Advanced Software Engineering
DESIGN REPORT

Team number:	0501

Team member 1	
Name:	Mohammadali Zaretorkmahalleh
Student ID:	12047123
E-mail address:	a12047123@unet.univie.ac.at

Team member 2	
Name:	Khadijeh Arabi
Student ID:	11945135
E-mail address:	a11945135@unet.univie.ac.at

Team member 3	
Name:	Reza Rezaie
Student ID:	12045585
E-mail address:	a12045585@unet.univie.ac.at

Team member 4	
Name:	Mehdi Benabed
Student ID:	11850068
E-mail address:	a11850068@unet.univie.ac.at

Team member 5	
Name:	Behrad Hemati
Student ID:	11849389
E-mail address:	a11849389@unet.univie.ac.at

1. Design Draft

1.1 Design Approach and Overview

1.1.1 Assumptions

For our smartHome we made several assumptions based on which we designed the web app. Regarding the users, we have made the following assumptions:

UserRoles:

- **Customer**: customers of system can register and login to the system and manage the layout and the user can add device or define the device for each room and set pre-value for each device and control the device,for example user can turn on or turn off the light or even the user can define the value of the lighting of the lamp in order to control dimness. The user can additionally remotely monitor the environment (light dimness and temperature) in each room.
- **Maintainer** : the person who can check the health of the system by sending requests to all the services.

1.1.2 Design Decisions

After discussing about domain model and bounded context we decided to model the following bounded contexts:

- **Authentication management** Users and their access roles (customer, maintainer) for entering the subdomain/ bounded context, authorization authentication. Users can register , login and logout of the system.
- **Notification management:** User get notified with Notiication service. Any action performed by control management should notify user by sending an email notification. The system should provide notification settings, we assume sending an email notification for the list of actions that a user can choose to be notified via email.
- **Monitor management:** Users can remotely monitor the environment (e.g. light dimness and temperature) in each room. This can be done by requesting the values that the sensors mock, it is assumed that each room in the layout with a smart device(s) (Heater/AC or Lamp(s)) is associated with a proper sensor (e.g. a temperature sensor for heater) that captures a mocked value (e.g. temperature) of the environment. A user can request the value he wishes to monitor by selecting a room and sensor type (temperature or light) which are predefined by the layout management. As for the unit of the variables, they can either be mocked in a specific unit that can be converted if the user wishes to switch the unit to a different unit than the default, otherwise a sensor can mock values in both units and the user's preference defines which value

to request. Introducing a “sensor type” attribute allows combining the overlapping “monitor light” and “monitor temperature system requirements.

- **Layout Management:** User can change the layout by adding and removing rooms, as well as choosing devices for each room. It is assumed each room one has one of each sensor, light and temperature. Also, that each device will be given a unique ID and at any time will be assigned to one room only.
- **Maintainer management:** the maintainer of the system sends request health checks to all services for checking whether they work or not.The other services should send response to the request of health checking.
- **Time-based management:** For certain devices that support time-based actions, the ability to add times of day in order to actuate certain actions, such as turning on a device. It is assumed that turning on the device is done by a smart plug which can connect to our system and simply sending electricity by the plug will turn the device on. This assumption is done for ease of general compatibility with any device type.
- **Preferences management:** Users’ supposed to put their preferred value(Unit-based) for devices in each user-defined layer. Once logged-in user access to this UI(form), the UserID will be retrieved through User-Management API-Gateway, which means that only registered users can gain access to this System. It needs to read data from Layout Management through its API, so User needs to select the layout and devices within, to set his/her Prefered value. This is vital for Control management to adjust the sensors.
- **Control management:** System will do the adjustment to meet the users’ preferred value. Regards to Control Management, it should be connected via User Preference API-Gateway to read the desirable value which have been set by user(user defined) and hence it need to read the real live value for sensors through API-Gateway of Monitor management to take action, change value to the one which is preferred by user. Another boundary, which maintainer need to get health check data from this domain through API

1.1.3 Design Overview

For the design, we followed the lecture material and selected suitable patterns and architecture variants, which we want to use.

As required in the specification, we designed the entire system according to domain driven design (DDD). For this purpose, we modelled the domain model with the corresponding architectural views. Following the DDD, we created the domain model with all subdomains/bounded contexts included.

We identified following subdomains:
1- Authentication management
2- Notification management
3- Monitor management
4- Control management
5- Layout Management
6- Maintainer management
7- Time-based management

8- Preferences management

We have chosen a layered architecture for our system since this is a core pattern for DDD. With this pattern, the concerns can be separated easily.

In the Layered Architecture the

- Presentation Layer
- Application Layer
- Domain Layer
- Infrastructure Layer

- **Model-View-Controller Pattern(MVC):**
  When using the Spring boot framework, the MVC pattern comes with it automatically.

- **Factory**
For dynamically creating objects without exposing the creation logic to the client and refer to newly created objects using a common interface.

- **One Database per service**
  For loosely coupling of services so that the service can be developed, deployed and scaled independently when possible.

1.2 Development Stack and Technology Stack

The following describes the set of technologies, software and tools used to develop and deploy the Smart Home Control project.

1.2.1 Development Stack

Maven vs Gradle vs Ant

Build tool for executable targets, dependency management and building Chosen: Maven or Gradle

We chose Maven because the team has previous experience.

Gitlab CI/CD:

Build, Test, Tag and Deployment stages.

For quality assurance reasons frequent execution of tests, build in docker, tagging... before a merge to master is executed. Based on the git tag deployment can be performed on a regular basis.

Developer specific IDE: free choice

Source code management: GIT in GitLab

GitLab is mandatory

API dev tool:

Postman vs Insomnia Rest Client

Chosen: Postman

We chose Postman because it is simple to use, and the team is familiar with the tool.

1.2.2 Technology Stack

- Spring Boot + extensions
- ApiGateway:
- Languages: Java 8, JS, HTML, Thymeleaf, CSS
- Docker + Docker compose
- Post-Man : API testing tool
- Junit : Unit testing tool
- Email notification service
- Databases: (Postgres + MongoDB + MySql) running in docker with docker volume
    - Up to the developer ○

Database must work with Spring and Docker

- Tracing: Jaeger
    - If necessary for debugging
    - To trace the flow of execution in the deployed application
    - Good for finding out issues in production
- Unit testing: Junit5
    - unit tests using Junit5 (latest) with build tool integration

- Component TestingCypress (Component + e2e tests) vs Selenium vs Arquillian Cube

- framework for e2e tests
- runs in docker and node
- test cases are written in js

All microservices will be written in java. They will be based on the spring boot framework and communicate through a message broker client.

2. System Requirements

2.1 Functional Requirements

SR1 The system should provide an API Gateway, as well as integration or end-to-end tests, and a client for testing.

Identifier:	SR1-ApiGateway
Priority:	high
Description:	All calls from the browser(=client) must go through the ApiGateway in order to reach the backend services. The ApiGateway is a reverse proxy that basically accepts all API calls, then calls the required services to fulfil them and returns the result. Furthermore the ApiGateway handles authentication, analytics or scaling.
Affected Use-Cases and User-Roles:	User-Roles: ● System Use-Cases: ● All
Verification Method:	● Check via E2E tests if all requests are handled properly.
How to verify	End-2-end tests or manually.
Implications on the implementation	limitations: All

Table 1 System Requirement 1

Identifier:	SR2-Event-driven-communication
Priority:	high
Description:	All services support event-driven communication. Services will subscribe to topics or publish to topics. The message broker will implement the notification mechanism.
Affected Use-Cases and User-Roles:	
Verification Method:	
How to verify	
Implications on the implementation	

Table 2 System Requirement 2

Identifier:	SR3 Temperature monitoring
Priority:	High
Description:	Software-based Management for Services in Domain, In order to monitor the value captured by the sensor
Affected Use-Cases and User-Roles:	User-Roles: ● System Use-Cases: ● Display values mocked by sensors ● Switch the unit of mocked variable
Verification Method:	Manually
How to verify	The displayed value matches the user's preference.
Implications on the implementation	Limitation: Dependency on the layout management

Table 3 System Requirement 3

Identifier:	SR4 **control Temperature**
Priority:	High
Description:	The system shall be able to control the Temperature of all devices in all layout based on user Preferences
Affected Use-Cases and User-Roles:	User-Roles: ● System Use-Cases: ● Adjustment/control
Verification Method:	Test Cases Manually
How to verify	Change the status of sensor
Implications on the implementation	Dependencies on Monitoring management and Preference Management

Table 4 System Requirement 4

Identifier:	SR5 Light monitoring
Priority:	High
Description:	Software-based Management for Services in Domain, In order to monitor the value captured by the sensor
Affected Use-Cases and User-Roles:	User-Roles: ● System Use-Cases: ● Display values mocked by sensors
Verification Method:	Manually
How to verify	The displayed value matches the user’s preference.
Implications on the implementation	Limitation: Dependency on the layout management

Table 5 System Requirement 5

Identifier:	SR6- control light Dimness
Priority:	high
Description:	The system shall be able to control the light dimness of all devices in all layout
Affected Use-Cases and User-Roles:	User-Roles: ● system Use-Cases: ● Adjustment/control
Verification Method:	Test Cases
How to verify	Dimness should be in preferred range
Implications on the implementation	Dependency on the Preference management,Monitoring Management

Table 6 System Requirement 6

Identifier:	SR7 **preferences management**
Priority:	High
Description:	User's Definition of Light Unit, Temperature Unit
Affected Use-Cases and User-Roles:	User-Roles: ● User Use-Cases: ● Set values-Update-Delete
Verification Method:	Test Cases Manually
How to verify	stored successfully
Implications on the implementation	Dependency on the layout management,user Management

Table 7 System Requirement 7

SR8 The system should support a login service, where role-based authentication is managed. 11 Users log in the system and receive an access token .

Identifier:	SR8 Create an account
Priority:	high
Description:	● The user should be able to create the account.
Affected Use-Cases and User-Roles:	User-Roles: ● User ● Maintainer Use-Cases: ● Registration
Verification Method:	Manually
How to verify	Account will be created
Implications on the implementation	

Table 8 System Requirement 8

Identifier:	SR9 Home Layout Service
Priority:	High
Description:	● Adding and removing rooms in a house ● Adding and removing devices for each room
Affected Use-Cases and User-Roles:	User-Roles: ● User Use-Cases: ● Add Room ● Remove Room ● Add Device ● Remove Device ● Edit/Update Device
Verification Method:	Manually
How to verify	Room/device was added
Implications on the implementation	

Table 9 System Requirement 9

Identifier:	SR10 Time Based Actions
Priority:	Medium
Description:	• Defining time for certain actions to take place.
Affected Use-Cases and User-Roles:	User-Roles: • User Use-Cases: • Specify a time for turning on device • Specify a repeating time for the action
Verification Method:	Manually
How to verify	Time was added
Implications on the implementation	

Table 10 System Requirement 10

Identifier:	SR11
Priority:	high
Description:	A system shall be able to support an email notification and inform customer about performed actions
Affected Use-Cases and User-Roles:	User-Roles: • User / Maintainer Use-Cases: • All
Verification Method:	Test Cases
How to verify	Received email
Implications on the implementation	

Table 11 System Requirement 11

SR12 The system should support a maintenance service. The maintainer uses this service to check the health of each microservice

Identifier:	SR12 Maintaining
Priority:	High
Description:	Maintainer should check health of all the services
Affected Use-Cases and User-Roles:	User-Roles: Maintainer Use-case: Maintaining
Verification Method:	Test cases
How to verify	Get response
Implications on the implementation	

Table 12 System Requirement 12

2.2 Non-Functional Requirements

Identifier:	13 Portability
Priority:	high
Description:	The application should be usable in different browsers.
Affected Use-Cases and User-Roles:	entire system
Verification Method:	Manual testing
How to verify	The application should be tested on different browsers.
Implications on the implementation	Limitations: The application will not run on operation systems as our application is a web application.

Table 13 System Requirement 13

Identifier:	14 Maintainability
Priority:	high
Description:	Applications always require new features. If developers design and implement maintainable applications then it will be easier to extend and fix bogs.
Affected Use-Cases and User-Roles:	entire system
Verification Method:	Manual testing
How to verify	Try to add new feature and find out if it is possible with current design and implementation
Implications on the implementation	

Table 14 System Requirement 14

Identifier:	15 GDPR Compliance
Priority:	High
Description:	Accordingly with the user's role, data should only be accessible to its owner.
Affected Use-Cases and User-Roles:	Entire system
Verification Method:	Manual testing
How to verify	
Implications on the implementation	

Table 15 System Requirement 15

3. 4+1 Views Model

3.1 Scenarios / Use Case View

3.1.1 Use Case Diagram(s)

Figure 1 Use case diagram - Overview

A simple overview of our Smart Home control system, Users and Email Notification Service. As well we have presented the services of each user. We have two users: Customer, Maintainer .

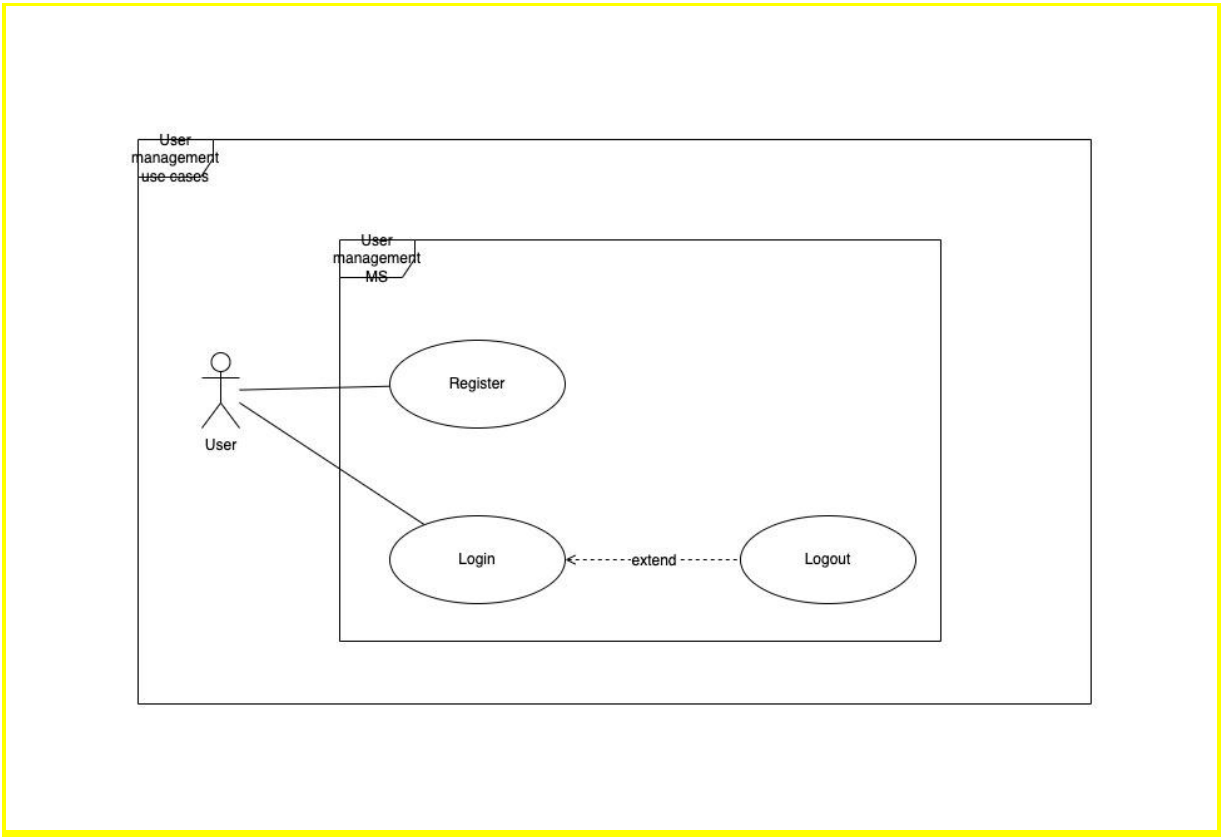


Figure 2 Use case user management

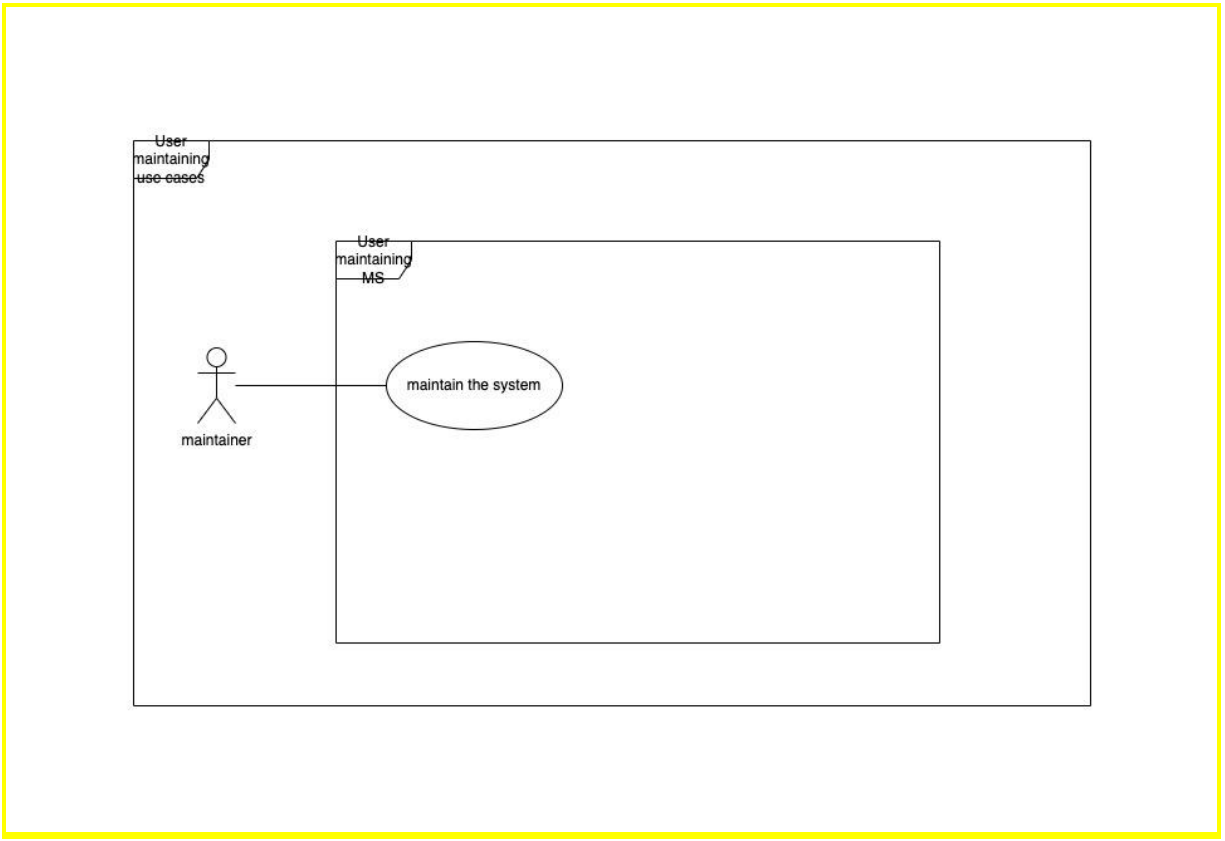


Figure 3 Use case maitaner

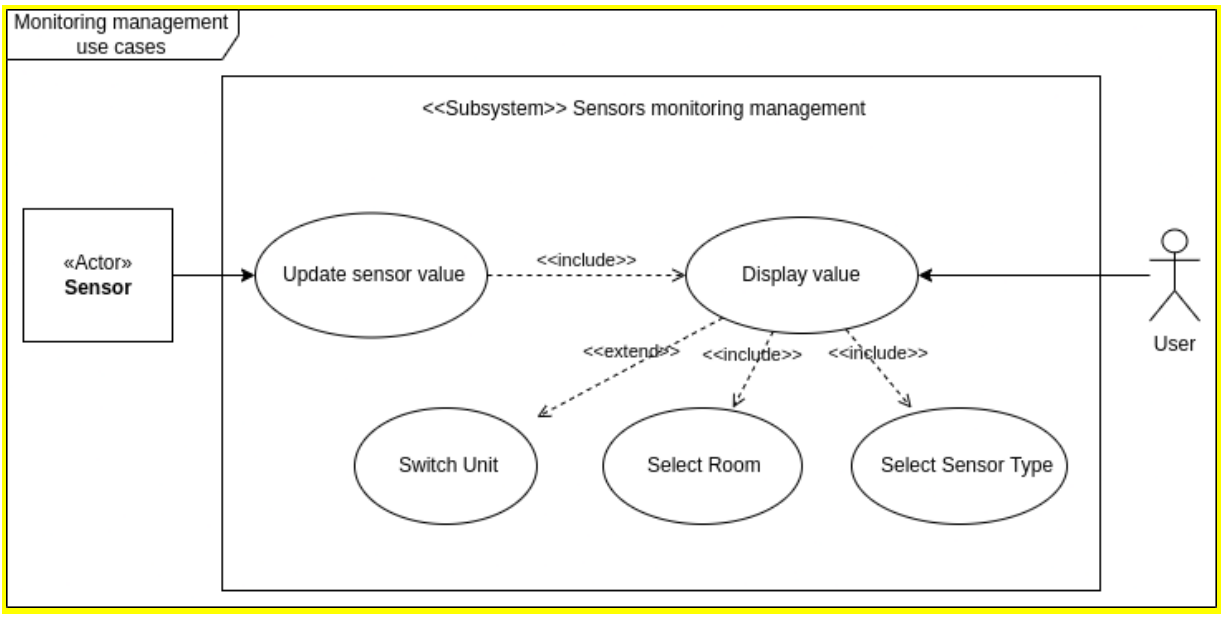


Figure 4 Use case Monitoring

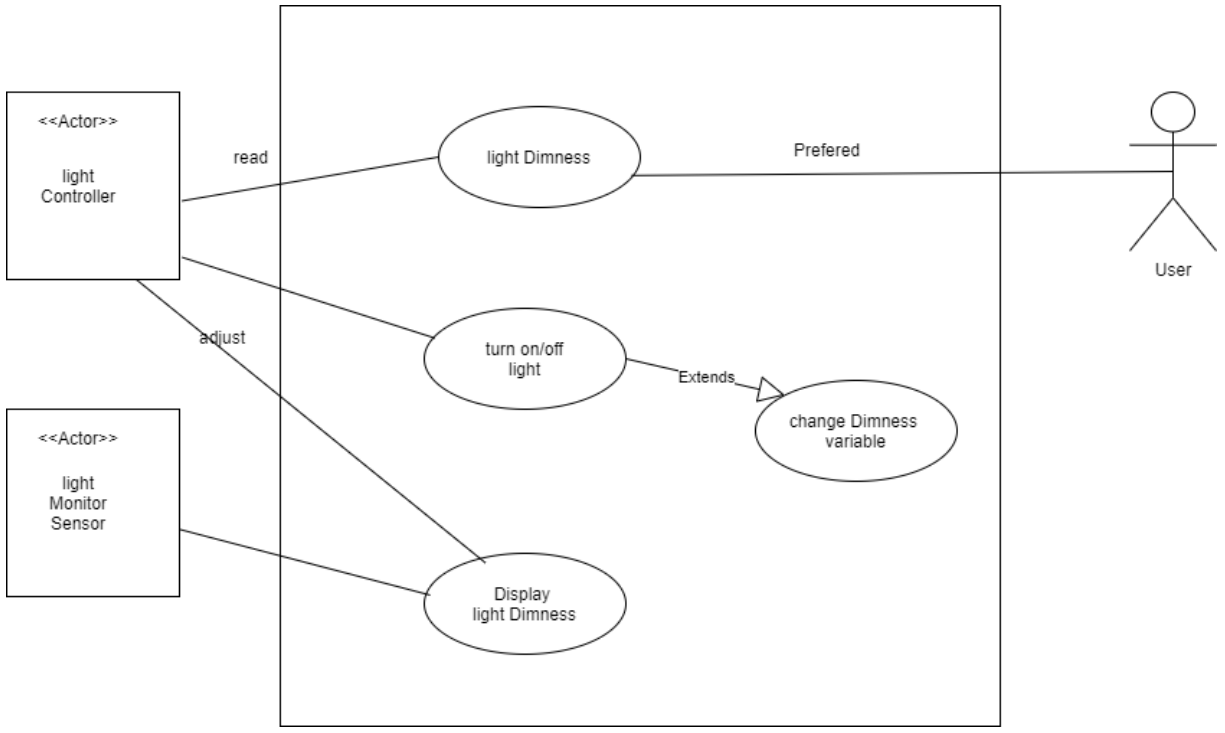


Figure 5 Use case Light Dimness control

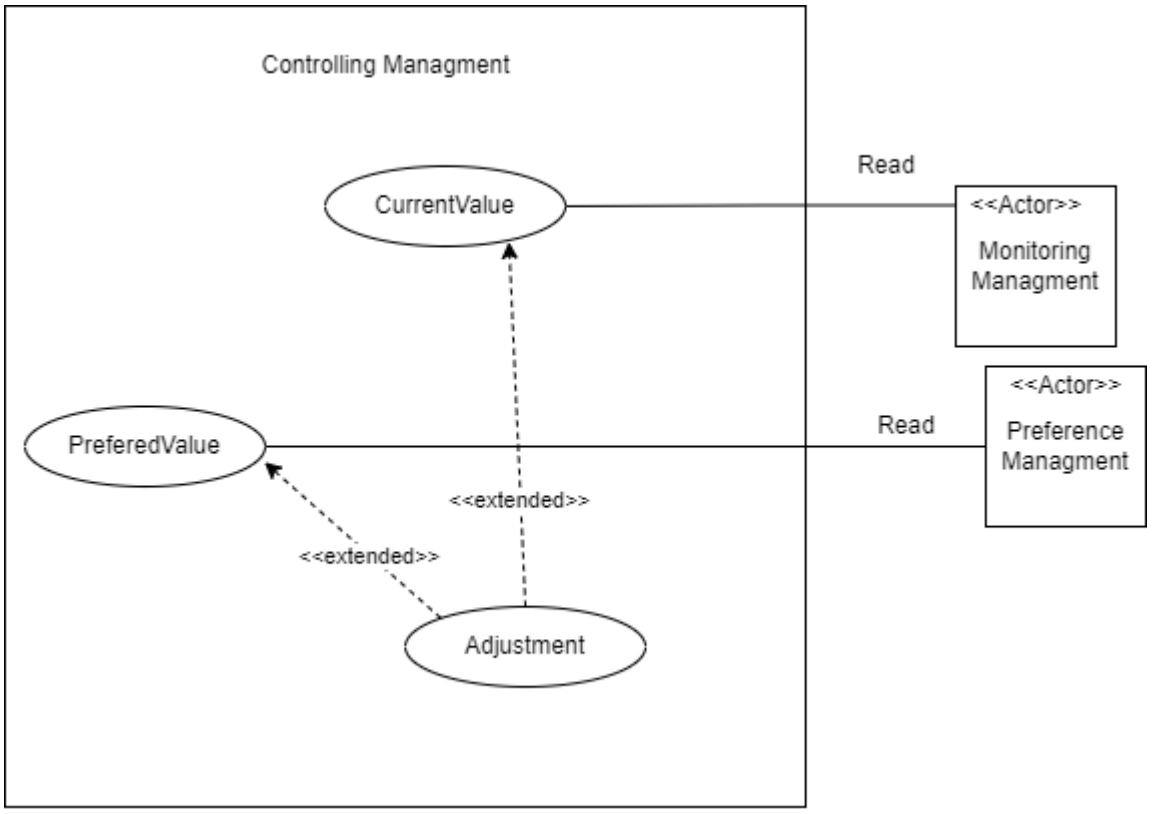


Figure 6 Use case Tempreture control

F

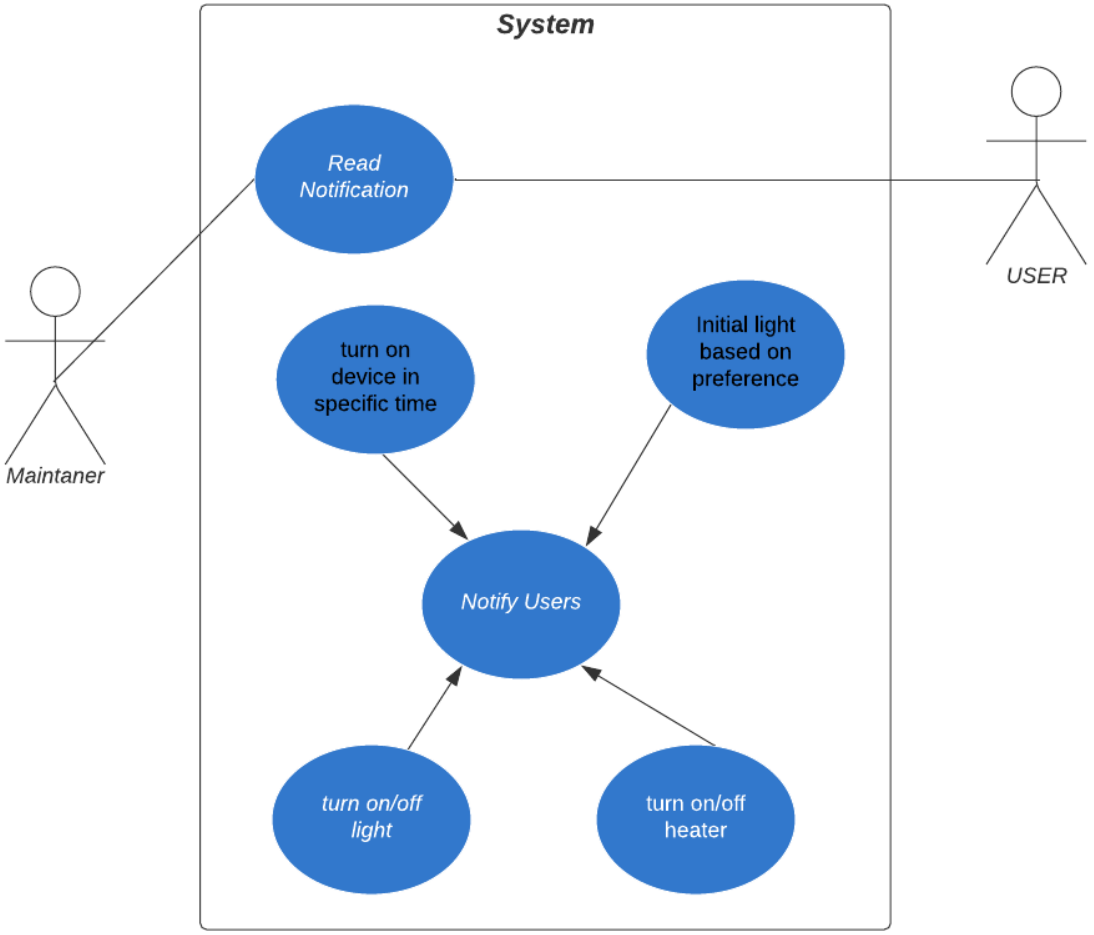


Figure 7 Use case Email notification

Preference Management:

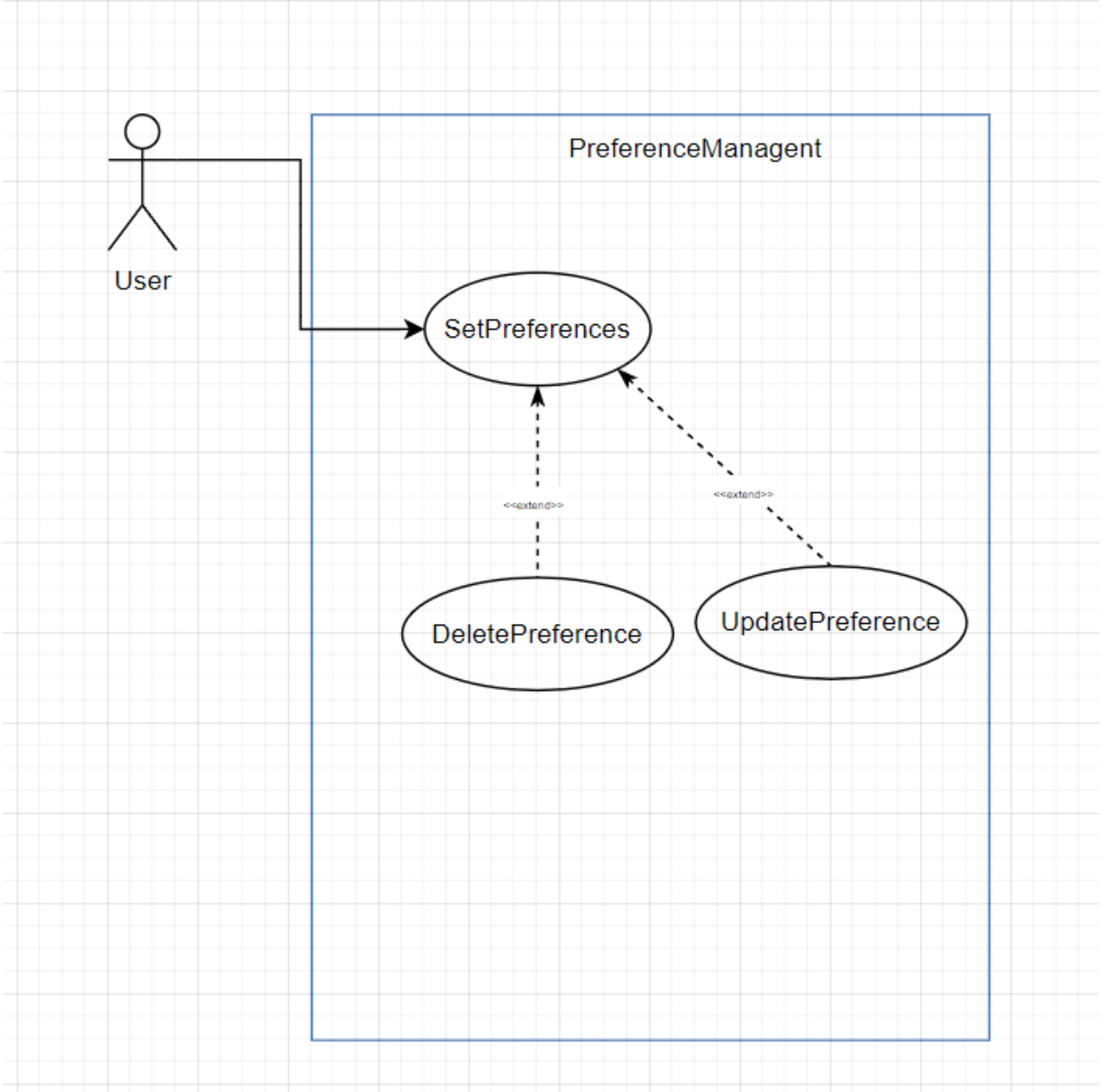


Figure 8 Use case Preference management

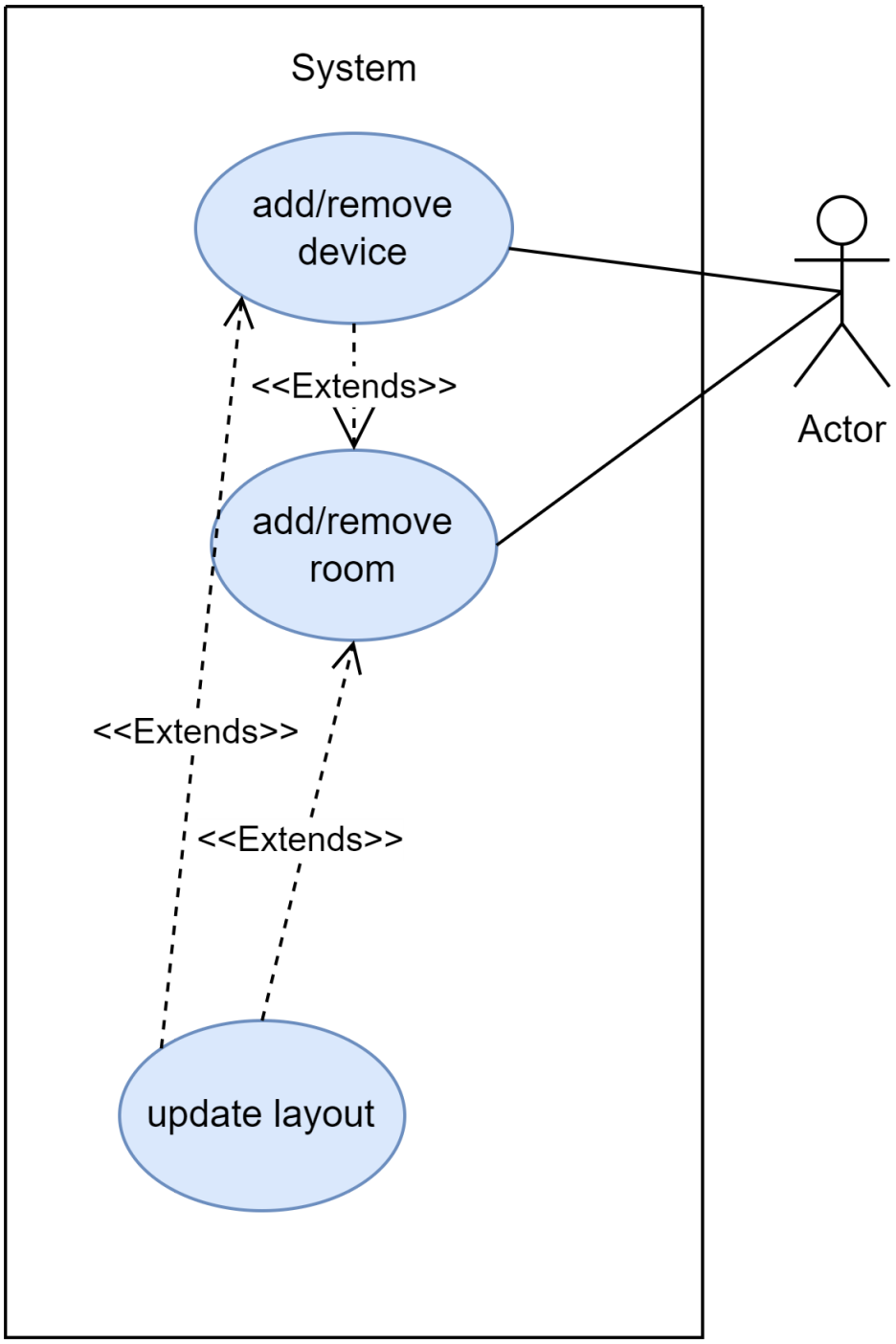


Figure 9 Use case Layout Management

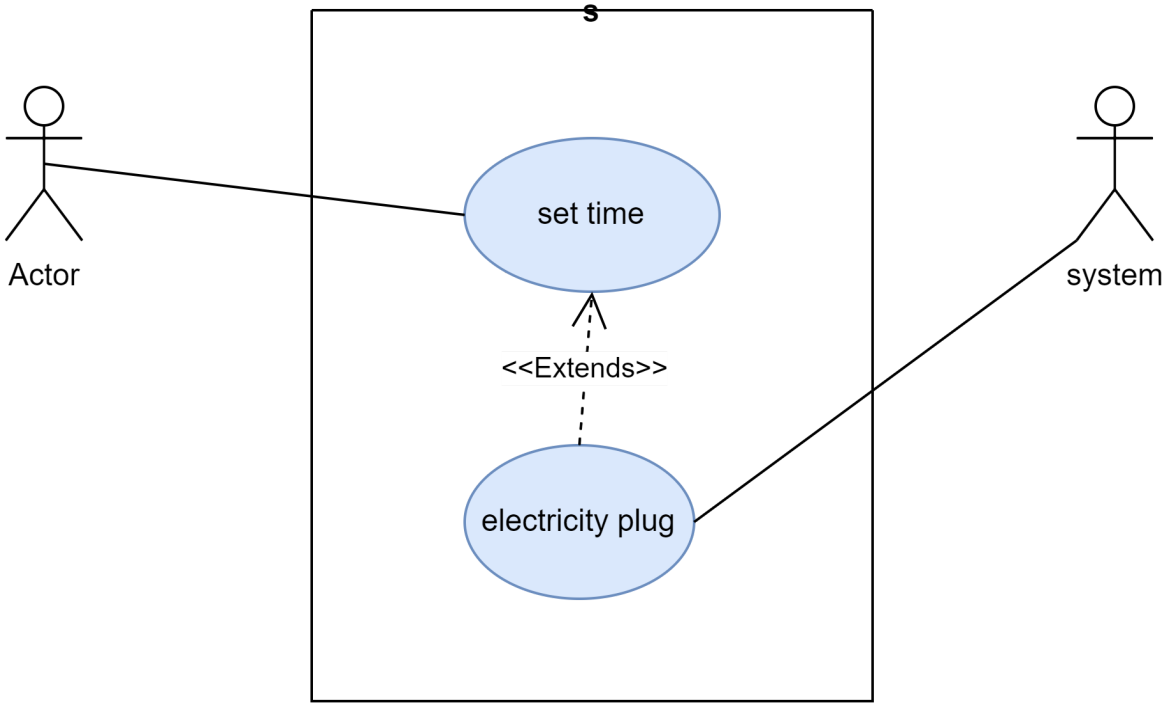


Figure 10 Use case Time based Management

3.1.2 Use Case Descriptions

Use Case:	Registration
Use Case ID:	1
Actor(s):	User,Maintainer
Brief Description:	The user wants to create an account in order to use the system
Pre-Conditions:	The user must fill in all required fields/information in addition to create a new account
Post-Conditions:	The user has a new account , a username and a password which he/she uses to access the account any time
Main Success Scenario:	1.The user opens the application 2.The user clicks the button “Register” 3.The user fills in all required fields/information to proceed with the registration 4. The user clicks the button “Save” after writing all information correct 5. If the user didn’t fill all fields that require a (*) sign he/she gets a notification to fill all the required fields 6. The registration was successfully done and the user has a personal username and password

Extensions:	
Priority:	High
Performance Target:	1000 new users can register per day
Issues:	Too many active accounts for the database to support

Use Case:	Login
Use Case ID:	2
Actor(s):	User, Maintainer
Brief Description:	The user wants to log in to the system
Pre-Conditions:	The user must fill in all required fields/information in addition to create a new account
Post-Conditions:	Login was successful , the users have accessed the system
Main Success Scenario:	1. The user opens the application 2. The user writes his/her required information (username/email and password) 3. If the login details are correct, the user loges into the system
Extensions:	Log out

Priority:	High
Performance Target:	1000 new users can register per day
Issues:	Too many users logged in at the same time, slowing down the system.

Use Case:	Logout
Use Case ID:	3
Actor(s):	User, Maintainer
Brief Description:	The user wants to log out of the system
Pre-Conditions:	The user is logged in
Post-Conditions:	The user is logged out
Main Success Scenario:	1. The user clicks the “logout” button 2. The user is logged out
Extensions:	
Priority:	High
Performance Target:	

Issues:	

Use Case:	Maintaining
Use Case ID:	4
Actor(s):	Maintainer
Brief Description:	The maintainer wants to check health of the system
Pre-Conditions:	The Maintainer is logged in
Post-Conditions:	The Maintainer is logged out
Main Success Scenario:	1. The maintainer send request to all services to check their health.
Extensions:	
Priority:	High
Performance Target:	
Issues:	

Use Case:	Read notification
Use Case ID:	5
Actor(s):	User
Brief Description:	The main goal is that user can read email notification about the action performed(e.g. turn on the light)
Pre-Conditions:	Users must have credentials to access the system. Credentials must be checked.
Post-Conditions:	The status of email notification is marked as “read”
Main Success Scenario:	1. User login in to the email(the registered email in system) 2. The System authenticate the user and start a session 3. User select the notification email from the list of email 4. The system redirects the user to the appropriate notification email page

Extensions:	1a. User is not logged in to the system / user login has a different role. system will display alert message of deny access 4a. the notification email is not available. system will display alert message
Priority:	medium
Performance Target:	Notification email is displayed correctly
Issues:	

Use Case:	Notify about performed action
Use Case ID:	6
Actor(s):	system
Brief Description:	The main goal is that system can notify user about action performed
Pre-Conditions:	1. the user preference (temp and light range) must be entered by user 2. user can choose when to be notified by email 3. a notification list must be available
Post-Conditions:	1. system can emailed a list of notification of all performed action 2. user were informed about action
Main Success Scenario:	1. the system select the user email from available list 2. The system makes sure that the performed action is in the user's selected list or not 3. The system send notification email to the user

	4. the system received confirmed delivery
Extensions:	1a. the user's email is not correct 3a. the notification did not send correctly 4a.the notification was not delivered correctly
Priority:	medium
Performance Target:	Notification has been successfully delivered
Issues:	how should the data set should be displayed

Use Case:	Turn Status of Light Device on/off
Use Case ID:	7
Actor(s):	Light Controller
Brief Description:	The main goal is that Controller can Turn the light on or off and transfer values to the system for monitor
Pre-Conditions:	1. home layout is registered by user 2. All devices that affect the measure of light must be registered in Preference Management
Post-Conditions:	1. the light values can be used for controller

Main Success Scenario:	1. the system requests light dimness values from sensors 2. The sensors measure light dimness. 3. Monitoring System transfer the light values to the system
Extensions:	2a. the sensors don't accept the request because it is broken. user will be modified
Priority:	high
Performance Target:	The light preferred values are correct
Issues:	In which form the values are transferred to the system?

Use Case:	adjust dimness of light
Use Case ID:	8
Actor(s):	Light Controller
Brief Description:	The main goal is that sensors can communicate with the respective monitoring service and adjust the dimness
Pre-Conditions:	1. Respective Monitoring management Posts The current Values of its Devices 2. light dimness preference must be initialised by user in Preference Managment

Post-Conditions:	1. after turning on/off the simulated light, the mocked light dimness variable changed
Main Success Scenario:	1. the system requests Monitoring Managment to control light dimness 2. sensor checks the dimness of different rooms based on user preference 3. sensor communicate with the monitoring service and adjust the dimness
Extensions:	2a. the sensors can't adjust the dimness because it is broken. user will be modified
Priority:	high
Performance Target:	The light value is in the prefered range
Issues:	

Use Case:	Turn Status of Temperature Device
Use Case ID:	9
Actor(s):	Controller
Brief Description:	The main goal is that Controller can Turn the Heater on or off and transfer values to the system for monitor
Pre-Conditions:	1. home layout is registered by user 2. All devices that affect the measure of Temperature must be registered in Preference Management

Post-Conditions:	1. the Temperature values can be used for controller
Main Success Scenario:	1. the system requests Temperature values from sensors 2. The sensors measure Temperature . 3. Monitoring System transfer the Temperature to the system
Extensions:	2a. the sensors don't accept the request because it is broken. user will be modified
Priority:	high
Performance Target:	The Temperature preferred values are correct
Issues:	

Use Case:	adjust Temperature
Use Case ID:	10
Actor(s):	Controller
Brief Description:	The main goal is that sensors can communicate with the respective monitoring service and adjust the Temperature
Pre-Conditions:	1. Respective Monitoring management Posts The current Values of its Devices 2. Tempreture preference must be initialised by user in Preference Managment

Post-Conditions:	1. after turning on/off the simulated heater , the mocked heaters' status changed
Main Success Scenario:	1. the system requests Monitoring Managment to control Temperature value 2. sensor checks the Temperature of different rooms based on user preference 3. sensor communicate with the monitoring service and adjust the Temperature
Extensions:	2a. the sensors can't adjust the dimness because it is broken. user will be modified
Priority:	high
Performance Target:	The Temperature value is in the prefered range
Issues:	

○

Use Case:	Layout Management
Use Case ID:	11
Actor(s):	User
Brief Description:	The user should be able to define rooms in a house, be able to name them and be able to add and remove and assign devices to each of the rooms. In addition, support actions for the whole house, i.e. every room at the same time.
Pre-Conditions:	1. If already not defined, a room gets defined by the user. 2. For each room, user-controllable devices gets added or removed.

	3. Every room can also be removed, which frees up every device already assigned to that room and changes their status to unassigned.
Post-Conditions:	1. After each change to the layout and devices, the change is reflected in the whole system.
Main Success Scenario:	1. The user creates/changes a room/device. 2. After saving the changes, the new layout gets reflected in the whole system. 3. The device cannot be assigned to another room.
Extensions:	
Priority:	high
Performance Target:	A device is only assigned to one room at a time.
Issues:	Too many rooms in a house. Too many devices in one room. A device may have shared functionality or be related to more than one room.

○

Use Case:	Layout Management Update
Use Case ID:	12
Actor(s):	User
Brief Description:	The user should be able to update the layout, as in, change the room that any device is assigned to.
Pre-Conditions:	If a device is already assigned, change the assignment to another room.

Post-Conditions:	After each change to the layout and devices, the change is reflected in the whole system.
Main Success Scenario:	• The user creates/changes a room/device. • After saving the changes, the new layout gets reflected in the whole system. • The device cannot be assigned to another room.
Extensions:	
Priority:	high
Performance Target:	A device is only assigned to one room at a time.
Issues:	Too many rooms in a house. Too many devices in one room. A device may have shared functionality or be related to more than one room.

Use Case:	Time-based Functions
Use Case ID:	13
Actor(s):	User
Brief Description:	The user should be able to define time-based actions for certain actions and devices that support time-based actions.
Pre-Conditions:	1. The user defines a certain time for an action to take place for a certain device that supports time-based actions.

Post-Conditions:	1. At the specified time, the System performs the pre-defined action for the specified device and notifies the user.
Main Success Scenario:	1. At the specified, the action takes place and the user gets notified about the success of the action.
Extensions:	
Priority:	high
Performance Target:	The time is in the proper format.
Issues:	There is no standard for devices that support time-based actions. We need to define what the action is and how it takes place, which can vary greatly depending on the device.

Use Case:	Repeating Time-based Functions
Use Case ID:	14
Actor(s):	User
Brief Description:	The user should be able to define repeating time-based actions for certain actions and devices that support time-based actions.
Pre-Conditions:	The user defines a certain time for an action to take place for a certain device that supports time-based actions which can repeat based on the user's preference

Post-Conditions:	At the specified time and days, the System performs the pre-defined action for the specified device and notifies the user.
Main Success Scenario:	At the specified time, the action takes place and the user gets notified about the success of the action.
Extensions:	
Priority:	high
Performance Target:	The time is in the proper format.
Issues:	There is no standard for devices that support time-based actions. We need to define what the action is and how it takes place, which can vary greatly depending on the device. Also, if the repeating action is defined per user or in general for the whole system.

Use Case:	Preference Management Add Temperature Measure Value
Use Case ID:	15
Actor(s):	User
Brief Description:	The main goal is that User can input its preferred value
Pre-Conditions:	1. user must be registered 2. home layout must be defined
Post-Conditions:	Layout and user must be defined
Main Success Scenario:	1. store user defined value 2. post it to control management
Extensions:	2a. the sensors don't accept the request because it is broken. user will be modified
Priority:	high
Performance Target:	The Temperatures' preferred values are correct
Issues:	

Use Case:	Preference Management Add Light Measure Value
Use Case ID:	16
Actor(s):	User

Brief Description:	The main goal is that User can input its preferred value
Pre-Conditions:	3. user must be registered 4. home layout must be defined
Post-Conditions:	Layout and user must be defined
Main Success Scenario:	3. store user defined value 4. post it to control management
Extensions:	2a. the sensors don't accept the request because it is broken. user will be modified
Priority:	high
Performance Target:	The light preferred values are correct
Issues:	

Use Case:	Monitoring Management : Display value
Use Case ID:	17
Actor(s):	User
Brief Description:	Users can remotely monitor the environment for a specific variable (light dimness and temperature) in a specific room. The value a specific sensor mocks can be viewed by the user upon request
Pre-Conditions:	Home layout is defined (rooms and their respective devices/sensors are registered)

Post-Conditions:	The user is able to view the value of the variable he wishes to display
Main Success Scenario:	1. The user selects the room of interest 2. The user selects the variable of interest (Sensor type: light or temperature) 3. The value of the variable in the specific room is displayed
Extensions:	The user is able to switch the unit of the displayed temperature.
Priority:	High
Performance Target:	The user should be able to view the value of every predefined sensor.
Issues:	The correct value can only be displayed if its sensor is properly defined in the proper room by the layout management.

Use Case:	Monitoring Management : Update sensor value
Use Case ID:	18
Actor(s):	Sensor
Brief Description:	The sensor should be able to return the correct mocked value when requested.

Pre-Conditions:	The (properly functioning) sensor is properly registered in the home layout
Post-Conditions:	The sensor properly captures the environment variable and returns it when requested.
Main Success Scenario:	1. The monitoring service requests the value of a specific sensor. 2. The sensor mocks the current value of the variable (current light dimness or current temperature) 3. The monitoring service receives the requested value.
Extensions:	None
Priority:	High
Performance Target:	The user should be able to view the current value mocked by the target sensor
Issues:	None

Use Case:	Monitoring Management : Select room
Use Case ID:	19
Actor(s):	User
Brief Description:	Users should be able to select the room for which they want to view a sensor's value
Pre-Conditions:	1. Home layout is defined (rooms and their respective devices are registered)

Post-Conditions:	The user is able to view the values of the variables in the selected room
Main Success Scenario:	1. The user can select a room from a set of predefined rooms 2. The user is able to view the values of the sensors
Extensions:	None
Priority:	Medium
Performance Target:	The user should be able to view the values in every predefined room.
Issues:	None

Use Case:	Monitoring Management : Select sensor type
Use Case ID:	20
Actor(s):	User
Brief Description:	Users should be able to select the sensor type and therefore the variable (temperature or light dimness) for which they wish to view the value.
Pre-Conditions:	Home layout is defined (rooms and their respective devices are registered), additionally sensors have a “sensorType” attribute which indicates the variable it is mocking.
Post-Conditions:	The user is able to view the values of the selected variables.

Main Success Scenario:	1. After selecting a room from a set of predefined rooms, the user is able to select a sensor type 2. The user is able to view the values of the sensors for a specific variable
Extensions:	None
Priority:	Medium
Performance Target:	The user should be able to view the values for the each variable.
Issues:	None

Use Case:	Monitoring Management : Switch unit
Use Case ID:	21
Actor(s):	User
Brief Description:	The user can switch the unit of the displayed value especially for temperature (e.g. from °C to °F)
Pre-Conditions:	The value for temperature is displayed in the default unit
Post-Conditions:	The value for temperature is displayed in the chosen unit
Main Success Scenario:	1. The user selects the temperature variable to view in a specific room, the temperature is displayed in the default unit. 2. The user can switch the unit to another available unit he wishes to view the temperature in.

Extensions:	None
Priority:	Medium
Performance Target:	The user should be able to view the value of temperature in any available unit he chooses to use.
Issues:	None

3.2 Logical View

The Smart Home Controlling System is a complex software system consisting of several components. To make the system landscape more flexible, expandable and robust, we decided to use domain-driven design to create a class diagram in the form of a domain model. We implemented several views. On one hand, we have a large diagram as a kind of big picture to appeal to the kind of experts: domain experts and software developers. On the other hand, we have several small sections (subdomains) where the individual sub-areas are examined more closely. The details of the individual subdomains are shown in separate class diagrams (see the following figures). In these detailed diagrams, the classes of DDD building blocks are divided according to the Layered Architecture. we identified the sub-domains as follows:

1- Authentication management
2- notification management
3- monitor management
4- control management
5-Layout Management
6-maintainer management
7-time-based management
8-preferences management

Bounded Contexts were used to define subdomains. This means that each subdomain is distinct from other subdomains. These subdomains can only be entered and used by individual entry points (usually aggregate roots).

User management:

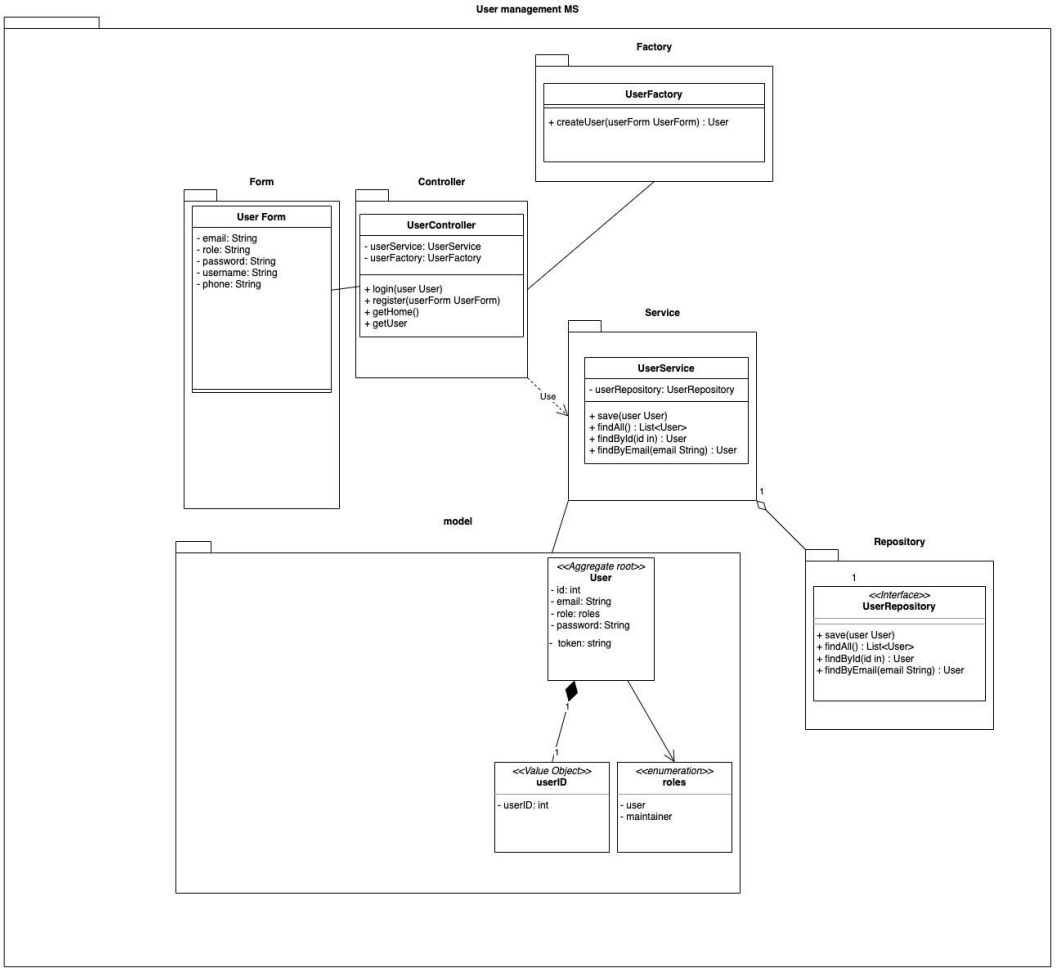


Figure 11 Class Diagram User Management

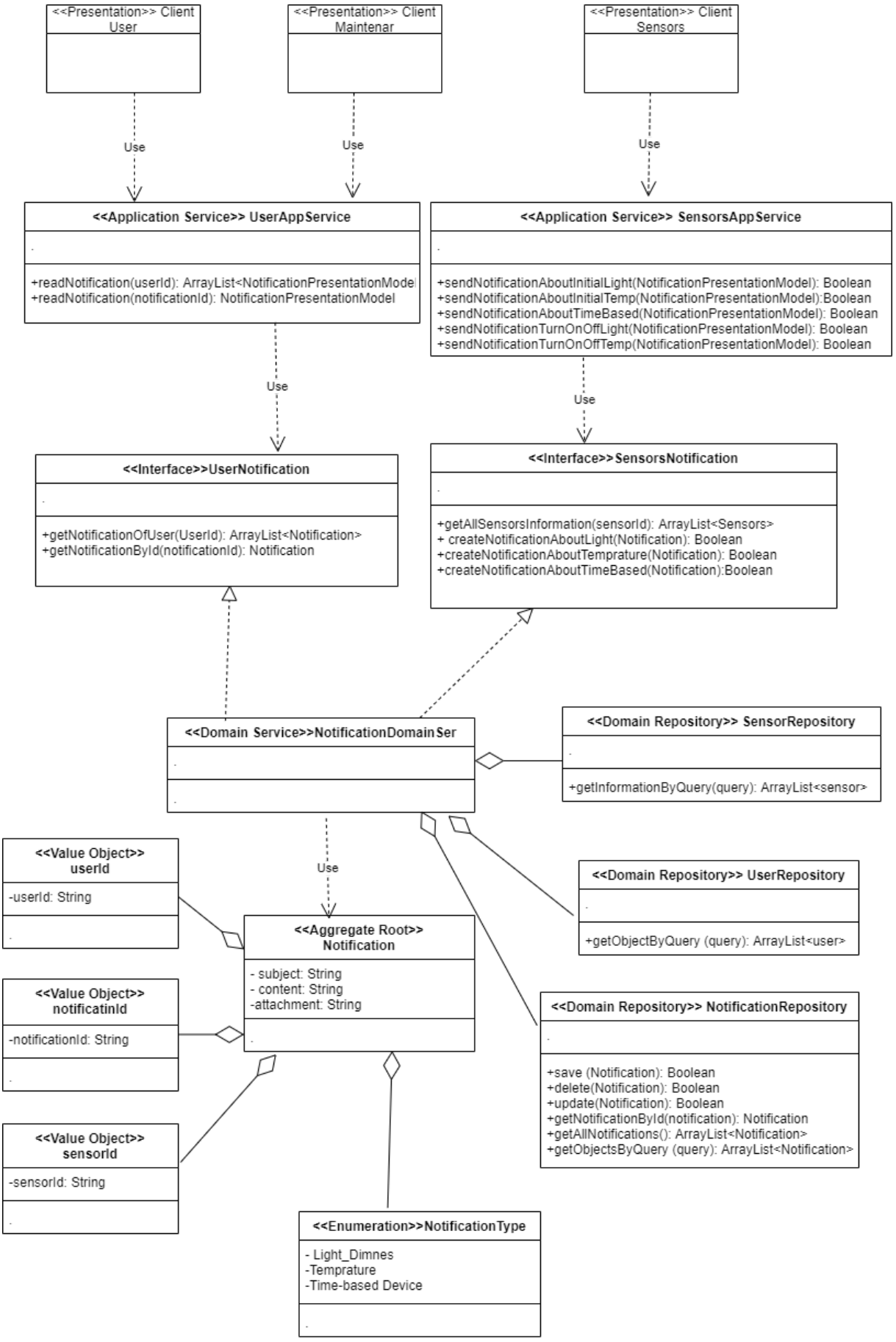


Figure 12 Class Diagram Notification management

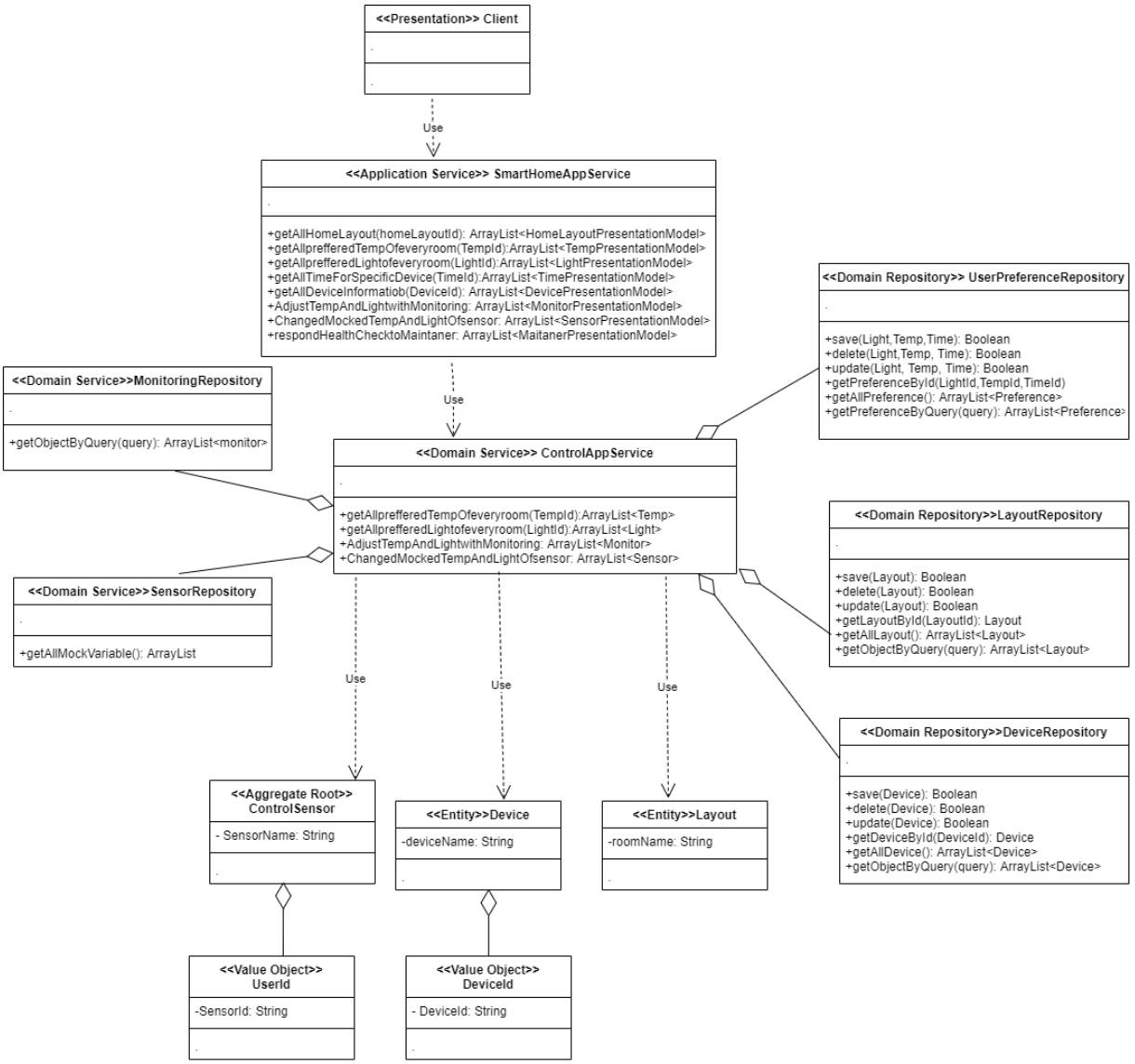


Figure 13 Class Diagram Control Management

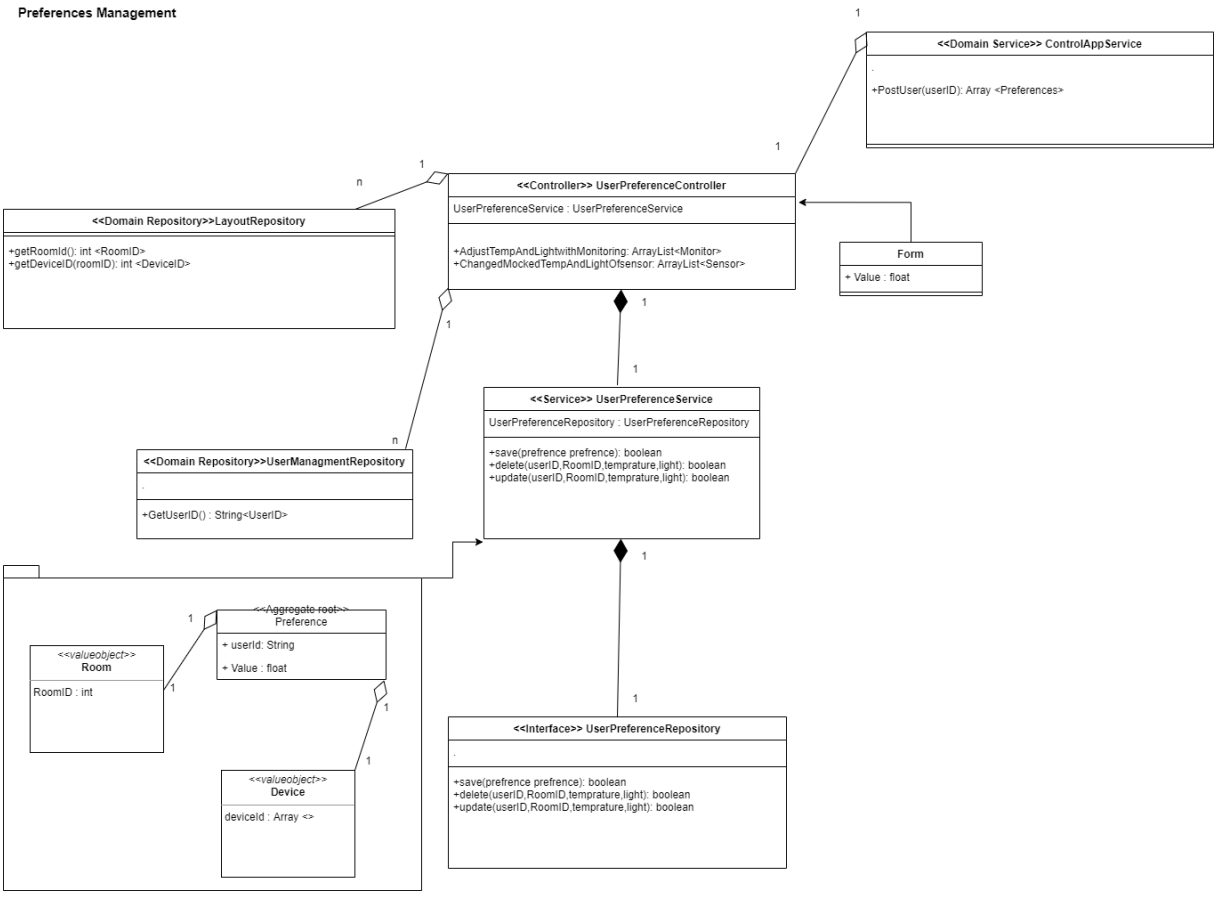


Figure 14 Class Diagram Preference Management

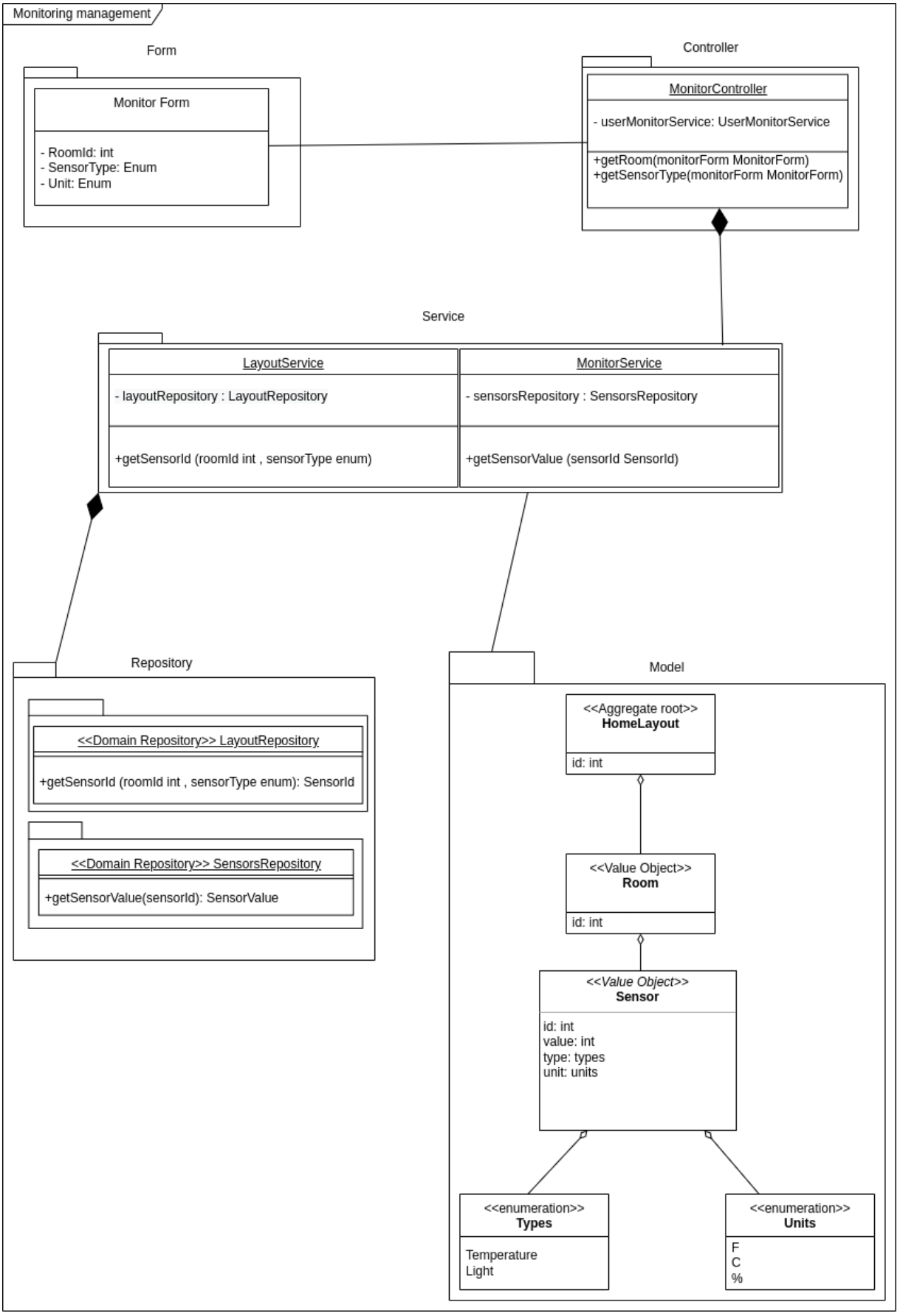


Figure 15 Class Diagram Monitor management

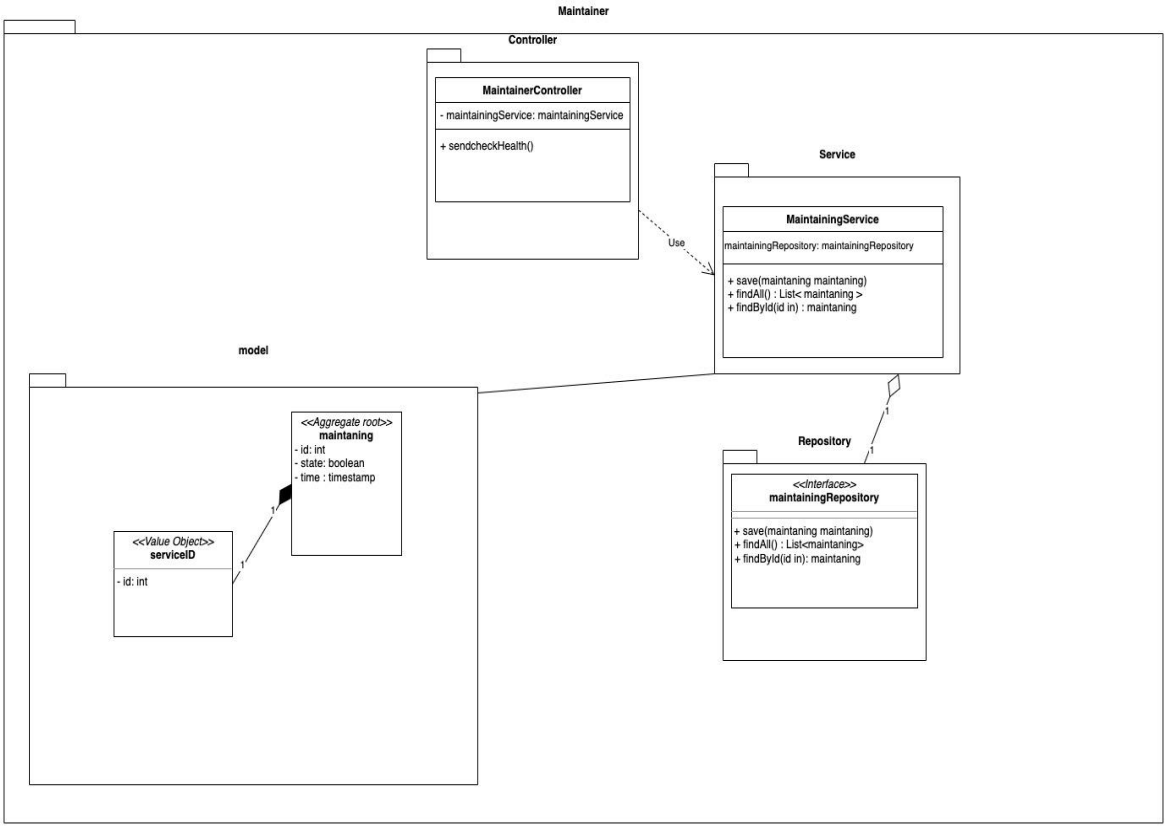


Figure 16 Class Diagram Maintenance management

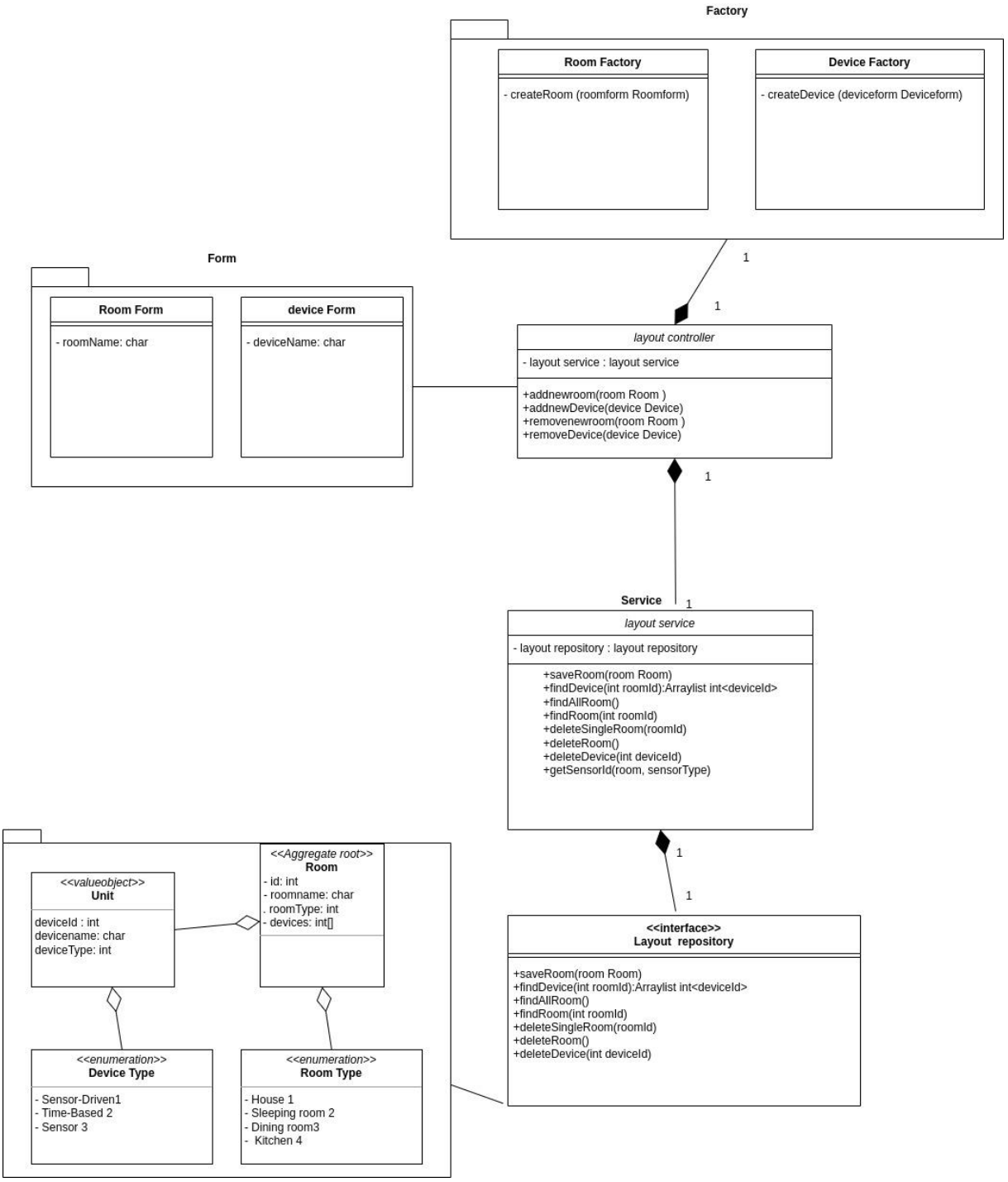


Figure 17 Class Diagram Layout management

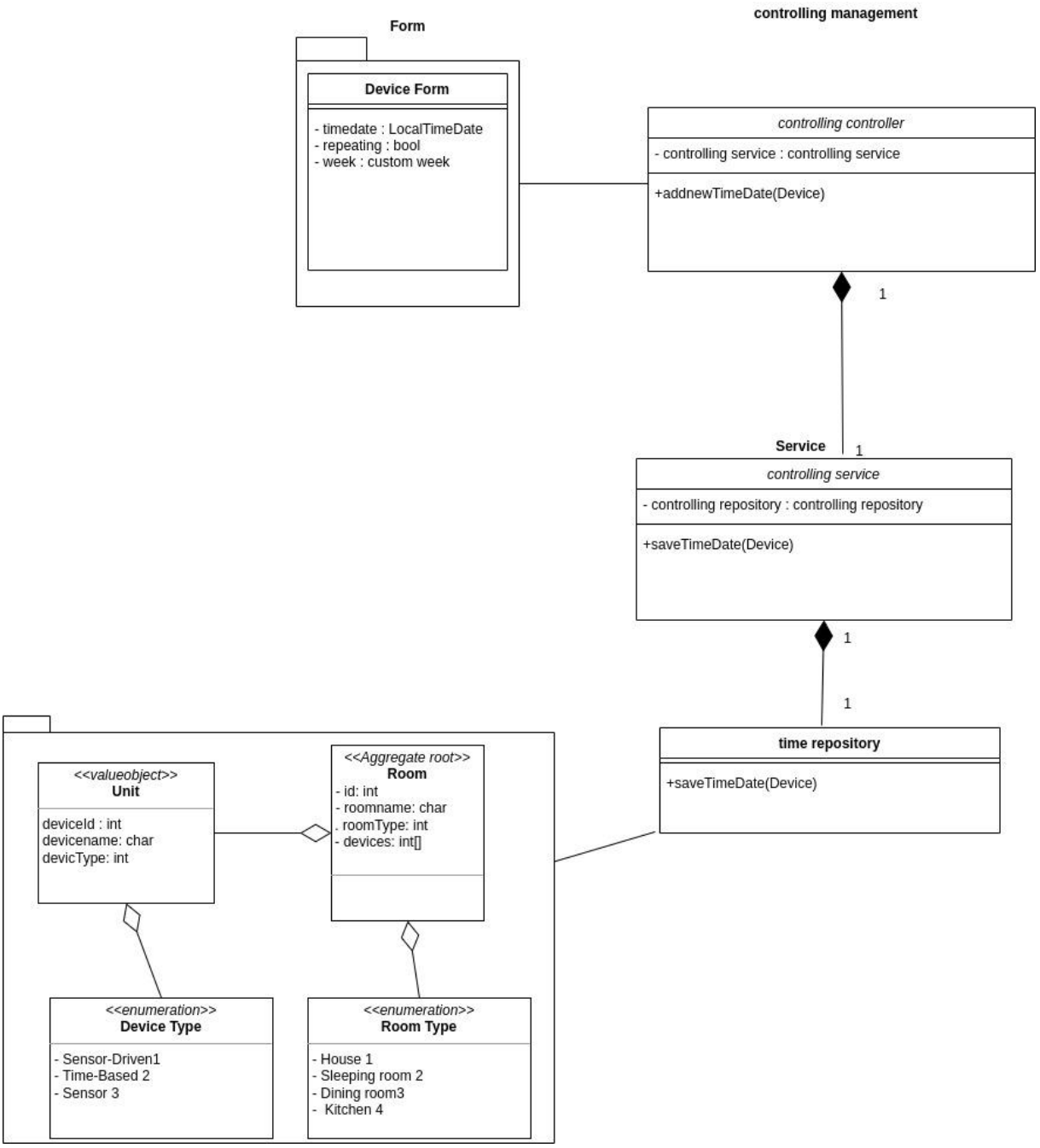


Figure 18 Class Diagram Time management

3.3 Process View

In this section the scenarios are transferred to the process view. Note that the proximity
can cause redundancy there. We have decided to map them nonetheless as this ensures a complete overview of all processes. A further advantage of our decision is the fact that involved areas can be clearly identified. This simplifies the process of implementation, as dependencies can be easily identified.

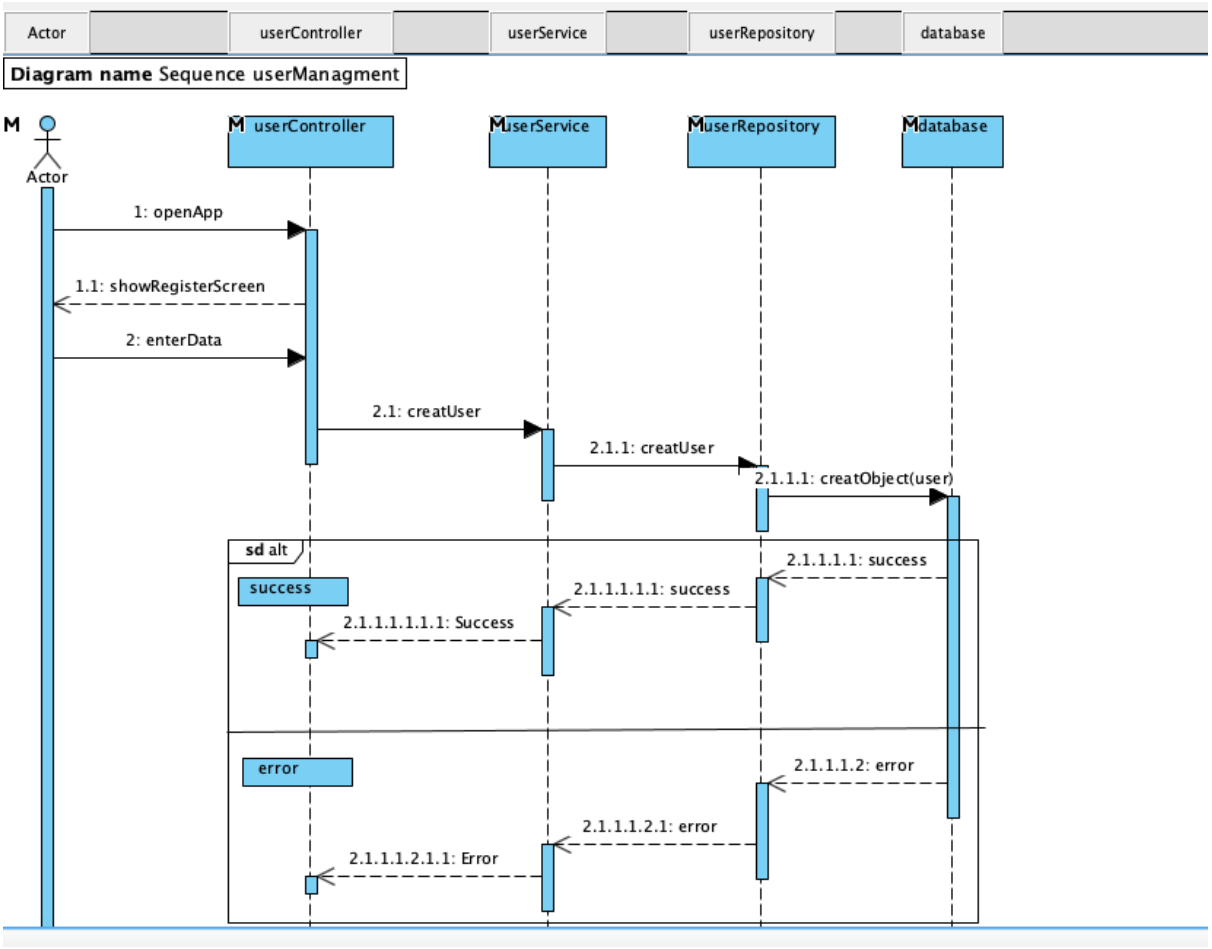


Figure 19 Sequence Diagram User Management

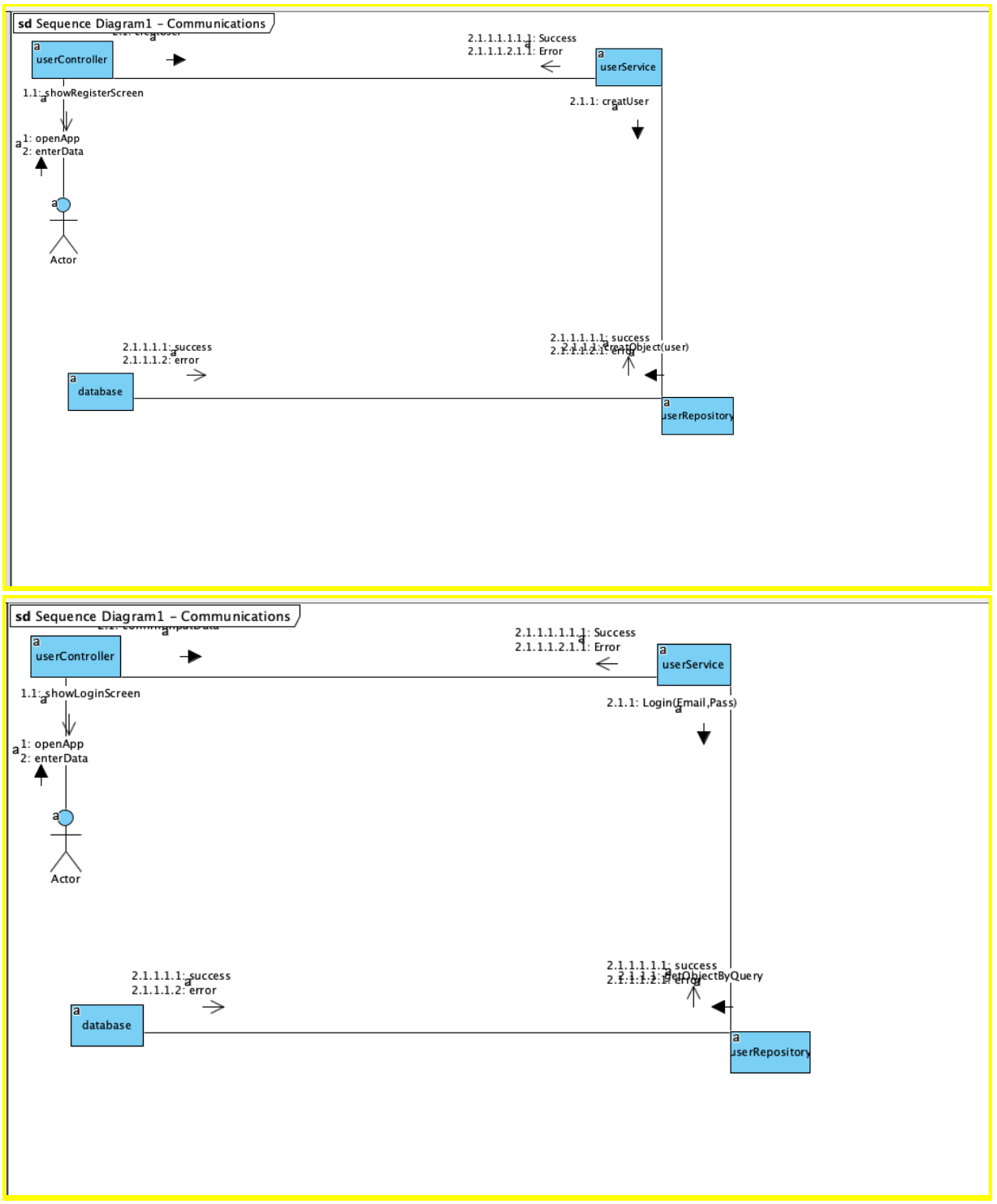


Figure 20 Communication Diagram User Management

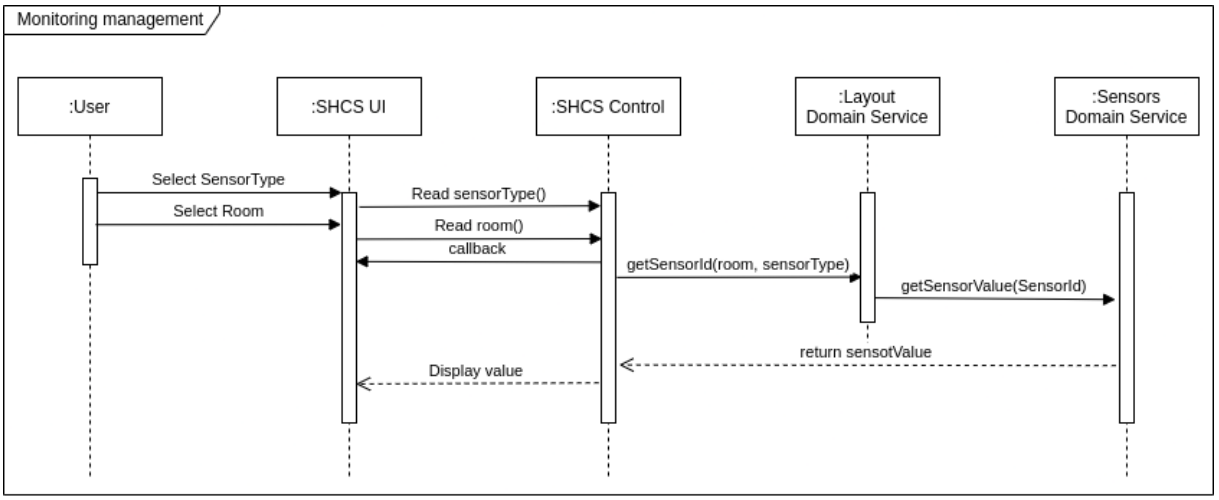
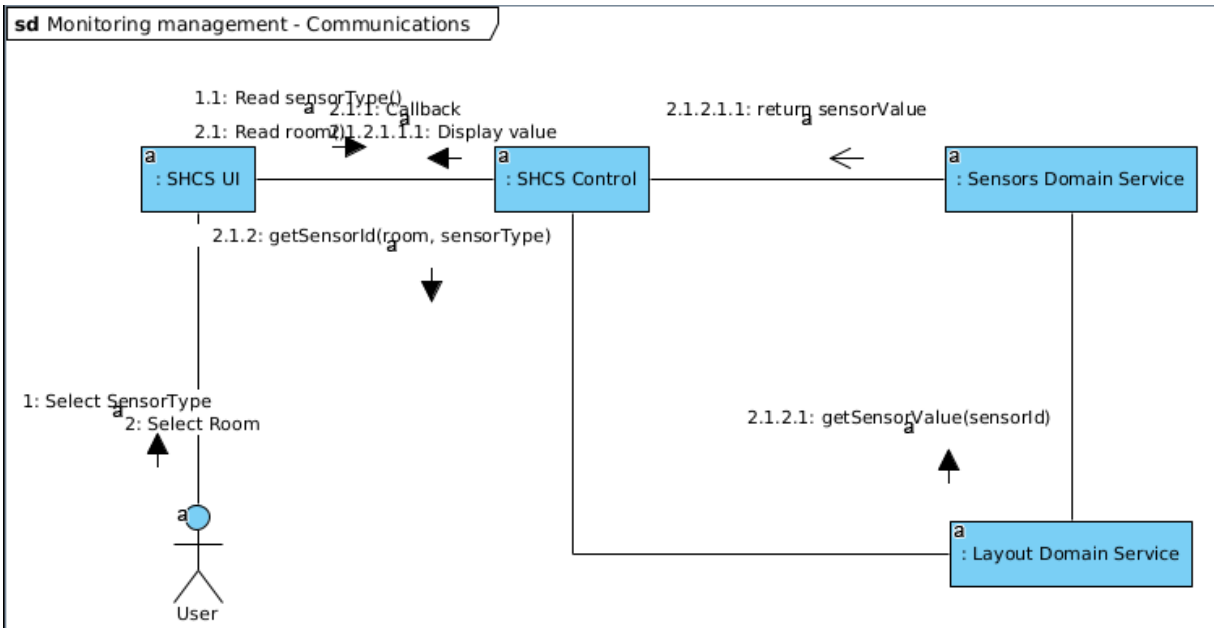


Figure 21 Sequence Diagram Monitoring Management



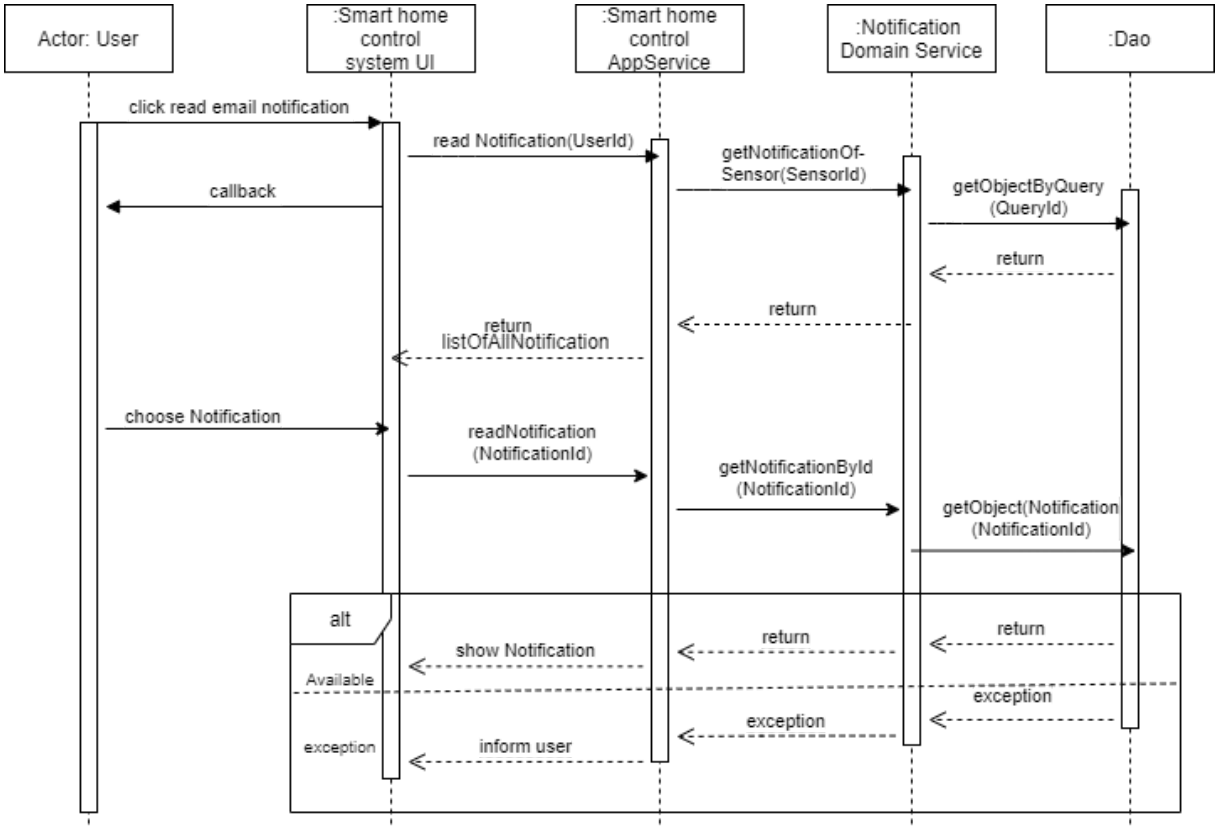


Figure 22 Sequence Diagram Read Notification

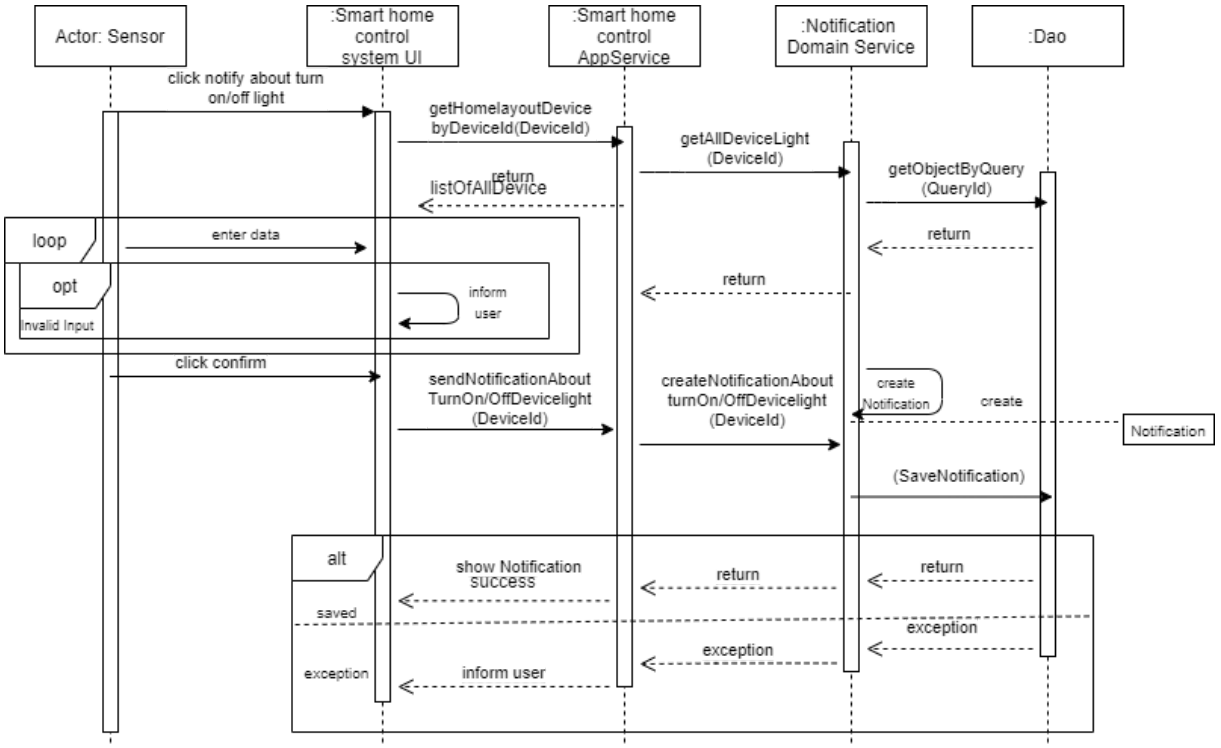


Figure 23 Sequence Diagram Notify about Light turn on/off

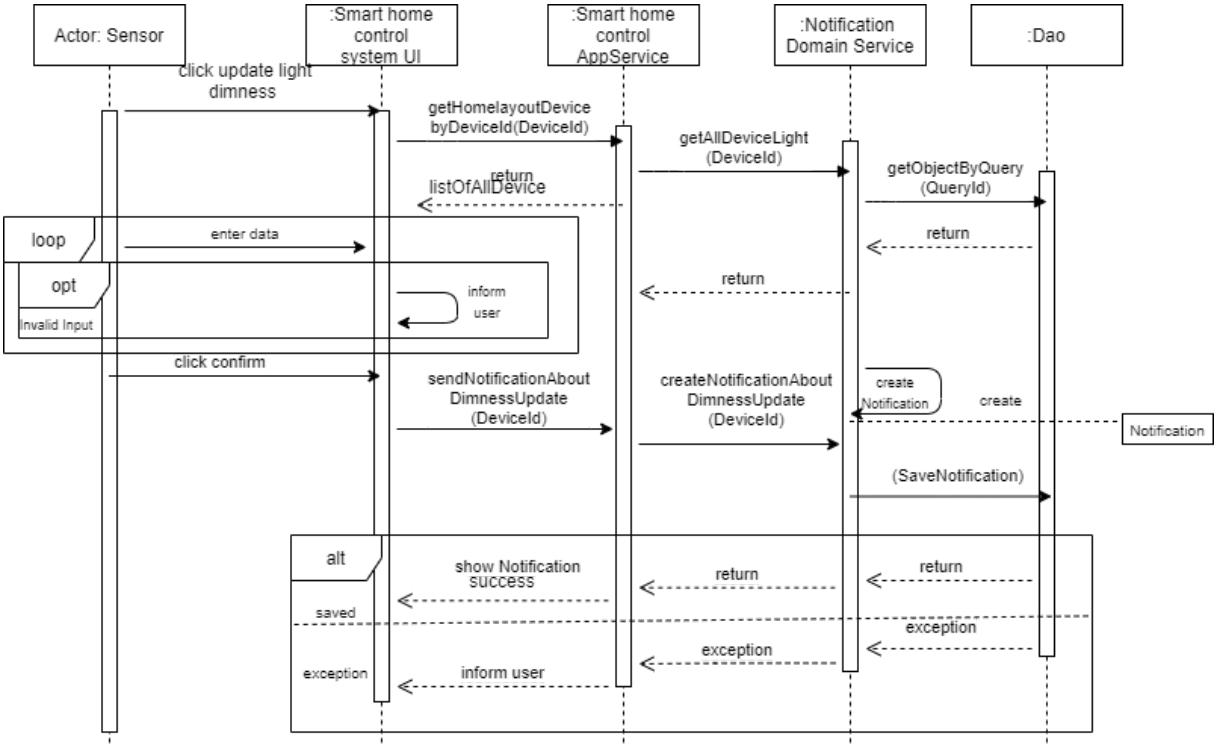


Figure 24 Sequence Diagram Notify about light Dimness update

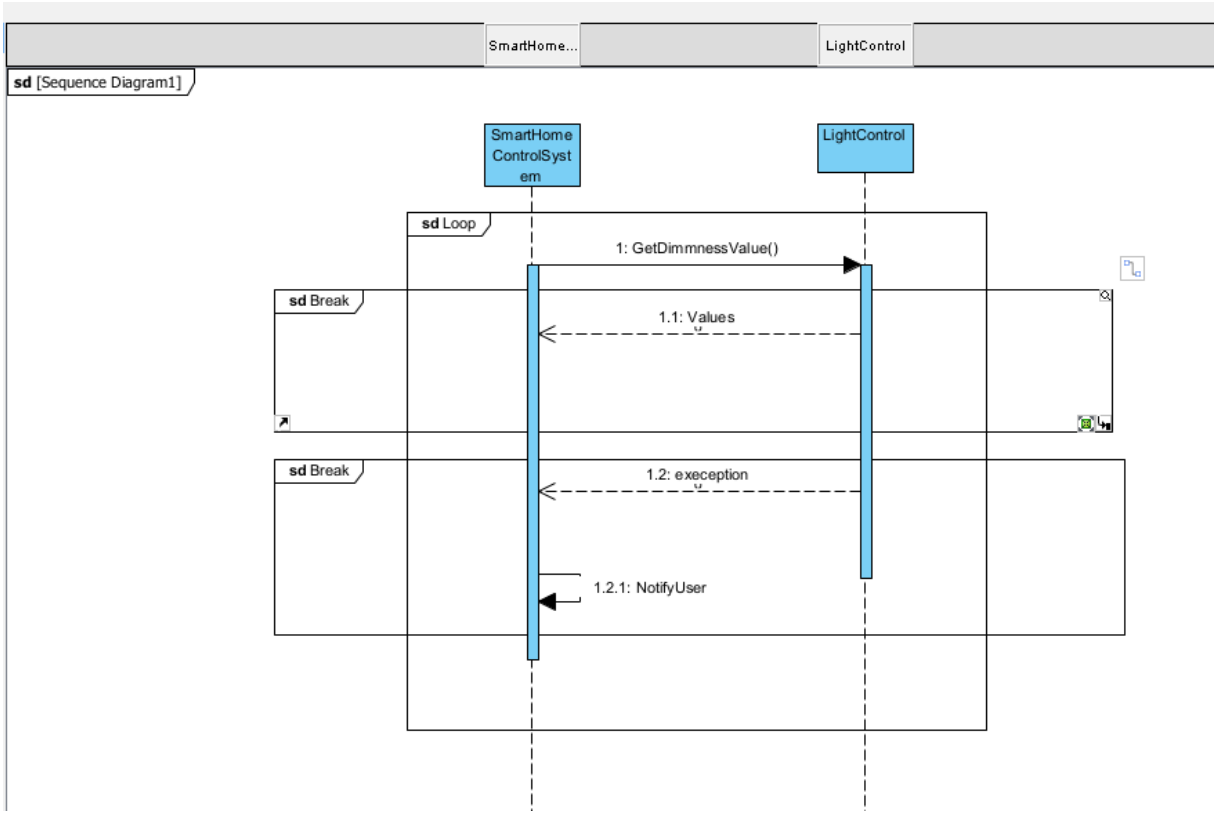
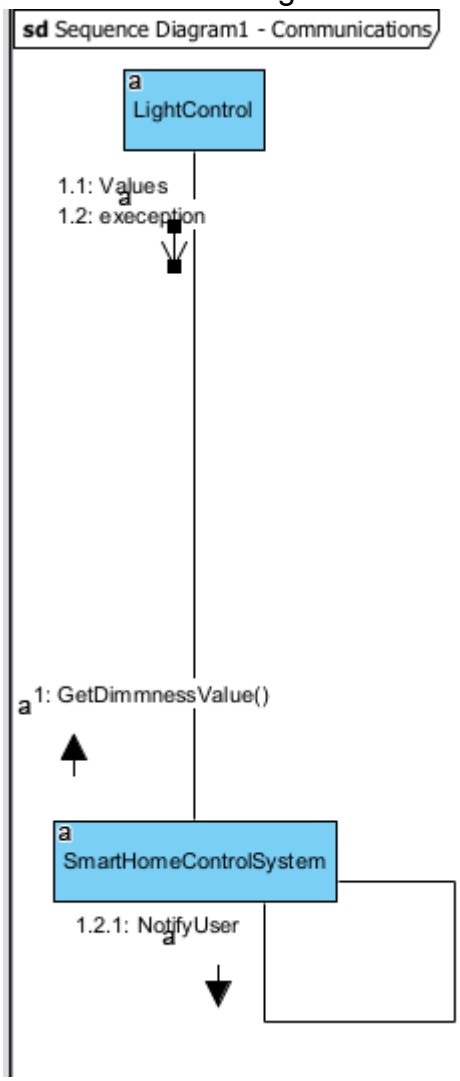


Figure 25 Sequence Diagram- measure light Dimness

Communication Diagram - Measure light Dimness



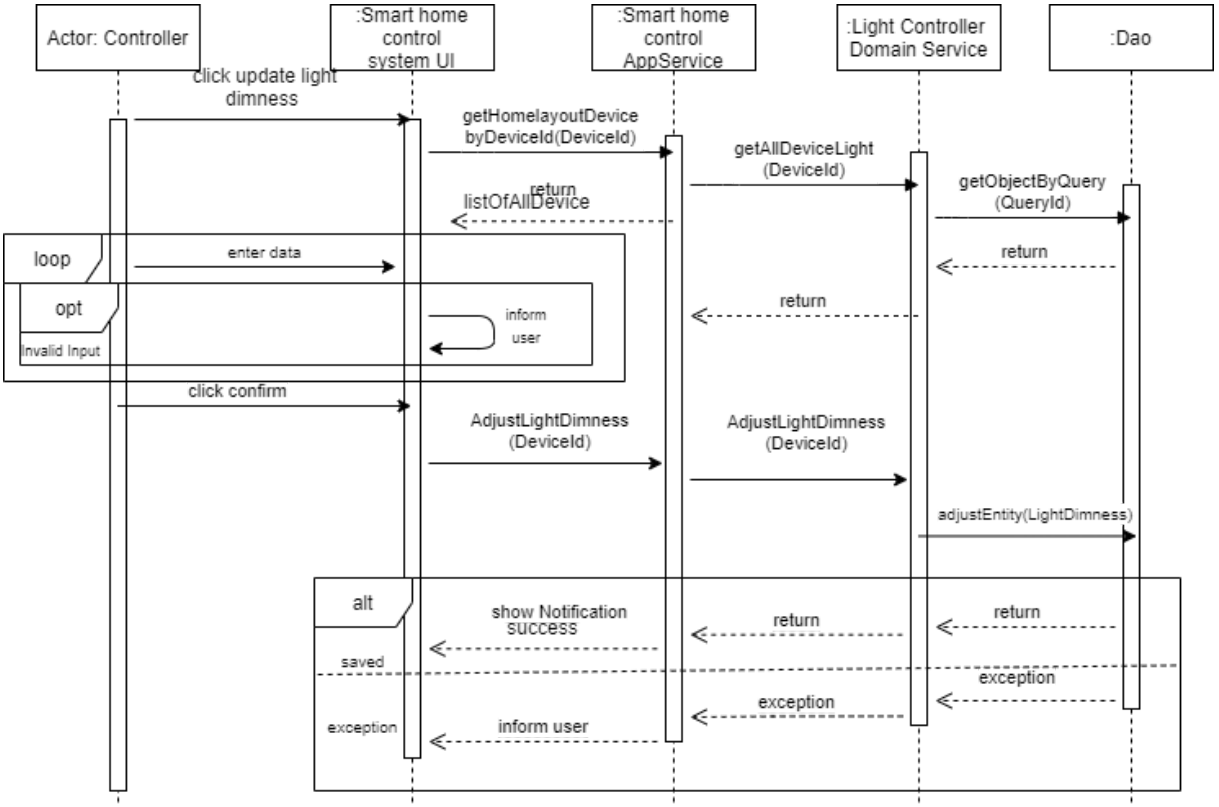


Figure 26 Sequence Diagram- adjust light Dimness

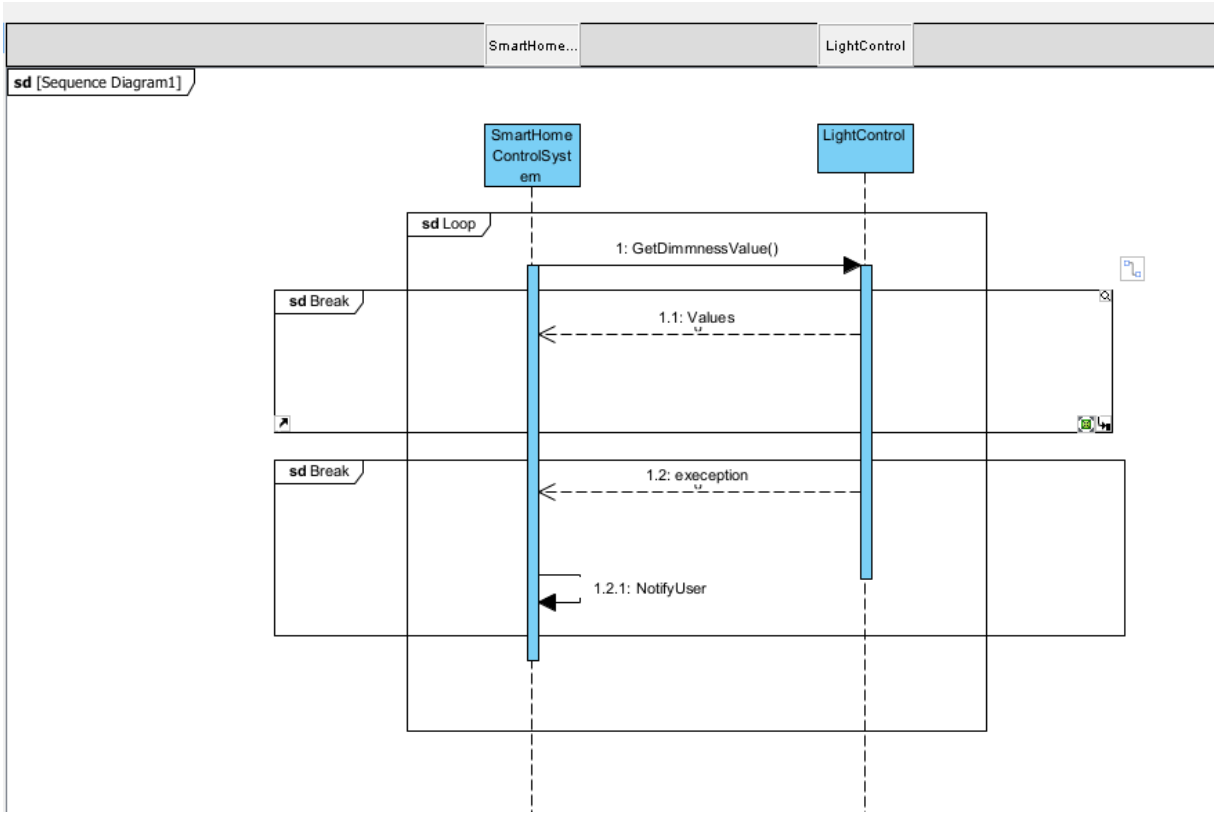
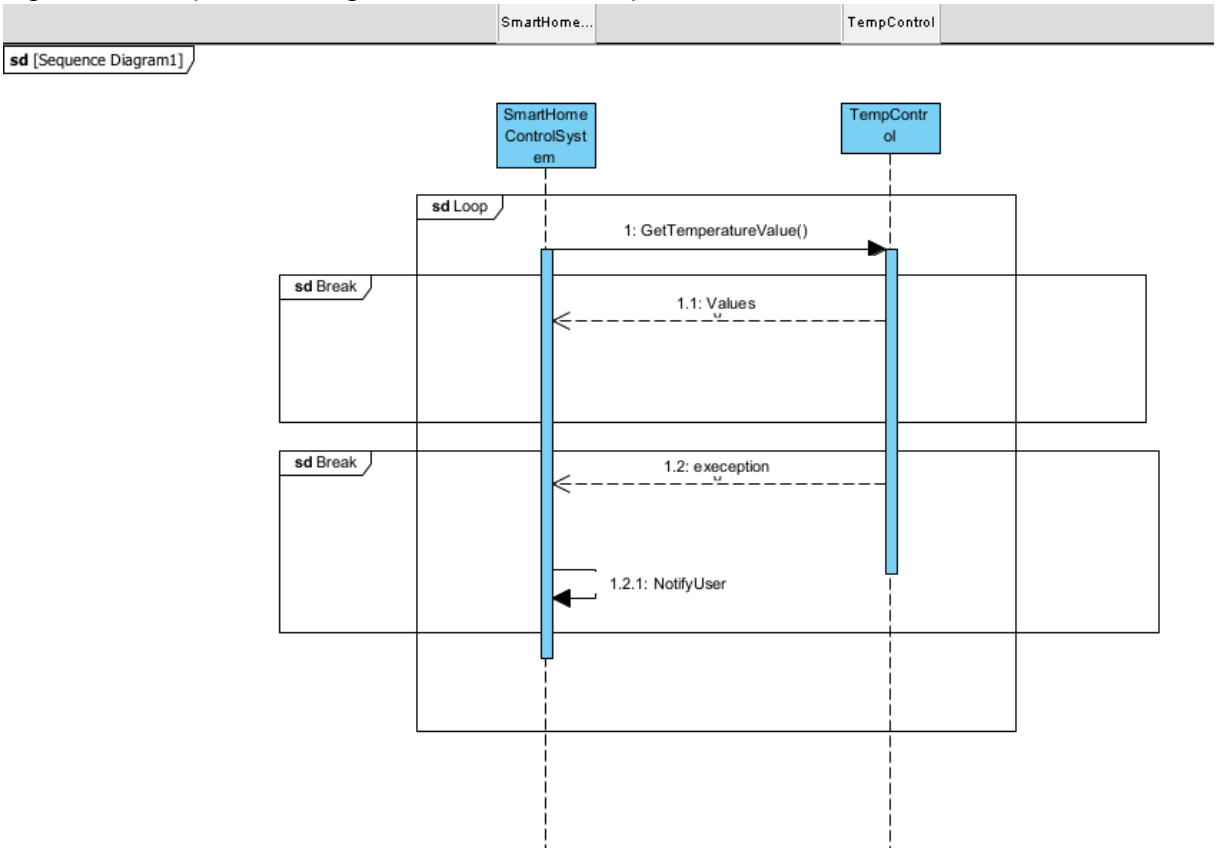
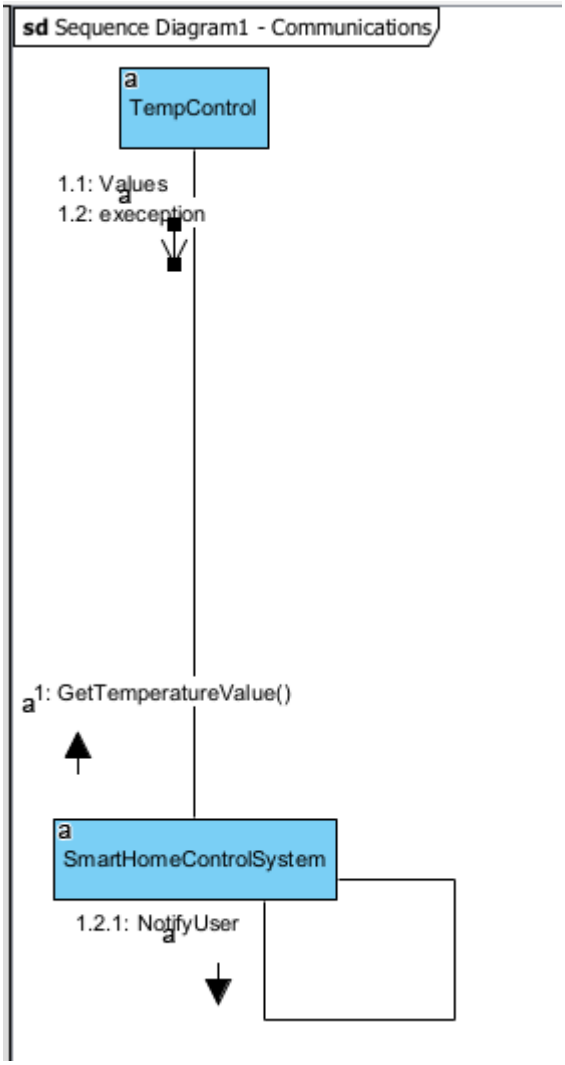


Figure 27 Sequence Diagram- measure Temperature



Communication Diagram - Measure Temperature



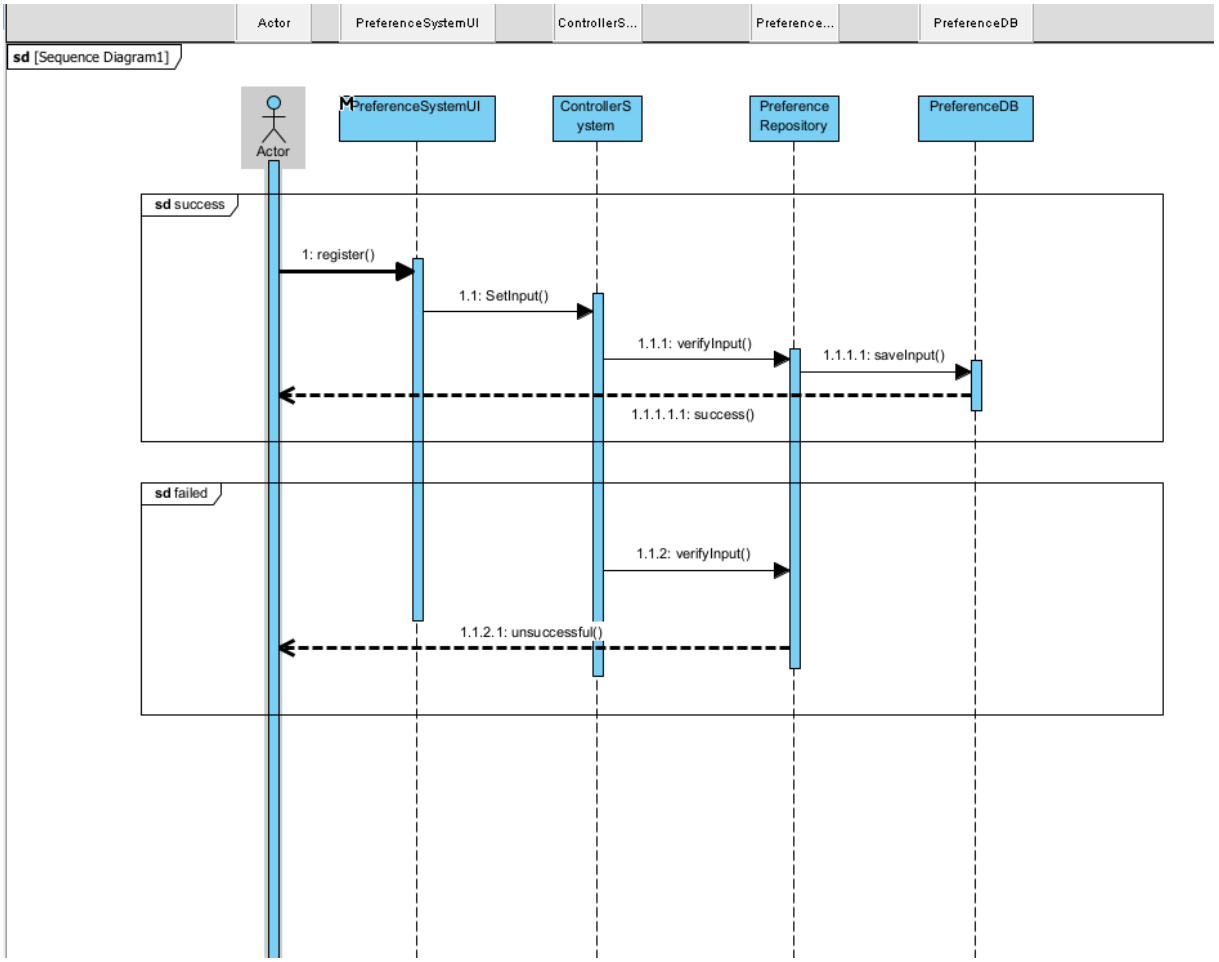
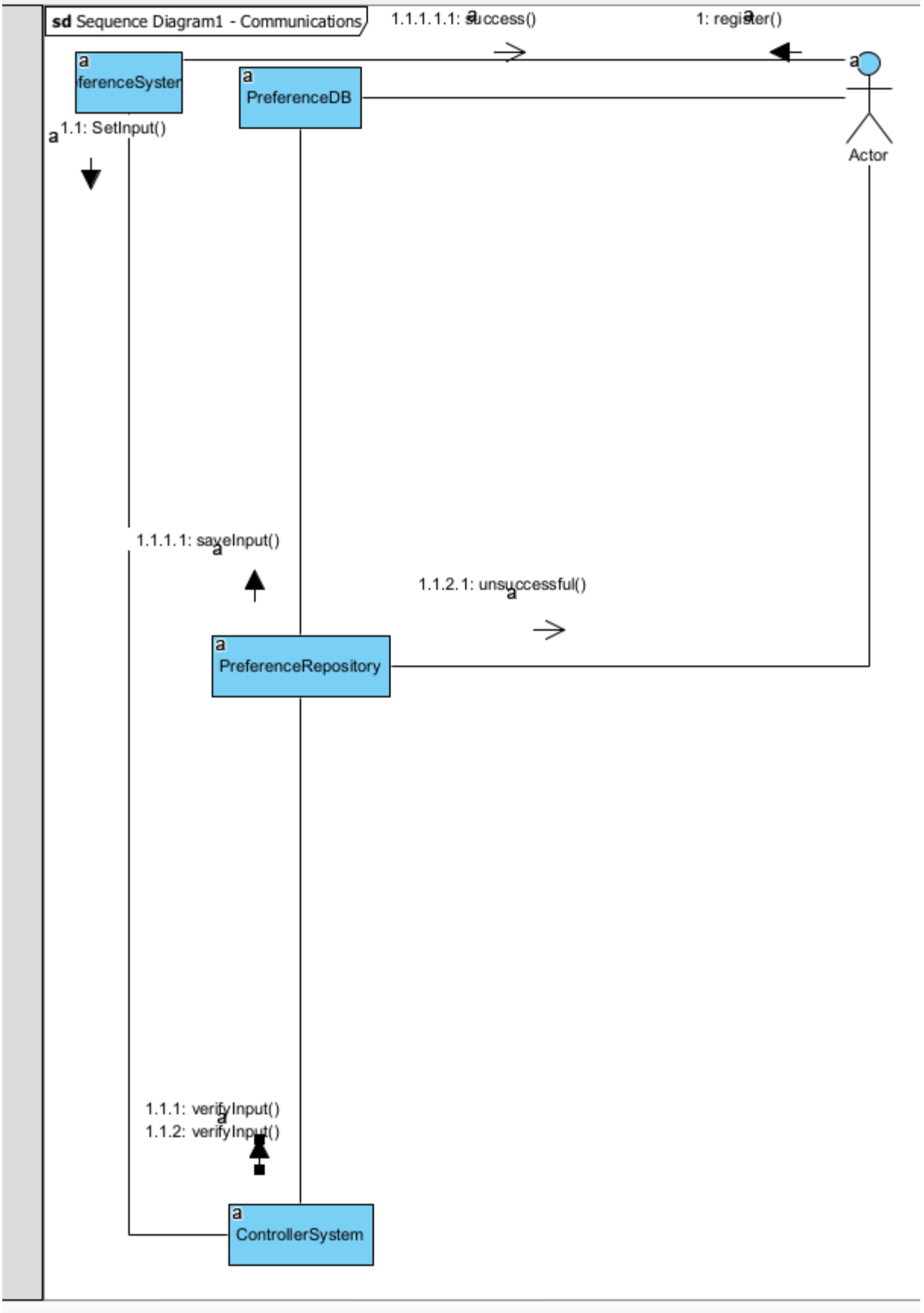


Figure 28 Sequence Diagram Preference Management

Preference Management Communication Diagram



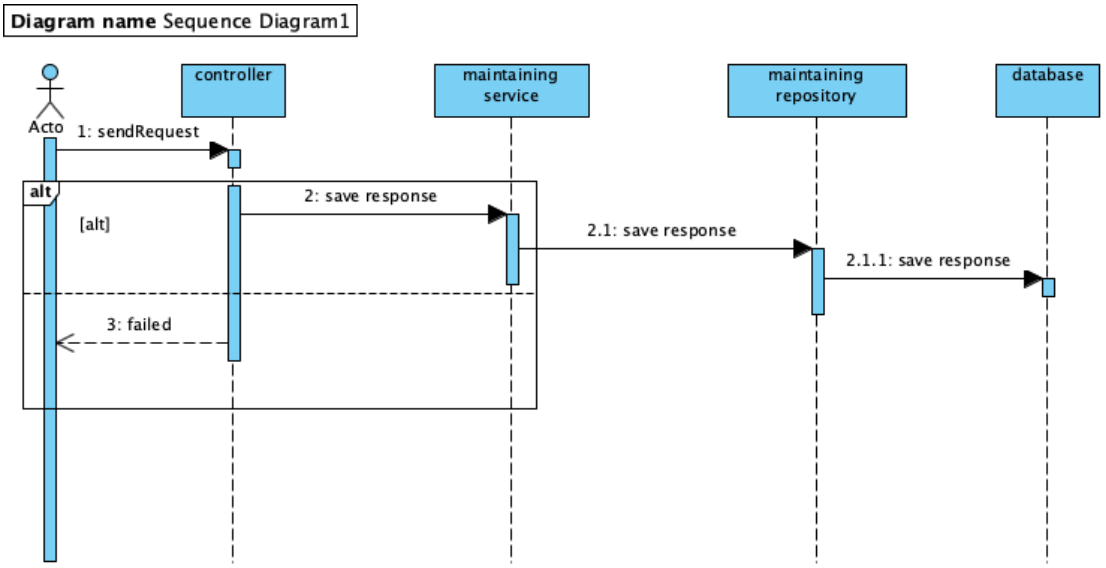
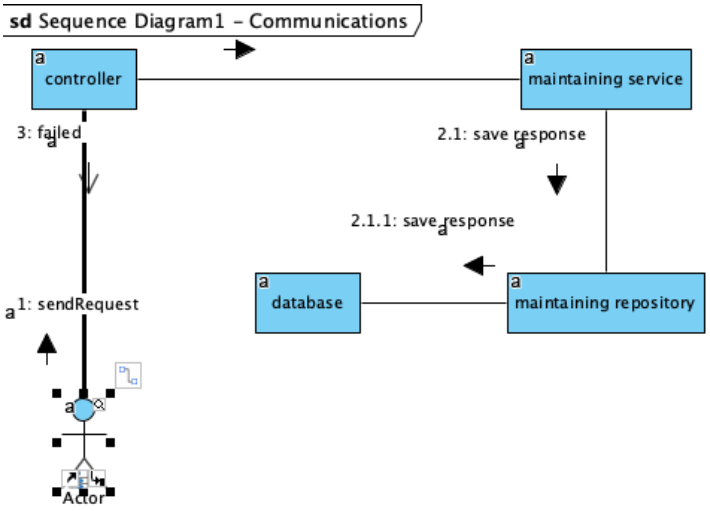


Figure 29 Sequence Diagram Maintenance Management



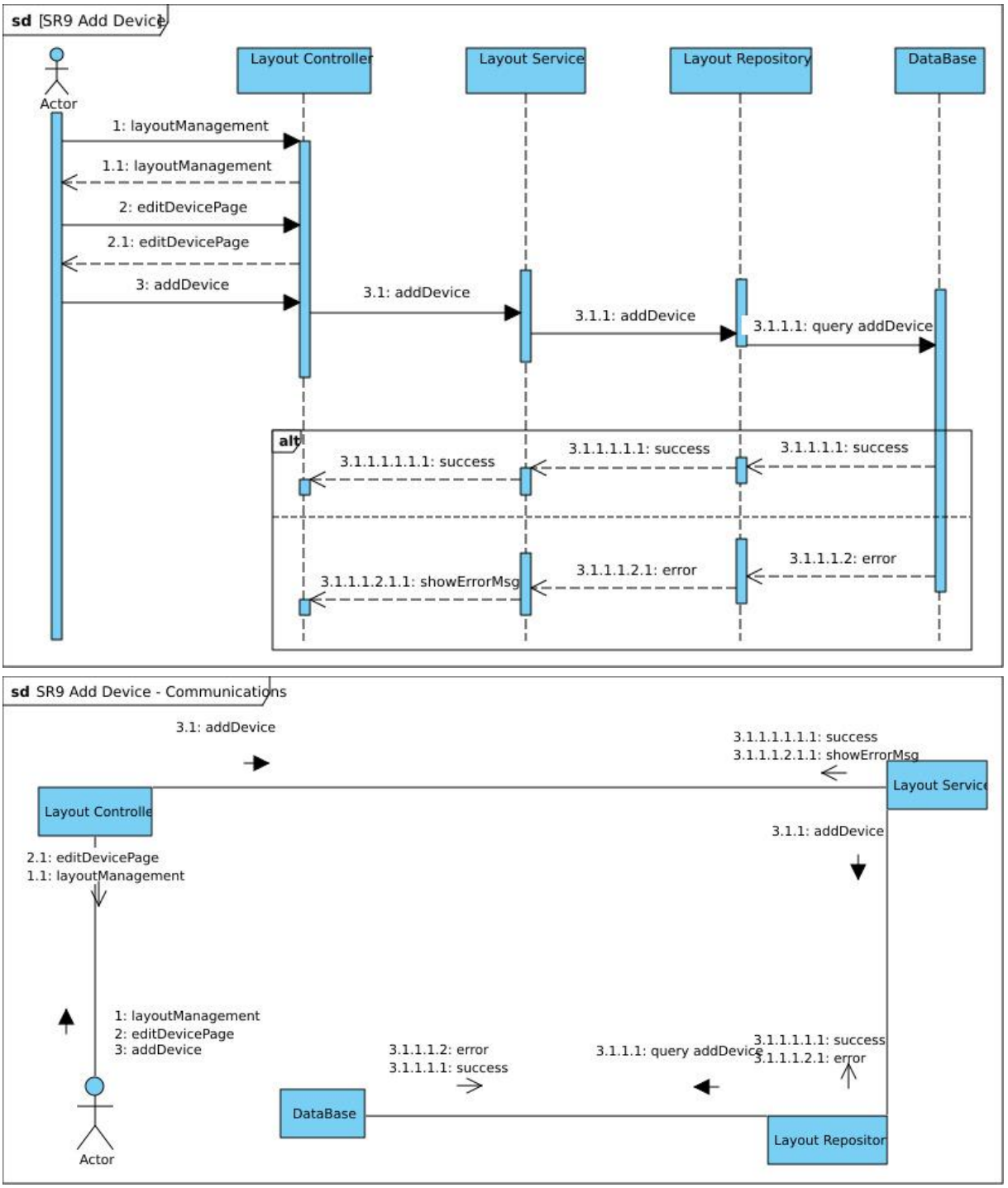


Figure 30 Layout Management Add Room

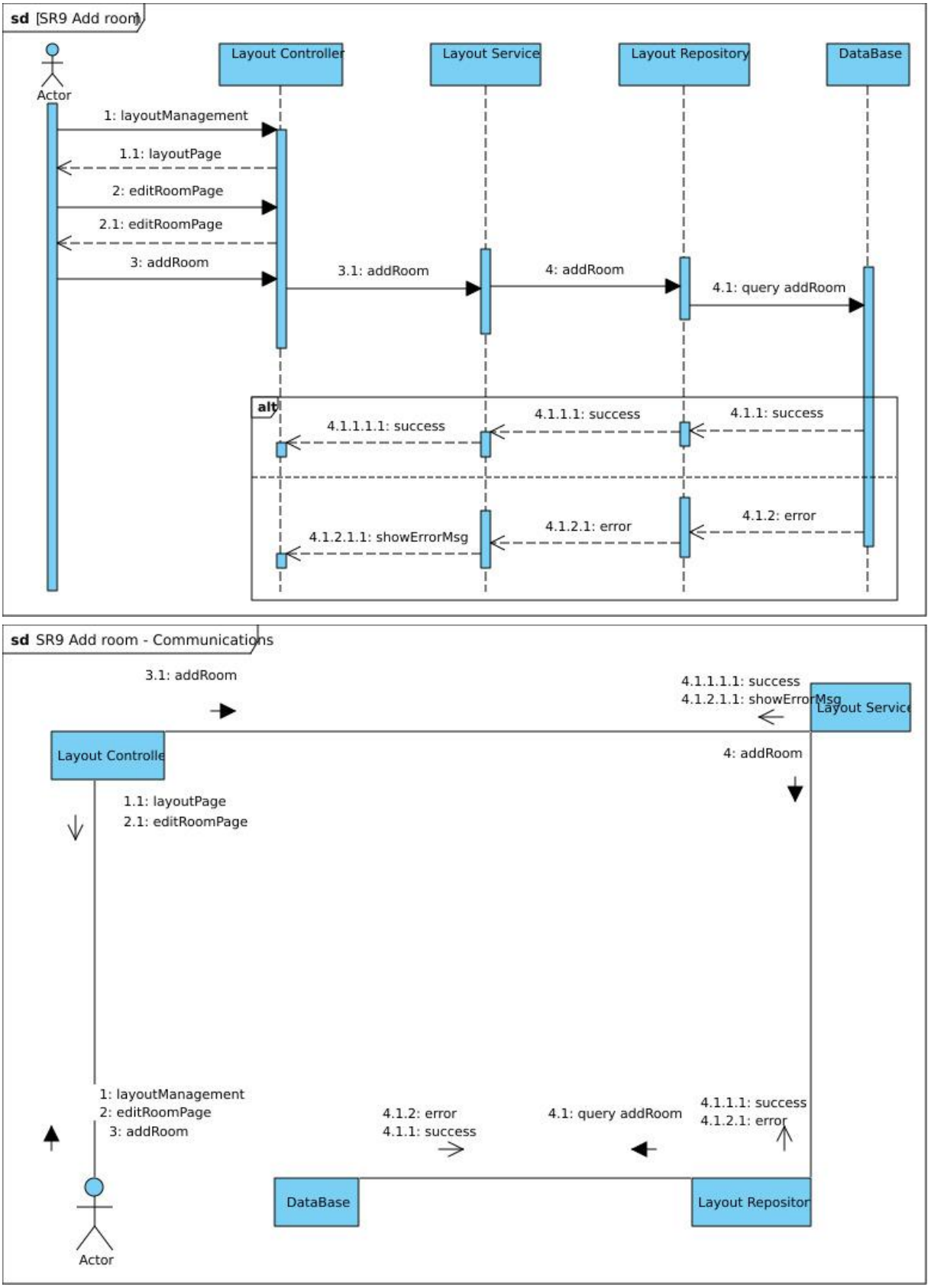


Figure 31 Layout Management Remove Device

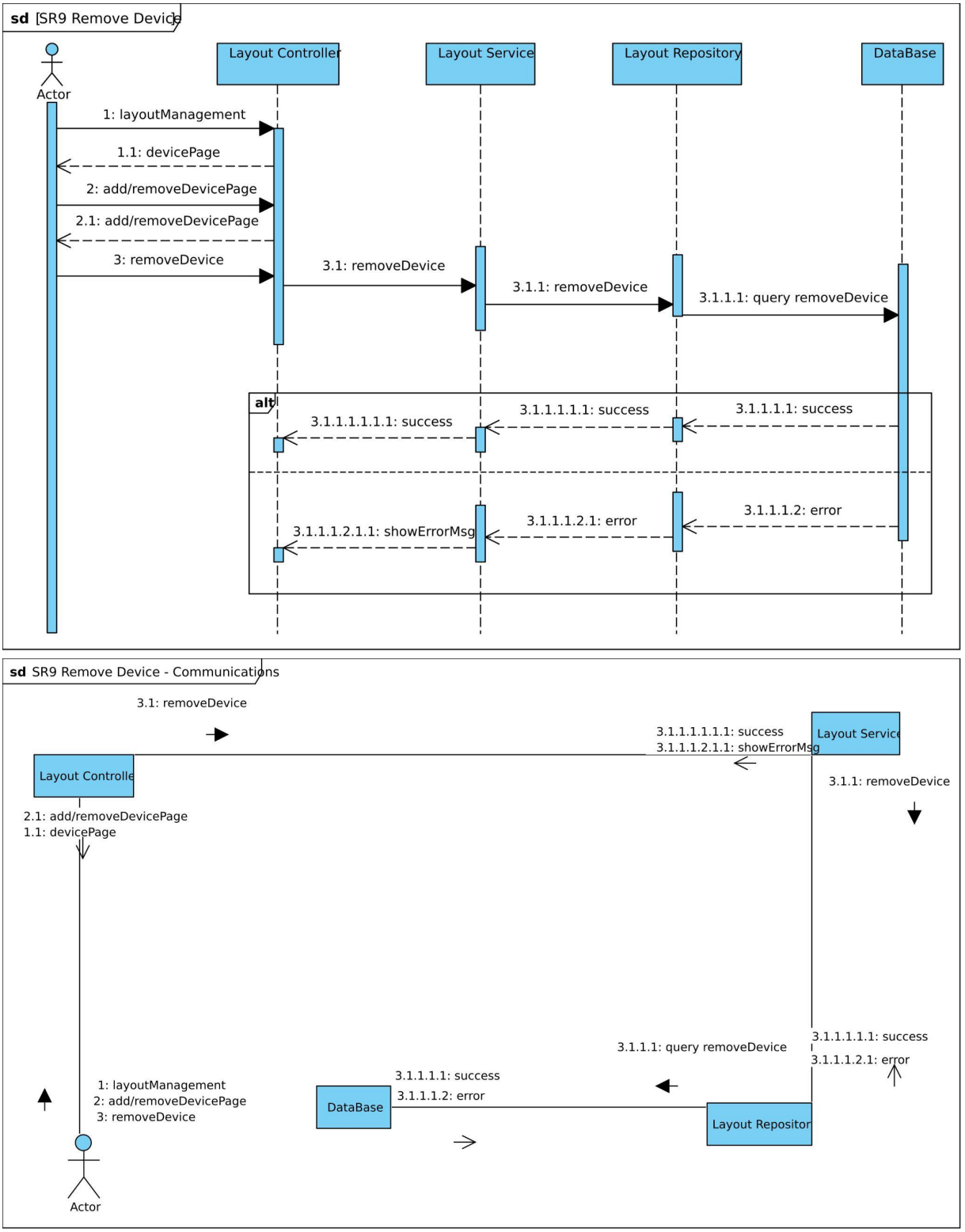
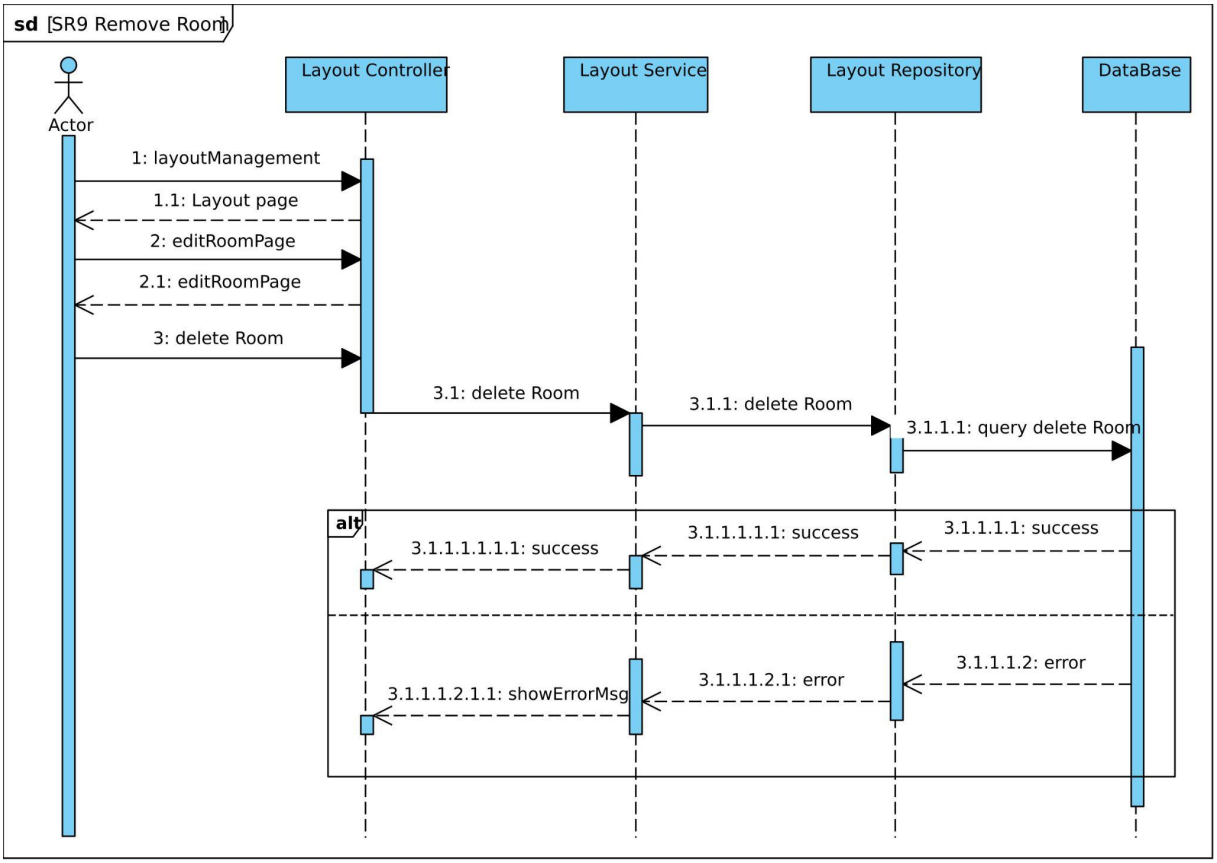


Figure 32 Layout Management Remove Room



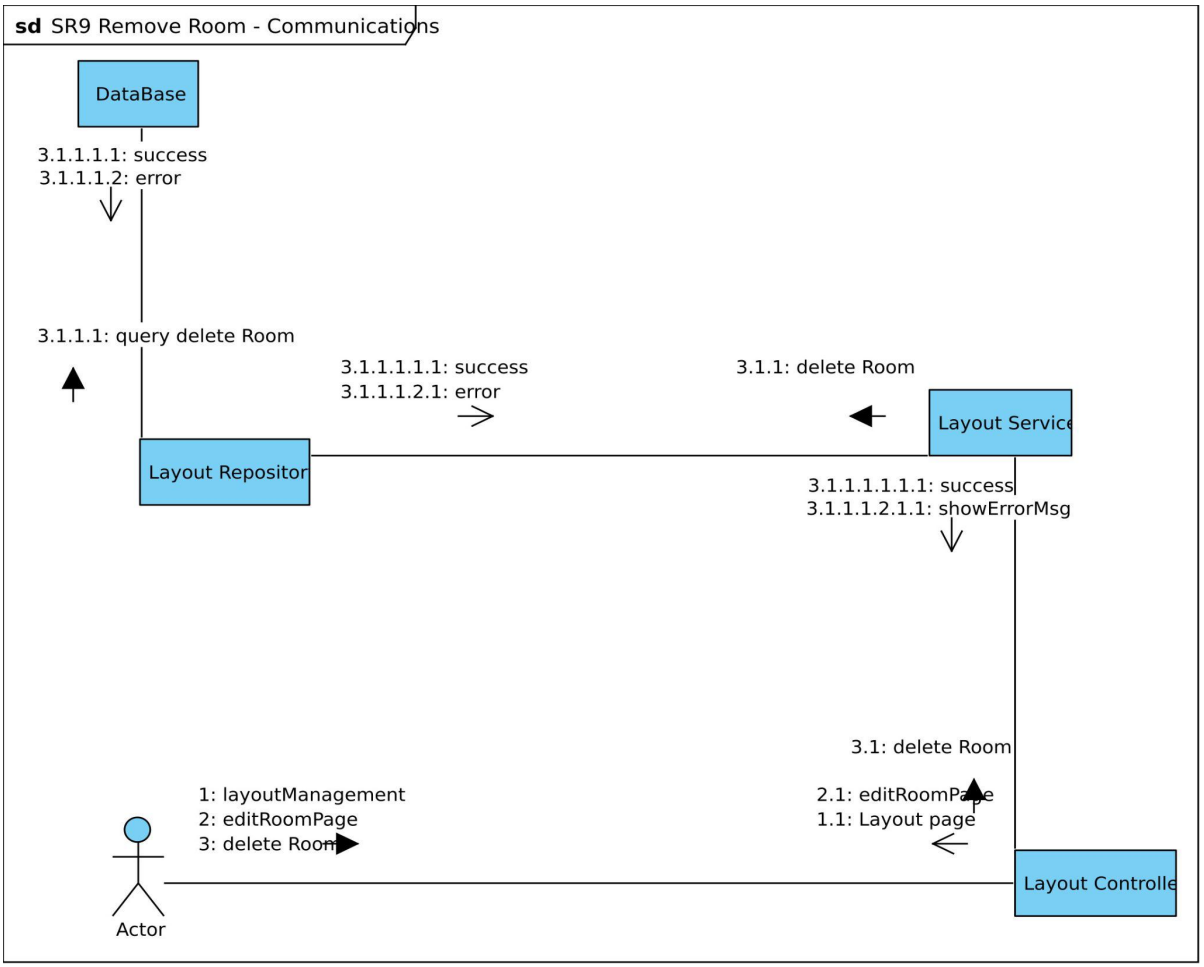


Figure 33 Layout Management Update Device

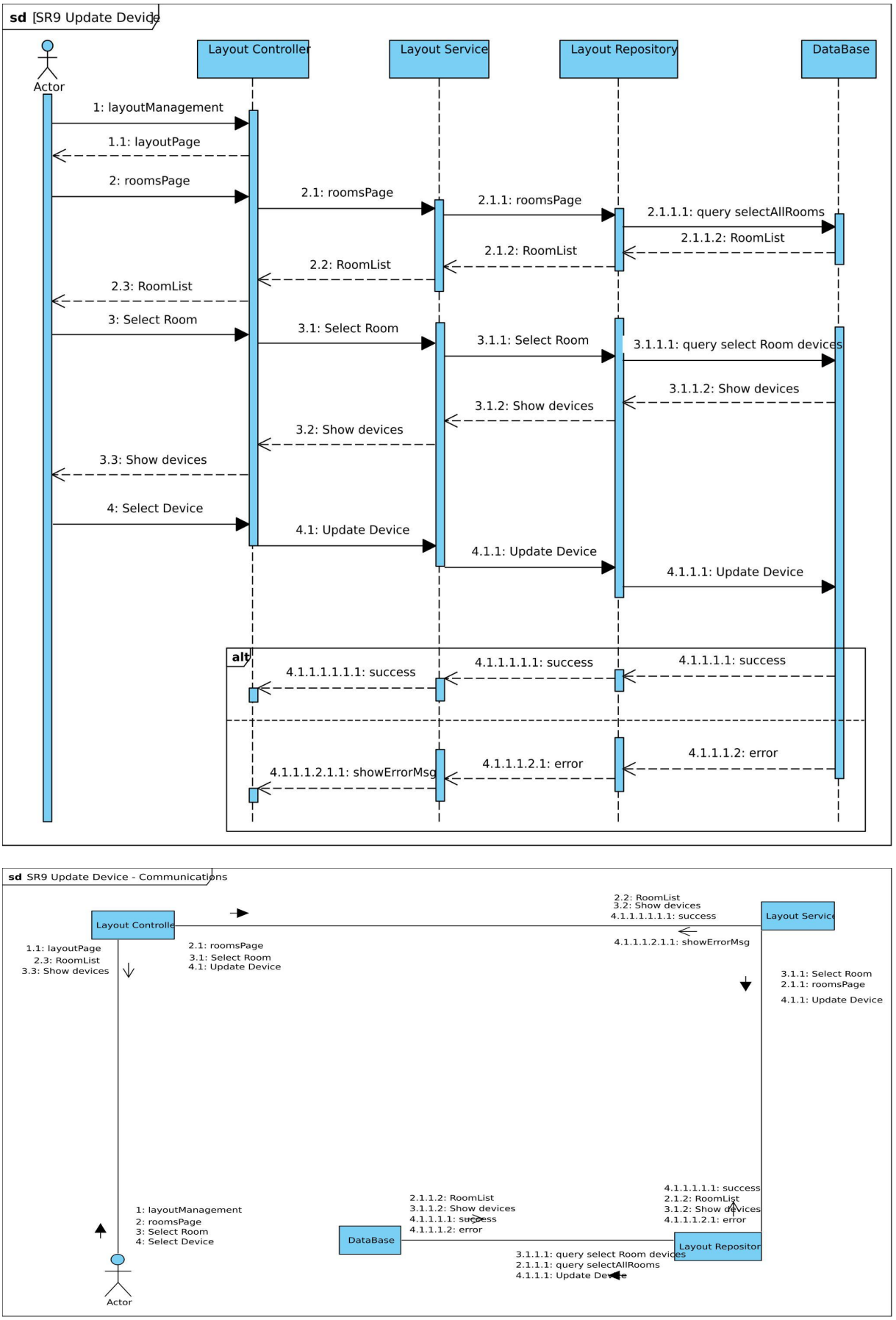


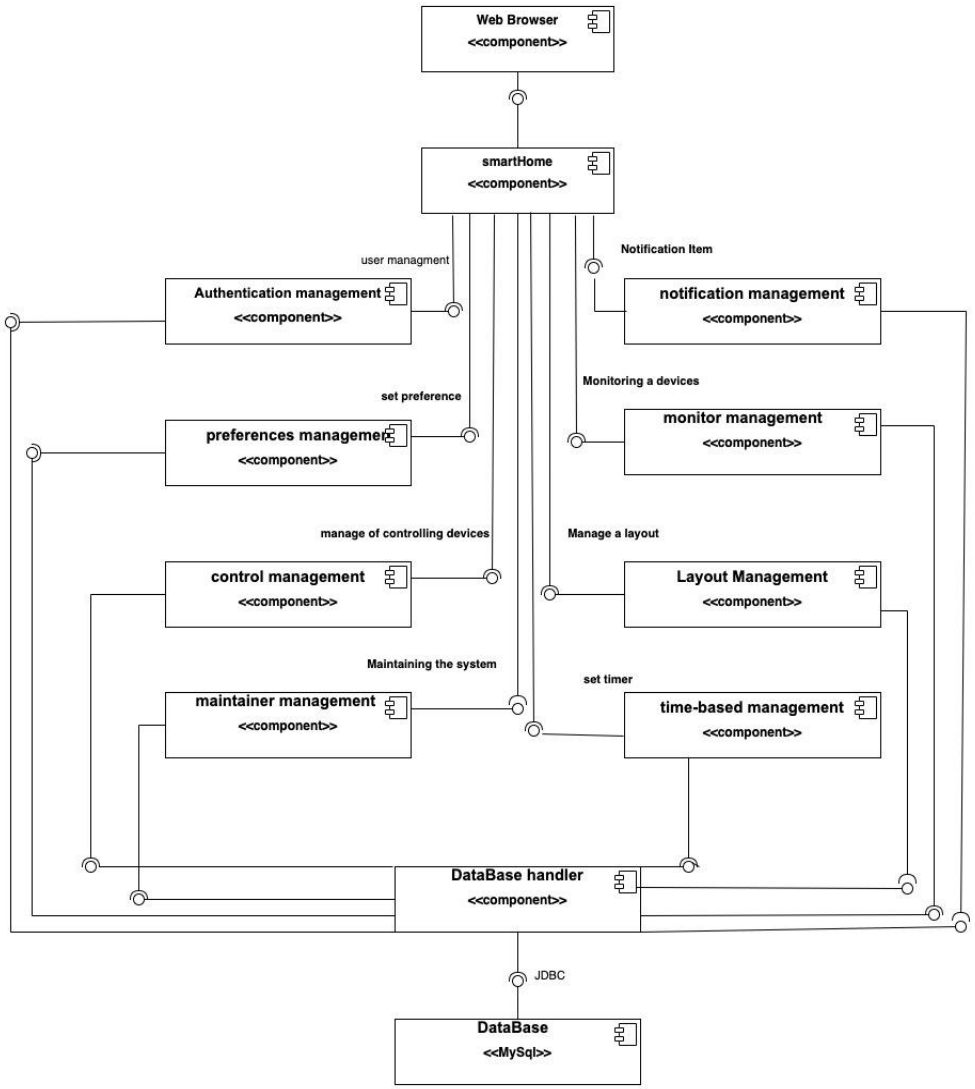
Figure 34 Update Device

3.4 Development View

We have decided that our application will be set up as a web application. Users can access the application on their devices using a web browser. To access each of the

user interface components from a Web browser, the user must log on with valid credentials. Each user interface provides functions that are specific to the user group in

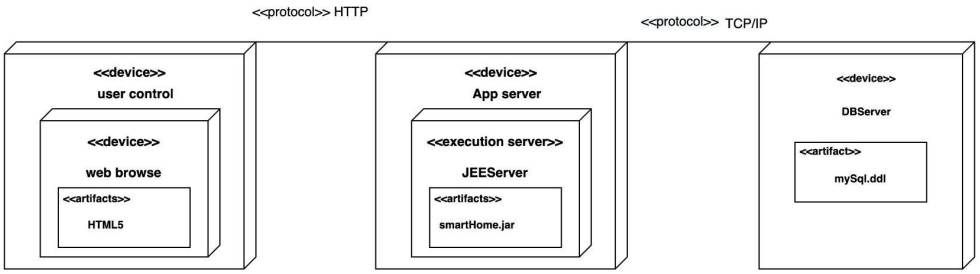
which the application is used for. All components that control the input of the user interfaces are in the screenshot. The Smart_Home is the central point of the application. The Database handler area acts as the connection between the application and the database.



3.5 Physical View

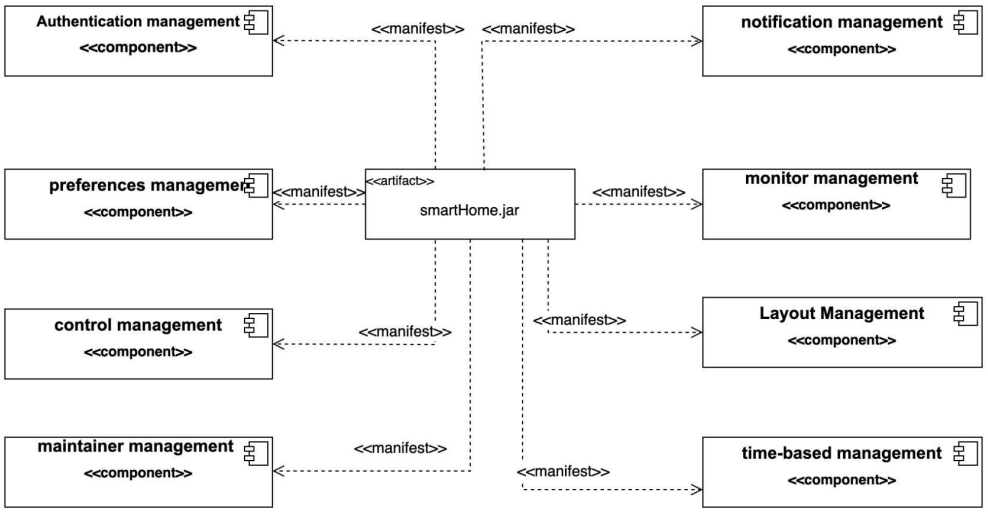
The deployment diagram shows the hardware components of our smartHome. Our artefacts are assigned to their respective deployment targets (nodes), i.e. hardware components or execution environments .

The smartHome is deployed in the JEE Execution Environment. The MySQL database contains the data for our smartHome and is connected to our application via TCP/IP.



3.6 **Deployment diagram**

The artefact smartHome.jar manifests (implements) the components of our subdomains



3.7 Team Contribution and Continuous development method

3.7.1 Project Tasks and Schedule

#	Task	Start	Start	Duration	March				April			
					W1	W2	W3	W4	W5	W6	W7	W8
1	Design phase	11/03/2022	24/04/2022	45 days								
2	Define domains and contexts	11/03/2022	25/03/2022	15 days								
3	Analyze the use cases	25/03/2022	27/03/2022	2 days								
4	Use case diagrams	27/03/2022	01/04/2022	6 days								
5	Class diagrams	01/04/2022	14/04/2022	14 days								
6	Sequence diagrams	14/04/2022	22/04/2022	9 days								
7	Activity diagrams	21/04/2022	23/04/2022	2 days								
8	Deployment diagrams	01/04/2022	14/04/2022	14 days								
9	Component diagrams	01/04/2022	14/04/2022	14 days								
10	Review diagrams	14/04/2022	24/04/2022	11 days								
11	Document design phase	11/04/2022	24/04/2022	14 days								

3.7.2 Continuous Integration, Delivery and Deployment Plan

Review/approval procedure:

Every developer must not push directly to master, but rather create a temporary feature branch. When the developer thinks the code is ready, he can create a merge request in gitlab and tag the rest of the team. The team will review the changes and can add comments or give approval. At least two approvals are necessary to merge into master.

Continuous integration strategy:

By providing a .gitlab-ci.yaml file and setting configurations in gitlab we will enable continuous integration.

In different predefined situations we will execute different targets in order to validate code, check for bugs or ensure code quality.

The following targets will be provided:

- **lint:** The project is analysed by a linter with low error tolerance.
- **build:** The service is built in docker to make sure everything works.
- **test unit:** unit tests are being executed
- **test component:** component tests are executed
- **deploy:** the service is deployed
- **tag**: the current branch is tagged using git tag

The following events will be configured:

merge_requests

> When a merge request is created we will execute the build in docker of the particular service, run the unit tests and the component tests. In addition linting will be performed.

master

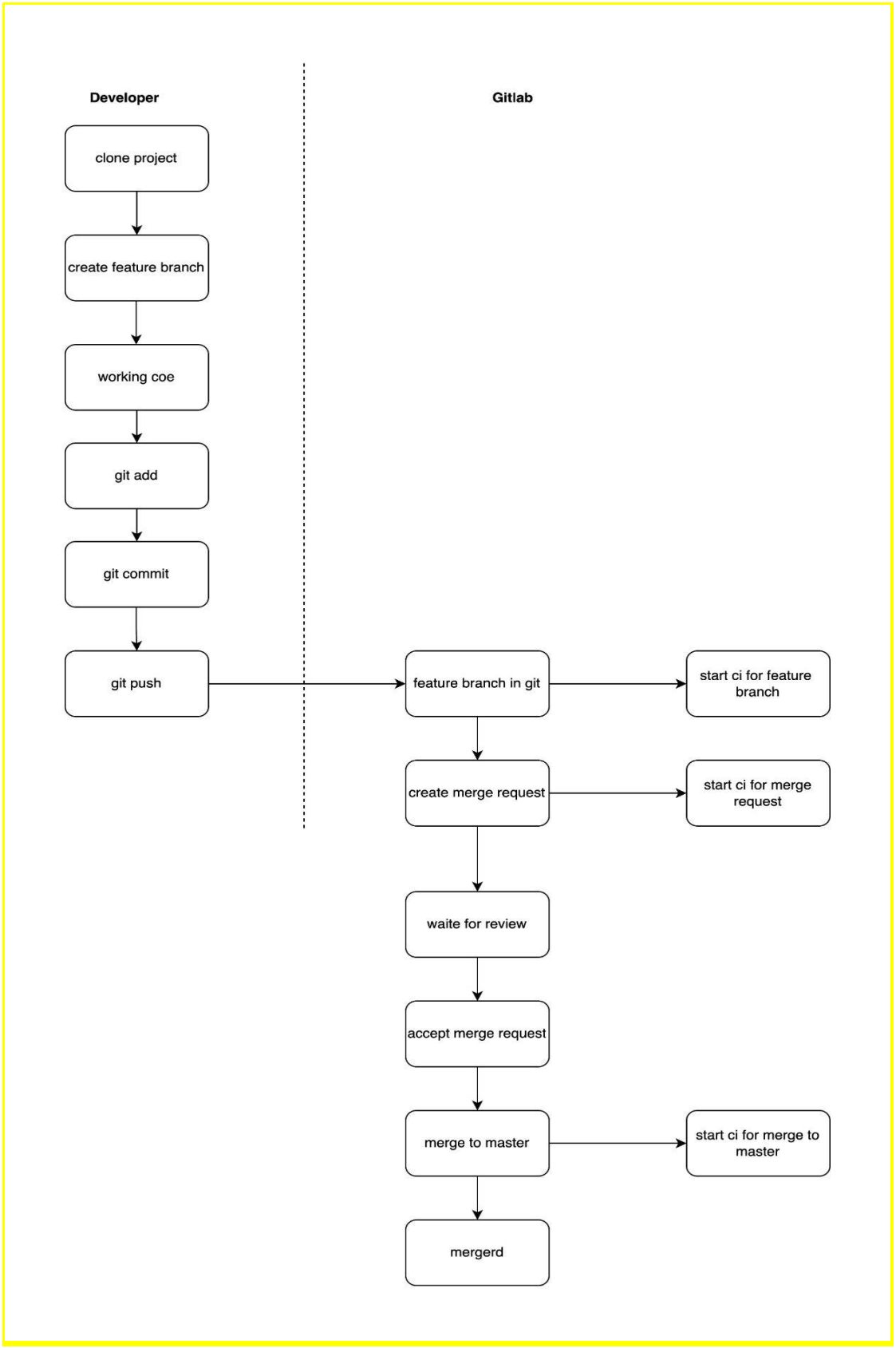
> When a merge request is approved, there will be a manual execution target to tag the current master branch using git tag with MAJOR.MINOR.PATCH (e.g. v1.1.5). This will be very useful for the deployment of a certain version.

feature branch:

> When pushing a feature branch, lint, build and unit tests are executed. This is a help for the developer to see if his code is clean before creating a merge request.

tags

> A tagged branch can be deployed using a manual deployment stage.


Developer
Gitlab
clone project
create feature branch
working coe
git add
git commit
git push
feature branch in git
start ci for feature branch
create merge request
start ci for merge request
waite for review
accept merge request
merge to master
start ci for merge to master
mergerd

3.8 Distribution of Work and Efforts

Contribution of Member 1:

Mohammadali Zaretorkmahalleh Time: 60h

- Design
  - Use caseView SR8,SR12
  - Use case Table SR8,SR12
  - Logical view SR8,SR12
  - Process view SR8,SR12
  - Component diagram
  - physical view
  - development view
  - deployment diagram
  - Design decision process view
  - Functional requirements : SR8,SR12
  - Skeleton implementation SR8,SR12
  - Continuous Integration, Delivery and Deployment Plan(documentation + diagram)

Contribution of Member 2:

Khadijeh Arabi Time 50H

- Design
  - Use caseView SR4&6,SR11
  - Use case Table SR4&6,SR11
  - Logical view SR4&6,SR11
  - Process view SR4&6,SR11
  - Sequential diagram SR4&6,SR11
  - Communication view SR4&6,SR11
  - development view
  - deployment diagram
  - Design decision process view

- ○ Mockup with SoapUI

Contribution of Member 3:

Reza Rezaie 50H

- Design
  - ○ Use caseView SR4&6,SR7
  - ○ Use case Table SR4&6,SR7
  - ○ Logical view SR4&6,SR7
  - ○ Process view SR4&6,SR7
  - ○ Sequential diagram SR4&6,SR7
  - ○ Communication view SR4&6,SR7
  - ○ Development view
  - ○ Deployment diagram
  - ○ Design decision process view
  - ○Task management via Trello

Contribution of Member 4 :

Mehdi Benabed 45H

- Design
  - ○ Design decisions monitoring management (SR3&5)
  - ○ Use case view and descriptions SR3&5
  - ○ Logical view SR3&5
  - ○ Process view SR3&5
  - ○ Functional requirements SR3&5
  - ○ Skeleton implementation SR3&5

Contribution of Member 5:

Behrad Hemati 50H

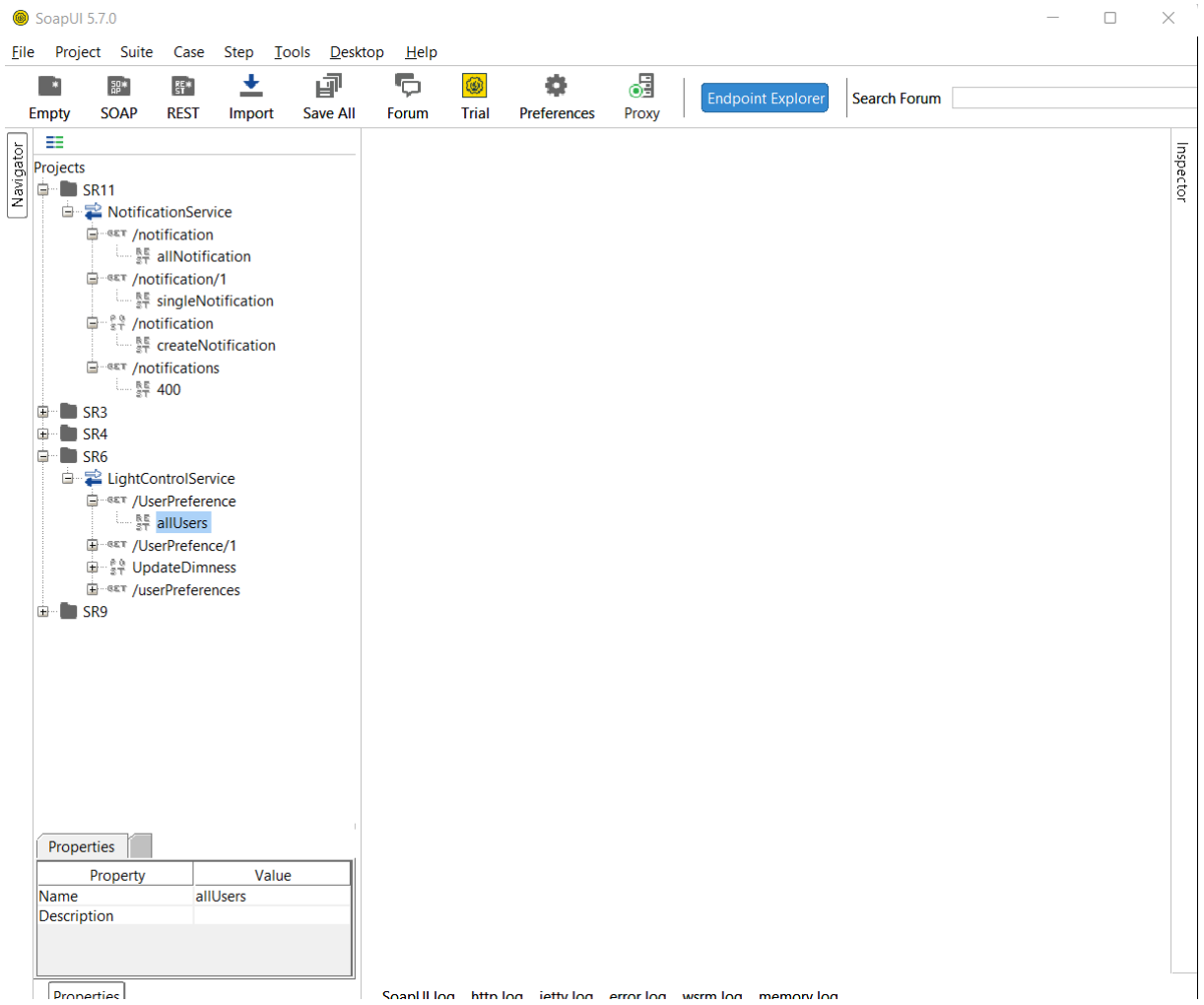
- Design
  - ○ Design layout management (SR9)
  - ○ Design time-based management (SR10)

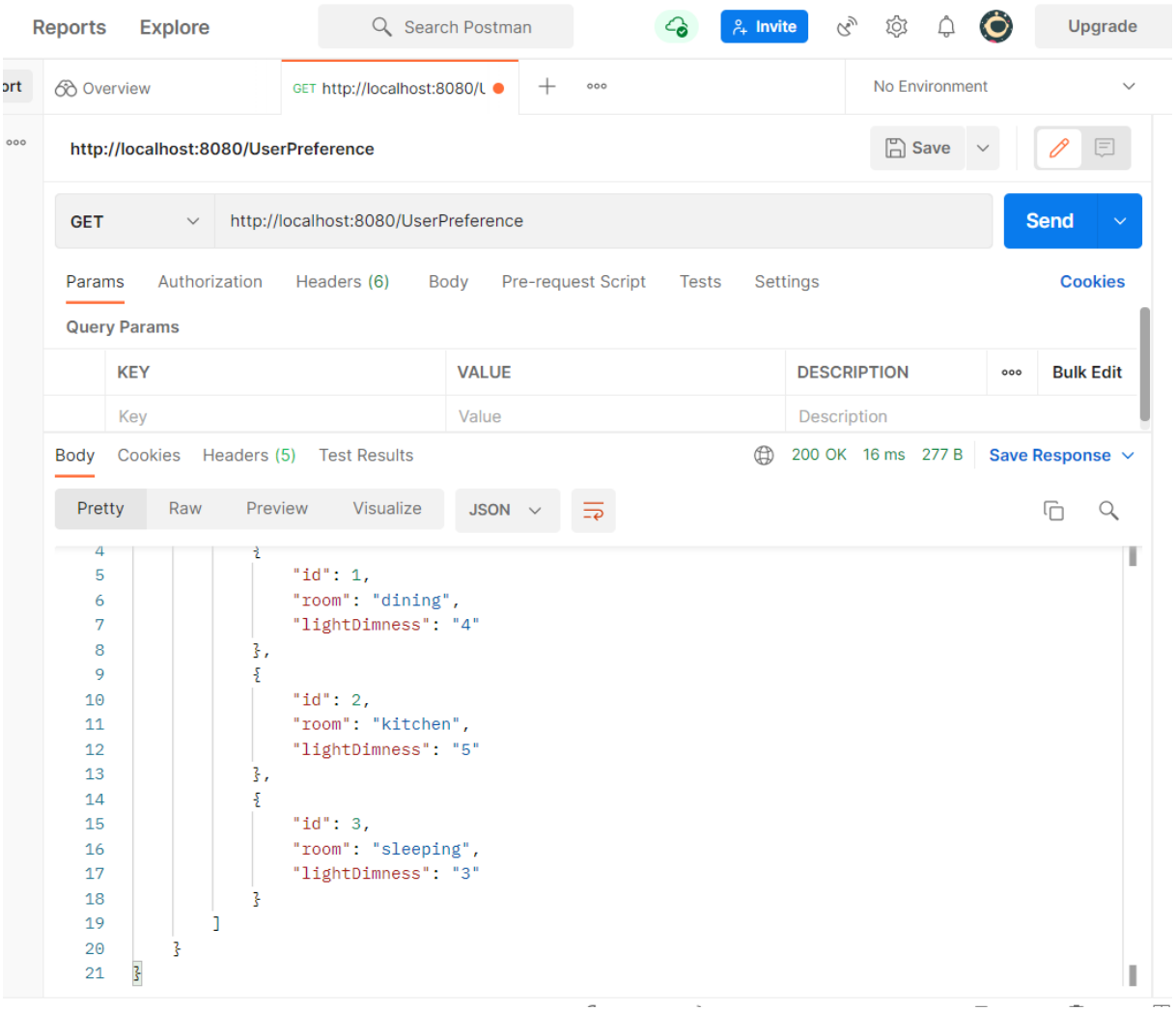
- Use case view and descriptions SR9&10
- Use case Table SR9&10
- Logical view SR9&10
- Process view SR9&10
- Sequential diagram SR9&10
- Skeleton implementation SR9&10

3.8 How-to / mock-up documentation

We mocked the individual Scope requirements as REST services with the mocking tool SoapUI. Our goal was to mock all important methods and to output the expected outputs for them. With this we simulated and tested the requests for our SmartHomeControlling system.

For example, we have created a GET request for the LightDimnessControl, which controls light Dimness of devices. The output values are hardcoded in JSON files. The queries and outputs contain the attributes from our domain model.


SoapUI 5.7.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Search Forum
Navigator
Inspector
Projects
SR11
NotificationService
/notification
allNotification
/notification/1
singleNotification
/notification
createNotification
/notifications
400
SR3
SR4
SR6
LightControlService
/UserPreference
allUsers
/UserPrefence/1
UpdateDimness
/userPreferences
SR9
Properties
Property
Value
Name
allUsers
Description
SoapUI log http log jetty log error log wsrm log memory log



The mockups can be done by importing the xml file into SOAPUI and then running the respective GET and POST requests via the browser, SOAPUI or an API testing tool such as Postman to see the expected outputs of the methods.

Build and install dependencies

cd $project_root/implementation/$ms_name

gradle clean build

Testing:

cd $project_root/implementation/$ms_name

gradle test

Run Service:

cd project_root/implementation/$ms_name

```
gradle compile exec:java -
Dexec.mainClass="at.ac.univie.ase.$service_name.AppName" -Dexec.args=""
```