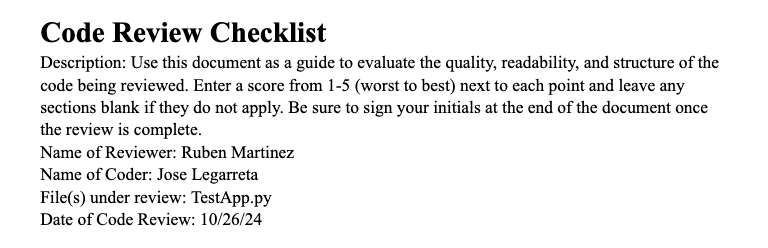
In addition to implementing each function with precise annotations, we adhered to the contracts specified in our design for every class. For instance, the DatabaseManager handled database interactions according to the "ManageDatabaseOperations" contract, while the UserManager was responsible for managing user sessions as per the "ManageUserSessions" contract. Each function within these classes was carefully documented with @requires and @ensures to define input expectations and output guarantees, thereby reinforcing our Design by Contract approach. This disciplined implementation method allowed each class to interact seamlessly, fulfilling the modular design goals we established at the project’s outset.

Overall, by translating our system collaboration graph and Design by Contract approach into code, we maintained a clear connection between the solution design and the implemented code. This structured approach, supported by GitHub and Caleb Lopez’s diligent management, enabled us to create a robust and organized codebase that faithfully represents the Muscle-Mind project’s design principles and functionality.

.

**Reviews**

Team 3 approached the design and code review process with a systematic, detailed, and collaborative mindset. The team was led by Ruben Martinez, with Alfonso Hernandez serving as the design manager, and Caleb Lopez as the implementation manager. Each of us played a specific role in ensuring that the Muscle-Mind project would be robust, modular, and scalable, meeting the project’s requirements while adhering to best practices. Throughout the project, we relied on checklists for both design and code review, updated to reflect lessons learned and feedback from the TA, Veronica Rivas.



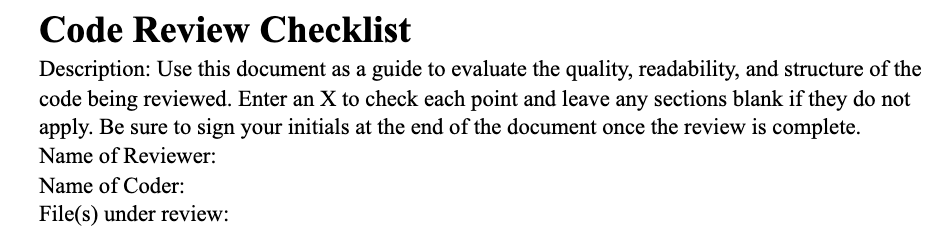


Figure 12: Comparison of updated code review document (top) and old code review document (bottom).

Initially, our review checklists used a simple checkbox system, where reviewers could mark an “X” if a specific criterion was met. However, after discussing ways to improve feedback quality, we decided to shift to a rating system, scoring each item from 1 to 5, with 1 indicating significant need for improvement and 5 indicating excellence. This change allowed us to provide more precise feedback and better track areas that required attention. The updated format helped the team identify specific strengths and weaknesses in each review area, allowing for more focused improvements. The goal was to make the review process more insightful, helping us enhance the design and code in iterative cycles.

To ensure our design was as solid as possible, we followed a rigorous process that incorporated feedback from our TA, which highlighted critical areas needing refinement. Our TA’s feedback included the following points: (1) avoid self-collaborations, as a class should not collaborate with itself, (2) assign unique contract numbers, as some contracts in our CRC cards initially had duplicate numbers, and (3) include method signatures in our CRC cards with pre- and post-conditions to clarify each method’s purpose and requirements. This feedback drove us to reconsider and refine our initial design choices, focusing on creating a clear, modular, and cohesive structure.

In response, we restructured the system, reducing the number of subsystems from six to two: the Database Management Subsystem and the Quiz Management Subsystem. This change simplified our design and enhanced modularity by defining clearer boundaries for each part of the system. By consolidating related components, we created a more manageable architecture where each subsystem had specific responsibilities. The Database Management Subsystem focused on data handling tasks like storing user information and quiz data, while the Quiz Management Subsystem managed the creation, submission, and scoring of quizzes. This streamlined approach eliminated unnecessary complexity and allowed for better maintainability and scalability, ensuring that future features could be added without major restructuring.

To verify the updated design, Ruben Martinez and Alfonso Hernandez conducted an official design review on October 20th. Using the updated Design Review Checklist, Ruben and Alfonso evaluated each aspect of the design, including Design Clarity and Design Structure and Modularity. Under Design Clarity, they assessed whether the design was easy to understand and logically clear. They reviewed the diagrams and visual aids to ensure they were complete, accurate, and labeled correctly, scoring this criterion highly due to the clarity achieved through tools like LucidChart. Additionally, they checked that assumptions and constraints were documented, which helped ensure that any future team members could easily understand the context and limitations of the design. For Design Structure and Modularity, Ruben and Alfonso reviewed the system's adherence to single responsibility principles, ensuring that each component focused on a specific function without excessive overlap. Each component's design was also evaluated for loose coupling, allowing us to score highly in areas related to reusability and modularity.

One key structural component was the App class, which served as the mediator between the two subsystems. By centralizing the communication between the Database Management Subsystem and the Quiz Management Subsystem in the App class, we simplified the design, reducing cross-dependencies and creating a clear interaction point for all system operations. This approach ensured that each subsystem could operate independently without needing to manage complex cross-subsystem communications directly. This mediator role of the App class became essential for maintaining clarity in our design, as it allowed each subsystem to handle its responsibilities while relying on the App class to route user requests and responses efficiently.

On October 25th, the team reconvened for a follow-up review session to assess the progress made in implementing the design. By this point, we were ready to conduct code reviews, ensuring that our transition from design to code was smooth and consistent with our original goals. During this meeting, Ruben and Alfonso reviewed the work completed so far and validated that the core design had been effectively translated into code. This preparation was crucial before handing off implementation responsibilities to Caleb Lopez, the implementation manager, who would lead the coding phase. Conducting this design review allowed us to confirm that our updated modular structure and contract-based approach were reflected accurately in the codebase.

Once Caleb began the implementation phase, he adhered to the principles established in our Design by Contract methodology. To enforce this, every function in our code was annotated with `@requires` and `@ensures` comments, specifying each function's preconditions and postconditions. These annotations provided informal contracts for each function, clarifying the inputs and outputs expected and ensuring that each function behaved predictably. For instance, in the DatabaseManager class, methods related to database interactions were annotated to indicate the required database connection state (`@requires`), as well as the successful execution of queries and closure of connections (`@ensures`). This documentation not only reinforced the Design by Contract approach but also made the code easier to understand and maintain by setting clear expectations for each function.

The team conducted code reviews on November 1st, using the updated Code Review Checklist to evaluate the quality and structure of the implementation. Ruben Martinez conducted a detailed review of the `TestApp.py` file written by Jose Legarreta. For this code review, the checklist provided a framework to evaluate Quality, Readability, and Structure and Modularity. Ruben assessed each function to ensure it adhered to the `@requires` and `@ensures` annotations, confirming that the implementation followed the pre- and post-conditions specified in our design document. Under Quality, Ruben examined the accuracy of each function, ensuring that the logic was correct and performed as intended. For Readability, he looked at code formatting, variable names, and inline comments, all of which contributed to the overall clarity of the code.

In checking Structure and Modularity, the team verified that each class and function adhered to the single responsibility principle, meaning that each component performed only one specific task, without overreaching into other areas. This helped us maintain a clean, modular codebase where each class fulfilled its intended role without introducing unnecessary complexity. The App class remained the central mediator, effectively routing requests and serving as the interaction layer between the Database Management and Quiz Management subsystems. The use of the GitHub platform for version control enabled us to track changes systematically and provided a means to revert to previous versions if defects were identified during the review.

By following these thorough review practices, meeting regularly every Friday, and applying both design and code review checklists, the team was able to maintain a high standard throughout the development process. Each review allowed us to identify areas needing refinement, and the updated 1-5 rating scale provided precise feedback that guided iterative improvements. Through these steps, the team ensured that the Muscle-Mind project met its design goals, producing a modular, maintainable, and scalable codebase that aligned with the original project requirements.

**Testing**

To verify the correctness of our Muscle-Mind implementation, the team adopted a comprehensive testing strategy that ensured every aspect of the platform was functioning as intended. We designed a structured test plan focused on validating each system component, ensuring data integrity, and verifying that the platform’s interactive elements operated seamlessly. Our approach included various testing levels—function-specific tests, integration tests, and boundary tests—all of which collectively provided complete coverage of the system’s functionality.

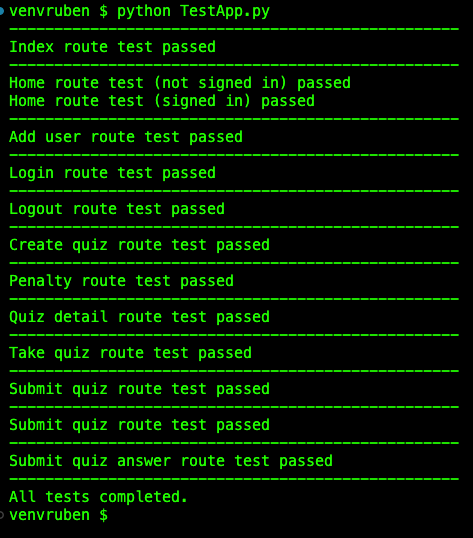


Figure 13: Screenshot of tests running and results.

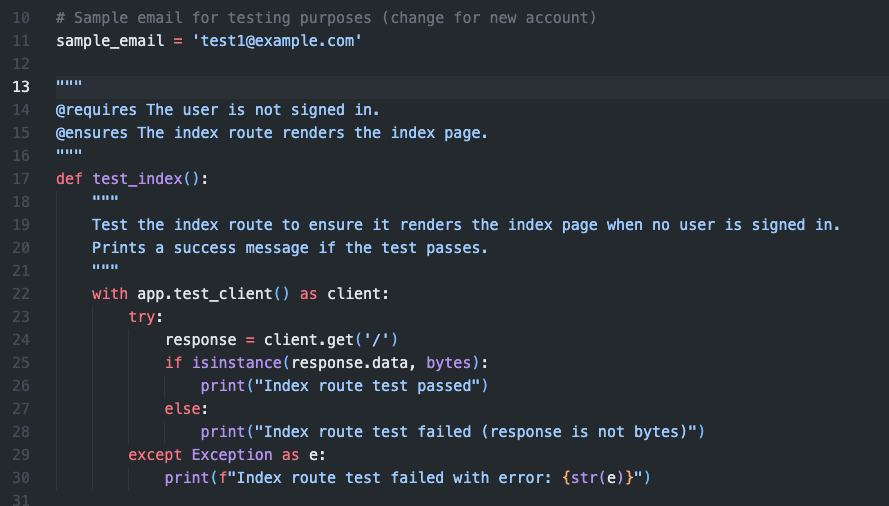


Figure 14: Source code translation for the test\_index() test case, which can be found in test document.

Since we had access to the source code, our team used white box testing to test the internal workings of each function. This approach allowed us to inspect and validate each code path, checking for logical accuracy and proper handling of data. Each test case was meticulously designed to address specific functionalities of the Muscle-Mind project, such as user account creation, login, quiz creation, quiz submission, and penalty application for incorrect answers. For each test case, we included @requires and @ensures annotations to specify preconditions and postconditions, following our Design by Contract methodology. These annotations clearly documented the expected state of the system before and after each test, providing a transparent reference for validation and debugging.

The purpose of our test plan, as outlined in our document, was to create a roadmap for verifying that the platform functions as expected, ensuring both stability and reliability. Given that Muscle-Mind is a fitness quiz platform aimed at engaging users in learning while providing a seamless experience, it was critical to validate each interactive component rigorously. Our test cases targeted essential features of the platform, including account creation, user login, session management, quiz creation and retrieval, quiz submission, and data integrity for stored results. For instance, test cases like test\_add\_user and test\_login checked that users could successfully register and log in, while test\_create\_quiz and test\_submit\_quiz confirmed the quiz creation and submission flows worked as intended.

To guarantee that our system was robust under real-world conditions, our test approach included three core levels of testing: Function-Specific tests, Integration tests, and Boundary tests. Additionally, our source in the TestApp.py file had a username global variable constant that could be changed, and the code would actually create an account and quizzes which can be viewed from the database! Without anyone creating an account, we wrote the code so that it attempts to perform all the actions a real user would do through the user interface.

Function-Specific Tests: These tests targeted individual functions and routes within the Muscle-Mind system. We tested routes such as test\_index, test\_home, test\_add\_user, and test\_login, ensuring that each route performed its designated function accurately. For example, test\_index verified that the index page rendered correctly, while test\_home checked that the home page responded appropriately based on the user's sign-in state. Function-specific tests helped us isolate potential issues within individual components, making it easier to address any bugs at the source.

Integration Tests: In addition to testing isolated functions, we performed integration tests to verify that various parts of the system interacted smoothly. These tests simulated typical user workflows, such as creating a quiz, taking the quiz, and submitting answers. By combining multiple functions into cohesive workflows, we could validate that data was managed correctly between different system components, ensuring a consistent user experience. For example, tests like test\_create\_quiz, test\_take\_quiz, and test\_submit\_quiz simulated the end-to-end process of a user interacting with quizzes, checking that data flowed seamlessly from quiz creation to submission and score calculation.

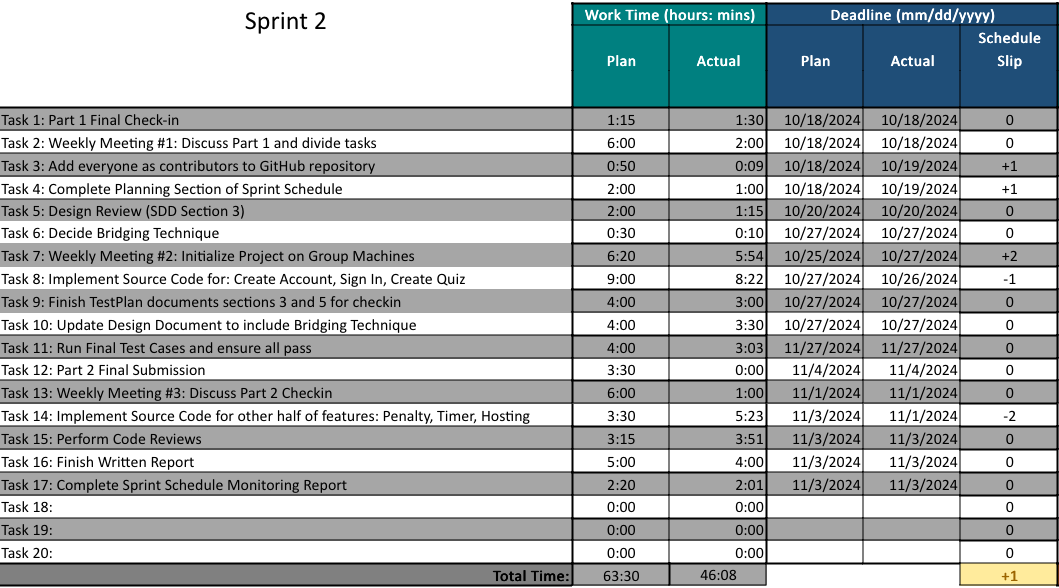
Boundary Tests: Our test plan also included boundary tests, which evaluated the system's resilience under edge cases and stress conditions. Boundary tests examined how the platform handled scenarios at the limits of its intended functionality, such as applying penalties when users answered questions incorrectly. For instance, test\_penalty validated that the system applied exercise penalties accurately based on incorrect answers, ensuring that the penalty functionality worked as expected under different conditions. These tests were essential for confirming the platform’s stability and ensuring that it would respond appropriately to user actions at all operational boundaries.

Our testing process was organized within the TestApp.py file, where we outlined each test case and structured them with @requires and @ensures comments. For example, test\_index had an @requires comment specifying that the user should not be signed in, with an @ensures comment confirming that the index page renders as expected. This approach allowed us to document each test case transparently, ensuring consistency across our testing suite. The detailed comments within each test case also served as an invaluable tool for understanding the purpose and expected outcomes, helping all team members follow the logic behind each test.

Moving forward, we plan to incorporate Pytest, a powerful automation tool, to streamline our testing process. Pytest will allow us to automate the execution of these test cases, making it easier to run comprehensive test suites regularly as we update and expand the platform. Automation with Pytest will not only save time but also reduce human error in test execution, ensuring that our system remains reliable and fully functional over time. By automating our test cases, we’ll be able to execute them as part of our CI/CD pipeline, ensuring consistent validation with every code change and preventing regressions.

In conclusion, our testing strategy provided a robust framework for verifying the Muscle-Mind platform's functionality, integrating thorough function-specific, integration, and boundary tests. The white-box testing approach allowed us to delve into each function’s logic, ensuring correctness at every level. By combining this with planned automation using Pytest, we are confident that Muscle-Mind will deliver a secure, reliable, and engaging user experience, fully aligned with our project’s goals.

**Sprint Retrospection**

Figure 15: Team 3 Sprint Schedule for sprint 2.

In Part 2, our team built upon our established workflow of assigning tasks based on each member's expertise and role, aiming to align responsibilities with strengths for optimal results. This sprint was particularly focused on testing and refining our design, which led us to emphasize roles that aligned with these tasks. We placed Jose Legarreta, our designated test manager, in charge of the testing document. His responsibility was to create and update test cases that rigorously evaluated the functionality of each feature in Muscle-Mind, leveraging his expertise to streamline the testing process and ensure thorough coverage. On the design side, Francisco Jimenez, our design manager, collaborated closely with me, Ruben Martinez, on updating and expanding the Software Design Document (SDD). Our primary focus was to refine the design details and incorporate bridging techniques as specified by the project requirements, ensuring that our documentation accurately represented the intended system functionality.

The division of roles allowed us to handle tasks efficiently. By assigning more technical, design-centric tasks to Francisco and me, and test-focused responsibilities to Jose, we ensured that each team member could deliver high-quality work within their area of expertise. To keep our workflow organized, we set multiple deadlines for major tasks throughout the sprint. This structure allowed us to regularly review each team member’s progress, catch any issues early on, and make adjustments to our plans as needed. For example, we scheduled initial deadlines on October 27th for preliminary drafts of sections 3 and 5 in the test document, and a final deadline for November 4th to complete the document. We also set an October 27th deadline for key feature implementations, such as user account creation, sign-in, and quiz functionalities, with additional updates following shortly thereafter.

One of our main strategies for this sprint was to hold frequent meetings to stay on track and address any arising issues in a timely manner. We scheduled weekly check-ins on Fridays, which proved valuable for reviewing completed work, providing constructive feedback, and reallocating resources if any team member faced unexpected challenges. For instance, Francisco, who was handling a heavy course load of 16 credit hours, found it difficult to complete his portion of the design updates on time. He communicated this to the team, allowing me to step in and finalize the design document before the October check-in. Francisco’s efforts to stay engaged and participate in the check-ins, despite his academic responsibilities, had a positive impact on our overall progress. The team's willingness to support each other and adapt to individual needs ultimately contributed to maintaining our project timeline.

To enhance coordination, I sent a structured message on October 20th in our group chat, outlining the various deadlines and tasks due by October 28th and 29th. This included completing the sprint schedule, implementing half of the source code features, updating the SDD with bridging techniques, and finalizing sections 3 and 5 of the test plan. Additionally, I provided a setup guide on GitHub, including detailed instructions for database connection and environment configuration, to help teammates get their development environments running smoothly. Setting staggered deadlines and providing clear resources allowed us to review and refine our work progressively, minimizing last-minute stress and ensuring a more balanced workload distribution.

Despite these proactive measures, we encountered some challenges that required us to make adjustments. During the design review, our TA provided feedback pointing out that classes should not collaborate with themselves, a detail we initially overlooked. This insight prompted us to revisit and refine portions of the design in the SDD, enhancing the clarity and logical structure of our documentation. We focused on aligning our design more closely with the actual implementation, ensuring that the source code reflected the architecture we envisioned. By refining class interactions and eliminating self-collaborations, we achieved a cleaner, more accurate representation of the Muscle-Mind system. This experience underscored the importance of external feedback, and we learned to approach our design with an eye for consistency between documentation and implementation.

GitHub played a critical role in helping us manage the adjustments and track changes throughout this sprint. Version control allowed us to efficiently implement design modifications without risking disruption to existing functionality. By creating a clear record of our changes, we could roll back any errors and review past versions as needed, which proved valuable during debugging sessions. In addition to design updates, we introduced a coding standards document specifically for CSS, addressing the need for consistent styling guidelines that became more relevant as we progressed in UI development. We took this opportunity to expand the coding standards document to include error handling practices in Python, ensuring uniformity across our codebase for handling unexpected scenarios.

One important update to our Software Configuration Management (SCM) document was the addition of definitions for major and minor changes. Our TA, Veronica Rivas, pointed out that our original SCM document lacked clear criteria for classifying changes, which could lead to confusion during version management. By defining these terms, we provided clarity around what constitutes a major versus a minor change, ensuring that future modifications could be categorized and tracked effectively. This addition enhanced our documentation quality and reinforced consistency across our deliverables, helping us present a more polished and professional submission for Part 2.

A significant challenge we encountered during this sprint involved cross-platform setup difficulties. With team members using different operating systems, such as Logan Armendariz on macOS Sierra (an older version) and David Duru on Windows, compatibility issues emerged during the setup process. Logan's older macOS version struggled with some of the latest dependencies, requiring manual adjustments to locate compatible versions of necessary packages. Meanwhile, David's Windows setup required additional configuration for environment variables and library management. Despite preparing a detailed setup guide with screenshots and instructions, I spent considerable time troubleshooting these issues for each member individually. This diversion of time from feature development highlighted the need for OS-specific setup instructions. Moving forward, we plan to incorporate OS-specific guidelines into our setup documentation to streamline the process and make it more efficient for all team members.

Additionally, database integration across local environments proved challenging. Although I had set up the database and provided connection instructions, some team members experienced issues with configuration and network settings. Logan encountered firewall restrictions that prevented him from accessing the database, while Caleb had trouble with his credentials. Troubleshooting these issues remotely was time-consuming and led to delays in testing, as team members could not fully interact with the database. As a solution, we are considering establishing a shared test environment or using a local network configuration to simplify database access for all team members. This shared environment would allow us to centralize data management and eliminate the inconsistencies that arise from individual setups.

Overall, this sprint was a valuable learning experience for our team. We confirmed the benefits of dividing tasks based on individual expertise, but also recognized areas for improvement in cross-platform setup and database integration. In terms of technical progress, we made significant strides in refining our design and testing documentation, implementing a more cohesive structure that aligns closely with our system's functionality. The challenges we faced with platform compatibility and database setup underscored the importance of anticipating and addressing system-specific needs early on, which we plan to incorporate into our workflow in future sprints.

**References**

Source Code for Tests: <https://github.com/riddle-me-ruben/muscle-mind/blob/main/src/test/TestApp.py>

Database Diagram: <https://lucid.app/lucidchart/e1aa3b95-1a68-4721-8086-70db7491ac93/edit?beaconFlowId=EEBE57428A0C8188&invitationId=inv_51e84202-70a7-4174-8b6e-437c4a9e4ccc&page=0_0>

Requirements: <https://github.com/riddle-me-ruben/muscle-mind/blob/main/README.md>

MySQL Coding Standards: [2.3-coding-standard-document-mysql.docx](https://minersutep-my.sharepoint.com/:w:/r/personal/rjmartinez12_miners_utep_edu/Documents/Team%203%20Software%20Construction/Project%20Part%201%20Documents/2.3-coding-standard-document-mysql.docx?d=wea0c500667414627aa626a855c496a6a&csf=1&web=1&e=VpcZYg)

Design Document: [6-sdd-report-updated.docx](https://minersutep-my.sharepoint.com/:w:/r/personal/rjmartinez12_miners_utep_edu/Documents/Team%203%20Software%20Construction/Project%20Part%202%20Documents/6-sdd-report-updated.docx?d=w4686903312d64e0c8cebe0caf8058784&csf=1&web=1&e=GPpa1b)

Design Diagram: <https://lucid.app/lucidchart/83f4a27e-4758-4084-93f1-664ac14a5b00/edit?invitationId=inv_4949206f-d26c-47a0-8a23-a68647534ca5&page=0_0>