Module Interface Specification for Sayyara

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

January 11, 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

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 $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$

The specification of 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 [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R**??**. —SS]

[It is also possible to use LaTeX for hyperlinks to external documents. —SS]

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

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]