

Module Interface Specification for Sandlot

Team 29

Nicholas Fabugais-Inaba

Casra Ghazanfari

Alex Verity

Jung Woo Lee

January 17, 2025

1 Revision History

Date	Version	Notes
January 13, 2025	1.0	TA Feedback
January 15, 2025	1.1	Rev0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/Nicholas-Fabugais-Inaba/Sandlot>

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	3
6	MIS of Account Module	4
6.1	Module	4
6.2	Uses	4
6.3	Syntax	4
6.3.1	Exported Constants	4
6.3.2	Exported Access Programs	4
6.4	Semantics	4
6.4.1	State Variables	4
6.4.2	Environment Variables	4
6.4.3	Assumptions	4
6.4.4	Access Routine Semantics	5
7	MIS of Player Module	6
7.1	Module	6
7.2	Uses	6
7.3	Syntax	6
7.3.1	Exported Constants	6
7.3.2	Exported Access Programs	6
7.4	Semantics	6
7.4.1	State Variables	6
7.4.2	Environment Variables	6
7.4.3	Assumptions	6
7.4.4	Access Routine Semantics	6
8	MIS of Team Module	7
8.1	Module	7
8.2	Uses	7
8.3	Syntax	7
8.3.1	Exported Constants	7
8.3.2	Exported Access Programs	7
8.4	Semantics	7
8.4.1	State Variables	7

8.4.2	Environment Variables	7
8.4.3	Assumptions	7
8.4.4	Access Routine Semantics	7
9	MIS of Commissioner Module	8
9.1	Module	8
9.2	Uses	8
9.3	Syntax	8
9.3.1	Exported Constants	8
9.3.2	Exported Access Programs	8
9.4	Semantics	8
9.4.1	State Variables	8
9.4.2	Environment Variables	8
9.4.3	Assumptions	8
9.4.4	Access Routine Semantics	9
9.4.5	Local Functions	9
10	MIS of Account Structure Module	10
10.1	Module	10
10.2	Uses	10
10.3	Syntax	10
10.4	Semantics	10
10.4.1	State Variables	10
10.4.2	Environment Variables	10
11	MIS of Team Structure Module	11
11.1	Module	11
11.2	Uses	11
11.3	Syntax	11
11.4	Semantics	11
11.4.1	State Variables	11
11.4.2	Environment Variables	11
12	MIS of Schedule Structure Module	12
12.1	Module	12
12.2	Uses	12
12.3	Syntax	12
12.4	Semantics	12
12.4.1	State Variables	12
12.4.2	Environment Variables	12

13 MIS of Standings Structure Module	13
13.1 Module	13
13.2 Uses	13
13.3 Syntax	13
13.4 Semantics	13
13.4.1 State Variables	13
13.4.2 Environment Variables	13
13.4.3 Local Functions	13
14 MIS of Reschedule Module	14
14.1 Module	14
14.2 Uses	14
14.3 Syntax	14
14.3.1 Exported Constants	14
14.3.2 Exported Access Programs	14
14.4 Semantics	14
14.4.1 State Variables	14
14.4.2 Environment Variables	14
14.4.3 Assumptions	14
14.4.4 Access Routine Semantics	14
14.4.5 Local Functions	15
15 MIS of Alerts Module	16
15.1 Module	16
15.2 Uses	16
15.3 Syntax	16
15.3.1 Exported Constants	16
15.3.2 Exported Access Programs	16
15.4 Semantics	16
15.4.1 State Variables	16
15.4.2 Environment Variables	16
15.4.3 Assumptions	16
15.4.4 Access Routine Semantics	16
15.4.5 Local Functions	17
16 MIS of Database Module	18
16.1 Module	18
16.2 Uses	18
16.3 Syntax	18
16.3.1 Exported Constants	18
16.3.2 Exported Access Programs	18
16.4 Semantics	18
16.4.1 State Variables	18

16.4.2	Environment Variables	18
16.4.3	Assumptions	18
16.4.4	Access Routine Semantics	18
16.4.5	Local Functions	19
17	MIS of Season Scheduler Module	20
17.1	Module	20
17.2	Uses	20
17.3	Syntax	20
17.3.1	Exported Constants	20
17.3.2	Exported Access Programs	20
17.4	Semantics	20
17.4.1	State Variables	20
17.4.2	Environment Variables	20
17.4.3	Assumptions	20
17.4.4	Access Routine Semantics	20
17.4.5	Local Functions	21
18	MIS of Web Application Framework Module	22
18.1	Module	22
18.2	Uses	22
18.3	Syntax	22
18.3.1	Exported Constants	22
18.3.2	Exported Access Programs	22
18.4	Semantics	22
18.4.1	State Variables	22
18.4.2	Environment Variables	22
18.4.3	Assumptions	22
18.4.4	Access Routine Semantics	22
18.4.5	Local Functions	23
19	Appendix	24

3 Introduction

The following document details the Module Interface Specifications for the implemented modules in a platform designed to organize a seasonal softball league. It is intended to ease navigation through the platform for design and maintenance purposes.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/Nicholas-Fabugais-Inaba/Sandlot>.

4 Notation

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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Sandlot.

Data Type	Notation	Description
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 $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$
schedule	S	a list of games to be played in a season, see Schedule Structure Module 12
standings	ST	a team's record in a season including wins, losses, ties, forfeits, and point differential, see Standings Structure Module 13
game	G	a time, date, location, score, team1, and team2 that defines a game to be played
player	P	a player on a team, uses the Account Structure Module 10
team	T	a team in the league including a team's id, team's name, team's division, list of players on the team, and the team's standing in the league, see Team Structure Module 11
start date	d_s	a date that represents the start of the season
end date	d_e	a date that represents the end of the season
location	l	an integer representing a field
division	D	an integer indexing a list of teams that play each other in a season
alert	A	a message to be sent to a list of players, see Alerts Module 15

The specification of Sandlot 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, Sandlot 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	
	Account Module
	Player Module
	Team Module
	Commissioner Module
	Account Structure Module
Behaviour-Hiding Module	Team Structure Module
	Schedule Structure Module
	Standings Structure Module
	Reschedule Module
	Alerts Module
	Database Module
Software Decision Module	Season Scheduler Module
	Web Application Framework Module

Table 1: Module Hierarchy

6 MIS of Account Module

6.1 Module

Account

6.2 Uses

account_structure [10](#)

Database [16](#)

web_app_framework [18](#)

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_account	name: <i>string</i> , email: <i>string</i> , pass: <i>string</i> , num: \mathbb{N}	-	-
delete_account	-	-	-
change_name	new_name: <i>string</i>	-	-
change_pass	new_pass: <i>string</i>	-	-
change_email	new_email: <i>string</i>	-	-
change_num	new_num: \mathbb{N}	-	-
account_login	email: <i>string</i> , pass: <i>string</i>	-	-

6.4 Semantics

6.4.1 State Variables

account: P

6.4.2 Environment Variables

None

6.4.3 Assumptions

For every team in the league, each team plays at least twice the amount of games as the number of teams in a division.

6.4.4 Access Routine Semantics

`create_account()`:

- transition: Update the database with the new account information.

`delete_account()`:

- transition: Update the database to remove the account.

`change_name()`:

- transition: Update the database to change the account's name.

`change_pass()`:

- transition: Update the database to change the account's password.

`change_email()`:

- transition: Update the database to change the account's email.

`change_num()`:

- transition: Update the database to change the account's phone number.

`account_login()`:

- transition: Access is given to the account if the email and password match.

7 MIS of Player Module

7.1 Module

Player

7.2 Uses

Account [6](#)

account_structure [10](#)

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
join_team	team: T	-	-
leave_team	-	-	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

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

7.4.4 Access Routine Semantics

join_team():

- transition: Modify the inputted team's data to include the player and update the player's data to include the team.

leave_team():

- transition: Modify the team the player is currently on structure to not include the player and update the player's data to not include the team.

8 MIS of Team Module

8.1 Module

Team

8.2 Uses

Account [6](#)

team_structure [11](#)

Reschedule [14](#)

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
submit_score	team.1:N,team.2:N	-	-

8.4 Semantics

8.4.1 State Variables

team: T

8.4.2 Environment Variables

None

8.4.3 Assumptions

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

8.4.4 Access Routine Semantics

submit_score():

- transition: The team's ST is updated with the new team record.

9 MIS of Commissioner Module

9.1 Module

Commissioner

9.2 Uses

Player [7](#)

Alerts [15](#)

season_scheduler [17](#)

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
overwrite_team_comp	team: T , player_list: $P[]$	-	-
overwrite_schedule	sched_new: S	-	-
overwrite_standings	new_standing: ST	-	-
overwrite_game_score	game: G , new_score: \mathbb{N}	-	-
update_team_contact	team: T , new_info: <i>string</i>	-	-
set_team_division	team: T division: D	-	-
add_division	division: D	-	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

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

9.4.4 Access Routine Semantics

`overwrite_team_comp()`:

- transition: The player list of the inputted team is updated with the inputted player list.

`overwrite_schedule()`:

- transition: The current schedule is updated with the inputted new schedule.

`overwrite_standings()`:

- transition: The current standings are updated with the inputted new standings.

`overwrite_game_score()`:

- transition: The inputted game's score is updated with the inputted new score.

`update_team_contact()`:

- transition: The inputted team's contact information is updated with the inputted new information.

`set_team_division()`:

- transition: The inputted team's division is updated with the inputted division.

`add_division()`:

- transition: The inputted division is added to the league.

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 Account Structure Module

10.1 Module

account_structure

10.2 Uses

N/A

10.3 Syntax

N/A

10.4 Semantics

10.4.1 State Variables

name: string
pass: string
email: string
num: string
team_id: N

10.4.2 Environment Variables

None

11 MIS of Team Structure Module

11.1 Module

team_structure

11.2 Uses

account_structure [10](#)

11.3 Syntax

N/A

11.4 Semantics

11.4.1 State Variables

team_id: \mathbb{N}

team_name: string

div: D

players: $P[]$

standing: ST

11.4.2 Environment Variables

None

12 MIS of Schedule Structure Module

12.1 Module

schedule_structure

12.2 Uses

N/A

12.3 Syntax

N/A

12.4 Semantics

12.4.1 State Variables

games: $G[]$

12.4.2 Environment Variables

None

13 MIS of Standings Structure Module

13.1 Module

standings_structure

13.2 Uses

N/A

13.3 Syntax

N/A

13.4 Semantics

13.4.1 State Variables

wins: \mathbb{N}

losses: \mathbb{N}

ties: \mathbb{N}

forfeits: \mathbb{N}

point_diff: \mathbb{Z}

13.4.2 Environment Variables

None

13.4.3 Local Functions

calc_score: $\mathbb{N}, \mathbb{N}, \mathbb{N}, \mathbb{N} \rightarrow \mathbb{N}$

$\text{calc_score}(\text{wins}, \text{ties}) = 2 * \text{wins} + \text{ties}$

14 MIS of Reschedule Module

14.1 Module

Reschedule

14.2 Uses

Team [8](#)

web_app_framework [18](#)

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
request_reschedule	old_game:G, new_game:G	-	-
accept_reschedule	-	-	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

None

14.4.3 Assumptions

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

14.4.4 Access Routine Semantics

request_reschedule():

- transition: The inputted old game is replaced with the inputted new game that has the new date, time, and location.

accept_reschedule():

- transition: If the the rescheduling is accepted, the old game is removed and the new game is added to the schedule.

14.4.5 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]

15 MIS of Alerts Module

15.1 Module

Alerts

15.2 Uses

web_app_framework [18](#)

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_alert	msg:string, players: $P[]$	alert: A	-
send_alert	alert: A	-	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

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

15.4.4 Access Routine Semantics

create_alert():

- output: A , an alert created according to the user specifications.

send_alert():

- transition: An alert is sent to the players in the inputted player list.

15.4.5 Local Functions

`select_target()`:

- output: $P[]$, a list of players to send the alert to

16 MIS of Database Module

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

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

16.1 Module

Database

16.2 Uses

team_structure 11

schedule_structure 12

standings_structure 13

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

None

16.4.2 Environment Variables

None

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 Season Scheduler Module

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

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

17.1 Module

season_scheduler

17.2 Uses

schedule_structure 12

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
gen_sched	$D[], d_s, d_e, l[]$	S	-

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

None

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

gen_sched():

- output: $out := S_o \in S$ such that:
 - No two games in S should share a date, time, and location:
$$c_1(s : S) = \forall (g_1, g_2 \in s | g_1 \neq g_2 : g_1.date \neq g_2.date \wedge g_1.time \neq g_2.time \wedge g_1.location \neq g_2.location)$$

- No game's parameters should include the same team twice.
 $c_2(s : S) = \forall(g \in s \mid : g.team1 \neq g.team2)$
- No team can play more than one game in a day.
 $c_3(s : S) = \forall(g_1, g_2 \in s \mid g_1 \neq g_2 \wedge g_1.date = g_2.date : g_1.team1 \neq g_2.team1 \wedge g_1.team1 \neq g_2.team2 \wedge g_1.team2 \neq g_2.team1 \wedge g_1.team2 \neq g_2.team2)$
- Teams in division 1 (A) should all play against every team in division 2 (B) once.
 $count(s : S, t_1 : T, t_2 : T) = +(g \in s \mid (g.team1 = t_1 \wedge g.team2 = t_2) \vee (g.team1 = t_2 \wedge g.team2 = t_1) : 1)$
 $c_4(s : S) = \forall(t_1 \in D_1, D_1 \in D \mid : \forall(t_2 \in D_2 \mid : count(s, t_1, t_2) = 1))$
- Teams in divisions 3 and onward should only play against teams in their own division.
 $c_5(s : S) = \forall(D_i \in D, i \in \mathbb{N} \mid i > 2 : \forall(g \in s \mid : g.team1.division = g.team2.division))$
- All games must be within the start and end dates of the season.
 $c_6(s : S) = \forall(g \in s \mid : d_s \leq g.date \leq d_e)$
- All constraints should hold for the output schedule S_o :
 $C = c_1, c_2, c_3, c_4, c_5$
 $\forall(c \in C \mid : c(S_o) = true)$
- The schedule S_o should also be optimized according the soft constants below:
 - * Minimize the maximum number of games played in a week over the season.
 $n_j i$ = number of games played by a team in a week
 m = frequency of n occuring over each week of the season
 $f(t) =$
 - * Minimize the number of games played on a team's scheduled off day according to the team's availability.

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 Web Application Framework Module

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

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

18.1 Module

web_app_framework

18.2 Uses

N/A

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

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

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

None

18.4.3 Assumptions

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

18.4.4 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]

18.4.5 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]

19 Appendix

[Extra information if required —SS]

Appendix — Reflection

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), if 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)

Team Reflection

1. 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?

Some of the design decisions pertained to the season scheduler module, which had come directly from the supervisor as they had clearly defined both hard constraints, that must be satisfied, and soft constraints, that should be optimized for. Additionally, it

was important to have a module that would specifically deal with the scheduling algorithm for the system's extensibility and maintainability purposes. This is because more constraints may be required by the system or clients to be added or removed depending on various circumstances, for example, the league's requirements may change overtime. Furthermore, our team had decided to create structure modules, that were not asked for by the client or a proxy as we believed it would help to simplify the complexity of the system and improve the system's maintainability, defining important parts such as an account or a team.

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

While creating the design doc, no other documents had to be changed to accomodate any design decisions made in the MIS. The requirements identified in the SRS were sufficient enough to guide our team in confidently following through with the design decisions that were made in the MIS.

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

The limitations of our solution are tightly tied to the hardware/software the system is hosted on. For example, with infinite computing power the website would be able to support upwards of a million concurrent users, and have near 100% availability (ie. minimal downtime). However, we do not have the computing power, or money to buy the computing power necessary to provide these benefits. Importantly its not critical that we provide these benefits for our project because they're only important for large scale projects. Our project is much smaller in scale, and as such we'll be able to provide the necessary concurrent user count and availability for a project of our size without the need for unlimited resources.

4. 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)

Other design solutions that were considered were to have a separate account verification module that would deal with the registration of accounts, which would include the signing of waivers, 2-factor authentication, and logging into an account. The benefits of this design would be the fact that it would help to separate the concerns of the account module to solely deal with the editing of an account's details, and the creation or deletion of an account, and the account verification module would have dealt with what had been mentioned above. The tradeoffs would be that it would have added

more complexity to the system, and would have required more time to implement and test each module. The reason why our team had selected the documented design was because we wanted to make sure that each account would have the same functionality, but the type of account (players, teams, and commissioners) would have different functionalities associated with the module. The structure modules were then selected to account for the core parts of the system including the schedule, standings, accounts, and teams.

Nicholas Fabugais-Inaba – Reflection

1. What went well while writing this deliverable?

When writing this deliverable, the things that went well were the collective brainstorming of the modules in the MG that would then be expanded upon in the MIS. Being able to not only visualize how the modules would operate together, but a general idea of how the system would work as a whole, helped a lot in our identification of the different features our system would need to highlight. Another thing that went well was our distribution of tasks as once we had identified all of the modules in the MG, we were able to each tackle a different module in the MIS.

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

Some of the pain points from this deliverable were figuring out what subsections would apply to each module as each module whether it was our structure modules or other modules, had varying purposes. This issue was resolved by referring back to what we had written in our MG for each module's secrets and services as that would help identify any functions that the module would have to accomplish for its dedicated tasks. The same resolution was applied to the structure modules as although they did not necessarily have functions as a part of their module, they still had state variables that would define the data structure it pertained to.

Jung Woo Lee – Reflection

1. What went well while writing this deliverable?

Coming up with the specification math for the scheduling module went very smoothly and working on it helped to solidify some of our ideas on the constraints that were only in our head or in plain writing. Furthermore, writing the MIS and thinking at a lower-level helped us refine our thinking on how the solution would behave or what data was required where. Lastly, Both documents were worked on diligently and efficiently by the team and we were able to complete them to great effect in a shorter amount of time than the previous documents.

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

Definitely a large issue I had was defining the modules and separating out their functions as some could initially be seen to have overlapping responsibilities. Discussion and more brainstorming with the team allowed us to better define the modules and come to our current solution. Another was thinking about the constraints for the specification math where some open questions were discovered and are left unanswered until our next meeting with our supervisor where it will be cleared up.

Casra Ghazanfari – Reflection

1. What went well while writing this deliverable?

Writing the mathematical specification for our scheduling algorithm went well because we already had a portion of its code implemented which allowed us to much more easily translate that existing code into a mathematical specification than if we were to create the specification before starting any implementation of the module. Additionally, because the scheduling algorithm is the most mathematically complex module in our system, having an existing implementation greatly reduced the overall workload of this deliverable making the overall process of writing this deliverable much easier.

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

The biggest pain point during this deliverable was our team's inexperience using LaTeX for formal mathematical specifications. Due to our inexperience the most difficult part of this deliverable wasn't creating the mathematical specifications, but instead properly formatting them. We resolved this issue by spending extra time learning how mathematical specification is done with LaTeX using online resources. After taking the time to learn, writing the deliverable became much easier and quicker.