Module Interface Specification for Sayyara

Team 3, Tiny Coders
Arkin Modi
Joy Xiao
Leon So
Timothy Choy

April 5, 2023

1 Revision History

Table 1: Revision History

Date	Developer(s)	Change
December 28, 2022	Arkin Modi	Create Revision History
January 7, 2023	Joy Xiao	Introduction
January 9, 2023	Arkin Modi	Add Module Hierarchy
January 11, 2023	Arkin Modi	Create MIS of Users Module
January 14, 2023	Arkin Modi	Create MIS of Work Orders Module
January 13, 2023	Arkin Modi	Create MIS of Database Driver Module
January 13, 2023	Arkin Modi	Create MIS of Services Module
January 15, 2023	Arkin Modi	Create MIS of Appointments Module
January 15, 2023	Leon So	Create MIS of Shop Module
January 15, 2023	Arkin Modi	Create MIS of Quotes Module
January 15, 2023	Joy Xiao	Create MIS of Employee Management Module
January 17, 2023	Timothy Choy	Add Shop Lookup to Shop Module
March 1, 2023	Arkin Modi	Update Quote Request Bodies
March 7, 2023	Timothy Choy	Update Shop Module MIS
March 14, 2023	Arkin Modi	Update Appointments Module
March 23, 2023	Joy Xiao	Update Employee Management Module
March 26, 2023	Arkin Modi	Remove createShop from Shop Module
April 2, 2023	Timothy Choy	Update createShopOwner from User Module
April 3, 2023	Timothy Choy	Update getShops for location
April 4, 2023	Arkin Modi	Add descriptions for modules
April 4, 2023	Arkin Modi	Update modules to match source code

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/arkinmodi/project-sayyara/blob/main/docs/SRS/SRS.pdf

symbol	description
CRUD	Create, Read, Update, Delete
MIS	Module Interface Specifications
MG	Module Guide
Sayyara	The name of the program being built

Contents

List of Tables

List of Figures

3 Introduction

The following document details the Module Interface Specifications for project Sayyara. Sayyara is a progressive web application (PWA) which will act as a single platform for independent auto repair shops and vehicle owners. This platform will allow independent auto repair shops and vehicle owners to interact in a more efficient and effective manner. Vehicle owners can search for auto repair shops and services; request quotes for service; book, view, and manage service appointments. On the application, auto repair shop owners will be able to manage a list of employees; manage a list of service types and corresponding service appointment availabilities; manage store information such as location, hours of operation, and contact information. Auto repair shop owners and employees will be able to manage quotes, service appointments, and work orders from a single application. The MIS will detail specifications for the project described above.

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

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 Sayyara.

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 $(-\infty, \infty)$
real positive	$\mathbb{R}+$	any number in $[0, \infty)$

The specification of Sayyara 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 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	Users Module
	Quotes Module
	Appointments Module
	Work Orders Module
	Employee Management Module
	Services Module
	Shop Module
Software Decision Module	Database Driver Module

Table 2: Module Hierarchy

6 MIS of Database Driver Module

The Database Driver Module provides a collection of exported types representing the database's schema and handles communication with the database. This module is implemented in combination with the external library, Prisma.

6.1 Module

schema.prisma

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Types

Employee

Output Name	Output Type	Description
id	String	ID of Employee
createTime	String	Create Time of Employee Account
updateTime	String	Update Time of Employee Account
firstName	String	First Name of Employee
lastName	String	Last Name of Employee
phoneNumber	String	Phone Number of Employee
email	String	Email of Employee
password	String	Password of Employee's Account
type	String	Type of User
shop	Shop	Shop which Employee is registered under
appointments	List<Appointment>	Appointments assigned to the Employee
workOrders	List<Work Order>	Work Orders assigned to the Employee
status	String	Employee status (i.e., Active or Suspended)

Customer

Output Name	Output Type	Description
id	String	ID of Customer
createTime	String	Create Time of Customer Account
updateTime	String	Update Time of Customer Account
firstName	String	First Name of Customer
lastName	String	Last Name of Customer
phoneNumber	String	Phone Number of Customer
email	String	Email of Customer
password	String	Password of Customer's Account
type	String	Type of User
appointments	List<Appointment>	Appointments assigned to the Customer
chatMessages	List<ChatMessage>	Chat Messages sent by the Customer
quotes	List<Quote>	Quotes initiated by the Customer
vehicles	List<Vehicle>	Vehicles associated with the Customer
workOrders	List<Work Order>	Work Orders assigned to the Customer

Appointment

Output Name	Output Type	Description
id	String	ID of Appointment
createTime	String	Create Time of Appointment
updateTime	String	Update Time of Appointment
quote	Optional<Quote>	Associated Quote
workOrder	Work Order	Associated Work Order
vehicle	Vehicle	Associated Vehicle
serviceType	String	Type of Service
employee	Optional<Employee>	Assigned Employee
customer	Customer	Assigned Customer
status	String	Progress Status of Appointment
startTime	DateTime	Start Time
endTime	DateTime	End Time
shop	Shop	Associated Shop
price	$\mathbb{R}+$	Price of appointment
service	Service	Associated Service
cancellationReason	String	Reason of cancellation

Quote

Output Name	Output Type	Description
id	String	ID of Quote
createTime	String	Create Time of Quote
updateTime	String	Update Time of Quote
customer	Customer	Assigned Customer
shop	Shop	Associated Shop
appointments	List<Appointment>	Associated Appointments
chatMessages	List<ChatMessage>	Associated Chat Messages
status	String	Status indicating whether a invitation has been created
estimatedPrice	$\mathbb{R}+$	Estimated price of job
duration	$\mathbb{R}+$	Estimated time need for job
service	Service	Associated Service
description	Optional<String>	Description of work needed

ChatMessage

Output Name	Output Type	Description
id	String	ID of Chat Message
createTime	String	Create Time of Chat Message
updateTime	String	Update Time of Chat Message
message	String	Chat Message
quote	Quote	Associated Quote
customer	Customer	Associated Customer
shop	Shop	Associated Shop

Vehicle

Output Name	Output Type	Description
id	String	ID of the Vehicle
createTime	String	Create Time of the Vehicle
updateTime	String	Update Time of the Vehicle
year	$\mathbb{N}$	Model Year of the Vehicle
make	String	Make of the Vehicle
model	String	Model of the Vehicle
vin	String	VIN of the Vehicle
licensePlate	String	License Plate of the Vehicle
customer	Customer	Associated Customer
appointments	List<Appointment>	Appointments assigned to the Vehicle
workOrders	List<Work Order>	Work Orders assigned to the Vehicle

WorkOrder

Output Name	Output Type	Description
id	String	ID of Work Order
createTime	String	Create Time of Work Order
updateTime	String	Update Time of Work Order
appointment	Appointment	Associated Appointment
title	String	Title of Work Order
customer	Customer	Assigned Customer
vehicle	Vehicle	Associated Vehicle
employee	Employee	Assigned Employee
body	String	Work Order Details
shop	Shop	Associated Shop

Shop

Output Name	Output Type	Description
id	String	ID of Shop
createTime	String	Create Time of Shop
updateTime	String	Update Time of Shop
name	String	Name of Shop
address	String	Address of Shop
city	String	City of Shop
province	String	Province of Shop
postalCode	String	Postal Code of Shop
email	String	Email of Shop
phoneNumber	String	Phone Number of Shop
hoursOfOperation	JSON	Hours of Operation of Shop
latitude	String	Latitude of Shop
longitude	String	Longitude of Shop
appointments	List<Appointment>	Appointments assigned to Shop
workOrders	List<Work Order>	Work Orders assigned to Shop
employees	List<Employee>	List of Employee assigned to Shop
chatMessages	List<ChatMessage>	List of Chat Messages sent by Shop
quotes	List<Quote>	List of Quotes
services	List<Service>	List of offered Services

Part

Output Name	Output Type	Description
quantity	$\mathbb{N}$	Number of Parts
cost	$\mathbb{R}+$	Price of Parts
name	String	Name of Part
condition	String	Condition of Part (i.e., New or Used)
build	String	Origin of Part (i.e., OEM or After Market)

Service

Output Name	Output Type	Description
id	String	ID of Service
createTime	String	Create Time of Service
updateTime	String	Update Time of Service
name	String	Name of Service
description	String	Description of Service
estimatedTime	$\mathbb{N}$	Estimated time needed for a Service
totalPrice	Optional< $\mathbb{R}+$ >	Estimated price for a Service
parts	List<Part>	List of parts for a Service
shop	Shop	Shop which offers the Service
type	String	Type of Service
appointments	List<Appointment>	Appointments associated to the Service
quotes	List<Quote>	Quotes associated to the Service

6.3.3 Exported Access Programs

None

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

None

6.4.5 Local Functions

None

7   MIS of Shop Module

The Shop Module provides CRUD operations for shop objects and the ability to query for a list of shops by shop name, service name, distance, part types, and part conditions.

7.1   Module

shopService

7.2   Uses

Database Driver Module

7.3   Syntax

7.3.1   Exported Constants

None

7.3.2   Exported Types

CreateShopType

Output Name	Output Type	Description
name	String	Name of Shop
address	String	Address of shop
city	String	City of Shop
province	String	Province of Shop
postalCode	String	Postal Code of Shop
email	String	Email of Shop
phoneNumber	String	Phone Number of Shop
location	tuple of (lat, long : $\mathbb{R}$)	Latitude and Longitude of Shop

UpdateShopType

Output Name	Output Type	Description
name	Optional<String>	Name of Shop
address	Optional<String>	Address of shop
city	Optional<String>	City of Shop
province	Optional<String>	Province of Shop
postalCode	Optional<String>	Postal Code of Shop
hoursOfOperation	Optional<JSON>	Hours of Operation of Shop
email	Optional<String>	Email of Shop
phoneNumber	Optional<String>	Phone Number of Shop

7.3.3 Exported Access Programs

Name	In	Out	Exceptions
getShopById	String	Shop ∨ None	
updateShopById	String, UpdateShopType	Shop	ShopNotFoundException, InvalidTimeException
getShopsByName	String, Optional< ℝ+ >, Optional< ℝ+ >	List<Shop> ∨ None	
getShopsByService	String, Optional< ℝ+ >, Optional< ℝ+ >	List<Shop> ∨ None	

7.4 Semantics

7.4.1 State Variables

7.4.2 Environment Variables

User's Display
Database

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

getShopById(id):

- output: $out :=$ A Shop such that it contains the ID, id, from the shop database table else None.

updateShopById($id, patch$):

- transition: Update all fields of a Shop, with an ID equal to id, with fields in $patch$ in shop database table. Location is updated locally through the getLatLong(address) local function.

- output: $out := getShopById(id) = \text{Shop} \Rightarrow (\text{Shop} = \text{None}$
  $\Rightarrow \text{ShopNotFoundException} \mid \text{Shop} \neq \text{None}$
  $\Rightarrow$ update all fields of Shop with fields in $patch$ in shop database table)

- exception: $exc := getShopById(id) = None \Rightarrow \text{ShopNotFoundException}$
  $\mid \exists day \in hoursOfOperation : day.openTime > day.closeTime$
  $\Rightarrow \text{InvalidTimeException}$

getShopsByName($name, minDistance, maxDistance$):

- output: $out := getShopsByName(name) = (\forall s \in Shops | s.name \in name \land minDistance \leq distance(s, user) \leq maxDistance)$

getShopsByService($service, minDistance, maxDistance$):

- output: $out := getShopsByService(service)$
  $= (\forall s \in Shops | \exists serv \in getServicesByShopId(s) | serv \in service \land minDistance \leq distance(s, user) \leq maxDistance)$

7.4.5 Local Functions

distance($shop, user$):

- output: $out := \arccos(\sin(\text{user.latitude}) * \sin(\text{shop.latitude}) + \cos(\text{user.latitude}) * \cos(\text{shop.latitude}) * \cos(\text{user.longitude} - \text{shop.longitude})) * 6371$

8 MIS of Users Module

The User Module provides CRUD operations for user objects, and the ability to verify a user's authorization.

8.1 Module

userService

8.2 Uses

Database Driver Module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Types

CreateCustomerType

Output Name	Output Type	Description
email	String	Email of Customer
password	String	Password of Customer Account
firstName	String	First Name of Customer
lastName	String	Last Name of Customer
phoneNumber	String	Phone Number of Customer
vehicle.year	$\mathbb{N}$	Model Year of Customer's Vehicle
vehicle.make	String	Make of Customer's Vehicle
vehicle.model	String	Model of Customer's Vehicle
vehicle.vin	String	VIN of Customer's Vehicle
vehicle.licensePlate	String	License Plate of Customer's Vehicle

CreateEmployeeType

Output Name	Output Type	Description
email	String	Email of Employee
password	String	Password of Employee's Account
firstName	String	First Name of Employee
lastName	String	Last Name of Employee
phoneNumber	String	Phone Number of Employee
shopId	String	ID of Shop that Employee Works For

CreateShopOwnerType

Output Name	Output Type	Description
email	String	Email of Shop Owner
password	String	Password of Shop Owner's Account
firstName	String	First Name of Shop Owner
lastName	String	Last Name of Shop Owner
phoneNumber	String	Phone Number of Shop Owner
shop.name	String	Name of Shop owned by Shop Owner
shop.address	String	Address of Shop owned by Shop Owner
shop.postalCode	String	Postal Code of Shop owned by Shop Owner
shop.city	String	City of Shop owned by Shop Owner
shop.province	String	Province of Shop owned by Shop Owner
shop.email	String	Email of Shop owned by Shop Owner
shop.phoneNumber	String	Phone Number of Shop owned by Shop Owner

AuthorizeReturnType

Output Name	Output Type	Description
id	String	User ID
firstName	String	First Name of User
lastName	String	Last Name of User
email	String	Email of User
type	String	Type of User
shopId	Optional<String>	Shop the User works at

8.3.3 Exported Access Programs

Name	In	Out	Exceptions
createCustomer	CreateCustomerType	Customer	CustomerAlreadyExistsException
createEmployee	CreateEmployeeType	Employee	EmployeeAlreadyExistsException
createShopOwner	CreateShopOwnerType	Employee	ShopOwnerAlreadyExistsException
getUserByEmail	String	Customer ∨ Employee ∨ None	
getEmployeeById	String	Employee ∨ None	
getCustomerById	String	Customer ∨ None	
authorize	String, String	AuthorizeReturnType	UnauthorizeException

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

User's Display
Database

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

createCustomer($customer$):

- transition: new Customer(customer), save customer to customer database table

- output: $out := (exc = $ CustomerAlreadyExistsException
  $\Rightarrow$ "User with email address already exists." $| \neg exc \Rightarrow$ new Customer($customer$))

- exception: $exc := getUserByEmail(customer.email) \neq$ None
  $\Rightarrow$ CustomerAlreadyExistsException

createEmployee($employee$):

- transition: new Employee(employee), save employee to employee database table

- output: $out := (exc = $ EmployeeAlreadyExistsException
  $\Rightarrow$ "User with email address already exists." $| \neg exc \Rightarrow$ new Employee($employee$))

- exception: $exc := getUserByEmail(employee.email) \neq$ None
  $\Rightarrow$ EmployeeAlreadyExistsException

createShopOwner($shopOwner$):

- transition: new Employee(shopOwner), save shop owner to employee database table
  and creates a new shop

- output: $out := (exc = $ ShopOwnerAlreadyExistsException
  $\Rightarrow$ "User with email address already exists." $| \neg exc \Rightarrow$ new Employee($shopOwner$)$\wedge$
  new Shop($shop.name, shop.address, shop.city, shop.province, shop.postalCode,$
  $shop.email, shop.phoneNumber$))

- exception: $exc := getUserByEmail(shopOwner.email) \neq$ None
  $\Rightarrow$ ShopOwnerAlreadyExistsException

getUserByEmail(e):

- output: $out := $ A User such that it contains the email, e, from the customer database
  table or employees database table, else None.

getEmployeeById(id):

- output: $out := $ A User such that it contains the ID, id, from the employees database
  table, else None.

getCustomerById(id):

- output: $out := $ A User such that it contains the ID, id, from the customer database
  table, else None.

authorize($email, password$):

- output: $out := getUserByEmail(email) = \text{User} \Rightarrow$
  (User = None $\Rightarrow exc = \text{UserNotFoundException} \Rightarrow$ "User not found."
  | User.password $= password \Rightarrow \text{AuthorizeReturnType}$
  | User.password $\neq password \Rightarrow exc = \text{UnauthorizeException} \Rightarrow$ "Unauthorized")

8.4.5 Local Functions

Shop($name, address, city, province, postalCode, email, phoneNumber$):

- transition: Creates a new shop object with the following parameters: the provided name, address, city, province, postal code, email, and phone number.

9 MIS of Employee Management Module

The Employee Management Module provides CRUD operations for managing employee's relation with shop.

9.1 Module

employeeManagementService

9.2 Uses

Database Driver Module

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Types

UpdateEmployeeType

Output Name	Output Type	Description
email	Optional<String>	Email of Employee
firstName	Optional<String>	First Name of Employee
lastName	Optional<String>	Last Name of Employee
phoneNumber	Optional<String>	Phone Number of Employee

9.3.3 Exported Access Programs

Name	In	Out	Exceptions
getEmployeeById	String	Employee	
getEmployeesByShopId	String	List<Employee>	
updateEmployeeById	String, UpdateEmployeeType		UserNotFoundException
suspendEmployeeById	String		

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

User's Display
Database

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

getEmployeeById($employeeId$):

- output: out := An employee such that it's ID matches $employeeId$ from the employee database table else None.

getEmployeesByShopId($shopId$):

- output: out := List of employees from the shop database table associated with the $shopId$, else None.

updateEmployeeById($employeeId, patch$):

- transition: Update all fields of Employee with fields in $patch$ in employee database table

- output: $out := getEmployeeById(id) =$ Employee $\Rightarrow$ (Employee $=$ None $\Rightarrow$ UserNotFoundException | Employee $\neq$ None $\Rightarrow$ update all fields of Employee with fields in $patch$ in service database table)

- exception: $exc := getEmployeeById(id) = None \Rightarrow$ UserNotFoundException

suspendEmployeeById($employeeId$):

- transition: Change the state of the employee to "suspended" where their ID matches $employeeId$ in the employee database table.

9.4.5 Local Functions

None

10 MIS of Services Module

The Services Module provides CRUD operations for service objects.

10.1 Module

serviceService

10.2 Uses

Database Driver Module

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Types

CreateServiceType

Output Name	Output Type	Description
name	String	Name of Service
description	Optional<String>	Description of Service
estimatedTime	Optional<String>	Estimated Time of Service
parts	List<Part>	Estimated Parts needed for Service
totalPrice	Optional< $\mathbb{R}+$ >	Estimated Price needed for Service
shopId	String	Shop that Service is available at
type	String	Type of Service (e.g., Canned or Custom)

UpdateServiceType

Output Name	Output Type	Description
name	Optional<String>	Name of Service
description	Optional<String>	Description of Service
estimatedTime	Optional<String>	Estimated Time of Service
parts	Optional<List<Part>>	Estimated Parts needed for Service
totalPrice	Optional< $\mathbb{R}+$ >	Estimated Price needed for Service

10.3.3 Exported Access Programs

Name	In	Out	Exceptions
createService	CreateServiceType	Service	
getServiceById	String	Service $\vee$ None	
getServicesByShopId	String	List<Service>	
getCannedServicesByShopId	String	List<Service>	
getCustomServicesByShopId	String	List<Service>	
updateServiceById	String, UpdateServiceType	Service	ServiceNotFoundException
deleteServiceById	String		

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

User's Display
Database

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

createService(*service*):

- transition: new Service(service), save service to service database table

- output: $out :=$ new Service(*service*)

getServiceById(*id*):

- output: $out :=$ A Service such that it's ID matches *id* from the service database table else None.

getServicesByShopId(*shopId*):

- output: $out :=$ A list of Service such that it's shop ID matches *shopId* from the service database table else an empty list.

getCannedServicesByShopId(*shopId*):

- output: $out :=$ A list of Service such that it's shop ID matches *shopId*, and its type is "Canned" from the service database table else an empty list.

getCustomServicesByShopId(*shopId*):

- output: $out :=$ A list of Service such that it's shop ID matches *shopId*, and its type is "Custom" from the service database table else an empty list.

updateServiceById(*id*, *patch*):

- transition: Update all fields of Service with fields in *patch* in service database table

- output: $out := getServiceById(id) = \text{Service} \Rightarrow (\text{Service} = \text{None}$ $\Rightarrow \text{ServiceNotFoundException} \mid \text{Service} \neq \text{None}$ $\Rightarrow$ update all fields of Service with fields in *patch* in service database table)

- exception: $exc := getServiceById(id) = None \Rightarrow \text{ServiceNotFoundException}$

deleteServiceById(*id*):

- transition: Delete all Services where their ID matches *id* from the service database table.

18

10.4.5   Local Functions

None

11 MIS of Appointments Module

The Appointments Module provides CRUD operations for appointment objects, the ability to verify if an appointment is acceptable (i.e., is possible to fulfill), and the ability to accept an appointment and reject all conflicting appointments.

11.1 Module

appointmentService

11.2 Uses

Database Driver Module

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Types

CreateAppointmentType

Output Name	Output Type	Description
quoteId	Optional<String>	Associated Quote
serviceId	String	Associated Service
price	$\mathbb{Q}$	Estimated Price
employeeId	Optional<String>	Assigned Employee
startTime	Date	Start Time of Appointment
endTime	Date	End Time of Appointment
vehicleId	String	Vehicle Being Serviced
customerId	String	Customer the Appointment is for
shopId	String	Shop the Appointment is taking place at

UpdateAppointmentType

Output Name	Output Type	Description
quoteId	Optional<String>	Associated Quote
workOrderId	Optional<String>	Associated Work Order
price	Optional< $\mathbb{Q}$ >	Estimated Price
employeeId	Optional<String>	Assigned Employee
startTime	Optional<Date>	Start Time of Appointment
endTime	Optional<Date>	End Time of Appointment
vehicleId	Optional<String>	Vehicle Being Serviced
status	Optional<String>	Acceptance status of appointment
cancellationReason	Optional<String>	Reason of cancellation

11.3.3 Exported Access Programs

Name	In	Out	Exceptions
createAppointment	CreateAppointmentType	Appointment	InvalidTimeException, MissingDataException
getAppointmentById	String	Appointment ∨ None	
getAppointmentsByShopId	String	List<Appointment>	
getAppointmentsByCustomerId	String	List<Appointment>	
updateAppointmentById	String, UpdateAppointmentType	Appointment	AppointmentNotFoundException, InvalidTimeException
deleteAppointment	String		

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

User's Display
Database

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

createAppointment($appointment$):

- transition: new Appointment(appointment) as "Pending Approval", save appointment to appointment database table. new WorkOrder() connected to the newly created appointment, save work order to work order database table.

- output: $out := (exc = $ InvalidTimeException
  $\Rightarrow$ "Invalid start time and/or end time." $| \neg exc \Rightarrow$ new Appointment($appointment$))

- exception: $exc := (appointment.vehicleId = $ None $\vee appointment.customerId = $ None $\vee appointment.shopId = $ None $\Rightarrow$ MissingDataException)
  $| (\neg isAppointmentValid(appointment.shopTd, appointment.startTime, appointment.endTime)$
  $\Rightarrow$ InvalidTimeException)

getAppointmentById(id):

- output: $out :=$ An Appointment such that it's ID matches id from the appointment database table else None.

getAppointmentsByShopId($shopId$):

- output: $out :=$ A list of Appointments such that their shop ID matches $shopId$ from the appointment database table else an empty list.

getAppointmentsByCustomerId($customerId$):

- output: $out :=$ A list of Appointments such that their customer ID matches $customerId$ from the appointment database table else an empty list.

updateAppointmentById($id, patch$):

- transition: Update all fields of an Appointment, with an ID equal to id, with fields in *patch* in appointment database table

- output: $out := getAppointmentById(id) = $ Appointment
$\Rightarrow$ (Appointment = None $\Rightarrow$ AppointmentNotFoundException
| Appointment $\neq$ None
$\Rightarrow$ update all fields of Appointment with fields in *patch* in appointment database table)

- exception: $exc := (getAppointmentById(id) = None \Rightarrow$ AppointmentNotFoundException)
$| (getAppointmentById(id) = Appointment \Rightarrow (patch.startTime \neq None \wedge patch.endTime \neq None \wedge \neg isAppointmentValid(Appointment.shopId, patch.startTime, patch.endTime)) \vee$
$(patch.startTime \neq None$
$\wedge \neg isAppointmentValid(Appointment.shopId, patch.startTime, Appointment.endTime)) \vee$
$(patch.endTime \neq None$
$\wedge \neg isAppointmentValid(Appointment.shopId, Appointment.startTime, patch.endTime))$
$\Rightarrow$ InvalidTimeException)

deleteAppointment(id):

- transition: Delete all Appointments where their ID matches id from the appointment database table.

11.4.5 Local Functions

acceptAppointment($appointment$):

- transition: Mark the appointment as "Accepted" and mark all other appointment with overlapping time slot as "Rejected".

isAppointmentValid($shopId, startTime, endTime$):

- output: $out := $ True if the given appointment is within the hours of operation for the shop else False.

12 MIS of Work Orders Module

The Work Orders Module provides CRUD operations for work order objects.

12.1 Module

workOrderService

12.2 Uses

Database Driver Module

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Types

UpdateWorkOrderType

Output Name	Output Type	Description
title	Optional<String>	Title of Work Order
body	Optional<String>	Work Order Details
employeeId	Optional<String>	ID of Assigned Employee
employeeEmail	Optional<String>	Email of Assigned Employee

12.3.3 Exported Access Programs

Name	In	Out	Exceptions
getWorkOrderById	String	WorkOrder ∨ None	
getWorkOrdersByShopId	String	List<WorkOrder> ∨ None	
updateWorkOrderById	String, UpdateWorkOrderType	WorkOrder	WorkOrderNotFoundException

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

User's Display
Database

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

getWorkOrderById(*id*):

- output: *out* := A Work Order such that it's ID matches *id* from the work order database table else None.

getWorkOrdersByShopId(*shopId*):

- output: *out* := A list of Work Orders such that their shop ID matches *shopId* from the work order database table else an empty list.

updateWorkOrderById(*id*, *patch*):

- transition: Update all fields of a Work Order, with an ID equal to *id*, with fields in *patch* in work order database table

- output: $out := getWorkOrderById(id) = \text{WorkOrder} \Rightarrow (\text{WorkOrder} = \text{None} \Rightarrow \text{WorkOrderNotFoundException} \mid \text{WorkOrder} \neq \text{None} \Rightarrow$ update all fields of WorkOrder with fields in *patch* in work order database table)

- exception: $exc := getWorkOrderById(id) = None \Rightarrow \text{WorkOrderNotFoundException}$

12.4.5 Local Functions

None

13 MIS of Quotes Module

The Quotes Module provides CRUD operations for quote and chat message objects.

13.1 Module

quoteService
chatService

13.2 Uses

Database Driver Module

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Types

CreateQuoteType

Output Name	Output Type	Description
customerId	String	ID of Customer
shopId	String	ID of Shop
serviceId	String	ID of Service

UpdateQuoteType

Output Name	Output Type	Description
status	Optional<String>	Status indicating whether an invitation has been created
estimatedPrice	Optional< $\mathbb{R}+$ >	Estimated price of job
duration	Optional< $\mathbb{R}+$ >	Estimated time need for job
description	Optional<String>	Description of quote

CreateChatMessageType

Output Name	Output Type	Description
customerId	Optional<String>	ID of Customer
shopId	Optional<String>	ID of Shop
message	String	Chat message

13.3.3   Exported Access Programs

Name	In	Out	Exceptions
createQuote	CreateQuoteType	Quote	
updateQuoteById	String, UpdateQuoteType	Quote	QuoteNotFoundException
getQuoteById	String	Quote ∨ None	
getQuotesByCustomerId	String	List<Quote>	
getQuotesByShopId	String	List<Quote>	
deleteQuoteAndChatById	String		
createChatMessage	CreateChatMessageType, String	ChatMessage	MissingSenderException, QuoteNotFoundException
getChatMessagesByQuoteId	String	List<ChatMessage>	

13.4   Semantics

13.4.1   State Variables

None

13.4.2   Environment Variables

User's Display
Database

13.4.3   Assumptions

None

13.4.4   Access Routine Semantics

createQuote($quote$):

- transition: new Quote(quote), save quote to quote database table

- output: $out :=$ new Quote($quote$)

updateQuoteById($id, patch$):

- transition: Update all fields of a Quote, with an ID equal to id, with fields in $patch$ in quote database table

- output: $out := getQuoteById(id) =$ Quote $\Rightarrow$ (Quote $=$ None
$\Rightarrow$ QuoteNotFoundException | Quote $\neq$ None
$\Rightarrow$ update all fields of Quote with fields in $patch$ in quote database table)

- exception: $exc := getQuoteById(id) = None \Rightarrow$ QuoteNotFoundException

getQuoteById(id):

- output: $out :=$ A Quote such that it's ID matches id from the quote database table else None.

getQuotesByCustomerId($customerId$):

- output: $out :=$ A list of Quotes such that it's customer ID matches $customerId$ from the quote database table else an empty list.

getQuotesByShopId($shopId$):

- output: $out :=$ A list of Quotes such that it's shop ID matches $shopId$ from the quote database table else an empty list.

deleteQuoteAndChatById(id):

- transition: Delete all Quote and ChatMessage where their quote ID matches id from the quote database table and the chat message database table, respectively.

createChatMessage($chatMessage, quoteId$):

- transition: new ChatMessage($chatMessage, quoteId$), save quote to quote database table

- output: $out := (exc = QuoteNotFoundException \Rightarrow$ "Quote Not Found")
  $| exc = MissingSenderException \Rightarrow$ "Missing Sender Information"
  $| \neg exc \Rightarrow$ new ChatMessage($chatMessage, quoteId$)

- exception: $exc := (getQuoteById(quoteId) = None \Rightarrow QuoteNotFoundException$
  $| chatMessage.customerId = $ None $\land chatMessage.shopId = $ None
  $\Rightarrow MissingSenderException)$

getQuoteById($quoteId$):

- output: $out :=$ A list of Chat Messages such that their quote ID matches $quoteId$ from the chat message database table else an empty list.

13.4.5   Local Functions

None

14   Bibliography

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.

15   Appendix