Project Title: System Verification and Validation Plan for SFWRENG 4G06

Team #7, Team FAAM, SweatSmart

Daniel Akselrod
Jonathan Avraham
Sophie Fillion
Sam McDonald

November 4, 2023

Revision History

Revision Version	Date	Developer(s)	Change
0	Nov 3, 2023	Sophie, Daniel, Sam, Jonathan	First draft

Contents

List of Tables

1 General Information

1.1 Summary

This document outlines the Verification and Validation (V&V) plan for SweatSmart, an AI workout application. The purpose of this V&V plan is to detail the methodologies, processes, and activities employed to ensure that SweatSmart meets the requirements laid out in our team's software requirements specification (SRS) document.

1.2 Objectives

The primary objectives of this plan are as follows:

1.2.1 Verification

To ensure that the algorithms, user interfaces, data model, and other design elements of the application align with the intended specifications.

1.2.2 Validation

To ascertain that when users utilize SweatSmart in their daily workouts, they experience the promised personalized touch, and the app contributes positively to their fitness journey. We also aim to validate that on the technical side, the application is working as expected and as we plan on from the beginning.

1.2.3 User Experience Assurance

While the technical aspects of the application are of utmost importance, this plan also seeks to verify and validate the intuitiveness, ease of navigation, and overall user satisfaction derived from using the application.

1.2.4 Data Integrity and Privacy

Given the sensitivity of health and fitness data, a crucial reason for this V&V plan is to ensure that data collection, storage, and processing adhere to the required standards of security and privacy.

1.2.5 Continuous Feedback and Improvement

To set up a mechanism that gathers useful user feedback, ensuring that any discrepancies, issues, or potential improvements are identified during the V&V process so they can be addressed before or during rollout of the application.

1.3 Relevant Documentation

The main supporting document for this V&V plan is our Software Requirements Specification which enumerates the features, functionalities, and performance requirements that SweatSmart must adhere to.

2 Plan

2.1 Verification and Validation Team

Although all team members will be responsible for writing/understanding all aspects of the project's test suite, each member will have a particular specific role within the area.

Team Member	**Project Verification Role**
Daniel Akselrod	Test Automation Expert: Responsible for all CD/CI testing
Jonathan Avraham	Test Designer/Case Writer: Create test cases
Sam McDonald	Test Strategist/Planner: Defines overall testing strategy
Sophie Fillion	Test Result Analysis: Analyze failing tests and create new issues

Table 1: Roles of the Verification and Validation Team

2.2 SRS Verification Plan

2.2.1 Objective of the Verification Plan

The objective of this SRS verification plan is to ensure that SweatSmarts software complies with both the functional and non-functional requirements specified in the accompanying Software Requirements Specification document. This comprehensive verification process aims to ensure that the software is not only capable of performing its intended functions but also excels in terms of usability, performance, security, and compliance with industry standards. Specifically, the test plan should satisfy

the following: Functional and nonfunctional requirement compliances, safety, user experiences and satisfaction, and the correct testing tools.

2.2.2 Scope

This test plan and verification will cover all of the functional and nonfunctional requirements that have been written in the SRS. This verification plan will describe what will be covered when undergoing verifications of requirements, the test cases and test criteria, the schedule of verification defining the amount of time it will take, the roles and responsibilities, and the acceptance criteria of each test.

2.2.3 Verification Methods/Tests

Testing is crucial in the Software development Life Cycle as it allows developers to find bugs within the system, can help identify whether certain components within the system work as anticipated and if it meets the needs of its stakeholders. Here are a few testing methods that will aid in the verification process:

- **Functional Testing:** Functional tests will be conducted to verify the functionality of the system. The purpose is to test the functional requirements of the system as described in the SRS.
- **Unit testing:** Unit tests verify individual components (units) of the software where they are isolated from the rest. The purpose is to create unit tests for specific functions or modules to verify that they behave correctly.
- **Manual and Automated Testing:** Manual testing are tests that are conducted by people without test scripts. Automated testing is when tests are executed automatically via testing tools and frameworks such as Jest or XUnit.
- **Static and Dynamic Testing:** Static testing means to examine the code without actually executing it to look for coding style and quality. Dynamic testing is used to verify the code by executing it and observing its behavior during runtime.

2.2.4 Test Cases and Criteria

In this section, we write detailed test cases for each requirement in the SRS, including both the functional and non-functional requirements. Each case will include the following:

- **The input data**
- **The expected output data**
- **The criteria for a pass/fail test**
- **The tools used to test the requirement**

2.2.5 Schedule and Result

The verification of the SRS will take place from [start date] to [end date]. The software will be considered verified once all test cases pass for all non-functional and functional requirements.

2.3 Design Verification Plan

2.3.1 Architecture Review

Validate the system design for scalability and robustness before development begins.

2.3.2 UI/UX Testing

Assess the interface and user flow for intuitiveness through user feedback and expert review.

2.3.3 Algorithm Efficiency

Confirm AI algorithm accuracy and backend efficiency through iterative testing with real-world data.

2.3.4 Security Checks

Evaluate application security features against industry best practices and standards.

2.4 Verification and Validation Plan Verification Plan

2.4.1 Plan Review

Implement periodic peer reviews, with a focus on comprehensive coverage and clarity.

2.4.2 Mutation Testing

Use mutation testing to assess the plan's ability to detect faults and improve test cases.

2.4.3 Checklist Creation

Develop checklists to standardize review processes and ensure all critical aspects of the plan are evaluated.

2.5 Implementation Verification Plan

Objective: Ensure the integrity and correctness of the implementation through static verification techniques.

2.5.1 Code Walk-Throughs

- Schedule regular sessions where developers present their code to the team for examination.
- Focus on logical flow, adherence to design specifications, and potential logic errors.
- Document insights and action items for improvement.

2.5.2 Code Inspection

- Formalize a code inspection process with checklists tailored to project standards and best practices.
- Assign roles for moderator, author, scribe, and reviewers to structure the inspection.
- Utilize inspection outcomes to refine coding guidelines and improve code quality.

2.5.3 Static Analyzers

- Integrate automated static analysis tools into the development workflow.
- Configure tools to flag coding standards violations, potential bugs, security vulnerabilities, and performance issues.

- Review and address flagged issues in a timely manner, prioritizing based on severity.

2.5.4 Automated Code Reviews

- Set up custom rules and linting based on the project's coding standards.

2.5.5 Continuous Integration (CI) Checks

- Enforce static verification checks within the CI pipeline.
- Prevent merging of code changes that fail to meet the defined static analysis criteria.

2.6 Automated Testing and Verification Tools

2.6.1 Unit Testing

- With React Native, we employ Jest to test our application's components, functions, and hooks thoroughly. This framework facilitates simulation of user interactions and verification of component behavior.
- XUnit serves as the backbone for testing our .NET Core Web API. It will ensure that all units of application logic are tested for expected outcomes.

2.6.2 Static Code Analysis

- ESLint and Prettier will be used in tandem to maintain code quality and consistency in our React Native codebase, ensuring adherence to our defined coding standards.
- dotnet format and other .NET analyzers will scrutinize our backend code, enforcing the .NET Core's conventions and coding practices.

2.6.3 Continuous Integration (CI)

- We will use GitHub Actions to automate our continuous integration workflow. On every push and pull request, GitHub Actions will trigger the build and test processes for both the mobile and web API components, ensuring that changes do not introduce regressions or break existing functionality.

2.6.4 Code Coverage

- Integrated code coverage tools within Jest for React Native and XUnit for .NET Core will provide insights into the portions of our codebase exercised by unit tests. Our goal is to maintain a high level of code coverage, correlating with fewer production bugs.

2.7 Software Validation Plan

2.7.1 Objective

The objective of this SRS validation plan is to ensure that the software meets user needs and expectations, delivers its appropriate functionality to its users, and runs properly in its environment. It is essentially the process of checking whether or not customers' requirements are being met.

2.7.2 Scope

This validation plan will typically occur after the development phase as its intent is to ensure the satisfaction of user requirements and is ready in the environment that it is being used. Validation checks the software against the actual user and system requirements, ensuring that it provides the intended functionality and value. The validation plan will include the following: Validation methods for testing, test input and criteria, as well as a schedule for the validation aspect of the development cycle.

2.7.3 Validation methods

Usability testing is essential for assessing the user experience and the effectiveness of a product. It is complemented by regression testing, which is crucial for ensuring that new updates or changes to the software don't introduce fresh issues. In tandem, performance testing evaluates the system's response time and resource usage to ensure it operates efficiently. Security testing is another vital component, aimed at identifying and mitigating any vulnerabilities in the code. Lastly, functional testing is carried out to verify that the system operates as intended by executing the code and checking the functionality. Together, these different types of testing form a comprehensive approach to ensuring the quality and reliability of software products.

2.7.4 Test Cases and Criteria

In this section, we write detailed test cases for each requirement in the SRS including both the functional and non-functional requirements. Each case will include the input data, the expected output data, the criteria for a passed/failed test, and the tools used to test the requirement.

2.7.5 Scheduling and result

It is important to note that the validation plan aims for the end product that is ready to be deployed. Therefore, the validation is scheduled to start from [start date] to [end date]. The software will be considered validated once it passes all validation tests successfully and meets user expectations.

3 System Test Description

3.1 Tests for Functional Requirements

The system tests for the functional requirements are broken down into the same subsections found in the SRS for easy traceability.

3.1.1 User Account Testing

This section covers the user account functional requirements (section 5.1 in the SRS).

1. **STF-UA-1: Successful registration**

   Description: A user can create a new account.

   Control: Functional, Dynamic, Manual.

   Initial State: User does not have an existing account; user is on registration page.

   Input: User provides valid email, available username, valid password, agrees to terms and conditions clause and clicks the register button.

   Output: User account is created and account information is stored in the database; user is redirected to the login page.

   Test Case Derivation: User registration is successful when a new user account is created. After a user has imputed valid and complete registration data, a

new account should be created and the user can sign in to the app to access the app's main features.

How test will be performed: The program will run given the specific inputs. Ensure the user is redirected to the login page. Check the database to make sure account information is stored.

2. **STF-UA-2: Missing registration data**

   Description: A user cannot create a new account with incomplete registration information.

   Control: Functional, Dynamic, Manual.

   Initial State: User does not have an existing account; user is on registration page.

   Input: User does not provide email, and/or username, and/or a password and clicks the register button.

   Output: User account is not created and account information is not stored in the database; the app displays an error message prompting the user to provide the missing information.

   Test Case Derivation: User registration should not be created if the registration information is incomplete. A username, an email and a password must be imputed for a user registration to be considered. A new account should not be created if one or more of the inputs is missing.

   How test will be performed: The program will run with one or more inputs missing. Ensure the app displays an error message and prompts for corrections. Check the database to make sure account information is not stored.

3. **STF-UA-3: Unsuccessful registration using existing username**

   Description: A user cannot create a new account if their chosen username is taken.

   Control: Functional, Dynamic, Manual.

   Initial State: User does not have an existing account; user is on registration page.

   Input: User provides valid email, a username that is already in use, a valid password and clicks the register button.

Output: User account is not created and account information is not stored in the database; user is prompted with a message saying that the username is taken and to try choosing another username.

Test Case Derivation: User registration should not be created if username is already in use. If a user inputs an existing username, a new account should not be created and the user must register using a different username.

How test will be performed: The program will run given the specific inputs. Ensure the app displays an error message and prompts for corrections. Check the database to make sure account information is not stored and there are no duplicate usernames.

4. **STF-UA-4: Terms and conditions agreement**

   Description: The app restricts account creation until the user agrees to the terms and conditions.

   Control: Functional, Dynamic, Manual.

   Initial State: User does not have an existing account; user is on registration page.

   Input: User provides valid email, available username, valid password, does not agree to terms and conditions clause and clicks the register button.

   Output: User account is not created and account information is not stored in the database; user is prompted with an error message saying that the user must agree to the terms and conditions clause.

   Test Case Derivation: User registration is not successful if a user does not agree to the terms and conditions agreement. Even if a user has valid registration credentials (i.e. valid email, username, and password), an account cannot be created until the user agrees to the terms and conditions.

   How test will be performed: The program will run given valid registration information but without agreeing to the terms and conditions. Ensure the user is prompted with an error message to agree to the clause. Check the database to make sure account information is not stored.

5. **STF-UA-5: Successful login**

   Description: A user, with an existing account, can successfully log into their account using their account information.

Control: Functional, Dynamic, Manual.

Initial State: User has an existing account; user is on the login page but not logged in.

Input: User provides valid credentials (existing username or email with corresponding password) and clicks login button.

Output: User is successfully logged into their account and can access the app's core features; user is redirected to homepage.

Test Case Derivation: A user who has an existing account but is not currently logged in, will have a successful login after inputting valid credentials. This should result in the user gaining access to their account.

How test will be performed: The program will run given the valid user login credentials. Ensure the user gains access to their account and can access the app's core features. Check that the user is redirected to the homepage.

6. **STF-UA-6: Unsuccessful login using unregistered username or email**

   Description: A user with an invalid username or email cannot log into the app.

   Control: Functional, Dynamic, Manual.

   Initial State: User is on the login page but not logged in.

   Input: User provides unregistered email or username and password, and clicks login button.

   Output: User is not logged into the app and cannot access the app's core features.

   Test Case Derivation: A user who inputs an incorrect email or username that is not associated to a user account will not be able to log into the app.

   How test will be performed: The program will run given the invalid username or email. Ensure the user is not logged into the app and cannot access the app's core features.

7. **STF-UA-7: Unsuccessful login with wrong password**

   Description: A user tries to log into their account with the wrong password.

   Control: Functional, Dynamic, Manual.

   Initial State: User is on the login page but not logged in.

Input: User provides registered email or username but invalid password, and clicks login button.

Output: User is not logged into the app and cannot access the app's core features.

Test Case Derivation: A user who inputs an incorrect password that does not match their account's password will not be able to log into the app.

How test will be performed: The program will run given the invalid username or email. Ensure the user is not logged into the app and cannot access the app's core features.

8. **STF-UA-8: Successful logout**

   Description: A user, who is already logged into their account, can successfully log out of their account.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User clicks the logout button.

   Output: User is successfully logged out of the account and is redirected to the login page.

   Test Case Derivation: When a user is logged in, clicking the logout button should result in successful logout. The app's core features can only be accessed again once the user logs back in.

   How test will be performed: The program will run given the specific input. Ensure the user is successfully logged out and redirected as expected.

9. **STF-UA-9: Update profile information**

   Description: A user can update their profile information, including their physical attributes and fitness goals, and their changes will be saved.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User updates their physical attributes (e.g age, gender, height, weight) and/or fitness goals (e.g. targeted muscle goals, workout frequency, preferred workout type).

Output: User profile is updated and all changes are saved to the user account.

Test Case Derivation: Once logged in, a user can update their profile information and should see their saved changes to their profile.

How test will be performed: The program will run given the specific inputs. Ensure the user can see their saved changes on their profile. Check the database to ensure the new data is updated and stored.

3.1.2 AI Workout Generation Testing

This section covers the AI workout generation functional requirements (section 5.2 in the SRS).

1. **STF-AI-1: Personalized workout**

   Description: The app generates a personalized workout plan based on the user's specified goals and characteristics.

   Control: Functional, Dynamic, Manual.

   Initial State: User has a completed profile with set fitness goals and specific physical attributes.

   Input: User requests a personalized workout plan.

   Output: The app generates a personalized workout plan based on the user's profile (i.e. physical attributes and fitness goals).

   Test Case Derivation: Given that the user has a completed profile, the app will generate a personalized workout that corresponds to the user's fitness goals.

   How test will be performed: The program will run given the specific input. Ensure the user is given a workout plan. Check the user's profile to ensure the workout plan is specific to the user's fitness goals.

2. **STF-AI-2: Workout plan alterations**

   Description: Workout plans are adjusted according to user feedback.

   Control: Functional, Dynamic, Manual.

   Initial State: User has an existing personalized workout plan.

   Input: User makes adjustments to the plan such as preferred exercises, duration of the workout, different intensity level, etc.

Output: The workout plan is updated to reflect the user's feedback.

Test Case Derivation: After receiving a generated workout plan, a user might want to alter it to really suit their needs. After making adjustments to their plan, the app should give the user an updated workout plan that incorporates the changes given.

How test will be performed: The program will run given the user feedback prompts. Ensure the user is given an updated workout plan. Check the workout plan's changes to ensure it reflects the user's feedback.

3. **STF-AI-3: Recursive AI model**

   Description: The system trains and refines the AI model based on user progress over time.

   Control: Functional, Dynamic, Automatic.

   Initial State: User has a workout history.

   Input: Data collected from the user's ratings after completing their workouts.

   Output: The AI mode is trained and updated based on the user's workout data.

   Test Case Derivation: The AI model should be recursively trained so that it can generate more personalized workout plans that align with the user's goals and progress.

   How test will be performed: The system should automatically collect user workout data and train the AI model over time based on the user's progress. This is an ongoing test that does not need manual intervention.

4. **STF-AI-4: AI Chatbot**

   Description: The chatbot is functional and responsive to user prompts.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged onto the app.

   Input: User asks fitness-related questions to chatbot.

   Output: Chatbot provides relevant fitness advice and information.

Test Case Derivation: When a user has a fitness-related question they can interact with the chatbot for a quick answer. The chatbot should provide accurate and relevant responses.

How test will be performed: The chatbot will run given the user prompt. Ensure the user is given relevant and accurate answers.

5. **STF-AI-5: In-workout messages**

   Description: Users receive helpful tips and guidance according to the specific exercise they are performing.

   Control: Functional, Dynamic, Automatic.

   Initial State: User starts a workout.

   Input: User tracks their workout live.

   Output: The app displays messages with exercise tips and guidance throughout the workout.

   Test Case Derivation: When a user is engaged in a live workout, they should receive in-workout messages to help them through exercises.

   How test will be performed: The program will run once the user starts a workout. The app's algorithm should automatically send in-workout messages with exercise tips. Ensure the user receives messages associated with their current exercise throughout their workout.

3.1.3 User Workout Creation

This section covers the user workout creation functional requirements (section 5.3 in the SRS).

1. **STF-UC-1: Calendar integration**

   Description: Users can link their calendar to plan workouts effectively without clashing with their schedule.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User accesses calendar integration feature; user clicks button to link their calendar.

Output: The app integrates the user's calendar successfully and workouts can be planned according to the user's schedule.

Test Case Derivation: After a user links their calendar to their account, the app can generate workout plans more effectively by planning workouts around their schedule.

How test will be performed: Tester will click the button to link the calendar. Ensure the calendar is successfully integrated. Check that the app can plan workouts based on the user's schedule.

2. **STF-UC-2: Workout rating**

Description: Users can rate a workout after completion.

Control: Functional, Dynamic, Manual.

Initial State: User has completed a workout and is prompted with rating form.

Input: User provides rating information and workout feedback.

Output: The app records the user's rating and feedback for future use.

Test Case Derivation: After a user has completed a workout, they will be prompted to rate their workout. After completing the rating, the app will store the user's rating which will serve to recursively train the AI model.

How test will be performed: The program will run given the user's rating feedback. Ensure the app stores user rating data.

3. **STF-UC-3: Custom workout creation**

Description: A user can design and save their own custom workout tailored to their preferences.

Control: Functional, Dynamic, Manual.

Initial State: User is logged into their account.

Input: User designs a custom workout by inputting their preferred exercises and clicks the save button.

Output: The custom workout is successfully saved and can be accessed for future use on the user's account.

Test Case Derivation: A user who wishes to design their own custom workout can do so by inputting the proper inputs. The output will result in a saved

custom workout that can be completed and tracked immediately or at a later date.

How test will be performed: The program will run given the user's custom workout inputs. Ensure the user can access their saved custom workout. Check the database to ensure the workout is saved.

4. **STF-UC-4: Custom workout access**

   Description: A user can access and use their previously created custom workout.

   Control: Functional, Dynamic, Manual.

   Initial State: User has designed and daved a custom workout.

   Input: User accesses their workout library and selects their saved custom workout.

   Output: The app displays the custom workout and can be started for live tracking.

   Test Case Derivation: After a user has saved their custom workout, they must be able to view that workout so that it can be completed and available for live tracking.

   How test will be performed: Tester will access the workout library. Ensure the user can see the saved workout and begin live tracking.

5. **STF-UC-5: Exercise list**

   Description: The app provides a list of all available exercises.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into the app.

   Input: User accesses exercise list.

   Output: The app displays a comprehensive list of all available exercises.

   Test Case Derivation: When the user accesses the list of exercises, the app must display all available exercises to be included in a workout.

   How test will be performed: Tester will access the exercise list. Ensure the app displays all available exercises.

3.1.4 Workout History

This section covers the workout history functional requirements (section 5.4 in the SRS).

1. **STF-WH-1: Display workout history**

   Description: Users can view their workout history.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User accesses their workout history.

   Output: The app displays an accurate history with details of the user's completed workouts.

   Test Case Derivation: A user can access their workout history and the app should display an accurate list of completed workouts.

   How test will be performed: Tester will access the account's workout history. Ensure the app displays all past completed workouts with accurate details.

2. **STF-WH-2: Saving completed workouts**

   Description: The app saves a workout as completed after a user finished their workout.

   Control: Functional, Dynamic, Manual.

   Initial State: User has finished a workout.

   Input: User clicks save completed workout button.

   Output: The app displays a message that the workout has been saved as completed.

   Test Case Derivation: Once a user completes their workout and saves it as completed, the app should display a message to notify the successful saving of the completed workout.

   How test will be performed: The program will run given the specific input. Ensure the user is prompted with a message notifying the successful saving. Check the database to ensure the workout is saved as completed.

3.1.5 Database

This section covers the database functional requirements (section 5.5 in the SRS).

1. **STF-DB-1: Database functionality**

   Description: All information saved on the app by a user, including user account information, workout plans, user feedback, and workout history is stored in the database

   Control: Functional, Dynamic, Manual, Automatic.

   Initial State: User interacts with app.

   Input: The app retrieves and displays user account information, workout plans, user feedback, and workout history.

   Output: The app correctly stores, retrieves, and displays user account information, workout plans, user feedback, and workout history.

   Test Case Derivation: The database should always accurately store, retrieve and display the proper information.

   How test will be performed: Tester will manually access the app's features to update, retrieve and display user-related data. The app automatically retrieves and displays the data requested by the user. Combine manual verification and automatic database operations to ensure that the database functionality is correct.

3.1.6 Additional Features

This section covers the additional features functional requirements (section 5.6 in the SRS).

1. **STF-AF-1: Live workout tracking**

   Description: A user can begin and engage in a planned workout while tracking their progress in real-time.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account and has a planned workout available.

   Input: User begins a planned workout.

Output: The user can track their progress in real time through the duration of their workout.

Test Case Derivation: When a user starts their workout, the app should allow them to engage and track their progress in real-time for immediate feedback and to collect user progress for future use.

How test will be performed: The program will run given the user's input to start the workout. Ensure the app displays the real-time progress of the workout and the user can engage as the workout continues.

2. **STF-AF-2: Push notifications**

   Description: The app delivers notifications as intended.

   Control: Functional, Dynamic, Automatic.

   Initial State: User has enabled push notifications in settings.

   Input: The app generates and sends a push notification.

   Output: The notification is delivered to the user's device as intended.

   Test Case Derivation: After enabling push notification in their settings, a user can receive notifications on their device as intended by the app.

   How test will be performed: The system should automatically push notifications when they are enabled in the settings. This is an ongoing test that does not need manual intervention.

3. **STF-AF-3: Offline access**

   Description: A user can download workouts for offline access.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User downloads a workout.

   Output: The workout is saved onto the user's device.

   Test Case Derivation: If a user wishes to access a workout offline, it can download their desired workout. The workout should be saved locally onto the user's device for offline access.

   How test will be performed: The program will run given the user input. Ensure the workout is downloaded onto the user's device.

4. **STF-AF-4: Social media sharing**

   Description: A user can share information from the app to social media platforms.

   Control: Functional, Dynamic, Manual.

   Initial State: User is logged into their account.

   Input: User accesses social media sharing feature and clicks share button.

   Output: The user can share information to social media platforms.

   Test Case Derivation: After clicking the share button, a user can share information, such as workout achievement and progress, to social media platforms.

   How test will be performed: The program will run given the user input. Ensure the information is shared to the appropriate social media platform(s).

3.2 Tests for Nonfunctional Requirements

The system tests for the non-functional requirements are broken down into the same subsections found in the SRS for easy traceability.

3.2.1 Appearance Requirements

1. **STN-APR-1: Interface Design**

   Description: The app shall have an intuitive and user-friendly interface with minimalist design.

   Type: Dynamic, functional, manual.

   Initial State: App opened for the first time.

   Input/Condition: User interacts with the app interface.

   Output/Result: The user finds the interface intuitive and user-friendly with a minimalist design.

   How the test will be performed: A group of users unfamiliar with the app will interact with it and provide feedback through a usability survey on the intuitiveness of the design. The survey is shown in section 6.2, question 1.

2. **STN-APR-2: Screen adaptability**

Description: The app shall be able to adapt to a variety of screen sizes and aspect ratios.

Type: Dynamic, functional, manual.

Initial State: App opened on various device sizes and aspect ratios.

Input/Condition: Design team views the app's layout and graphics.

Output/Result: The app adapts smoothly to different screen sizes and aspect ratios.

How the test will be performed: The app will be tested on devices with varying screen sizes and aspect ratios to ensure consistent appearance/proper display for each.

3. **STN-APR-3: Feedback**

   Description: The app shall provide visual feedback whenever a user action takes place.

   Type: Dynamic, functional, manual.

   Initial State: User performs various actions (e.g., button presses, form submissions).

   Input/Condition: User actions.

   Output/Result: Visual feedback (e.g., animations, button color changes) is observed.

   How the test will be performed: Testers will interact with the app and note any visual feedback that occurs upon their actions.

4. **STN-APR-4: Reducing delays**

   Description: The app shall load data at a reasonable time so there are no noticeable delays for users.

   Type: Dynamic, functional, manual.

   Initial State: App opened with data-dependent pages.

   Input/Condition: User navigates the app.

   Output/Result: Data loads without noticeable delays.

   How test will be performed: Testers will navigate through various data-dependent sections of the app and measure the time it takes for data to load, ensuring our internet connection is consistent with the typical expected user.

5. **STN-APR-5: Portrait Mode**

   Description: The app shall only support portrait mode.

   Type: Dynamic, functional, manual.

   Initial State: App opened on a device.

   Input/Condition: User attempts to change the device orientation.

   Output/Result: The app remains in portrait mode and does not adjust to landscape.

   How test will be performed: Testers will attempt to change the orientation of their devices to landscape mode and note the app's behavior.

3.2.2 Styling Requirements

1. **STN-STR-1: Colour and styling**

   Description: The app should have consistent styling and design throughout by using the same colour palette.

   Type: Dynamic, functional, manual.

   Initial State: App opened with various pages visible.

   Input/Condition: User navigates through these pages.

   Output/Result: Consistent styling and design is observed and noted.

   How the test will be performed: Users will provide feedback for this requirement in our user experience survey, as shown in section 6.2, question 4.

2. **STN-STR-2: Modern styling**

   Description: The application's styling should be modern and similar to iOS apps.

   Type: Dynamic, functional, manual.

   Initial State: App opened for the first time.

   Input/Condition: User observes app styling

   Output/Result: Styling is modern and mirrors that of typical iOS apps.

   How the test will be performed: A comparison will be made between the app's styling and that of popular iOS apps by the user as prompted by question 8 in the survey in section 6.2.

3. **STN-STR-3: Visual appeal**

   Description: The styling should be visually appealing and align with the app's logo or theme.

   Type: Dynamic, functional, manual.

   Initial State: App opened with branding elements visible (e.g., logo).

   Input/Condition: User observes app's styling elements.

   Output/Result: Styling observed to be aligned with the app's logo or overarching theme.

   How test will be performed: The usability survey in section 6.2, question 4 will help to test this requirement.

3.2.3 Ease of Use Requirements

1. **STN-EUR-1: Intuitiveness of System**

   Description: The system shall be intuitive such that new users can quickly understand basic functions and commands by using intuitive icons.

   Type: Dynamic, functional, manual.

   Initial State: App opened for the first time by a new user.

   Input/Condition: User attempts to use basic functions and commands.

   Output/Result: The user can quickly understand and use basic functions without any guidance.

   How test will be performed: New users will be given tasks to perform using the app. Their ease and speed of understanding will be evaluated. They will also be asked about their experience in question 1 of section 6.2.

2. **STN-EUR-2: Consistency**

   Description: The user interface elements, terminology, and interactions should remain consistent throughout the application's various pages.

   Type: Dynamic, functional, manual.

   Initial State: User navigates through various pages of the app.

   Input/Condition: User interaction with different elements.

   Output/Result: User interface elements and interactions remain consistent throughout the app.

How test will be performed: Users will note if UI elements, terms, and interactions are consistent across different pages when asked about it in question 10 of the survey in section 6.2.

3.2.4 Personalization Requirements

1. **STN-PER-1: Date and Time**

   Description: The app will support local date/time formats based on the user's device settings.

   Type: Dynamic, functional, manual.

   Initial State: App opened on a device set to a non-default date/time format.

   Input/Condition: User observes date and time displays.

   Output/Result: The app displays date/time according to the user's device settings.

   How test will be performed: Devices with various date/time formats will be used to verify the app's adaptability.

2. **STN-PER-2: Personalization settings**

   Description: The app shall stay consistent with the personalization settings on the user's device regarding display and font.

   Type: Dynamic, functional, manual.

   Initial State: App opened on a device with specific display and font settings such as dark mode and bolded font size.

   Input/Condition: User observes app display and font.

   Output/Result: The app's display and font are consistent with the user's device settings.

   How test will be performed: Devices with various display and font settings will be used to test the app's consistency.

3.2.5 Learning Requirements

1. **STN-LER-1: Required Prior Knowledge**

   Description: Users shall not require external resources to navigate the application; everything they need to know to use the app should be taught in-app.

   Type: Dynamic, functional, manual.

Initial State: App opened by a user unfamiliar with it.

Input/Condition: User attempts to navigate and use the app.

Output/Result: User does not need external resources to understand and use the app.

How test will be performed: New users will be asked to perform tasks without external assistance. Their ability to navigate and use the app will be evaluated. They will then be asked about their experience in the survey in section 6.2.

3.2.6 Understandability and Politeness Requirements

1. **STN-UPR1: Tutorial**

   Description: The system should provide a tutorial for first-time users.

   Type: Dynamic, functional, manual.

   Initial State: App opened for the first time.

   Input/Condition: User starts using the app.

   Output/Result: A tutorial is provided for first-time users.

   How test will be performed: First-time users will be observed to see if a tutorial is presented upon initial app use.

2. **STN-UPR-2:FAQ**

   Description: The application shall contain an FAQ to answer expected questions about the application.

   Type: Dynamic, functional, manual.

   Initial State: User accesses the help or settings section.

   Input/Condition: User looks for an FAQ section.

   Output/Result: An FAQ section is available to answer questions about the app.

   How test will be performed: Testers will navigate to the app's help or settings sections to locate and review the FAQ.

3.2.7 Accessibility Requirements

1. **STN-ACR-1:** Accessibility Guidelines

   Description: The system will follow Apple's design guidelines regarding accessible applications.

   Type: Dynamic, functional, manual.

   Initial State: App installed on a device.

   Input/Condition: User with accessibility needs uses the app.

   Output/Result: The app follows Apple's design guidelines for accessible applications.

   How test will be performed: Testers with accessibility needs (e.g., vision or hearing impairment) will use the app, and the design team will note how easily they are able to navigate the app.

3.2.8 Performance Requirements

1. **STN-PR-1: System Capacity**

   Description: The system shall be capable of handling a minimum of 1000 simultaneous users and processing 200 transactions per minute while maintaining performance standards.

   Type: Automated, functional, automated.

   Initial State: Simulation of 100 simultaneous users using the app at once.

   Input/Condition: Users start processing transactions.

   Output/Result: The system can handle 1000 users and process 200 transactions per minute.

   How test will be performed: Automated load testing tools will simulate 1000 users connecting and processing transactions.

2. **STN-PR-2: Availability**

   Description: The system shall achieve a minimum of 99.9% availability, ensuring services are accessible to users at all times, excluding scheduled maintenance windows.

   Type: Non-functional, dynamic, automated.

   Initial State: System running continuously.

Input/Condition: Monitor system availability over a month.

Output/Result: The system achieves a minimum of 99.9% availability.

How test will be performed: System monitoring tools will track uptime over a specified period, to be determined closer to the testing date.

3. **STN-PR-3: Required Throughput**

   Description: The system shall support a data throughput of at least 200 Mbps to ensure data transmission between the server and users is seamless and efficient under peak usage.

   Type: Dynamic, non-functional, automated.

   Initial State: Peak usage period.

   Input/Condition: Large data transfer between server and users.

   Output/Result: Data throughput is at least 200 Mbps.

   How test will be performed: While testing capacity of the system, we will also automate data transfers of more than 200Mbps and measure the system's performance.

4. **STN-PR-4: Response Time**

   Description: The system shall deliver a response time of no more than 2 seconds for 99% of the transaction processed under standard operating conditions.

   Type: Dynamic, non-functional, automated.

   Initial State: System under standard operating conditions.

   Input/Condition: User sends a request to the server.

   Output/Result: Response time is no more than 2 seconds for 99% of the transactions.

   How test will be performed: Response times will be logged and analyzed to ensure 99% fall under the 2-second threshold.

3.2.9 Operational and Environment Requirements

1. **STN-OER-1: Locational Requirements**

   Description: The system shall be able to be used in a gym/workout facility or home workout room.

   Type: Dynamic, functional, manual.

Initial State: App installed on a device.

Input/Condition: User tries to use the app in a gym/workout facility or home workout room.

Output/Result: The app functions correctly without any environmental interference.

How test will be performed: A user will test the app in various gym/workout environments to ensure functionality.

2. **STN-OER-2: OS Requirements**

   Description: The system shall be able to run on an iOS phone.

   Type: Non-functional, dynamic, manual.

   Initial State: App is ready to be installed.

   Input/Condition: User tries to install the app on an iOS phone.

   Output/Result: The app installs, opens, and functions correctly.

   How test will be performed: Installation and basic functionality test on an iOS device.

3. **STN-OER-3: Audio Functionality**

   Description: The system shall allow the audio output to be transmitted through the device's speakers or a connected Bluetooth device.

   Type: Functional, dynamic, manual.

   Initial State: App running with audio features activated.

   Input/Condition: User connects to device's speakers or a Bluetooth device.

   Output/Result: Audio is transmitted clearly without any issues.

   How test will be performed: Test audio output on device's speakers and on different Bluetooth devices.

4. **STN-OER-4: Network Connection**

   Description: The system should connect to the internet for back-end services.

   Type: Non-functional, dynamic, manual.

   Initial State: App installed and requires backend services.

   Input/Condition: User tries to use features that require internet connection.

Output/Result: App connects to the internet and back-end services function correctly.

How test will be performed: Use the app with an active internet connection and verify backend services.

3.2.10 Test for Maintainability and Support

1. **STN-MSR-1:** Error Detection

   Description: The system shall automatically detect, log, and report any errors or system failures to the development and support team without requiring user intervention.

   Type: Functional, dynamic, automated.

   Initial State: The app is functioning with intentional errors introduced.

   Input/Condition: App is running and encounters an error.

   Output/Result: The app should automatically detect, log, and report the error.

   How test will be performed: Developers will introduce intentional errors, run the app, and verify if errors are detected and reported with the necessary feedback statements.

2. **STN-MSR-2: User Feedback**

   Description: The system shall incorporate an accessible user feedback mechanism to enable users to easily report issues.

   Type: Functional, dynamic, manual.

   Initial State: App is open and functional.

   Input/Condition: User accesses the feedback section and submits feedback.

   Output/Result: The feedback is successfully sent and recorded. How test will be performed: Testers will send mock feedback and check if the feedback mechanism is accessible and works properly.

3. **STN-MSR-3: Update Mechanism**

   Description: The system shall support periodic, backward-compatible software updates, notifying users of availability and enabling convenient download and installation while providing information about the changes.

   Type: Non-functional, dynamic, manual.

Initial State: An older version of the app is installed.

Input/Condition: A new update is available.

Output/Result: The user is notified of the update, and upon choosing to update, the app updates smoothly with information about the changes.

How test will be performed: An older version of the app will be installed on a device. When a new update is rolled out, testers will verify the update mechanism by downloading and installing the application and running tests to check for new features/bug fixes.

3.2.11 Test for Compliance Requirements

1. **STN-COMR-1: Data Protection Laws**

   Description: The application will adhere to all data protection laws in the region(s) it operates within.

   Type: Non-functional, static, manual.

   Initial State: The app is designed and ready for review.

   Input/Condition: The app's data handling and storage methods.

   Output/Result: The app should adhere to all data protection laws in the region it operates within.

   How test will be performed: The design team will consult all applicable Canadian laws regarding digital privacy and ensure proper precautions have been made to protect data. Or, where data cannot be protected, users are informed in accordance with the law of the risks associated with using SweatSmart.

2. **STN-COMR-2: Informed Consent**

   Description: The informed consent of users shall be obtained before collecting and processing their personal data.

   Type: Functional, dynamic, manual.

   Initial State: User is registering or using a feature that requires personal data.

   Input/Condition: Registration or usage of personal data-dependent features.

   Output/Result: A clear and comprehensive consent form should be presented to the user.

   How test will be performed: Testers will simulate user registration and verify that informed consent is properly obtained.

3. **STN-COMR-3: Security Measures**

   Description: Robust security measures will be put in place to safeguard user data.

   Type: Non-functional, dynamic, manual.

   Initial State: App is functioning and storing user data.

   Input/Condition: App's data storage and security measures.

   Output/Result: The app should have robust security measures that safeguard user data.

   How test will be performed: The design team will perform a security audit on the app to validate its data safeguarding measures.

4. **STN-COMR-4: Data Collection Transparency**

   Description: The application will be transparent about its data collection.

   Type: Non-functional, static, manual.

   Initial State: App is downloaded and installed.

   Input/Condition: User's first interaction with the app.

   Output/Result: Clear information about data collection practices should be presented.

   How test will be performed: Testers will initiate the app for the first time and verify that the app is transparent about its data collection.

5. **STN-COMR-5: Safe Fitness Guidance**

   Description: The app shall always recommend safe and proven fitness guidance and information.

   Type: Functional, dynamic, manual.

   Initial State: App is open and provides fitness guidance.

   Input/Condition: User accesses fitness guidance or information sections.

   Output/Result: Only safe and proven fitness guidance should be provided.

   How test will be performed: Design team will compare fitness advice from the application to that from experts in fitness for the 50 most common exercises.

6. **STN-COMR-6: Injury Disclaimer**

   Description: The system must state that the application is not responsible for any injuries or accidents resulting from its use.

   Type: Non-functional, static, manual.

   Initial State: App is installed and opened.

   Input/Condition: User's first interaction with fitness guidance or exercise tutorials.

   Output/Result: A clear disclaimer about potential injuries should be presented.

   How test will be performed: Testers will access fitness guidance or tutorials and ensure that the app clearly states its non-responsibility for injuries.

3.2.12 Access Requirements

1. **STN-ACR-1: Strong Password Restrictions**

   Description: The app shall enforce strong password restrictions to ensure users are properly protecting their own data.

   Type: Functional, dynamic, manual.

   Initial State: User is at the registration or password change page.

   Input/Condition: User attempts to set or change a password.

   Output/Result: Password meets specified strength criteria.

   How test will be performed: Testers will attempt to register or change passwords using weak and strong password examples and verify that only strong passwords are accepted.

2. **STN-ACR-2: Authentication Credentials**

   Description: Users shall authenticate themselves with their credentials securely.

   Type: Functional, dynamic, manual.

   Initial State: User is at the login page.

   Input/Condition: User inputs their credentials.

   Output/Result: Successful login with valid credentials; unsuccessful login with invalid credentials.

   How test will be performed: Testers will use valid and invalid credentials to ensure proper authentication.

3. **STN-ACR-3: Authorized Data Access**

   Description: Users will only have access to data and features for which they are authorized.

   Type: Functional, dynamic, manual.

   Initial State: User is logged in.

   Input/Condition: User tries to access data/features.

   Output/Result: Users can only access authorized data/features.

   How test will be performed: Different user profiles will be created with different levels of authorization. Testers will try accessing restricted areas to ensure authorization controls are working.

3.2.13 Integrity Requirements

1. **STN-INR-1: Restrict Sensitive Data**

   Description: Sensitive data shall be restricted from the users of the app.

   Type: Non-functional, static, manual.

   Initial State: App is functioning and displaying user data.

   Input/Condition: Review of displayed data in user profile or other sections.

   Output/Result: No sensitive data is displayed.

   How test will be performed: Testers will review user data sections to ensure no sensitive data is revealed.

2. **STN-INR-2: Data Encryption API**

   Description: Data between the app and the server should be encrypted when an API call is made.

   Type: Non-functional, dynamic, manual.

   Initial State: Data is being transmitted between the app and the server.

   Input/Condition: Sending or receiving data via API.

   Output/Result: Data transmission is encrypted.

   How test will be performed: Design team will monitor data transmissions for encryption.

3. **STN-INR-3: Data Backup**

   Description: The system should regularly back up data automatically to prevent data loss in the event of database failures.

   Type: Non-functional, dynamic, automated.

   Initial State: System is operational.

   Input/Condition: Automated process to backup data.

   Output/Result: Successful backup of data without loss.

   How test will be performed: Testers will trigger the backup process, then will retrieve and verify the data from the backup storage.

4. **STN-INR-4: Local Data Storage**

   Description: Unstored data should be stored locally if the data cannot be updated.

   Type: Functional, dynamic, manual.

   Initial State: Data cannot be updated to the main database.

   Input/Condition: Application tries to save user data.

   Output/Result: Unstored data is saved locally.

   How test will be performed: Testers will simulate server connection failures and see if the app saves data locally.

5. **STN-INR-5: Unique Primary Key**

   Description: The system must incorporate backend validation to enforce the uniqueness of primary keys associated with user accounts in the database.

   Type: Functional, dynamic, automated.

   Initial State: Database with existing user accounts.

   Input/Condition: Registering a new user or changing primary key of an existing user.

   Output/Result: System ensures the uniqueness of primary keys.

   How test will be performed: Testers will attempt to register users with duplicate primary keys or modify existing keys to mimic another user's primary key. The system should reject these attempts.

6. **STN-INR-6: Progress Save On Interruption**

   Description: The system must have a mechanism to store the user's progress when completing a live workout, allowing them to resume their workout from the point of interruption when the application is reopened.

   Type: Functional, dynamic, manual.

   Initial State: User is in the middle of a live workout.

   Input/Condition: Application is forcefully closed or interrupted.

   Output/Result: User's progress is saved.

   How test will be performed: While in the middle of a live workout, testers will forcefully close the app and then reopen it to check if the progress is saved and can be resumed.

7. **STN-INR-7: Exercise Safety Warnings**

   Description: The system should display clear and prominent warnings to users regarding exercise safety.

   Type: Functional, dynamic, manual.

   Initial State: User is accessing exercise instructions.

   Input/Condition: User views different exercises.

   Output/Result: Clear and prominent warnings are displayed for exercises that might pose risks.

   How test will be performed: Testers will navigate through different exercise instructions and ensure safety warnings are prominently displayed.

8. **STN-INR-8: AI Model Updates**

   Description: The AI model system should be constantly updated with accurate practices and the data should be reviewed

   Type: Non-functional, dynamic, automated.

   Initial State: AI model is operational.

   Input/Condition: Feeding the AI model with new practices and reviewing data.

   Output/Result: The AI model should be updated with accurate practices.

   How test will be performed: AI experts will feed new accurate data into the model and monitor how it incorporates the updates.

3.2.14 Privacy Requirements

1. **STN-PRR-1: Encrypted Data**

   Description: Sensitive data should be encrypted both in transit and at rest to protect it from unauthorized access.

   Type: Non-functional, dynamic, manual.

   Initial State: User data is stored or being transmitted.

   Input/Condition: Review of how data is stored and transmitted.

   Output/Result: Data is encrypted in both transit and at rest.

   How test will be performed: Design team will analyze the data storage and transmission methods to ensure encryption.

3.2.15 Error Handling Requirements

1. **STN-EHR-1: Error Handling**

   Description: The system should have robust error handling that doesn't reveal sensitive system information to users in error messages.

   Type: Functional, dynamic, manual.

   Initial State: An error occurs in the application.

   Input/Condition: Triggering various errors in the application.

   Output/Result: Error handling mechanisms work and no sensitive system information is shown.

   How test will be performed: Testers will deliberately induce errors in the application to observe how they are handled

2. **STN-EHR-2: User Feedback Mechanism**

   Description: The system should include a user feedback mechanism, allowing users to report issues, including unexpected application closures.

   Type: Functional, dynamic, manual.

   Initial State: User wants to report an issue.

   Input/Condition: User accesses the feedback mechanism.

   Output/Result: Users can successfully send feedback.

   How test will be performed: Testers will use the feedback mechanism to send feedback and see if it reaches the desired destination.

3.3 Traceability Between Test Cases and Requirements

The traceability matrix for the functional requirements can be found below. Note that since the tests for the non-functional requirements map directly to the non-functional requirements, no matrix was created.

	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 2: Traceability Matrix Between Test Cases and Functional Requirements

4 Unit Test Description

This section is to be completed when the MIS (detailed design document) is completed.

5 Appendix

5.1 Usability Survey Questions

1. On a scale of 1 to 10, how intuitive did you find the app's interface?
2. Did you experience any noticeable delays or lag when using the app? (Options: Always, Often, Sometimes, Rarely, Never)
3. Were you able to easily navigate and understand the app's various pages and features?
4. How would you rate the app's visual appearance and design consistency? (Options: Excellent, Good, Average, Below Average, Poor)
5. How helpful did you find the in-app tutorials and guides?
6. Were there any features or functions you found confusing or difficult to use?
7. How would you describe the overall speed and responsiveness of the app?
8. Was the app's styling and design in line with what you'd expect from a modern iOS application?
9. Did you encounter any issues or difficulties when trying to customize or personalize your app settings?
10. How consistent do you feel terminology and required interactions stayed throughout your navigation of the app on a scale of 1 to 10? (1 being very inconsistent, 10 being very consistent)

Appendix — Reflection

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

   Answer: The successful verification and validation of our project necessitates a diverse set of knowledge and skills that our team needs to harness. To begin, Daniel will need expertise in dynamic testing knowledge. His role will largely revolve around verifying how the software behaves during runtime, ensuring the system reacts as expected to various inputs. Jonathan, given his talent in analyzing code, will delve into static testing knowledge, evaluating the system's code, documentation, and design without actually executing the software. This will be crucial for spotting potential errors at the code level. Sam, who has displayed an inclination towards tool mastery, will focus on becoming proficient with specific validation tools such as Jest for React Native and XUnit for .NET Core. These tools will be pivotal in streamlining the testing process and ensuring that our application is robust. Finally, Sophie will tackle user experience (UX) testing. Given her keen eye for design and user flow, she will be responsible for gauging the intuitiveness of the system, ensuring that users can navigate and utilize the application with ease.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   Answer: To master dynamic testing, Daniel has opted for learning via youtube tutorials due to the large amounts of free, quality content. Jonathan, understanding the value of experience, will seek to to grasp the intricacies of static testing via reading texts and materials. For Sam, the combination of official documentation and community forums offers a strong approach to mastering specific validation tools, ensuring he understands both foundational concepts and practical applications. Sophie recognizes the importance of user feedback in UX testing and will prioritize direct user feedback sessions, offering her insights into user preferences and areas for design improvement. The choices made by each team member reflect a balance between their learning preferences, the project's immediate needs, and the methods most likely to yield

comprehensive understanding.