Module Guide 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 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

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

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The layout of the UI components on the screens

AC2: The fields of the various forms (eg. maybe vehicle make becomes no longer required when creating a quote request)

AC3: The theme (color palette) or the app

AC4: The branding of the app (logos and name)

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Change in framework or stack (we will be sticking with NextJS, Django, PostgreSQL)

UC2: Changing from PWA to Native

UC3: Change in platform

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: UI Module

M2: Authentication Module

M3: Dashboard Module

M4: Shop Creation Module

M5: Quote Request Module

M6: Quote Module

M7: Chat Module

M8: Account Information Module

M9: Work Order Module

M10: Service Module

M11: Appointment Module

M12: Available Appointments Module

M13: Update Appointments Module

M14: Appointment Slots Module

M15: Update Appointment Slots Module

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Sayyara Automotive Matcher* means the module will be implemented by the Sayyara Automotive Matcher software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

Level 1	**Level 2**
Hardware-Hiding Module	N/A
Behaviour-Hiding Module	UI Module (M1) Authentication Module (M2) Dashboard Module (M3) Shop Creation Module (M4) Quote Request Module (M5) Quote Module (M6) Chat Module (M7) Account Information Module (M8) Work Order Module (M9) Service Module (M10) Appointment Module (M11)
Software Decision Module	Available Appointments Module (M12) Update Appointments Module (M13) Appointment Slots Module (M14) Update Appointment Slots Module (M15)

Table 1: Module Hierarchy

7.1 Hardware Hiding Modules

Since we are building a high-level web app that is already highly abstracted away from hardware concerns, we do not have any hardware hiding modules.

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 UI Module M1

Secrets: Component UI logic.

Services: Encapsulates information pertaining to the look of all UI components and handles their logic and states

Implemented By: Frontend

Type of Module: Abstract Object

7.2.2 Authentication Module M2

Secrets: User credential authenticity

Services: Allows user to create unique credentials and authenticates user, allowing them to log in

Implemented By: Frontend

Type of Module: Abstract Object

7.2.3 Dashboard Module M3

Secrets: Page access of logged in user

Services: Allows user to locate pages they have access to depending on account type

Implemented By: Frontend

Type of Module: Abstract Object

7.2.4 Shop Creation Module M4

Secrets: Shop Creation form data state

Services: Submission of form will send request to backend to create shop or show error.

Implemented By: Frontend

Type of Module: Abstract Object

7.2.5 Quote Request Module M5

Secrets: Quote request data for users

Services: Vehicle owners can create, edit, view and delete quote requests

Implemented By: Frontend

Type of Module: Abstract Object

7.2.6 Quote Module M6

Secrets: Quote data for user

Services: Shop owners can create, edit, view and delete quotes

Implemented By: Frontend

Type of Module: Abstract Object

7.2.7 Chat Page Module M7

Secrets: Message history data

Services: To send a message, sends a request to the backend which updates the chat of both parties.

Implemented By: Frontend

Type of Module: Abstract Object

7.2.8 Account Information Module M8

Secrets: User account data

Services: All users are able to manage personal account info and shop owners can alo manage employee account data

Implemented By: Frontend

Type of Module: Abstract Object

7.2.9 Work Order Module M9

Secrets: Work order data for shop

Services: Shop owners/employees can view and edit work orders

Implemented By: Frontend

Type of Module: Abstract Object

7.2.10 Service Module M10

Secrets: Service type data

Services: Shop owners/employees can create, edit, view and delete shop services

Implemented By: Frontend

Type of Module: Abstract Object

7.2.11 Appointment Module M11

Secrets: User appointment data

Services: Data of all the appointments is received from the backend. Users are able to manage appointments and view appointment information.

Implemented By: Frontend

Type of Module: Abstract Object

7.3 Software Decision Module

7.3.1 Available Appointments Module M12

Secrets: Algorithm for finding available appointments.

Services: Finds available appointment slots given a range of dates and a duration.

Implemented By: Python

7.3.2 Update Appointments Module M13

Secrets: Algorithm to update existing appointments.

Services: Updates/cancels existing appointments automatically when a change occurs in the shop's available hours or in the number of employees.

Implemented By: Python

7.3.3 Appointment Slots Module M14

Secrets: Algorithm to generate appointment slots.

Services: Generates appointment slots based on a shop's hours and the available employees and bays.

Implemented By: Python

7.3.4 Update Appointment Slots Module M15

Secrets: Algorithm to update/delete appointment slots.

Services: Updates/deletes appointment slots if a change in the shop hours or employees results in overbooking.

Implemented By: Python

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR1	M1, M4
FR2	M1, M2, M8
FR3	M1, M2, M8
FR4	M1, M2, M8
FR5	M1,M8
FR6	M1, M8
FR7	M1, M8
FR8	M1, M2
FR9	M1, M2
FR10	M1, M6
FR11	M1, M6
FR12	M1, M11, M12, M13, M14, M15
FR13	M1, M2, M8
FR14	M1, M8
FR15	M1, M9
FR16	M1, M9
FR17	M1, M8
FR18	M1, M11, M12, M13
FR19	M1, M8, M13, M15
FR20	M1, M11, M12, M13, M14, M15
FR21	M1, M9, M7
FR22	M1, M6
FR23	M1, M9
FR24	M1, M7
FR25	M9
FR26	M9
FR27	M1, M9
FR28	M1, M11
FR29	M1, M5
FR30	M5

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M4, M5, M6, M10
AC3	M1
AC4	M1

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

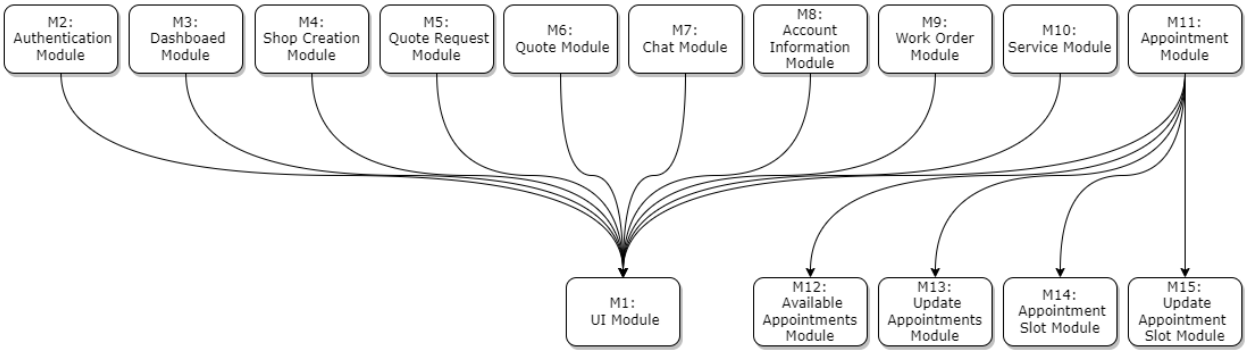

Figure 1: Use hierarchy among modules