

Module Interface Specification for SFWRENG 4G06

Team #12, Team 12.0
Kanugalawattage, Anton
Subedi, Dipendra
Rizkalla, Youssef
Leung, Tamas
Zhao, Zhiming

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation [here](#).

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	1
6	ClientT Module	3
6.1	Template Module	3
6.2	Uses	3
6.3	Syntax	3
6.3.1	Exported Types	3
6.3.2	Exported Access Programs	3
6.4	Semantics	3
6.4.1	State Variables	3
6.4.2	Environment Variables	4
6.4.3	Assumptions	4
6.4.4	Access Routine Semantics	4
6.4.5	Local Functions	5
7	GameT Module	6
7.1	Template Module	6
7.2	Uses	6
7.3	Syntax	6
7.3.1	Exported Types	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	7
7.4.5	Local Functions	7
8	MatchT Module	8
8.1	Template Module	8
8.2	Uses	8
8.3	Syntax	8
8.3.1	Exported Types	8
8.3.2	Exported Access Programs	8

8.4	Semantics	8
8.4.1	State Variables	8
8.4.2	Environment Variables	8
8.4.3	Assumptions	8
8.4.4	Access Routine Semantics	9
8.4.5	Local Functions	9
9	UserT Module	10
9.1	Template Module	10
9.2	Uses	10
9.3	Syntax	10
9.3.1	Exported Types	10
9.3.2	Exported Access Programs	10
9.4	Semantics	10
9.4.1	State Variables	10
9.4.2	Environment Variables	10
9.4.3	Assumptions	11
9.4.4	Access Routine Semantics	11
9.4.5	Local Functions	11
10	UserStatsT Module	12
10.1	Template Module	12
10.2	Uses	12
10.3	Syntax	12
10.3.1	Exported Types	12
10.3.2	Exported Access Programs	12
10.4	Semantics	12
10.4.1	State Variables	12
10.4.2	Environment Variables	12
10.4.3	Assumptions	12
10.4.4	Access Routine Semantics	12
10.4.5	Local Functions	13
11	ProblemT Module	14
11.1	Template Module	14
11.2	Uses	14
11.3	Syntax	14
11.3.1	Exported Types	14
11.3.2	Exported Access Programs	14
11.4	Semantics	14
11.4.1	State Variables	14
11.4.2	Environment Variables	15
11.4.3	Assumptions	15

11.4.4	Access Routine Semantics	15
11.4.5	Local Functions	16
12	Difficulty Module	17
12.1	Module	17
12.2	Uses	17
12.3	Syntax	17
12.3.1	Exported Constants	17
12.3.2	Exported Access Programs	17
12.4	Semantics	17
13	Language Module	18
13.1	Module	18
13.2	Uses	18
13.3	Syntax	18
13.3.1	Exported Constants	18
13.3.2	Exported Access Programs	18
13.4	Semantics	18
14	TestCaseT Module	19
14.1	Template Module	19
14.2	Uses	19
14.3	Syntax	19
14.3.1	Exported Types	19
14.3.2	Exported Access Programs	19
14.4	Semantics	19
14.4.1	State Variables	19
14.4.2	Environment Variables	19
14.4.3	Assumptions	19
14.4.4	Access Routine Semantics	20
14.4.5	Local Functions	20
15	SubmissionT Module	21
15.1	Template Module	21
15.2	Uses	21
15.3	Syntax	21
15.3.1	Exported Types	21
15.3.2	Exported Access Programs	21
15.4	Semantics	21
15.4.1	State Variables	21
15.4.2	Environment Variables	21
15.4.3	Assumptions	21
15.4.4	Access Routine Semantics	21

15.4.5	Local Functions	22
16	JudgeResultT Module	23
16.1	Template Module	23
16.2	Uses	23
16.3	Syntax	23
16.3.1	Exported Types	23
16.3.2	Exported Access Programs	23
16.4	Semantics	23
16.4.1	State Variables	23
16.4.2	Environment Variables	23
16.4.3	Assumptions	23
16.4.4	Access Routine Semantics	24
16.4.5	Local Functions	24
17	JudgeVerdict Module	25
17.1	Module	25
17.2	Uses	25
17.3	Syntax	25
17.3.1	Exported Constants	25
17.3.2	Exported Types	25
17.3.3	Exported Access Programs	25
17.4	Semantics	25
17.4.1	State Variables	25
17.4.2	Environment Variables	25
17.4.3	Assumptions	25
17.4.4	Access Routine Semantics	25
17.4.5	Local Functions	25
18	TestCaseVerdictT Module	26
18.1	Template Module	26
18.2	Uses	26
18.3	Syntax	26
18.3.1	Exported Types	26
18.3.2	Exported Access Programs	26
18.4	Semantics	26
18.4.1	State Variables	26
18.4.2	Environment Variables	26
18.4.3	Assumptions	26
18.4.4	Access Routine Semantics	27
18.4.5	Local Functions	27

19 Home Page Module	28
19.1 Module	28
19.2 Uses	28
19.3 Syntax	28
19.3.1 Exported Constants	28
19.3.2 Exported Access Programs	28
19.4 Semantics	28
19.4.1 State Variables	28
19.4.2 Environment Variables	28
19.4.3 Assumptions	28
19.4.4 Access Routine Semantics	28
19.4.5 Local Functions	29
20 Profile Page Module	30
20.1 Module	30
20.2 Uses	30
20.3 Syntax	30
20.3.1 Exported Constants	30
20.3.2 Exported Access Programs	30
20.4 Semantics	30
20.4.1 State Variables	30
20.4.2 Environment Variables	30
20.4.3 Assumptions	30
20.4.4 Access Routine Semantics	30
20.4.5 Local Functions	31
21 Leaderboard Page Module	32
21.1 Module	32
21.2 Uses	32
21.3 Syntax	32
21.3.1 Exported Constants	32
21.3.2 Exported Access Programs	32
21.4 Semantics	32
21.4.1 State Variables	32
21.4.2 Environment Variables	32
21.4.3 Assumptions	32
21.4.4 Access Routine Semantics	32
21.4.5 Local Functions	33
22 Lobby Page Module	34
22.1 Module	34
22.2 Uses	34
22.3 Syntax	34

22.3.1	Exported Constants	34
22.3.2	Exported Access Programs	34
22.4	Semantics	34
22.4.1	State Variables	34
22.4.2	Environment Variables	34
22.4.3	Assumptions	34
22.4.4	Access Routine Semantics	35
22.4.5	Local Functions	35
23	Game Page Module	36
23.1	Module	36
23.2	Uses	36
23.3	Syntax	36
23.3.1	Exported Constants	36
23.3.2	Exported Access Programs	36
23.4	Semantics	36
23.4.1	State Variables	36
23.4.2	Environment Variables	36
23.4.3	Assumptions	36
23.4.4	Access Routine Semantics	37
23.4.5	Local Functions	37
24	Login Page Module	38
24.1	Module	38
24.2	Uses	38
24.3	Syntax	38
24.3.1	Exported Constants	38
24.3.2	Exported Access Programs	38
24.4	Semantics	38
24.4.1	State Variables	38
24.4.2	Environment Variables	38
24.4.3	Assumptions	38
24.4.4	Access Routine Semantics	38
24.4.5	Local Functions	39
24.4.6	Considerations	39
25	SubmissionService Module	40
25.1	Module	40
25.2	Uses	40
25.3	Syntax	40
25.3.1	Exported Constants	40
25.3.2	Exported Access Programs	40
25.4	Semantics	40

25.4.1	State Variables	40
25.4.2	Environment Variables	40
25.4.3	Assumptions	40
25.4.4	Access Routine Semantics	40
25.4.5	Local Functions	41
26	ProblemsService Module	42
26.1	Module	42
26.2	Uses	42
26.3	Syntax	42
26.3.1	Exported Constants	42
26.3.2	Exported Access Programs	42
26.4	Semantics	42
26.4.1	State Variables	42
26.4.2	Environment Variables	42
26.4.3	Assumptions	42
26.4.4	Access Routine Semantics	42
26.4.5	Local Functions	42
27	UserService Module	43
27.1	Module	43
27.2	Uses	43
27.3	Syntax	43
27.3.1	Exported Constants	43
27.3.2	Exported Access Programs	43
27.4	Semantics	43
27.4.1	State Variables	43
27.4.2	Environment Variables	43
27.4.3	Assumptions	43
27.4.4	Access Routine Semantics	43
28	AuthService Module	44
28.1	Module	44
28.2	Uses	44
28.3	Syntax	44
28.3.1	Exported Constants	44
28.3.2	Exported Access Programs	44
28.4	Semantics	44
28.4.1	State Variables	44
28.4.2	Environment Variables	44
28.4.3	Assumptions	44
28.4.4	Access Routine Semantics	44
28.4.5	Local Functions	45

29 LobbyService Module	46
29.1 Module	46
29.2 Uses	46
29.3 Syntax	46
29.3.1 Exported Access Programs	46
29.4 Semantics	46
29.4.1 State Variables	46
29.4.2 Environment Variables	46
29.4.3 Assumptions	46
29.4.4 Access Routine Semantics	46
30 WebSocketService Module	48
30.1 Module	48
30.2 Uses/Imports	48
30.3 Syntax	48
30.3.1 Exported Constants	48
30.3.2 Exported Access Programs	48
30.4 Semantics	48
30.4.1 State Variables	48
30.4.2 Environment Variables	48
30.4.3 Assumptions	48
30.4.4 Access Routine Semantics	49
30.4.5 Local Functions	49
31 GameHandler Module	50
31.1 Module	50
31.2 Uses	50
31.3 Syntax	50
31.3.1 Exported Constants	50
31.3.2 Exported Access Programs	50
31.3.3 Exported Access Programs	50
31.3.4 Environment Variables	50
31.3.5 Assumptions	51
31.4 Semantics	51
31.4.1 State Variables	51
31.4.2 Access Routine Semantics	51
31.4.3 Local Functions	52
32 Judge Module	53
32.1 Module	53
32.2 Uses	53
32.3 Syntax	53
32.3.1 Exported Constants	53

32.3.2	Exported Access Programs	53
32.4	Semantics	53
32.4.1	State Variables	53
32.4.2	Assumptions	53
32.4.3	Access Routine Semantics	53
32.4.4	Local Functions	54
33	CodeRunner Module	55
33.1	Module	55
33.2	Uses	55
33.3	Syntax	55
33.3.1	Exported Constants	55
33.3.2	Exported Types	55
33.3.3	Exported Access Programs	55
33.4	Semantics	55
33.4.1	State Variables	55
33.4.2	Environment Variables	55
33.4.3	Assumptions	56
33.4.4	Access Routine Semantics	56
33.4.5	Local Functions	56
34	Auth Module	57
34.1	Module	57
34.2	Uses	57
34.3	Syntax	57
34.3.1	Exported Constants	57
34.3.2	Exported Access Programs	57
34.4	Semantics	57
34.4.1	State Variables	57
34.4.2	Environment Variables	57
34.4.3	Assumptions	57
34.4.4	Access Routine Semantics	57
34.4.5	Local Functions	57
35	Problems Module	58
35.1	Module	58
35.2	Uses	58
35.3	Syntax	58
35.3.1	Exported Constants	58
35.3.2	Exported Access Programs	58
35.4	Semantics	58
35.4.1	State Variables	58
35.4.2	Environment Variables	58

35.4.3	Assumptions	58
35.4.4	Access Routine Semantics	58
35.4.5	Local Functions	58
36	User Module	59
36.1	Template Module	59
36.2	Uses	59
36.3	Syntax	59
36.3.1	Exported Constants	59
36.3.2	Exported Access Programs	59
36.4	Semantics	59
36.4.1	State Variables	59
36.4.2	Environment Variables	59
36.4.3	Assumptions	59
36.4.4	Access Routine Semantics	60
36.4.5	Local Functions	60
37	Database Module	61
37.1	Module	61
37.2	Uses	61
37.3	Syntax	61
37.3.1	Exported Constants	61
37.3.2	Exported Access Programs	61
37.4	Semantics	61
37.4.1	State Variables	61
37.4.2	Environment Variables	62
37.4.3	Assumptions	62
37.4.4	Access Routine Semantics	62
37.4.5	Local Functions	63
38	Router Module	64
38.1	Module	64
38.2	Uses	64
38.3	Syntax	64
38.3.1	Exported Constants	64
38.3.2	Exported Access Programs	64
38.4	Semantics	64
38.4.1	State Variables	64
38.4.2	Environment Variables	64
38.4.3	Assumptions	64
38.4.4	Access Routine Semantics	64
38.4.5	Local Functions	65

3 Introduction

The following document details the Module Interface Specifications for CodeChamp which is a collaborative and accessible environment is intended to gamify the learning experience.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/Tamas-Leung/CodeChamp>.

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

The following table summarizes the primitive data types used by SFWRENG 4G06.

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 SFWRENG 4G06 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, SFWRENG 4G06 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 Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 ClientT Module

6.1 Template Module

ClientT

6.2 Uses

[LobbyService Module](#)

[Router Module](#)

6.3 Syntax

6.3.1 Exported Types

ClientT = ?

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
ClientT	String, String, String, String, String, N	ClientT	
getID		String	
getEmail		String	
getName		String	
getPicture		String	
getGame		String	
getCompleteRound		N	
getLobbyService		LobbyService	
getRouterModule		RouterModule	

6.4 Semantics

6.4.1 State Variables

id: String

email: String

name: String

picture: String

game: String

lastCompletedRound: N

ls: LobbyService = LobbyService()

rm: RouterModule = RouterModule()

6.4.2 Environment Variables

None

6.4.3 Assumptions

The constructor ClientT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

6.4.4 Access Routine Semantics

ClientT(*i, e, n, p, g, lcr*):

- transition: *id, email, name, picture, game, lastCompletedRound, ls, rm* := *i, e, n, p, g, lcr*
- output: *out* := *self*
- exception: None

getID():

- output: *out* := *id*
- exception: None

getEmail():

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

getName():

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

getPicture():

- output: *out* := *picture*
- exception: None

getGame():

- output: *out* := *game*
- exception: None

getLastCompletedRound():

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

getLobbyService():

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

getRouterModule():

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

6.4.5 Local Functions

None

7 GameT Module

7.1 Template Module

GameT

7.2 Uses

[ClientT Module](#)

7.3 Syntax

7.3.1 Exported Types

GameT = ?

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
GameT	seq of ClientT, String, \mathbb{N}	GameT	
getClients		seq of ClientT	
getID		String	
getRound		\mathbb{N}	

7.4 Semantics

7.4.1 State Variables

id: String

clients: seq of ClientT

round: \mathbb{N}

7.4.2 Environment Variables

None

7.4.3 Assumptions

The constructor GameT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

7.4.4 Access Routine Semantics

GameT(*c*, *id*, *r*):

- transition: $userID, clients, id, round := c, id, r$
- output: $out := self$
- exception: None

getClients():

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

getID():

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

getRound():

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

7.4.5 Local Functions

None

8 MatchT Module

8.1 Template Module

MatchT

8.2 Uses

[Problems Module](#)

8.3 Syntax

8.3.1 Exported Types

MatchT = ?

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
MatchT getUserID getProblems getRanking getDate	String, seq of ProblemT, \mathbb{Z} , String	MatchT String seq of ProblemT \mathbb{Z} String	

8.4 Semantics

8.4.1 State Variables

userID: String
problems: seq of ProblemT
ranking: \mathbb{Z}
date: String

8.4.2 Environment Variables

None

8.4.3 Assumptions

The constructor MatchT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

8.4.4 Access Routine Semantics

MatchT(uid, ps, r, d):

- transition: $userID, problems, ranking, date := uid, ps, r, d$
- output: $out := self$
- exception: None

getUserID():

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

getProblems():

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

getRanking():

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

getDate():

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

8.4.5 Local Functions

None

9 UserT Module

9.1 Template Module

UserT

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Types

UserT = ?

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
UserT	String, String, String, String, String	UserT	
getUserID		String	
getUsername		String	
getEmail		String	
getProfilePicture		String	

9.4 Semantics

9.4.1 State Variables

userID: String
username: String
password: String
email: String
profilePicture: String

9.4.2 Environment Variables

None

9.4.3 Assumptions

The constructor UserT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

The password for the user is not sent or stored by any of the modules and is instead handled by an external service.

9.4.4 Access Routine Semantics

UserT(id, u, p, e, pic):

- transition: $userID, username, password, email, profilePicture := id, u, p, e, pic$
- output: $out := self$
- exception: None

getUserID():

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

getUsername():

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

getEmail():

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

getProfilePicture():

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

9.4.5 Local Functions

None

10 UserStatsT Module

10.1 Template Module

UserStatsT

10.2 Uses

10.3 Syntax

10.3.1 Exported Types

UserStatsT = ?

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
UserStatsT	String, \mathbb{Z} , \mathbb{Z}	UserStatsT	IllegalArgumentException
getUserID		String	
getWins		\mathbb{Z}	
getLosses		\mathbb{Z}	
getWinRate		\mathbb{R}	

10.4 Semantics

10.4.1 State Variables

userID: String

wins: \mathbb{Z}

losses: \mathbb{Z}

10.4.2 Environment Variables

None

10.4.3 Assumptions

The constructor UserStatsT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

10.4.4 Access Routine Semantics

UserStatsT(id, w, l):

- transition: $userID, wins, losses := id, w, l$

- output: $out := self$
- exception: $(w < 0 \vee l < 0) \implies IllegalArgumentException$

getUserID():

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

getWins():

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

getLosses():

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

getWinRate():

- output: $out := (wins/losses) * 100$
- exception: None

10.4.5 Local Functions

None

11 ProblemT Module

11.1 Template Module

ProblemT

11.2 Uses

[Difficulty Module](#)

[TestCaseT Module](#)

11.3 Syntax

11.3.1 Exported Types

ProblemT = ?

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
ProblemT	String, String, String, \mathbb{R} , \mathbb{N} , seq of TestCaseT, Difficulty, seq of String	ProblemT	
getID		String	
getName		String	
getDescription		String	
getTimeLimit		\mathbb{R}	
getMemoryLimit		\mathbb{N}	
getTestCases		seq of Test- CaseT	
getDifficulty		Difficulty	
getProblemType		seq of String	

11.4 Semantics

11.4.1 State Variables

id: string
name: string
description: string
time_limit: \mathbb{R}
memory_limit: \mathbb{N}

test_cases: seq of TestCaseT
difficulty: Difficulty
problem_type: seq of string

11.4.2 Environment Variables

None

11.4.3 Assumptions

The constructor ProblemT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

11.4.4 Access Routine Semantics

ProblemT(id, n, d, tl, ml, tc, d, pt):

- transition: $id, name, description, time_limit, memory_limit, test_cases, difficulty, problem_type := id, n, d, tl, ml, tc, d, pt$
- output: $out := self$
- exception: None

getID():

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

getName():

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

getDescription():

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

getTimeLimit():

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

getMemoryLimit():

- output: *out := memory_limit*
- exception: None

getTestCases():

- output: *out := test_cases*
- exception: None

getDifficulty():

- output: *out := difficulty*
- exception: None

getProblemType():

- output: *out := problem_type*
- exception: None

11.4.5 Local Functions

None

12 Difficulty Module

12.1 Module

Difficulty

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

Difficulty = { Easy, Medium, Hard }

12.3.2 Exported Access Programs

None

12.4 Semantics

None

13 Language Module

13.1 Module

Language

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Constants

Language = { JavaScript, Python }

13.3.2 Exported Access Programs

None

13.4 Semantics

None

14 TestCaseT Module

14.1 Template Module

TestCaseT

14.2 Uses

None

14.3 Syntax

14.3.1 Exported Types

TestCaseT = ?

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
TestCaseT	String, String, \mathbb{B} , \mathbb{R} , \mathbb{N}	TestCaseT	
getInput		String	
getOutput		String	
getHidden		\mathbb{B}	
getTimeLimit		\mathbb{R}	
getMemoryLimit		\mathbb{N}	

14.4 Semantics

14.4.1 State Variables

input: String
output: String
hidden: \mathbb{B}
timeLimit: \mathbb{R}
memoryLimit: \mathbb{N}

14.4.2 Environment Variables

None

14.4.3 Assumptions

The constructor TestCaseT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

14.4.4 Access Routine Semantics

TestCaseT(i, o, h, ml, tl):

- transition: *input, output, hidden, timeLimit, memoryLimit := i, o, h, ml, tl*
- output: *out := self*
- exception: None

getInput():

- output: *out := input*
- exception: None

getOutput():

- output: *out := output*
- exception: None

getHidden():

- output: *out := hidden*
- exception: None

getTimeLimit():

- output: *out := time_limit*
- exception: None

getMemoryLimit():

- output: *out := memory_limit*
- exception: None

14.4.5 Local Functions

None

15 SubmissionT Module

15.1 Template Module

SubmissionT

15.2 Uses

[Language Module](#)

15.3 Syntax

15.3.1 Exported Types

SubmissionT = ?

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
SubmissionT	String, Lan-	SubmissionT	
getCode	guage	String	
getLanguage		Language	

15.4 Semantics

15.4.1 State Variables

code: string
language: Language

15.4.2 Environment Variables

None

15.4.3 Assumptions

The constructor SubmissionT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

15.4.4 Access Routine Semantics

SubmissionT(*c*, *l*):

- transition: *code, language* := *c, l*

- output: *out* := *self*
- exception: None

getCode():

- output: *out* := *code*
- exception: None

getLanguage():

- output: *out* := *language*
- exception: None

15.4.5 Local Functions

None

16 JudgeResultT Module

16.1 Template Module

JudgeResultT

16.2 Uses

[JudgeVerdict Module](#)

[TestCaseVerdictT Module](#)

16.3 Syntax

16.3.1 Exported Types

JudgeResultT = ?

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
JudgeResultT getVerdict getTestCaseVerdicts	set of TestCaseVerdictT	JudgeResultT JudgeVerdict set of TestCa- seVerdictT	IllegalArgumentException

16.4 Semantics

16.4.1 State Variables

verdict: JudgeVerdict

testCaseVerdicts: set of TestCaseVerdictT

16.4.2 Environment Variables

None

16.4.3 Assumptions

The constructor JudgeResultT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

16.4.4 Access Routine Semantics

JudgeResultT(testVerdicts):

- transition: $verdict := ((\forall v : TestCaseVerdictT \mid v \in testCaseVerdicts : v.getVerdict() = Correct) \implies Correct) \vee (tcv : JudgeVerdict \text{ such that } (\exists v : TestCaseVerdictT \mid tcv \in testCaseVerdicts : tcv.getVerdict() \neq Correct \wedge v = tcv.getVerdict()))$
- output: $out := self$
- exception: $exc := (testVerdicts = \{\}) \implies IllegalArgumentException$

getVerdict():

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

getTestCaseVerdicts():

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

16.4.5 Local Functions

None

17 JudgeVerdict Module

17.1 Module

JudgeVerdict

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Types

JudgeVerdict = { Correct, Wrong, TimeLimitExceeded, MemoryLimitExceeded, RuntimeError, CompileError }

17.3.3 Exported Access Programs

None

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

None

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

None

17.4.5 Local Functions

None

18 TestCaseVerdictT Module

18.1 Template Module

TestCaseVerdictT

18.2 Uses

[TestCaseT Module](#)

18.3 Syntax

18.3.1 Exported Types

TestCaseVerdictT = ?

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
TestCaseVerdictT getVerdict getUserOutput getTestCase	JudgeVerdict, String, Test- CaseT	TestCaseVerdictT JudgeVerdict String TestCaseT	

18.4 Semantics

18.4.1 State Variables

verdict: JudgeVerdict

userOutput: string

testCase: TestCaseT

18.4.2 Environment Variables

None

18.4.3 Assumptions

The constructor TestCaseVerdictT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

18.4.4 Access Routine Semantics

getVerdict():

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

getUserOutput():

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

getTestCase():

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

18.4.5 Local Functions

None

19 Home Page Module

19.1 Module

HomePage

19.2 Uses

[Router Module](#)

[WebSocketService Module](#)

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event	-	-

19.4 Semantics

19.4.1 State Variables

code: String

19.4.2 Environment Variables

Screen: A window displayed on user's screen

Browser: The user's browser

19.4.3 Assumptions

init() will be ran everytime the browser url is set to "/"

handleEvent() will be called by the browser for input events

19.4.4 Access Routine Semantics

init():

- transition: $code := ""$
 $Screen :=$ Displays the home page.

handleEvent(event: Browser.Event):

	event	function
• transition:	On-click onto create game button	CreateGameButton()
	On-click onto join game button	JoinGameButton()
	On-click onto find game button	FindGameButton()
	On-click onto profile page button	ProfilePageButton()
	On-click onto leaderboard button	LeaderboardButton()
	Handle typing into code field	handleCodeChangeField()

19.4.5 Local Functions

CreateGameButton():

- transition: WebSocketService.createGame()

JoinGameButton():

- transition: WebSocketService.joinGame()

FindGameButton():

- transition: WebSocketService.findGame(code)

ProfilePageButton():

- transition: Router.navigate('/profile')

LeaderboardButton():

- transition: Router.navigate('/profile')

handleCodeChangeField():

- transition: Modify code with new changes of input

20 Profile Page Module

20.1 Module

ProfilePage

20.2 Uses

[Router Module](#)

[User Module](#)

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event	-	-

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

Screen: A window displayed on user's screen

Browser: The user's browser

20.4.3 Assumptions

init() will be ran everytime the browser url is set to "/profile"

handleEvent() will be called by the browser for input events

20.4.4 Access Routine Semantics

init():

- transition: *currentStats* := "",
currentStats := User.getUserStats(Browser.params.id),
matchHistory := User.getUserMatches(Browser.params.id),
Screen := Displays the profile page with the *currentStats* and *matchHistory*.

handleEvent(event: Browser.Event):

• transition:	event	function
	On-click onto leave button	handleLeave()

20.4.5 Local Functions

handleLeave():

- transition: Router.navigate("/")

21 Leaderboard Page Module

21.1 Module

Leaderboard

21.2 Uses

[Router Module](#)

[User Module](#)

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event	-	-

21.4 Semantics

21.4.1 State Variables

None

21.4.2 Environment Variables

Screen : A window displayed on user's screen

Browser : The user's browser

21.4.3 Assumptions

init() will be ran everytime the browser url is set to "/leaderboard"

handleEvent() will be called by the browser for input events

21.4.4 Access Routine Semantics

init():

- transition: *currentLeaderboard* := "",
currentLeaderboard := User.getLeaderboard(),
Screen := Displays the leaderboard page with the currentLeaderboard.

handleEvent(event: Browser.Event):

• transition:	event	function
	On-click onto leave button	handleLeave()

21.4.5 Local Functions

handleLeave():

- transition: Router.navigate("/")

22 Lobby Page Module

22.1 Module

LobbyPage

22.2 Uses

[LobbyService Module](#)

[Router Module](#)

[WebSocketService Module](#)

22.3 Syntax

22.3.1 Exported Constants

None

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event		-

22.4 Semantics

22.4.1 State Variables

lobbyCode: String

currentLobby: seq of UserT

22.4.2 Environment Variables

screen : a browser window displayed on the user's screen.

clipboard : the user's clipboard, which string variables can be copied into.

Browser : Includes input events and params for browser url params

22.4.3 Assumptions

init() will be ran everytime the browser url is set to "/lobby"

handleEvent() will be called by the browser for input events

22.4.4 Access Routine Semantics

init():

- transition: $currentCode := \text{Browser.params}$,
 $currentLobby := \text{LobbyService.getWaitingRoom}()$,
 $Screen := \text{Displays the Lobby Page screen with lobbyCode displayed and a card for every user in currentLobby.}$

handleEvent(event: Browser.Event):

	event	function
	On-click onto copy code	copyCode()
• transition:	button	
	On-click onto start button	startGame()
	On-click onto leave button	leaveLobby()

22.4.5 Local Functions

copyCode():

- transition: $clipboard := lobbyCode$
- exception: $exc := \text{User does not grant permission to access clipboard content} \implies \text{PermissionDeniedException}$

startGame():

- transition: $|currentLobby| > 1 \implies \text{WebSocketService.nextRound}()$
- exception: None

leaveLobby():

- transition: $\text{Router.navigate}("/")$
- exception: None

23 Game Page Module

23.1 Module

GamePage

23.2 Uses

[ProblemsService Module](#)

[LobbyService Module](#)

[SubmissionService Module](#)

[Router Module](#)

23.3 Syntax

23.3.1 Exported Constants

None

23.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event	-	-

23.4 Semantics

23.4.1 State Variables

currentCode: string

currentProblem: ProblemT

currentLobby: seq of UserT

23.4.2 Environment Variables

Browser: Includes input events and params for browser url params

Screen

23.4.3 Assumptions

init() will be ran everytime the browser url is set to ”/problem”

handleEvent() will be called by the browser for input events

23.4.4 Access Routine Semantics

init():

- transition: *currentCode* := "",
currentProblem := ProblemsService.getProblem(Browser.params),
currentLobby := LobbyService.getWaitingRoom(),
Screen := Displays the Game Page screen with *currentCode* and *currentProblem*. Displays players current status in LobbyService.waitingRoom, If LobbyService.endData is not empty, displays end game screen.

handleEvent(event: Browser.Event):

	event	function
• transition:	On-click onto submission button	handleSubmit()
	On-click onto back button	handleLeave()
	Handle typing into submission box	handleCodeEdit()

23.4.5 Local Functions

handleSubmit():

- transition: SubmissionService.submitSolution(*currentCode*)

handleLeave():

- transition: Router.navigate("/")

handleCodeEdit():

- transition: Modify *currentCode* with new changes of input

displaySubmissionResult(judgeResult: JudgeResultT)

- transition: Display *judgeResult* into pop up onto *Screen*

24 Login Page Module

24.1 Module

LoginPage

24.2 Uses

[AuthService Module](#)

24.3 Syntax

24.3.1 Exported Constants

None

24.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
handleEvent	Browser.Event	-	-

24.4 Semantics

24.4.1 State Variables

usernameField: String passwordField: String profilePicField: String emailField: String

24.4.2 Environment Variables

screen : a browser window displayed on the user's screen.

24.4.3 Assumptions

init() will be ran everytime the browser url is set to "/login"

handleEvent() will be called by the browser for input events

24.4.4 Access Routine Semantics

init():

- transition: *screen* := update the browser window such all users see a login button.

handleEvent(event: Browser.Event):

	event	function
	On-click onto login button	handleLogin()
	Handle typing into user-name field	hanldeUserNameFieldEdit()
• transition:	Handle typing into password field	hanldePasswordFieldEdit()
	Handle typing into profile pic field	hanldeProfilePicFieldEdit()
	Handle typing into email field	hanldeEmailFieldEdit()

24.4.5 Local Functions

handleLogin():

- transition: AuthService.login(usernameField, passwordField, emailField, profilePicField)

hanldeUserNameFieldEdit():

- transition: Modify usernameField with new changes of input

hanldePasswordFieldEdit():

- transition: Modify passwordField with new changes of input

hanldeProfilePicFieldEdit():

- transition: Modify profilePicField with new changes of input

hanldeEmailFieldEdit():

- transition: Modify emailField with new changes of input

24.4.6 Considerations

The login module functions are implemented by Google's OAuth component

25 SubmissionService Module

25.1 Module

SubmissionService

25.2 Uses

[Judge Module](#)

25.3 Syntax

25.3.1 Exported Constants

None

25.3.2 Exported Access Programs

Name	In	Out	Exceptions
submitSolution	ProblemT, SubmissionT	JudgeResultT	

25.4 Semantics

25.4.1 State Variables

None

25.4.2 Environment Variables

None

25.4.3 Assumptions

The exported access programs successfully make an HTTP request to the Judge API and receive a response.

25.4.4 Access Routine Semantics

submitSolution(problem, submission):

- output: $out := Judge.judgeSubmission(problem, submission)$
- exception: None

25.4.5 Local Functions

None

26 ProblemsService Module

26.1 Module

ProblemsService

26.2 Uses

[Problems Module](#)

26.3 Syntax

26.3.1 Exported Constants

None

26.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProblem	String	ProblemT	-

26.4 Semantics

26.4.1 State Variables

None

26.4.2 Environment Variables

None

26.4.3 Assumptions

None

26.4.4 Access Routine Semantics

getProblem(problemID: string): ProblemT

- output: ProblemsModule.getProblem(problemT)

26.4.5 Local Functions

None

27 UserService Module

27.1 Module

UserService

27.2 Uses

[User Module](#)

27.3 Syntax

27.3.1 Exported Constants

27.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLeaderboard	-	seq of UserT	-
getUser	string	UserStats T	-
getUserMatches	string	Seq of MatchT T	-

27.4 Semantics

27.4.1 State Variables

27.4.2 Environment Variables

27.4.3 Assumptions

27.4.4 Access Routine Semantics

getLeaderboard():

- transition: `User.getLeaderboard()`

getUserStats(userid):

- output: `out := User.getUserStats(userid)`

getUserMatches(userid):

- output: `out := User.getUserMatches(userid)`

28 AuthService Module

28.1 Module

AuthService

28.2 Uses

[Auth Module](#)

[User Module](#)

28.3 Syntax

28.3.1 Exported Constants

28.3.2 Exported Access Programs

Name	In	Out	Exceptions
logout login	String, String, String, String		
isLoggedIn getToken			

28.4 Semantics

28.4.1 State Variables

user: UserT

28.4.2 Environment Variables

28.4.3 Assumptions

28.4.4 Access Routine Semantics

logout():

- transition: user := <>

login(username, password, password, profilepic):

- transition: user := Auth.login(username, password)
if user is empty then User.createUser(username, email, password, profilepic) then user
:= User.getUserByEmail(email)

isLoggedIn():

- $\text{output} := (\text{user is not empty} \implies \text{true} | \text{false})$

`getUser():`

- $\text{output} := \text{user}$

28.4.5 Local Functions

29 LobbyService Module

29.1 Module

LobbyService

29.2 Uses

None

29.3 Syntax

29.3.1 Exported Access Programs

Name	In	Out	Exceptions
updateWaitingRoom	Seq of User	-	-
updateEndData	\mathbb{B}	-	-
updateCurrentRound	\mathbb{N}	-	-
getWaitingRoom	-	Seq of User	-
getEndData	-	\mathbb{B}	-
getCurrentRound	-	\mathbb{N}	-

29.4 Semantics

29.4.1 State Variables

waitingRoom: Seq of User

endData: \mathbb{B}

currentRound: \mathbb{N}

29.4.2 Environment Variables

29.4.3 Assumptions

29.4.4 Access Routine Semantics

updateWaitingRoom(clients):

- transition: *waitingRoom* := *clients*

updateEndData(endData):

- transition: *endData* := *endData*

updateCurrentRound(round):

- transition: *currentRound* := *round*

getWaitingRoom():

- output: $out := waitingRoom$

getWaitingRoom():

- output: $out := endData$

getCurrentRound():

- output: $out := currentRound$

30 WebSocketService Module

30.1 Module

WebSocketService

30.2 Uses/Imports

[GameHandler Module](#)

[Router Module](#)

30.3 Syntax

30.3.1 Exported Constants

None

30.3.2 Exported Access Programs

None

Name	In	Out	Exceptions
createGame	-	-	-
findGame	-	-	-
joinGame	String	-	-
nextRound	-	-	-

30.4 Semantics

30.4.1 State Variables

currentGameID: string clientID: String

30.4.2 Environment Variables

UUID

gen() → string:

- output: *out* := returns a unique id
- exception: None

30.4.3 Assumptions

None

30.4.4 Access Routine Semantics

createGame():

- transition: $clientID, currentGameID := UUID.gen(),$
 $GameHandler.createGame(clientID)$
 $Router.navigate("/problem/{currentGameID}")$

findGame():

- transition: $currentGameID := GameHandler.findGame(clientID),$
 $GameHandler.joinGame(clientID, gameID),$
 $Router.navigate("/problem/{currentGameID}")$

joinGame(gameID):

- transition: $currentGameID := GameHandler.joinGame(clientID, gameID),$
 $Router.navigate("/problem/{currentGameID}")$

nextRound():

- transition: $GameHandler.gameNextRound(gameID)$
if

30.4.5 Local Functions

None

31 GameHandler Module

31.1 Module

GameHandler

31.2 Uses

[ClientT Module](#)

[GameT Module](#)

[Router Module](#)

[Problems Module](#)

31.3 Syntax

31.3.1 Exported Constants

None

31.3.2 Exported Access Programs

None

31.3.3 Exported Access Programs

Name	In	Out	Exceptions
addClient	String, String, String, String	-	-
createGame	String	String	-
findGame	String	String	-
joinGame	String, String	-	-
endGame	String	-	-
gameNextRound	String	-	-
playerCompleteRound	String, String	-	-
sendUpdatedPlayers	String	-	-

31.3.4 Environment Variables

UUID

gen() → string:

- output: *out* := returns a unique id
- exception: None

31.3.5 Assumptions

UUID technology is supported by implementation environment

31.4 Semantics

31.4.1 State Variables

clients: Seq of Tuple(ID, ClientT)

games: Seq of Tuple(ID, GameT)

31.4.2 Access Routine Semantics

addClient(clientID: String, email: String, name: String, picture: String)

- transition: $clients := clients \cup Client(clientID, email, name, picture, NULL, 0)$

createGame(clientID: String) : String

- transition:
 $gameID := UUID.gen() \wedge games := games \cup Game(gameID, [Clients[ClientID]], 0)$
- output: $out := gameID$

findGame(clientID: String) : String

- transition:
 $(\exists game \in games : game.round = 0) \Rightarrow game \in games : game.round = 0 \Rightarrow$
 $gameID = game.id \wedge$
 $(\forall game \in games : game.round > 0) \Rightarrow gameID = createGame()$
- output: $out := gameID$

joinGame(clientID: String, gameID: String)

- transition: $addClient(clientID) \Rightarrow$
 $(games[gameID].clients := games[gameID].clients. \cup clients[clientID]$
 $clients[clientID].game := gameID) \Rightarrow sendUpdatedPlayers(gameID)$

endGame(gameID: String)

- transition:
 $client \in games[gameID].clients : client.ls.updateEndData(client.lastCompletedRound =$
 $games[gameID].round) \Rightarrow games := games - games[gameID]$

gameNextRound(gameID: String)

- transition:

$$\begin{aligned} & (game, games[gameID].round := games[gameID], games[gameID].round + 1) \Rightarrow \\ & (game.round > 1 \Rightarrow client \in game.clients : client.lastCompletedRound < game.round \Rightarrow \\ & client.ls.updateEndData(false)) \Rightarrow game.clients := game.clients - client \Rightarrow \\ & sendUpdatedPlayers(gameID) \Rightarrow client \in clients : client.rm('problem/' + ProblemsModule.getRa \end{aligned}$$

playerCompleteRound(clientID: String, gameID: String)

- transition:

$$\begin{aligned} & clients[clientID].lastCompletedRound := clients[clientID].lastCompletedRound + 1 \\ & playersCompleted := size(client \in games[gameID] | client.lastCompletedRound = \\ & games[gameID].round) \\ & playersCompleted = 1 \wedge size(games[gameID].clients = 2) \Rightarrow endGame(gameID) \vee \\ & playersCompleted \geq size(games[gameID].clients) / 2 > 1 \Rightarrow gameNextRound(gameID) \end{aligned}$$

31.4.3 Local Functions

sendUpdatedPlayers(gameID: String)

- transition:

$$client \in games[gameID].clients : client.ls.updateWaitingRoom(games[gameID].clients)$$

getUsersInLobby(gameID: String): set of Users

- output: $out := \{users : gameID | client \exists games[gameID].clients : User(client)\}$

32 Judge Module

32.1 Module

Judge

32.2 Uses

[CodeRunner Module](#)

[GameHandler Module](#)

[ProblemT Module](#)

[SubmissionT Module](#)

[JudgeResultT Module](#)

[JudgeVerdict Module](#)

32.3 Syntax

32.3.1 Exported Constants

None

32.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	CodeRunner	Judge	
judgeSubmission	ProblemT, SubmissionT, String, String	JudgeResultT	

32.4 Semantics

32.4.1 State Variables

codeRunner: CodeRunner

32.4.2 Assumptions

The init function is called at the start of the life-cycle of the module before any other access routine is called for that object. The user of the module provides a CodeRunner instance.

32.4.3 Access Routine Semantics

init(cr):

- transition: *codeRunner* := *cr*
- output: *out* := *self*

- exception: None

judgeSubmission(problem, submission, clientID, gameID):

- transition: $(JudgeResultT(getVerdicts(problem, submission)).getVerdict() = Correct) \implies gameHandler.playerCompleteRound(clientID, gameID)$
- output: $out := JudgeResultT(getVerdicts(problem, submission))$
- exception: None

32.4.4 Local Functions

getVerdicts(p: ProblemT, s: SubmissionT) \rightarrow set of JudgeVerdict:

- output: $out := \{tc : TestCaseT \mid tc \in p.getTestCases() : codeRunner.runCode(s, tc)\}$
- exception: None

33 CodeRunner Module

33.1 Module

CodeRunner

33.2 Uses

[SubmissionT Module](#)

[TestCaseT Module](#)

[TestCaseVerdictT Module](#)

33.3 Syntax

33.3.1 Exported Constants

None

33.3.2 Exported Types

CodeRunner = ?

33.3.3 Exported Access Programs

Name	In	Out	Exceptions
runCode	SubmissionT, TestCaseT	TestCaseVerdictT	

33.4 Semantics

33.4.1 State Variables

None

33.4.2 Environment Variables

containerEngine: A containerized run-time environment providing intractability with the run-time's standard input and standard output streams as well as code compilation capabilities.

compileAndRun(code: string, language: Language, input: string, timeLimit: \mathbb{R} , memoryLimit: \mathbb{N}) \rightarrow string:

- output: *out* := the output from the standard output stream

	condition	exc :=
	Code execution time exceeds <i>timeLimit</i>	TimeoutException
• exception:	Memory used exceeds <i>memoryLimit</i>	MemoryExceededException
	Code crashes unexpectedly	RuntimeException
	Code fails to compile	CompileException

33.4.3 Assumptions

The container engine is provided enough memory to run with the specified memory limit. The container engine supports the provided input language. The container engine is capable of emitting signals that correspond to the specified exceptions.

33.4.4 Access Routine Semantics

runCode(submission, testCase):

	condition	out :=
	No exceptions \wedge <i>userOutput</i> = <i>testCase.output</i>	TestCaseVerdictT(Correct, userOutput, testCase)
	No exceptions \wedge <i>userOutput</i> \neq <i>testCase.output</i>	TestCaseVerdictT(Wrong, userOutput, testCase)
• output:	TimeoutException	TestCaseVerdictT(TimeLimitExceeded, <>, testCase)
	MemoryExceededException	TestCaseVerdictT(MemoryLimitExceeded, <>, testCase)
	RuntimeException	TestCaseVerdictT(RuntimeError, <>, testCase)
	CompileException	TestCaseVerdictT(CompileError, <>, testCase)

where *userOutput* = *compileAndRun(submission.getCode(), submission.getLanguage(), testCase.getInput(), testCase.getTimeLimit(), testCase.getMemoryLimit())*

- exception: None

33.4.5 Local Functions

None

34 Auth Module

34.1 Module

Auth

34.2 Uses

[Database Module](#)

34.3 Syntax

34.3.1 Exported Constants

34.3.2 Exported Access Programs

Name	In	Out	Exceptions
login	String, String	UserT	

34.4 Semantics

34.4.1 State Variables

34.4.2 Environment Variables

34.4.3 Assumptions

34.4.4 Access Routine Semantics

login(username, password):

- output:= Database.getUserByUsernameAndPassword(username, password)

34.4.5 Local Functions

None

35 Problems Module

35.1 Module

Problems

35.2 Uses

[Database Module](#)

35.3 Syntax

35.3.1 Exported Constants

35.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProblem	String	Problem	DoesNotExist
getRandomProblem	-	Problem	DoesNotExist

35.4 Semantics

35.4.1 State Variables

35.4.2 Environment Variables

35.4.3 Assumptions

35.4.4 Access Routine Semantics

getProblem(problemID: String): Problem

- output: *out* := problem where there exists a problem in Database.getProblems() where problem.ID equals problemID
- exception: There exists no problem in Database.getProblems() where problem.ID equals problemID \implies DoesNotExist

getRandomProblem(): Problem

- output: *out* := single random selection in Database.getProblems()
- exception: There exists no problem in Database.getProblems() \implies DoesNotExist

35.4.5 Local Functions

36 User Module

36.1 Template Module

User

36.2 Uses

[Database Module](#)

[UserT Module](#)

[MatchT Module](#)

36.3 Syntax

36.3.1 Exported Constants

36.3.2 Exported Access Programs

Name	In	Out	Exceptions
createUser	String, String, String, String	-	-
saveUserMatch	MatchT	-	-
getUserMatches	String	Seq of MatchT	-
getUser	String	UserT	-
getUserByEmail	String	UserT	-
getUserStats	String	UserStatsT	-
getLeaderboard	-	Seq of UserT	-

36.4 Semantics

36.4.1 State Variables

36.4.2 Environment Variables

UUID

gen() → string:

- output: *out* := returns a unique id
- exception: None

36.4.3 Assumptions

UUID technology is supported by implementation environment

36.4.4 Access Routine Semantics

`createUser(username, password, email, profilepic):`

- transition: `Database.createUser(new UserT(UUID.gen(), username, password, email, profilepic))`
- exception: `None`

`saveUserMatch(match):`

- transition: `userMatches[userID] := userMatches[userID] \cup match`
- exception: `None`

`getUserMatches(userID):`

- output: `out := Database.getUserMatches(userID)`
- exception: `None`

`getUser(userID):`

- output: `out := Database.getUser(userID)`
- exception: `None`

`getUserByEmail(email):`

- output: `out := Database.getUserByEmail(email)`
- exception: `None`

`getUserStats(userID):`

- output: `out := Database.getUserStats(userID)`
- exception: `None`

`getLeaderboard():`

- output: `out := Database.getLeaderboard(userID)`
- exception: `None`

36.4.5 Local Functions

`None`

37 Database Module

37.1 Module

Database

37.2 Uses

[ProblemT Module](#)

[UserStatsT Module](#)

[UserT Module](#)

[MatchT Module](#)

37.3 Syntax

37.3.1 Exported Constants

37.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProblems	-	Set of ProblemT	-
addProblems	ProblemT	-	-
createUser	UserT	-	-
saveMatch	MatchT	-	-
getUserMatches	String	set of MatchT	-
getUser	String	UserT	-
getUserByUsernameAndPassword	String, String	UserT	-
getUserStats	String	UserStatsT	-
getLeaderboard	-	seq of UserT	-

37.4 Semantics

37.4.1 State Variables

problems: set of ProblemT

users: set of UserT

userStats: set of UserStatsT

matches: set of MatchT

37.4.2 Environment Variables

None

37.4.3 Assumptions

None

37.4.4 Access Routine Semantics

getProblems():

- output: $out := \text{Problems}$
- exception: None

addProblems(newProblem):

- transition: $problems := \text{problems} \cup \text{newProblem}$
- exception: None

createUser(user):

- transition: $users := \text{users} \cup \text{user}$
- exception: None

saveMatch(match):

- transition: $\text{matches} \cup \text{match}$
- exception: None

getUserMatches(userID):

- output: $out := \text{userMatches}$ such that userMatches is all userMatch in matches where $\text{userMatch.getUserID}() = \text{userID}$
- exception: None

getUser(userID):

- output: $out := \text{user}$ such that user exists in users where $\text{user.userID} = \text{userID}$
- exception: None

getUserByUsernameAndPassword(username):

- output: $out := \text{user}$ such that user exists in users where $\text{user.getUsername}() = \text{username}$ and $\text{user.getPassword}() = \text{password}$

- exception: DoesNotExist when user is not in users

getUserStats(userID):

- output: *out* := user such that user exists in users where user.userID = userID
- exception: None

getLeaderboard():

- output: *out* := topUsers where the topUsers is the first 100 user in users where users is sorted in descending order based on user.wins
- exception: None

37.4.5 Local Functions

None

38 Router Module

38.1 Module

Router

38.2 Uses

None

38.3 Syntax

38.3.1 Exported Constants

None

38.3.2 Exported Access Programs

Name	In	Out	Exceptions
navigate	string	-	-

38.4 Semantics

38.4.1 State Variables

None

38.4.2 Environment Variables

browser: the web browser used to handle page navigation
navigate(path: string):

- transition: set the web browser url to path
- exception: None

38.4.3 Assumptions

None

38.4.4 Access Routine Semantics

navigate(path: string):

- transition: Browser.navigate(path)
- exception: None

38.4.5 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>.

39 Appendix

None