Module Interface Specification for Sayyara Automotive Matcher

Team 27, Kappastone
Tevis Doe, doet
Gilbert Cherrie, cherrieg
Rachel Johnson, johnsr12
Harkeerat Kanwal, kanwalh
Himanshu Aggarwal, aggarwah

January 19, 2023

1 Revision History

Date	Version	Notes
January 18th, 2023	0.0	Rev 0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS for Sayyara Automotive Matcher

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
FR	Functional Requirement
M	Module
MG	Module Guide
OS	Operating System
PWA	Progressive Web Application
Sayyara Automotive Matcher	PWA for maintenance appointment scheduling for vehicle owners
SRS	Software Requirements Specification
UC	Unlikely Change
UI	User Interface

Contents

3 Introduction

The following document details the Module Interface Specifications for the Sayyara Automotive Matcher: An application designed to ease communcation between mechanics and potential clients.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/HKanwal/kapstone.

4 Notation

The structure of the MIS for modules comes from Software Design, Automated Testing, and Maintenance: A Practical Approach (Hoffman Strooper, 1995), with the addition that template modules have been adapted from Fundamentals of Software Engineering (Ghezzi et al., 2003). The mathematical notation comes from Chapter 3 of Software Design, Automated Testing, and Maintenance: A Practical Approach. 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 Sayyara Automotive Matcher.

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 (-∞, ∞)

The specification of Sayyara Automotive Matcher 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, Sayyara Automotive Matcher 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	N/A
Behaviour-Hiding Module	UI Module
	Authentication Module
	Dashboard Module
	Shop Creation Module
	Quote Request Module
	Quote Module
	Chat Module
	Account Information Module
	Work Order Module
	Service Module
	Appointment Module
Software Decision Module	Available Appointments Module
	Update Appointments Module
	Appointment Slots Module
	Update Appointment Slots Module

Table 1: Module Hierarchy

6 MIS of UI Module

6.1 Module

UI Module

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
useThemePlease	string, string	-	-

6.4 Semantics

6.4.1 State Variables

primaryColor: string
secondaryColor: string

6.4.2 Assumptions

6.4.3 Access Routine Semantics

useThemePlease(primaryColor: string, secondaryColor: string):

- transition: primaryColor = $primaryColor \wedge$ secondaryColor = $secondaryColor$
- output: None
- exception: None

6.4.4 Local Functions

None

7 MIS of Authentication Module

7.1 Module

Authentication Module

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
pleaseCheckTheseCredentials	string, string	bool	SomethinAintRightException
pleaseRegisterMe	string, string	-	-

7.4 Semantics

7.4.1 State Variables

username: string
password: string

7.4.2 Environment Variables

7.4.3 Assumptions

7.4.4 Access Routine Semantics

pleaseCheckTheseCredentials(username, password):

- transition: None
- output: username = *username* $\wedge$ password = *password*
- exception: (username! = *username*$\vee$password! = *password*) $\implies$ SomethinAintRightException

pleaseRegisterMe(username, password):

- transition: username = *username* $\wedge$ password = *password*
- output: None
- exception: None

7.4.5 Local Functions

None

8 MIS of Dashboard Module

8.1 Module

Dashboard

8.2 Uses

UI

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
getAccessiblePages	string		

8.4 Semantics

8.4.1 State Variables

allPages: sequence of Pages
accessiblePages: sequence of Pages

8.4.2 Environment Variables

8.4.3 Assumptions

The arguments provided to access program will be of correct type.

8.4.4 Access Routine Semantics

getAccessiblePages(userType):

- transition: $(\forall i : \text{Page} | i \in \text{allPages} \wedge i.user = userType : \text{accessiblePages} + \text{i})$
- output: None
- exception: None

8.4.5 Local Functions

None

9 MIS of Shop Creation Module

9.1 Module

ShopCreation

9.2 Uses

UI

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
newShop	sequence of strings		InvalidInfo
updateShop	integer, sequence of strings		InvalidID, InvalidInfo

9.4 Semantics

9.4.1 State Variables

shopInfo: sequence of strings

9.4.2 Access Routine Semantics

newShop(input):

- transition: $shopInfo := input$
- output: $out := self$
- exception: $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

updateShop(shop, input):

- transition: $shopInfo :=$ input
- output: none
- exceptions: $(invalidID(shop) \Rightarrow InvalidID)$, $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

9.4.3 Local Functions

invalidValue(s, i): Verifies that s is a valid parameter for a shop's data at position i.
invalidID(shop): Verifies that the given shop ID corresponds with an existing shop in the API.

10 MIS of Quote Request Module

10.1 Module

QR Module

10.2 Uses

UI

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
newQuoteRequest	sequence of strings		InvalidInfo
updateQuoteRequest	integer, sequence of strings		InvalidID, InvalidInfo
getQuoteRequest	integer	Quote Request	InvalidID
sendQuoteRequest	integer		InvalidID

10.4 Semantics

10.4.1 State Variables

quoteRequestInfo

10.4.2 Access Routine Semantics

newQuoteRequest(input):

- transition: *quoteRequestInfo* := input
- output: *out* := *self*
- exception: $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

updateQuoteRequest(quoteRequest, input):

- transition: *quoteRequestInfo* := input
- output: none

- exceptions: $(invalidID(quoteRequest) \Rightarrow InvalidID)$, $(\forall val \in |input| | invalidValue(val[i], i) \Rightarrow InvalidInfo)$

getQuoteRequest(quoteRequest):

- transition: none
- output: out := quoteRequest
- exceptions: $(invalidID(quoteRequest) \Rightarrow InvalidID)$

sendQuoteRequest(quoteRequest):

- transition: quoteRequest → Quote Module
- output: none
- exceptions: $(invalidID(quoteRequest) \Rightarrow InvalidID)$

10.4.3 Local Functions

invalidValue(s, i): Verifies that s is a valid parameter for a shop's data at position i.
invalidID(quoteRequest): Verifies that the given quote request ID corresponds with an existing quote request in the API.

11 MIS of Quote Module

11.1 Module

Quote

11.2 Uses

UI

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
newQuote	sequence of strings		InvalidInfo
updateQuote	integer, sequence of strings		InvalidID, InvalidInfo
getQuote	integer	Quote	InvalidID
sendQuote	integer		InvalidID

11.4 Semantics

11.4.1 State Variables

quoteInfo

11.4.2 Access Routine Semantics

newQuote(input):

- transition: *quoteInfo* := input
- output: *out* := *self*
- exception: $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

updateQuote(quote, input):

- transition: *quoteInfo* := input
- output: none

- exceptions: $(invalidID(quote) \Rightarrow InvalidID)$, $(\forall val \in |input| | invalidValue(val[i], i) \Rightarrow InvalidInfo)$

getQuote(quote):

- transition: none
- output: out := quote
- exceptions: $(invalidID(quote) \Rightarrow InvalidID)$

sendQuote(quote):

- transition: quote → Quote Request Module
- output: none
- exceptions: $(invalidID(quote) \Rightarrow InvalidID)$

11.4.3 Local Functions

invalidValue(s, i): Verifies that s is a valid parameter for a shop's data at position i.
invalidID(quote): Verifies that the given quote ID corresponds with an existing quote in the API.

12 MIS of Chat Module

12.1 Module

Chat

12.2 Uses

UI

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
new Chat	string	Chat	
SendMessage	string		
GetMessages		sequence of strings	

12.4 Semantics

12.4.1 State Variables

recipient: string

12.4.2 Environment Variables

12.4.3 Assumptions

12.4.4 Access Routine Semantics

new Chat(r):

- transition: *recipient* := *r*
- output:*out* := *self*
- exception: none

SendMessage(s):

- transition: send s to API

- exception: none

GetMessages():

- transition: get messages from API
- output: *out* := sequence of strings
- exception: none

12.4.5 Local Functions

None

13 MIS of Account Information Module

13.1 Module

AccountInformation

13.2 Uses

UI

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
new AccountInformation	sequence of strings	AccountInformation	Invalid Information
getInfo		sequence of strings	
edit	sequence of strings		Invalid Information
delete			

13.4 Semantics

13.4.1 State Variables

info: sequence of strings

13.4.2 Environment Variables

13.4.3 Assumptions

13.4.4 Access Routine Semantics

new AccountInformation(i):

- transition: *info* := *i*
- output: *out* := *self*

- exception: $(\forall val \in |i||invalidValue(val[i], i) \Rightarrow InvalidInformation)$

  getInfo():

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

  edit(i):

  - transition: *info* := *i*

- exception: $(\forall val \in |i||invalidValue(val[i], i) \Rightarrow InvalidInformation)$

  delete(i):

  - transition: *info* := *[]*
  - exception: None

13.4.5 Local Functions

invalidValue(s,i): check that s is a valid value for a string in position i

14 MIS of Work Order Module

14.1 Module

WorkOrder

14.2 Uses

UI

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
updateWorkOrder	int, sequnece of strings		InvalidID, InvalidInfo
getWorkOrder	int	workOrder	InvalidID

14.4 Semantics

14.4.1 State Variables

workOrderInfo: sequence of strings

14.4.2 Environment Variables

None

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

updateWorkOrder(workOrder, input):

- transition: *workOrderInfo* := *input*
- output: none

- exception: $(invalidID(workOrder) \Rightarrow InvalidID)$, $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

getWorkOrder(workOrder):

- transition: none
- output: out := workOrder
- exceptions: $(invalidID(workOrder) \Rightarrow InvalidID)$

14.4.5 Local Functions

invalidValue(s, i): Verifies that s is a valid parameter for a shop's data at position i.
invalidID(workOrder): Verifies that the given work order ID corresponds with an existing work order in the API.

15 MIS of Service Module

15.1 Module

Service

15.2 Uses

UI

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
newService	sequence of strings		InvalidInfo
updateService	int, sequence of strings		InvalidInfo, InvalidID
getService	int	Service	InvalidID

15.4 Semantics

15.4.1 State Variables

serviceInfo: sequence of strings

15.4.2 Environment Variables

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

newService(input):

- transition: *serviceInfo* := input
- output: *out* := *self*
- exception: $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

updateService(service, input):

- transition: *serviceInfo* := input
- output: none
- exceptions: $(invalidID(service) \Rightarrow InvalidID)$, $(\forall val \in |input||invalidValue(val[i], i) \Rightarrow InvalidInfo)$

getService(service):

- transition: none
- output: out := service
- exceptions: $(invalidID(service) \Rightarrow InvalidID)$

15.4.5 Local Functions

invalidValue(s, i): Verifies that s is a valid parameter for a shop's data at position i.
invalidID(service): Verifies that the given service ID corresponds with an existing service in the API.

16 MIS of Appointment Module

16.1 Module

Appointment

16.2 Uses

UI

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
create	string, string		
get	string	string	
edit	string, string		

16.4 Semantics

16.4.1 State Variables

appointments: sequence of sequences of strings

16.4.2 Environment Variables

16.4.3 Assumptions

16.4.4 Access Routine Semantics

create(name, time):

- transition: *appointments* := *appointments + [name,time]*
- exception: None

get(name):

- output: *output* := *appointments[i] where appointments[i][0] = name*
- exception: None

create(name, time):

- transition: *appointments* := *appointments + [name,time]*
- exception: None

16.4.5 Local Functions

None

17 MIS of Available Appointments Module

17.1 Module

Available Appointments

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
list	String, Date, Date, Integer	Dictionary	None

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

apt_slots: Database table containing Appointment Slots (set)

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

list(shop, from, to, duration):

- transition: None
- output: out := $\{e : AppointmentSlot \mid e \in apt\_slots \ \wedge \ e.from \geq from \ \wedge \ e.to \leq to \ \wedge \ e.shop = shop : get\_slots(e, duration)\}$
- exception: None

17.4.5 Local Functions

get_slots : AppointmentSlot $\times$ Duration $\rightarrow$ AppointmentSlotSet
get_slots(a, d) $\equiv$ $(s : \mathit{AppointmentSlotSet} \mid s[0] = a \ \wedge \ s[|s| \ - \ 1] = d : s)$

18 MIS of Update Appointments Module

18.1 Module

Update Appointments

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
update	String, Date	None	None

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

apt_slots: Database table containing Appointment Slots (set)

18.4.3 Assumptions

None

18.4.4 Access Routine Semantics

update(shop, date):

- transition:
  $(\forall i : AppointmentSlot \mid i \in apt\_slots \land i.date = date \land i.shop = shop \land i.is\_cancelled : cancel(i.appointments))$
- output: None
- exception: None

18.4.5 Local Functions

cancel : set of Appointment $\rightarrow$ *None*
cancel(s) $\equiv$ $(\forall j : Appointment | j \in s : j.cancel())$

19 MIS of Appointment Slots Module

19.1 Module

Appointment Slots

19.2 Uses

None

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
generate	String, Date	None	None

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

apt_slots: Database table containing Appointment Slots (set)

19.4.3 Assumptions

None

19.4.4 Access Routine Semantics

generate(shop, date):

- transition:
  $apt\_slots :=$ apt_slots $\cup$ AppointmentSlotSet($\langle e : ShopAvailability | : generate\_slots(e) \rangle$)
- output: None
- exception: None

19.4.5 Local Functions

generate_slots : ShopAvailability → set of AppointmentSlot
generate_slots(s) ≡ $\{i : \mathbb{N} | i \in [s.start..s.end, s.step] : \text{new AppointmentSlot}(s.shop, i, i + s.step)\}$

20 MIS of Update Appointment Slots Module

20.1 Module

Update Appointment Slots

20.2 Uses

None

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
update	String, Date	None	None

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

apt_slots: Database table containing Appointment Slots (set)

20.4.3 Assumptions

None

20.4.4 Access Routine Semantics

update(shop, date):

- transition:
  $apt\_slots := (\text{AppointmentSlotSet}(\langle e : AppointmentSlot | e.date = date \wedge e.shop = shop \wedge check\_hours(e, shop.hours, date) : e \rangle))$
- output: None
- exception: None

20.4.5 Local Functions

check_hours : AppointmentSlot × set of ShopAvailability × Date → $\mathbb{B}$
check_hours(e, hours, date) $\equiv (\exists i : \mathbb{N} | i \in [0..|hours| - 1] : e.start \geq hours[i].start \wedge e.end \leq hours[i].end)$