Gym Whisper

CMPT276: Introduction to Software Engineering

Final Project Report

Group 9 - Pines

Sven Jensen, Nick Deinego, Parminder Randhawa, Ilian Khankhalaev

Instructor: Parsa Rajabi

Summer 2025

GitHub Repository: [Click Here]

Deployed Website: [Visit Gym Whisper]

Gym Whisper

1. Introduction

Gym Whisper is a voice-driven workout tracking web application that enables users to log their gym sessions through natural speech, removing the need for manual data entry. It offers a convenient, hands-free experience designed to streamline workout tracking without disrupting exercise routines.

The app integrates a speech-to-text API with a large language model (LLM) API to transcribe voice input and convert it into structured workout data. This approach simplifies fitness tracking and enhances accessibility for a wide range of users.

Built with React, the application follows a client-side architecture structured around the Model-View-Controller (MVC) pattern. Throughout development, the team applied core software engineering practices, including SDLC planning, version control, testing, and peer feedback. This report outlines the full development process, details, key implementation decisions, reviews testing and system design, and discusses areas for future improvement.

2. Meeting User Needs

2.1 Peer Testing Feedback

To evaluate how well *Gym Whisper* met user needs, a peer testing session was conducted with seven participants. The goal was to collect feedback on the app's accuracy, ease of use, interface design, and overall usefulness. According to the survey results, all participants said that the AI correctly recognized and recorded their spoken workouts. This shows that the speech-to-text and language model integration worked reliably in most cases.

If the AI made a mistake, most users found it either “very easy” or “easy” to fix. This suggests the interface allowed users to make changes without much difficulty. When asked whether the app could help them stay more consistent with their workouts, 100% of respondents answered “definitely” or “maybe.”

The design of the user interface was rated positively by all users, with everyone giving it a score of 4 or 5 out of 5. Most users also felt that the app saved them time compared to traditional tracking methods. Finally, all seven participants said they would recommend *Gym Whisper* to a friend who goes to the gym.

Peers noted a few usability concerns — such as what happens when no input is detected, how errors are communicated, and the inability to track previously completed workouts. The absence of a **workout history** and **database-backed storage** for past sessions was repeatedly mentioned as a major limitation.

This feedback directly guided our design and development priorities for the final phase. For instance, we improved error handling and added UI polish (like background images, typography, and dark mode) to enhance the overall user experience. Although we had originally considered implementing a workout database and language selection feature, the peer testers' interest in workout tracking led us to shift our focus and prioritize the development of a **Workout History** feature instead.

We believe this shift better aligns with user expectations and long-term utility of the app. The feedback collected during this session not only shaped our final deliverables but also heavily informed our **Future Work** roadmap. Based on the consistent demand for workout history, nutrition tracking, and better onboarding, we plan to include these features in the next development cycle.

Overall, the feedback shows that the app successfully meets its goal of making workout tracking faster and easier. It also provided helpful insights for future updates. Full survey results are included in *Appendix C*.

3. Software Develpment Life Cycle (SDLC)

3.1 Chosen SDLC Model

For this project, our team selected the **Agile Scrum** model as the primary Software Development Life Cycle (SDLC) framework. This approach allowed us to break the project into smaller, manageable sprints with clearly defined goals and deliverables. Scrum was well-suited to our team because it promotes collaboration, regular communication, and iterative progress — all of which were essential for a project with evolving user feedback and multiple development components.

Each sprint focused on specific tasks, such as UI development, API integration, and user testing. We tracked our progress using GitHub Projects and maintained communication through weekly meetings. Scrum ceremonies like sprint planning and retrospectives were informally conducted to keep everyone aligned. The use of short iterations helped us remain flexible and adjust to unexpected changes, such as feature scope adjustments and API behavior.

3.2 Adaptations to the Model

While we followed the core principles of Scrum, several adaptations were made to better fit our team structure and project constraints:

- **Sprint Lengths**: Instead of fixed two-week sprints, we adjusted sprint durations based on workload and academic deadlines. For example, some sprints were shorter to accommodate milestone due dates.

- **Role Flexibility**: Scrum roles were not rigidly assigned. Responsibilities like "Product Owner" or "Scrum Master" were shared informally depending on the stage of development and individual strengths.
- **Async Collaboration**: Due to differences in schedules and availability, much of our collaboration occurred asynchronously through GitHub and shared documentation, with most of our team communication taking place on Discord rather than through daily stand-ups.
- **Reduced Formality**: Formal Scrum documentation (e.g., sprint backlogs and burndown charts) was replaced with simpler tools like GitHub Issues and milestone boards for tracking tasks and progress.

These adjustments made the process more realistic and manageable for a small team while still preserving the iterative and adaptive nature of Agile development.

Overall, the Scrum-based process supported our team's productivity and adaptability, especially in responding to evolving feedback and technical challenges. Earlier phases like planning and UI design were well-structured due to clear milestones and informal sprint goals. However, later stages such as bug fixing and polishing would have benefited from tighter scope control and more time for testing.

In future projects, we would consider introducing clearer sprint boundaries, assigning Scrum roles more deliberately, and holding regular (even brief) check-ins to enhance accountability and task distribution. These changes could further improve team coordination and ensure timely delivery of features with fewer bottlenecks.

For visual examples of how we applied these tools and practices — such as GitHub Issues, Milestones, Labels, and Kanban boards — please refer to *Appendix D*.

4. Implemented Features

4.1 Changes Since Planning Stage

During development, several adjustments were made to our initial feature plan in response to technical constraints, time limitations, and feedback from peer testing. While we originally intended to implement an exercise database API, we ultimately decided to remove this feature. The integration complexity and limited short-term value made it a lower priority, especially as we aimed to focus on core functionality.

In contrast, a new feature that emerged from user feedback and was successfully implemented is the **Workout History**. This feature allows users to view and track their previously logged workouts, helping them monitor progress and maintain consistency over time. It became a core addition after peer testers expressed strong interest in being able to reflect on past sessions.

Additionally, the original "Workout Summary" feature was refined and rebranded as a "Performance Report." While the initial plan focused on generating a simple text recap of the exercises completed, the updated implementation emphasizes a more analytical overview of the workout session. This includes highlighting exercise variety, total volume lifted, and overall performance trends based on the structured data generated by the APIs. The change was made to provide more meaningful feedback to users, aligning the output more closely with the project's goal of improving workout tracking and user engagement.

While the core set of planned features — such as voice input, transcription, and structured workout output — remained consistent, we made several adjustments to their implementation based on technical limitations and peer feedback. We also modified how some features function: the live transcription was improved to provide better real-time feedback, and the language support was narrowed to include only French, with full interface and speech recognition

adaptation. These changes didn't alter the overall goals of the project but reflect refinements in how the features were developed and delivered.

We also implemented visual and interface enhancements such as **dark mode**, **custom-styled Gym Whisper text**, and **background images**, which improved the overall polish and user experience of the app.

4.2 Web Speech API Integration

Feature	*Description*
Voice Input	The **Voice Input** feature enables users to record their workouts by speaking naturally into the application, providing a hands-free and convenient logging experience. This feature is powered by the Web Speech API, which is built into modern web browsers and supports real-time speech recognition. When the user clicks the microphone button on the interface, a JavaScript event listener activates the Web Speech API's SpeechRecognition service. The system immediately begins capturing audio input and transcribing the user's speech in real time. The captured transcription is then passed to the next stage of the processing pipeline (handled by the LLM API) for conversion into structured workout data. The voice input component is a critical entry point in the app's workflow, as it initiates the user's interaction and sets the foundation for

	downstream features like workout summarization and history logging.
Live Transcription	The **Live Transcription** feature provides users with real-time visual feedback as they speak, allowing them to observe how their spoken input is being transcribed by the system. This is achieved using the Web Speech API's continuous recognition and interim result capabilities. Once voice recording begins (triggered by the microphone button), the SpeechRecognition object captures not only final transcriptions but also **interim results**, which are partial guesses of what the user is currently saying. These interim results are updated frequently and displayed dynamically on the screen, giving the user an immediate sense of how their voice is being interpreted by the system. This feature is particularly valuable for: • Identifying misrecognized words early in the process. • Allowing users to monitor system accuracy in real time. • Enhancing user trust by making the "thinking" process of the system visible. While this phase is **read-only** — users cannot edit or correct transcription errors before submission — the transparency

	helps set expectations and encourages users to speak more clearly.
Language Options	To support a more inclusive and accurate user experience, the **Language Options** feature allows users to switch the app's interface and voice recognition system to **French**. This is particularly useful for non-native English speakers who are more comfortable interacting in their preferred language. The implementation involved two main aspects: ***1. UI Language Switching:*** A dropdown selector was added to the header of the application, enabling users to choose their language preference. When **French** is selected: • All interface text (buttons, labels, feedback messages, etc.) dynamically updates using a simple **localization mechanism** powered by conditional rendering based on the selected language. • A language state variable is maintained at the root component and passed down to child components using props or context to ensure consistent language display across the app.

	2. Speech Recognition in French: The **Web Speech API** allows setting the language used for recognition via the lang property of the SpeechRecognitionobject. When the user selects French, the system updates this property to "fr-FR", enabling accurate recognition of spoken French input. This ensures that voice commands spoken in French are interpreted correctly by the system and that interim and final transcriptions appear in the expected language. Although currently limited to French and English, the architecture is designed to support future expansion to other languages by: • Extending the localization dictionary • Mapping more lang codes to the Speech API • Adjusting prompts and formatting logic accordingly Together, these enhancements improve accessibility, reduce transcription errors for French-speaking users, and set the foundation for broader language support in future updates.

4.3 LLM API Integration (Google Gemini API)

Feature	*Description*
Workout Data Extraction	This feature leverages a large language model (LLM) API to extract structured workout details — such as exercise name, number of sets, reps, and weights —

	from the raw transcribed text, regardless of how the user phrases their workout. After the Web Speech API converts voice input into plain text, this text is sent to the LLM API, where prompt engineering is used to guide the model to output a structured JSON format. For example, if a user says, "Today I did 8 bench presses with 100 pounds," the model parses the sentence and returns structured data like: • exercise: Bench Press • reps: 8 • weight: 100 lbs We designed the system to handle various phrasings and natural language variations, making it flexible and user-friendly. Prompt formatting ensures that the LLM responds with consistent field names and value types. The extracted data is then displayed in a structured table, where users can visually confirm or later edit their input before finalizing it. This feature plays a central role in transforming spoken input into actionable, trackable fitness data.
Error Handling & Data Validation	After the LLM API processes the transcribed input and returns structured workout data, the system prompts the user to review the information displayed in a table format. This confirmation step serves as a key checkpoint for catching parsing

	errors or misinterpretations. If the AI's output does not match what the user intended (e.g., incorrect number of reps or exercise name), the user is given the option to manually correct the data directly in the table. This approach ensures accuracy in logged workouts while keeping the user in control of the final data. By allowing manual validation and correction, the system combines the efficiency of automation with the reliability of user confirmation, preventing incorrect entries from being saved.
Performance Report	Once a workout session is completed, the user can click the **"Performance Report"** button in the Workout History section to generate an AI-driven analysis of their session. The system compiles the report using the confirmed and validated structured data from earlier in the workflow, including exercise names, sets, reps, and weights. For each exercise, the report provides a detailed breakdown of what was completed, and — if historical entries exist for the same exercise on different dates — it compares the latest results against previous records to highlight progress, plateaus, or regressions. In cases where there is insufficient historical data to analyze trends, the report clearly states that multiple dated entries are required for progress tracking. By offering a concise

	yet informative overview, the Performance Report allows users to easily review their training consistency and performance trends, helping them make informed decisions about future workouts.

4.4 Workout History (New Feature)

In response to peer feedback, we implemented a **Workout History** feature that allows users to view their past logged workouts. After confirming a workout, the structured data is saved locally and displayed in a history table. This table is updated dynamically and stored in the browser using local Storage.

The workout history improves the app's utility by helping users track progress over time, review previous sessions, and maintain consistency in their fitness routine. It also laid the groundwork for potential future enhancements like calendar-based navigation, trend visualization, or data export.

5. CI/CD Pipeline

Our CI/CD (Continuous Integration and Continuous Deployment) pipeline was designed to support automated testing, improve development efficiency, and ensure consistent and stable updates to the application throughout the project lifecycle. We successfully implemented a functioning CI workflow using GitHub Actions and Jest for testing.

5.1 Continuous Integration Setup

We used **GitHub Actions** to implement continuous integration for our project. The workflow was configured using custom .yml files that automatically ran **unit and integration tests** with the **Jest JavaScript testing library** on every pull request. This setup ensured that new changes didn't introduce bugs or break existing functionality before being merged into the main branch.

The CI process helped catch several issues during development, such as:

- Incorrect formatting of numerical values in workout summaries
- App crashes when the voice input was silent or incomplete
- Errors when the LLM API returned unexpected results

By catching these problems early, the CI setup helped us maintain code quality and reduce the need for time-consuming manual testing.

5.2 Deployment Process

Our deployment process was integrated with the CI workflow to ensure that only thoroughly tested and reviewed code was released. Once changes passed all tests and were merged into the main branch, the application was automatically deployed to a hosting environment. This process allowed us to push updates confidently and consistently without manual intervention.

This approach enabled us to:

- Deliver working builds during key milestones
- Test features in a realistic environment
- Quickly iterate based on issues or feedback
- Ensure that the app remained functional during milestone submissions

5.3 Monitoring and Updates

After each deployment, the application was tested by both team members and peer testers to monitor for runtime errors, UI issues, and incorrect API behavior. We did not implement automated monitoring tools (such as logging or error tracking services), but we regularly reviewed feedback and tested user flows manually.

Bug reports and feature requests were tracked using **GitHub Issues**, allowing us to quickly identify and address any problems that arose post-deployment. Updates were scheduled based on milestone planning and tracked using GitHub Projects.

6. Testing Strategy

Our testing strategy incorporated different techniques to ensure that both individual components and overall user functionality were working as intended. These methods helped us identify bugs, validate core functionality, and improve usability based on real user feedback. Testing was conducted throughout development, primarily using manual execution, with some automated test coverage for core logic.

6.1 Unit and Functional Testing

Unit testing was used to validate individual components of our application, such as helper functions that parsed user input or handled formatting logic. These tests helped isolate bugs at a small scale before they could impact larger workflows. We wrote unit tests for specific logic involved in handling the LLM API response and structuring it into a readable workout summary.

Alongside unit testing, we performed manual functional testing to ensure that the application worked correctly from the user's point of view. This involved testing features such as voice input, workout history, output generation, and UI feedback under different usage scenarios. For example, we confirmed that the microphone button correctly triggered the Web Speech API, that incomplete or unexpected input was handled gracefully, and that the summary table displayed accurate information. Functional testing was performed frequently during development and before each major milestone to verify that full user flows remained consistent and intuitive.

6.2 Integration and Automated Testing

Integration testing ensured that components such as the speech-to-text API, the LLM API, and the front-end rendering logic worked together smoothly. For example, we verified that the transcribed text from the voice input was correctly passed into the LLM API and that the resulting structured data was displayed in a consistent format in the UI. These tests helped confirm the workflow (from recording a workout to viewing it in structured form) was stable and error resistant.

In addition to manual and local testing, we implemented a working CI pipeline to support automated testing and maintain code quality. The continuous integration process was built using **GitHub Actions**, where custom **.yml workflow files** were configured to automatically run **unit and integration tests** using the **Jest JavaScript testing library**. Every pull request triggered these automated tests to verify that new changes didn't introduce regressions before merging into the main branch. The CI setup significantly improved our development process by helping maintain stability across updates and reducing the burden of manual testing.

Our CI testing pipeline helped us catch several issues early during development. For example, one test caught a bug where the workout summary was showing numbers with too many decimal places, making it hard to read. Another test found that if the LLM API gave an unexpected response, the app would crash instead of showing an error message. We also discovered a problem where empty voice input (when the user said nothing) was still being sent to the API, causing confusion in the results. These issues were all fixed before merging, thanks to the automated tests triggered in our GitHub Actions workflow.

6.3 Peer Testing (UX)

In addition to developer-side testing, we conducted a peer testing session with seven participants to evaluate the app's usability, clarity, and feature effectiveness.

Users were asked to interact with the app, perform a spoken workout entry, and answer a short survey about their experience.

This feedback helped us identify strengths and weaknesses in the user interface and input handling. Most users rated the app's UI highly (4 or 5 out of 5), found the AI transcription accurate, and considered the app potentially helpful for improving workout consistency. They also made feature suggestions (including workout history, progress tracking, and nutrition logging), which were noted for future development.

A detailed breakdown of peer feedback and survey results can be found in *Appendix C*.

7. System Architecture

Our system architecture is designed following the Model-View-Controller (MVC) paradigm and is supported by a clearly defined data flow across modules and APIs. The architecture reflects the project's emphasis on voice-first interaction, seamless API integration, and reactive UI updates. It evolved iteratively, incorporating feedback and testing insights throughout the development lifecycle.

7.1 Architecture Overview

The application is structured according to the MVC (Model-View-Controller) design pattern to separate concerns and enhance maintainability. In this architecture:

- **Model** represents the structured workout data produced by the AI and stored locally.
- **View** handles all user interface components, including the microphone button, output table, and history display.

- **Controller** acts as the intermediary, handling user events (e.g., voice input), triggering API calls, and updating the view based on results.

The system relies on two main APIs:

1. **Web Speech API** – for capturing and transcribing user voice input.
2. **Gemini LLM API** – for converting transcribed text into structured workout data.

The app also includes a lightweight, front-end-only storage system to temporarily store workout history during a session.

7.2 Level 1 Data Flow Diagram

Although we did not make changes to the Level 1 DFD (see *Appendix E*) since Milestone 1, it remains relevant as it captures the key stages of interaction between the user, APIs, and internal logic.

Flow Summary:

1. User activates voice input by clicking the microphone button.
2. Voice is transcribed by the Web Speech API into plain text.
3. The raw transcription is processed and sent to the Gemini API for parsing into structured JSON data.
4. The structured data is validated and optionally saved into a temporary workout log.
5. The workout log is used to generate a session summary.

This DFD emphasizes how input is processed and transformed at each stage of the pipeline. It reflects a clean, modular approach to system design, ensuring that each processing step is isolated and testable.

7.3 Updated MVC Diagram

The updated MVC diagram (see *Appendix E*) shows the refined interaction between the user, application components, and third-party APIs. Key updates include:

- The controller coordinates between the **View** and both external APIs, forwarding audio and textual input.
- The **Web Speech API** transcribes the audio input into plain text, which is optionally passed to the **Gemini API** for enhanced understanding.
- Once structured data is returned, the controller updates the **View**, allowing the user to confirm or correct the results.
- Confirmed data is stored in the **Model** and displayed in the workout log or summary table.

This diagram clearly delineates control flow and data responsibilities across the application, aligning with our MVC structure and improving clarity for future contributors.

8. Known Bugs and Issues

Overall, Gym Whisper functioned reliably during development, and the number of bugs encountered was relatively low. Most issues were minor and typically identified early through peer testing or during CI test runs. Our use of GitHub Actions and Jest for continuous integration helped catch regressions promptly, allowing us to maintain a stable and functional codebase. The table below provides a summary of the main bugs that were encountered, including a brief description and severity classification based on their impact and frequency.

Bug Description	*Severity*	*Reproduction Steps*
Live transcription cut off the last word spoken during fast speech	Medium	1) Click the microphone button to start recording. 2) Speak a workout description quickly without long pauses. 3) Observe that the last word is sometimes missing from the live transcription display. 4) Issue confirmed during manual peer testing.
API occasionally misunderstood or failed to recognize spoken input	Medium	1) Click the microphone button and speak a workout description with background noise or a non-native accent. 2) Wait for the transcription. 3) Check that certain words are misinterpreted or replaced with unrelated terms. 4) Issue observed during manual testing sessions.
Button to stop recording remained clickable after speech ended, causing confusion	Low	1) Click the microphone button to start recording. 2) Stop speaking and allow the speech recognition to end automatically. 3) Observe that the stop button remains clickable even though recording has stopped. 4) Minor UI issue reported during peer testing.
Linting warnings triggered by inconsistent spacing or unused variables (see *Appendix F*)	Low	1) Run npm run lint or let GitHub Actions run the linting workflow. 2) Observe warnings about unused imports, unused variables, and console.log statements across multiple files. 3) These warnings were automatically flagged during CI runs by GitHub Actions.

9. Future Work

While *Gym Whisper* successfully achieved its core goal of enabling voice-based workout tracking, several potential improvements and new features were identified during development and peer review. These additions would enhance usability, personalization, and long-term user engagement.

One of the most frequently requested features from the peer testing session was a **workout history with calendar integration**. This would allow users to view, search, and filter their past workout sessions, helping them monitor progress and maintain consistency over time. We believe this feature would significantly improve user experience by adding accountability and structure.

Another key feature we originally planned, but ultimately postponed, was the creation of a **workout database**. This would store previously entered workouts and offer users access to a library of common exercises, potentially including recommended rep/set ranges or muscle group categorization. This database could also serve as the foundation for personalized workout suggestions or goal tracking.

In addition, **nutrition tracking** was suggested by some peer testers. Integrating basic nutrition logging or macros tracking could expand the app into a more holistic fitness tool. While this would increase complexity, it opens the door to combining voice input with daily food and supplement logs in future versions.

Other future possibilities include:

- Custom voice commands for starting, stopping, or editing workouts
- Cloud sync and user accounts for persistent data storage
- Integration with fitness APIs or wearable devices

These features were not implemented due to time constraints and prioritization of core functionality, but we consider them high-value enhancements for future development cycles. These improvements reflect both user needs and the project team's ongoing vision for *Gym Whisper* as an intelligent, voice-first fitness assistant.

10.Lessons Learned

10.1 Challenges Faced

Throughout the development of *Gym Whisper*, our team encountered several technical, logistical, and coordination challenges. One of the primary difficulties was managing **asynchronous collaboration** due to differences in team members' availability, schedules, and workloads. Without regular live meetings, we had to rely heavily on GitHub, Discord, and shared documents to communicate, which occasionally led to misunderstandings.

Another significant challenge was **integrating two separate APIs** (the Web Speech API and a language model API) into a seamless workflow. Ensuring accurate speech transcription and correctly structured output required careful prompt engineering and error handling, especially given the variety of ways users might phrase their workouts. Debugging API behaviors and formatting responses to fit our frontend layout took more time than initially estimated.

We also faced some issues with **task management and GitHub workflow**, particularly around ensuring branches were up to date, avoiding merge conflicts, and keeping issue tickets aligned with milestone deadlines. While these were learning experiences, they revealed the importance of clearer Git practices and stronger version control discipline in team-based projects.

Finally, we had to deal with **scope creep** — originally planning to implement more features than we realistically could. Deciding what to cut and how to prioritize was difficult but ultimately necessary for delivering a complete, polished minimum viable product on time.

10.2 Individual Growth

Despite the challenges, each team member gained valuable experience in both technical and collaborative areas. We strengthened our skills in **frontend**

development (especially with React), **API integration**, and **prompt engineering** for LLMs.

From a project management perspective, we learned how to **adapt agile practices** to a small student team, using GitHub issues, milestones, and Kanban boards to plan and track work. We also improved our communication skills, learning to coordinate effectively through tools like Discord and shared documents.

Individually, each member had the opportunity to take ownership of specific parts of the project — whether it was designing the UI, integrating APIs, managing documentation, or preparing the final deliverables. This encouraged accountability and gave everyone hands-on experience with real-world software engineering workflows.

Overall, this project served not only as a technical challenge but also as a valuable opportunity to learn how to navigate real-world development obstacles. Many of the lessons we learned — from managing asynchronous teamwork and prioritizing scope, to debugging API workflows and enforcing Git discipline — will directly inform how we approach future projects, both academically and professionally. We leave this experience with a clearer understanding of what makes effective software teams and how we can improve as developers and collaborators.

Appendix A

Group Member Contributions

This appendix provides a detailed breakdown of each group member's individual contributions to the project. Roles were distributed based on members' strengths and interests, and collaboration was maintained throughout the development process. The table below outlines the specific tasks each member was responsible for across various stages of the project, including planning, design, implementation, testing, and documentation.

Group Member	*Contributions*
Sven Jensen	Sven served as the **project leader** and was the driving force behind the technical development of *Gym Whisper*. He set up the **initial project environment**, configured the **React architecture**, and implemented the **core backbone** of the application — laying the groundwork for other team members to contribute UI design and interaction features. Sven integrated the **Web Speech API** for voice input and the **LLM API** for processing user speech into structured workout data. He also implemented the **CI/CD pipeline** using **GitHub Actions** with custom **.yml workflows and** configured **automated testing** using the **Jest JavaScript testing library**. His leadership and programming contributions ensured the structural integrity, functionality, and deployment readiness of the application.
Nick Deinego	Nick played a crucial role in the early and mid-development stages of the project. He was primarily responsible for the **UI/UX design**, developing both low- and mid-fidelity prototypes, which laid the foundation for the app's overall look and feel. His sketches and mockups helped align the team's vision and informed later implementation. Beyond design, Nick made significant **frontend contributions**, building key layout components and collaborating closely with the team on styling, structure, and user flow. His

	consistent work ethic and clear communication helped keep the project on track.
Parminder Randhawa	Parminder focused extensively on the **frontend implementation**, working alongside Nick to convert designs into functioning components. He handled large parts of the **React UI development**, including layout responsiveness and user interaction features. Parminder also took initiative in **team coordination**, reminding the group of upcoming deadlines, organizing task lists, and stepping up when additional support was needed. His reliability and leadership helped maintain workflow momentum throughout the project.
Ilian Khankhalaev	Ilian contributed to multiple areas of the project. He was active in **documentation**, writing and refining reports, organizing the project structure, and helping maintain clarity throughout the development cycle. He worked on **testing protocols** and participated in preparing the peer testing materials. Ilian also worked on the early implementation of the exercise **database API**, which was later removed, and assisted with **unit testing**. He created and managed all **GitHub issues**, including custom labels and milestones, helping structure the team's workflow. During the final stages, Ilian helped with **finalizing the report and presentation deliverables**. He played an important role in maintaining quality and consistency across all aspects of the project.

Appendix B

Changelog Since Milestone 1

This appendix presents a summary of all revisions and changes made to the project since previous milestones. The changelog includes updates to features, UI elements, project scope, codebase improvements, and any other modifications made during the development cycle. This table helps track the project's evolution and provides transparency regarding iterative changes based on feedback and internal review.

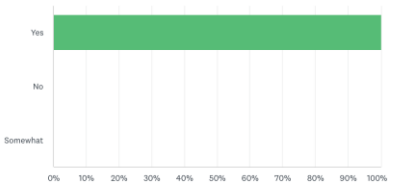
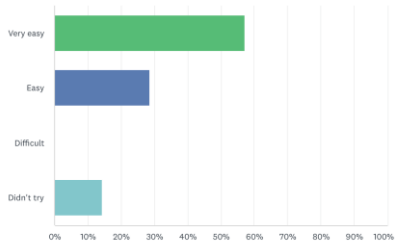
Change Type	*Description*
New Feature	Implemented Workout History interface to allow users to view past session
Feature Removed	Dropped planned Language Selection functionality due to limited scope
Feature Modified	Updated transcription confirmation UI for better error handling
LLM API Update	Refined prompt sent to LLM for more accurate table generation
Testing Added	Introduced unit, functional, and integration tests for API and UI components
CI/CD Pipeline	Implemented continuous integration and deployment via GitHub actions
UI Polish	Improved button spacing, visual layout (added background images, a custom Gym Whisper header, and dark mode toggle for a more engaging and accessible user interface), and overall responsiveness
Architecture Update	Refined MVC structure

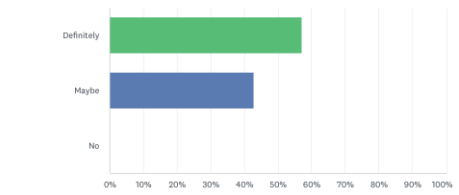
Bug Fixes	Fixed multiple critical issues, including: • a white screen bug on app reloads • crashes when users remained silent after recording • LLM misinterpretation when user described multiple exercises in a single sentence
Documentation	Added detailed GitHub issues, README updates, and user instructions
Peer Feedback	Incorporated UX changes based on peer testing session
Performance	Optimized LLM calls and loading state handling

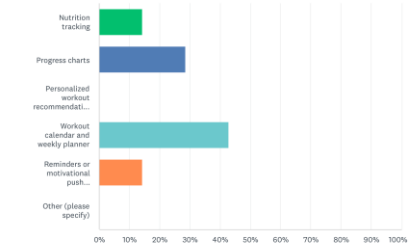
Appendix C

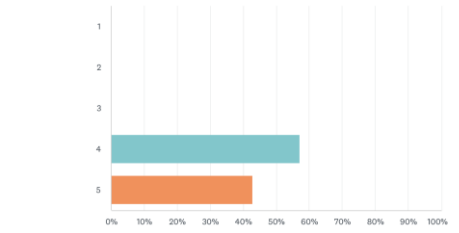
Peer Testing Survey & Results

This appendix includes the survey questions and results collected during the peer testing session of *Gym Whisper*. Seven users tested the application and completed a feedback form. The goal was to evaluate usability, accuracy, and overall user satisfaction. Their responses were used to validate key design choices and to guide future improvements.

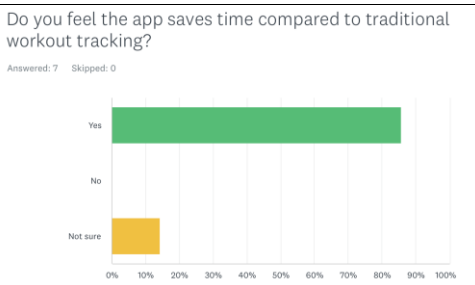
Survey Question & Result	*Interpretation*
Was the AI assistant accurate in recognizing and recording the workouts? Answered: 7 Skipped: 0  ANSWER CHOICES / RESPONSES Yes 100.00% 7 No 0.00% 0 Somewhat 0.00% 0 TOTAL 7	All 7 participants (100%) responded "Yes" when asked if the AI assistant accurately recognized and recorded their workouts. This result indicates that the speech-to-text and LLM integration worked reliably and met user expectations during the testing session.
How easy was it to fix the workout if the AI misunderstood it? Answered: 7 Skipped: 0  ANSWER CHOICES / RESPONSES Very easy 57.14% 4 Easy 28.57% 2 Difficult 0.00% 0 Didn't try 14.29% 1 TOTAL 7	Most users (57%) found it “Very easy” to fix a workout if the AI misunderstood it, and another 29% said it was “Easy.” No one found the process difficult. This suggests that the app’s interface for correcting errors was intuitive and user-friendly. One user did not try editing, which may indicate either no errors occurred or they were unsure how to do it.

<table>
<tr>
<td>
Do you think this app could help you stay more consistent with your workouts?
Answered: 7 Skipped: 0


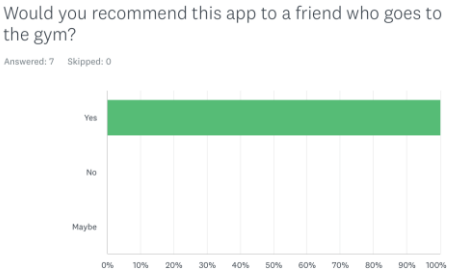
<table>
<tr><th>ANSWER CHOICES</th><th>RESPONSES</th><th></th></tr>
<tr><td>Definitely</td><td>57.14%</td><td>4</td></tr>
<tr><td>Maybe</td><td>42.86%</td><td>3</td></tr>
<tr><td>No</td><td>0.00%</td><td>0</td></tr>
<tr><td>TOTAL</td><td></td><td>7</td></tr>
</table>
</td>
<td>All participants (100%) believed the app could help them stay more consistent with their workouts — 57% answered “Definitely” and 43% chose “Maybe.” This suggests that the app supports long-term workout habits and is seen as a potentially effective motivational or tracking tool by users.</td>
</tr>
<tr>
<td>
Which feature would you most like to see added to this app in the future?
Answered: 7 Skipped: 0


<table>
<tr><th>ANSWER CHOICES</th><th>RESPONSES</th><th></th></tr>
<tr><td>Nutrition tracking</td><td>14.29%</td><td>1</td></tr>
<tr><td>Progress charts</td><td>28.57%</td><td>2</td></tr>
<tr><td>Personalized workout recommendations based on goals</td><td>0.00%</td><td>0</td></tr>
<tr><td>Workout calendar and weekly planner</td><td>42.86%</td><td>3</td></tr>
<tr><td>Reminders or motivational push notifications</td><td>14.29%</td><td>1</td></tr>
<tr><td>Other (please specify)</td><td>0.00%</td><td>0</td></tr>
<tr><td>TOTAL</td><td></td><td>7</td></tr>
</table>
</td>
<td>The most requested features were a workout calendar and weekly planner (43%) and progress charts (29%), indicating strong interest in features that support long-term tracking and visualization. Other users expressed interest in nutrition tracking and motivational push notifications. No respondents selected personalized workout recommendations or submitted additional ideas, which suggests the predefined feature list covered most user expectations.</td>
</tr>
<tr>
<td>
How would you rate the overall user interface design? 1 (poor) to 5 (excellent)
Answered: 7 Skipped: 0


<table>
<tr><th>ANSWER CHOICES</th><th>RESPONSES</th><th></th></tr>
<tr><td>1</td><td>0.00%</td><td>0</td></tr>
<tr><td>2</td><td>0.00%</td><td>0</td></tr>
<tr><td>3</td><td>0.00%</td><td>0</td></tr>
<tr><td>4</td><td>57.14%</td><td>4</td></tr>
<tr><td>5</td><td>42.86%</td><td>3</td></tr>
<tr><td>Total Respondents: 7</td><td></td><td></td></tr>
</table>
</td>
<td>All participants rated the user interface positively, with 57% giving it a score of 4 and 43% giving it a 5 out of 5. No one selected scores below 4, indicating that users found the interface clean, intuitive, and easy to use overall.</td>
</tr>
</table>

<table>
<tr>
<td>


ANSWER CHOICES	RESPONSES	
Yes	85.71%	6
No	0.00%	0
Not sure	14.29%	1
TOTAL		7

</td>
<td>A large majority of users (86%) felt that the app saved time compared to traditional workout tracking methods. One user selected “Not sure,” and no users disagreed. This suggests that the voice-based logging approach was effective in streamlining the workout tracking process for most participants.</td>
</tr>
<tr>
<td>


ANSWER CHOICES	RESPONSES	
Yes	100.00%	7
No	0.00%	0
Maybe	0.00%	0
TOTAL		7

</td>
<td>All participants (100%) said they would recommend Gym Whisper to a friend who goes to the gym. This result reflects strong overall satisfaction with the app and suggests a high level of trust and perceived value among users.</td>
</tr>
</table>

Appendix D

Software Development Lifecycle (SDLC) Artifacts

This appendix presents visual artifacts that reflect the SDLC model we applied during the development of our project. Using an Agile-inspired approach, we organized our work through GitHub's Issues, Milestones, Labels, and Kanban board. These tools enabled us to structure development into manageable phases, prioritize tasks effectively, and maintain continuous progress. The following images illustrate how each component contributed to our workflow, from planning to final delivery.

Artifact D.1 – GitHub Issues

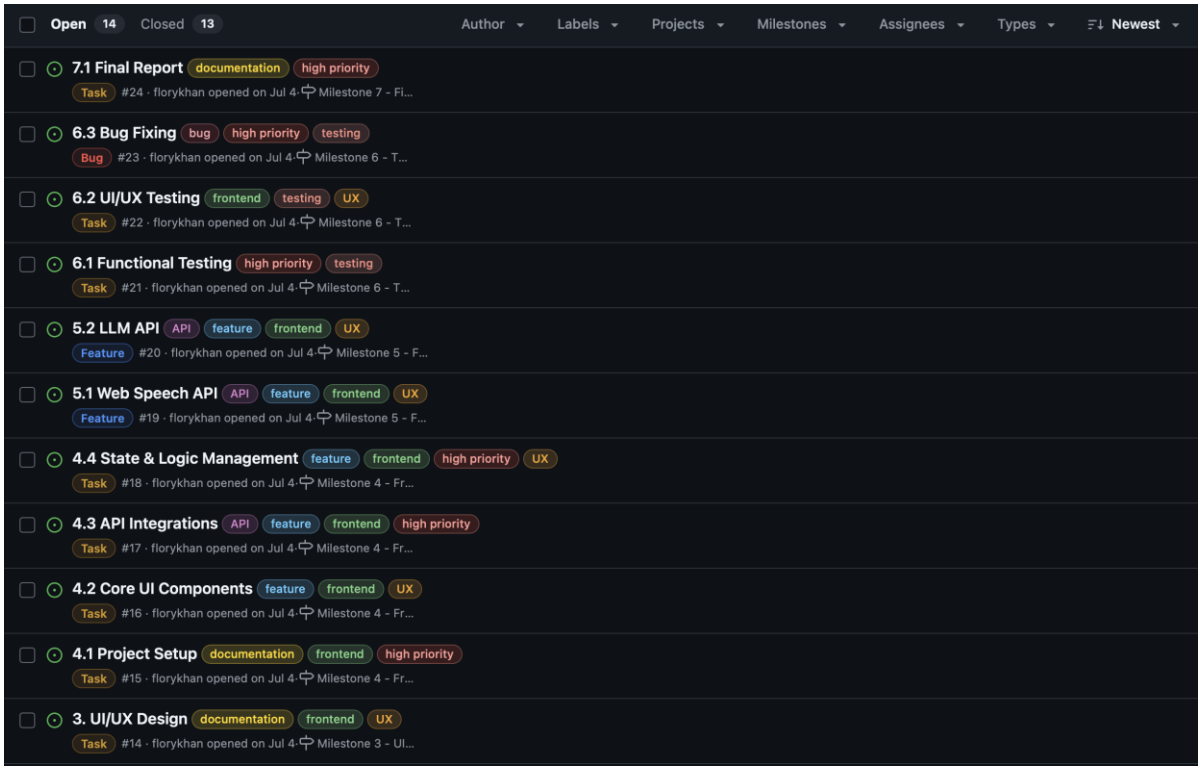


Figure D.1

Figure D.1 displays the open GitHub Issues used to manage and track tasks throughout our project. Each issue is clearly labeled with tags such as feature, bug, frontend, high priority, and documentation, allowing the team to quickly identify the nature and urgency of a task. The consistent naming convention (e.g., “4.3 API

Integrations") reflects the project's SDLC phase structure and milestone mapping. This systematic approach helped ensure each task was aligned with our Agile development process and appropriately scheduled for implementation.

Artifact D.2 – GitHub Labels

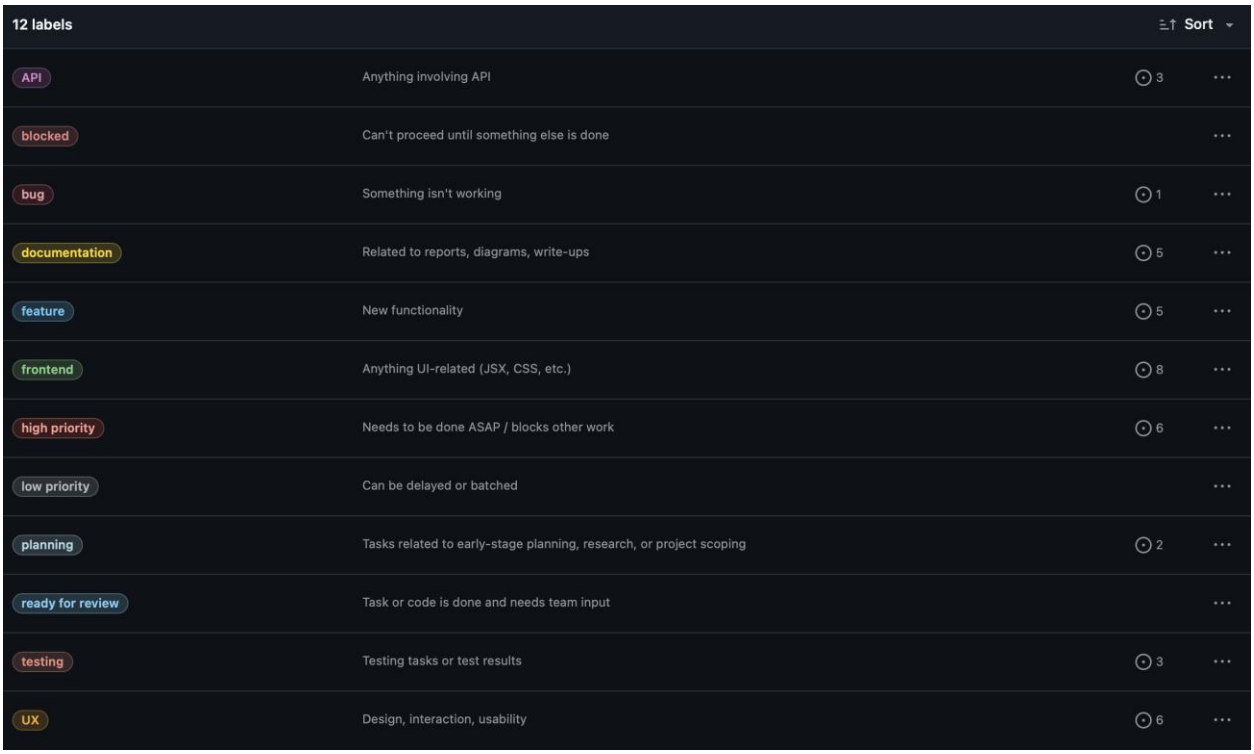


Figure D.2

Figure D.2 displays the custom labels our team used to categorize and prioritize tasks within the GitHub Issues board. Labels such as feature, bug, documentation, testing, and UX helped identify the nature of the task, while priority-based labels like high priority and low priority supported effective scheduling. The use of labels was essential to our Agile-inspired SDLC model, allowing us to filter tasks efficiently, maintain a clear development focus, and ensure that work items were appropriately assigned and tracked throughout each phase.

Artifact D.3 – GitHub Milestones

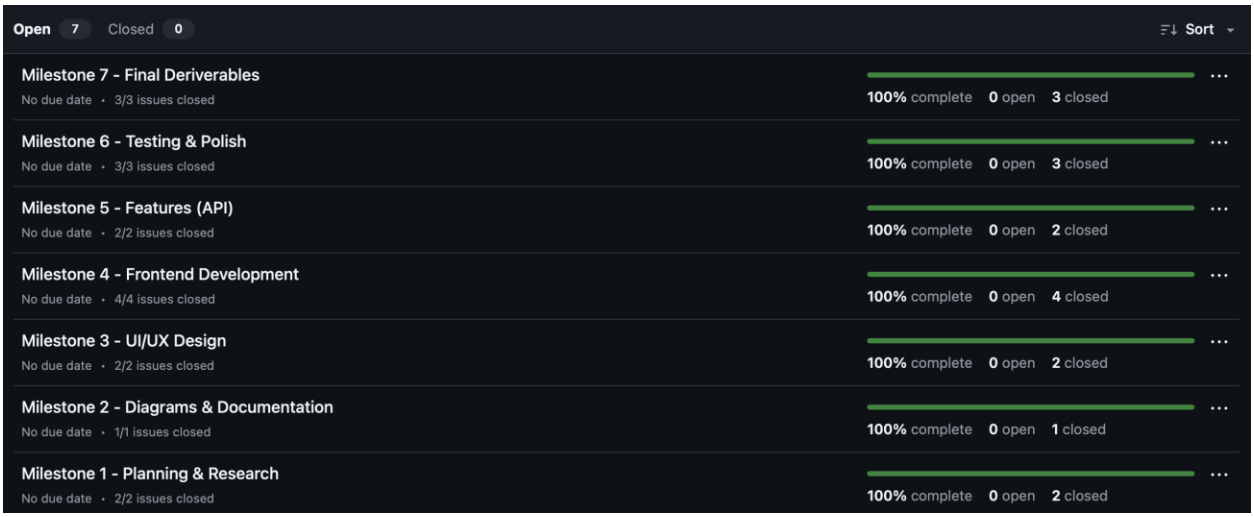


Figure D.3

Figure D.3 presents the GitHub Milestones used to structure our project into clearly defined phases. Each milestone (e.g., UI/UX Design, Frontend Development, Testing & Polish) grouped relevant tasks together, allowing us to monitor progress at a higher level. As shown, all milestones were completed 100%, which reflects our consistent adherence to timelines and goals. The milestone structure directly aligns with our chosen SDLC model, the **Agile Development Model**.

Artifact D.4 – Kanban Board

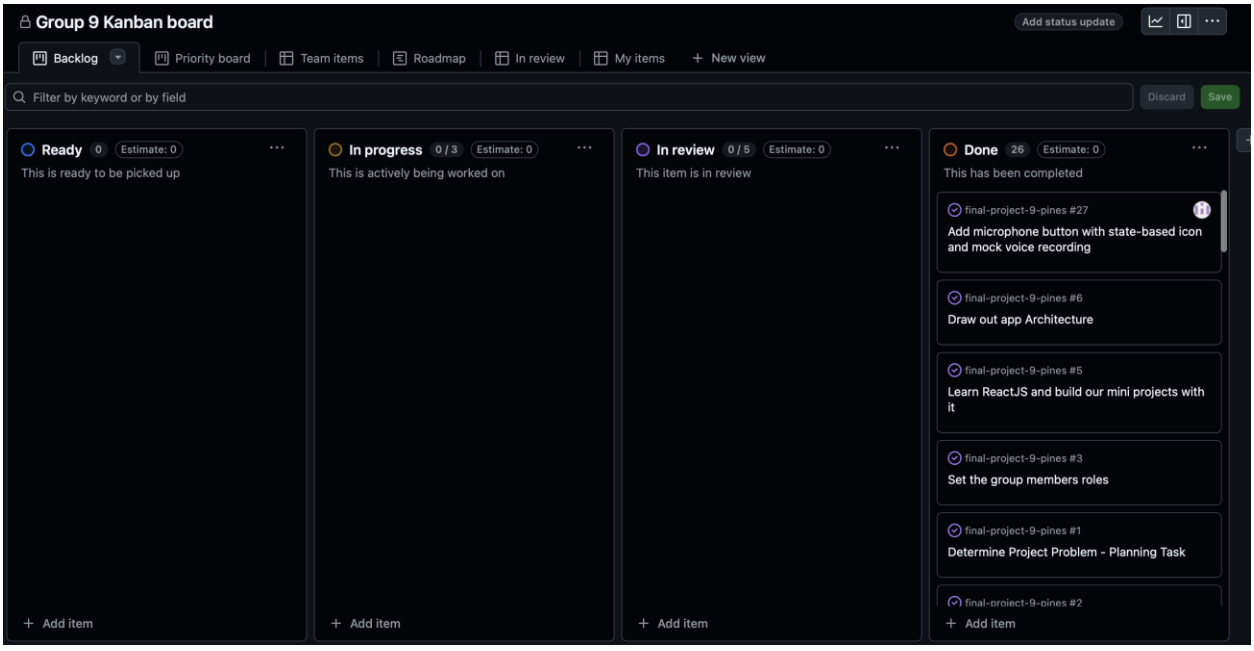


Figure D.4

Figure D.4 displays our GitHub Kanban board, which we used to visually manage the status of tasks throughout the development cycle. The board includes four main columns: Ready, In Progress, In Review, and Done. Each task moved through these columns based on its stage of completion, enabling the team to easily track progress and maintain workflow clarity.

The large number of completed tasks under the **“Done”** column demonstrates our team's consistent execution and iterative progress, in alignment with the **Agile development model** we adopted. The Kanban board supported effective sprint planning, encouraged shared accountability, and gave us a real-time overview of our deliverables.

Appendix E

System Design Diagrams

This appendix includes two core system design diagrams developed for the Gym Whisper application: the Level 1 Data Flow Diagram (DFD) and the updated Model-View-Controller (MVC) diagram. These diagrams provide a visual representation of how data moves through the system and how responsibilities are separated across architectural components.

The **Level 1 DFD** outlines the end-to-end workflow, starting from user voice input to structured workout summary generation. It illustrates the role of each major subsystem, including third-party APIs and parsing logic.

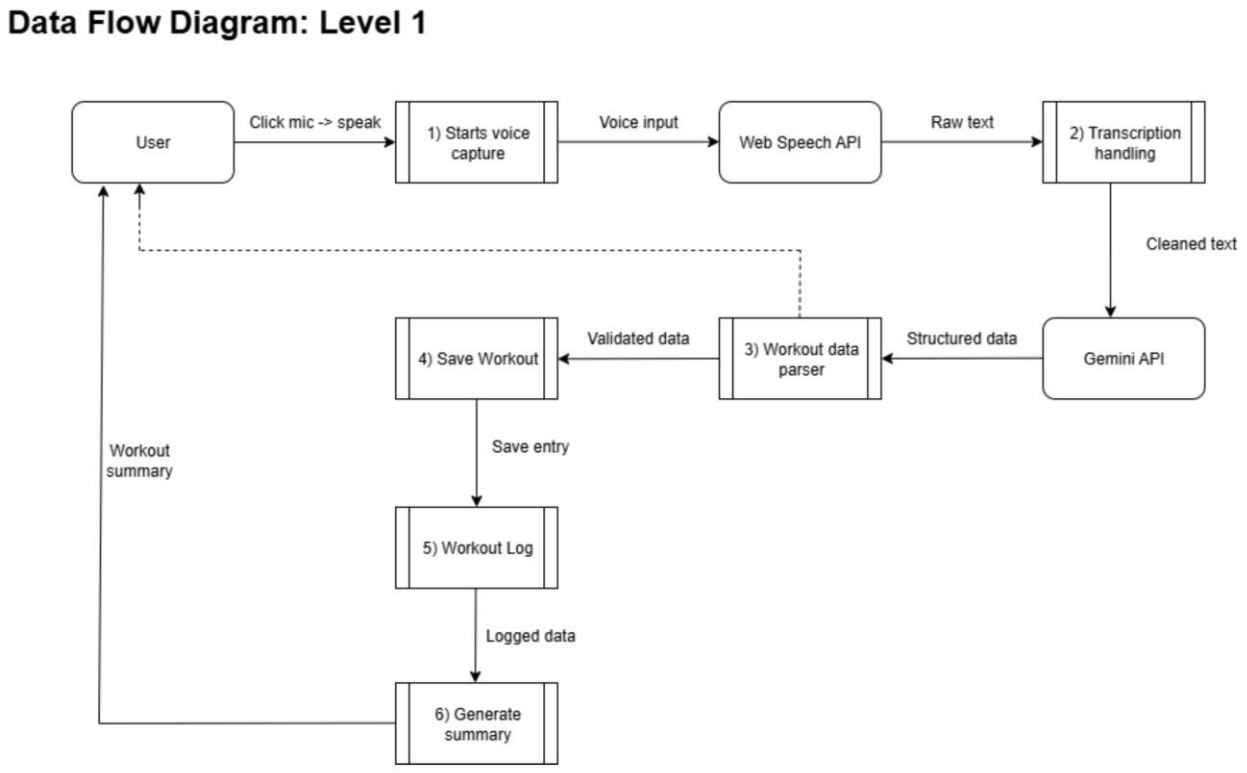


Figure E.1

The **MVC diagram** highlights how the app adheres to the Model-View-Controller pattern, detailing how input events, data processing, and interface updates are coordinated across components.

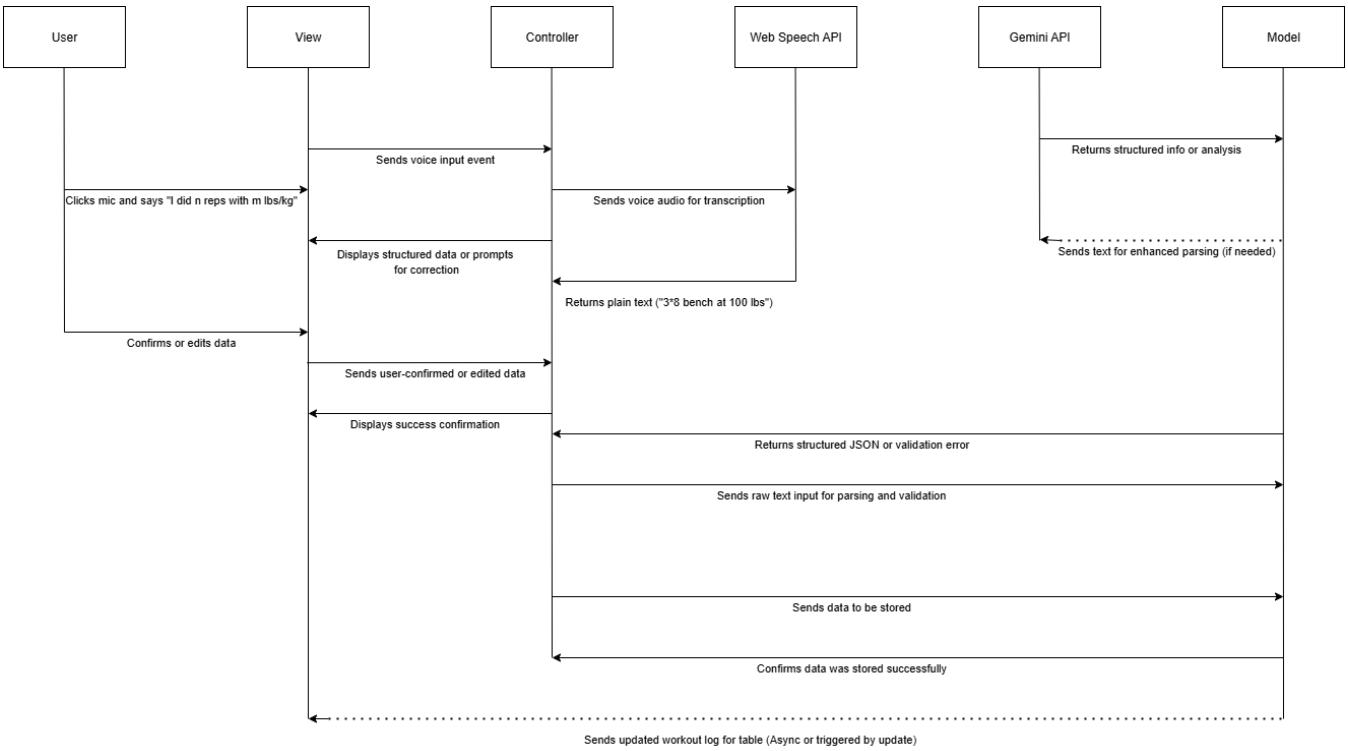


Figure E.2

Together, these diagrams serve to reinforce our architectural decisions, document system logic, and provide future developers with a clearer understanding of the application's internal structure.

Appendix F

Additional Visual Assets

This appendix includes supplementary visual materials that support the content of the main report but were not included due to formatting or space limitations. These materials provide further insight into the system design, development process, and feedback interpretation, enhancing the overall understanding of the project.

ESLint Warnings and Static Code Analysis

During the final phase of development, we ran ESLint across the entire codebase. This analysis yielded 24 warnings and 0 errors, as shown in the output (see figure below). Most warnings were related to **unused variables** or **React imports that were defined but never used**, which is common in rapidly iterated frontend projects. These included:

- Unused imports like React, CheckCircle, and others
- Unused variable declarations (e.g., storeWorkoutDataInLocalStorage, x, scale, e)
- Console statements left over from development (e.g., console.log() used for debugging)

These warnings did **not impact the app's runtime behavior** or core functionality but highlight areas where the code can be cleaned up for better readability and maintainability.

We intentionally left some console statements in place for logging real-time API responses and for peer testing demos. However, for production-level deployment, we would remove these to maintain a cleaner console output and adhere to best practices.

The ESLint output acted as a useful reminder of development hygiene and will guide future refactoring or cleanup if the project is extended.

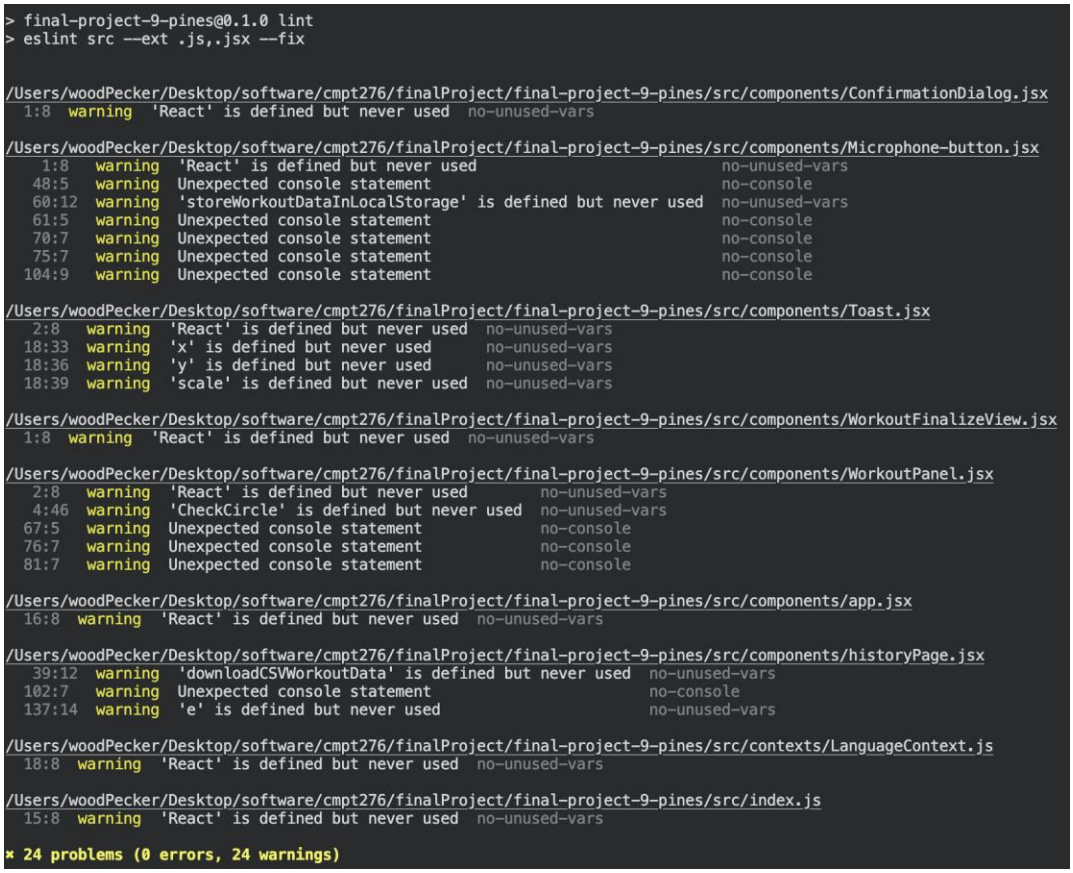


Figure F.1