Module Guide for Software Engineering

Team #13, ARC
Avanish Ahluwalia
Russell Davidson
Rafey Malik
Abdul Zulfiqar

January 17, 2025

1 Revision History

Date	Version	Notes
2025-01-17	1.0	Initial Version
2025-03-26	1.1	Fixed minor mistakes in the dates
2025-04-01	1.2	Changes for Rev1

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change

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 supported hardware device platforms and versions.

AC2: The API endpoints on the server.

AC3: The supported in-app settings.

AC4: The tour creation workflow.

AC5: The algorithm to generate AR Objects.

AC6: The features available on different devices due to performance or hardware limitations.

AC7: The provider of the maps/routing functionality

AC8: The algorithm used to determine which AR Objects should be rendered in the Realm Screen.

AC9: The render style of AR Objects (shaders/materials).

AC10: The interface layout for all the screens.

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: The file format of AR Objects (including metadata).

UC2: The account details associated with a user.

UC3: Changes in login procedure (sequence of events to be carried out for a successful login to the application)

UC4: The types of users (General and Organization).

UC5: The game engine used (Unity for rendering and platform support)

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: Hardware Module

M2: Inventory Module

M3: Touring Module

M4: Tour List Module

M5: Tour Management Module

M6: Settings Module

M7: Maps Module

M8: Realm Interface Module

M9: Object Prompt Generation Module

M10: Object Placement Module

M11: Object Interaction Module

M12: Object Render Module

M13: Collision Detection Module

M14: Restricted Area Detection Module

M15: Weather Detection Module

M16: Tour Proximity Detection Module

M17: Popup Module

M18: REST API Connection Module

M19: Local Database Manager Module

M20: Data Sync Module

M21: Server Database Manager Module

M22: Authentication Module

Level 1	**Level 2**
Hardware-Hiding Module	Hardware Module
Behaviour-Hiding Module	Inventory Module
	Touring Module
	Tour List Module
	Tour Management Module
	Settings Module
	Maps Module
	Realm Interface Module
	Object Prompt Generation Module
	Object Placement Module
	Object Interaction Module
	Object Render Module
	Collision Detection Module
	Restricted Area Detection Module
	Weather Detection Module
	Tour Proximity Detection Module
	Popup Module
Software Decision Module	REST API Connection Module
	Local Database Manager Module
	Data Sync Module
	Server Database Manager Module
	Authentication Module

Table 1: Module Hierarchy

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 5.

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. *Software Engineering* means the module will be implemented by the Software Engineering 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.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software, allowing the system to display outputs or accept inputs.

Implemented By: OS

7.1.1 Device Input/Output Module

Secrets: How data is collected from and sent to device sensors like the camera, GPS, or microphone.

Services: Provides a standard interface for capturing sensor input (e.g., photos, location data) and delivering it to the higher-level system modules.

Implemented By: OS

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: REALM

7.2.1 Inventory Module (M2)

Secrets: The format and structure of the inventory interface.

Services: Display's a list of available objects and allows for detailed previews of AR Objects and their associated metadata.

Implemented By: REALM

Type of Module: Abstract Object

7.2.2 Touring Module (M3)

Secrets: The format and structure of touring.

Services: Allows *General Users* to go on pre-made and geo-anchored tours by following a defined path that has AR objects placed along the route. Users can view the tours through the Realm Interface and Map views.

Implemented By: REALM

Type of Module: Abstract Object

7.2.3 Tour List Module (M4)

Secrets: The format and structure of the tour list.

Services: Display's a list of available tours and allows a user to view more details about a specific tour.

Implemented By: REALM

Type of Module: Abstract Object

7.2.4 Tour Management Module (M5)

Secrets: The format and structure of tour management.

Services: Display's a list of available tours and allows a user to view more details about a specific tour.

Implemented By: REALM

Type of Module: Abstract Object

7.2.5 Settings Module (M6)

Secrets: The format and structure of settings.

Services: Gives users the option to customize their experience and accommodate disabilities.

Implemented By: REALM

Type of Module: Abstract Object

7.2.6 Maps Module (M7)

Secrets: The format and structure of maps.

Services: Display's AR object locations superimposed on a 2D map of the area around the user. User can zoom in and out to see less or more objects.

Implemented By: REALM

Type of Module: Abstract Object

7.2.7 Realm Interface Module (M8)

Secrets: The structure and interaction of the Realm Interface.

Services: This is the primary interface that the user interacts with to start the placement or interaction with AR Objects.

Implemented By: REALM

Type of Module: Abstract Object

7.2.8 Object Prompt Generation Module (M9)

Secrets: The structure and workflow of object prompt generation.

Services: Allows the user to generate AR objects through a specified prompt.

Implemented By: REALM

Type of Module: Abstract Object

7.2.9 Object Placement Module (M10)

Secrets: The structure and workflow of object placement.

Services: Allows a user to place an AR object in the world. The placement can be fine tuned after the initial location is selected.

Implemented By: REALM

Type of Module: Abstract Object

7.2.10 Object Interaction Module (M11)

Secrets: The structure and workflow of object interaction.

Services: Allows users to interact with existing AR objects in the world. They can add reactions, report, or save the objects to their inventory.

Implemented By: REALM

Type of Module: Abstract Object

7.2.11 Object Render Module (M12)

Secrets: The format and algorithm used to render AR objects.

Services: Balances the performance with the number and resolution of AR objects to ensure a fluid user experience.

Implemented By: REALM

Type of Module: Abstract Object

7.2.12 Collision Detection Module (M13)

Secrets: The format and algorithm to achieve collision detection.

Services: Uses device sensor data to detect when a user is close to a physical object and warn them to prevent a possible collision.

Implemented By: REALM

Type of Module: Abstract Object

7.2.13 Weather Detection Module (M15)

Secrets: The format and algorithm to achieve weather detection.

Services: Display's warnings for inclement weather in the area of the user.

Implemented By: REALM

Type of Module: Abstract Object

7.2.14 Tour Proximity Detection Module (M16)

Secrets: The format and algorithm to achieve tour proximity detection.

Services: Monitors the user's location to find when they are in close proximity to a tour.

Implemented By: REALM

Type of Module: Abstract Object

7.2.15 Popup Module (M17)

Secrets: The structure and format of popups.

Services: Shows a popup on device.

Implemented By: REALM

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: REALM

7.3.1 REST API Connection Module (M18)

Secrets: The structure and format of the connection to the server REST API.

Services: Makes calls to and from the REST API running on the server.

Implemented By: REALM

Type of Module: Abstract Object

7.3.2 Local Database Manager Module (M19)

Secrets: The structure and format of the local database interface.

Services: Interacts with the local database by performing CRUD operations.

Implemented By: REALM

Type of Module: Record

7.3.3 Data Sync Module (M20)

Secrets: The structure and algorithm to sync data.

Services: Keeps the local app data and server data in sync.

Implemented By: REALM

Type of Module: Abstract Object

7.3.4 Server Database Manager Module (M21)

Secrets: The structure and format of the server database interface.

Services: Interacts with the server database by performing CRUD operations.

Implemented By: REALM

Type of Module: Record

7.3.5 Authentication Module (M22)

Secrets: The structure and format of the authentication.

Services: Verifies a user's credentials and which type of user they are (General/Organization user).

Implemented By: REALM

Type of Module: Abstract Object

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**
EI-LF1	M8
MP-FR1	M7
MP-FR2	M7, M18
MP-FR3	M7
MP-FR4	M7
MP-FR6	M7
MP-FR7	M7
MP-FR8	M7
MP-FR9	M7
MP-FR10	M7, M14
IV-FR1	M2, M18
IV-FR2	M2, M18, M9, M11
IV-FR3	M2
IV-FR4	M2
IV-FR5	M2, M18, M9, M11
IV-FR6	M2
IV-FR7	M2
IV-FR8	M2
IV-FR9	M2
IV-FR10	M2
IV-FR11	M2, M6

Table 2: Trace Between Requirements and Modules

Req.	**Modules**
OUI-FR1	M9
OUI-FR2	M9
OUI-FR3	M9
OUI-FR4	M9, M18
OUI-FR5	M9
OUI-FR6	M9
OUI-FR7	M9, M18
OUI-FR8	M9, M18
OUI-FR9	M9, M18
OUI-FR10	M9, M18
OUI-FR11	M9, M18
OUI-FR12	M9
TM-FR1	M5, M22
TM-FR2	M5, M18, M22
TM-FR3	M5, M18
TM-FR4	M5, M22
TM-FR5	M5, M22
TM-FR6	M5, M18, M22
TR-FR1	M3, M22
TR-FR2	M3, M22, M4 M16, M17
TR-FR3	M3
TR-FR4	M3, M7, M8
OP-FR1	M10, M18
OP-FR2	M10, M18
OP-FR3	M10
OP-FR4	M10, M18

Table 3: (Cont.) Trace Between Requirements and Modules

Req.	Modules
RI-FR1	M8, M1
RI-FR3	M8, M10
RI-FR4	M8, M9
RI-FR5	M8, M5
RI-FR6	M8, M15
RI-FR7	M8, M13
RI-FR8	M8, M19, M20
AI-FR1	M22
AI-FR2	M22, M18
PS-FR1	M22, M18
PS-FR2	M22, M18
PS-FR3	M22, M18
PS-FR4	M22, M18
G-FR1	M18
G-FR2	M18
G-FR3	M18
G-FR4	M18
G-FR5	M18
S-FR1	M6, M19
S-FR2	M6, M19
S-FR3	M6, M18
S-FR4	M6, M18
S-FR5	M6, M18
S-FR6	M6, M19
DB-FR1	M19, M21, M20
DB-FR2	M19, M21, M20

Table 4: (Cont.) Trace Between Requirements and Modules

Req.	**Modules**
QS-P1	M7
QS-P2	M2
QS-P3	M12
QS-P4	M9
QS-P5	M12
QS-U1	M6
QS-U2	*All*
QS-SC1	M19, M21, M20
QS-SC2	M19, M21, M20
QS-SA1	M8, M12, M13, M14
QS-SA2	M8, M12, M13, M14
QS-R1	M19, M21
QS-A1	M19, M21
DI-I1	M1
DI-I2	M1
DI-D1	M1
DI-D2	M1
DI-D3	M1
DI-D4	M1
DI-M1	M18
DI-R1	M2, M3, M5, M6, M8, M7, M9, M10, M11
DI-P1	M1
DI-P2	M1

Table 5: (Cont.) Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M20, M18
AC3	M6
AC4	M5
AC**??**	M1, M9
AC6	M1
AC7	M7
AC8	M12
AC9	M12
AC10	M2, M3, M5, M6, M8, M7, M9, M10, M11

Table 6: 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.

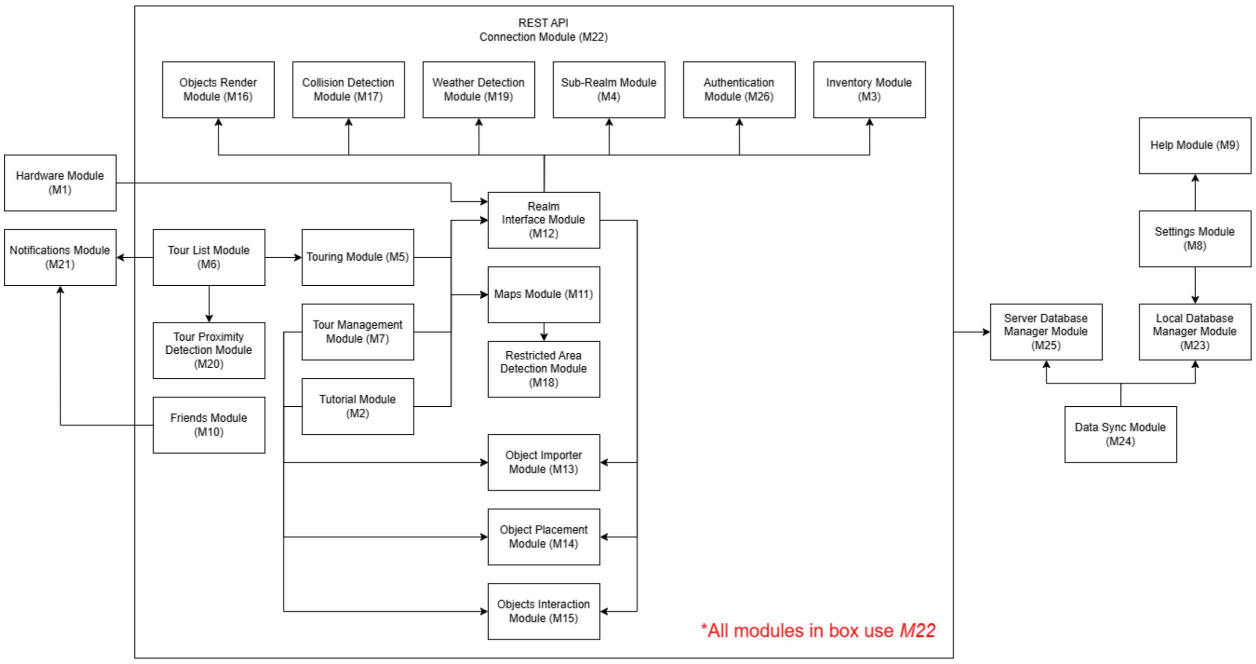
REST API
Connection Module (M22)
Objects Render Module (M16)
Collision Detection Module (M17)
Weather Detection Module (M19)
Sub-Realm Module (M4)
Authentication Module (M26)
Inventory Module (M3)
Hardware Module (M1)
Realm Interface Module (M12)
Help Module (M9)
Notifications Module (M21)
Tour List Module (M6)
Touring Module (M5)
Settings Module (M8)
Maps Module (M11)
Tour Proximity Detection Module (M20)
Tour Management Module (M7)
Restricted Area Detection Module (M18)
Server Database Manager Module (M25)
Local Database Manager Module (M23)
Tutorial Module (M2)
Friends Module (M10)
Data Sync Module (M24)
Object Importer Module (M13)
Object Placement Module (M14)
Objects Interaction Module (M15)
*All modules in box use M22

Figure 1: Use hierarchy among modules

10 User Interfaces

The following are images of various major interfaces:

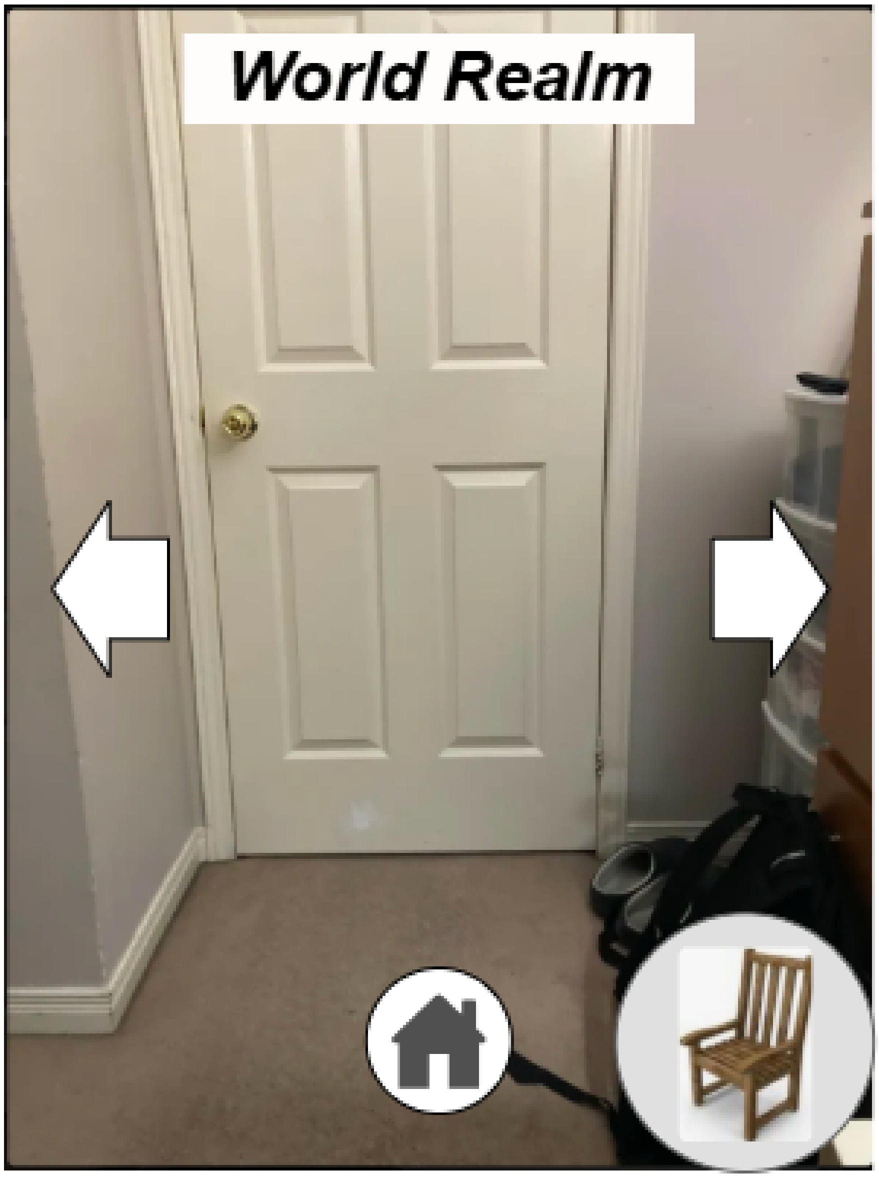

Figure 2: Realm Interface Screen

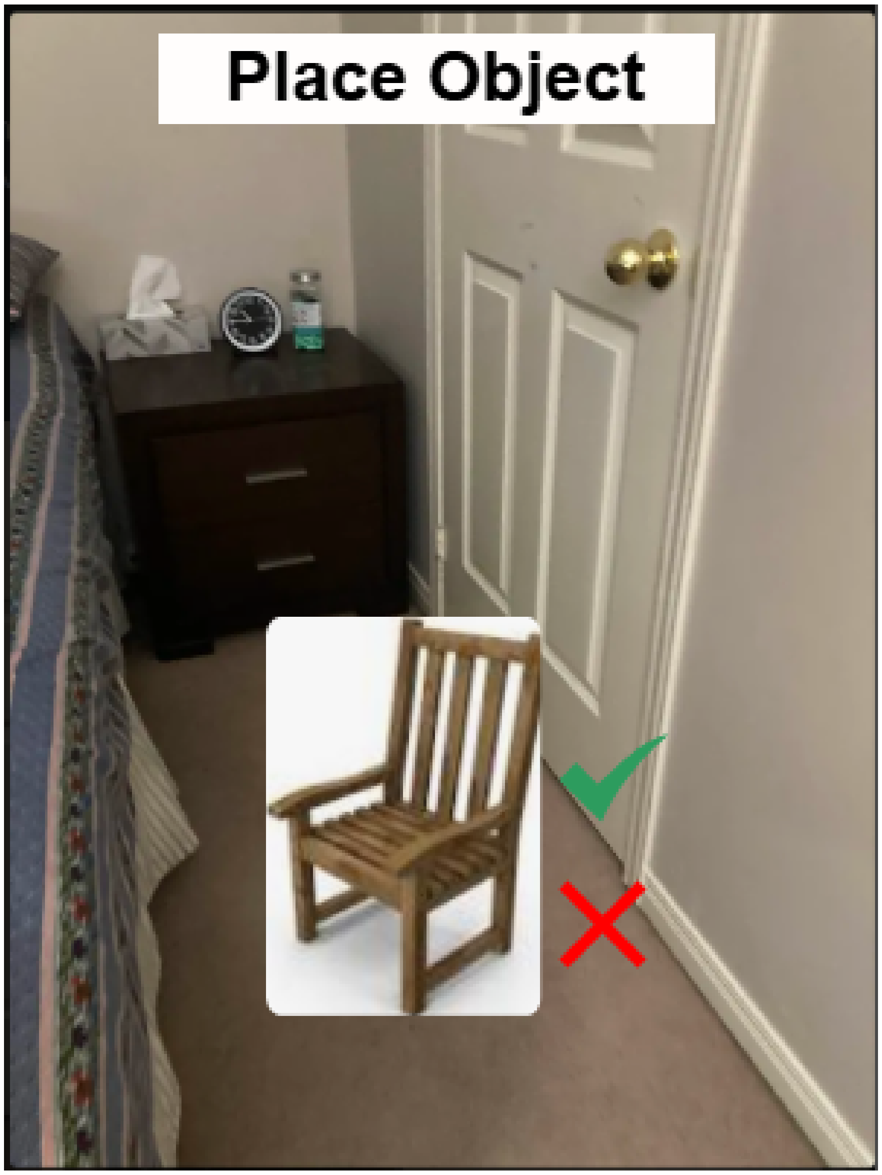

Figure 3: Object Placement Screen

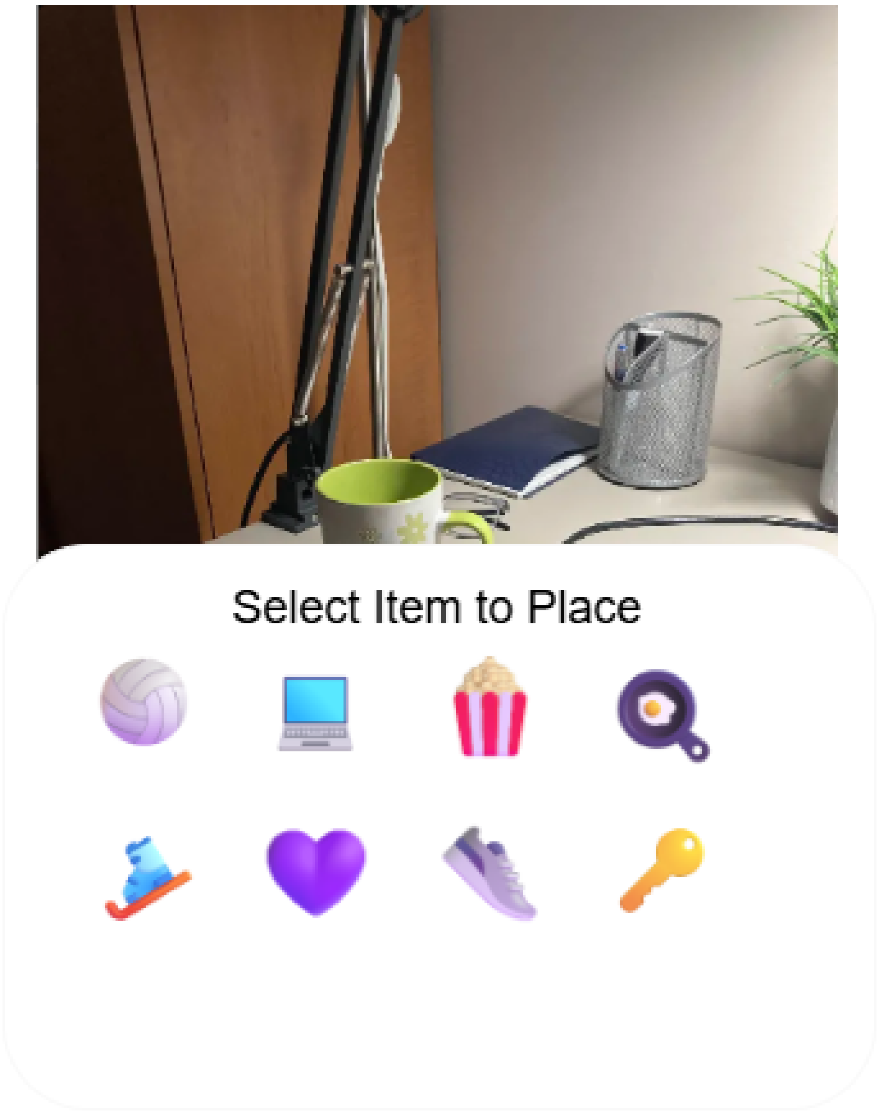

Figure 4: Inventory Screen

11 Design of Communication Protocols

N/A

12 Timeline

Below is a schedule for our implementation and work distribution of the modules.

Due Date	**Work to be Completed**	**Assigned Member**
Jan 15 - Jan 20	Hardware Module	Avanish
	Inventory Module	Avanish
	Touring Module	Russell
	Tour List Module	Russell
	Tour Management Module	Russell
Jan 25 - Feb 1	Settings Module	Rafey
	Maps Module	Abdul
Jan 30 - Feb 5	Realm Interface Module	Russell
	Object Prompt Generation Module	Abdul
	Object Placement Module	Avanish
	Object Interaction Module	Abdul
	Object Render Module	Abdul
Feb 5 - Feb 10	Collision Hazard Detection Module	Rafey
	Restricted Area Detection Module	Avanish
	Weather Hazard Detection Module	Avanish
	Tour Proximity Detection Module	Rafey
Feb 6 - Feb 10	Popup Module	Rafey
	REST API Module	Abdul
	Local Database Manager Module	Abdul
	Data Sync Module	Russell
Feb 8 - Feb 10	Server Database Manager Module	Abdul
	Authentication Module	Rafey

Table 7: Module Development Schedule

References