Module Interface Specification for Software Engineering

Team #12, Lower Earth Orbiters
Diamond Ahuja
Rishi Vaya
Buu Ha
Dhruv Cheemakurti
Umang Rajkarnikar

January 18, 2024

1 Revision History

Date	**Version**	**Notes**
January 18, 2024	1.0	Finished MIS doc

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [. —SS] This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change
FIFO	First In First Out
MCT	Mission Control Terminal [etc. —SS]
[... —SS]	

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

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 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 (-∞, ∞)
Command	Object	Object representing command sequence record from database.

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	N/A
Behaviour-Hiding	Satellite Module
	Schedule Module
	User Module
	Log Module
	Database Module
	Authentication Module
	BackendService Module
Software Decision	SatelliteQueue Module

Table 1: Module Hierarchy

6 MIS of Database Module

6.1 Overview

This module utilizes the date and collectionType, formatted to align with the MongoDB database, to locate and retrieve stored data from the specified location. Additionally, it has the capability to store provided data using the inputs of date, collectionType, and data. Lastly, it can store edited data by utilizing date, collectionType, id, and data as inputs.

6.2 Uses

The database module is used to manage all of the data stored for the satellites, users, commands, logs and scheduling. This involves addition of new data, editing/modifying existing data and deleting values.

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

N/A

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

Name	**Type**	**Description**
DB_USER	String	Login information for MongoDB Cluster
DB_PASSWORD	String	Login information for MongoDB Cluster
DB_URI	String	Connection path to MongoDB instance

6.4.3 Assumptions

MongoDB is an ACID compliant database and ensures persisted data is consistent.

6.4.4 Access Routine Semantics

N/A

6.4.5 Local Functions

Name	Input	Output	Description
insert()	{databaseId: string, values: list of strings}	N/A	Adds new values to the database
update()	{databaseId: string, rowselector: string, values: list of strings}		Updates an existing entry in the specified database
read()	{databaseId: string, selector: string}	{result: list of strings}	Returns values from the specified database based on the selectors and filter provided.
delete()	{databaseId: string, rowselector: string}	N/A	Deletes an entry from the specified database.

7 MIS of Log Module

7.1 Overview

The Log module implements logging functionality which displays the runtime logs of the commands sent to a satellite and its response.

7.2 Uses

The log module is used to view information regarding the commands transmitted to a satellite. This information contains data pertaining to the commands sent during a particular schedule, their run time and the response received.

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	Input	Output	Exceptions
/getLogs	satelliteId	logs	N/A

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

Name	Type	Description
DB_USER	String	Login information for MongoDB Cluster
DB_PASSWORD	String	Login information for MongoDB Cluster
DB_URI	String	Connection path to MongoDB instance

7.4.3 Assumptions

MongoDB is an ACID compliant database and ensures persisted data is consistent.

7.4.4 Access Routine Semantics

N/A

7.4.5 Local Functions

Name	Input	Output	Description
fetchLogs()	(satelliteId: string)	{logs: string}	Returns the logs associated with every command for the specified satellite.

8 MIS of Satellite Module

8.1 Overview

The Satellite module is responsible for all information related to a satellite, including calculating positional data, future overpasses, and generating polar plot data.

8.2 Uses

The satellite module is used to provide relevant information regarding a satellite, including detailed information regarding its position, future overpasses, and relevant polar plot data for each overpass.

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
/getSatelliteInfo	N/A	positionEci: $\mathbb{R}$, velocityEci: $\mathbb{R}$, longitude: $\mathbb{R}$, latitude: $\mathbb{R}$, height: $\mathbb{R}$, azimuth: $\mathbb{R}$, elevation: $\mathbb{R}$	N/A
/getPolarPlotData	startDate: Date, endDate: Date	[azimuth: $\mathbb{R}$, elevation: $\mathbb{R}$]	N/A
/getMaxElevation	startDate: Date, endDate: Date	maxElevation: $\mathbb{R}$	N/A
/getNextPasses	startDate: Date, endDate: Date	type: string, time: string, azimuth: $\mathbb{R}$, elevation: number	N/A

8.4 Semantics

8.4.1 State Variables

Name	**Type**	**Description**
tleLine1	String	TLE information for satellite tracking
tleLine2	String	TLE information for satellite tracking
defaulttleLine1	String	Default TLE information for satellite tracking
defaulttleLine2	String	Default TLE information for satellite tracking
observerGd	{ longitude: Number latitude: Number height: Number }	Observer location data

8.4.2 Environment Variables

Name	Type	Description
SPACE_TRACK_USERNAME	String	Login information for SpaceTrack API
SPACE_TRACK_PASSWORD	String	Login information for SpaceTrack API

8.4.3 Assumptions

Assuming that external APIs (Spacetrack) and libraries (SGP4) are always accurate for measurements.

8.4.4 Access Routine Semantics

N/A

8.4.5 Local Functions

Name	Input	Output	Description
getSatelliteInfo()	(date : Date, tleLine1: string, tleLine2: String)	{ positionEci : number, velocityEci : number, longitude : number, latitude : number, height : number, azimuth : number, elevation : number, rangeSat : number, }	Returns relevant satellite information given a certain date. Used by /getSatelliteInfo, /getPolarPlotData, /getMaxElevation.

9 MIS of Schedule Module

9.1 Overview

The Schedule module is responsible for all logic pertaining to storing, updating, and executing command sequence(s) for a satellite target.

9.2 Uses

The Schedule module is used to schedule execution of command sequences for a satellite system during a specified overpass. This includes reading, adding, and updating command sequences as well as executing scheduled commands.

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	Input	Output	Exceptions
/schedule/addCommand	scheduleId: string, satelliteId: string, userId: string, commandSequence: string	command: Command	Ensure command sequence is included in both user and satellite system's permission list criteria.
/schedule/updateCommand	commandId: string, scheduleId: string, satelliteId: string, userId: string, commandSequence: string	command: Command	Ensure command sequence is included in both user and satellite system's permission list criteria.
/schedule/cancelCommand	commandId: string, userId: string, scheduleId: string	void	Ensure the user has required permissions to cancel the command. Ensure the overpass has not passed.
/getNextPasses	startDate: Date, endDate: Date	type: string, time: string, azimuth: $\mathbb{R}$, elevation: number	N/A

9.4 Semantics

9.4.1 State Variables

Name	Type	Description
overpassQueueMap	[key:string]: SatelliteQueue	A hashmap of key-value pairs. Each key is the ID of a satellite record and corresponds to a value of type SatelliteQueue, which represents the list of command sequences to be executed in the next overpass for a satellite system.

9.4.2 Environment Variables

Name	Type	Description
DB_USER	string	Login information for MongoDB Cluster
DB_PASSWORD	string	Login information for MongoDB Cluster
DB_URI	string	Connection path to MongoDB instance

9.4.3 Assumptions

- MongoDB is an ACID compliant database and ensures persisted data is consistent.

9.4.4 Access Routine Semantics

Name	Transition	Output	Description	Exceptions
executeOverpass()	Indexes state variable overpassQueueMap, then unloads a queue of type SatelliteQueue. Then, loads queue with command sequences scheduled for the satellite's next overpass.	N/A	Executes all command sequences scheduled for the overpass in a FIFO order and records logs of the responses in the database	If a connection cannot be established with the satellite system, the indexed queue shall not be unloaded and therefore, will not be executed.

9.4.5 Local Functions

Name	Input	Output	Description
executeOverpass()	satelliteId: string, overpassId: string	void	Executes all command sequences scheduled for the overpass in a FIFO order and records logs of the responses in the database

10 MIS of SatelliteQueue Module

10.1 Overview

The SatelliteQueue module is responsible for managing future execution of command sequences for all satellite systems.

10.2 Uses

The SatelliteQueue module is used to store execution of command sequences for a satellite system during a specified overpass.

10.3 Syntax

10.3.1 Exported Constants

N/A

10.3.2 Exported Access Programs

N/A

10.4 Semantics

10.4.1 State Variables

Name	**Type**	**Description**
overpassQueueMap	[key:string]: SatelliteQueue	A hashmap of key-value pairs. Each key is the ID of a satellite record and corresponds to a value of type SatelliteQueue, which represents the list of command sequences to be executed in the next overpass for a satellite system.

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

- Node.js is single-threaded and does not permit two requests to modify the state variable, overpassQueueMap to occur simultaneously.

10.4.4 Access Routine Semantics

Name	Transition	Output	Description	Exceptions
push()	Modifies the overpassQueueMap and adds a command sequence to the queue of type SatelliteQueue.	N/A	Adds a command sequence to the queue of type SatelliteQueue.	N/A
pop()	Modifies the overpassQueueMap and removes the top item from the queue of type SatelliteQueue.	N/A	Removes the top command sequence from the queue of type SatelliteQueue.	N/A

10.4.5 Local Functions

N/A

11 MIS of [User Module —SS]

11.1 Overview

The User module is responsible for storing information about users in the MIST application. This includes creating new users and updating information for existing users.

11.2 Uses

The User Module in the MIST application is designed to manage user-related data. Its primary functions include creating new users, updating information for existing users and getting a list of all the current operators.

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 Exported Access Programs

Name	Input	Output	Exceptions
/createUser	Email, role	New user	N/A
/getAllOperators	N/A	Returns all the operators	N/A
/updateOperatorRole/:userId	userId, role	Updates the user's role	N/A

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

Name	Type	Description
DB_USER	String	Login information for MongoDB Cluster
DB_PASSWORD	String	Login information for MongoDB Cluster
DB_URI	String	Connection path to MongoDB instance

11.4.3 Assumptions

MongoDB is an ACID compliant database and ensures persisted data is consistent.

11.4.4 Access Routine Semantics

N/A

11.4.5 Local Functions

N/A

12 MIS of [Authentication Module —SS]

12.1 Overview

This module will employ the user's email and password for authentication purposes and to fetch their complete information, a prerequisite for accessing the satellite and scheduling data.

12.2 Uses

The Authentication Module is a key part of the system, using the user's email and password to confirm their identity. After verification, it allows access to important features like satellite and scheduling data, requiring the user's complete information. This module ensures that only authorized users can interact with the system, enhancing security and controlling access to different functions.

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

N/A

12.4 Semantics

12.4.1 State Variables

Name	Type	Description
email	String	The email address that operator uses to login
password	String	The operator's password associated with their email

12.4.2 Environment Variables

Input Name	Input Type	Description
DB_USER	String	Login information for MongoDB Cluster
DB_PASSWORD	String	Login information for MongoDB Cluster
DB_URI	String	Connection path to MongoDB instance

12.4.3 Assumptions

- MongoDB is an ACID compliant database and ensures persisted data is consistent.
- OAuth0 is the service used for authentication

12.4.4 Access Routine Semantics

N/A

12.4.5 Local Functions

N/A

13 MIS of [Backend Service —SS]

13.1 Overview

This module bridges the front end user interface with all the other aforementioned modules that deal with the functionality of the application. It oversees the initiation of the application's backend processes.

13.2 Uses

The Backend Service module is responsible for ensuring the information from backend modules that deal with user authentication and managing user accounts, and logging of satellite commands, responses and errors is accurately displaying in the UI according to requirements.

13.3 Syntax

13.3.1 Exported Constants

N/A

13.3.2 Exported Access Programs

N/A

13.4 Semantics

13.4.1 State Variables

N/A

13.4.2 Environment Variables

Name	**Type**	**Description**
DB_USER	String	Login information for MongoDB Cluster
DB_PASSWORD	String	Login information for MongoDB Cluster
DB_URI	String	Connection path to MongoDB instance

13.4.3 Assumptions

The only assumption that Backend service module depends on would be for the front-end and back-end to be running properly.

13.4.4 Access Routine Semantics

N/A

13.4.5 Local Functions

N/A

14 Appendix

14.1 User Interface Design - Figma

Figma link for the user interface design

14.2 Component Diagram

Link to Component Diagram

14.3 Reflection

The current limitations of the project stem from difficulties in verification and testing, particularly in connection with an operational ground system for scheduling on actual satellites. If we had unlimited resources, we would create our own fully operational ground server and test satellite architecture to ensure validity. However, the application's requirement is to be hosted on the MIST linux based server making communication challenging as functionalities like pinging are restricted. Having unlimited resources would allow us to establish our server capable of sending commands directly to the satellite. Another limitation of the project is the testing of command scheduling and satellite communication features. Currently, we lack a satellite for direct communication, so we must rely on a mock server for our tests. If unlimited resources were available, launching a test satellite to create a real-like environment for satellite communication testing would be the approach.

Given the set of requirements imposed by external stakeholders, our design solutions were somewhat constrained. However, we actively managed complexity by decomposing our solution into distinct modules and implementing a high cohesion-low coupling approach to enhance the independence of these modules. This design approach aligned with good software engineering practices, particularly in terms of architectural considerations, as a component-based architecture was well-suited to our project requirements. Since the project description provided us with most of the functional and nonfunctional requirements, there was limited room for exploring alternative design ideas for our application. While we did contemplate design decisions related to the application's user interface, the core functionalities remained consistent. Our project's predefined requirements for both functionality and technology stack further limited the scope for exploring diverse implementation strategies at the outset of the project.