Module Interface Specification for Software Engineering

Team 1, BANDwidth
Declan Young
Ben Dubois
Nathan Uy
Aidan Mariglia

March 18, 2025

1 Revision History

Date	Version	Notes
January 17 2025	1.0	Initial design of the MIS

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/AidanMariglia/SOCAlgoTestPlatform/blob/main/docs/SRS/SRS.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for **SOCAlgoTestPlatform**.

Battery state of charge (SOC) estimation is challenging, requiring specialized algorithms. Standardized testing is necessary to determine which of the hundreds of SOC estimation approaches proposed yearly are the best. This project will expand upon an existing, early stage online SOC estimation algorithm testing tool. The tool is Matlab based, receives submissions through a Google form, and tests algorithms in serial on a server at McMaster. This approach is not scalable though since testing each algorithm takes an hour or more and the software regularly crashes due to unhandled errors from the submitted algorithms. The project objectives are to create: (1) A cloud based software implementation which can test multiple algorithm submissions in parallel, (2) A secure algorithm submission portal which prevents malware attacks etc., (3) A web interface which reports and compares algorithm performance (see Kaggle as an example), (4) A robust version of the model testing software which can handle any error. These software improvements will allow the testing tool to be scaled up to having several hundred active users.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/AidanMariglia/SOCAlgoTestPlatform.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). 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 Software Engineering.

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 (-∞, ∞)
dictionary	dict	a key value pair
query set	QuerySet	the set of items returned from a query to the database
account	Account	A user account containing id, username, email and organization string fields
model	Model	A model that is being submitted for testing
result	Result	The test results for a submitted algorithm
test data	TestData	Constant test data for the test suite to use
submission	Submission	A submitted algorithm and relevant metadata
user	user	A username and password pair
file	file	A file that could be .csv (comma separated values), .m (matlab file), or .png (portable network graphics)

The specification of Software Engineering 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, Software Engineering 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 Module	M1
Behaviour-Hiding Module	Login Registration Intro Home Test Algorithm Submit Submission Leaderboard Results Admin Control Panel
Software Decision Module	App User Authentication Database Management Web Server Model Execution Test Data

Table 1: Module Hierarchy

6 MIS of Login

6.1 Module

Login

6.2 Uses

UserAuthentication (17)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
handleLogin	username, password: str	-	-

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

screen: The screen that the view wll be displayed on

6.4.3 Assumptions

The UserAuthentication module handles login validation correctly.

6.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the login module
- output: None
- exception: None

handleLogin(username, password):

- transition: Update the *screen* to display Home page | Update the *screen* to display Login page with an error message.
- output: None
- exception: None

6.4.5 Local Functions

None

7 MIS of Registration

7.1 Module

Registration

7.2 Uses

UserAuthentication (17)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
handleRegistration	email, username, password, confirm_password: str	-	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

screen: The screen that the view wll be displayed on

7.4.3 Assumptions

UserAuthentication contains the registration functionality

7.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the registration module
- output: None
- exception: None

handleRegistration(email, username, password, confirm_password):

- transition: Update the screen to display Login page | Update the screen to display Registration page with an error message.
- output: None
- exception: 400 (if the user input is invalid), 409 (if the specified user already exists)

7.4.5 Local Functions

None

8 MIS of Intro

8.1 Module

Intro

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

screen: The screen that the view wll be displayed on

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the intro module
- output: None
- exception: None

8.4.5 Local Functions

None

9 MIS of Home

9.1 Module

Home

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

screen: The screen that the view wll be displayed on

9.4.3 Assumptions

The WebServer module handles the routing functionality.

9.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the home module
- output: None
- exception: None

9.4.5 Local Functions

None

10 MIS of TestAlgorithm

10.1 Module

TestAlgorithm

10.2 Uses

Test Data (21)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
executeTest	model:Model, testData:TestData	result:Result	modelError

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

screen: The screen that the view wll be displayed on

10.4.3 Assumptions

The test data is provided by the TestData module.

10.4.4 Access Routine Semantics

executeTest(model, testData):

- transition: None
- output: Result (after executing test algorithm against TestData)
- exception: modelError (If the test encounters an error during execution)

10.4.5 Local Functions

None

11 MIS of Submit

11.1 Module

Submit

11.2 Uses

DatabaseManagement (18)

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
handleSubmit	-	-	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

screen: The screen that the view wll be displayed on

11.4.3 Assumptions

The WebServer module handles the routing functionality. The user must be logged in to submit a model.

11.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the submit module
- output: None
- exception: None

handleSubmit():

- transition: Update the *screen* to display Submission page with new submission | Update the *screen* to display Submit page with an error message. Submissions will begin in a pending state, and will enter the started state once the number of submissions running is less than the maximum number of concurrent submissions allowed.
- output: None
- exception: None

11.4.5 Local Functions

validateFile(file: file):

- transition: None
- output: None
- exception: InvalidFileFormat | FileTooLarge Error

12 MIS of Submission

12.1 Module

Submission

12.2 Uses

DatabaseManagement (18)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
downloadData	-	results: file	-

12.4 Semantics

12.4.1 State Variables

data: QuerySet

12.4.2 Environment Variables

screen: The screen that the view wll be displayed on

12.4.3 Assumptions

The WebServer module handles the routing functionality.
The download functionality is disabled if there are no files to be downloaded.
The submissionId is passed in by the WebServer module

12.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the submission module
- output: None
- exception: None

downloadData():

- transition: None
- output: zipped file containing csv and png files
- exception: None

12.4.5 Local Functions

getData(data: QuerySet, submissionId: $\mathbb{Z}$):

- transition: data := The results from submissionId
- output: None
- exception: None

13 MIS of Leaderboard

13.1 Module

Leaderboard

13.2 Uses

DatabaseManagement (18)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
filter	category, condition: str	filtered data	-
sort	category, sortBy: str	sorted data	-

13.4 Semantics

13.4.1 State Variables

appliedFilters: dict
appliedSort: dict
data: QuerySet

13.4.2 Environment Variables

screen: The screen that the view wll be displayed on

13.4.3 Assumptions

The WebServer module handles the routing functionality.
The category, condition and sortBy fields are all valid values.

13.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the leaderboard module

- output: None
- exception: None

filter():

- transition: appliedFilters := dictionary with category and condition as keys. data := Filtered data based on appliedFilters
- output: None
- exception: None

sort():

- transition: appliedSort := dictionary with category and sortBy as keys. data := Sorted data based on appliedSort
- output: None
- exception: None

13.4.5 Local Functions

filterData(data: QuerySet, category: str, condition: str):

- output: Filtered data based on category and condition
- exception: 400 for invalid categories/conditions specified by the user

sortData(data: QuerySet, category: str, sortBy: str):

- output: Sorted data based on category and sortBy
- exception: None

getData(data: QuerySet):

- transition: data := all users' submission results in the database
- output: None
- exception: None

14 MIS of Results

14.1 Module

Results

14.2 Uses

DatabaseManagement (18)

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
display	-	-	-
filter	category, condition: str	filtered data	-
sort	category, sortBy: str	sorted data	-

14.4 Semantics

14.4.1 State Variables

appliedFilters: dict
appliedSort: dict
data: QuerySet

14.4.2 Environment Variables

screen: The screen that the view wll be displayed on

14.4.3 Assumptions

The WebServer module handles the routing functionality.
The category, condition and sortBy fields are all valid values.

14.4.4 Access Routine Semantics

display():

- transition: Update the *screen* to display an updated view of the results module

- output: None
- exception: None

filter():

- transition: appliedFilters := dictionary with category and condition as keys.
  data := Filtered data based on appliedFilters
- output: None
- exception: None

sort():

- transition: appliedSort := dictionary with category and sortBy as keys.
  data := Sorted data based on appliedSort
- output: None
- exception: None

14.4.5 Local Functions

filterData(data: QuerySet, category: str, condition: str):

- output: Filtered data based on category and condition
- exception: None

sortData(data: QuerySet, category: str, sortBy: str):

- output: Sorted data based on category and sortBy
- exception: None

getData(data: QuerySet):

- transition: data := all submission results in the database of the current user.
- output: None
- exception: None

15 MIS of AdminControlPanel

15.1 Module

AdminControlPanel

15.2 Uses

Database Management (18)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
approve_user_request	user_id:str	-	notFound
delete_user	user_id:str	-	notFound
delete_submission	user_id:str	-	notFound

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

approve_user_request(user_id):

- transition: User account enters active state
- output: None
- exception: notFound

delete_user(user_id):

- transition: User account delete from database
- output: None
- exception: notFound

delete_submission(submission_id):

- transition: Submission deleted from database
- output: None
- exception: notFound

15.4.5 Local Functions

None

16 MIS of App

16.1 Module

App

16.2 Uses

Web Server (19), Model Execution (20)

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
startApp	-	-	-

16.4 Semantics

16.4.1 State Variables

webserver: The webserver the application is running

16.4.2 Environment Variables

None

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

startApp():

- transition: Initialize the webserver for the application (*webserver* is initialized)
- output: None
- exception: None

16.4.5 Local Functions

None

17 MIS of UserAuthentication Module

17.1 Module

UserAuthentication

17.2 Uses

Database Management (18)

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
register	dict of username, password, email, and organization	Account	registrationFailure
authenticate	dict of username and password	-	authenticationFailure
getUsername	-	String	-
getOrganization	-	String	-

17.4 Semantics

17.4.1 State Variables

username: String
email: String
organization: String

17.4.2 Environment Variables

None

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

register(username, password, email, organization):

- transition: None
- output: The account that was created
- exception: *exc* := registrationFailure if the use input user parameters already existed/were invalid

authenticate(username, password):

- transition: Authenticates the user and retrieves a token for authorization of other actions
- output: None
- exception: *exc* := authenticationFailure if the credentials used were invalid

getUsername():

- transition: None
- output: *username* := The username of the authenticated user
- exception: None

getOrganization():

- transition: None
- output: *organization* := The organization of the authenticated user
- exception: None

17.4.5 Local Functions

None

18 MIS of DatabaseManagement

18.1 Module

DatabaseManagement

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_user	user_id:str	user:User	notFound
store_user	user:User	user_id:str	validationError
get_submission	submisison_id:str	submission:Submission	notFound
store_submission	submission:Submission	submission_id:str	validationError

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

database: The database server which the module will communicate with.

18.4.3 Assumptions

Database is online and has been built with schema matching the object definitions for user and submisison.

18.4.4 Access Routine Semantics

get_user(user_id):

- transition: None
- output: user

- exception: notFound

store_user(user):

- transition: User data stored in database server
- output: user_id
- exception: validationError

get_submission(submission_id):

- transition: None
- output: submission
- exception: notFound

store_submission(submission):

- transition: submission data is stored in database server
- output: submission_id
- exception: validationError

18.4.5 Local Functions

None

19 MIS of WebServer

19.1 Module

WebServer

19.2 Uses

Home (9), Intro (8), Hardware Hiding, Login (6), Registration (7), Submission (12), Admin Control Panel (15), Leaderboard (13), Results (14), Submit (11)

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
start_server	-	-	ServerError

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

Network: Network environment to allow for HTTP traffic

19.4.3 Assumptions

URL patterns are correctly configured to route to their corresponding handler

19.4.4 Access Routine Semantics

start_server():

- transition: Initializes the webserver and begins accepting HTTP requests
- output: None
- exception: None

19.4.5 Local Functions

route_request(request):

- transition: None
- output: Appropriate HTTP response
- exception: HTTP404 (when no handler exists for a URL pattern), Server Error

20 MIS of ModelExecution

20.1 Module

ModelExecution

20.2 Uses

TestAlgorithm (10), DatabaseManagement (18)

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
executeModel	model:Model	Result	invalidModel
validateModel	model:Model	Boolean	None

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

None

20.4.3 Assumptions

None

20.4.4 Access Routine Semantics

executeModel(model):

- transition: None
- output: *out* := Result of the model that was executed
- exception: *exc*:= InvalidModel if input model failed validation

validateModel(model):

- transition: None
- output: *out* := True (for valid model) | False (for invalid model)
- exception: None

20.4.5 Local Functions

None

21 MIS of TestData

21.1 Module

TestData

21.2 Uses

None

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
submitTestData	testData:TestData	-	InvalidTestData

21.4 Semantics

21.4.1 State Variables

testData: TestData

21.4.2 Environment Variables

None

21.4.3 Assumptions

None

21.4.4 Access Routine Semantics

submitTestData(newTestData):

- transition: *testData*:= newTestData
- output: None
- exception: *exc* := InvalidTestData (if the input test data fails validation)

21.4.5 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

22 Appendix

Appendix — Reflection

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)

Nathan Uy

1. I liked that our modules were very well-defined, making the writing of MIS easy.

2. An issue I encountered was deciding which modules were necessary. It is very easy just to add as many modules as I want. To address this, every time I added a module, I ask myself does this module separate a distinct concern or responsibility? As a result, it limited the number of our modules to just the necessary ones.

Declan Young

1. While writing this deliverable, creating the MIS for our application went well. This is because once we had broken the application into different modules with the MG, it was relatively straight-forward to create specification for these different modules, in order to satisfy the requirements of our application.

2. One of the pain points we encountered while writing this deliverable was determining how to break the application into different modules in the MG. This was a pain point because it was quite difficult to find a balance between breaking it into too many simple modules, versus not breaking it into enough modules, therefore leading to modules that were too general and complex. Another pain point was creating the timeline for all of the modules that needed to be implemented. This was difficult because it was hard

to determine the complexity of the different modules in order to properly estimate the necessary time to complete it, as well as to fairly distribute the implementation of all the modules between team members fairly.

Ben Dubois

1. While writing this deliverable, creating the MIS for each module went well as we already had each module clearly defined within the MG. This ensured that creating the MIS for each module was just adding a few extra details to each module and refining our definitions.

2. One of the pain points was creating the Use Hierarchy between modules. For this we had to determine which modules needed to be stand alone and which modules could use the existing features of another module. Therefore, for each new module we needed to look through all existing modules to see if there could be any use relation and this was time consuming. To resolve this, we started by defining all of the modules that we knew for sure could use an existing module and then created stand alone modules after this. This made the process much faster.

Aidan Mariglia

1. During this deliverable, the module decomposition went well. We were able to create clear lines of where the modules exist, defining the system structure and simplifying the process of implementation.

2. Producing the access routine semantics for each module ahead of time was more of a pain point. Trying to foresee possible needs of modules which depend on other modules was difficult, and left me with the feeling there will be modifications needed when the implementation stage begins.

Team

3. One of the major design decisions that originated from Dr. Kollmeyer was the design decisions regarding the user interface. Dr. Kollmeyer had a fairly specific vision for what the UI should look like, so after some discussions and elicitation of details, we were able to create the design consisting of both the different modules for the UI, as well as the visual design for the UI (which is seen in the Figma design).

   For the decisions that we did not consult our client, we instead brainstormed as a group for all major design decisions. We then ensured that all design decisions we made as a group followed the requirements for our application as well as it following the different properties/characteristics of design that we were trying to achieve.

4. While creating our design document, we realized that the SRS document needed to be updated because we realized that some of our requirements were not necessary so those requirements were removed. Also, we noticed that some of our requirements were ambiguous and needed some clarification.

5. One limitation of our solution is being able to display all the Matlab graphs for each submission due to limitations in our storage. In addition, with unlimited resources, we should be able to run more submissions concurrently, saving users time.

6. Another design we considered was only having one behavior hiding module, in an effort to reduce complexity. However, we came to the conclusion that although this would make implementation more simple, it would make maintenance and readability significantly worse.

   We ended up with our current design by ensuring it maintained a balanced of many of the different properties of design we prioritized, such as: Maintenance, complexity, readability and scalability