Smart Sleep System

Software Design Specification

2022.05.15

Introduction to Software Engineering 42

TEAM 14

Team Leader	Younghoon	Jun
Team Member	SanzBernal	Maria
Team Member	Aitana	Morote
Team Member	Yunseong	Kim
Team Member	Seryeong	Kim
Team Member	Heegwan	Son
Team Member	Eunki	Song

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Preface

This chapter contains the readership information, readership, scope, objective of this document and the document structure of this Software Design Document for this project, Smart Sleep System.

1.1. Readership

This Software Design Document is divided into 8 sections with various subsections. The structure of the Software Design Document can be found as listed below, in the Document Structure subsection of this SDD. In this document, Team 14 is the main reader. Additionally, professors, TAs, team members in the Introduction to Software Engineering class and all relevant stakeholders can be the main readers. Developers should implement the software according to the software design described in this document, and stakeholders can use this document to confirm that this project meets the requirements.

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 the Smart Sleep System which users can set their own personalized and optimized sleep environment.

1.3. Objective

The primary purpose of this Software Design Document is to provide a description of the technical design aspects for this project, Smart Sleep System. This document describes the software architecture and software design decisions for the implementation of Smart Sleep System. It also provides an architectural overview of the system to describe different aspects of the system. Furthermore, it specifies the structure and design of some of the modules discussed in the Software Requirement Specification document and displays some of the use cases that have been transformed into sequential and 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 this project.

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 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. Testing Plan**: this chapter describes testing plan for our system.
- **7. Development Plan**: this chapter describes which tools to use to develop the system, constraints, assumption, and dependencies for developing this system.
- **8. Supporting Information**: this chapter describes baseline of this document and history of this document.

2. Introduction

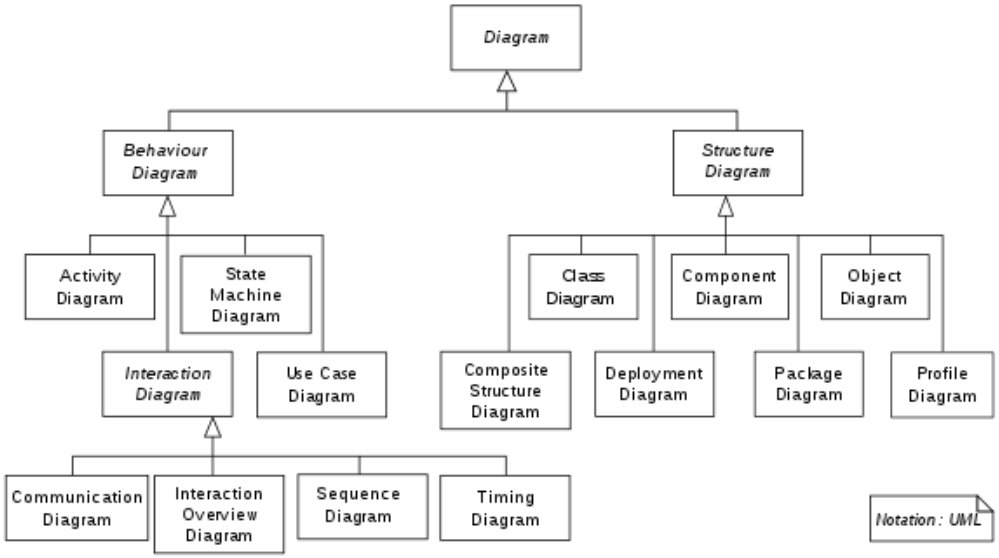
The project will provide smart-sleep-system for many people who want better quality sleep as more people study or enjoy hobbies while reducing their sleep in busy daily lives. The system assumes that there is one person in a space and will help create the optimal sleep environment (temperature, humidity, brightness) that the user wants. This design document presents the designs used or intended to be used in implementing the project. The designs described, follow the requirements specified in the Software Requirement Specification document prepared earlier for the project.

2.1. Objectives

In this chapter, we describe the various tools and diagrams which we have applied to this project in the design phase.

2.2. Applied Diagrams

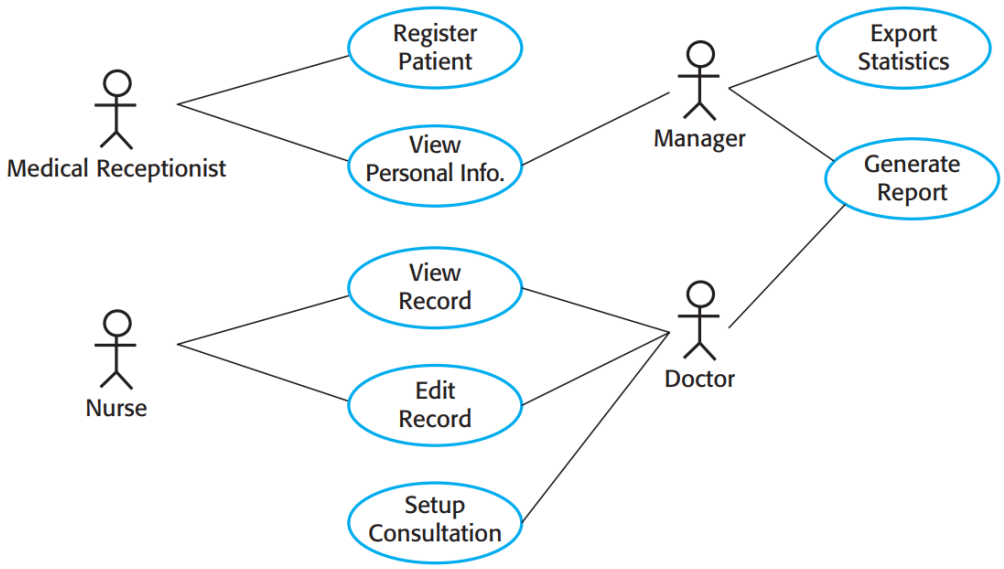
2.2.1. UML



[Figure 1] Hierarchy of UML 2.5 Diagrams

System modeling is the process of developing an abstract model of a system, with different perspectives on the system model. It is also called an object-oriented modeling language because it is closely related to an object-oriented language. UML (Unified Modeling Language) is a language that defines, visualizes, and documents software systems, task modeling, and output of systems, but is not a programming language. UML has many diagram types and supports the creation of many different types of system model. The reason for using UML is that visual modeling is becoming essential to make it easier to understand complex systems that are difficult to understand at once by designing models before building software systems. UML is also essential for designing systems to want and sound communication with team members in the project.

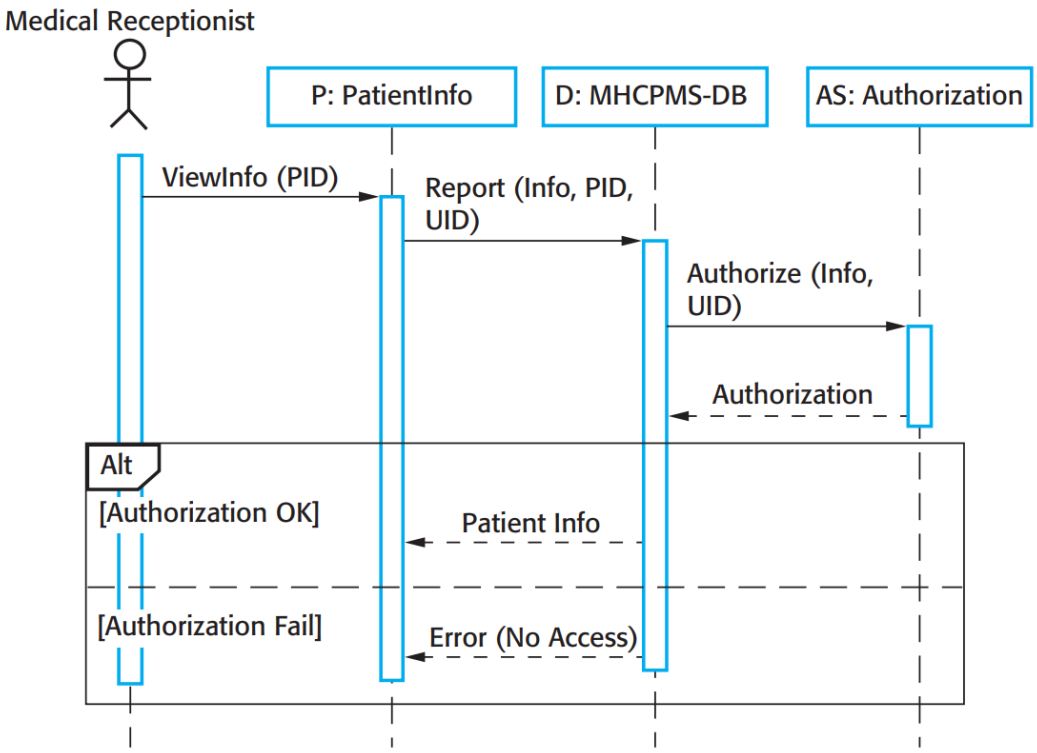
2.2.2. Use Case Diagram



[Figure 2] Use cases for the MHC-PMS

USE-CASE DIAGRAM of UML is a diagram that specifies the features or services that the system must provide. USE-CASE DIAGRAM is characterized by a focus on interaction between users and systems. Note that in USE-CASE DIAGRAM, DIAGRAM should not be drawn in the order of function, as it does not represent the flow. Also, each use case should be documented with a textual description. These can then be linked to other models in the UML that will develop the scenario in more detail. The set of use cases represents all possible interactions that will be described in the system requirements. Use cases are documented using a high-level use case diagram because it is difficult to provide detailed information about all possible cases between the system and the users with only use cases. In USE-CASE DIAGRAM, an actor is a person who is outside the system and interacts with the system (who uses the function of the system), and a system (another system that provides information to the system). Use-case is a function of the system viewed from the user's point of view, and represents the requirements of the system with the function that the system must provide to the actor. Mark the mark with an ellipse and fill in the name of the case. Lines link the actors with the interaction. Optionally, arrowheads may be added to lines to show how the interaction is initiated. A Use case Diagram shows various use cases and different types of users. The main purpose of Use case Diagram is to visualize the functional requirements of the system and it allows customers and developers to coordinate their opinions on the requirements.

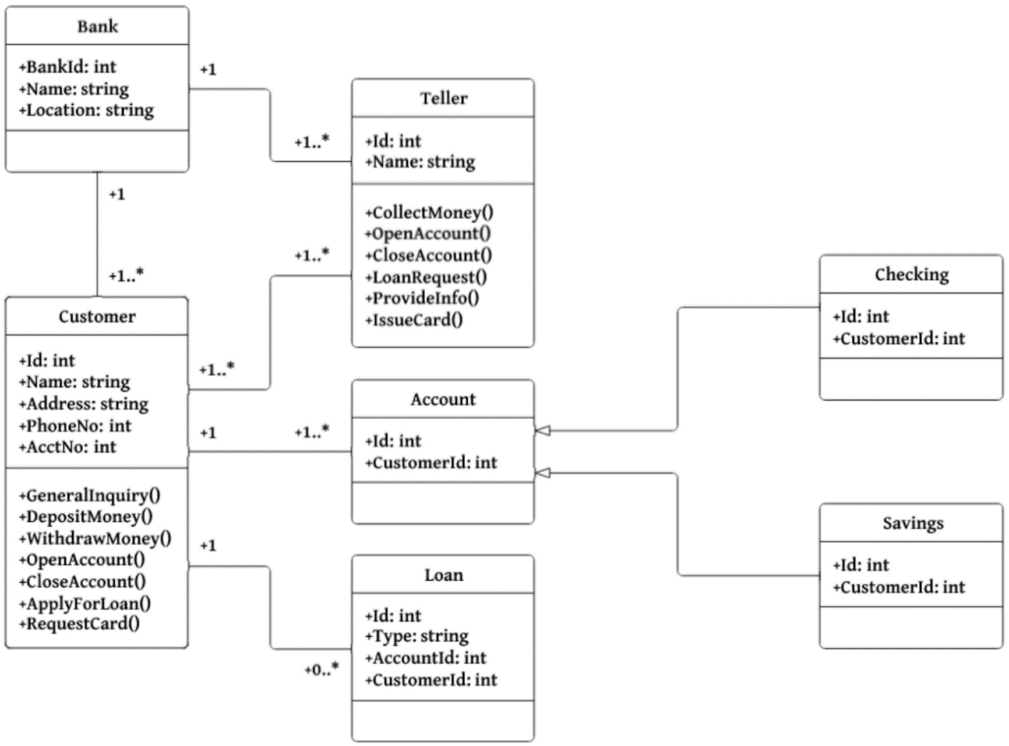
2.2.3. Sequence Diagram



[Figure 3] Sequence diagram for View patient information

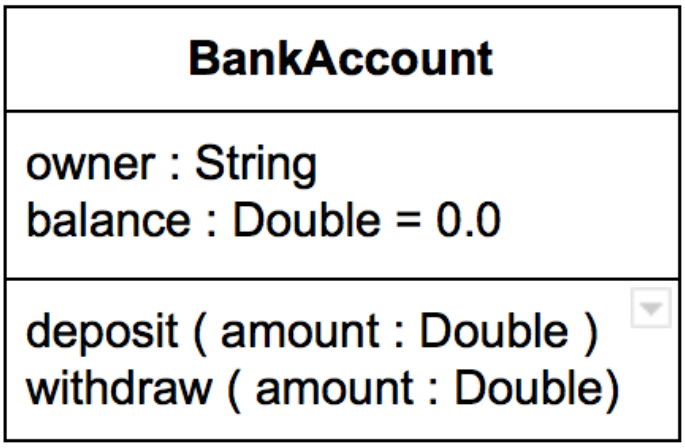
A sequence diagram or system sequence diagram (SSD) shows object interactions arranged in time sequence in the field of software engineering. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. As the name implies, a sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The components of a sequence diagram are actors, objects, lifelines, experimental boxes, messages, object extinction, and frames. A dotted lifeline is a line that indicates how long an actor or object is present. An actor or object is marked with a rectangle above the dotted line. The memory is alive up to where the dotted line is drawn, and the object is marked with an x table. The annotations on the arrows indicate the calls to the objects, their parameters, and the return values. Sequence Diagram can be used to model dynamic aspects of the system by depicting data exchanges, message sequences, etc. between each object for a specific use case or part of a specific use case, and it is useful for viewing the flow of the system at runtime.

2.2.4. Class Diagram



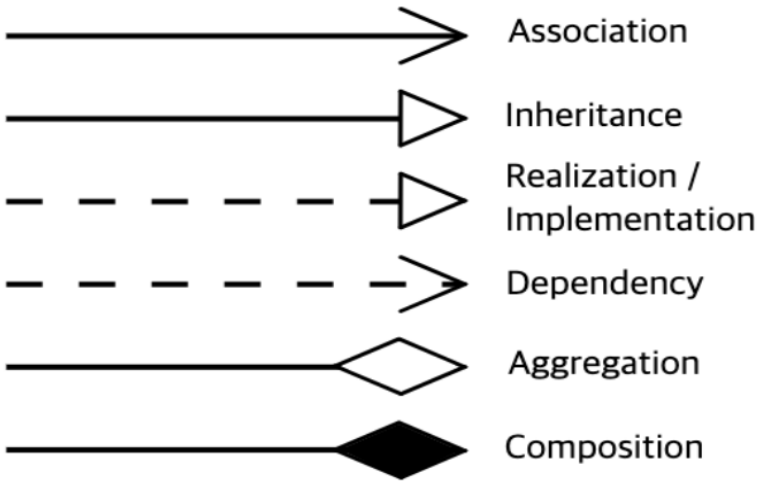
[Figure 4] Classes and associations in Class Diagram

Class diagrams are structural diagrams that schematize the internal components of the class and the relationship between the classes to structure a specific module or part of the system, and the whole. Before development, drawing a class diagram makes it easier to understand the dependence between classes in the system and communicate with team members. Depending on the purpose of the class diagram, it can be divided into concepts, specifications, and implementation stages. At the conceptual stage, the objective is to derive only classes and simplify relationships. Specifications and implementation phases are related to code because they are used for explanatory purposes after design or implementation just prior to development and are implemented as code based on this diagram or drawn as a diagram based on the code.



[Figure 5] Class with 3 components

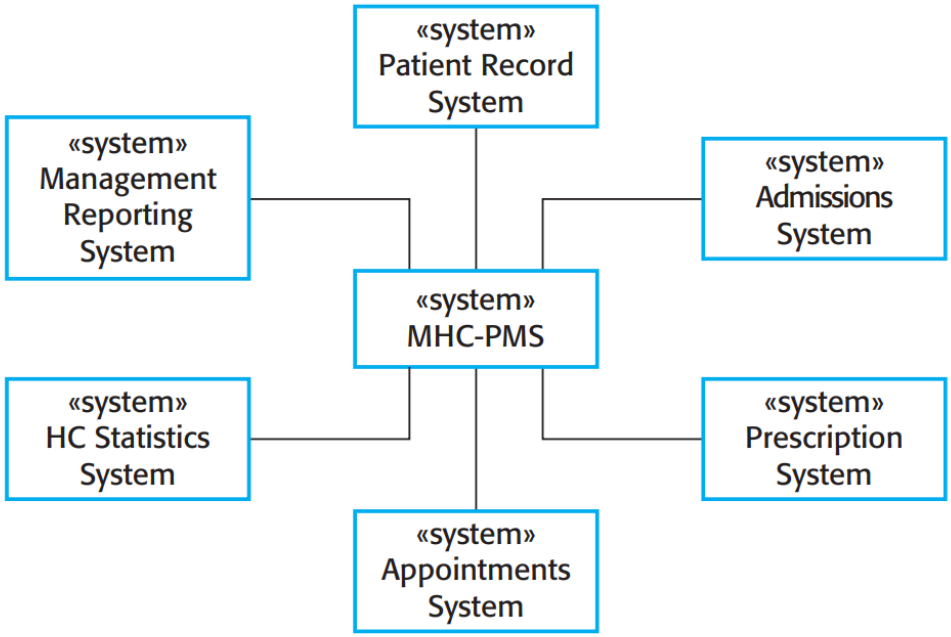
In the Class Diagram, classes are represented with boxes that contain three sections: The name of the object class is in the top section. The class attributes are in the middle section. This must include the attribute names and, optionally, their types. The operations (or methods) associated with the object class are in the lower section of the rectangle.



[Figure 6] Class with 3 components

Association is a relationship between two separate classes. It joins two entirely separate entities. There are four different types of association: bi-directional, uni-directional, aggregation (includes composition aggregation) and reflexive. Bi-directional and uni-directional associations are the most common ones. This can be specified using multiplicity (one to one, one to many, many to many, etc.).

2.2.5. Context Diagram



[Figure 7] The context of the MHC-PMS

Context Diagrams in engineering 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 is a high-level view of system. However, they do not show the types of relationships between the systems in the environment and the system that is being specified. Context Diagrams represent all external entities that may interact with the system, and it is a high-level view of the system. The objective of the Context Diagram is to focus attention on external factors and events that should be considered in developing a complete set of systems requirements and constraints.

2.3. Applied Tools

2.3.1. Microsoft PowerPoint

Microsoft PowerPoint is well known as a tool for creating presentations. It is convenient to draw various types of diagrams by freely placing text and figures. It has the advantages of being easy to access and convenient to use and edit.

2.3.2. App.diagrams.net

It is a web-based site where you can draw diagrams, and although there are various tools, you

can draw them quickly and easily.

2.4. Project Scope

More and more people are studying or doing hobbies while reducing their sleep in busy daily lives. Therefore, there are many people who want better quality sleep. Accordingly, it provides a smart-sleep-system that can adjust the smart device (furniture) of the house using a smartphone or a smart watch to adjust the appropriate temperature, humidity, and brightness. The specific project scope is as follows: By default, A smart sleep system is a system for a single user. The surrounding devices are controlled for optimal sleep of one user who writes the application. Users can only be used by users registered by the administrator. The reason is that when multiple people are in the same space, it is difficult to match because their optimal environmental conditions (temperature, humidity, brightness) are different. Systems that will be improved in the future will need to address this problem so that our system can be applied to a large number of users.

2.5. References

The user of this SDS may need the following documents for reference:

- Team 16, 2021 Fall. Software Design Document, SKKU.
- Ian Sommerville, *Software Engineering.*
- Wikipedia

3. System Architecture - Overall

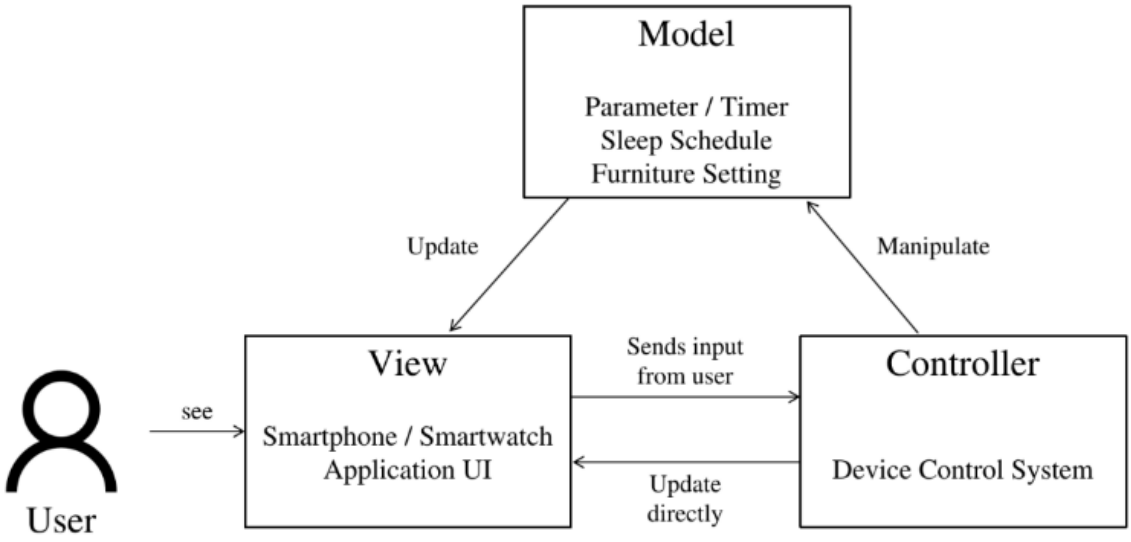
3.1. Objectives

We describe and show the overall system architecture, context diagram, sequence diagram and use case diagram in this chapter.

3.2. System Organization

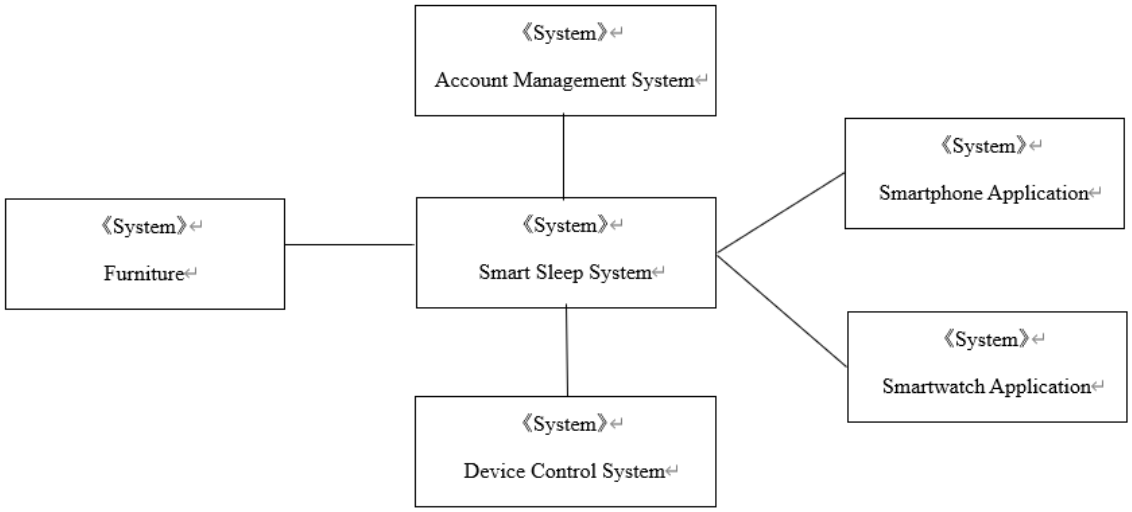
The system architecture can be shown by using MVC (Model – View – Controller) pattern. The user may create a desired sleep environment through the smartphone and smartwatch UI, and

may check a list of devices registered in the user's space. Through each registered device, the user may maintain the devices at a desired setting value or delete the registered device. The alarm system, which is a key part of our system, operates at a time set by the user himself or based on a database that analyzes the user's sleep pattern, which gives the user the best sleep experience.



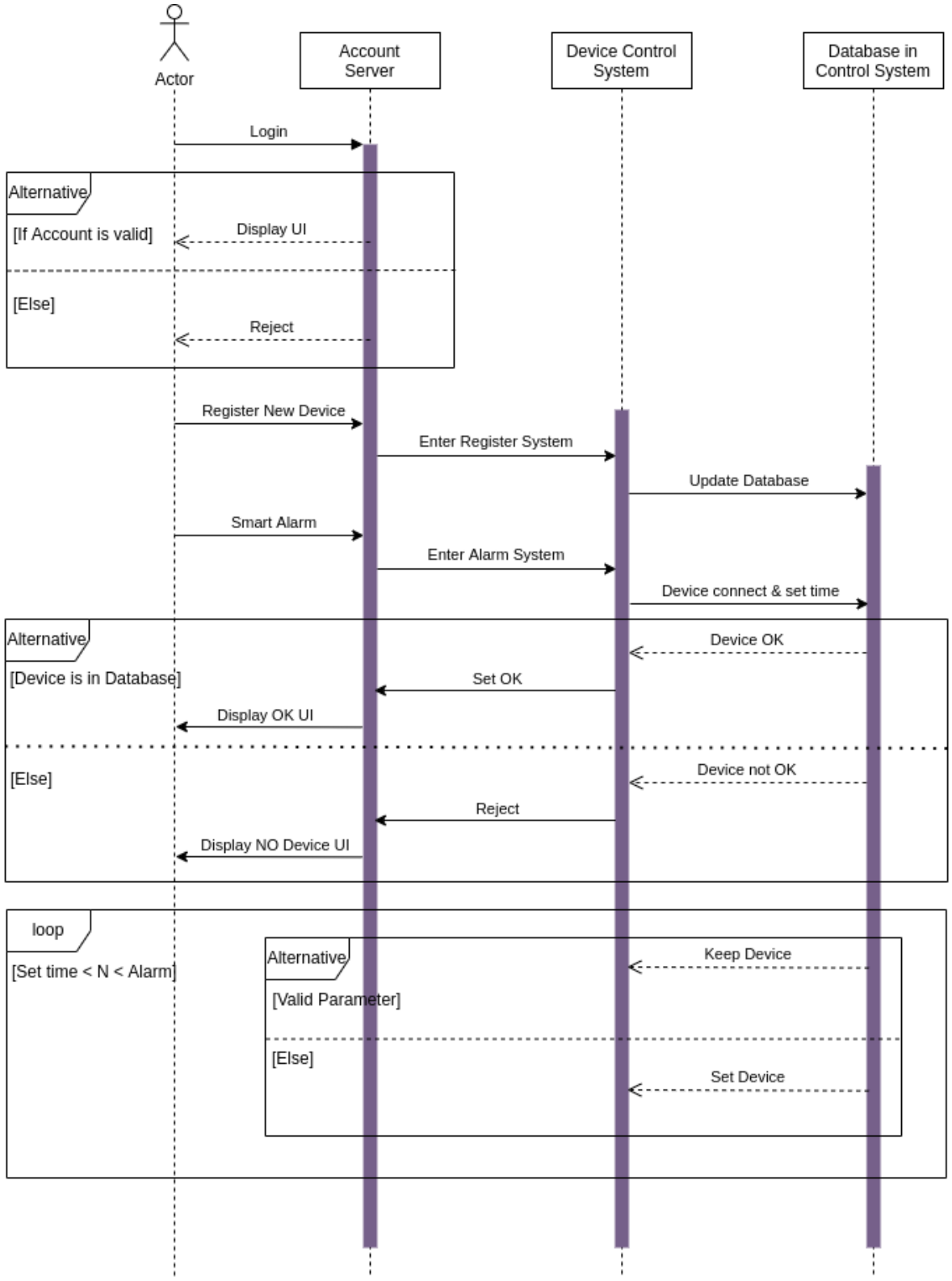
[Figure 8] Overall system architecture

3.2.1. Context Diagram



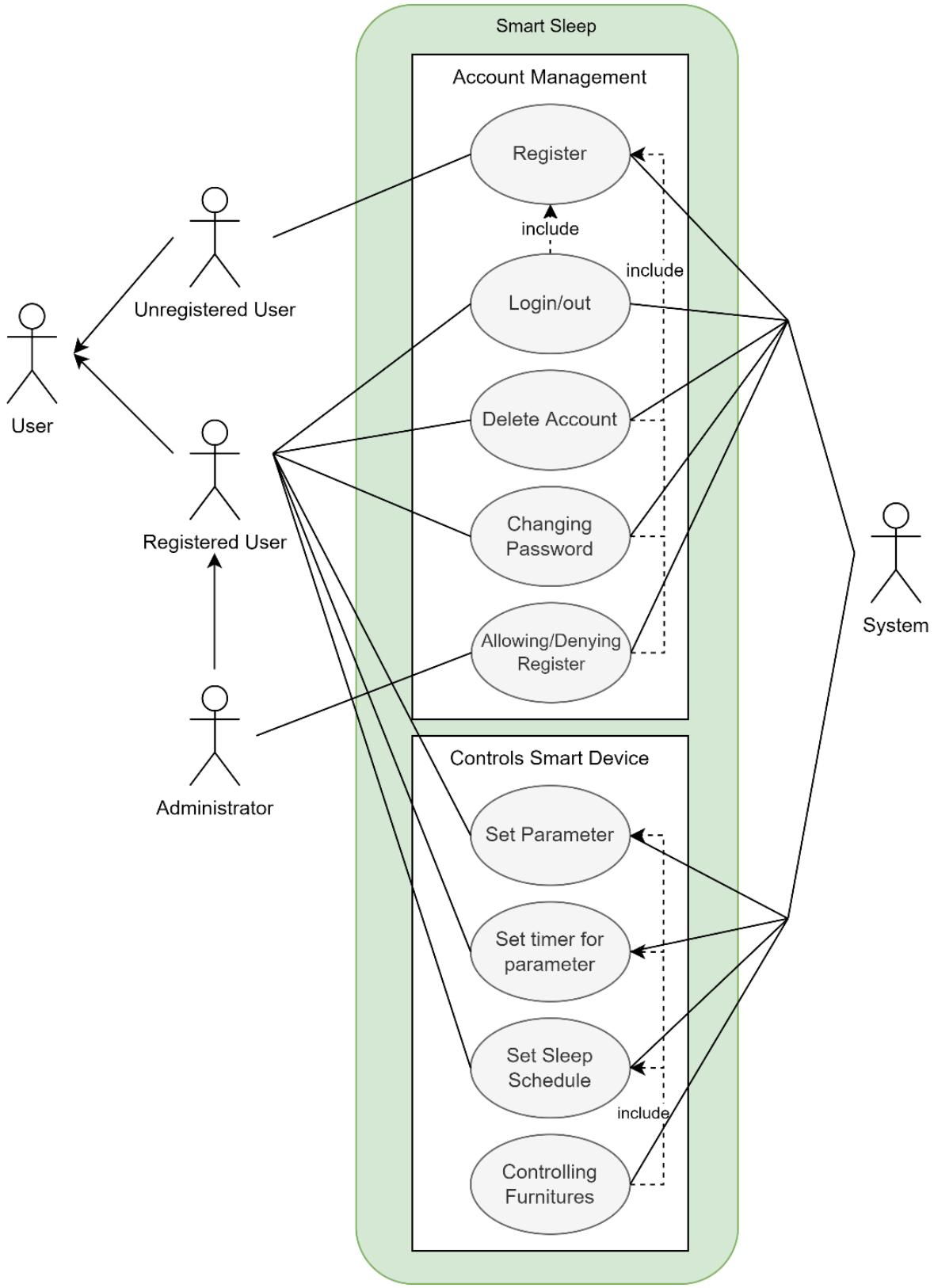
[Figure 9] Overall context diagram

3.2.2. Sequence Diagram



[Figure 10] Overall Sequence Diagram

3.2.3. Use Case Diagram



[Figure 11] Use case diagram

4. System Architecture - Frontend

4.1. Objectives

In this section, the frontend system is divided in components. The structure and the relation between components are described.

4.2. Subcomponents

4.2.1. Account

The user creates an account introducing an id. He has the option of recovering his password or id. Log in functionality is also covered.

4.2.1.1. Attributes

Attributes of SignUp objects:

- **Id:** identificator for accounts
- **Password:** password chosen by the user
- **email:** email address linked with the account
- **phone:** phone number linked with the account

Attributes of LogIn objects:

- **Id:** identificator for accounts
- **Password**: password chosen by the user

Attributes of RecoverData objects:

- **email**: email address linked with the account

4.2.1.2. Methods

Methods of SignUp class:

- CreateId()
- ShowErrorMessage()

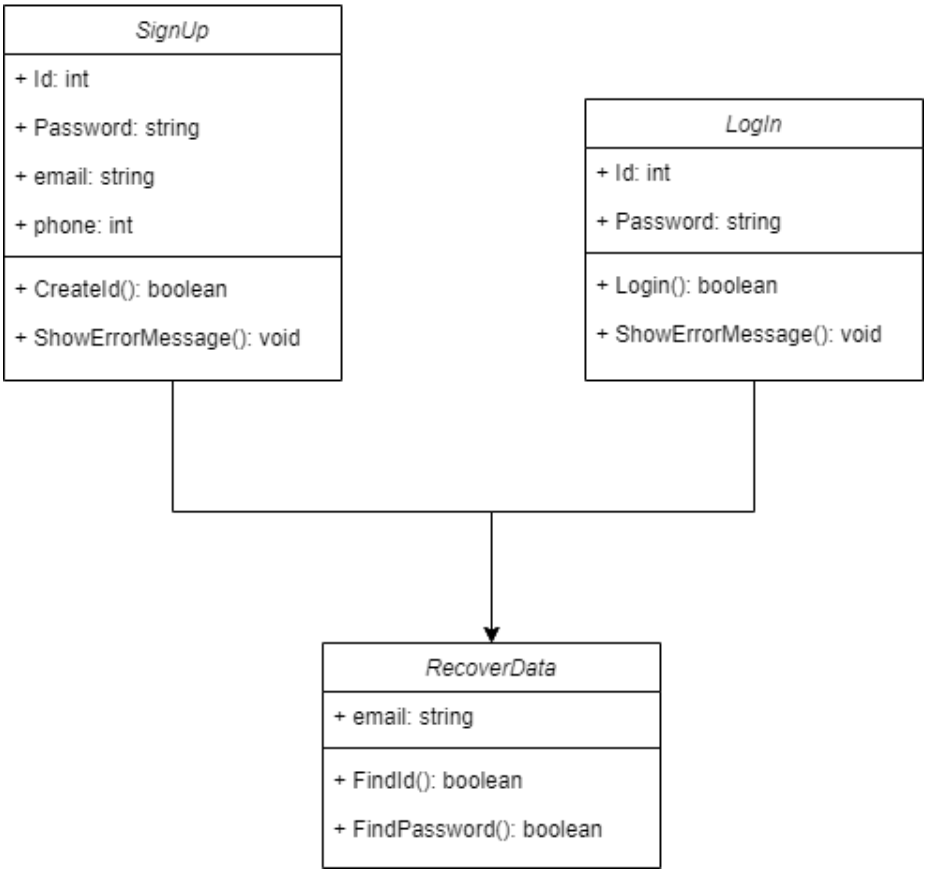
Methods of LogIn class:

- Login()
- ShowErrorMessage()

Methods of RecoverData class:

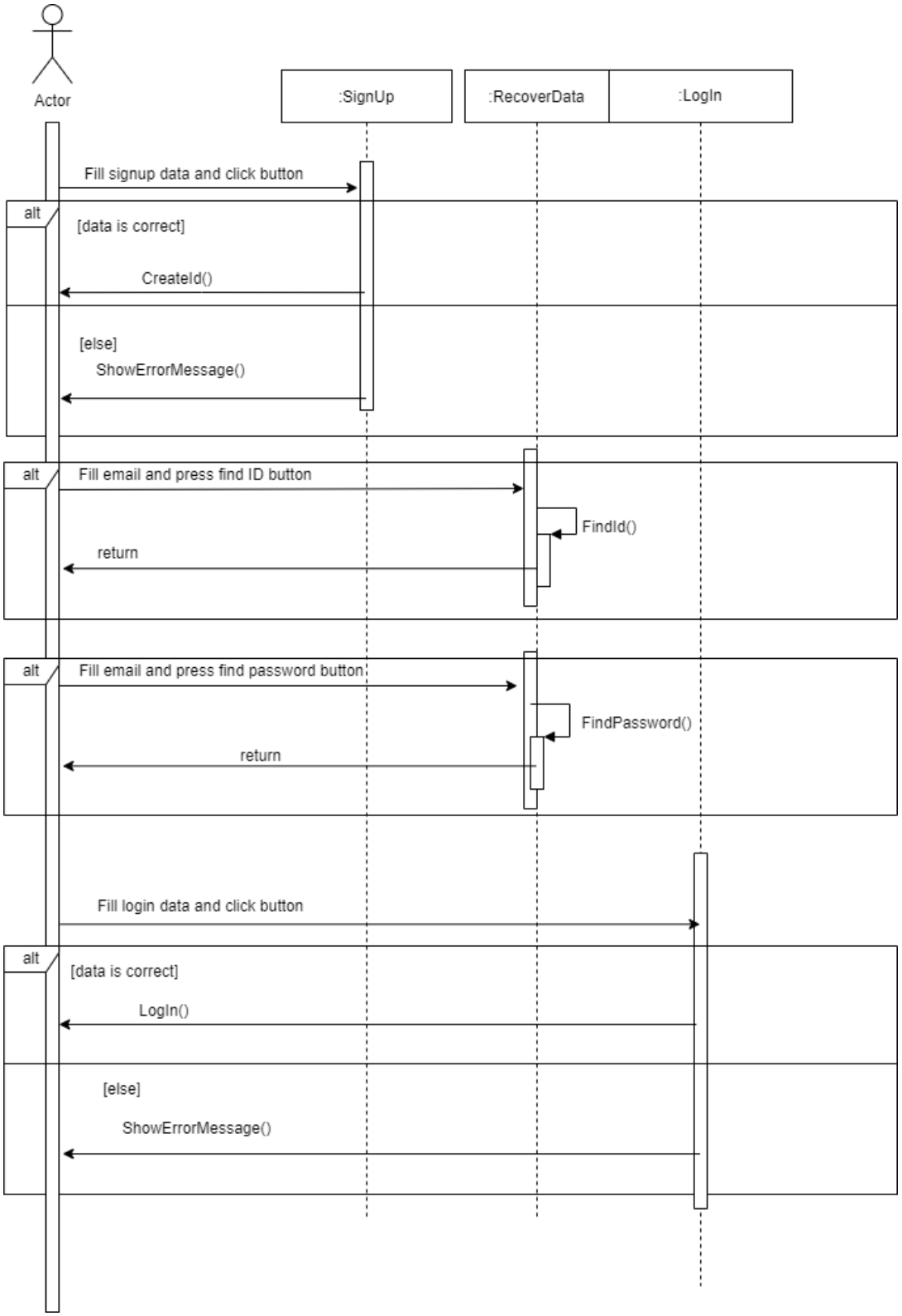
- FindId()
- FindPassword()

4.2.1.3. Class Diagram



[Figure 12] Class diagram – Account

4.2.1.4. Sequence Diagram



[Figure 13] Sequence diagram – Account

4.2.2. Register new smart device

Register new smart device allows to add to the working system a new device, which is identified by the verification code attached to it.

4.2.2.1. Attributes

Attributes of Register Device objects:

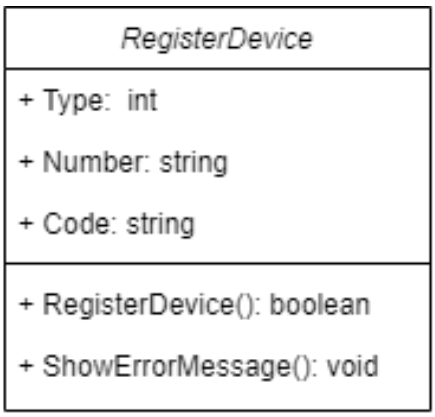
- **Type**: smart light, smart temperature, mart humidity
- **Number**: identificator of the device
- **Code**: second identificator of the device

4.2.2.2. Methods

Methods of Register Device class:

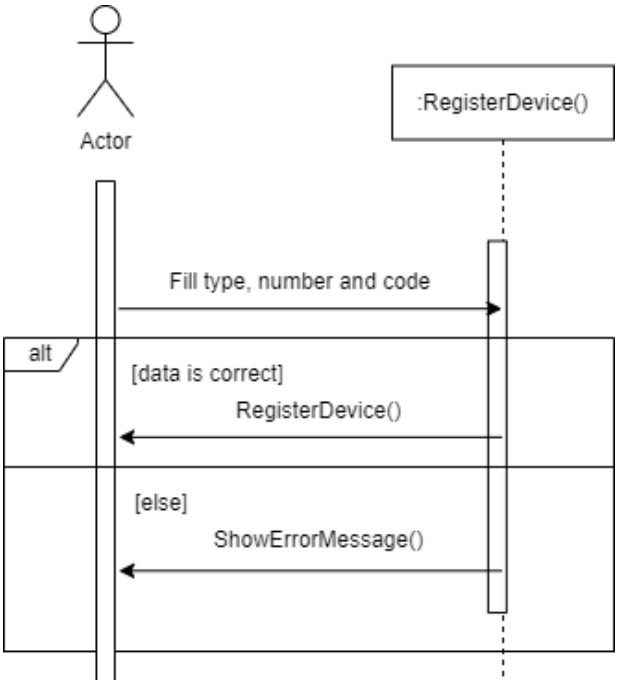
- RegisterDevice()
- ShowErrorMessage()

4.2.2.3. Class Diagram



[Figure 14] Class diagram – Register Device

4.2.2.4. Sequence Diagram



[Figure 15] Sequence diagram – Register Device

4.2.3. Smart Alarm

Smart alarm system is largely divided into two types. One is that the user may directly set the alarm time. The other is that the system recommends the optimal sleep time for the user. Based on the user's sleep pattern and input information, the optimal sleep time is recommended and the alarm time is set.

4.2.3.1. Attributes

Attributes of Smart Alarm objects:

- **Time**: Show the set alarm time
- **Button**: Enter time setting mode or time recommendation mode

Attributes of Set Time objects:

- **Time**: User sets alarm time

Attributes of Recommend Time objects:

- **Time**: Show the recommended time
- **Parameter**: Provides information on optimal sleep patterns for user

4.2.3.2. Methods

Methods of Smart Alarm class:

- ShowTimeOver()

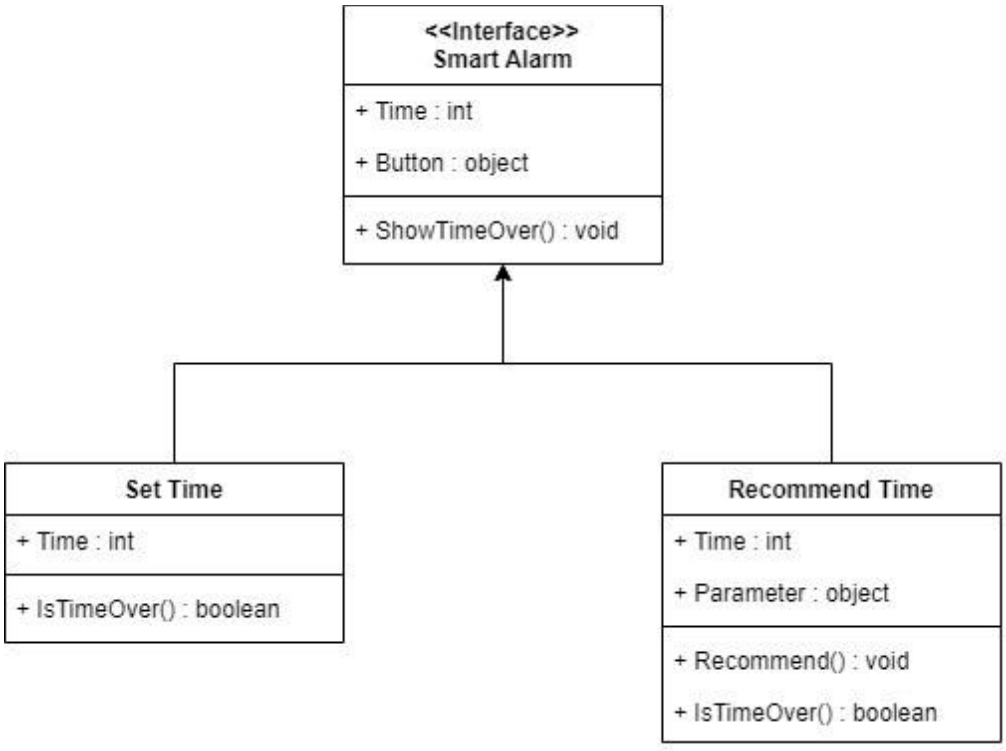
Methods of Set Time class:

- IsTimeOver()

Methods of Recommend Time class:

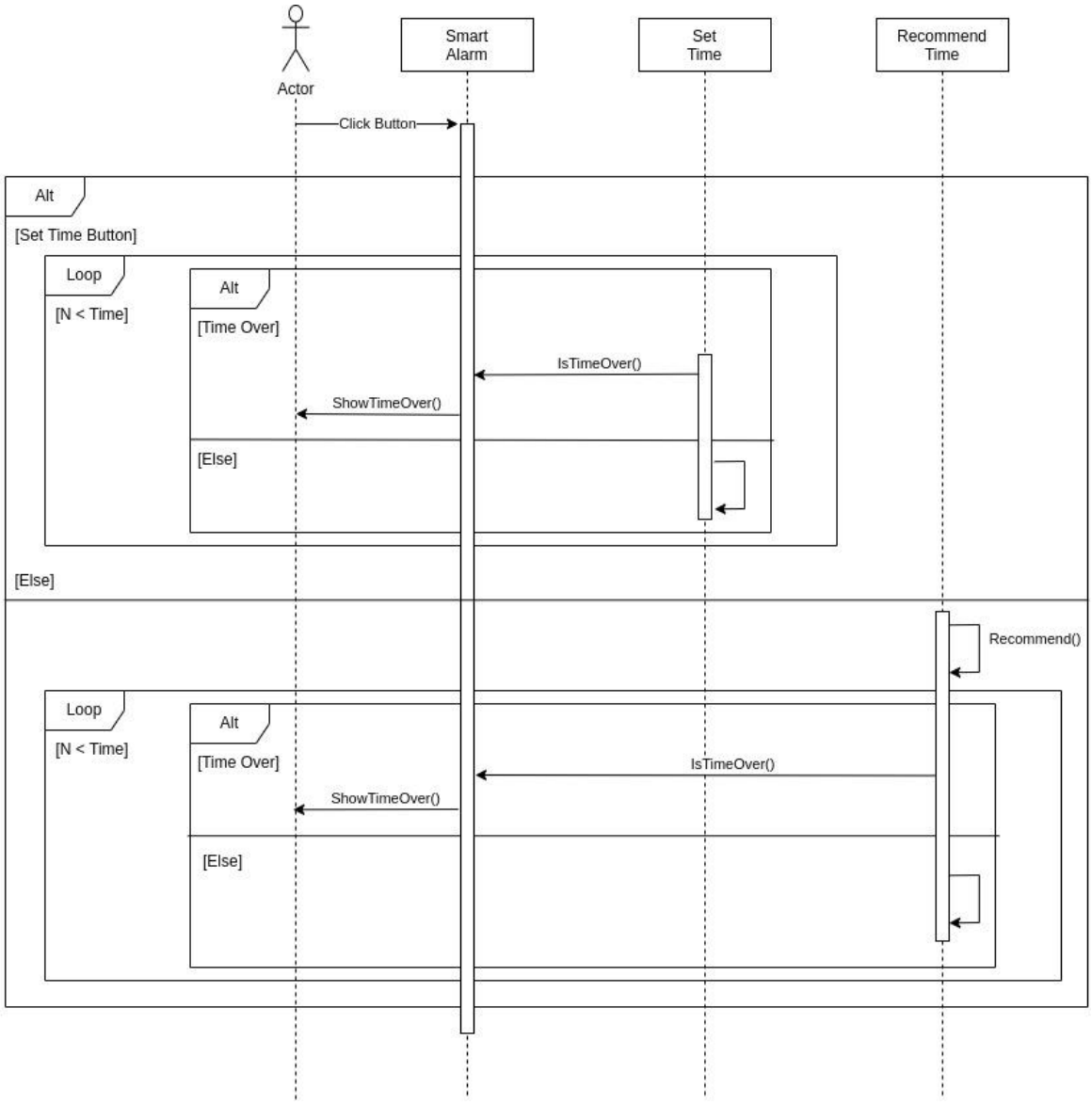
- Recommend()
- IsTimeOver()

4.2.3.3. Class Diagram



[Figure 16] Class diagram – Smart Alarm

4.2.3.4. Sequence Diagram



[Figure 17] Sequence diagram – Smart Alarm

4.2.4. Smart Light

Smart light system adjusts the ambient brightness so that the user can sleep in an optimal environment.

4.2.4.1. Attributes

Attributes of Smart Light objects:

- **Brightness:** Show the current brightness to the user

Attributes of Optimized Light objects:

- **OptimizedBrightness:** Brightness value recommended by the optimal scheduler
- **Parameter:** Provides information on optimal sleep patterns for user

4.2.4.2. Methods

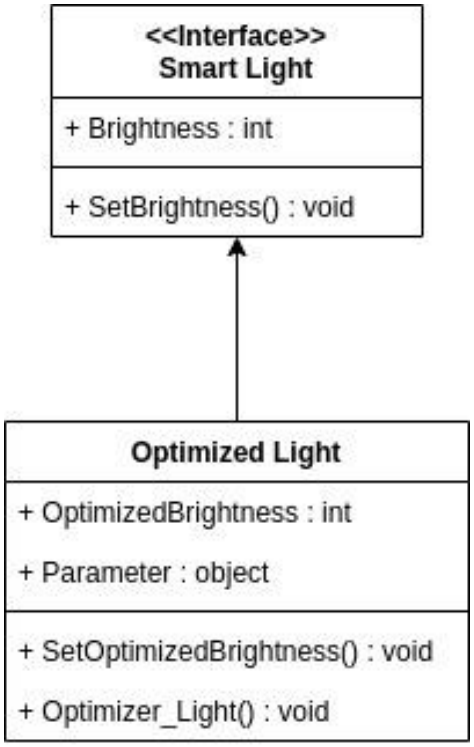
Methods of Smart Light class:

- SetBrightness()

Methods of Optimized Light class:

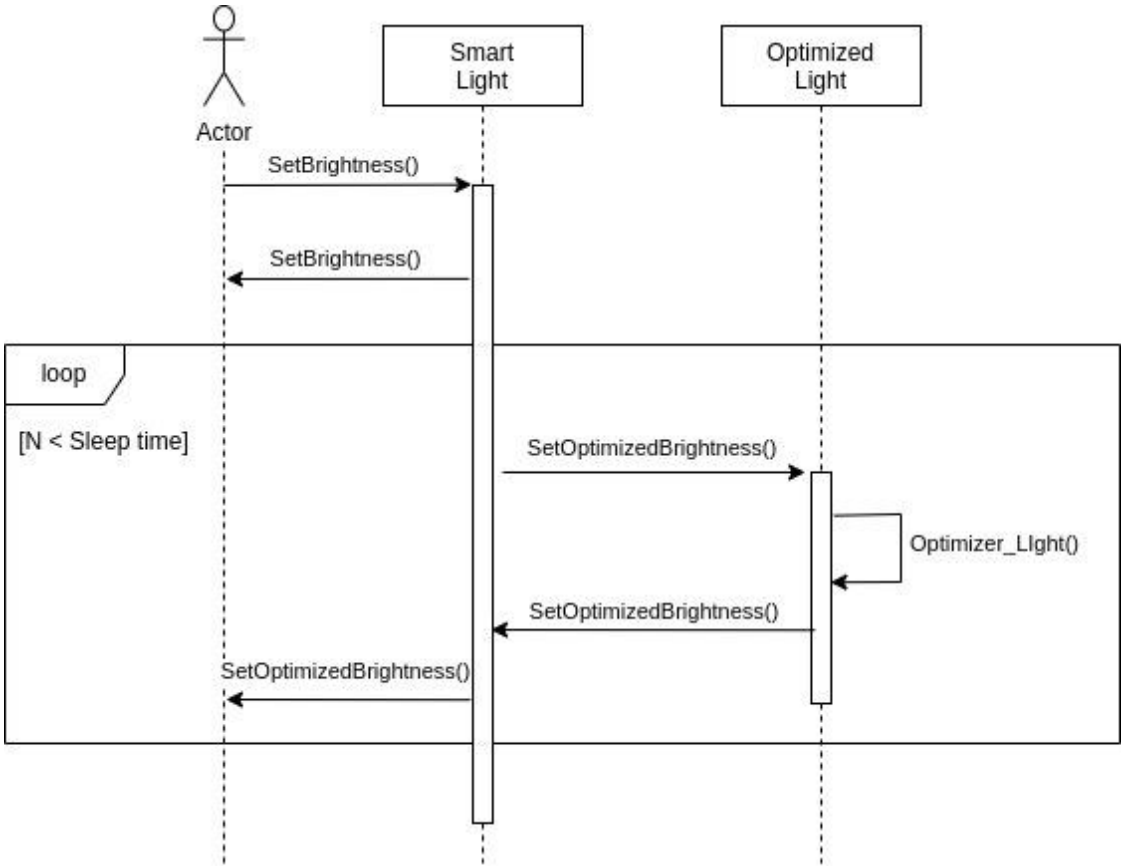
- SetOptimizedBrightness()
- Optimizer_Light()

4.2.4.3. Class Diagram



[Figure 18] Class diagram – Smart Light

4.2.4.4. Sequence Diagram



[Figure 19] Sequence diagram – Smart Light

4.2.5. Smart Temperature / Humidity

Smart temperature / humidity system adjusts the appropriate temperature and humidity so that the user can sleep in an optimal environment.

4.2.5.1. Attributes

Attributes of Smart Temperature / Humidity objects:

- **Temperature**: Show the current temperature to the user
- **Humidity**: Show the current humidity to the user

Attributes of Optimized Temperature objects:

- **OptimizedTemperature**: Temperature value recommended by the optimal scheduler

- **Parameter**: Provides information on optimal sleep patterns for user

Attributes of Optimized Humidity objects:

- **OptimizedHumidity**: Humidity value recommended by the optimal scheduler
- **Parameter**: Provides information on optimal sleep patterns for user

4.2.5.2. Methods

Methods of Smart Temperature / Humidity class:

- SetTemperature()
- SetHumidity()

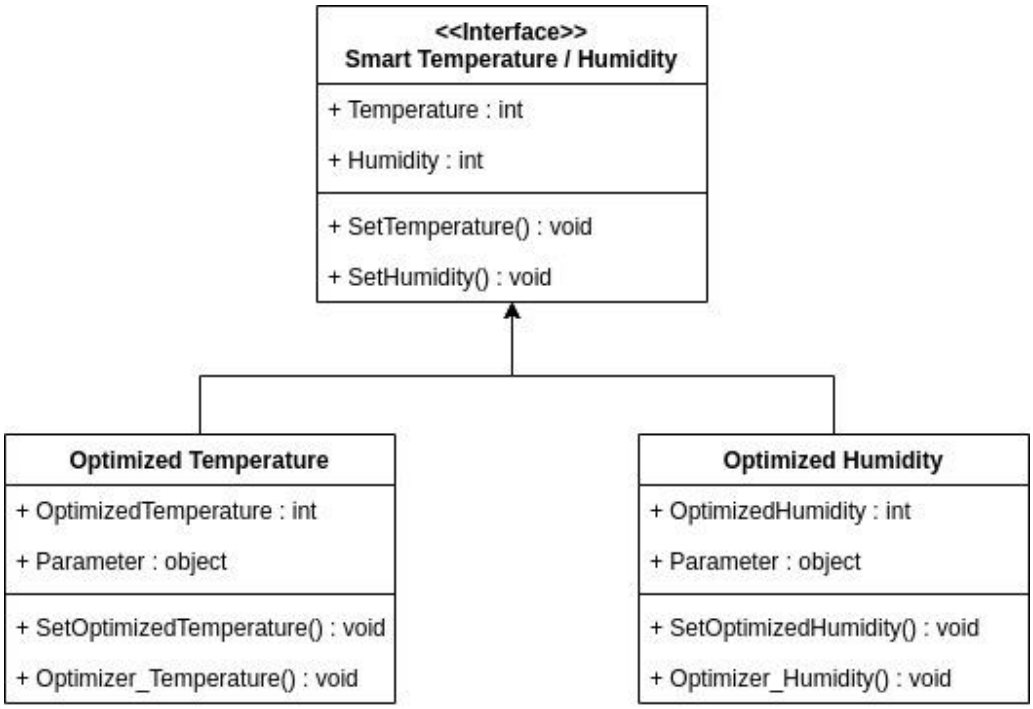
Methods of Optimized Temperature class:

- SetOptimizedTemperature()
- Optimizer_Temperature()

Methods of Optimized Humidity class:

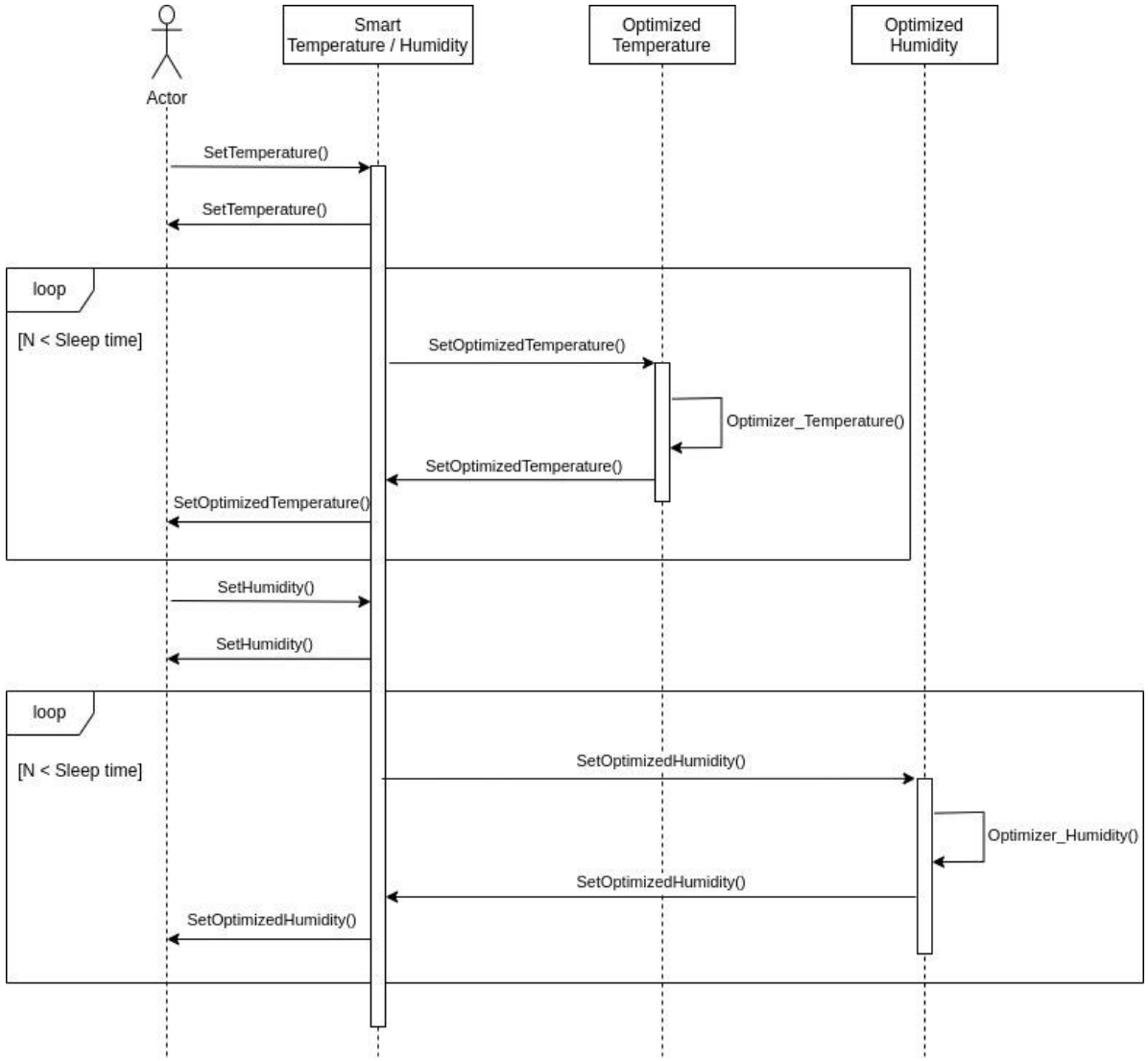
- SetOptimizedHumidity()
- Optimizer_Humidity()

4.2.5.3. Class Diagram



[Figure 20] Class diagram – Smart Temperature / Humidity

4.2.5.4. Sequence Diagram



[Figure 21] Sequence Diagram – Smart Temperature / Humidity

5. System Architecture - Backend

5.1. Objectives

In this section, the backend system is divided in components. The structure and the relation between components are described.

5.2. Subcomponents

5.2.1. Furniture Control

The furniture control is the object in charge of controlling the smart devices/furniture devices (i.e., light bulb, thermostat, and humidity controller, etc.). When the furniture control is triggered -either by the scheduler or the user- the furniture control connects to the furniture over the network and uses the device's API in order to adjust their parameters.

5.2.1.1. Attributes

These are the attributes that furniture control object has.

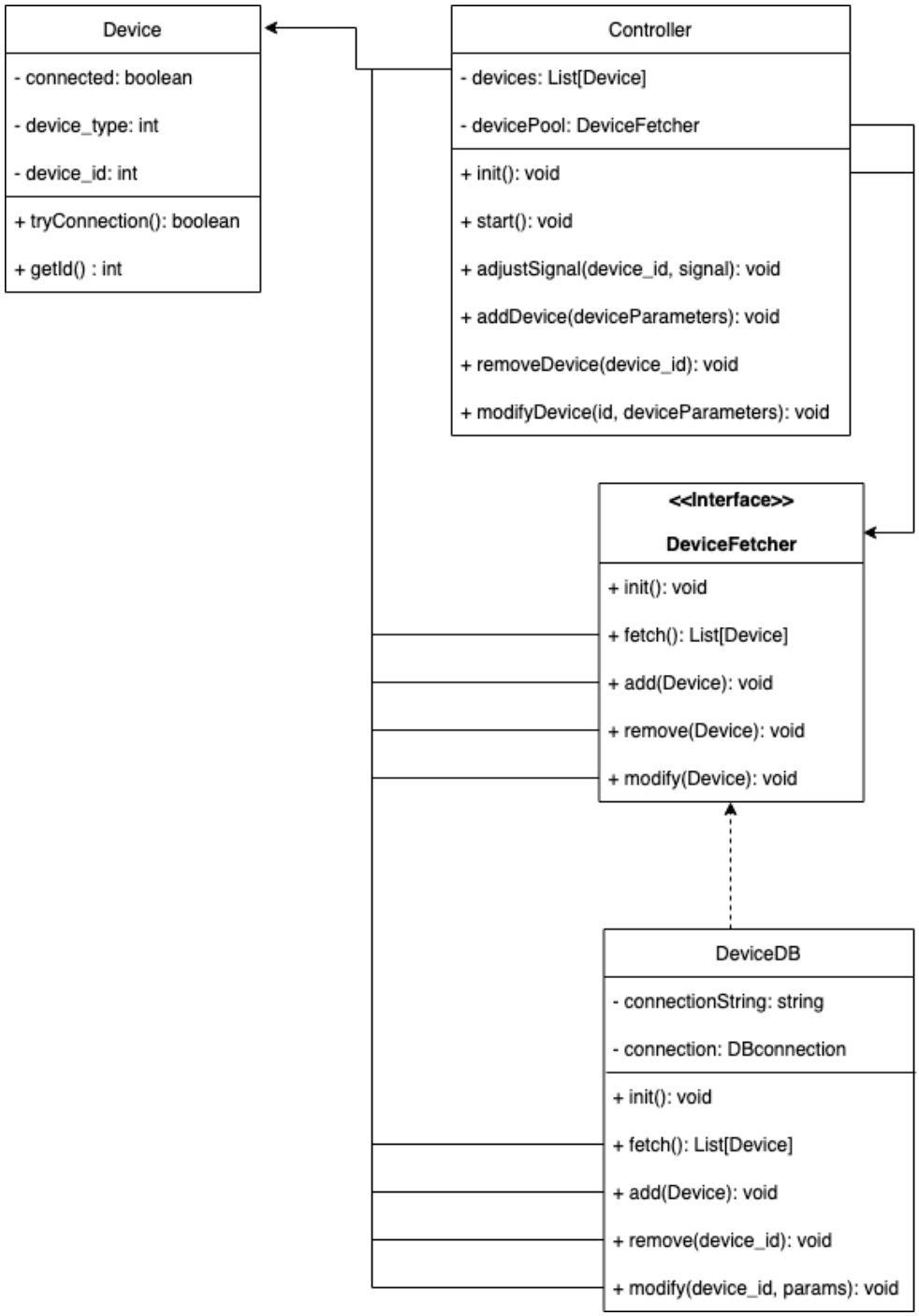
- **devices**: It is list of devices that are connected over to the smart system. The furniture control checks the list of devices and selects the appropriate one in order to adjust its parameters.

5.2.1.2. Methods

There following are the methods in the controller.

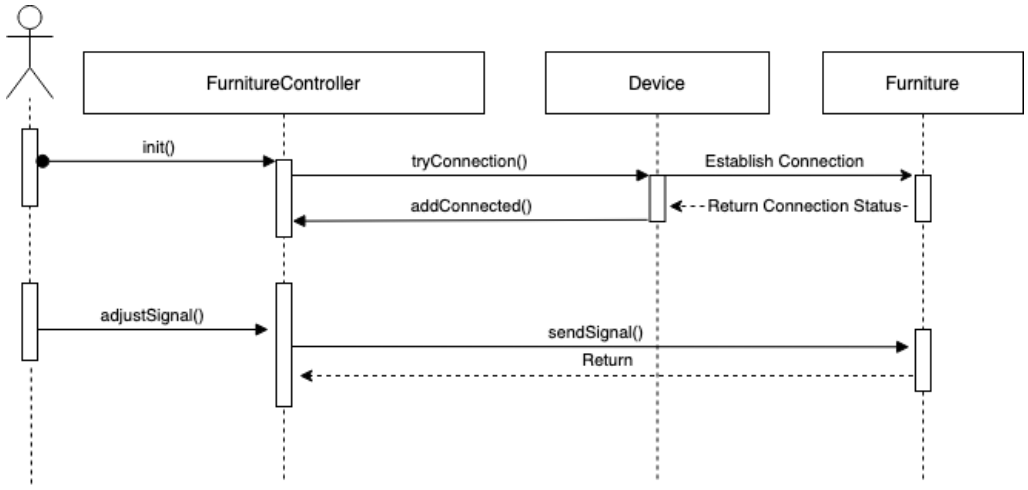
- init()
- adjustSignal()
- addDevice()
- removeDevice()
- modifyDevice()

5.2.1.3. Class Diagram



[Figure 22] Class diagram – Furniture Controller

5.2.1.4. Sequence Diagram



[Figure 23] Sequence diagram – Furniture Controller

5.2.2. User Action

This User Action class receives information from the front end and deals with the actions accordingly (i.e. registering an account, verifying and account, adding/removing/modifying a device, adjusting parameters, etc.)

5.2.2.1. Attributes

These are the attributes that the user profile has:

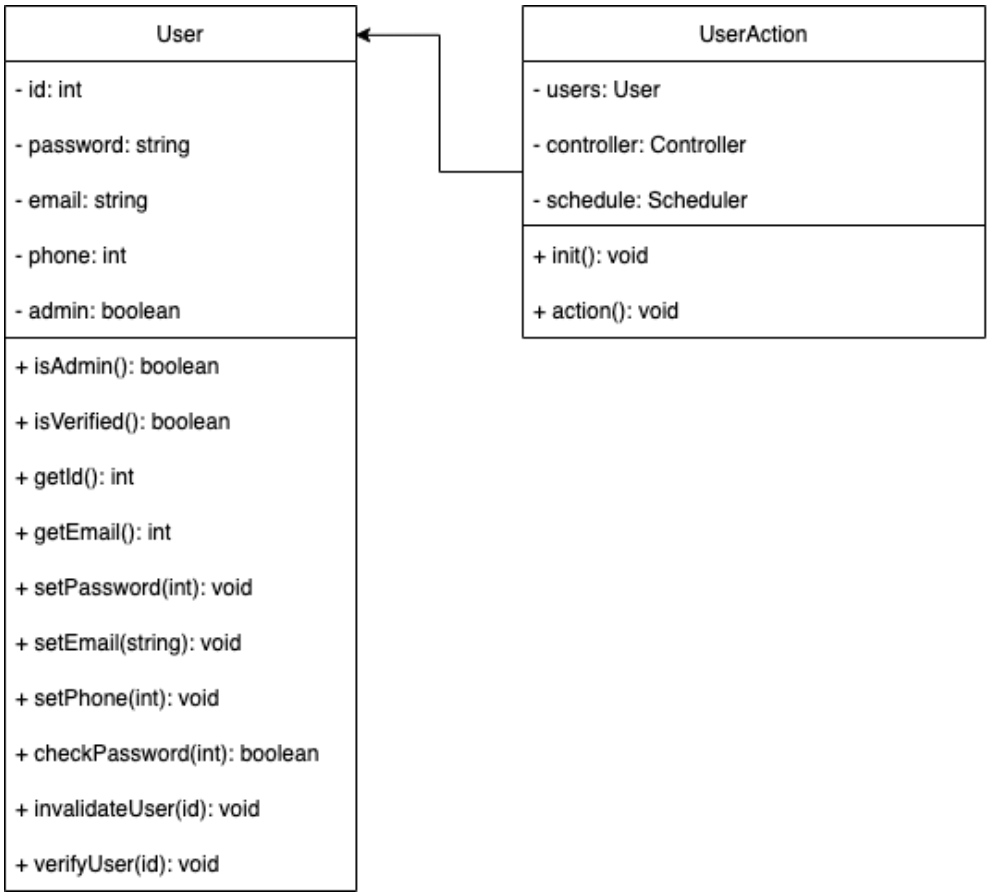
- **users**: an instance of the user class.
- **controller**: The instance of controller which is alive in system. To know what type it is, check 5.2.1.
- **scheduler**: The instance of scheduler which is alive in system. To know what type it is, check 5.2.4.

5.2.2.2. Methods

There are two methods in the user action.

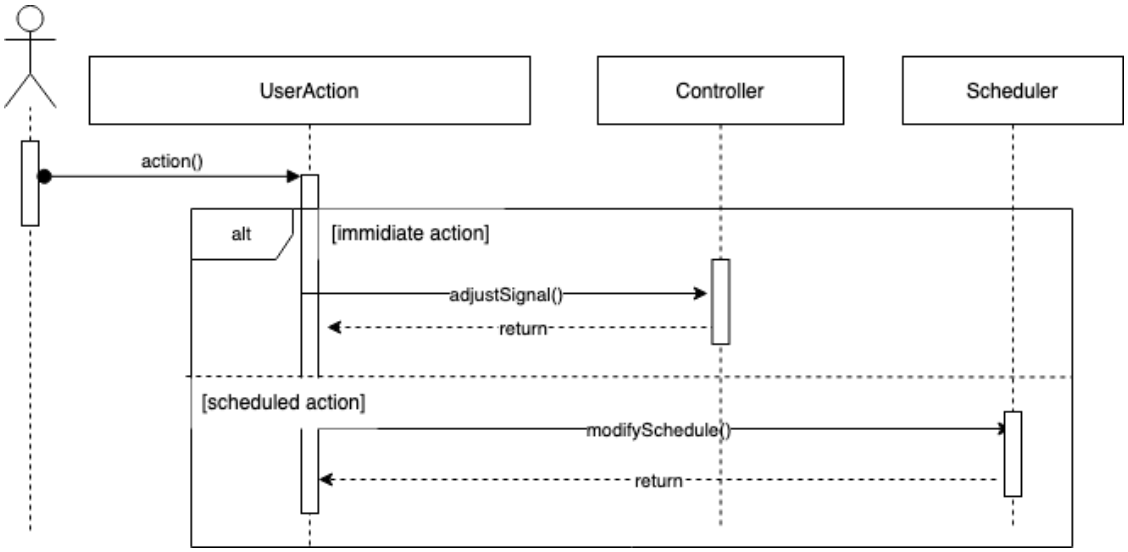
- init()
- action()

5.2.2.3. Class Diagram



[Figure 24] Class diagram – User Action

5.2.2.4. Sequence Diagram



[Figure 25] Sequence diagram – User Action

5.2.3. Account Registration Management

The registration is object that is created when the user tries to sign up. This object stores account information which user class also has. When it's accepted, new user is created in DB, when it's denied, nothing happens. In both ways, the registration object is deleted from DB.

5.2.3.1. Attributes

These are the attributes that Registration object has.

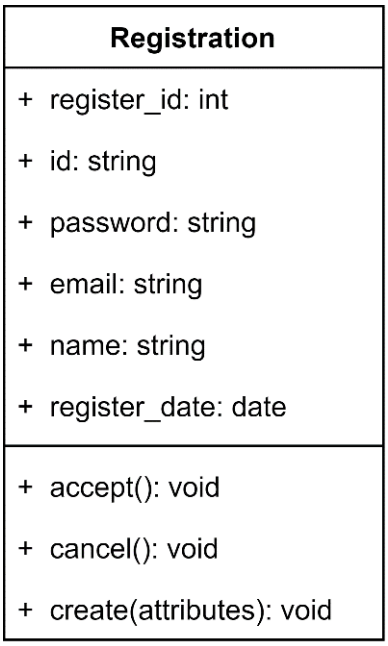
- **register_id**: The id of registration. It is unique value.
- **id**: The user id.
- **password**: The user password.
- **Email:** The user email.
- **Name:** User name
- **Register_date:** Date when registration is registered.

5.2.3.2. Methods

There are three methods in the Registration.

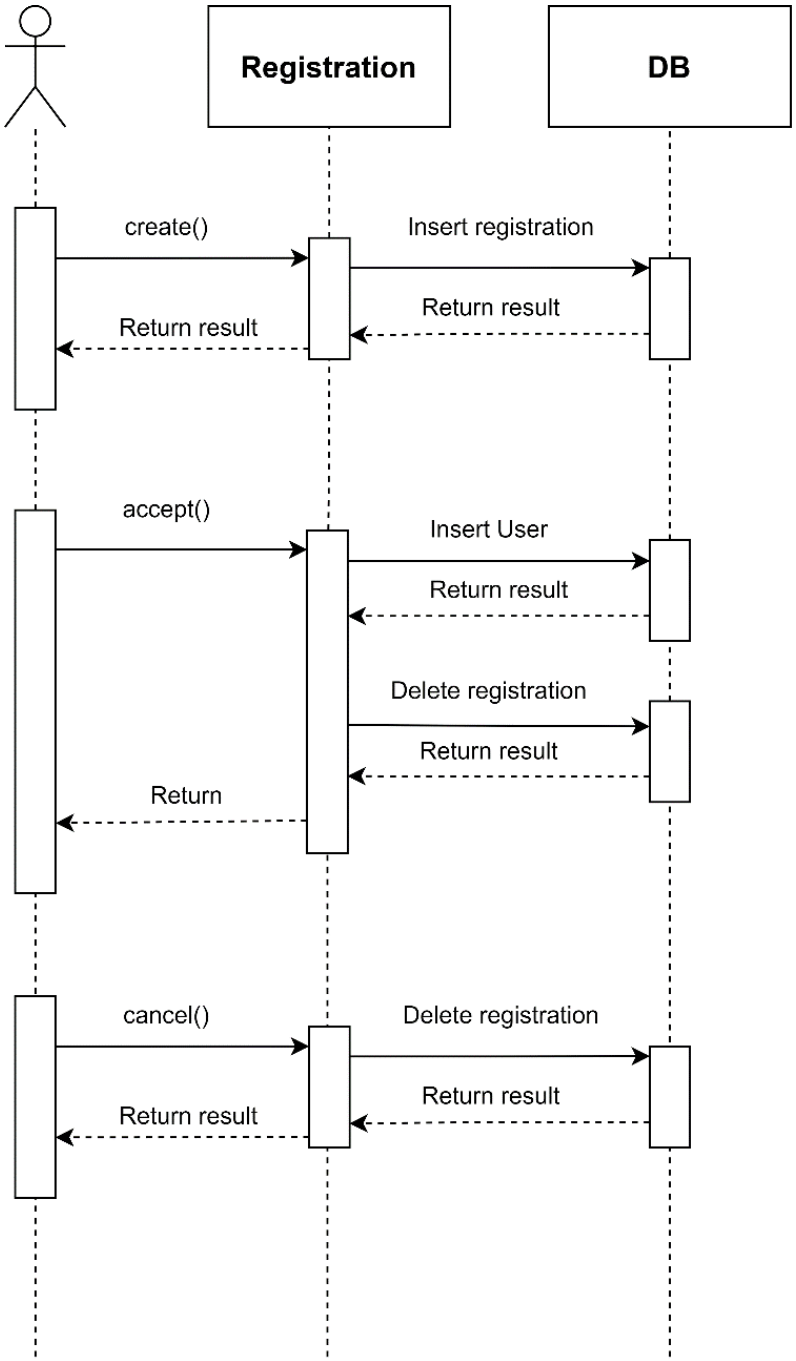
- accept()
- cancel()
- create(id, password, email, name)

5.2.3.3. Class Diagram



[Figure 26] Class diagram – Account Registration Management

5.2.3.4. Sequence Diagram



[Figure 27] Sequence diagram – Account Registration Management

5.2.4. Scheduler

The scheduler is object that is responsible for scheduling action in Smart Sleep. For checking time of schedule in real-time, the scheduler checks every registered schedule time every time

in infinite loop. The scheduler stores and fetches schedules via ScheduleFetcher interface. ScheduleFetcherDB is one of the implemented classes of ScheduleFetcher interface, which stores every schedule in database. Schedule class describes what schedule object is. Scheduler can check time and acquire control values of schedule by Schedule class methods.

5.2.4.1. Attributes

These are the attributes that Scheduler object has.

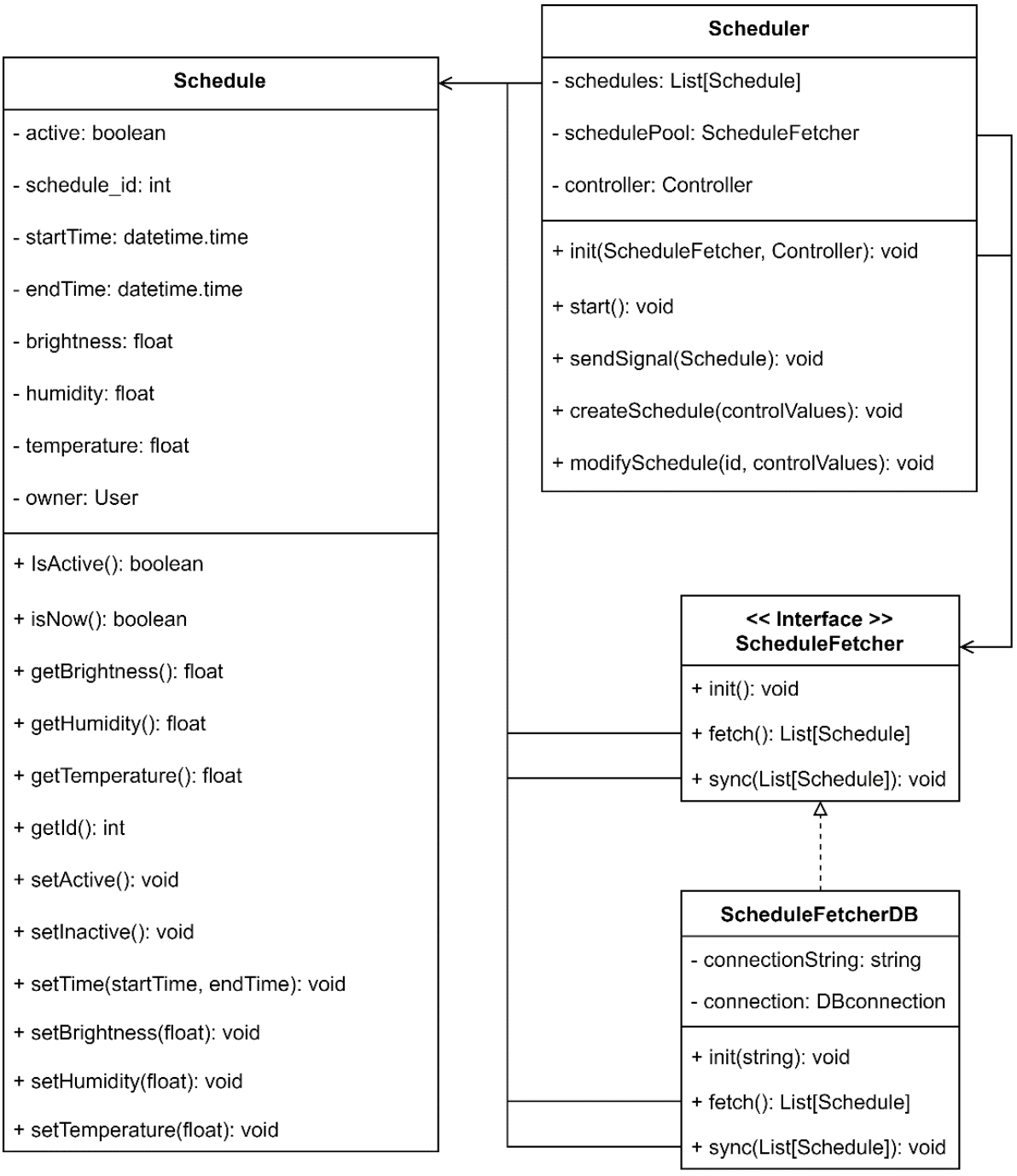
- **schedules**: It is list of schedules that scheduler has. The scheduler iterates over this list and check time of every schedule.
- **schedulePool**: This attribute points the source where the scheduler can fetches schedules. The type of it is scheduleFetcher interface.
- **controller**: The instance of controller which is alive in system. To know what type it is, check 5.2.1.

5.2.4.2. Methods

There are five methods in the Scheduler.

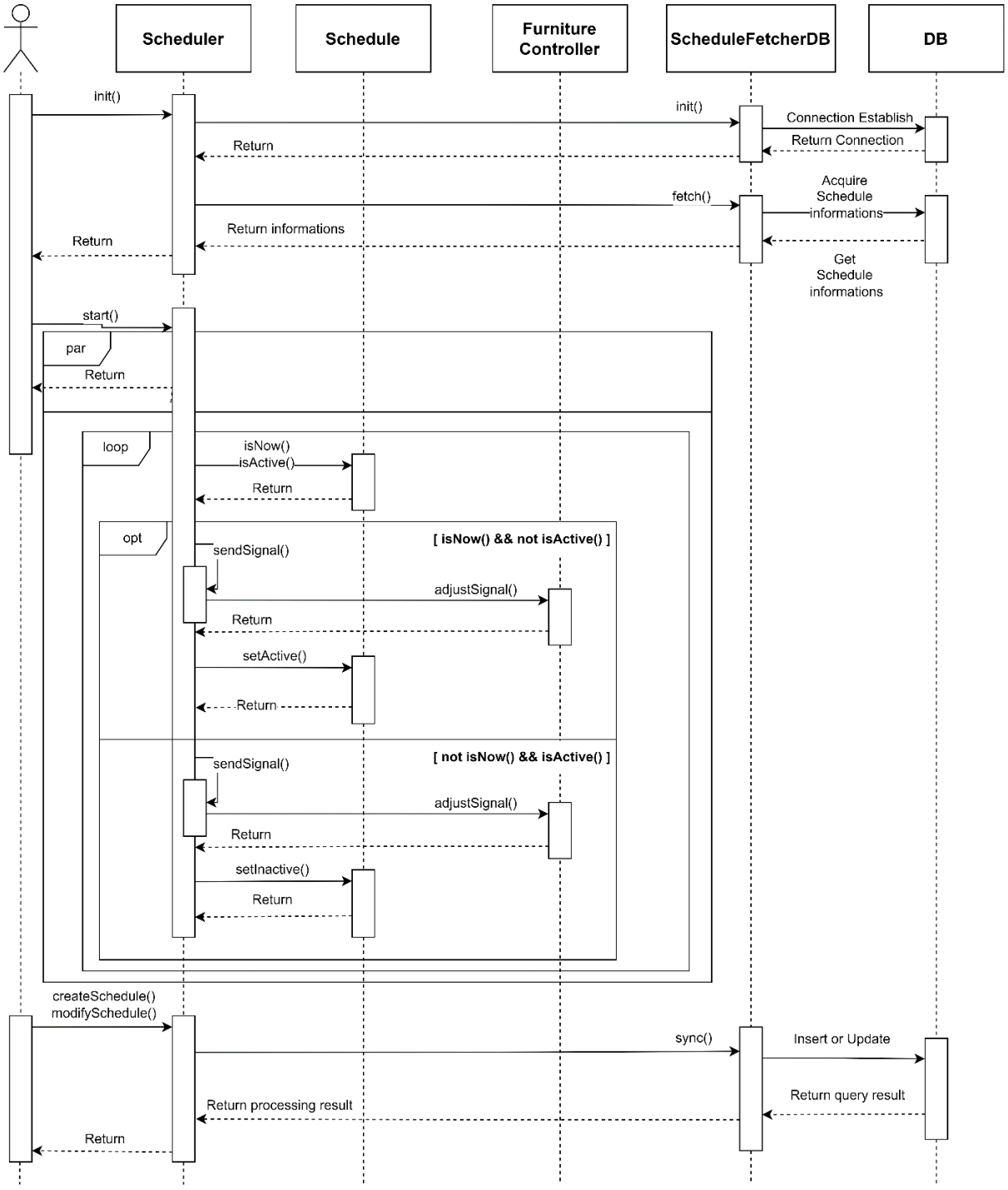
- init(ScheduleFetcher, Controller)
- start()
- sendSignal(Schedule)
- createSchedule(startTime, endTime, brightness, humidity, temperature)
- modifySchedule(id, startTime, endTime, brightness, humidity, temperature) : The default value of arguments except id is null or value indicates null.

5.2.4.3. Class Diagram



[Figure 28] Class diagram – Scheduler

5.2.4.4. Sequence Diagram



[Figure 29] Sequence diagram – Scheduler

6. Testing Plan

6.1. Objectives

This chapter describes plans for tests with three main subgroups: development testing, release testing, and user testing. These tests help to detect potential errors and defects in the application and provide stable functionality to customers.

6.2. Testing Policy

6.2.1. Development Testing

Development testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. Also, by performing development testing, we can get higher code quality at any given time because new code is continually being tested and we can shortened time to market for new features. It reduces software errors and facilitates additional implementations such as providing new features to customers and fixing bugs. In addition, rapid feedback is possible, so even if the design is wrong, correction can be made quickly through the test case. Through this process, we can check whether we meet the following three non-functional requirements: Performance, reliability, security.

6.2.1.1. Performance

In software quality assurance, performance is to test whether the tasks requested by the user are handled smoothly or maintained reliably. Therefore, we will conduct a performance evaluation on how fast the user can control the desired sleep environment and whether unregistered users cannot use it.

6.2.1.2. Reliability

Reliability tests are performed to verify the reliability of the software and ensure that the software meets the intended purpose for a given time in a given environment and can operate without errors. In a mechanized world, people blindly believe in any software these days. No matter what the software system shows, people believe and follow that software will always be accurate. In fact, that is a common mistake we all make. The user believes that the displayed data is accurate and the software always works correctly. Reliability test is required here.

6.2.1.3. Security

Today, the importance of security is increasing as almost all companies (banks, shopping malls, and securities transactions) switch to digital methods. In other words, the online application has gained the user's trust by having the core function called security. The importance of security has grown exponentially. If the online system can't protect transaction data, no one will think about using it. This will make security testing critical, and it will be able to detect risks by identifying potential risks (vulnerabilities, security loopholes). This approach also provides developers with ample time to fix these problems before it becomes a significant security incident.

6.2.2. Release Testing

Release testing is the process that the team responsible for testing actually goes through to deliver the software that has been developed to users. In other words, it is a process of collecting user requirements in an intermediate process before being delivered to the user, organizing the developers' needs for development, and testing whether they have been properly implemented according to the organized specification.

The release testing is carried out under the assumption that the customer is not aware of the internal technology of the system (black box testing). The main test is to make sure that the system responds well to the commands requested by the user, or that the system-level requirements are well-equipped.

The objective of release testing is to build confidence into a release candidate. Release testing is a testing approach or strategy rather than one single grand testing method.

Like any other testing approach, release testing aims at breaking the system running the software release candidate in a controlled environment.

6.2.3. User Testing

User testing is done for users and customers (relatively uninterested). User testing is important because it's hard to get good results if you don't have the features you want or don't work properly as an end product testing. The types of user testing are alpha testing (selective user -> software knowledge), beta testing (someone who doesn't have software knowledge), and acceptance testing (checking that the test is done properly). Once the acceptance testing is

complete, the distribution of the product will begin thereafter.

6.2.4. Testing Case

The test case will basically ensure that the smart-device (furniture) is properly controlled for the optimal sleep environment (temperature, humidity, and brightness) that the user wants, and that it works properly to prevent unregistered users from using smart-sleep-system. Also, even registered users will check if they can't control smart-device (furniture) in a space other than their own.

7. Development Plan

7.1. Objectives

This chapter explains the technology and environment for application development.

7.2. Frontend Environment

7.2.1. Kakao Oven

[Figure 30] Kakao Oven

Kakao Oven is a web/application prototyping tool provided by Kakao. Sample UI(User Interface) can be designed before developing the software, so clients can be informed about how the software is going to be developed. The biggest advantage of Kakao Oven is that people do not need any difficult skills to use it. It does not require any other program, so people can

use it if only computer and internet are there. Also, it provides various screen sizes, and it contains more than 100 UI components, more than 500 vector icons.

7.2.2. Flutter

[Figure 31] Flutter

Flutter is an open-source UI software development kit created by Google and it allows developing IOS and Android applications. It is a multi-platform framework, so it fits our goal to make cross-platform applications in a single codebase.

7.2.3. Android Studio (Wear OS)

[Figure 32] Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system. We are going to develop application based on Wear OS which will be operated in Galaxy Watch.

7.2.4. Xcode (WatchOS)

[Figure 33] Xcode

Xcode is Apple's integrated development environment(IDE) for macOS, used to develop software for macOS, iOS, iPadOS, watchOS, and tvOS. We are going to develop application based on watchOS which will be operated in Apple Watch.

7.3. Backend Environment

7.3.1. Github

[Figure 34] Github

GitHub provides Internet hosting for software development and version control using Git. Software developers can work collaboratively using GitHub and view the history of the documentation. It is a mandatory tool for software developing, so Smart Sleep System is also going to be worked on GitHub to collaborate and control the version of it.

7.3.2. Flask

[Figure 35] Flask

Flask is a Python based micro web framework for developing web application. Flask as a web application framework collects libraries and modules so that developers can make web application easily. Backend server can communicate with mobile device and smart watch REST API using flask.

7.3.3. Eclipse Paho

[Figure 36] Eclipse Paho

To communicate with real-life IoT device such as light bulb, MQTT(Message Queuing Telemetry Transport) is great protocol to be used. Eclipse Paho is a MQTT implementation, and available on various platforms and programming languages.

7.4. Constraints

The software will be designed and implemented based on the contents mentioned in this document. Other details will be carried out in the direction preferred by the developer while developing, and they must comply with the following.

- User information must be protected and never be leaked in any way.
- Smart devices must not disturb any other products already exist.
- Use open source if it is possible.

- Save system development costs and maintenance costs as much as possible.
- Do not use technologies or software that require separate licenses or pay royalties.
- Convenience of users is prioritized.
- Follow the opinion of users regularly.
- Develop the system as the version of operating system is increased.
- Optimize the source code so that implemented function is not overloading the server.
- System requirements are provided in the table below.

[Table 1] Smart Sleep System requirement

System Requirement	
Section	**Minimum Specification**
OS	Android 12 iOS 14 wear OS API 31 watch OS 8
SoC	Qualcomm Snapdragon 845
RAM	2GB

7.5. Assumptions and Dependencies

All systems in this document are designed and implemented based on Android and iOS devices. Therefore, all content is based on the Android operating system with a minimum API version 31 or iOS 14. In case of wear OS, it should be API version 31, and for watch OS, it should be watch OS 8. It may not apply to other operating systems or versions.

8. Supporting Information

8.1. 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).

8.2. Document History

[Table 2] Document History

Date	Version	Description	Writer
2022/05/08	1.0	Style and Overview	Younghoon Jun
2022/05/09	1.1	Addition of 4.2.1, 4.2.2	Aitana Morote
2022/05/09	1.2	Addition of 7	Seryeong Kim
2022/05/09	1.3	Addition of 6	Yunseong Kim
2022/05/10	1.4	Addition of 5.2.3, 5.2.4	Heegwan Son
2022/05/10	1.5	Addition of 4.2.3, 4.2.4, 4.2.5	Younghoon Jun
2022/05/10	1.6	Addition of 5.2.1, 5.2.2	SenzBernal Maria
2022/05/11	1.6.1	Revision of 5.2.4	Heegwan Son
2022/05/11	1.7	Addition of 2	Yunseong Kim
2022/05/11	1.7.1	Revision of 4.2.4, 4.2.5	Younghoon Jun
2022/05/12	1.8	Addition of 6	Yunseong Kim
2022/05/12	1.9	Addition of 1, 3	Eunki Song
2022/05/13	1.10	Revision of 7	Seryeong Kim
2022/05/14	2.0	Revision of All	Younghoon Jun
2022/05/15	2.1	Revision of 1	Younghoon Jun