# Module Interface Specification for SFWRENG 4G06

Team #7, Team FAAM, SweatSmart

Daniel Akselrod
Jonathan Avraham
Sophie Fillion
Sam McDonald

January 18, 2024

# 1 Revision History

| Revision Version | Date | Developer(s) | Change |
|---|---|---|---|
| 0 | January 17, 2024 | Sam, Sophie, Jonathan, Daniel | First draft of Document |

Table 1: Revision History

# 2 Symbols, Abbreviations and Acronyms

| symbol | description |
|---|---|
| M | Module |
| MG | Module Guide |
| MIS | Management Information System |
| R | Requirement |
| SRS | Software Requirements Specification |
| SFWRENG 4G06 | Explanation of program name |

# Contents

# 3 Introduction

This document presents the Module Interface Specifications (MIS) for **SweatSmart**, a fitness application designed to provide evidence-based workout plans using predictive analytics based on user data and methods proven through exercise science. *SweatSmart* aims to lift the veil of mystery from the exercise world and enable people of all ages with little workout experience to infiltrate the market with a straightforward approach to fitness and workout prescriptions.

## 3.1 Complementary Documents

- **System Requirement Specifications (SRS)**: This document outlines the detailed functional and non-functional requirements that *SweatSmart* is designed to fulfill.
- **Module Guide (MG)**: The MG provides an overview of the system's architecture, describing how different modules within *SweatSmart* interact and work together to achieve the application's objectives.

## 3.2 Repository and Implementation

The complete documentation, including implementation details, source code, and additional resources for *SweatSmart*, is maintained and regularly updated in our repository. This can be accessed at: https://github.com/d-akselrod/SweatSmart/.

This MIS document is structured to give a clear understanding of each module within *SweatSmart*, focusing on the interfaces they expose and the interactions they have with each other. By defining these interfaces explicitly, the MIS ensures that any changes or enhancements to individual modules can be made with a comprehensive understanding of their impact on the overall system.

Throughout the development and maintenance of *SweatSmart*, this document will serve as a key reference to ensure consistency, modularity, and scalability in the application's architecture.

# 4 Notation

The structure of the MIS for modules comes from *HoffmanAndStrooper1995*, with the addition that template modules have been adapted from *GhezziEtAl2003*. The mathematical notation comes from Chapter 3 of *HoffmanAndStrooper1995*. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by SFWRENG 4G06.

| Data Type | Notation | Description |
|---|---|---|
| boolean | bool | a result of true or false |
| character | char | a single symbol or digit |
| integer | $\mathbb{Z}$ | a number without a fractional component in (-$\infty$, $\infty$) |
| natural number | $\mathbb{N}$ | a number without a fractional component in [1, $\infty$) |
| sequence | seq | multiple |
| string | string | a sequence of characters |
| real | $\mathbb{R}$ | any number in (-$\infty$, $\infty$) |

The specification of SFWRENG 4G06 uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, SFWRENG 4G06 uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

# 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

| Level 1 | Level 2 |
|---|---|
| Behaviour-Hiding | User Profile Management<br>Workout Generation<br>Data Management<br>User Interaction<br>Reporting and Feedback<br>Chatbot System |
| Software Decision | Algorithm Selection<br>Input Format |

Table 2: Module Hierarchy

# 6 MIS of User Profile Management Module

## 6.1 Module

The User Profile Management Module (M1) is responsible for handling all aspects of user profile data, including account creation, authentication, and personal information management within the SweatSmart application. It ensures the secure handling of sensitive user data and provides essential functionalities for user identification and profile customization.

## 6.2 Uses

M1 uses the Data Management Module (M3) for storing and retrieving user data. It also interfaces with the Algorithm Selection Module (M7) to provide necessary user data for personalized algorithm selection and with the Input Format Module (M7) for data validation and formatting. M1 is used to store all data pertaining to the functionality of our application. From user data, to exercise data, it will encompass our database.

## 6.3 Syntax

### 6.3.1 Exported Constants

- MAX_USERNAME_LENGTH: $\mathbb{Z}$
- MAX_PASSWORD_LENGTH: $\mathbb{Z}$
- MIN_PASSWORD_LENGTH: $\mathbb{Z}$

### 6.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| createUser | username: String, password: String | Bool | UserAlreadyExistsException |
| authenticateUser | username: String, password: String | Bool | AuthenticationException |
| updateUser | username: String, UserProfile: User | Bool | UserNotFoundException |
| deleteUser | username: String | Bool | UserNotFoundException |
| getUserProfile | username: String | UserProfileData | UserNotFoundException |

## 6.4 Semantics

### 6.4.1 State Variables

- userDatabase := Map<String, UserProfileData>

### 6.4.2 Environment Variables

- userProfileDB := a database for storing user information.

### 6.4.3 Assumptions

- createUser is called only after checking that the username does not exist.
- updateUser and deleteUser are called only for existing users.

### 6.4.4 Access Routine Semantics

createUser(username, password):

- transition: N/A
- output: userProfile := User
  Adds a new user with the given username and password to userDatabase.
- exception: N/A

authenticateUser(username, password):

- transition: N/A
- output: Returns true if a username and password match an entry in userDatabase.
- exception: N/A

updateUser(username, UserProfileData):

- transition: N/A
- output: Returns true if the update is successful.
- exception: N/A

deleteUser(username):

- transition: N/A
- output: Returns true if the deletion is successful.
- exception: N/A

getUserProfile(username):

- transition: N/A
- output: userProfile: := User
  Returns the UserProfileData for the given username from userDatabase.
- exception: N/A

### 6.4.5 Local Functions

validateUsername(username := String) → Bool

- Checks if the username meets the specified criteria (e.g., length, format).

validatePassword(password := String) → Bool

- Checks if the password meets the security criteria (e.g., length, complexity).

# 7 MIS of Workout Generation Module

## 7.1 Module

This module (M2) is used to generate workouts for a user based on some personal information. It will take in certain characteristics and goals from the user and then use it to generate a workout that can be used. It will return a workout list filled with exercises with the number of sets and reps corresponding to each, and the time it takes to complete it.

## 7.2 Uses

M2 plays a pivotal role within the broader Management Information System (MIS) by seamlessly integrating with various modules to enhance the fitness experience for users. Leveraging the data stored in the User Profile Module, M2 tailors workout recommendations based on individual characteristics, goals, and preferences. The Data Management Module assists in the storage and retrieval of workout data, enabling the module to refine future recommendations and track user progress. Through the User Interaction Module, M2 provides an intuitive interface for users to input information and receive personalized workout plans, ensuring a user-friendly experience.

## 7.3 Syntax

### 7.3.1 Exported Constants

N/A

### 7.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| generateWorkout | userProfile: UserProfile | List<Objects>exerciseList | MissingProfileException |

## 7.4 Semantics

### 7.4.1 State Variables

- workoutData := List<Workouts>

### 7.4.2 Environment Variables

N/A

### 7.4.3 Assumptions

- Assumes that the User Profile Management Module (M1) provides accurate and up-to-date user characteristics and goals.

### 7.4.4 Access Routine Semantics

generateWorkout(userProfile):

- transition: Utilizes the user profile information to generate a personalized workout and adds the workout data to the workoutData set.
- output: workout := seq of strings
  Returns a workout list filled with exercises, each specifying the number of sets and reps, and the time it takes to complete.
- exception: There is no profile completed.

### 7.4.5 Local Functions

getUserProfileData(userProfile := User) → UserProfileInfo

- Getting the data from the user profile in order to generate the workout.

# 8 MIS of Data Management Module

## 8.1 Module

This module (M3) acts as a data storage, where all data posts, retrievals, and other management operations are completed. It will make sure that the data is served correctly and efficiently whether it may be user or workout data.

## 8.2 Uses

The Data Management module (M3) within the Management Information System (MIS) serves as the backbone for efficient storage and retrieval of critical information, encompassing both user and workout data. This module seamlessly integrates with other components to ensure the seamless functioning of the entire system. The Workout Generator Module leverages M3 to store generated workout data, enabling the module to refine future recommendations and track user progress. The Chatbot System Module benefits from M3 by accessing historical user interactions, enhancing the chatbot's ability to provide personalized responses.

## 8.3 Syntax

### 8.3.1 Exported Constants

N/A

### 8.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| storeUserData | userProfile: User | N/A | N/A |
| getUserData | username: String | UserProfile | UserProfileNotFound |
| storeWorkoutData | workout: Workout | N/A | N/A |
| getWorkoutData | username: String | List<Workouts> | UsernameNotFound |
| storeChatData | interaction: ChatbotInteraction | N/A | N/A |
| getChatData | username: String | List<ChatbotInteraction> | UsernameNotFound |

## 8.4 Semantics

### 8.4.1 State Variables

- userData := Map<username := String, UserProfile>
- workoutData := Map<username := String, List<Workout>>
- chatbotInteractionData := Map<username := String, List <ChatbotInteraction>>
- feedbackData := Map<username := String, List<UserFeedback>>

### 8.4.2 Environment Variables

N/A

### 8.4.3 Assumptions

- Assumes that data provided to store functions is in a valid format.
- Assumes the existence of a secure and reliable storage mechanism.

### 8.4.4 Access Routine Semantics

storeUserData(userProfile):

- transition: Stores the provided user profile data in the userData map.
- output: N/A
- exception: N/A

getUserData(username):

- transition: N/A
- output: userProfile := User
  Returns the user profile data associated with the provided username.
- exception: Raises UserProfileNotFoundException if the specified user profile is not found.

storeWorkoutData(workout):

- transition: Stores the provided workout data in the workoutData map.
- output: N/A
- exception: N/A

retrieveWorkoutData(username):

- transition: N/A
- output: workouts := List<Workouts>
  Returns a list of workout data associated with the provided username.
- exception: Raises UserProfileNotFoundException if the specified user profile is not found.

storeChatbotInteractionData(interaction):

- transition: Stores the provided chatbot interaction data in the chatbotInteractionData map.
- output: N/A

- exception: N/A

retrieveChatbotInteractionData(username):

- transition: N/A
- output: chatMessage := List<ChatbotInteration>
  Returns a list of chatbot interaction data associated with the provided username.
- exception: Raises UserProfileNotFoundException if the specified user profile is not found.

#### 8.4.5 Local Functions

N/A

# 9 MIS of User Interaction Module

## 9.1 Module

This module showcases the user interface and interaction of the application. Through the use of inputs including buttons, text fields, or scrolling, users can interact with the interface in order to achieve the software's functionalities.

## 9.2 Uses

The User Interaction module (M4) serves as the gateway for users to interact with the Management Information System (MIS). It seamlessly integrates with other modules to facilitate user engagement. Linked with the User Profile Module, it provides a user-friendly interface for inputting personal information and preferences. This module collaborates with the Workout Generator Module, allowing users to input characteristics and goals, receiving personalized workout plans. In conjunction with the Chatbot System Module, it enables dynamic back-and-forth messaging, answering fitness-related queries. The Data Management Module ensures efficient handling of user inputs, optimizing the overall user experience and contributing to the MIS cohesive functionality.

## 9.3 Syntax

#### 9.3.1 Exported Constants

N/A

### 9.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| getUserInput | N/A | String | N/A |
| displayMessage | message: String | N/A | N/A |
| storeWorkoutData | input: String | N/A | N/A |

## 9.4 Semantics

### 9.4.1 State Variables

N/A

### 9.4.2 Environment Variables

N/A

### 9.4.3 Assumptions

- Assumes that the user interface components (buttons, text fields, scrolling) are available and functional.

### 9.4.4 Access Routine Semantics

getUserInput():

- transition: Waits for user input from the interface and returns the input as a string.
- output: userIn := String
  Returns the user input as a string.
- exception: NA.

displayMessage(message):

- transition: message := String
  Displays the provided message on the user interface.
- output: None
- exception: None

processUserInput(input):

- transition: input := String
  Processes the user input received from the interface.
- output: None
- exception: None

#### 9.4.5 Local Functions

N/A

# 10 MIS of Reporting and Feedback Module

## 10.1 Module

This module is used to understand user satisfaction by taking in user feedback. The user feedback is collected, processed, and served in order to provide the user with improvements and other workout insights by generating reports.

## 10.2 Uses

The Reporting and Feedback module (M5) is integral to the Management Information System, focusing on user satisfaction and system enhancement. It collaborates with the User Interaction Module to collect and process user feedback. The Data Management Module efficiently stores and retrieves feedback data. Teaming up with the Workout Generator Module, it generates reports for users, providing insights into their progress and suggesting improvements. The Chatbot System Module benefits from this feedback loop, refining responses. Overall, M5 plays a vital role in maintaining user satisfaction, fostering system improvements, and creating a responsive MIS.

## 10.3 Syntax

### 10.3.1 Exported Constants

N/A

### 10.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| SubmitFeedback | feedback: UserFeed | N/A | ProcessingError |
| generateUserReports | Username: String | List<UserReport> | UsernameNotFound |
| generateWorkoutReports | Username: String | List<WorkoutReport> | UsernameNotFound |

## 10.4 Semantics

### 10.4.1 State Variables

- feedbackData: Map<username := String, List<UserFeedback>>

### 10.4.2 Environment Variables

N/A

### 10.4.3 Assumptions

- Assumes that user feedback provided to ‘submitFeedback‘ is in a valid format.
- Assumes that there is a predefined structure for user and workout reports.

### 10.4.4 Access Routine Semantics

submitFeedback(feedback):

- transition: feedback := UserFeed
  Stores the provided user feedback in the feedbackData map.
- output: N/A
- exception: N/A

generateUserReports(username):

- transition: N/A
- output: userRep := List<UserReport>
  Returns a list of user reports based on the feedback associated with the provided username.
- exception: N/A

generateWorkoutReports(username):

- transition: N/A
- output: workoutRep := List<WorkoutReport>
  Returns a list of workout reports based on the feedback associated with the provided username.
- exception: N/A

### 10.4.5 Local Functions

N/A

# 11 MIS of Chatbot System Module

## 11.1 Module

This module (M6) serves the user by answering any fitness-related questions or guiding them towards their goal in their fitness journey. The input to the system includes a message from the user's end. The system will then process the message and provide a response back to the user.

## 11.2 Uses

M6 within the Management Information System (MIS) represents a vital component dedicated to facilitating user engagement and support in the fitness journey. Operating collaboratively with other modules, M6 is designed to answer fitness-related inquiries and guide users toward their health and wellness goals. Through the fully-automated Conversational User Interface (CUI), this module enables seamless interactions between users and an online chatbot, fostering dynamic and informative back-and-forth messaging. Users input messages to the system, seeking advice, information, or assistance in their fitness endeavors.

## 11.3 Syntax

### 11.3.1 Exported Constants

N/A

### 11.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| processMessage | userMessage: String | response: String | N/A |
| retrieveHistory | userProfile: User | List<String> | N/A |

## 11.4 Semantics

### 11.4.1 State Variables

- chatbotInteractions: Set of ChatbotInteraction

### 11.4.2 Environment Variables

N/A

### 11.4.3 Assumptions

- Assumes that the User Profile Management Module (M1) provides accurate and up-to-date user information.

- Assumes that user messages provided to ‘interactWithChatbot‘ are in a valid format.

### 11.4.4 Access Routine Semantics

interactWithChatbot(userMessage):

- transition: userMessage := String
  Processes the user message, generates a chatbot response, and adds the interaction data to the chatbotInteractions set.
- output: response := String
  Returns a response generated by the chatbot based on the user message.
- exception: N/A

retrieveChatbotHistory(userProfile):

- transition: N/A
- output: chatInterac := List<String>
  Returns a list of chatbot interactions for the specified user profile.
- exception: N/A

# 12 MIS of Algorithm Selection Module

## 12.1 Module

This module selects the best algorithm that should be used to generate workouts. Through real-time data and analytics for the user, the module will choose and compare algorithms in which a workout will be generated that best suits the user's needs.

## 12.2 Uses

The Algorithm Selection module (M7) is a crucial component in the Management Information System, dedicated to optimizing workout generation. Operating in real-time, this module assesses user data and analytics to select the most suitable algorithm for generating workouts. By comparing algorithms dynamically, it ensures that the generated workout aligns precisely with the user's needs. Collaborating with the Data Management Module for real-time data access, the Algorithm Selection Module enhances the effectiveness of the Workout Generator Module. This module plays a pivotal role in adapting workout recommendations based on evolving user profiles and preferences, contributing to the overall personalization and efficiency of the fitness management system.

## 12.3 Syntax

### 12.3.1 Exported Constants

N/A

### 12.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| selectAlgorithm | userProfile: User | algorithm: String | AlgorithmNotFoundException |

## 12.4 Semantics

### 12.4.1 State Variables

- userData := Map<username:= String to UserProfile>

### 12.4.2 Environment Variables

N/A

### 12.4.3 Assumptions

- Assumes that the data received is accurate and up-to-date
- Assumes the available algorithms are property implemented and configured

### 12.4.4 Access Routine Semantics

selectAlgorithm():

- transition: userProfile := User
  compares available algorithms based on user data
- output: algorithm := String
  returns selected algorithm for Workout Generator Module
- exception: None

### 12.4.5 Local Functions

N/A

# 13 Appendix

## 13.1 Reflection

One limitation of our proposed solution is the accuracy of the generated workouts. The accuracy for the personalized workouts, specifically for future workouts to build upon a user's progress, relies heavily on user data, which must be both available and accurate. If a user provides inaccurate and/or incomplete data, this will lead to suboptimal workout recommendations and thus affect the results. For example, a user who does not provide feedback on their progress of completed workouts will likely see much less progress in their fitness journey, as the workout recommendations are not being modified and updated according to the user's progress. Another limitation lies in the accuracy of personalized workout exercises. By using evidence-based studies and research as the backbone of our app, we have seen that workouts are more or less prescribed based on a user's situation.

Other potential solutions that may not rely as much on user data and feedback are not as restricted and dependent on user input. However, these solutions will not cater to the individual's progress and thus produce suboptimal results. Therefore, choosing to have our solution rely on user feedback is the best option to provide the best results. With this in mind, we will focus on user engagement to motivate users to provide us with that data to provide them with the best recommendations, while making the process as short and simple as possible. We have designed our application with software design principles in mind. Our module decomposition implements a high-cohesion, low-coupling approach to ensure independence.