Smart Home Remote Control

Software Design Specification

2022.05.15.

Introduction to Software Engineering

TEAM 12 (스마트 리모컨)

Team Leader	조윤근
Team Member	금상인
Team Member	전종문
Team Member	이대희
Team Member	Stefan Hofmann

1. Preface **9**

1.1. Readership 9

1.2. Scope 9

1.3. Objective 9

1.4. Document Structure 10

2. Introduction **10**

2.1 Objectives 11

2.2 Applied Diagrams 11

2.2.1 UML 11

2.2.2 Use case Diagram 11

2.2.3 Sequence Diagram 11

2.2.4 Class Diagram 12

2.2.5 Context Diagram 12

2.2.6 Entity Relationship Diagram 12

2.3 Applied Tools 13

2.3.1 ERD Plus 13

2.3.2 Microsoft PowerPoint 13

2.4 Project Scope 13

3. System Architecture - Overall **14**

3.1. Objectives 14

3.2. System Organization 14

3.2.1 Context Diagram 15

3.2.2. Use Case Diagram 16

4. System Architecture - Frontend **17**

4.1. Objectives 17

4.2. Subcomponents 17

4.2.1. User 17

4.2.2. Appliance 19

4.2.3. Control 21

5. System Architecture - Backend **25**

5.1. Objectives 25

5.2. Overall Architecture 25

5.3. Subcomponents 26

5.3.1. Webserver functions 26

5.3.2. Wired Appliances System 28
5.3.3. Wireless Smart Appliances System 29
5.3.4. Wireless Non-Smart Appliances System 32
5.4. Hardware architecture 34
6. Protocol Design **35**
6.1. Objectives 35
6.2. JSON 35
6.3. OAuth 35
6.4. Authentication 35
6.4.1. Register 35
6.4.2. Log-In 37
6.5. Appliances 38
6.5.1. Set Appliances List 38
6.5.2. Get Appliances List 39
6.6. Get Control of Appliances 40
6.7. Search Appliances 41
6.8. Macro list 42
6.8.1. Set Marco List 42
6.8.2. Get Macro List 42
6.9. Macro 43
6.9.1. Set Marco 43
6.9.2. Get Macro List 45
6.10. Control 46
6.10.1. Single Control command 46
6.10.2. Macro command 46
7. Database Design **48**
7.1. Objectives 48

7.2. ER Diagram 48

7.2.1 Entities 50

7.3. Relational Schema 57

7.4. SQL DDL 58

7.4.1 User 58

7.4.2 Log_list 58

7.4.3 Log 59

7.4.4 Macro_list 59

7.4.5 Macro 59

7.4.6 Command 60

7.4.7 Application_list 60

7.4.8 Appliance 60

7.4.9 Interface 61

8. Testing Plan 62

8.1. Objectives 62

8.2. Testing Policy 62

8.2.1. Development Testing 62

8.2.2. Release Testing 63

8.2.3. User Testing 64

8.2.4. Testing Case 64

9. Development Plan 65

9.1. Objectives 65

9.2. Frontend Environment 65

9.2.1. Adobe Photoshop (UI/UX Design) 65

9.2.2. Adobe Xd (UI/UX Design) 65

9.2.3. Kakao Oven (UI/UX Design) 66

9.2.4. Android Studio (Application) 66

9.3. Backend Environment 67
9.3.1. Github (Open source) 67
9.3.2. Firebase (DBMS) 67
9.3.3. SQLite Database (DBMS) 68
9.3.4. Android Studio (Application) 68
9.3.5. Node.js (Server) 69
9.3.6. AWS EC2 (Server) 69
9.4. Constraints 70
9.5. Assumptions and Dependencies 70
10. Supporting Information **71**
10.1. Software Design Specification 71
10.2. Document History 71

LIST OF FIGURES

[Figure 1] Overall context diagram 15
[Figure 2] Use case diagram 16
[Figure 3] Overall architecture 25
[Figure 4] Webserver Function 26
[Figure 5] Class diagram - Wired Appliances System 28
[Figure 6] Sequence diagram - Wired Appliances System 29
[Figure 7] Class diagram - Wireless Smart Appliances System 30
[Figure 8] Sequence diagram - Wireless Smart Appliances system 31
[Figure 9] Class diagram - Wireless Non-Smart Appliances System 32
[Figure 10] Sequence diagram - Wireless Non-Smart Appliances System 33
[Figure 11] ER-Diagram 49
[Figure 12] ER diagram, Entity, User 50
[Figure 13] ER diagram, Weak Entity, Log list 51
[Figure 14] ER diagram, Weak Entity, Log 52
[Figure 15] ER diagram, Weak Entity, Appliance list 53
[Figure 16] ER diagram, Entity, Appliance 54

[Figure 17] ER diagram, Entity, Interface..... 54
[Figure 18] ER diagram, Entity, Macro list..... 55
[Figure 19] ER diagram, Entity, Macro..... 56
[Figure 20] Relational Schema..... 57
[Figure 21] Software Release Life Cycle..... 64
[Figure 22] Adobe Photoshop logo..... 65
[Figure 23] Adobe Xd logo..... 65
[Figure 24] Adobe Xd logo..... 66
[Figure 25] Android Studio logo..... 66
[Figure 26] Github logo..... 67
[Figure 27] Firebase logo..... 67
[Figure 28] SQLite Database logo..... 68
[Figure 29] Android Studio logo..... 68
[Figure 30] Nodejs logo..... 69
[Figure 31] AWS logo..... 69
[Figure 32] Document History..... 71

LIST OF TABLES

[Table 1] register request 36

[Table 2] register response 36

[Table 3] Log-in request 37

[Table 4] Log-in response 37

[Table 5] Set Appliances List request 38

[Table 6] Set Appliances List response 38

[Table 9] Get Control of Appliances request 40

[Table 10] Get Control of Appliances response 40

[Table 11] Search Appliances request 41

[Table 12] Search Appliances request 41

[Table 13] Set Macro List response 43

[Table 14] Get Macro List request 45

[Table 15] Get Macro List response 45

[Table 16] Single Control command request 46

[Table 17] Single Control command response 46

[Table 18] Macro command request 46

[Table 19] Macro command response 47

1. Preface

This chapter contains the readership information, readership, scope, objective of this document and the document structure of this Software Design Document for “Smart Remote Control”.

1.1. Readership

This Software Design Document is divided into 10 sections with various subsections. Each section contains the overall structure of the system, the structure at the front end, the structure at the back end, protocols, database design and testing plan, and development plan. The detailed structure of the Software Design Document can be found as listed below, in the Document Structure subsection of this SDD. In this document, Team 12 is the main reader. Additionally, professors, TAs, and team members in the Introduction to Software Engineering class can be the main readers.

1.2. Scope

This Design Specification is to be used by Software Engineering and Software Quality Engineering as a definition of the design to be used to implement an application that users can find friends for the same purpose at close distance.

1.3. Objective

The primary purpose of this Software Design Document is to provide a description of the technical design aspects for our mobile phone recommendation application, " Smart Remote Control ". This document describes the software architecture and software design decisions for the implementation of " Smart Remote Control ". It also provides an architectural overview of the system to depict different aspects of the system. It further specifies the structure and design of some of the modules discussed in the SRS document and in addition, displays some of the use cases that have been transformed into sequential and activity diagrams, including the class diagrams which show how the programming team would implement the specific module. The

intended audience of this document is, but not limited to, the stakeholders, developers, designers, and software testers of the " Smart Remote Control "mobile application.

1.4. Document Structure

- **1. Preface:** this chapter describes readership, scope of this document, object of this system, and structure of this document
- **2. Introduction:** this chapter describes several tools used for this document, several diagrams used in this document and the references, and object of this project.
- **3. Overall System Architecture:** this chapter describes the overall architecture of the system using context diagram, sequence diagram, and use case diagram.
- **4. System Architecture - Frontend:** this chapter describes architecture of the frontend system using class diagram and sequence diagram.
- **5. System Architecture - Backend:** this chapter describes architecture of the backend system using class diagram and sequence diagram.
- **6. Protocol Design:** this chapter describes design of several protocols which are used for communication between client and server.
- **7. Database Design:** this chapter describes database design using several ER diagrams and SQL DDL.
- **8. Testing Plan:** this chapter describes the testing plan for our system.
- **9. Development Plan:** this chapter describes which tools to use to develop the system, constraints, assumption, and dependencies for developing this system.
- **10. Supporting Information:** this chapter describes the baseline of this document and history of this document.

2. Introduction

The purpose of this project is to create a mobile application that is to provide both Android and iOS users full access to home appliances. The system provides control over home appliances using Rasberry Pi and computer networking. Users can connect whatever home appliances they want via Android and iOS and control with their phone.

This document is a design document and contains information about the design used to implement the project. For design details, refer to the description of the requirements specification.

2.1 Objectives

This part explains the diagrams and tools applied to the project.

2.2 Applied Diagrams

2.2.1 UML

UML is a standardized modeling language to facilitate communication between developers in the process of system development such as requirements analysis, system design, and system implementation. UML has the advantage of being a language with a strong expressive power for modeling and a logical notation with relatively few contradictions. Therefore, communication between developers is easy, and it is easy to point out modeling structures that are omitted or inconsistent and can be applied to all systems regardless of the scale of the system to be developed. UML provides a wealth of analysis and design devices for developing object-oriented software based on diagrams such as use case diagrams and class diagrams, so it is expected to be used as an industry standard for a considerable period in the future.

2.2.2 Use case Diagram

The use case of UML describes the functional unit provided by the system. The main purpose of the Use Case Diagram is for development teams to visualize the functional requirements of the system, including relationships between different use cases as well as the relationships between the system and interacting actors for the main process. Use Case Diagram is used to describe the advanced functions of the system and the scope of the system.

2.2.3 Sequence Diagram

UML's sequence diagram shows a detailed flow of a specific use case or even a part of a specific use case. Sequence Diagram shows the calling relationship between different objects in the sequence, and it can also show different calls to different objects in detail. The sequence diagram is expressed in two dimensions, the vertical shows the message/call sequence in the order of

occurrence time, and the horizontal shows the object instance to which the message is transmitted. The sequence diagram is very simple, and class instances (objects) are classified by placing an instance of each class in a box at the top of the diagram.

2.2.4 Class Diagram

UML's class diagram represents how different entities (people, products, data) relate to each other. In other words, it can be said to be the static structure of the system. Class diagrams are mainly used to show implementation classes handled by programmers, and implementation class diagrams show classes like logical class diagrams.

2.2.5 Context Diagram

Context diagram in UML is a diagram that defines the boundary between the system, or part of a system, and its environment, showing the entities that interact with it. Context diagram represents a central system without any details of the internal structure surrounded by all the systems, environments, and activities with which it interacts. The purpose of the context diagram is to focus on external factors and events that must be considered when developing overall system requirements and constraints. Best context diagrams are used to show how systems interoperate at a very high level, or how systems work and interact logically. Context diagrams are the tools you need to develop basic interactions between systems and actors.

2.2.6 Entity Relationship Diagram

When modeling the database structure of a system, the entity relationship diagram is a diagram showing the properties of entities that have unique characteristics that constitute them and the relationships between them in a network-type structure. In the entity relationship diagram, the entity set is represented by a rectangle, the attribute is represented by an ellipse, the entity set and its attributes are connected by a line, the relationship set is represented by a rhombus, and the mapping form of the relationship set is represented by an arrow.

2.3 Applied Tools

2.3.1 ERD Plus

ERD Plus is a database modeling tool for creating Entity Relationship Diagrams, Relational Schemas, Star Schemas, and SQL DDL statements. ERD Plus is a web-based database modeling tool that lets you quickly and easily create Entity Relationship Diagrams (ERDs), Relational Schemas (Relational Diagrams), Star Schemas (Dimensional Models). ERD Plus enables drawing standard ERD components (Entities, Attributes, Relationships).

2.3.2 Microsoft PowerPoint

The PowerPoint gives you access to the familiar slideshow maker tool you already know. Create, edit, view, present, or share presentations quickly and easily. PowerPoint provides a quick view of your recent slides and presentations for easy access on any device. PowerPoint lets you make a lasting impression with powerful and customizable slides and presentations that make you stand out. PowerPoint gives you templates to work from and automatic design ideas for your presentations.

PowerPoint is widely used as presentation software by a variety of users such as businessmen, teachers, and students, and is one of the most appropriate ways to use it in the form of persuasion skills. Microsoft Office PowerPoint, part of Microsoft Office, has become the most widely used presentation program in the world.

2.4 Project Scope

Through this project, users will have an opportunity to easily connect and control home appliances. With continuous updates, we will have a database server that conatin home appliances usage patterns for hundreds or thousands of users. By doing that user can experience more optimized use of their home appliances and electricity usage.

3. System Architecture - Overall

3.1. Objectives

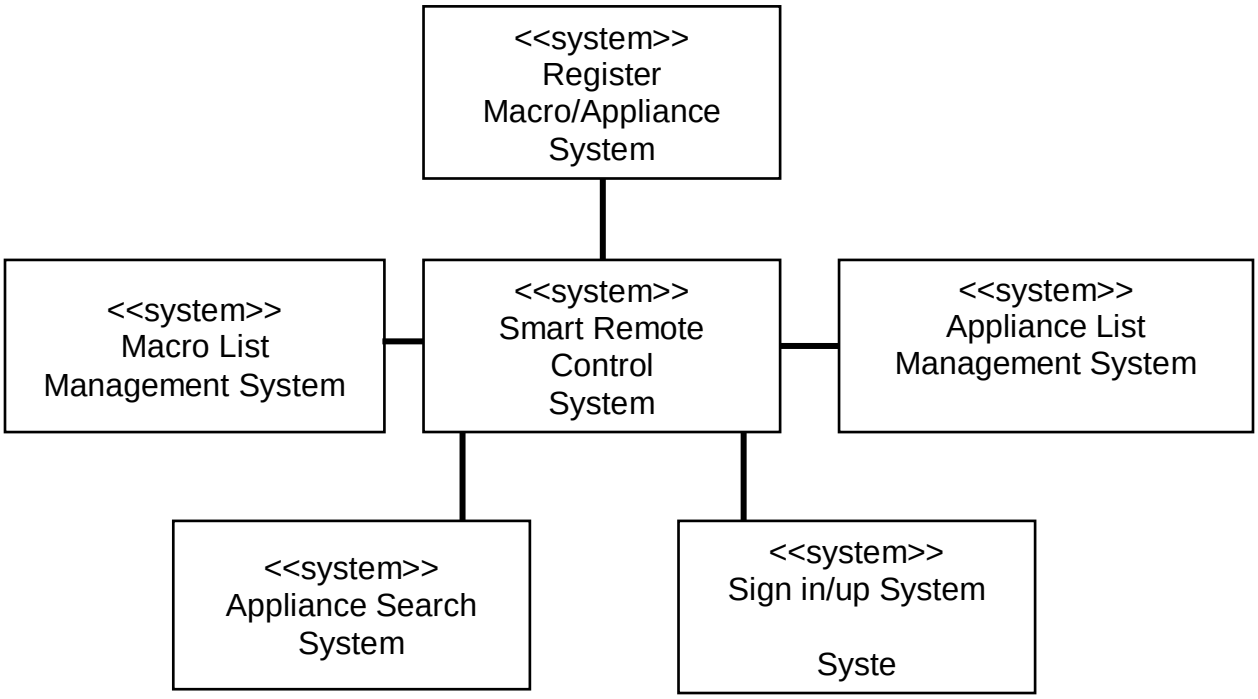
This chapter describes the overall system architecture with organization of the frontend and backend design of the system.

3.2. System Organization

This service is designed by applying the Model-View-Controller pattern. The frontend handles parts of interacting with the user in the view, such as signing in, signing up, connect to new appliances, control existing appliances. In the backend, the user's interaction information is delivered to the controller, and requested data such as the user information, appliances information, macro control information are delivered from the database, and the required result is delivered to the user with the view. For the communication between frontend and backend, we use HTTP communication with JSON format

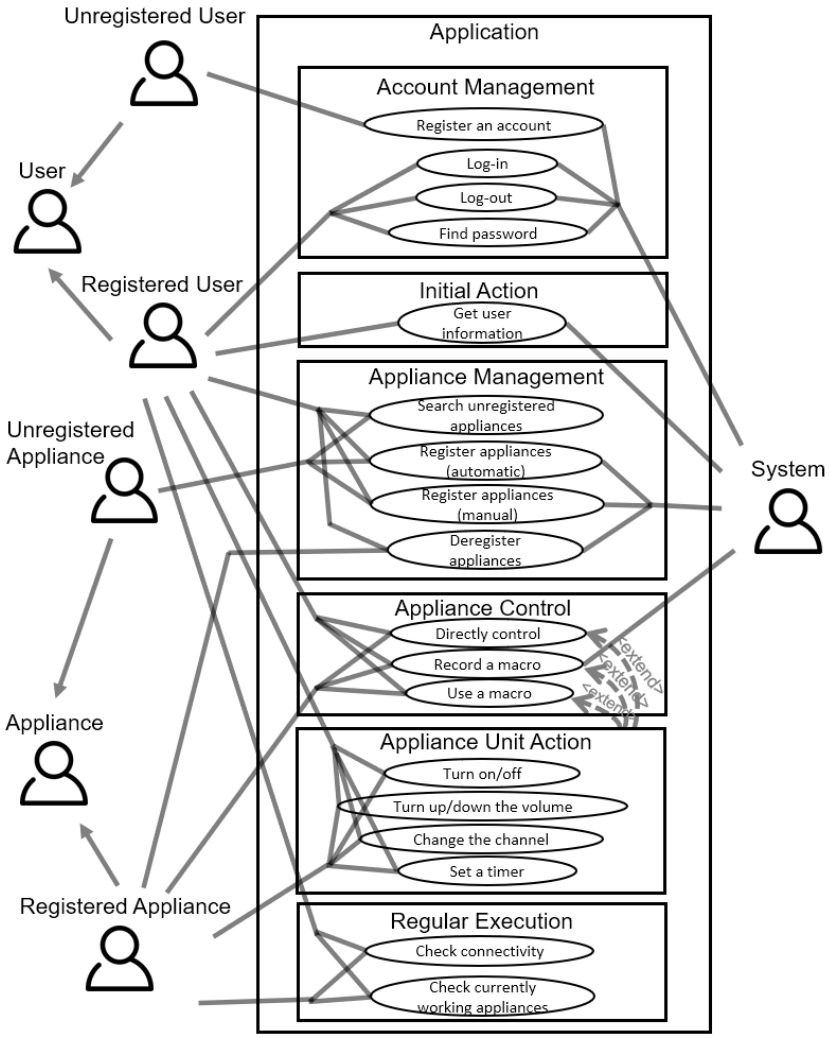
3.2.1 Context Diagram

[Figure 1] Overall context diagram



3.2.2. Use Case Diagram

[Figure 2] Use case diagram



4. System Architecture - Frontend

4.1. Objectives

This chapter describes the architecture of the frontend system, including the attributes and methods of each subcomponent and the relationship between components.

4.2. Subcomponents

4.2.1. User

The User class is a class that handles user's information. When a user registers, the initial user information is set, and user can access main page with using their registered user's information.

4.2.1.1. Attributes

These are the attributes that the User class has.

- **user_id** : id(name) of the user.
- **information** : information of the user.

These are the attributes that the information object has.

- **user_id** : id(name) of the user.
- **email** : email address of the user.
- **password** : password of the user.

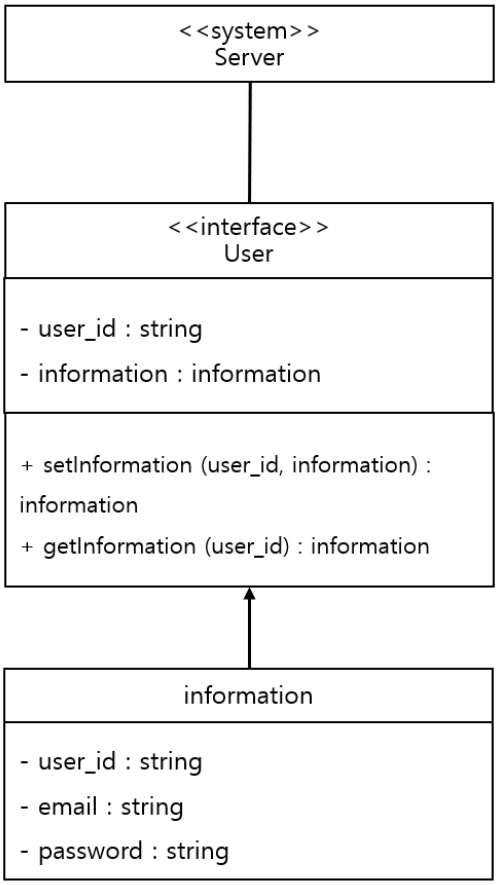
4.2.1.2 Methods

These are the methods that the User class has.

- setInformation()
- getInformation()

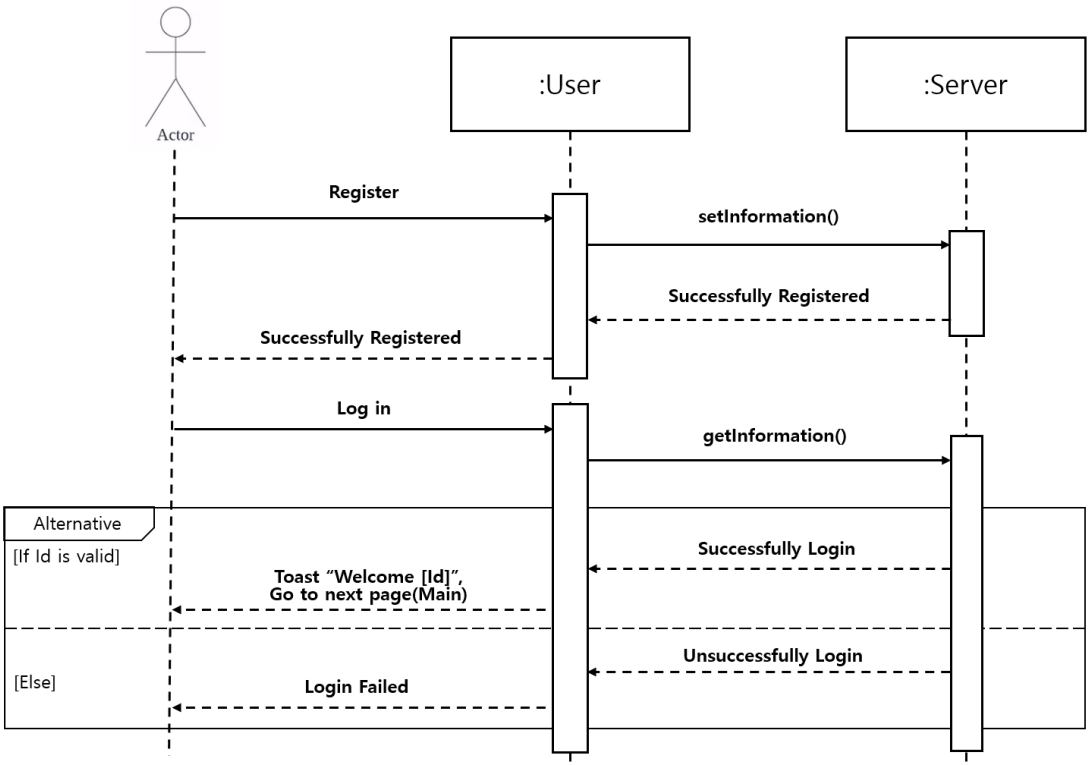
4.2.1.3 Class Diagram

[Figure 3] Class diagram – User



4.2.1.4 Sequence Diagram

[Figure 4] Sequence diagram – User



4.2.2. Appliance

The Appliance class is a class that handles appliance's information. When a user registers, the initial appliance information is set, and appliance appear on the appliance list.

4.2.2.1. Attributes

These are the attributes that the Appliance class has.

- **appliance_id** : id(name) of the appliance.
- **status** : status of appliance.

These are the attributes that the status object has.

- **appliance_id** : id(name) of the appliance.
- **on** : status of whether the appliance is on or off.
- **option** : special condition of appliance.

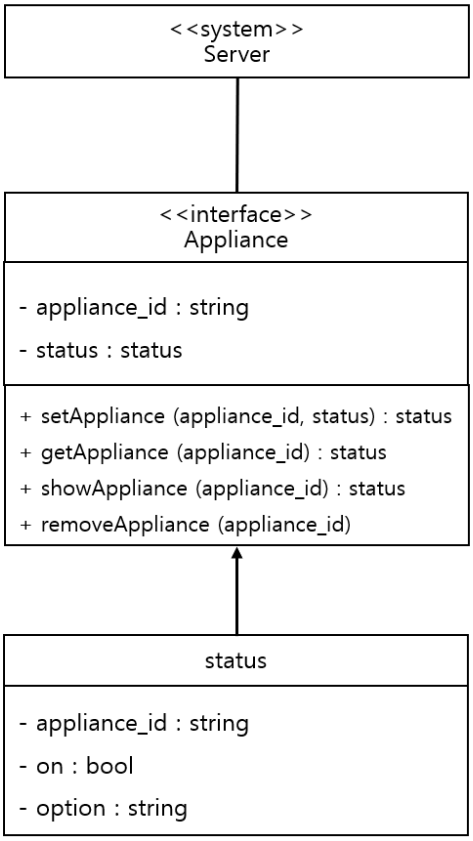
4.2.2.2 Methods

These are the methods that the Appliance class has.

- setAppliance()
- getAppliance()
- showAppliance()
- removeAppliance()

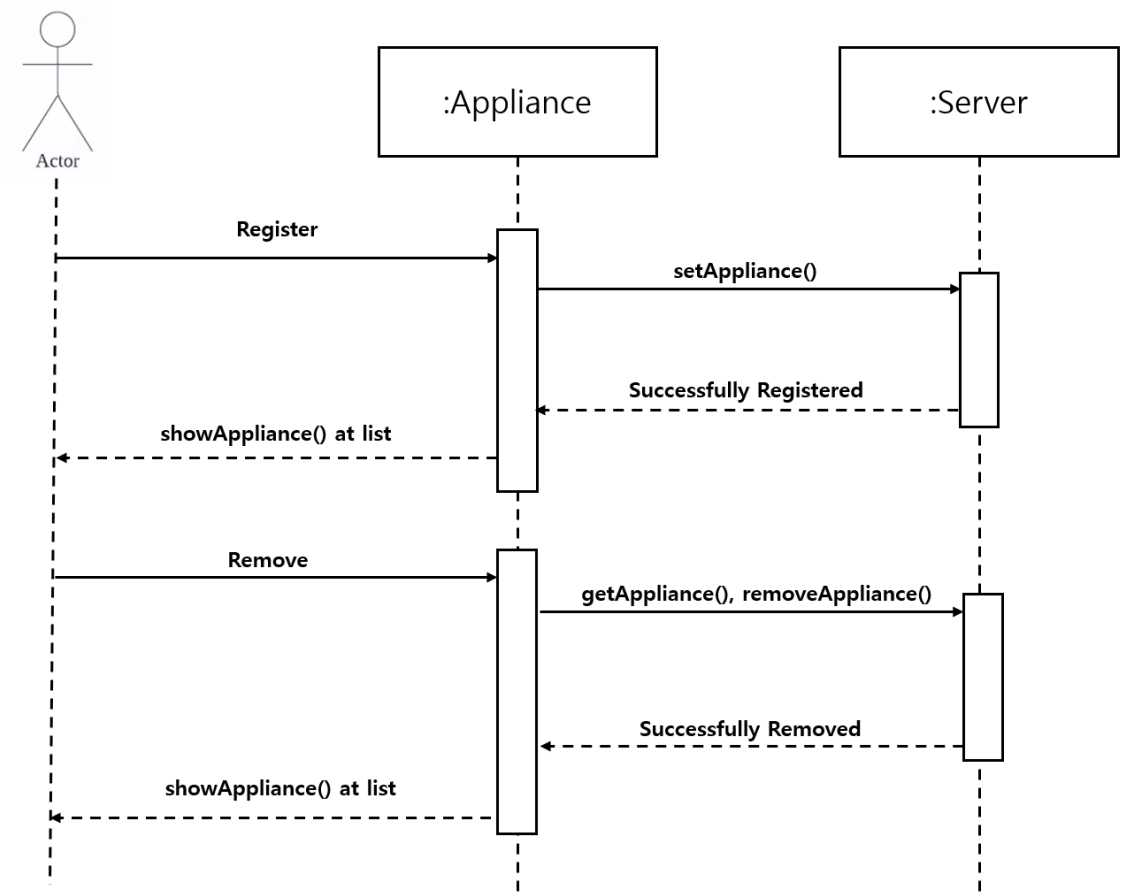
4.2.2.3 Class Diagram

[Figure 5] Class diagram – Appliance



4.2.2.4 Sequence Diagram

[Figure 6] Sequence diagram – Appliance



4.2.3. Control

The Control class receives information of the control information of appliance and can show list of controls that combined by user.

4.2.3.1. Attributes

These are the attributes that the Control class has.

- **appliance_id** : id(name) of appliance.
- **controller** : information of control per each key.
- **controlList_id** : id of control list that combined by user.
- **control_list** : list of control that combined by user.

These are the attributes that the controller object has.

- **appliance_id** : id(name) of the appliance.
- **on** : status of whether the control is on or off.
- **upDown** : status of the control up or down.
- **option** : special control option of appliance.

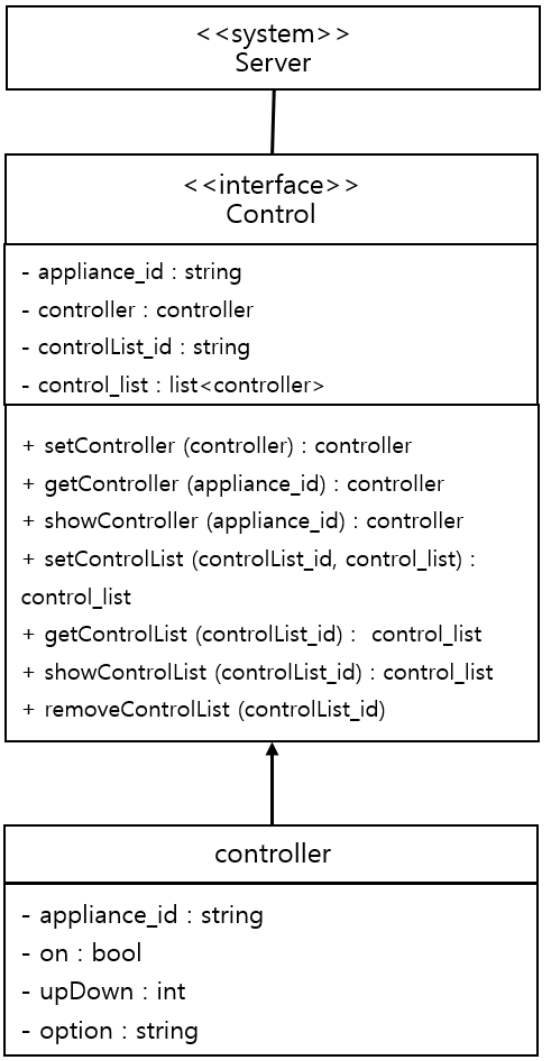
4.2.3.2 Methods

These are the methods that the Control class has.

- setController()
- getController()
- showController()
- setControlList()
- getControlList()
- showControlList()
- removeControlList()

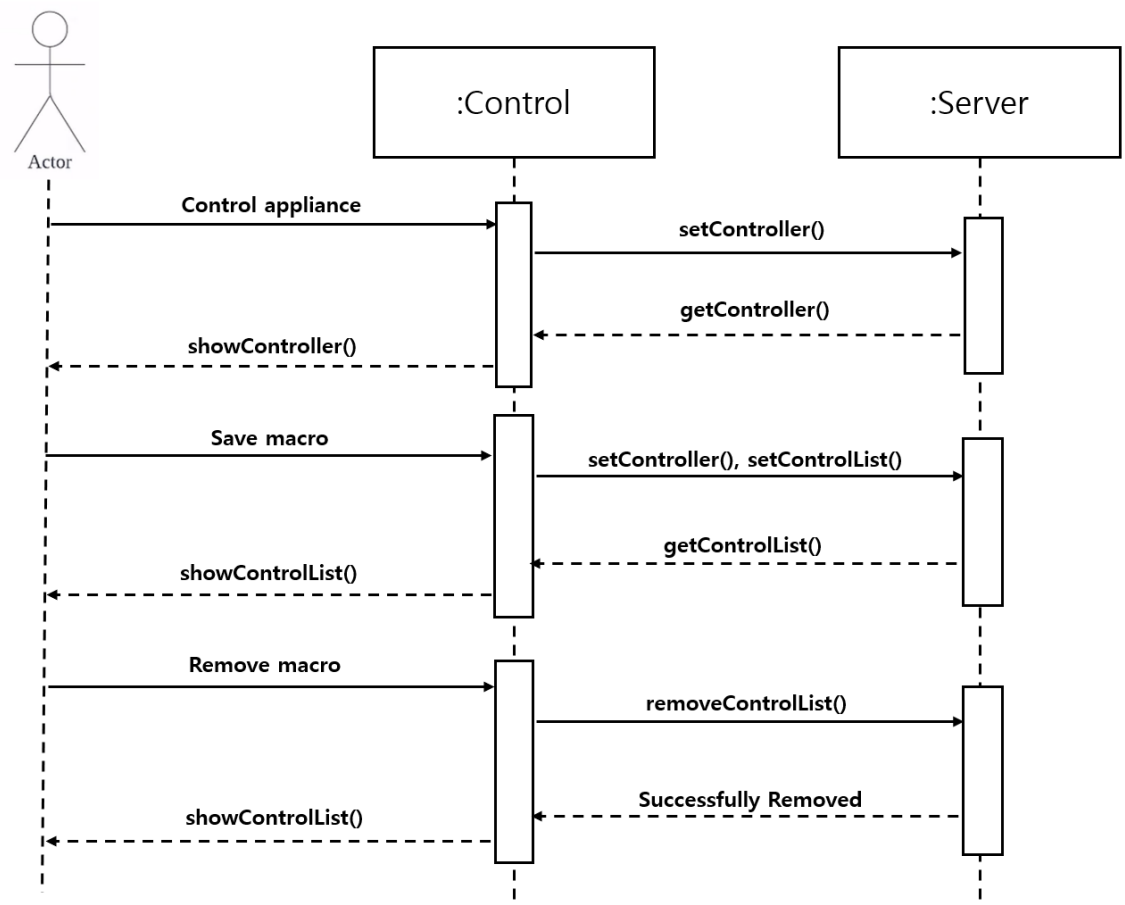
4.2.3.3 Class Diagram

[Figure 7] Class diagram – Control



4.2.3.4 Sequence Diagram

[Figure 8] Sequence diagram – Control



5. System Architecture - Backend

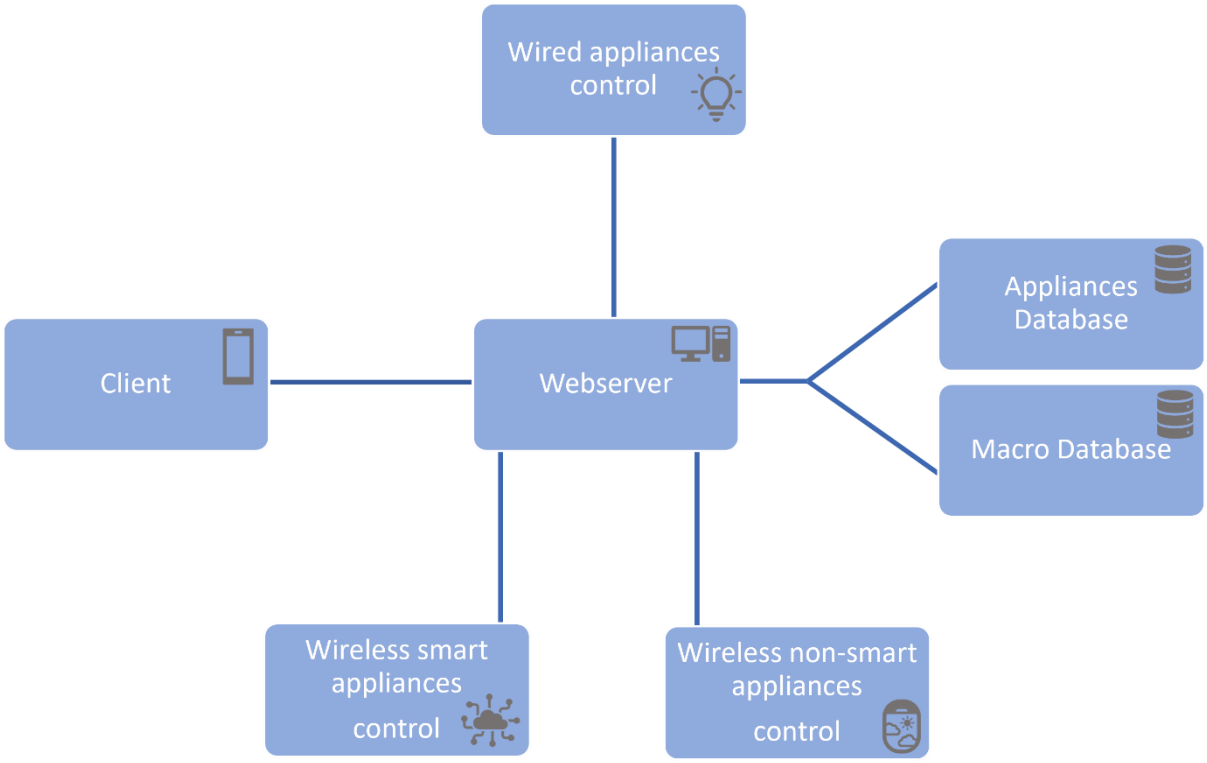
5.1. Objectives

This chapter describes the structure of the back-end system including the DB and the Webserver.

5.2. Overall Architecture

The overall architecture of the system is as above. The Webserver (e.g. Raspberry Pi) receives API calls from the front-end (e.g. client). The actions from the received API calls will be executed. For this purpose, different scripts are executed that control wired appliances, appliances upgraded with the Wi-Fi chip, and appliances that are already smart-home capable. In addition, the database is updated.

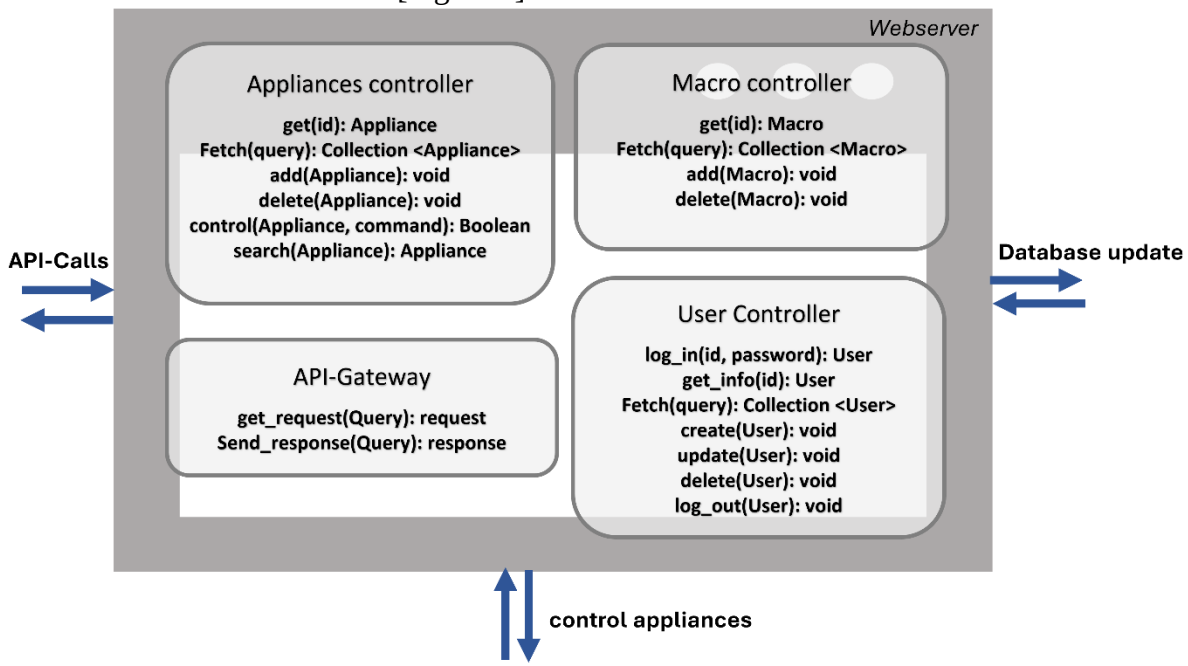
[Figure 3] Overall architecture



5.3. Subcomponents

5.3.1. Webserver functions

[Figure 4] Webserver Function



5.3.1.1. User Controller Class

This class operates all the commands concerning the user including creating, updating, deleting, or getting user information out of the database. All information regarding the user will be saved in the table “user” inside the database.

5.3.1.2. Appliances Controller Class

This class contains all the functions for appliances. Appliances can either be added, deleted, or searched inside the table “appliances” from the database. Moreover, the function *control* will control appliances and can turn then on and several other options.

5.3.1.3. Macro Controller Class

The Macro Controller Class controls all the macros. New Macros can be added, and existing macros can be deleted. Macros will be stored in the table “macros” in the database. Appliances can only be added if they are found in the table “appliances” within the database.

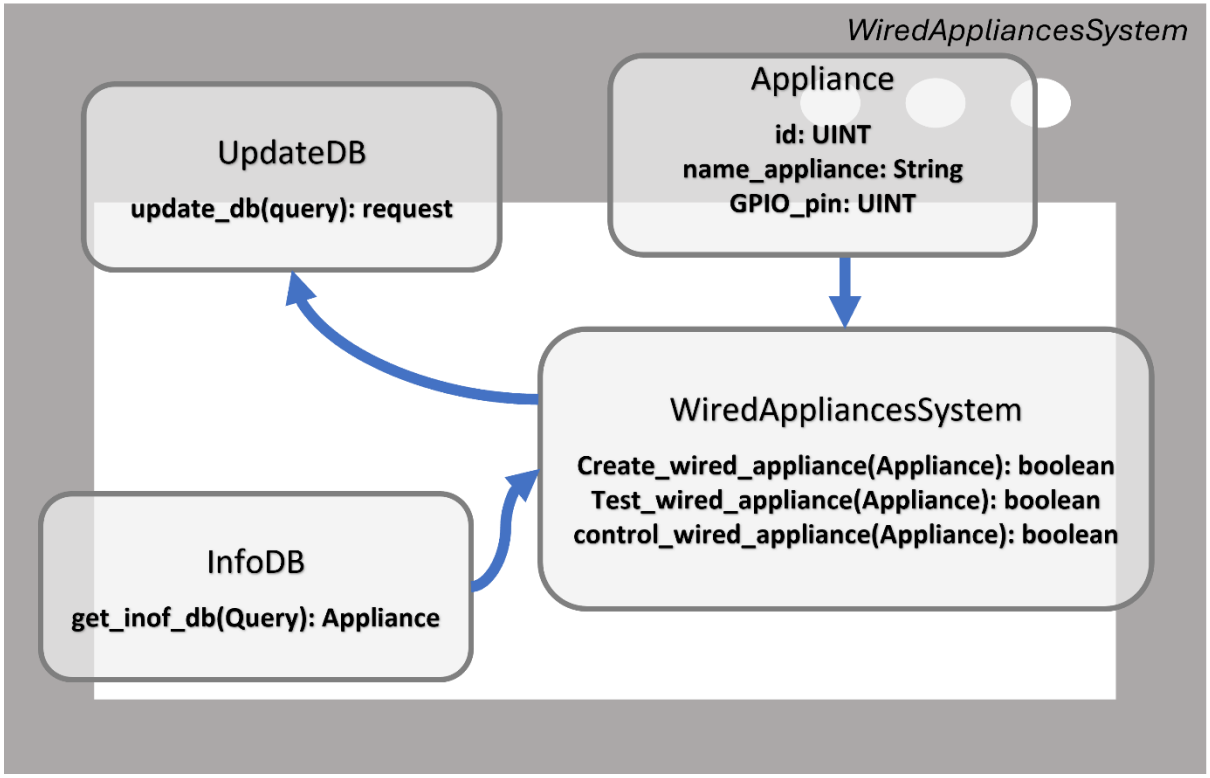
5.3.1.4. API Controller Class

This class distributes requests from the frontend to the appropriate controller (appliances controller, user controller or macro controller) and sends the responses back to the frontend.

5.3.2. Wired Appliances System

5.3.2.1. Class Diagram

[Figure 5] Class diagram - Wired Appliances System

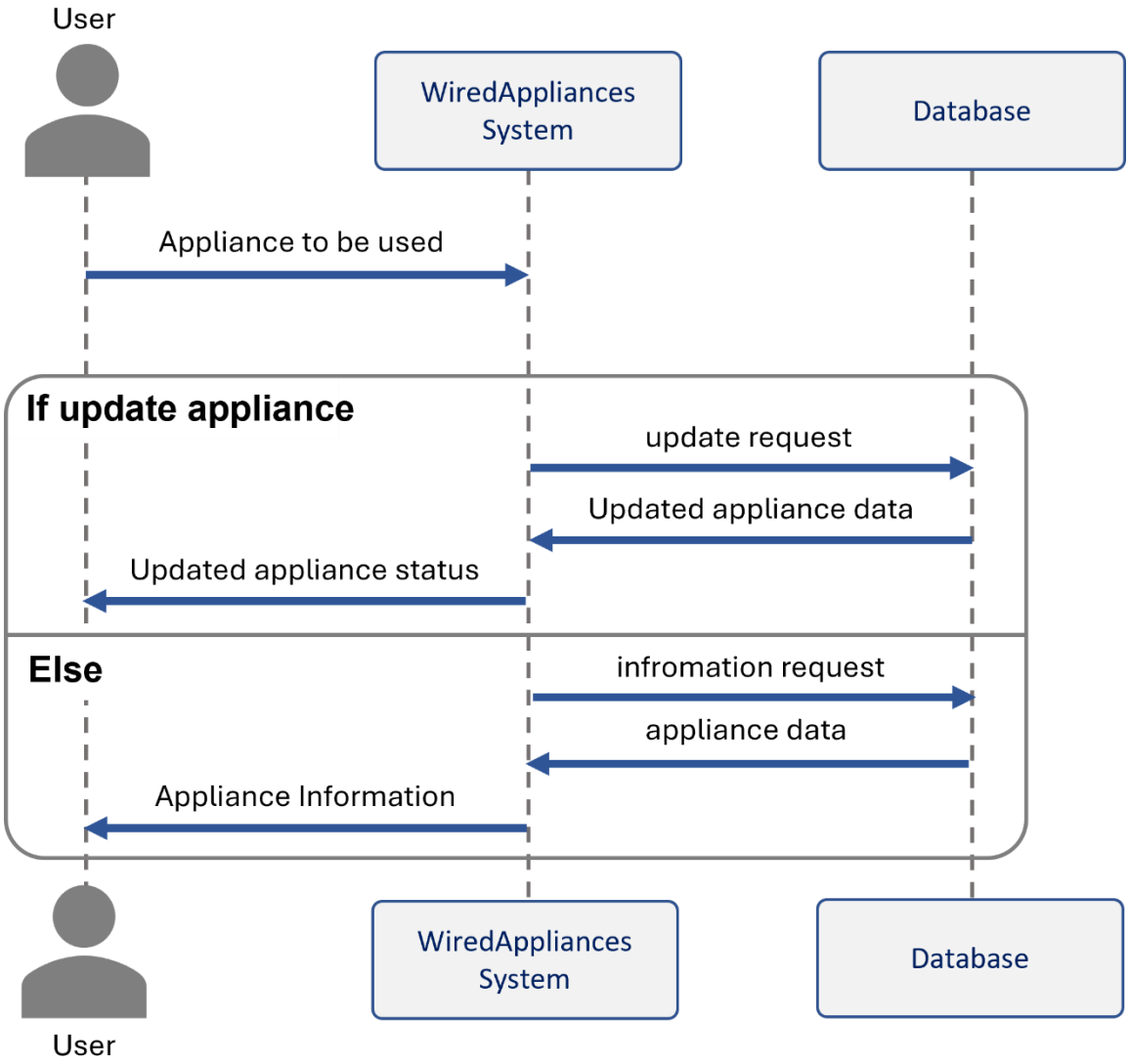


- Class description

  - ✓ **Wired Appliances Class:** For creating, testing, or controlling a wired appliance you need an instance of the class appliance with an id, a name, and a GPIO-Pin number. The GPIO number is necessary, so the Raspberry Pi can control the correct output-pin. The functions of WiredAppliancesSystem need access to the database to update or get data.

5.3.2.2. Sequence Diagram

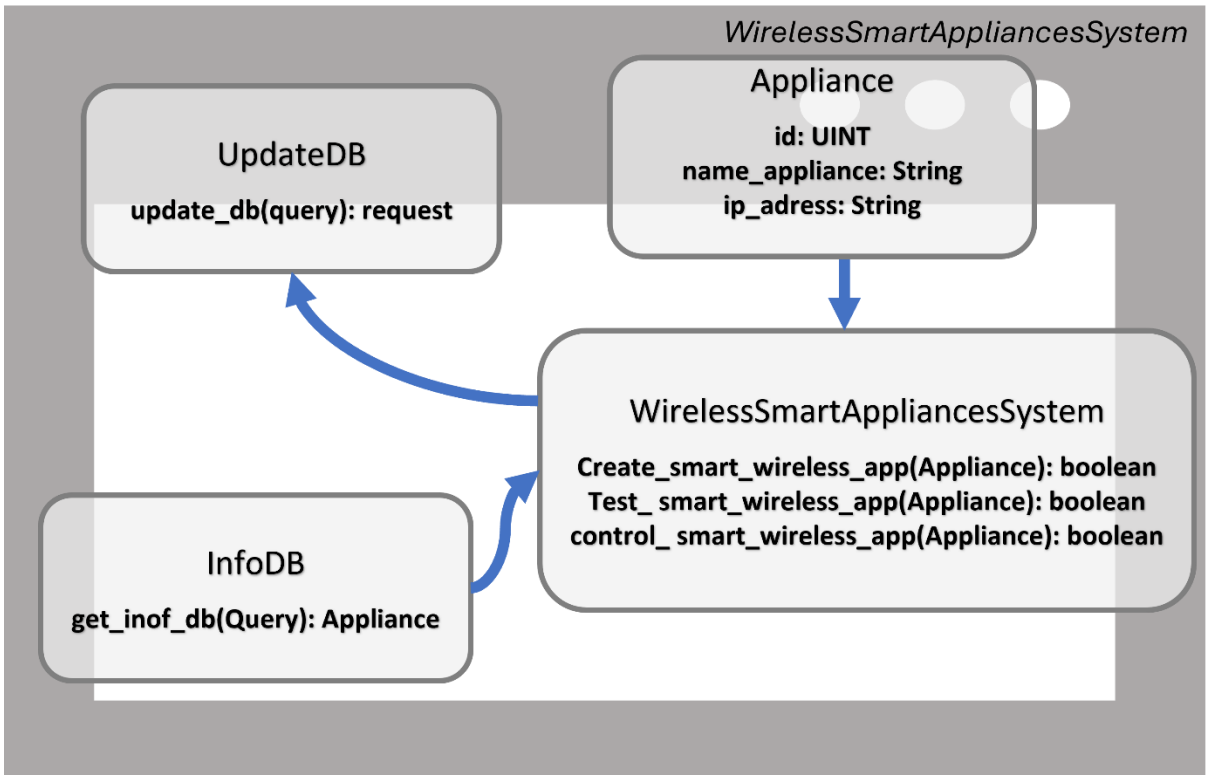
[Figure 6] Sequence diagram - Wired Appliances System



5.3.3. Wireless Smart Appliances System

5.3.3.1. Class Diagram

[Figure 7] Class diagram - Wireless Smart Appliances System

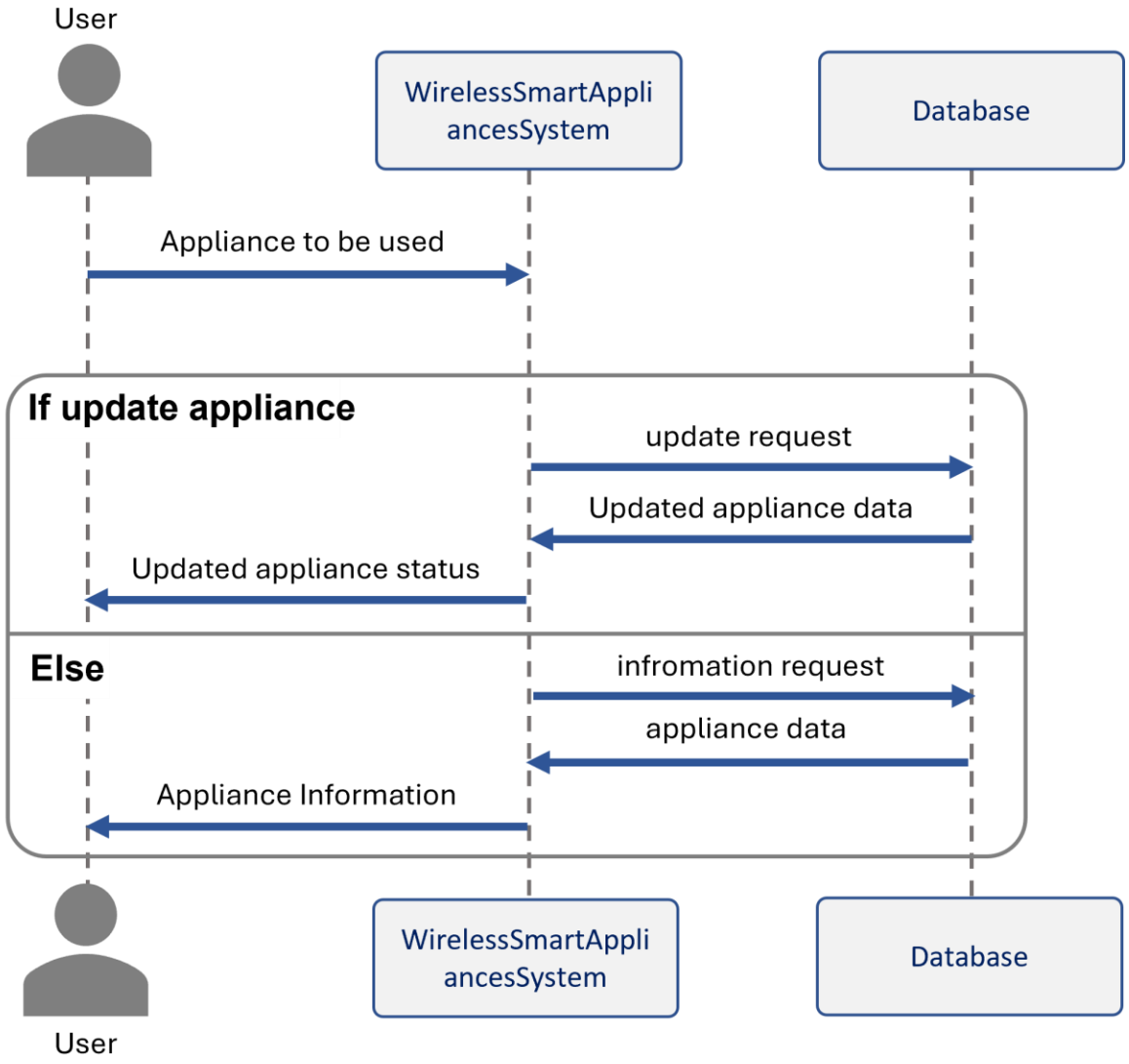


- Class description

  - ✓ **Wireless Smart Appliances Class:** Smart-home appliances with a built-in WiFi-Chip off the same functionality as wired appliances. However instead of an GPIO-Pin the smart appliances need an IP-address. With the IP-address you can create an instance of a smart appliance and test and control it. The functions of WirelessSmartAppliancesSystem need access to the database to update or get data.

5.3.3.2. Sequence Diagram

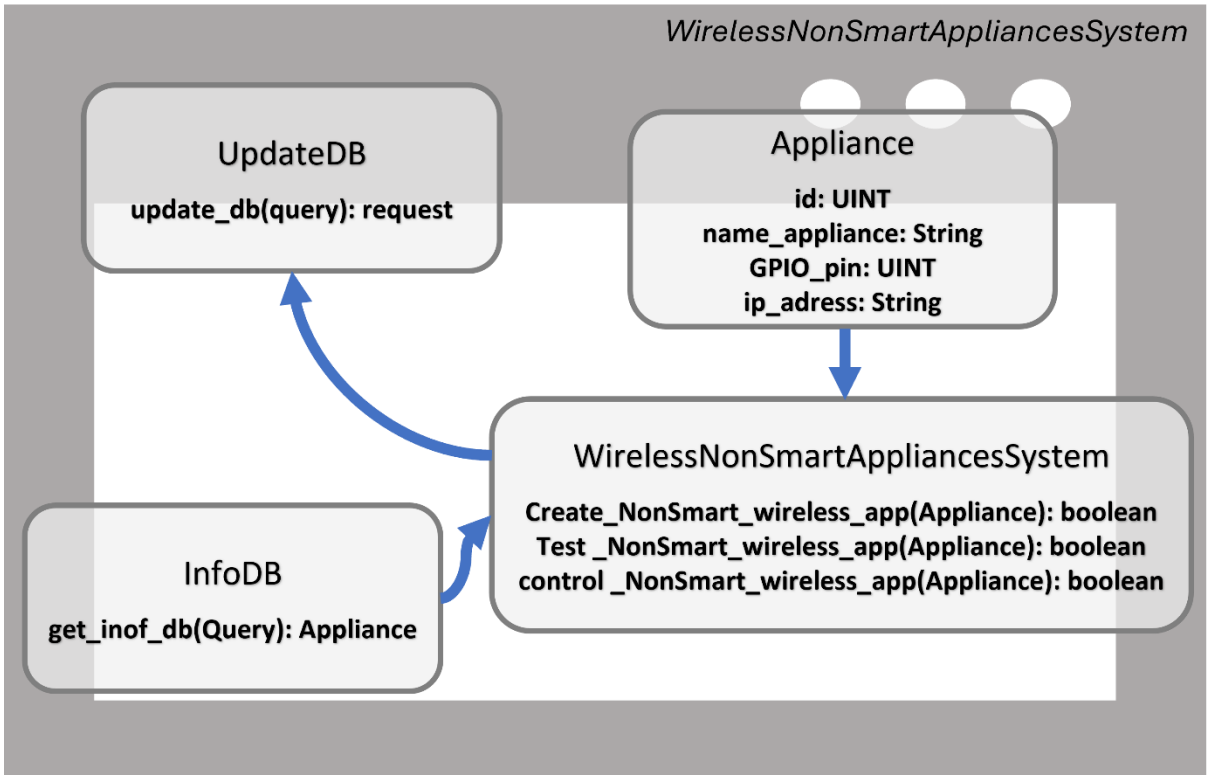
[Figure 8] Sequence diagram - Wireless Smart Appliances system



5.3.4. Wireless Non-Smart Appliances System

5.3.4.1. Class Diagram

[Figure 9] Class diagram - Wireless Non-Smart Appliances System

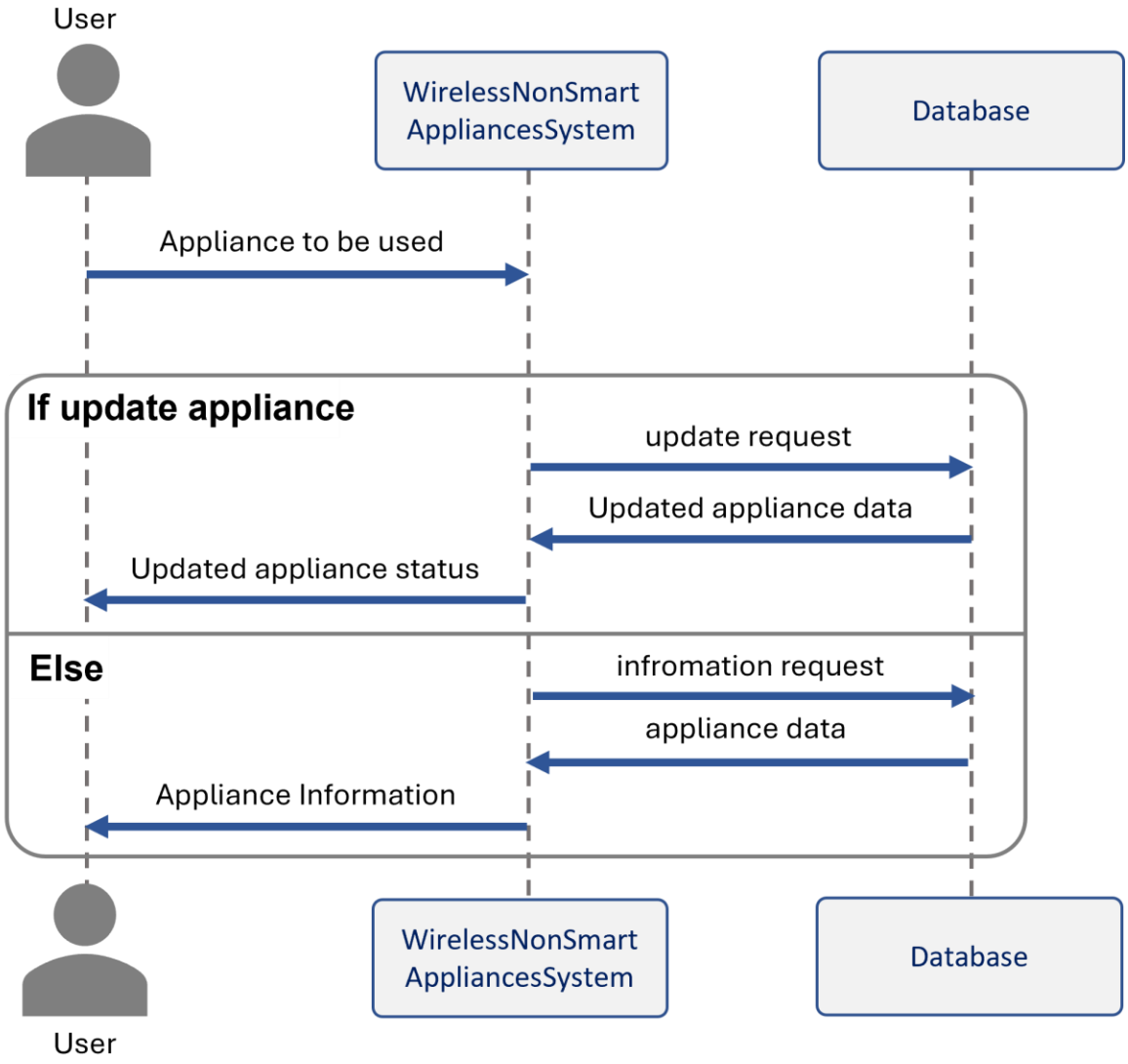


- Class description

  - ✓ **Wireless Non-Smart Appliances Class:** This class is a mix off both previous classes. Since non-smart appliances need an additional WiFi-Chip they also need an ip-address. Additional WiFi-Chips like the ESP8266 are usually mounted on a board which has GPIO-Pins. Therefore, non-smart appliances also need the GPIO-Pin number to define the Pin on the WiFi-Chip board. The functionality is the same as for the other appliances. The functions of WirelessNonSmartAppliancesSystem need access to the database to update or get data.

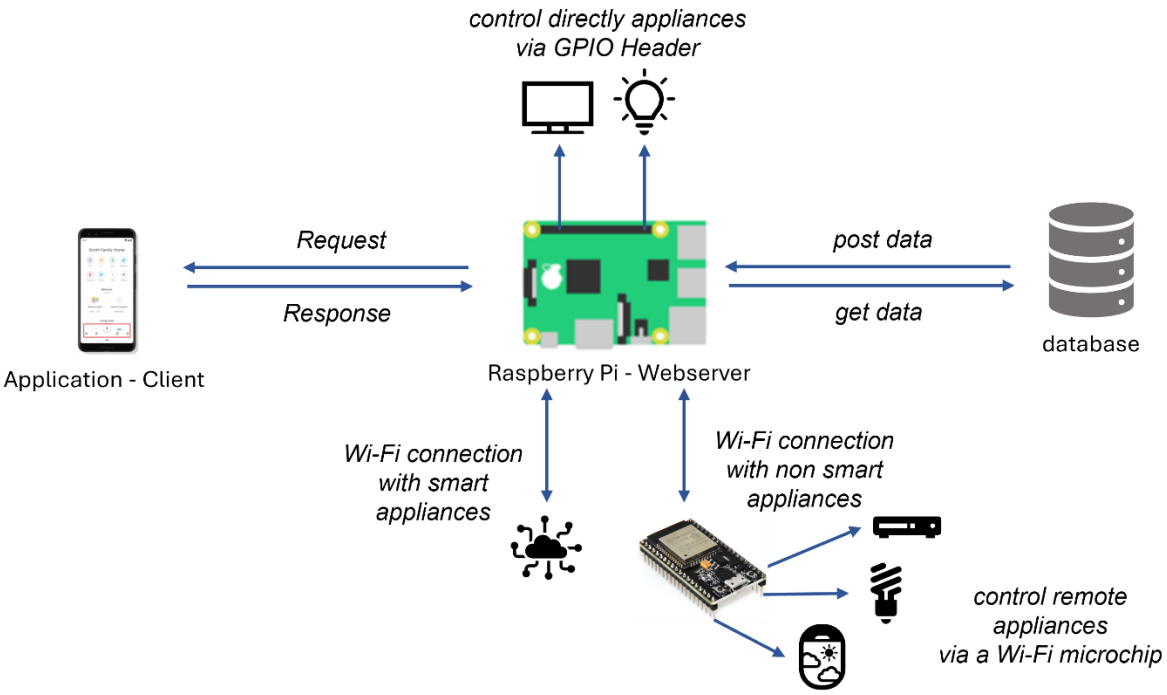
5.3.4.2. Sequence Diagram

[Figure 10] Sequence diagram - Wireless Non-Smart Appliances System



5.4. Hardware architecture

[Figure 11]: Hardware architecture



6. Protocol Design

6.1. Objectives

This chapter explains the protocols that are used for interaction between each subsystem and describe the whole structure of that protocols. It will focus on the application and its server and also explain the definition method of each interface in application.

6.2. JSON

JSON is abbreviation for JavaScript Object Notation, a lightweight data exchange format that is widely used when storing or storing data. And it refers to an expression used when creating an object in Javascript. JSON expressions are easy to understand for both humans and machines and its capacity is small, so recently, JSON has replaced XML and is widely used for data transmission. JSON is just a data format, not any communication method or programming grammar, it is simply a way of expressing data.

6.3. OAuth

OAuth is an open standard for delegation of access, used as a common means by which Internet users can grant website or application access to their information on other websites without providing a password. This mechanism is used by several companies, such as Amazon, Google, Facebook, Microsoft, and Twitter, and allows users to share information about accounts on third-party applications or websites. Before OAuth was used, there was no standard for authentication method, so the existing basic authentication ID and password were used, which is a weak structure in terms of security. In the case of non-basic authentication, each application confirmed the user according to the method of their own developed company. OAuth is a standardized authentication method. With OAuth, applications that share this authentication do not need separate authentication. Therefore, it becomes possible to integrate and use multiple applications.

6.4. Authentication

6.4.1. Register

- Request

[Table 1] register request

Attribute	Detail	
Protocol	OAuth	
Request body	Id / Password	Id / Password
	Request token	Token for OAuth
	User	User information

- Response

[Table 2] register response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 401 (unauthorized)	
	HTTP 404 (Not found)	
	HTTP 500 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Register success”
Failure response body	Message	Message: “Register fail”

6.4.2. Log-In

- Request

[Table 3] Log-in request

Attribute	Detail	
Protocol	OAuth	
Request body	Id / Password	Id / Password
	Request token	Token for OAuth
	User	User information

- Response

[Table 4] Log-in response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 401 (unauthorized)	
	HTTP 404 (Not found)	
	HTTP 500 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Login success”
Failure response body	Message	Message: “Login fail”

6.5. Appliances

6.5.1. Set Appliances List

- Request

[Table 5] Set Appliances List request

Attribute	Detail	
URI	/:userid/appliances/list	
Method	POST	
Parameter	User	Basic User Information
	Appliance	Appliance info
Header	Authorization	User authentication

- Response

[Table 6] Set Appliances List response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 403 (Forbidden)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.5.2. Get Appliances List

- Request

[Table 7] Get Appliances List request

Attribute	Detail	
URI	/:userid/appliances/list	
Method	Get	
Header	Authorization	User authentication

- Response

[Table 8] Get Appliances List response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 403 (Forbidden)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
	Appliance	Appliance info list
Failure response body	Message	Message: “Access fail”

6.6. Get Control of Appliances

- Request

[Table 7] Get Control of Appliances request

Attribute	Detail	
URI	/:userid/:appliance_id/control/list	
Method	Get	
Parameter	User	Basic User Information
	Appliance	Specific appliance id
Header	Authorization	User authentication

- Response

[Table 8] Get Control of Appliances response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Controls	Control list of specific appliance
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.7. Search Appliances

- Request

[Table 9] Search Appliances request

Attribute	Detail	
URI	/appliances	
Method	Get	
Parameter	User	Basic User Information
Header	Authorization	User authentication

- Response

[Table 10] Search Appliances request

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Appliances	Appliances list registered
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.8. Macro list

6.8.1. Set Marco List

- Request

[Table 13] Set Marco List request

Attribute	Detail	
URI	/:userid/macro/list	
Method	POST	
Parameter	User	Basic User Information
	Macro	Marco list info
Header	Authorization	User authentication

- Response

[Table 14] Set Marco List response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 403 (Forbidden)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.8.2. Get Macro List

- Request

[Table 15] Get Macro List request

Attribute	Detail
URI	/:userid/macro/list
Method	GET

Header	Authorization	User authentication

- Response

[Table 16] Get Macro List response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
	Macro List	List of user’s macro
Failure response body	Message	Message: “Access fail”

6.9. Macro

6.9.1. Set Marco

- Request

[Table 17] Set Macro List request

Attribute	Detail	
URI	/:userid/macro/:macroid	
Method	POST	
Parameter	User	Basic User Information
	Macro	Macro info
Header	Authorization	User authentication

- Response

[Table 13] Set Macro List response

Attribute	Detail
Success Code	HTTP 200 OK
Failure Code	HTTP 400 (Bad request, overlap)
	HTTP 403 (Forbidden)

	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.9.2. Get Macro List

- Request

[Table 14] Get Macro List request

Attribute	Detail	
URI	/:userid/macro/:macroid	
Method	GET	
Header	Authorization	User authentication

- Response

[Table 15] Get Macro List response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
	Macro	Specific information of macro
Failure response body	Message	Message: “Access fail”

6.10. Control

6.10.1. Single Control command

- Request

[Table 16] Single Control command request

Attribute	Detail	
URI	/:userid/control/single/:applianceid	
Method	Get	
Parameter	User	Basic User Information
	Control	Control command for appliance
Header	Authorization	User authentication

- Response

[Table 17] Single Control command response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

6.10.2. Macro command

- Request

[Table 18] Macro command request

Attribute	Detail	
URI	/:userid/control/macro/:macroid	
Method	Get	
Parameter	User	Basic User Information
	Macro	Macro command for appliance
Header	Authorization	User authentication

- Response

[Table 19] Macro command response

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

7. Database Design

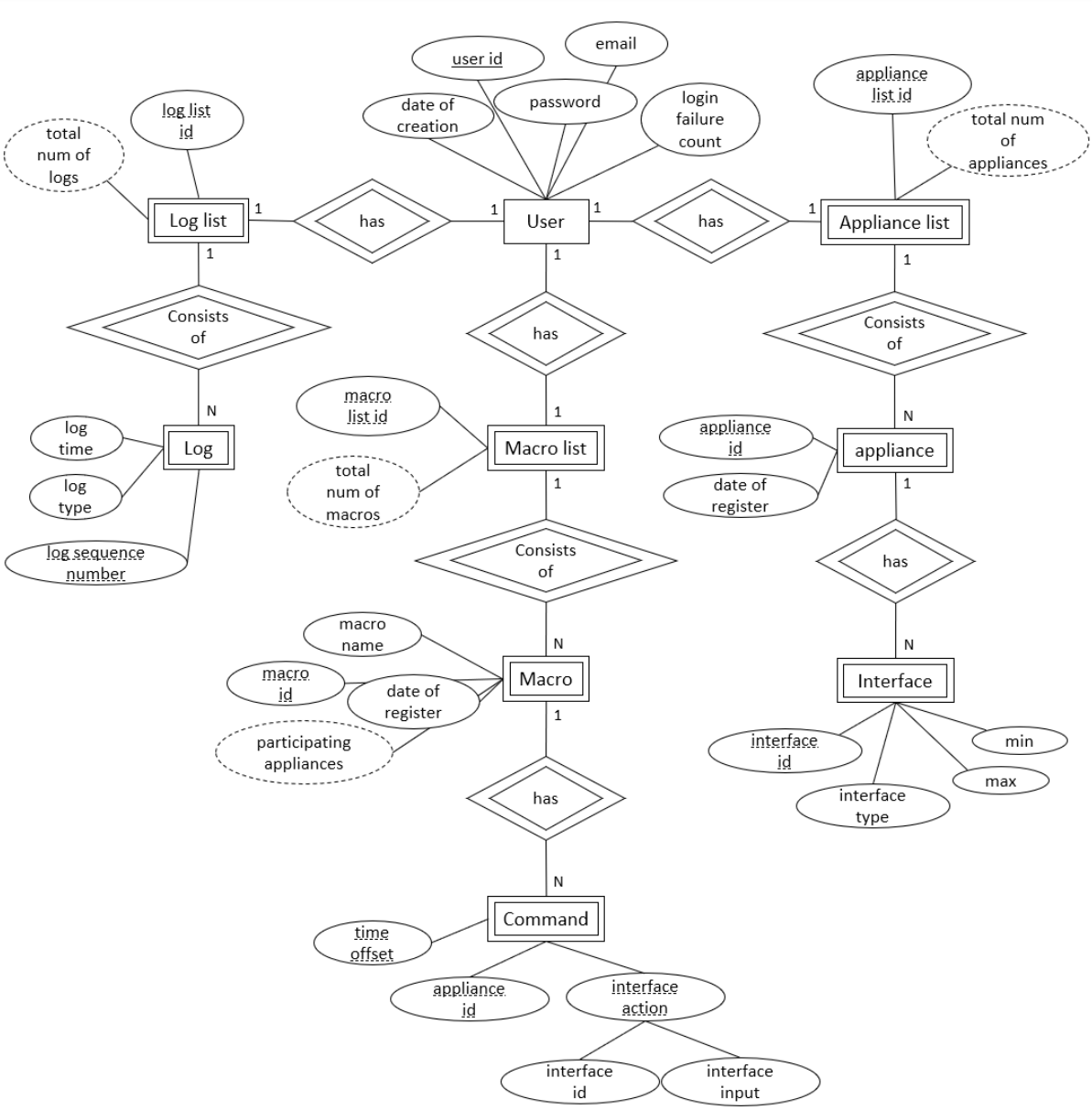
7.1. Objectives

This section describes the system data structures and how these are to be represented in a database. It first identifies entities and their relationship through ER-diagram (Entity Relationship diagram). Then, it generates Relational Schema and SQL DDL (Data Description Language) specification.

7.2. ER Diagram

The system consists of nine entities: User, Log list, Log, Appliance list, Appliance, Interface, Macro List, Macro, and Command. ER-diagram expresses each entity as a rectangle and their relationship as a rhombus. The numbers attached to the lines connecting two different entities show the multiplicity of the relationship between those entities. For example, for a one-to-one relationship, two 1s are attached to the lines connecting two entities having that relationship. Normal attributes of an entity are expressed as an eclipse. Derived attributes are expressed as a dashed eclipse. Weak entities are expressed as a double rectangle, and Identifying relationships are expressed as a double rhombus. Key attributes are underlined and partial key attributes are underlined with a dashed line.

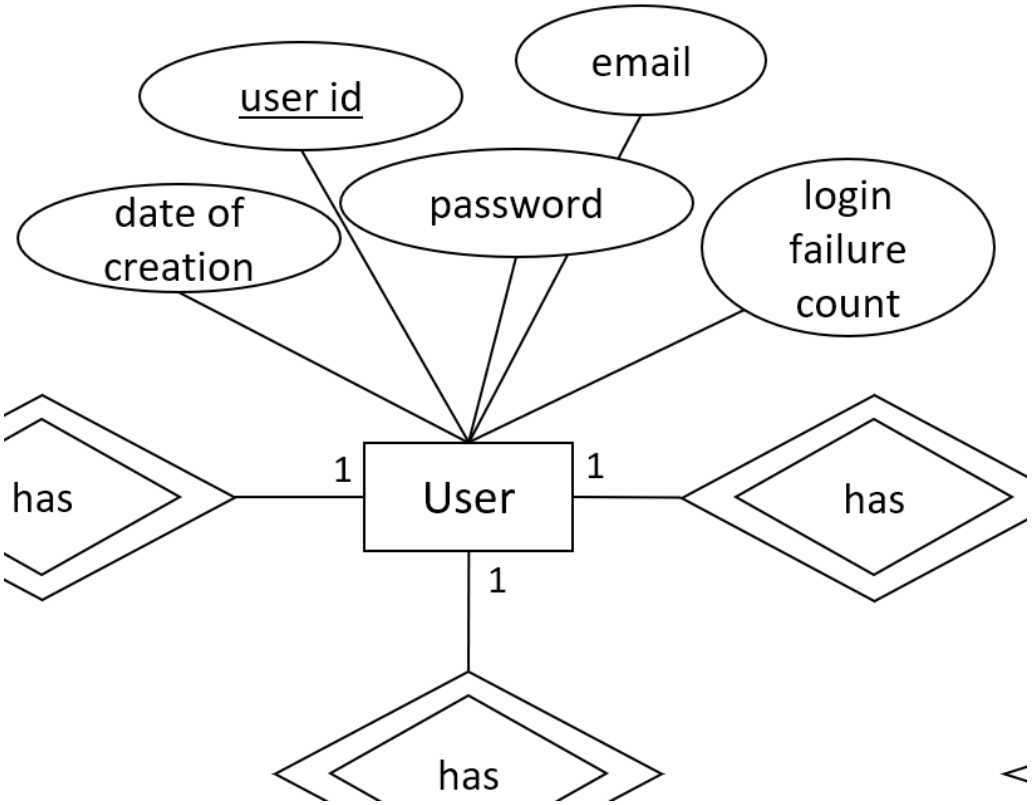
[Figure 1] ER-Diagram



7.2.1 Entities

7.2.1.1. User

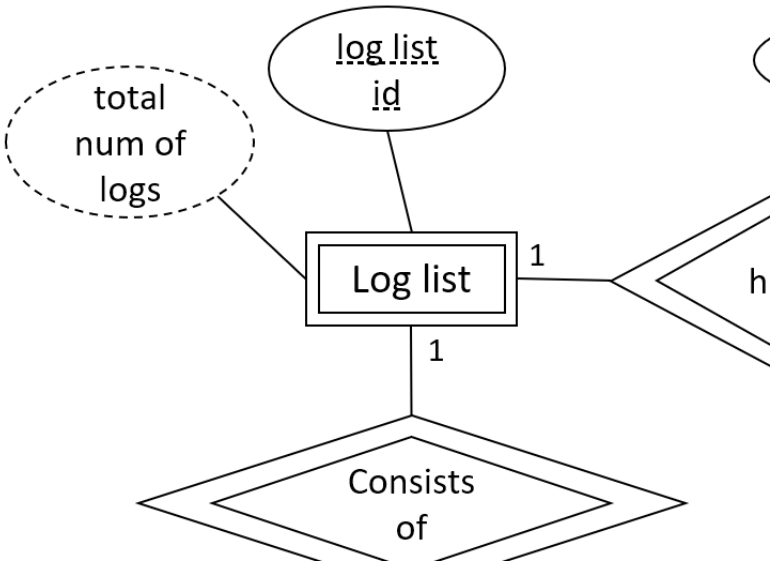
[Figure 2] ER diagram, Entity, User



User entity represents a user of our application. This entity consists of user id, password, email, date of creation, and login failure count attributes. ‘user id’ attribute is the primary key and is underlined. Everytime a user fails to log-in, ‘login failure count’ value is incremented, and if it reaches a specific value, the user can be blocked from logging in. The blocked user should verify him/herself with his/her email account.

7.2.1.2. Log list

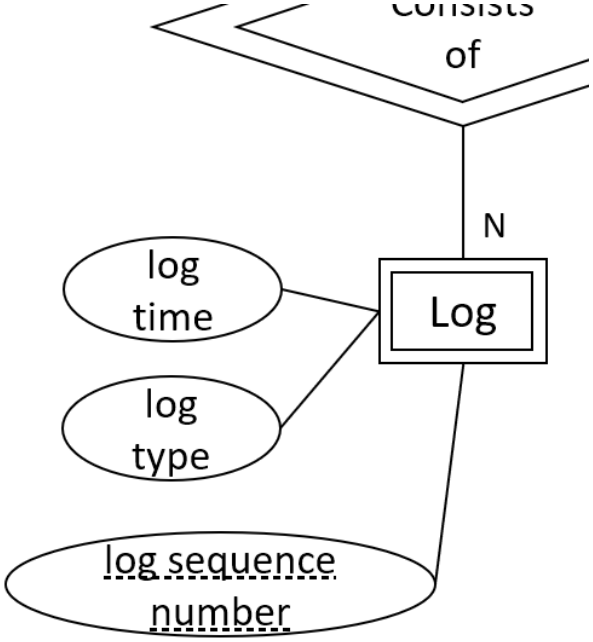
[Figure 3] ER diagram, Weak Entity, Log list



Log list represents a list of logs which contains records about user activities. It consists of only log list id. ‘total num of logs’ is a derived attribute. Thus, it is not explicitly stored in the DB. Log list has a one-to-one relationship with User entity and a one-to-many relationship with Log entity. ‘log list id’ is used as the partial key.

7.2.1.3. Log

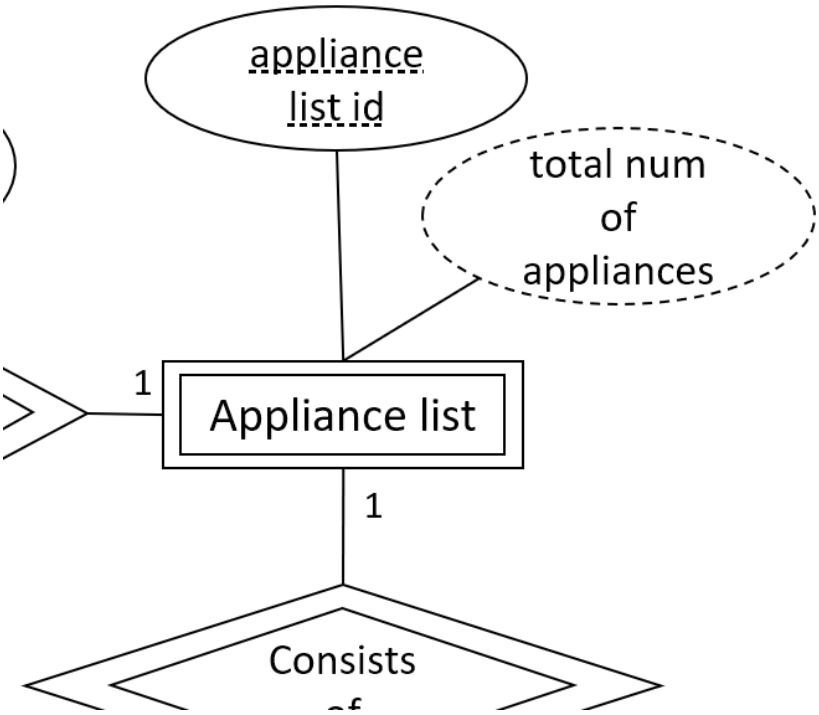
[Figure 4] ER diagram, Weak Entity, Log



Log represents a specific user activity that occurred at a specific time. It consists of log time, log type, and log sequence number. ‘log type’ specifies what kind of user activity was recorded in the log. ‘log time’ specifies when the user activity was recorded. ‘log sequence number’ is used to identify a specific log. Log has a many-to-one relationship with Log list entity. ‘log sequence number’ is used as the partial key.

7.2.1.4. Appliance list

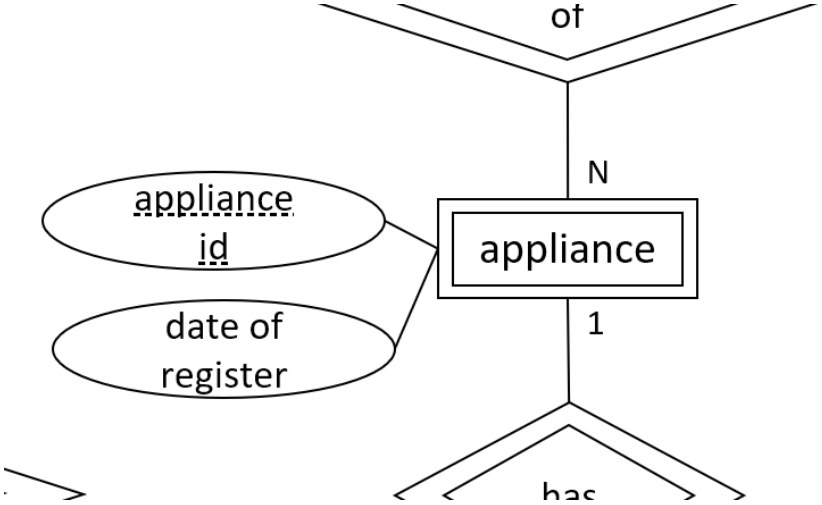
[Figure 5] ER diagram, Weak Entity, Appliance list



Appliance list represents a list of appliances that are registered to the user's account. It consists of only appliance list id. 'total num of appliances' is a derived attribute. Thus, it is not explicitly stored in the DB. Appliance list has a one-to-one relationship with User entity and a one-to-many relationship with Appliance entity. 'appliance list id' is used as the partial key.

7.2.1.5. Appliance

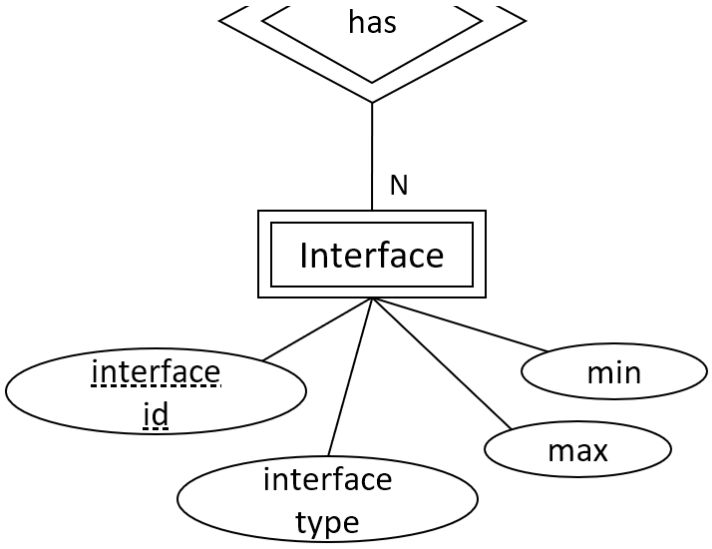
[Figure 6] ER diagram, Entity, Appliance



Appliance represents an appliance that was registered to the user's account. It consists of appliance id, and date of register. 'appliance id' is used to identify a specific appliance. 'date of register' specifies when the appliance was registered. Appliance has a many-to-one relationship with Appliance list entity, and has a one-to-many relationship with interface entity. 'appliance id' is used as the partial key.

7.2.1.6. Interface

[Figure 7] ER diagram, Entity, Interface

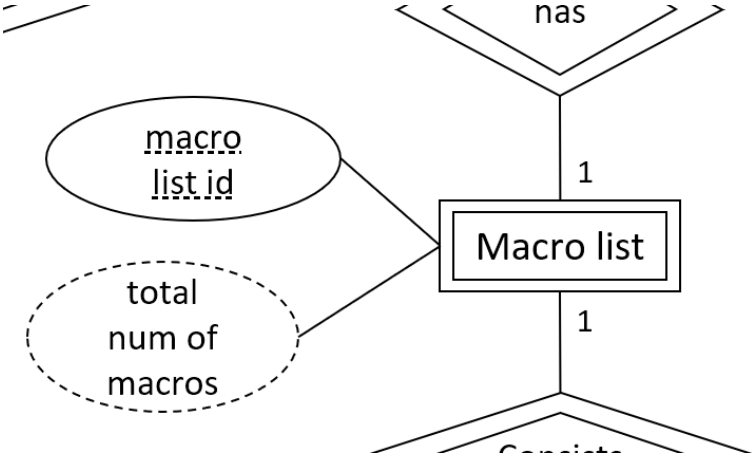


Interface represents an interface that is part of an appliance. There are many types of interfaces such as 'volume interface', 'channel interface', and 'power interface'. 'max' and 'min' values

are used when the user wants to control an interface that has a slider bar such as volume bar. They specifies the maximum and minimum values the slider can indicate. ‘interface id’ is used to identify a specific interface. Interface has a many-to-one relationship with Appliance entity. ‘interface id’ is used as the partial key.

7.2.1.7. Macro list

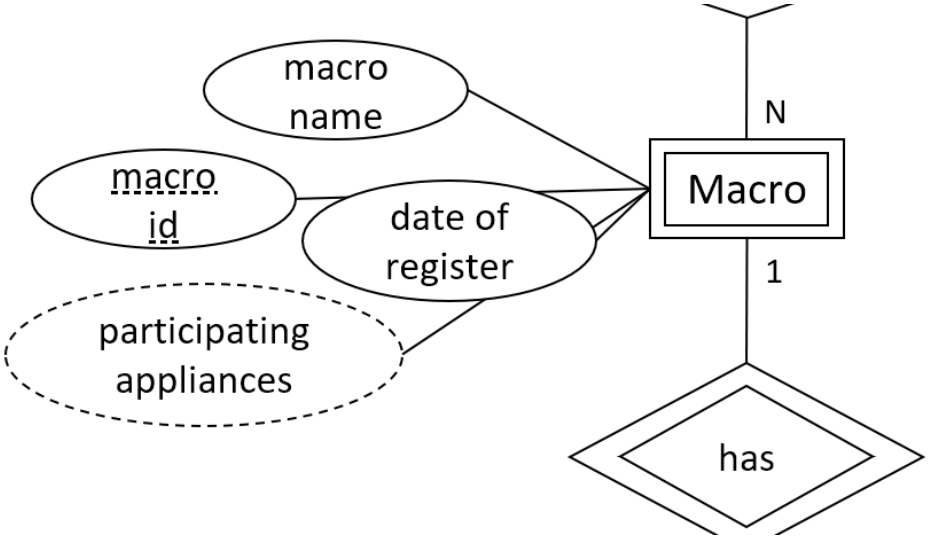
[Figure 8] ER diagram, Entity, Macro list



Macro list represents a list of macros that the user recorded. It consists of only macro list id. ‘macro list id’ is used as the partial key. ‘total num of macros’ is a derived attribute. Thus, it is not explicitly stored in the DB. Macro list has a one-to-one relationship with User entity and a one-to-many relationship with Macro entity.

7.2.1.8. Macro

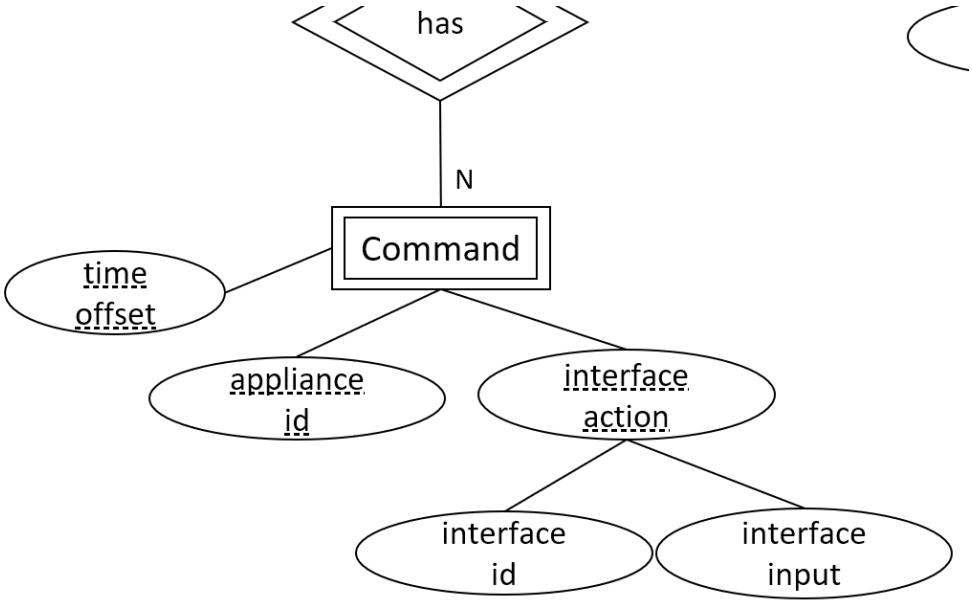
[Figure 9] ER diagram, Entity, Macro



Macro represents a sequence of commands that the user's appliances should execute. 'macro name' is the user-specified name and it is not used to id identify a macro. 'date of register' is the date when the macro was registered to the user's account. 'participating appliances' is a derived attribute. Thus, it is not explicitly stored in the DB. 'macro id' is used as the partial key.

7.2.1.9. Command

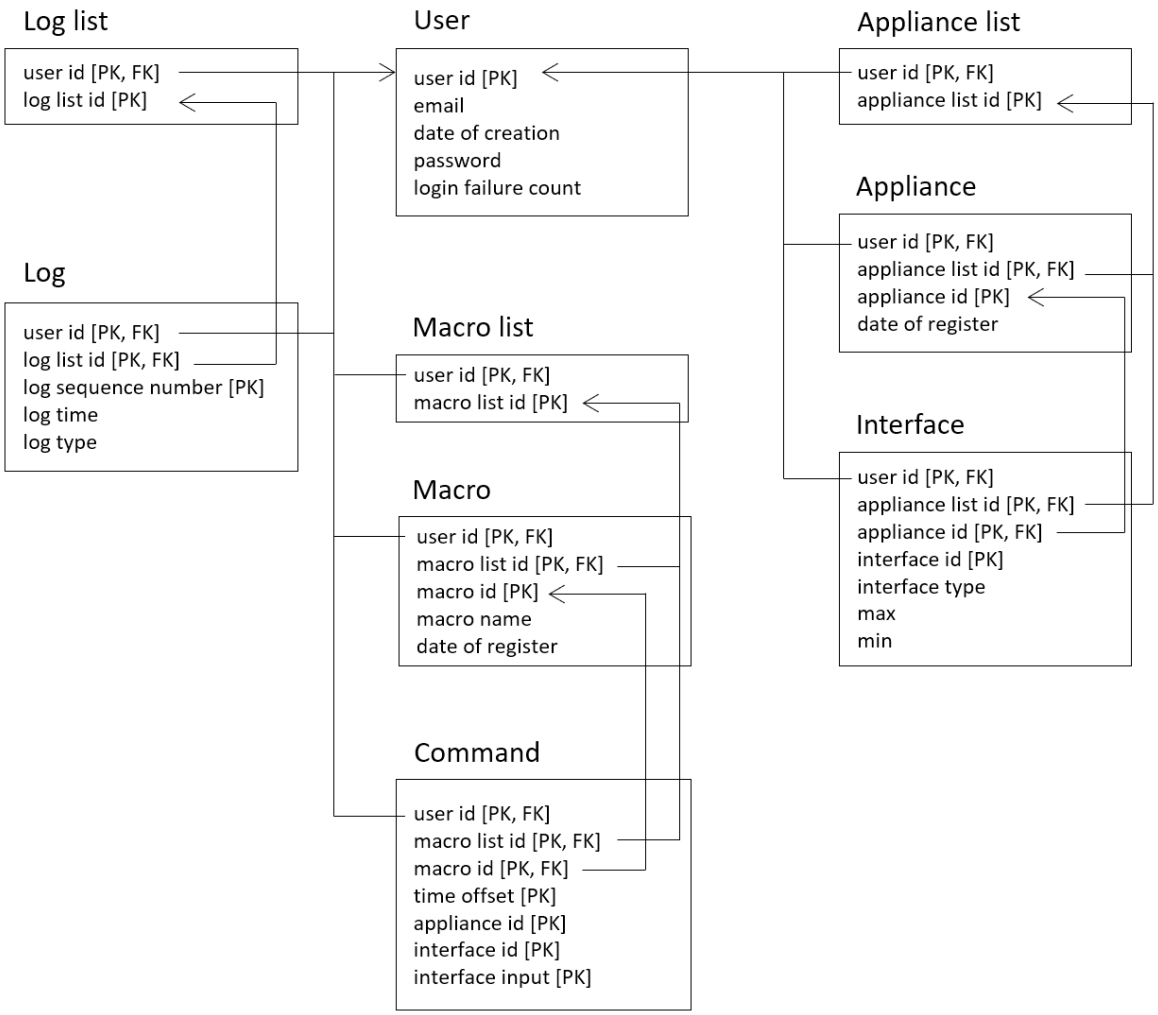
[Figure 34] ER diagram, Entity, Command



Command represents a single job that a specific interface in an appliance should execute. ‘time offset’ specifies the duration of waiting time after the macro started. ‘interface action’ consists of two attributes: ’interface id’ and ‘interface input’. ’interface id’ is the id of the target interface that we want to use, and ‘interface input’ the input value to the interface. ‘time offset, appliance id, and interface action’ are used as the partial key.

7.3. Relational Schema

[Figure 10] Relational Schema



7.4. SQL DDL

7.4.1 User

```
CREATE TABLE User(
  user_id  INT(8) NOT NULL
  email VARCHAR(320) NOT NULL
  date_of_creation DATE NOT NULL
  password VARCHAR(30) NOT NULL
  login_failure_count INT(1) NOT NULL
  PRIMARY KEY (user_id)
);
```

7.4.2 Log_list

```
CREATE TABLE Log_list(
  user_id INT(8) NOT NULL
  log_list_id INT(4) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  PRIMARY KEY (user_id, log_list_id)
);
```

7.4.3 Log

```
CREATE TABLE Log(
  user_id  INT(8) NOT NULL
  log_list_id INT(4) NOT NULL
  log_sequence_number INT(8) NOT NULL
  log_time DATE NOT NULL
  log_type INT(2) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  FOREIGN KEY (log_list_id) REFERENCES Log_list(log_list_id)
  PRIMARY KEY (user_id, log_list_id, Log_sequence_number)
);
```

7.4.4 Macro_list

```
CREATE TABLE Macro_list(
  user_id INT(8) NOT NULL
  macro_list_id INT(4) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  PRIMARY KEY (user_id, macro_list_id)
);
```

7.4.5 Macro

```
CREATE TABLE Macro(
  user_id  INT(8) NOT NULL
  macro_list_id INT(4) NOT NULL
  macro_id INT(4) NOT NULL
  macro_name VARCHAR(30) NOT NULL
  date_of_register DATE NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  FOREIGN KEY (macro_list_id) REFERENCES Macro_list(macro_list_id)
  PRIMARY KEY (user_id, macro_list_id, macro_id)
);
```

7.4.6 Command

```
CREATE TABLE Command(
  user_id  INT(8) NOT NULL
  macro_list_id INT(4) NOT NULL
  macro_id INT(4) NOT NULL
  time_offset FLOAT NOT NULL
  appliance_id INT(4) NOT NULL
  interface_id INT(4) NOT NULL
  interface input INT(4) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  FOREIGN KEY (macro_list_id) REFERENCES Macro_list(macro_list_id)
  FOREIGN KEY (macro_id) REFERENCES Macro(Macro_id)
  PRIMARY KEY (user_id, macro_list_id, macro_id, time_offset, appliance_id, interface_id, interface_input)
);
```

7.4.7 Application_list

```
CREATE TABLE Application_list(
  user_id INT(8) NOT NULL
  appliance_list_id INT(4) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  PRIMARY KEY (user_id, appliance_list_id)
);
```

7.4.8 Appliance

```
CREATE TABLE Appliance(
  user_id  INT(8) NOT NULL
  appliance_list_id INT(4) NOT NULL
  appliance_id INT(4) NOT NULL
  date_of_register DATE NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  FOREIGN KEY (appliance_list_id) REFERENCES Application_list(appliance_list_id)
  PRIMARY KEY (user_id, appliance_list_id, appliance_id)
);
```

7.4.9 Interface

```
CREATE TABLE Interface(
  user_id  INT(8) NOT NULL
  appliance_list_id INT(4) NOT NULL
  appliance_id INT(4) NOT NULL
  interface_id INT(4) NOT NULL
  interface_type INT(4) NOT NULL
  max INT(4) NOT NULL
  min INT(4) NOT NULL
  FOREIGN KEY (user_id) REFERENCES User(User_id)
  FOREIGN KEY (appliance_list_id) REFERENCES Application_list(appliance_list_id)
  FOREIGN KEY (appliance_id) REFERENCES  Appliance(appliance_id)
  PRIMARY KEY (user_id, appliance_list_id, appliance_id, interface_id)
);
```

8. Testing Plan

8.1. Objectives

This chapter describes plans for tests that include three main subgroups: development testing, release testing, and user testing. These tests are critical in that they detect potential errors and defects in the product and ensure flawless operation and reliable product market and customer release.

8.2. Testing Policy

8.2.1. Development Testing

Development testing is mainly performed for synchronized application of a wide range of fault prevention and detection strategies to reduce the potential risk of software development and save time and costs.

At this stage, the software may be unstable because it has not been sufficiently tested, and components may crash. Therefore, static code analysis, data flow analysis, peer code review, and unit testing should be performed at this stage. Through these processes, we focus primarily on achieving 1) performance, 2) reliability, ensuring safe and fault-free operations, and 3) security, which define the identity of the software.

8.2.1.1. Performance

Mapping within applications is the most time-consuming task in the system, and it is important for developers to reduce mapping time for many concurrent users. As specified in the recommended specification, the system should provide the user with results within 5 seconds. We will prepare test cases for different preferences, evaluate the speed of mapping functions, and improve the flow of code with respect to mapping algorithms and communication with servers.

8.2.1.2. Reliability

For the system to operate safely without failure, the subcomponents and units that make up the system must first operate and connect correctly. Therefore, we have to go through development

tests from the unit development stage and repeatedly check the failure while each unit is integrated into the system.

8.2.1.3. Security

Securing information of users and the system is a crucial matter to be handled by developers. Regardless of the value of the information, it should be protected from unwanted visitors to the system.

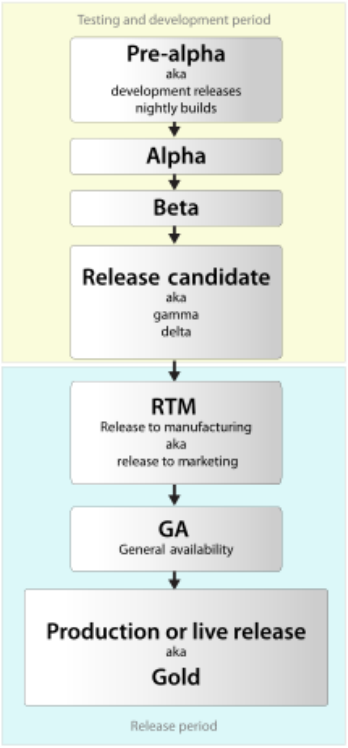
For the security of the system, we can access a near-finished version of the app to identify security issues and create reports through manual code review. In addition, other mobile app security testing services provided by Ostorlab, Appvigil, and others are available, indicating that developers have overlooked application vulnerabilities.

8.2.2. Release Testing

One of the most critical parts of any software development project is to release the product to the market, and customers. A technically good software can go wrong due to a wrong way of release. Therefore, release testing is inevitable for better connection between the product and the market. Release testing is testing a new version of a software/application to verify that that the software can be released in a flawless, so it does not have any defects on its working. It should be carried out before release.

Depending on the software release life cycle, tests typically begin with the 'alpha' version of the software that has completed its basic implementation. We will start development testing in Alpha version and release beta for further testing including user and release testing.

[Figure 11] Software Release Life Cycle



After Beta version is released, we would get feedback from actual users as well as developers.

8.2.3. User Testing

We need to set up a possible scenario and a realistic situation to proceed with the necessary user test. We assume that the “Smart Remote Control” application will have 30 users. After setting up this situation, we conducted our own use case testing with an Android emulator, distributing the “Smart Remote Control” application beta version to them and collecting user reviews.

8.2.4. Testing Case

Test cases are set up based on three basic aspects: functionality (interaction), performance, and security. Set up 5 test cases (15 cases in total) on each side, test the application, and complete the assessment sheet.

9. Development Plan

9.1. Objectives

This chapter illustrates the technologies and environment for the development of the application.

9.2. Frontend Environment

9.2.1. Adobe Photoshop (UI/UX Design)

[Figure 12] Adobe Photoshop logo

It is a raster graphics editor developed and published by Adobe Inc. for Windows and macOS. This program would provide aesthetical layout and icons for improved user experience during our project.

9.2.2. Adobe Xd (UI/UX Design)

[Figure 13] Adobe Xd logo

It is an all-in-one UX/UI solution program for designing web and mobile app. It provides interactive prototypes in various platforms, which facilitates communication among teammates and enables prompt reflection of feedbacks.

9.2.3. Kakao Oven (UI/UX Design)

[Figure 14] Adobe Xd logo

It is a prototyping tool provided by Kakao. Prototyping refers to designing UI (interface) and UX (user experience) in advance before software development. We can use it for the 'planning' stage of the app. Unlike other tools such as 'Sketch' and 'Protofi', it is basically free. Although Adobe XD is also a free tool, Kakao Oven has the advantage of not having to install a separate program thanks to its HTML5 base. Even if it's a free tool, but most of the planning we imagine is visualizable.

9.2.4. Android Studio (Application)

[Figure 15] Android Studio logo

It is the official integrated development environment for Google's Android operating system, and it is the most fundamental environment for developing "Smart Remote Control" application. Supporting various programming languages and featuring an intelligent code editor equipped with IntelliJ IDEA interface, Android Studio enables developers to make code more easily and accurately.

For frontend development, we can build a structure of the application, by setting visible actions to be followed according to user interaction in the application using XML layouts.

9.3. Backend Environment

9.3.1. Github (Open source)

[Figure 16] Github logo

It provides hosting for software development version control using Git. It enables teammates to develop a single project together, resulting in easy integration of components. We are now using GitHub for developing “Smart Remote Control” application and controlling version of it.

9.3.2. Firebase (DBMS)

[Figure 17] Firebase logo

It supports development of mobile and web applications by providing various features such as cloud storage, real-time database, machine learning kit, etc. Among them, we would use real-time database feature for managing data of users, friends, chatrooms, etc. Thanks to real-time database, data is synchronized across all the clients connected to this database. It means all clients can share a single real-time database instance and receive updated data with automated update.

9.3.3. SQLite Database (DBMS)

[Figure 18] SQLite Database logo

It is a database management system, such as MySQL and PostgreSQL, but is a relatively lightweight database used by applications, not servers. Although it is not suitable for large-scale tasks compared to typical RDBMS, it is good for small and medium-sized tasks. The API is also characterized by simply calling the library and using only one file to store data. It is also a database built into the Google Android operating system.

9.3.4. Android Studio (Application)

[Figure 19] Android Studio logo

We can add additional features by connecting the application to external APIs, and connect XML files to activities of the application. In addition, we connect the application to DB server for the control of data.

9.3.5. Node.js (Server)

[Figure 20] Nodejs logo

It is a software platform used to develop scalable network applications (especially server-side). It utilizes JavaScript as a written language and has high processing power over non-blocking I/O and single-thread event loops. It includes a built-in HTTP server library, which allows web servers to operate without separate software, such as Apache, allowing more control over the behavior of web servers.

9.3.6. AWS EC2 (Server)

[Figure 21] AWS logo

It is the center of Amazon's cloud computing platform Amazon Web services, allowing users to rent virtual machines and run their own computer applications on them. EC2 encourages scalable application deployment by providing a web service that allows users to boot into the Amazon Machine Image (AMI) and configure virtual machines that Amazon calls "instances" with the desired software. We can create, start, and shut down server instances if necessary, and they use the term "elastic" (elastic) because they pay per hour for running servers. EC2 provides us with

control over geographic instance locations that allow for latency optimization and high levels of multiplexing.

9.4. Constraints

The system will be designed and implemented based on the contents mentioned in this document. Other details are designed and implemented by selecting the direction preferred by the developer, but the following items are observed.

- Use the technology that has already been widely proven.
- Mapping speed should not exceed 5 seconds.
- Avoid using technology or software that requires a separate license or pays for royalty. (Exclude this provision if this is the only technology or software that the system must require.)
- Decide in the direction of seeking improvement of overall system performance.
- Decide in a more user-friendly and convenient direction
- Use open source software whenever possible
- Consider the system cost and maintenance cost
- Consider future scalability and availability of the system
- Optimize the source code to prevent waste of system resources
- Consider future maintenance and add sufficient comments when writing the source code
- Develop with Windows 10 environment and Android Studio whose build tools version is 29.0.3
- Develop with minimum Android version 6.0 (API 23) and target Android version 29
- Emulate the system using Android version 10 (API 29)

9.5. Assumptions and Dependencies

All systems in this document are designed and implemented based on Android devices and open sources. Therefore, all content is based on the Android operating system with a minimum API version 23 and may not be applicable to other operating systems or versions.

10. Supporting Information

10.1. Software Design Specification

This software design specification was written in accordance with the IEEE Recommendation (IEEE Recommended Practice for Software Design Description, IEEE-Std-1016).

10.2. Document History

[Figure 22] Document History

Date	Version	Description	Writer
2022/05/08	0.1	Style and overview	조윤근
2022/05/09	1.0	Addition of 8, 9	Stefan
2022/05/10	1.1	Addition of 1,7	금상인
2022/05/10	1.2	Addition of 6	이대희
2022/05/11	1.3	Addition of 3	전종문
2022/05/12	1.4	Addition of 2	전종문
2022/05/12	1.5	Addition of 5	Stefan
2022/05/13	1.6	Addition of 4	이대희
2022/05/13	1.7	Addition of 4	금상인
2022/05/14	1.8	Revision of 5	Stefan
2022/05/14	1.9	Addition of 10	금상인
2022/05/14	2.0	Revision of 6	조윤근
2022/05/15	2.1	Revision of style	조윤근