Module Interface Specification for Sayyara

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

January 13, 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 User Module

2 Symbols, Abbreviations and Acronyms

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

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

symbol	**description**
Sayyara	Explanation of program name
MIS	Module Interface Specifications
MG	Module Guide

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

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Sayyara.

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 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 Lookup Module Shop Profile Module
Software Decision Module	Database Driver Module

Table 2: Module Hierarchy

6 MIS of Users Module

6.1 Module

userService

6.2 Uses

Database Driver Module

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Types

CreateCustomerType

Output Name	**Output Type**	**Description**
email	String	Email of Customer
password	String	Password of Customer Account
first_name	String	First Name of Customer
last_name	String	Last Name of Customer
phone_number	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.license_plate	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
first_name	String	First Name of Employee
last_name	String	Last Name of Employee
phone_number	String	Phone Number of Employee
shop_id	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
first_name	String	First Name of Shop Owner
last_name	String	Last Name of Shop Owner
phone_number	String	Phone Number of Shop Owner
shop.—	String	. . .

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

6.3.3 Exported Access Programs

Name	In	Out	Exceptions
createCustomer	CreateCustomerType	Customer	CustomerAlreadyExistsException
createEmployee	CreateEmployeeType	Employee	EmployeeAlreadyExistsException
createShopOwner	CreateShopOwnerType	Employee	ShopOwnerAlreadyExistsException
getUserByEmail	String	Customer $\vee$ Employee $\vee$ None	UserNotFoundException
authorize	String, String	AuthorizeReturnType	UnauthorizeException

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

User's Display
Database

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

createCustomer($customer$):

- transition: new Customer(customer), save customer to customer database table
- output: out := (exc = CustomerAlreadyExistsException
  ⇒ "User with email address already exists."| $\neg exc$ ⇒ new Customer($customer$))
- exception: $exc := getUserByEmail(customer.email) \neq$ None
  ⇒ CustomerAlreadyExistsException

createEmployee($employee$):

- transition: new Employee(employee), save employee to employee database table
- output: out := (exc = EmployeeAlreadyExistsException
  ⇒ "User with email address already exists."| $\neg exc$ ⇒ new Employee($employee$))
- exception: $exc := getUserByEmail(employee.email) \neq$ None
  ⇒ EmployeeAlreadyExistsException

createShopOwner($shopOwner$):

- transition: new Employee(shopOwner), save shop owner to employee database table
- output: out := (exc = ShopOwnerAlreadyExistsException
  ⇒ "User with email address already exists."| $\neg exc$ ⇒ new Employee($shopOwner$))
- exception: $exc := getUserByEmail(shopOwner.email) \neq$ None
  ⇒ 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.

authorize($email, password$):

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

6.4.5 Local Functions

hash(s):

- output: out := hashed value of s

7 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.

8 Appendix

[Extra information if required —SS]