Verification and Validation Report: SFWRENG 4G06

Team #7, Team FAAM, SweatSmart

Daniel Akselrod
Jonathan Avraham
Sophie Fillion
Sam McDonald

April 5, 2024

1 Revision History

Date	Version	Notes
Mar. 6, 2024	0	first draft of document
April 4, 2024	1	revision 1 changes

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

3.1 User Account

3.1.1 UA1: Account Registration

- Evaluation: Ensure users can successfully register an account with basic information.
- Test Method: Conduct manual testing by registering accounts with various sets of valid and invalid input data.
- Expected Outcome: Account should be created successfully with valid input data, and appropriate error messages should be displayed for invalid data. Should be directed to the onboarding page if successful.
- Input Test: email: "avrahamj@gmail.com", username: "avrahamj", password: "Jonny123!"
- Actual Outcome: app was directed to the onboarding page (PASSED)

3.1.2 UA2: Login/Logout

- Evaluation: Verify users can log in and log out of their accounts.
- Test Method: Perform manual testing by logging in and out using valid and invalid credentials.
- Expected Outcome: Users should be able to log in with correct credentials and log out successfully, with sessions terminated. If credentials are in the database, app should navigate to home page. When logged out, app is navigated to login page.
- Input Test: email: "avrahamj@gmail.com", username: "avrahamj", password: "Jonny123!" (This account exists)
- Actual Outcome: app was directed to the home page (PASSED)

3.1.3 UA3: Profile Creation/Update

- Evaluation: Confirm users can create and update their profiles by filling in their name, fitness goals, and characteristics.

- Test Method: Manually test profile creation and update functionalities with various inputs.
- Expected Outcome: Profiles should be successfully created and updated with valid input data.
- Input Test: name: ”Dan Fin”, fitness goals: ”Gain strength”
- Actual Outcome: Profile was updated and saved

3.2 Workout Generation

3.2.1 AI1: Personalized Workout Plan Generation

- Evaluation: Ensure that the system generates personalized workout plans based on user preferences and characteristics.
- Test Method: Given that the user has a completed profile, the app will generate a personalized workout plan that corresponds to the user’s goals and characteristics.
- Expected Outcome: A workout plan is generated and the user is navigated back to the home page.
- Input Test: Fitness level = ”Beginner”, fitness goals: ”Gain strength”, workout frequency = ”3 times a week”.
- Actual Outcome: A workout plan is generated and the user is navigated back to the home page with 3 workouts in their programs section.

3.2.2 AI2: Personalized Single Workout Generation

- Evaluation: Ensure that the system generates personalized workout plans based on user preferences and characteristics.
- Test Method: Given that the user has a completed profile, the app will generate a singular personalized workout that corresponds to the user’s goals and characteristics.
- Expected Outcome: A single workout is generated and the user is navigated back to the home page.

- Input Test: Fitness level = ”Beginner”, fitness goals: ”Gain strength”
- Actual Outcome: A single workout is generated and the user is navigated back to the home page.

3.2.3 AI3: Workout plan alteration

- Evaluation: Verify that users can alter and revise their workout plans by adding or removing exercises and changing the exercise details
- Test Method: Test workout plan alteration functionality with user requests.
- Expected Outcome: Users should be able to alter their workout plans as needed.
- Input Test: input a review after completed workout
- Actual Outcome: To Be Determined (Fail)

3.2.4 AI4: ChatBot

- Evaluation: Verify that users can interact with the AI chatbot.
- Test Method: Test chatbot interaction with various user inputs, including asking questions related to fitness and not related to fitness.
- Expected Outcome: If questions are related to fitness, it is a valid request and an answer will be provided, else, it will not provide an answer.
- Input Test: User asks how to perform a proper push up
- Actual Outcome: A valid response is given on how to perform a proper push up (PASS).

3.3 User Workout Creation

3.3.1 UC1: Calendar Linking

- Evaluation: Verify that users can link their calendars to plan workouts.
- Test Method: Perform integration testing with various calendar applications (e.g., Google Calendar, Apple Calendar) to ensure seamless linking functionality.

- Expected Outcome: Users should be able to link their calendars successfully without encountering any errors.

3.3.2 UC2: Workout Rating

- Evaluation: Ensure that users can rate their workouts after completion.
- Test Method: Conduct both manual and automated testing to validate workout rating functionality across different devices and platforms (e.g., iOS, Android).
- Expected Outcome: Users should be able to rate their workouts accurately, and the rating should be recorded correctly in the system.

3.3.3 UC3: Custom Workout Creation

- Evaluation: Verify that users can design and save custom workouts.
- Test Method: Conduct manual testing to simulate user interactions with the custom workout creation interface, covering various scenarios and edge cases.
- Expected Outcome: Users should be able to create and save custom workouts as intended, with all preferences accurately reflected in the saved workout.

3.3.4 UC4: Exercise Listing

- Evaluation: Ensure that the system lists all available exercises.
- Test Method: Perform automated unit testing to verify the completeness and accuracy of the exercise listing functionality, checking against a predefined list of exercises.
- Expected Outcome: The system should display a comprehensive list of exercises without any missing or duplicate entries.

3.4 Workout History

3.4.1 WH1: Display Workout History

- Evaluation: Verify that users can view their workout history.

- Test Method: Conduct manual and automated testing to validate the retrieval and display of workout history data, including testing with large datasets to assess performance.
- Expected Outcome: Users should be able to view their workout history accurately, with all relevant details presented in a clear and organized manner.

3.4.2 WH2: Save Completed Workout

- Evaluation: Ensure that completed workouts are saved properly.
- Test Method: Perform automated testing to validate the saving and retrieval of completed workout data from the database, including testing for data integrity and security.
- Expected Outcome: Completed workouts should be saved securely and accurately, with all relevant information preserved for future reference.

3.5 Databases

3.5.1 DB1: Data Storage

- Evaluation: Verify that user account information, workout plans, feedback, and history are stored in the database.
- Test Method: Conduct manual and automated testing to assess the reliability and scalability of the database storage mechanisms, including stress testing under heavy loads.
- Expected Outcome: User data should be stored securely and retrievable when needed, with minimal risk of data loss or corruption.

3.6 Additional Features

3.6.1 AF1: Live Workout Tracking

- Evaluation: Ensure that users can track live workouts.
- Test Method: Conduct manual testing to simulate live workout sessions and validate real-time tracking functionality, including testing with different workout types and durations.

- Expected Outcome: Users should be able to track their workouts in real-time, with accurate data presented throughout the session.

3.6.2 AF2: Push Notifications

- Evaluation: Verify that push notifications are sent to users.
- Test Method: Perform automated testing to validate the delivery and content of push notifications across different devices and platforms, including testing for reliability and latency.
- Expected Outcome: Users should receive push notifications promptly and consistently, with relevant information conveyed in each notification.

3.6.3 AF3: Live Workout Sessions

- Evaluation: Ensure that users can start and follow live workout sessions.
- Test Method: Perform manual testing to simulate live workout sessions and validate user participation and interaction during the session, including testing for session stability and responsiveness.
- Expected Outcome: Users should be able to join and participate in live workout sessions seamlessly, with minimal disruptions or technical issues.

3.6.4 AF4: Workout Downloads

- Evaluation: Verify that users can download workouts for offline access.
- Test Method: Conduct manual and automated testing to validate the download and synchronization of workout data, including testing for file integrity and download speed.
- Expected Outcome: Users should be able to download workouts for offline access reliably and efficiently, with all downloaded data accessible offline.

3.6.5 AF5: Social Media Sharing

- Evaluation: Ensure that users can share information from the app to social media.

- Test Method: Conduct manual testing to simulate social media sharing functionality with various user information, including testing for compatibility with different social media platforms.
- Expected Outcome: Users should be able to share information from the app to social media platforms seamlessly, with all shared content displayed accurately.

4 Nonfunctional Requirements Evaluation

4.1 Performance Requirements

4.1.1 STN-PR-1: System Capacity

- Evaluation: Ensure that the system can handle a minimum of 10 simultaneous users and process 5 transactions per minute while maintaining performance standards.
- Test Method: Automated load testing tools will simulate 10 users connecting and processing transactions.
- Expected Outcome: The system can handle 10 users and process 5 transactions per minute.
- Input: Simulation of 10 simultaneous users using the app at once.

4.1.2 STN-PR-2: Availability

- Evaluation: Confirm that the system achieves a minimum of 99.9% availability, ensuring services are accessible to users at all times.
- Test Method: System monitoring tools will track uptime over a specified period.
- Expected Outcome: The system achieves a minimum of 99.9% availability.
- Input: Monitoring system availability over a month.

4.2 Operational and Environment Requirements

4.2.1 STN-OER-1: Locational Requirements

- Evaluation: Ensure that the system functions correctly without environmental interference in gym/workout facilities or home workout rooms.

- Test Method: A user will test the app in various gym/workout environments to ensure functionality.
- Expected Outcome: The app functions correctly without any environmental interference.
- Input: Testing the app in various gym/workout environments.

4.2.2 STN-OER-2: OS Requirements

- Evaluation: Verify that the system can run on an iOS phone.
- Test Method: Installation and basic functionality test on an iOS device.
- Expected Outcome: The app installs, opens, and functions correctly on an iOS device.
- Input: Attempting to install and run the app on an iOS device.

4.2.3 STN-OER-3: Audio Functionality

- Evaluation: Ensure that the system allows audio output to be transmitted through the device's speakers or a connected Bluetooth device.
- Test Method: Test audio output on device's speakers and on different Bluetooth devices.
- Expected Outcome: Audio is transmitted clearly without any issues.
- Input: Testing audio output on device's speakers and different Bluetooth devices.

4.2.4 STN-OER-4: Network Connection

- Evaluation: Verify that the system can connect to the internet for back-end services.
- Test Method: Use the app with an active internet connection and verify back-end services.
- Expected Outcome: App connects to the internet and back-end services function correctly.
- Input: Using the app with an active internet connection.

4.3 Access Requirements

4.3.1 STN-ACR-1: Strong Password Restrictions

- Evaluation: Ensure that the app enforces strong password restrictions to protect user data.
- Test Method: Attempt to register or change passwords using weak and strong examples and verify acceptance criteria with the use of a regular expression.
- Expected Outcome: Only strong passwords are accepted.
- Input: Attempting to register or change passwords using weak and strong examples.

4.3.2 STN-ACR-2: Authentication Credentials

- Evaluation: Verify that users can authenticate themselves securely.
- Test Method: Use valid and invalid credentials to ensure proper authentication.
- Expected Outcome: Successful login with valid credentials; unsuccessful login with invalid credentials.
- Input: Using valid and invalid credentials for login.

4.3.3 STN-ACR-3: Authorized Data Access

- Evaluation: Ensure that users can only access authorized data and features.
- Test Method: Create different user profiles with different levels of authorization and attempt to access restricted areas.
- Expected Outcome: Users can only access authorized data and features.
- Input: Attempting to access restricted areas with different user profiles.

4.4 Integrity Requirements

4.4.1 STN-INR-1: Restrict Sensitive Data

- Evaluation: Ensure that sensitive data is restricted from users.

- Test Method: Review user data sections to confirm sensitive data is properly restricted.
- Expected Outcome: Sensitive data is not accessible to users.
- Input: Reviewing user data sections for sensitive information.

4.4.2 STN-INR-2: Data Encryption

- Evaluation: Confirm that user data is encrypted to protect it from unauthorized access.
- Test Method: Review encryption methods used and verify data protection mechanisms.
- Expected Outcome: User data is encrypted and protected from unauthorized access.
- Input: Reviewing encryption methods and data protection mechanisms.

4.5 Privacy Requirements

4.5.1 STN-PVR-1: User Consent

- Evaluation: Verify that user consent is obtained before collecting personal data.
- Test Method: Review the app's data collection processes and confirm user consent mechanisms.
- Expected Outcome: User consent is required and obtained before collecting personal data.
- Input: Reviewing the app's data collection processes.

4.5.2 STN-PVR-2: Data Anonymization

- Evaluation: Ensure that user data is anonymized to protect user privacy.
- Test Method: Review data storage and processing methods to confirm anonymization.
- Expected Outcome: User data is anonymized to protect user privacy.
- Input: Reviewing data storage and processing methods.

4.6 Audit Requirements

4.6.1 STN-AUR-1: Data Logging

- Evaluation: Verify that the system logs user activities for audit purposes.
- Test Method: Review logging mechanisms and confirm logging of user activities.
- Expected Outcome: User activities are logged for audit purposes.
- Input: Reviewing logging mechanisms and user activity logs.

4.6.2 STN-AUR-2: Access Logs

- Evaluation: Ensure that access logs are maintained to track user interactions with the system.
- Test Method: Review access log storage and retrieval processes.
- Expected Outcome: Access logs are maintained and accessible for auditing.
- Input: Reviewing access log storage and retrieval processes.

4.7 Social Requirements

4.7.1 STN-SR-1: Social Media Integration

- Evaluation: Ensure that users can share information from the app to social media.
- Test Method: Conduct manual testing to simulate social media sharing functionality with various user information, including testing for compatibility with different social media platforms.
- Expected Outcome: Users should be able to share information from the app to social media platforms seamlessly, with all shared content displayed accurately.
- Input: Testing social media sharing functionality with various user information.

5 Comparison to Existing Implementation

As the digital fitness landscape overflows with a variety of workout generator and tracker applications, we have plenty of material to compare our workout application. This section aims to dissect how SweatSmart, developed under constraints of time, budget, and expertise, measures against the backdrop of existing, often well-resourced, applications. We do not expect this application to be comparable to some of the fully-funded, regularly updated options out there, but it is still important to make comparisons to learn where we can improve, and where we could improve if we had time. There are so many existing implementations out there, we decided to look mostly at the apps used in the past by the kin academics and professionals we interviewed for our usability testing session on February 28, 2024.

5.1 UI Design and Feature Set

SweatSmart's UI design is intentionally minimalist, aimed at providing a straightforward user experience. This design philosophy contrasts with more feature-rich apps, where the complexity of options can overwhelm users. Our app includes basic functionalities such as workout logging, a social feature being actively implemented to enhance user interaction, and the addition of guidance videos and muscle group images for a more informative workout experience. These planned enhancements aim to address the gap in instructional content, a limitation in our current offering. All of the above mentioned features appeared in the other implementations that came up in usability testing.

Another feature present in other available applications is the visualization of workout history and progress, something we have noted and are working to implement by the end of the project.

5.2 Evidence-Based Approach and Limitations

Central to our development philosophy is an evidence-based approach to fitness, which inherently restricts the scope of personalization we can offer, since the evidence suggests a simple, easy to follow workout plan is much more beneficial than one with lots of features and specifics. This approach has led to a simpler app design, focusing on universally beneficial workouts over highly personalized plans. The broad consensus in fitness evidence supports the benefits of general physical activity, guiding our decision to prioritize straightforward, effective workout options that cater to the average user. When comparing to existing apps, like Peloton or JEfit, our app my appear to lack some personalization aspects. What we have learned, however, is that a lot of these aspects are based in pseudoscience to appear more tailored than they really need.

5.3 Planned Enhancements

Acknowledging feedback received from the Kin experts we interviewed and our comparative analysis, we outline key areas for future development:

- Enhancing the social aspect of SweatSmart to engage users through shared experiences and motivational features.
- Improving the workout logging UI for a more intuitive and visually appealing process, and ironing out existing bugs that made workout logging more complicated than it needed to be.
- Adding guidance videos and images to offer detailed instructions and highlight targeted muscle groups, enhancing the educational value of workouts.
- Implementing a system which visualizes important summary data of past workouts and workout history of a user, giving them at-a-glance updates on their progress to keep them motivated.

5.4 Conclusion

While SweatSmart may not match the depth of features found in more established apps, its development underscores a commitment to evidence-based fitness principles within the constraints we face. Our focus on UI improvements and specific feature enhancements reflects a strategic response to user feedback and a comparative understanding of the market, aiming for meaningful improvements within our app's defined scope.

This section will not be appropriate for every project.

6 Unit Testing

Unit testing in our app is focused on our back-end infrastructure with client-facing endpoints being tested. Every service has at least 1 unit test pertaining to different input combinations an endpoint may receive and their respective responses. Below are all endpoint and helper methods within their respective module.

6.1 Account Service

6.1.1 HashPassword

- CanHashPassword(): Ensures password hashing works

6.1.2 Login

- CanLoginWithEmail(): Ensures login via email works
- CanLoginWithUsername(): Ensures login via username works
- BadRequestHandled(): Ensures invalid requests are properly handled
- IncorrectPasswordHandled(): Ensures wrong password does not login and returns error message

6.1.3 Register

- CanRegisterUser(): Ensures user can register with edge cases for input fields
- BadRequestHandled(): Ensured a bad request is handled and returned to user

6.1.4 UpdateUser

- CanUpdateUsername(): Ensures a user can update their username.
- CanUpdateEmail(): Ensures a user can update their email.
- CanUpdateName(): Ensures a user can update their name.
- HandleNonExistentUser(): Ensures update fails if the user does not exist.

6.2 ChatBot Service

6.2.1 GetResponse

- CanGetChatResponse(): Tests if the chatbot can provide a response given a valid history.
- ReturnsBadRequestOnInvalidInput(): Ensures a bad request is returned for invalid input data.

- HandlesAPIFailure(): Ensures service handles failures in the OpenAI API correctly.
- MaintainsSystemMessageIntegrity(): Checks if the system message integrity (role and content) is maintained in the response.
- FiltersNonFitnessContent(): Ensures the bot responds correctly to non-fitness related queries, as per system message instructions.

6.3 Profile Service

6.3.1 UpdateUserPreferences

- CanUpdateUserPreferences(): Ensures user preferences are updated correctly.
- HandlesNonExistentUser(): Ensures response for non-existent user is handled correctly.
- OverwritesExistingPreferences(): Checks if existing preferences are correctly overwritten.

6.3.2 GetUserPreferences

- CanRetrieveUserPreferences(): Tests if the service can retrieve all user preferences.
- HandlesNonExistentUser(): Ensures the service correctly handles a non-existent user.
- HandlesNoPreferencesSet(): Ensures the service handles cases where no preferences are set.

6.3.3 GenerateSingularWorkout

- CanGenerateSingularWorkout(): Tests if a single workout is generated correctly.
- HandlesNonExistentUser(): Ensures proper handling when the user does not exist.
- HandlesNoPreferences(): Ensures correct handling when user preferences are not set.

- ReturnsEmptyWorkoutForUnavailableTime(): Checks for proper response when user's time availability is insufficient for any workout.

6.3.4 GenerateWorkoutPlan

- CanGenerateWeeklyWorkoutPlan(): Tests if a weekly workout plan is generated correctly based on frequency.
- HandlesNonExistentUser(): Ensures correct response when the user does not exist.
- HandlesInvalidFrequency(): Checks handling of frequencies outside the valid range.

6.4 Encryption Helper

6.4.1 Initialization

- ConstructorWithValidKey(): Tests if the constructor correctly initializes with a valid encryption key.
- ConstructorWithInvalidKey(): Checks the behavior when an invalid or empty key is provided.

6.4.2 Encrypt

- EncryptValidString(): Tests if a valid string is correctly encrypted.
- EncryptEmptyString(): Ensures that an empty string is handled appropriately.
- EncryptNullString(): Checks the response to a null string input.

6.4.3 Decrypt

- DecryptValidString(): Tests if a valid encrypted string is correctly decrypted.
- DecryptEmptyString(): Ensures that an empty string is handled appropriately.
- DecryptNullString(): Checks the response to a null string input.
- DecryptCorruptedData(): Tests the behavior when decrypting corrupted or altered data.

6.4.4 Integration

- EncryptDecryptCycle(): Validates that a string can be encrypted and then decrypted back to its original form.

6.5 Social Service

6.5.1 GetUserProfile

- GetUserProfileWithValidId(): Tests if a user profile is correctly retrieved with a valid user ID.
- GetUserProfileWithInvalidId(): Checks the behavior when an invalid or non-existing user ID is provided.

6.5.2 FilterUsers

- FilterUsersWithValidCriteria(): Tests if users are correctly filtered based on provided criteria.
- FilterUsersWithEmptyCriteria(): Ensures that appropriate results are returned when an empty filter is applied.
- FilterUsersWithNullCriteria(): Checks how the method behaves when null is passed as filter criteria.
- PrioritizeFriendsInResults(): Validates if friends are prioritized in the filtered results.
- LimitNumberOfResults(): Ensures that the number of results returned is within the specified limit.

7 Changes Due to Testing

The process of developing our workout application has been significantly influenced by user feedback and testing, leading to an adaptive approach in its evolution. Initially, our focus was predominantly on creating a wide array of features, aiming to compete with the breadth offered by established fitness apps. However, early user feedback highlighted the importance of quality over quantity. Users expressed a need for more intuitive navigation, seamless user experiences, and reliability in the features we offered, rather than an overwhelming variety.

To address these insights, we've prioritized refining the core functionalities of our app. This includes enhancing the user interface for easier navigation, improving the reliability of workout tracking, and ensuring that our workout recommendations are more personalized and scientifically sound. We're also planning to integrate a more robust feedback loop, allowing users to provide immediate responses on workouts, which will help tailor the app's recommendations more closely to individual needs.

Moreover, testing revealed some technical limitations in our initial approach to data management and security. As a result, we're in the process of restructuring our backend to not only secure user data more effectively but also to improve the speed and responsiveness of the app. This involves adopting more scalable cloud services and employing advanced encryption methods for data protection.

Additionally, in response to user requests for a more community-focused feature set, we're developing a social sharing and competition feature. This aims to motivate users through shared fitness goals and achievements, capitalizing on the motivational aspect of social connectivity.

Lastly, recognizing the diverse needs and preferences of our user base, we're expanding the customizability of workouts. This includes offering more granular control over workout types, durations, and intensities, as well as incorporating options for users with specific physical limitations or goals.

In summary, user feedback and testing have been invaluable in steering our development process towards a more user-centered product. While we're yet to implement all these changes fully, our roadmap reflects a commitment to adapting and refining our app in alignment with user needs and expectations. The lessons learned from this iterative process will continue to guide our development philosophy, ensuring that our final product not only meets but exceeds user requirements.

8 Automated Testing

GitHub Actions is used for continuous integration, one of its many tasks being to run tests on branch pushes. This includes front-end Jest tests and back-end XUnit tests. Since only branches with successful builds can merge into the main branch, this maintains the integrity of the main branch. The back-end is deployed only if the merged branch builds successfully, ensuring that in the rare scenario where the integration of a branch leads to either a failed compilation or a failed integration build, the integrity of the deployed web application remains unaffected.

9 Trace to Requirements

The traceability between the functional requirements and the test cases is shown in Table 1. The traceability between the nonfunctional requirements and the test cases is shown in Table 2, 3, and 4.

	UA1	UA2	UA3	UA4	UA5	AI1	AI2	AI3	AI4	AI5	UC1	UC2	UC3	UC4	WH1	WH2	DB1	AF1	AF2	AF3	AF4	AF5
STF-UA-1	X			X	X																	
STF-UA-2	X																					
STF-UA-3	X																					
STF-UA-4				X	X																	
STF-UA-5		X																				
STF-UA-6		X																				
STF-UA-7		X																				
STF-UA-8		X																				
STF-UA-9			X																			
STF-AI-1						X																
STF-AI-2							X															
STF-AI-3								X														
STF-AI-4									X													
STF-AI-5										X												
STF-UC-1											X											
STF-UC-2												X										
STF-UC-3													X									
STF-UC-4													X									
STF-UC-5														X								
STF-WH-1															X							
STF-WH-2																X						
STF-DB-1																	X					
STF-AF-1																		X		X		
STF-AF-2																			X			
STF-AF-3																					X	
STF-AF-4																						X

Table 1: Traceability Matrix Between Test Cases and Functional Requirements

	APR1	APR2	APR3	APR4	APR5	STR1	STR2	STR3	EUR1	EUR2	PER1	PER2	LER1	UPR1	UPR2	ACR1
STN-APR-1	X															
STN-APR-2		X														
STN-APR-3			X													
STN-APR-4				X												
STN-APR-5					X											
STN-STR-1						X										
STN-STR-2							X									
STN-STR-3								X								
STN-EUR-1									X							
STN-EUR-2										X						
STN-PER-1											X					
STN-PER-2												X				
STN-LER-1													X			
STN-UPR-1														X		
STN-UPR-2															X	
STN-ACR-1																X

Table 2: Traceability Matrix Between Test Cases and Nonfunctional Requirements (Part 1)

	PR1	PR2	PR3	PR4	OER1	OER2	OER3	OER4	MSR1	MSR2	MSR3	COMR1	COMR2	COMR3	COMR4	COMR5	COMR6
STN-PR-1	X																
STN-PR-2		X															
STN-PR-3			X														
STN-PR-4				X													
STN-OER-1					X												
STN-OER-2						X											
STN-OER-3							X										
STN-OER-4								X									
STN-MSR-1									X								
STN-MSR-2										X							
STN-MSR-3											X						
STN-COMR-1												X					
STN-COMR-2													X				
STN-COMR-3														X			
STN-COMR-4															X		
STN-COMR-5																X	
STN-COMR-6																	X

Table 3: Traceability Matrix Between Test Cases and Nonfunctional Requirements (Part 2)

	ACR1	ACR2	ACR3	INR1	INR2	INR3	INR4	INR5	INR6	INR7	INR8	PRR1	EHR1	EHR2
STN-ACR-1	X													
STN-ACR-2		X												
STN-ACR-3			X											
STN-INR-1				X										
STN-INR-2					X									
STN-INR-3						X								
STN-INR-4							X							
STN-INR-5								X						
STN-INR-6									X					
STN-INR-7										X				
STN-INR-8											X			
STN-PRR-1												X		
STN-EHR-1													X	
STN-EHR-2														X

Table 4: Traceability Matrix Between Test Cases and Nonfunctional Requirements (Part 3)

10 Trace to Modules

Table 5 shows the traceability between the modules and the requirements.

M1: User Profile Management Module

M2: Workout Generation Module

M3: Data Management Module

M4: User Interaction Module

M5: Reporting and Feedback Module

M6: ChatBot Integration Module

M7: Algorithm Selection Module

M8: Input Format Module

11 Code Coverage Metrics

Code coverage in our back-end is calculated using Coverlet, a cross platform coverage library for .NET. Reports for the test coverage can be found in "SweatSmart/test/coverage/coverage.info". Currently, we have about 50% code coverage in our back-end with test still being written. We expect about 90% code coverage when complete with the remaining 10% involving bad endpoint requests handling.

References

Requirement	**Modules**
UA1	M1, M3, M4, M8
UA2	M1, M3, M4, M7
UA3	M1, M3, M4, M7
UA4	M1, M4, M7
UA5	M1, M4, M7
AI1	M1, M2, M3, M4, M7
AI2	M2, M3, M7
AI3	M2, M3, M5, M7
AI4	M3, M4, M6
AI5	M3, M4, M5, M7
UC1	M1, M2, M3, M4
UC2	M4, M5
UC3	M1, M3, M4
UC4	M3
WH1	M1, M3
WH2	M3, M5
DB1	M1, M2, M3, M4, M5
AF1	M4, M7
AF2	M3, M4
AF3	M1, M2, M3, M4
AF4	M1, M3
AF5	M1, M3, M4

Table 5: Trace Between Requirements and Modules

Appendix — Reflection

Reflecting on the Verification and Validation (VnV) Plan versus the activities actually conducted reveals a learning curve and adaptability in our approach. Initially, our VnV Plan was outlined with a rigid set of activities focused on comprehensive testing across all functionalities. However, as development progressed, the realization of the complexity and interdependencies within the system required us to shift towards a more dynamic and iterative testing approach.

This shift was primarily due to unanticipated technical challenges and feedback from early user testing sessions, which highlighted areas needing more focused testing and refinement than originally planned. For instance, integrating real-time data feeds into the app introduced complexities not fully appreciated at the planning stage, necessitating additional stress testing and security assessments. Similarly, user feedback on usability issues prompted revisions in our UI/UX testing strategies to ensure a more user-friendly experience.

The experience underscored the importance of flexibility in VnV planning and execution. It highlighted the value of iterative development and testing, allowing for adjustments based on real-world feedback and technical challenges as they arise. This approach not only facilitated more effective identification and resolution of issues but also contributed to a more robust and user-centric final product.

Looking forward, the lessons learned from this project will inform future VnV planning. Recognizing the need for adaptability in the face of unforeseen challenges and the invaluable role of user feedback will enable more realistic and effective VnV planning in future projects. This experience has reinforced the notion that while thorough planning is critical, the ability to adapt and revise in response to new information and challenges is equally vital for the success of software development projects.