# Module Interface Specification for Software Engineering

Team 6, Pitch Perfect
Damien Cheung
Jad Haytaoglu
Derek Li
Temituoyo Ugborogho
Emma Wigglesworth

January 3, 2025

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Jan 3 | 1.0 | Added MIS for TeamT, GameT, PlayerT, Backend |
| Date 2 | 1.1 | Notes |

# 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —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 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 (-$\infty$, $\infty$) |
| natural number | $\mathbb{N}$ | a number without a fractional component in [1, $\infty$) |
| real | $\mathbb{R}$ | any number in (-$\infty$, $\infty$) |

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 | |
| Behaviour-Hiding | Input Parameters<br>Output Format<br>Output Verification<br>Temperature ODEs<br>Energy Equations<br>Control Module<br>Specification Parameters Module |
| Software Decision | Sequence Data Structure<br>ODE Solver<br>Plotting |

Table 1: Module Hierarchy

# 6 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

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

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

## 6.1 Module

[Short name for the module —SS]

## 6.2 Uses

## 6.3 Syntax

### 6.3.1 Exported Constants

### 6.3.2 Exported Access Programs

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

## 6.4 Semantics

### 6.4.1 State Variables

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

### 6.4.2 Environment Variables

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

### 6.4.3 Assumptions

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

### 6.4.4 Access Routine Semantics

[accessProg —SS]():

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

- exception: [if appropriate —SS]

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

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

### 6.4.5 Local Functions

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

# 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.

# 7 MIS of PlayerT Module

## 7.1 Module

**PlayerT**: Abstract Player Module.

## 7.2 Uses

TeamT

- **GameT**: The Player module interacts with the Game module to track player participation in games.
- **TeamT**: The Player module is connected to the Team module, as players are assigned to teams.

## 7.3 Syntax

### 7.3.1 Exported Constants

### 7.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| PlayerT | String, String, String, String, Bool, String | - | - |
| getPlayerId | | String | - |
| getName | | String | - |
| getEmail | | String | - |
| getWaiverStatus | | Bool | - |
| getTeam | | Bool | - |
| setWaiverStatus | Bool | - | - |
| setTeam | String | - | - |

## 7.4 Semantics

### 7.4.1 State Variables

### 7.4.2 Environment Variables

// TODO: some UUID auth

### 7.4.3 Assumptions

### 7.4.4 Access Routine Semantics

PlayerT(id, n, e, p, w, t):

- transition: $playerId, name, email, password, waiverStatus, team := id, n, e, p, w, t$
- output: $out := self$
- exception: None

getPlayerId():

- output: $out := playerId$
- exception: None

getName():

- output: $out := name$
- exception: None

getEmail():

- output: $out := email$
- exception: None

getWaiverStatus():

- output: $out := waiverStatus$
- exception: None

getTeam():

- output: $out := team$
- exception: None

setTeam(t):

- transition: $team := t$
- exception: None

setWaiverStatus(w):

- transition: $waiverStatus := w$
- exception: None

### 7.4.5 Local Functions

# 8 MIS of GameT Module

## 8.1 Module

**GameT**: Abstract Game Type

## 8.2 Uses

TeamT

## 8.3 Syntax

### 8.3.1 Exported Constants

### 8.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| GameT | String, String, Date, String, String, Integer, Integer, String | - | - |
| getGameId | | String | - |
| getTeamsInGame | | TeamT[] | GameNotFound |
| getGameDetails | | Object | GameNotFound |
| getStatus | | String | GameNotFound |
| getField | | String | GameNotFound |
| setStatus | String | - | InvalidStatus |
| setField | String | - | InvalidField |
| setScore | Integer, Integer | - | InvalidScore |

## 8.4 Semantics

### 8.4.1 State Variables

### 8.4.2 Environment Variables

None

### 8.4.3 Assumptions

- A game must have two teams assigned to it.
- A game must have a field assigned to it.
- A game's score can only be updated after the game is completed.

- The game status must be updated to reflect its current state (e.g., scheduled, completed).

#### 8.4.4 Access Routine Semantics

GameT(id, t1, t2, d, t, s1, s2, f):

- transition: $gameId, team1Id, team2Id, gameDate, gameTime, scoreTeam1, scoreTeam2, field := id, t1, t2, d, t, s1, s2, f$
- output: $out := self$
- exception: None

getGameId():

- output: $out := gameId$
- exception: None

getGameDetails():

- output: $out :=$ Object containing game details: $gameId, team1Id, team2Id, gameDate, gameTime,$
- exception: Game not found if the game ID does not exist.

getTeamsInGame():

- output: $out :=$ Array of Teams participating in the game
- exception: Game not found if the game ID does not exist.

getStatus():

- output: $out := status$
- exception: Game not found if the game ID does not exist.

getField():

- output: $out := field$
- exception: Game not found if the game ID does not exist.

setStatus(status):

- transition: $status := status$
- exception: Invalid status if the provided status is not valid.

setField(field):

- transition: $field := field$
- exception: Invalid field if the provided field is not valid.

setScore(score1, score2):

- transition: $scoreTeam1 := score1, scoreTeam2 := score2$
- exception: Invalid score if the provided scores are not valid integers.

#### 8.4.5 Local Functions

- **validateGameDetails()**: A function to validate the input details when creating a new game.

# 9 MIS of TeamT Module

## 9.1 Module

**TeamT Module**: Abstract Team Module

## 9.2 Uses

PlayerT, GameT

## 9.3 Syntax

### 9.3.1 Exported Constants

### 9.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| TeamT | String, String, String, PlayerT[], PlayerT | - | - |
| getTeamId | | String | - |
| getTeamName | | String | - |
| getDivision | | String | - |
| getRoster | | PlayerT[] | - |
| getCaptain | | PlayerT | - |
| addPlayer | PlayerT | - | InvalidPlayer |
| removePlayer | PlayerT | - | PlayerNotFound |

## 9.4 Semantics

### 9.4.1 State Variables

### 9.4.2 Environment Variables

None

### 9.4.3 Assumptions

- Each team must have a unique team ID.
- A team can belong to one division at a time.
- The team roster must be an array or list of players (with unique player identifiers).

### 9.4.4 Access Routine Semantics

TeamT(id, n, d, r, c):

- transition: $teamId, teamName, division, roster, captain := id, n, d, r, c$
- output: $out := self$
- exception: None

getTeamId():

- output: $out := teamId$
- exception: None

getTeamName():

- output: $out := teamName$
- exception: None

getDivision():

- output: $out := division$
- exception: None

getRoster():

- output: $out := roster$
- exception: None

getCaptain():

- output: $out := captain$
- exception: None

addPlayer(player):

- transition: $roster := roster + player$
- exception: Invalid player if the player is invalid or already in the roster.

removePlayer(player):

- transition: $roster := roster - player$
- exception: Player not found if the player is not in the roster.

#### 9.4.5 Local Functions

None

# 10 MIS of Backend Module

## 10.1 Module

Backend/Database

## 10.2 Uses

PlayerT, GameT, TeamT

## 10.3 Syntax

### 10.3.1 Exported Constants

### 10.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| createPlayer | String, String, String, String, Bool, String | - | PlayerCreationError |
| getPlayer | String | PlayerT | PlayerNotFound |
| updatePlayer | String, String, String, String, Bool, String | - | PlayerNotFound |
| deletePlayer | String | - | PlayerNotFound |
| createTeam | String, String, String, PlayerT[], PlayerT | - | TeamCreationError |
| getTeam | String | TeamT | TeamNotFound |
| updateTeam | String, String, String, PlayerT[], PlayerT | - | TeamCreationError |
| deleteTeam | String | - | TeamNotFound |
| createGame | String, String, Date, String, String, Integer, Integer, String | - | GameCreationError |
| getGame | String | GameT | GameNotFound |
| updateGame | String, String, Date, String, String, Integer, Integer, String | - | GameCreationError |
| deleteGame | String | - | GameNotFound |
| getAllPlayersForTeam | String | PlayerT[] | TeamNotFound |
| getAllGamesForTeam | String | GameT[] | TeamNotFound |

## 10.4 Semantics

### 10.4.1 State Variables

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

### 10.4.2 Environment Variables

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

### 10.4.3 Assumptions

- The backend is connected to a database

- Each database operation (CRUD) will be encapsulated in a backend method to ensure separation of concerns
- Data consistency and integrity are maintained by the backend during each operation.

#### 10.4.4 Access Routine Semantics

createPlayer(name, email, password, waiverStatus, team):

- transition: $playerId, name, email, password, waiverStatus, team := name, email, password, waiver$
- exception: "Player Creation Error" if player cannot be created

getPlayer(playerId):

- output: $out := player$ (retrieves player object based on playerId)
- exception: "Player Not Found" if player does not exist

updatePlayer(playerId, name, email, password, waiverStatus, team):

- transition: $name, email, password, waiverStatus, team := name, email, password, waiverStatus, te$ (updates player's details)
- exception: "Player Not Found" if player does not exist

deletePlayer(playerId):

- transition: $player := null$ (deletes the player from the database)
- exception: "Player Not Found" if player does not exist

createTeam(teamName, division, captain, roster):

- transition: $teamId, teamName, division, captain, roster := teamName, division, captain, roster$
- exception: "Team Creation Error" if team cannot be created

getTeam(teamId):

- output: $out := team$ (retrieves team object based on teamId)
- exception: "Team Not Found" if team does not exist

updateTeam(teamId, teamName, division, roster):

- transition: $teamName, division, roster := teamName, division, roster$ (updates team's details)
- exception: "Team Not Found" if team does not exist

deleteTeam(teamId):

- transition: $team := null$ (deletes the team from the database)
- exception: "Team Not Found" if team does not exist

createGame(teams, date, time, field, score):

- transition: $gameId, teams, date, time, field, score := teams, date, time, field, score$ (creates a new game)
- exception: "Game Creation Error" if game cannot be created (e.g., scheduling conflict)

getGame(gameId):

- output: $out := game$ (retrieves game object based on gameId)
- exception: "Game Not Found" if game does not exist

updateGame(gameId, updates):

- transition: $gameId, updates := gameId, updates$ (updates the game's details)
- exception: "Game Not Found" if game does not exist

deleteGame(gameId):

- transition: $game := null$ (deletes the game from the database)
- exception: "Game Not Found" if game does not exist

getAllPlayersForTeam(teamId):

- output: $out := players$ (retrieves all players associated with the team)
- exception: "Team Not Found" if team does not exist

getAllGamesForTeam(teamId):

- output: $out := games$ (retrieves all games associated with the team)
- exception: "Team Not Found" if team does not exist

#### 10.4.5 Local Functions

# 11 Appendix

[Extra information if required —SS]

# Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

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

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

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

1. What went well while writing this deliverable?

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

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

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

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

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