Module Interface Specification for SFWRENG 4G06A

Team #25, RapidCare
Pranav Kalsi
Gurleen Rahi
Inreet Kaur
Moamen Ahmed

January 12, 2025

1 Revision History

Date	Version	Notes
Jan8	Rev0	Added MIS of UI
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS document

Symbol	Description
MG	Module Guide
M	Module
MIS	Module Interface Specification
API	Application Programming Interface
MFA	Multi-Factor Authentication

Contents

3 Introduction

The following document details the Module Interface Specifications for the RapidCareapplication.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/PKALXI/RapidCare/blob/main/docs/Design/SoftArchitecture/MG.pdf.

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. 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 4G06A.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of SFWRENG 4G06A 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 4G06A 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**
Hardware-Hiding	None
Behaviour-Hiding	User Authentication Module API Module Administrator View Module Client View Module Diagnosis Data Module Medicine Data Module
Software Decision	App Module Report Generating Module Transcription Module Classification Module Diagnosis Prediction Module Medicine Prediction Module Administrator Account Management Module Patient Account Management Module

Table 1: Module Hierarchy

6 MIS of App Module

Gurleen [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

7 MIS of User Authentication Module

Moamen

7.1 Module

User Authentication

7.2 Uses

Provides secure login and session management. It validates user credentials and enforces multi-factor authentication (MFA).

7.3 Syntax

7.3.1 Exported Constants

- `MAX_LOGIN_ATTEMPTS`: Maximum allowed login attempts before locking the account.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`login`	Username, Password	Session Token	Invalid credentials
`logout`	Session Token	Boolean	Invalid/expired session
`resetPassword`	Email Address	Boolean	Invalid email address

7.4 Semantics

7.4.1 State Variables

`activeUsers`: Tracks currently logged-in users.

7.4.2 Environment Variables

- Database for storing user credentials and session data.

7.4.3 Assumptions

- Passwords follow defined complexity rules.

7.4.4 Access Routine Semantics

`login()`:

- **Transition:** Validates credentials and generates a session token.
- **Output:** Active session token.
- **Exception:** Invalid credentials.

`resetPassword()`:

- **Transition:** Sends a reset link to the provided email.
- **Output:** Confirmation status.
- **Exception:** Invalid email address.

7.4.5 Local Functions

`encryptPassword(password: String) -> String`: Encrypts a plain-text password.

8 MIS of API Module (OAuth)

Moamen

8.1 Module

OAuth API Module

8.2 Uses

This module facilitates secure authentication and authorization processes. Implements the OAuth 2.0 protocol to manage user authentication, issue access tokens, and validate token requests for secure resource access.

8.3 Syntax

8.3.1 Exported Constants

- `TOKEN_EXPIRY`: Defines the duration of token validity (e.g., 3600 seconds).
- `AUTH_URL`: URL endpoint for authorization.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`authorize`	Client ID, Scope	Authorization Code	Invalid credentials
`getToken`	Auth Code, Client Secret	Access Token	Expired/invalid code
`validateToken`	Access Token	Boolean	Expired/invalid token

8.4 Semantics

8.4.1 State Variables

`activeTokens`: Stores active access tokens and their metadata.

8.4.2 Environment Variables

- Requires a stable database connection for token storage.
- Relies on network connectivity for OAuth communications.

8.4.3 Assumptions

- The external client configurations align with OAuth 2.0 standards.
- Tokens are used within their defined expiry period.

8.4.4 Access Routine Semantics

`authorize()`:

- **Transition:** Generates an authorization code upon successful client validation.
- **Output:** Authorization code.
- **Exception:** Invalid client credentials.

`getToken()`:

- **Transition:** Issues an access token and stores it in `activeTokens`.
- **Output:** Access token.
- **Exception:** Invalid or expired authorization code.

8.4.5 Local Functions

`hashSecret(secret: String) -> String`: Hashes the provided client secret for secure storage.

9 MIS of Administrator View Module

Inreet

9.1 Module

Administrator

9.2 Uses

API Module (OAuth)
User Authentication Module
Account Management

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
Administrator	r	React.component	-

9.4 Semantics

9.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

9.4.2 Environment Variables

Screen interface
Keyboard

9.4.3 Assumptions

User has a functional screen and keyboard.

9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

10 MIS of Client View Module

Inreet [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

10.1 Module

[Short name for the module —SS]

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

10.4 Semantics

10.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

10.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of Report Generating Module

Gurleen [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

11.1 Module

[Short name for the module —SS]

11.2 Uses

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

11.4 Semantics

11.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

11.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

12 MIS of Transcription Module

Gurleen [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

12.1 Module

[Short name for the module —SS]

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of Classification Module

Gurleen [Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

13.1 Module

[Short name for the module —SS]

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

14 MIS of Diagnosis Prediction Module

14.1 Module

DiagnosisPred

14.2 Uses

- Report Generation Module. Specific inputs for exported access programs:
  - Past medical history : text
  - Symptoms : text
  - User characteristics :
    - Age : integer
    - Weight : integer
- Preprocessing Module
- Tensorflow
- Scikit-Learn
- Flask
- Flask-CORS
- Diagnostic Database (for training the model) [18]

14.3 Syntax

14.3.1 Exported Constants

The exported constants for this module would a prediction of the possible diagnosis.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
Flask Application (Python)	Symptoms, Past medical History, User characteristics	Prediction of possible Diagnosis	InputDimError

14.4 Semantics

14.4.1 State Variables

- model: Tensorflow model

14.4.2 Environment Variables

There will be no environment variables that this module with interact with.

14.4.3 Assumptions

- Patients are not making up symptoms and all input features are accurate.

14.4.4 Access Routine Semantics

diagnosePatient(request : FormData):

- transition: preprocess the request body (FormData) which contain the inputs as specified above using the preprocess local function.
- output: Returns the predicted diagnosis for the patient.
- exception: InputDimError - The expected request body items were not received or were in the wrong format.

14.4.5 Local Functions

preProcessData(pastHistory: String, symptoms: String, user -¿ age, weight):

- transition: Preprocess the text using TF-ID and normalize the continuous inputs.
- output: Return the preprocessed data.
- exception: InputDimError - The expected arguments were not received or were in the wrong format.

15 MIS of Medicine Prediction Module

15.1 Module

MedPred

15.2 Uses

- Report Generation Module. Specific inputs for exported access programs:
  - Past medical history : text
  - Symptoms : text
  - User characteristics :
    - Age : integer
    - Weight : integer
  - Physician confirmed diagnosis.
- Preprocessing Module
- Tensorflow
- Flask
- Flask-CORS
- Scikit-Learn
- Medicine Database (for training the model) [19]

15.3 Syntax

15.3.1 Exported Constants

The exported constants for this module would be a prediction of the possible medicine to treat the diagnosis.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
Flask Application (Python)	Symptoms, Past medical History, User characteristics, Diagnosis	Prediction of possible Medicine	InputDimError

15.4 Semantics

15.4.1 State Variables

- model: Tensorflow model

15.4.2 Environment Variables

There will be no environment variables that this module will interact with.

15.4.3 Assumptions

- All input features are accurate.

15.4.4 Access Routine Semantics

medicatePatient(request : FormData):

- transition: preprocess the request body (FormData) which contain the inputs as specified above using the preprocess local function.
- output: Returns the predicted medicine for the patient.
- exception: InputDimError - The expected request body items were not received or were in the wrong format.

15.4.5 Local Functions

preProcessData(diagnosis: String, pastHistory: String, symptoms: String, user -¿ age, weight):

- transition: Preprocess the text using TF-ID and normalize the continuous inputs, finally the diagnosis will be encoded using a LabelEncoder.
- output: Return the preprocessed data.
- exception: InputDimError - The expected arguments were not received or were in the wrong format.

16 MIS of Administrator Account Management Module

Inreet [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

16.1 Module

[Short name for the module —SS]

16.2 Uses

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

16.4 Semantics

16.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

16.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

16.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

16.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

16.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

17 MIS of Patient Account Management Module

Inreet [Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

17.1 Module

[Short name for the module —SS]

17.2 Uses

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

17.4 Semantics

17.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

17.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

17.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

17.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

17.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

18 MIS of Diagnosis Data Module

Pranav

18.1 Module

DiagnosisData

18.2 Uses

- Database of Diagnosis Prediction Data (features -¿ labels)

18.3 Syntax

18.3.1 Exported Constants

- Trained Model

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
addData	id:string, value: FormData	-	InputDimError
deleteData	id:string	-	IdNotFound
updateData	id:string, value: FormData	-	InputDimError

18.4 Semantics

18.4.1 State Variables

dbConnection : Database connection point.

18.4.2 Environment Variables

N/A

18.4.3 Assumptions

Database containing the training data for the Diagnosis prediction module exists/

18.4.4 Access Routine Semantics

addData(id:string, record: FormData):

- transition: Add record to database.
- output: N/A
- exception: InputDimError-Request data contained wrong typing or incorrect format attributes.

deleteData(id:string):

- transition: Delete record from the record with the attitude.
- output: N/A
- exception: IdNotFound - Provided Id does not exist.

updateData(id:string, record: FormData):

- transition: Update record in database based on key.
- output:
- exception: InputDimError-Request data contained wrong typing or incorrect format attributes.

18.4.5 Local Functions

N/A

19 MIS of Medicine Data Module

Pranav

19.1 Module

MedData

19.2 Uses

- Database of Medicine Prediction Data (features -¿ labels)

19.3 Syntax

19.3.1 Exported Constants

- Trained Model

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
addData	id:string, value: FormData	-	InputDimError
deleteData	id:string	-	IdNotFound
updateData	id:string, value: FormData	-	InputDimError

19.4 Semantics

19.4.1 State Variables

dbConnection : Database connection point.

19.4.2 Environment Variables

N/A

19.4.3 Assumptions

Database containing the training data for medicine prediction module exists.

19.4.4 Access Routine Semantics

addData(id:string, record: FormData):

- transition: Add record to database.
- output: N/A
- exception: InputDimError-Request data contained wrong typing or incorrect format attributes.

deleteData(id:string):

- transition: Delete record from the record with the attitude.
- output: N/A
- exception: IdNotFound - Provided Id does not exist.

updateData(id:string, record: FormData):

- transition: Update record in database based on key.
- output:
- exception: InputDimError-Request data contained wrong typing or incorrect format attributes.

19.4.5 Local Functions

N/A

20 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)