Module Interface Specification for SFWRENG 4G06

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

January 19, 2023

1 Revision History

Date	Version	Notes
01/17/2023	1.0	Initial draft
01/18/2023	1.1	Small Changes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation here.

Contents

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

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 SFWRENG 4G06. The boolean and map data types were added as well.

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 (-∞, ∞)
boolean	$\mathbb{B}$	truth values in $\{true, false\}$
map	$map < T1, T2 >$	a generic symbol table supporting getting and setting values using indexing notation

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 Module	
Behaviour-Hiding Module	ClientT Module GameT Module MatchT Module UserT Module UserStatsT Module ProblemT Module Difficulty Module Language Module TestCaseT Module SubmissionT Module JudgeResultT Module JudgeVerdict Module TestCaseVerdictT Module HomePage Module ProfilePage Module LeaderboardPage Module LobbyPage Module GamePage Module LoginPage Moudle SubmissionService Module ProblemsService Module UserService Module AuthService Module LobbyService Module WebSocketService Module GameHandler Module Judge Module Auth Module Problems Module User Module
Software Decision Module	CodeRunner Module Database Module Router Module

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

None

8.3 Syntax

8.3.1 Exported Types

MatchT = ?

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
MatchT	String, $\mathbb{Z}$, String	MatchT	
getUserID		String	
getIsWin		$\mathbb{B}$	
getDate		String	

8.4 Semantics

8.4.1 State Variables

userID: String
win: $\mathbb{B}$
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, r, d):

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

getUserID():

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

getIsWin():

- output: $out := win$
- 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	UserT	
getUserID		String	
getUsername		String	
getEmail		String	
getProfilePicture		String	

9.4 Semantics

9.4.1 State Variables

userID: String
username: 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, e, pic):

- transition: $userID, username, email, profilePicture := id, u, e, pic$
- 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

None

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 Constants

MINIMUM_MEMORY = 32
MINIMUM_TIME = 0.5

11.3.2 Exported Types

ProblemT = ?

11.3.3 Exported Access Programs

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

11.4 Semantics

11.4.1 State Variables

id: string
name: string
description: string
timeLimit: $\mathbb{R}$
memoryLimit: $\mathbb{N}$
testCases: seq of TestCaseT
difficulty: Difficulty
problemType: seq of string

11.4.2 State Invariant

$\forall\, tc : TestCaseT \mid tc \in testCases \;:\; tc.getMemoryLimit() = memoryLimit \wedge tc.getTimeLimit() = timeLimit$

11.4.3 Environment Variables

None

11.4.4 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.

11.4.5 Considerations

The minimum memory represents the memory in megabytes. The minimum time represents the time in seconds.

11.4.6 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: $exc := (tl < MINIMUM\_TIME \vee ml < MINIMUM\_MEMORY) \implies IllegalArgumentException$

getID():

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

getName():

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

getDescription():

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

getTimeLimit():

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

getMemoryLimit():

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

getTestCases():

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

getDifficulty():

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

getProblemType():

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

11.4.7 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, Language	SubmissionT	
getCode		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 TestCaseVerdictT	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: $verdict := ((\forall\, v : TestCaseVerdictT \mid v \in testCaseVerdicts \,:\, v.getVerdict() = Correct) \implies Correct) \vee (tcv : JudgeVerdict$ 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	JudgeVerdict, String, TestCaseT	TestCaseVerdictT	
getVerdict		JudgeVerdict	
getUserOutput		String	
getTestCase		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

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):

- transition:

event	function
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

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

handleEvent(event: Browser.Event):

- transition:

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

22.4.5 Local Functions

copyCode():

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

startGame():

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

leaveLobby():

- transition: Router.navigate("/")
- exception: None

23 Game Page Module

23.1 Module

GamePage

23.2 Uses

ProblemsService Module
SubmissionService 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):

- transition:

event	function
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):

- transition:

event	function
On-click onto login button	handleLogin()
Handle typing into username field	hanldeUserNameFieldEdit()
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
Router 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		$\mathbb{B}$	
getToken		UserT	

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) then Router.navigate("/")

isLoggedIn():

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

getUser():

- output:= 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

GameHandler Module
User 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

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

Date: Gets the current time on the machine
getTime() → string:

- output: *out* := returns the current time on the machine
- exception: None

31.3.5 Assumptions

UUID technology is supported by implementation environment

31.4 Semantics

31.4.1 State Variables

clients: map<String, ClientT>
games: map<String, GameT>

31.4.2 Access Routine Semantics

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

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

createGame(clientID: String,) : String

- transition:
  $gameID := UUID.gen() \ \wedge \ games := games \cup$
  $Tuple(gameID, GameT(gameID, [Clients[ClidentID]], 0))$
- 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:
  $client \in games[gameID].clients : client.ls.updateEndData(client.lastCompletedRound =$
  $games[gameID].round) \Rightarrow games := games - games[gameID] \Rightarrow$
  $User.saveMatch(MatchT(client.getUserId(),$
  $client.lastCompletedRound = games[gameID].round, Date.getTime()))$

gameNextRound(gameID: String)

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

playerCompleteRound(clientID: String, gameID: String)

- transition:
  $clients[clientID].lastCompletedRound := clients[clientID].lastCompletedRound + 1$
  $playersCompleted := size(client \in games[gameID]|client.lastCompletedRound =$
  $games[gameID].round)$
  $playersCompleted = 1 \wedge size(games[gameID].clients = 2) \Rightarrow endGame(gameID) \vee$
  $playersCompleted \geq size(games[gameID].clients)/2 > 1 \Rightarrow gameNextRound(gameID)$

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
JudgeResultT 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) → 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
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

- exception:

condition	exc :=
Code execution time exceeds *timeLimit*	TimeoutException
Memory used exceeds *memoryLimit*	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):

- output:

condition	out :=
No exceptions $\wedge$ $userOutput = testCase.output$	TestCaseVerdictT(Correct, userOutput, testCase)
No exceptions $\wedge$ $userOutput \neq testCase.output$	TestCaseVerdictT(Wrong, userOutput, testCase)
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

None

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

None

34.4.2 Environment Variables

OAuth: Authentication provided by oauth

34.4.3 Assumptions

None

34.4.4 Access Routine Semantics

login(username, password):

- output:= OAuth.login(username, password)

34.4.5 Local Functions

None

34.4.6 Considerations

This module will be implemented through the usage of Google's OAuth

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

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, email, profilepic):

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

saveUserMatch(match):

- transition: Database.saveMatch(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	-
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* := Problems
- exception: None

addProblems(newProblem):

- transition: *problems* := problems ∪ newProblem
- exception: None

createUser(user):

- transition: users := users ∪ user
- exception: None

saveMatch(match):

- transition: matches ∪ match
- exception: None

getUserMatches(userID):

- output: *out* := userMatches such that userMatches is all userMatch in matches where userMatch.getUserID() = userID
- exception: None

getUser(userID):

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

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