Module Interface Specification for Sandlot

Team 29
Nicholas Fabugais-Inaba
Casra Ghazanfari
Alex Verity
Jung Woo Lee

April 4, 2025

1 Revision History

Date	Version	Notes
January 13, 2025	1.0	TA Feedback
January 15, 2025	1.1	Rev0
March 10, 2025	1.2	Based on TA feedback: Completed MIS for all modules. Cleaned up other sections.
April 2, 2025	1.3	Based on TA feedback: Ensured consistent names with update or overwrite. References seem to be working appropriately.
April 4, 2025	1.4	Based on TA feedback: Updated document semantics and get_score access routine. Based on TA feedback: Not applicable to formalize objective functions. Removed optimization objective functions for specification math as schedule creation is no longer optimal.
April 4, 2025	1.5	Based on TA feedback: Updated general reflection to include more discussion of SE principles in question 3 and 4.
April 4, 2025	1.6	Based on TA feedback: Updated multi-character variables to be properly formatted (specifically in section 16.4.4).

2 Symbols, Abbreviations and Acronyms

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

Contents

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

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 (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
schedule	S	a list of games to be played in a season, see Schedule Record Module 12
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 Records 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 Records 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 14

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	
Behaviour-Hiding Module	Account Module Player Module Team Module Commissioner Module Account Record Module Team Record Module Schedule Record Module Reschedule Module Alerts Module Database Module
Software Decision Module	Season Scheduler Module

Table 1: Module Hierarchy

6 MIS of Account Module

6.1 Module

Account

6.2 Uses

account_record 10
Database 15

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

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

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(name:*string*, email:*string*, pass:*string*, num:$\mathbb{N}$):

- transition: New account information is inserted into the database.

delete_account():

- transition: The account is removed from the database.

change_name(new_name:*string*):

- transition: The account's name in the database is changed to the inputted name.

change_pass(new_pass:*string*):

- transition: The account's password in the database is changed to the inputted password.

change_email(new_email:*string*):

- transition: The account's email in the database is changed to the inputted email.

change_num(new_num:$\mathbb{N}$):

- transition: The account's phone number in the database is changed to the inputted phone number.

account_login(email:*string*, pass:*string*):

- transition: If the email and password match the saved account, their session is updated to allow them to access all features related to that account type.

7 MIS of Player Module

7.1 Module

Player

7.2 Uses

Account 6
account_record 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

None

7.4.4 Access Routine Semantics

join_team(team:T):

- transition: Insert the player into the team's player list and insert the team into the player's team data.

leave_team():

- transition: Remove the player from the team's player data and remove the team from the player's team data.

8 MIS of Team Module

8.1 Module

Team

8.2 Uses

Account 6
team_record 11
Reschedule 13

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
submit_score	game:G, home_team_score:$\mathbb{N}$, away_team_score:$\mathbb{N}$, forfeit:$\mathbb{N}$	-	-

8.4 Semantics

8.4.1 State Variables

team: T

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

submit_score(game:G, home_team_score:$\mathbb{N}$, away_team_score:$\mathbb{N}$, forfeit:$\mathbb{N}$):

- transition: The game's score data is changed to be the user inputted home team score, away team score and forfeit data.

9 MIS of Commissioner Module

9.1 Module

Commissioner

9.2 Uses

Player 7
Alerts 14
Reschedule 13
season_scheduler 16

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_game_score	game:G, home_team_score:$\mathbb{N}$, away_team_score:$\mathbb{N}$, forfeit:$\mathbb{N}$	-	-
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

None

9.4.4 Access Routine Semantics

overwrite_team_comp(team:T, player_list:P[]):

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

overwrite_schedule(sched_new:S):

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

overwrite_game_score(game:G, home_team_score:$\mathbb{N}$, away_team_score:$\mathbb{N}$, forfeit:$\mathbb{N}$):

- transition: The game's score data is changed to be the commissioner inputted home team score, away team score and forfeit data.

set_team_division(team:T division:D):

- transition: The inputted team's division is change to be the inputted division.

add_division(division:D):

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

9.4.5 Local Functions

None

10 MIS of Account Records Module

10.1 Module

account_record

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: $\mathbb{N}$

10.4.2 Environment Variables

None

11 MIS of Team Records Module

11.1 Module

team_record

11.2 Uses

account_record 10

11.3 Syntax

N/A

11.4 Semantics

11.4.1 State Variables

team_id: $\mathbb{N}$
team_name: string
div: D
players: $P[]$

11.4.2 Environment Variables

None

12 MIS of Schedule Records Module

12.1 Module

schedule_record

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 Reschedule Module

13.1 Module

Reschedule

13.2 Uses

Team 8

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

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

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

request_reschedule(old_game:G, new_game:G):

- transition: A request is sent to the opposing team's captain to reschedule the game, sending the old game's information as well as the inputted new proposed game information. The other captain is given the opportunity to accept or deny the request.

accept_reschedule(old_game:G, new_game:G):

- transition: The old game is removed from the schedule and the new game is added to the schedule.

13.4.5 Local Functions

None

14 MIS of Alerts Module

14.1 Module

Alerts

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

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

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

None

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

create_alert(msg:string, players:$P[]$):

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

send_alert(alert:A):

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

14.4.5 Local Functions

select_target():

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

15 MIS of Database Module

15.1 Module

Database

15.2 Uses

team_record 11
schedule_record 12

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
insert_player	player: P	-	-
insert_team	team: T	-	-
get_player	login_email: string	record: P	-
get_team	login_username: string	record: T	-
get_all_teams	-	records[]: T	-
insert_game	game: G	-	-
get_all_games	-	records[]: G	-
get_team_games	team: T	records[]: G	-
update_game	new_game: G	-	-

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

insert_player(player: P):

- transition: A new player record is added to the database.

insert_team(team: T):

- transition: A new team record is added to the database.

get_player(login_email: string):

- output: The player record with the specified email is retrieved from the database.

get_team(login_username: string):

- output: The team record with the specified username is retrieved from the database.

get_all_teams():

- output: All team records are retrieved from the database.

insert_game(game: G):

- transition: A new game record is added to the database.

get_all_games():

- output: All game records are retrieved from the database.

get_team_games(team: T):

- output: All game records for the specified team are retrieved from the database.

update_game(new_game: G):

- transition: The game record with the specified ID is updated in the database.

15.4.5 Local Functions

None

16 MIS of Season Scheduler Module

16.1 Module

season_scheduler

16.2 Uses

schedule_record 12

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

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

16.4 Semantics

16.4.1 State Variables

None

16.4.2 Environment Variables

None

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

gen_sched($D[], d_s, d_e, l[]$):

- output: $out := S_o \in S$ such that:
  - No two games in S should share a date, time, and location:
    $c_1(s : S) = \forall(g_1, g_2 \in s | g_1 \neq g_2 : g_1.\text{date} \neq g_2.\text{date} \wedge g_1.\text{time} \neq g_2.\text{time} \wedge g_1.\text{location} \neq g_2.\text{location})$
  - No game's parameters should include the same team twice.
    $c_2(s : S) = \forall(g \in s| : g.\text{team1} \neq g.\text{team2})$

- No team can play more than one game in a day.
  $c_3(s : S) = \forall(g_1, g_2 \in s | g_1 \neq g_2 \wedge g_1.\text{date} = g_2.\text{date} : g_1.\text{team1} \neq g_2.\text{team1} \wedge g_1.\text{team1} \neq g_2.\text{team2} \wedge g_1.\text{team2} \neq g_2.\text{team1} \wedge g_1.\text{team2} \neq g_2.\text{team2})$
- Teams in division 1 (A) should all play against every team in division 2 (B) once.
  $\text{count}(s : S, t_1 : T, t_2 : T) = +(g \in s | (g.\text{team1} = t_1 \wedge g.\text{team2} = t_2) \vee (g.\text{team1} = t_2 \wedge g.\text{team2} = t_1) : 1)$
  $c_4(s : S) = \forall(t_1 \in D_1, D_1 \in D | : \forall(t_2 \in D_2 | : \text{count}(s, t_1, t_2) = 1))$
- Teams in divisions 3 and onward should only play against teams in their own division.
  $c_5(s : S) = \forall(D_i \in D, i \in \mathbb{N} | i > 2 : \forall(g \in s | : g.\text{team1.division} = g.\text{team2.division}))$
- All games must be within the start and end dates of the season.
  $c_6(s : S) = \forall(g \in s | : d_s \leq g.\text{date} \leq d_e)$
- All constraints should hold for the output schedule S_o:
  $C = c_1, c_2, c_3, c_4, c_5$
  $\forall(c \in C | : c(S_o))$

16.4.5 Local Functions

None

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), 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)

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 record 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.

   Additionally, given infinite resources (ie. time) we would like to be able to better adhere to the "degrade gently" software principle. Currently, there are parts of the code that have been set-up to throw exceptions when certain conditions are not met and display generic error messages to the user. However, we would like to be able to provide a more detailed description to the user with a set of possible solutions or next steps to help them work through the error quickly. Achieving this would greatly increase the usability of the system and improve the end-user's experience

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 record modules were then selected to account for the core parts of the system including the schedule, accounts, and teams.

We also considered developing our own schedule interface to accommodate for our potential need to display many events in one day at the same time if a commissioner were set many custom fields with overlapping timeslots. However, we decided to use a pre-existing library instead and adhere to the software design principle of not "reinventing the wheel". While developing our own interface would allow us to better display certain edge case scenario schedules, it would greatly increase our development time and complexity. In the end we felt that the interface wasn't at the heart of the projects mission and so opted to use a pre-existing library.

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 record 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 record 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 contraints that were only in our head or in plain writing. Futhermore, 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.

Alexander Verity – Reflection

1. What went well while writing this deliverable?

   The writing of the MIS went well as we were able to quickly identify the modules that we needed to implement and the functions that they would need. The modules were not always easy to identify, but they helped us to break figure out the structure of our solution and will help when developing the final product.

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

   The biggest pain point was writing the specification math for the scheduling module. It was difficult to figure out how to express the constraints in a percise way that fully defines the module. We resolved this by working together to brainstorm the constraints and researching a little more into how specification math works.