Advanced Software Engineering  
DESIGN REPORT

|  |  |
| --- | --- |
| **Team number:** | 0508 |

|  |  |
| --- | --- |
| **Team member 1** | |
| **Name:** | Klavio Tarka |
| **Student ID:** | 12146253 |
| **E-mail address:** | a12146253@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 2** | |
| **Name:** | Michael Phan |
| **Student ID:** | 11839153 |
| **E-mail address:** | a11839153@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 3** | |
| **Name:** | Kevin Richardo Grote |
| **Student ID:** | 12143222 |
| **E-mail address:** | a12143222@unet.univie.ac.at |

This document suggests an outline for the required contents of the DESIGN report. The included descriptions and examples – highlighted in yellow - are supposed to help you write a clear report that documents and presents your actual solution well. Please remove this additional text (similarly written as this text) and exemplary material before you submit your report!

The DESIGN report document can be based on this template or can be written using an arbitrary text editing program such as Latex, LibreOffice etc., as long as the required material (described in the assignment document) is contained.

In general, write the report in such a manner as to provide all information (i.e., a clear and complete picture of the purpose of the project, the domain design decision space, the reasons behind major design decisions, a picture of the structure and current state of the project, and of its eventual implementation) to a third party who is not involved in the design and development and unfamiliar with the exact tasks of the semester project assignment. Think of this third party as a company that roughly wants to offer a system as described in the assignment document and commissioned your team to start this software engineering project and produce a viable project which follows best practices, is of high quality, and is worth to be funded further after the DESIGN deadline.

# **Design Draft**

## Design Approach and Overview

Describe your high-level design approach, i.e., how you designed your current solution parts, and give an overview of the system under development. During the construction of your 4+1 Views Model (see Section 3) you will need to investigate intentions, concerns, and interests of involved user roles, make appropriate assumptions to guide your design decisions, define the scope and boundaries of the system under development, and consider design options regarding the development stack. Use text and appropriate UML diagrams for documenting your **careful exploration of the given domain, intentions and design options:**

* **Assumptions** (i.e., what you assume as given). Assumptions should be used to limit scope and simplify design choices. Try to keep them consistent with stakeholder interests, and with currently existing technology.
* **Design decisions** (i.e., details about selection among multiple design alternatives). Mention all your major design decisions regarding alternatives in your 4+1 Views Model (e.g., multiple alternatives to structure the system, distribute it, or deploy it). Discuss and explain a few (at least one per team member, at the absolute minimum four) of these major design decisions in detail. For this detailed discussion where appropriate, include design descriptions and/or UML diagrams showing (a part of) the design before a re-factorization step (in case you changed a decision or added more detail or another feature) and/or alternatively considered (parts of) designs.
* **Design overview of the current state of your solution** at DESIGN milestone. Note that used design patterns should be well visible in this design overview (e.g., in the class diagrams).

Carefully check all your UML diagrams for syntactic and semantic correctness!

### Assumptions

Assumptions:

* The maintainer will only have access to the health check system, he will check if the service is alive or dead.
* There can be multiple devices in the house that can control rooms and the devices in each room.
* The user can only interact with their devices once he’s logged in.
* Each service shall have it’s own database, unrestricted by other databases.
* Each service shall have it’s own API endpoints and can only communicate with other classes via these points.
* Each action will be followed by a notification via E-Mail.

### Design Decisions

Design Decisions on the logical level:

* Decision for using an n-to-m mapping

Decision to use sequence diagram, because it allows the detailed description of communication flows.

### Design Overview

We have finished developing all the architectural views for the 4 + 1 model, and beforehand we also worked with the context map which gave us a basic idea of how the system should work and communicate with each other.

## Development Stack and Technology Stack

We decided to use C# as our coding language as it is being developed by microsoft and thus it has well written documents, a large amount of resources both in their official website and spread on the internet. As for our architecture builds, microsoft has official documentation for most stuff as well, we mainly read information about the publish/subscribe event system, DDD explained by the official docs.microsoft website, Microservice system which is thoroughly explained using an example github project about online shopping, and lastly, a few books which are offered for free, again by the official microsoft website.

.Net 6.0 was used for most simple microservice examples that were created by the team, because its the latest version of .NET and the one that holds the most up to date resources.   
  
dotnet will most likely be the nuget providing system we are going to use as it is really easy to implement external libraries in our actual system, and it also helps the developer with creating the UI part of the microservices.

Links:

<https://docs.microsoft.com/en-us/>

<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture>

<https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber>

<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice>

<https://docs.microsoft.com/en-us/microsoft-365/solutions/cloud-architecture-models?view=o365-worldwide>

<https://www.youtube.com/watch?v=r8ucofiI8vY&t=2536s>

### Development Stack

GitLab, Draw.io, Google drive

### Technology Stack

C#, Draw.io, dotnet, .NET 6.0, GitHub, Google drive to share our files and work on them at the same time, Discord for communication.

# **System Requirements**

Consider the characteristics of good requirements[[1]](#footnote-2) and decide on a set of properties (e.g., identifier, priority, description, affected use-cases and user roles, verification method) to define a requirement well and how you intend to verify it (e.g., test cases, manual inspection/review). List and discuss the identified main functional and non-functional requirements of the system and their implications on the implementation (e.g., limitations, constraints).

Regarding grading we expect approximately 3 requirements per team member.

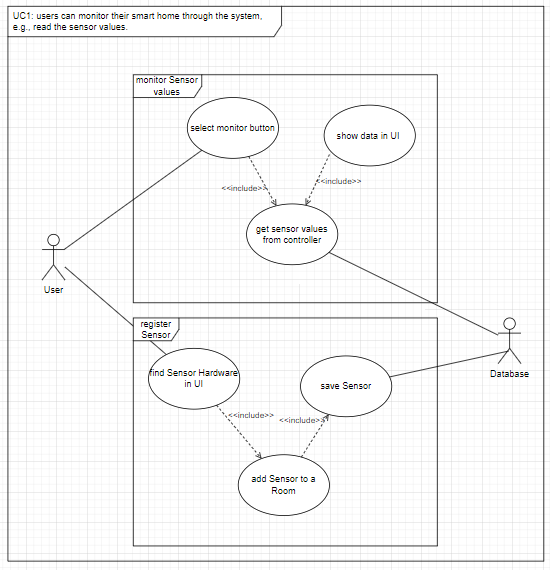
# **4+1 Views Model**

Describe all 5 views contained in the 4+1 views model in suitable detail including the appropriate UML diagrams and Use Case Descriptions.

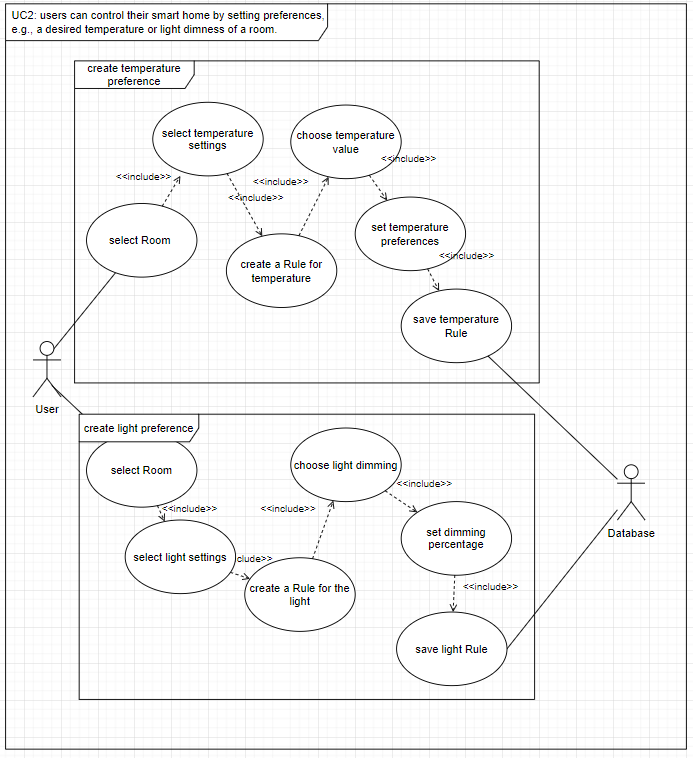
You are free to use all UML diagram types and present as many individual UML diagrams as appropriate to adequately present a view and include all relevant design aspects. Ensure that all UML diagrams are legible (the reader should not need to excessively zoom and there must not be any unreadable text/diagram elements).

## Scenarios / Use Case View

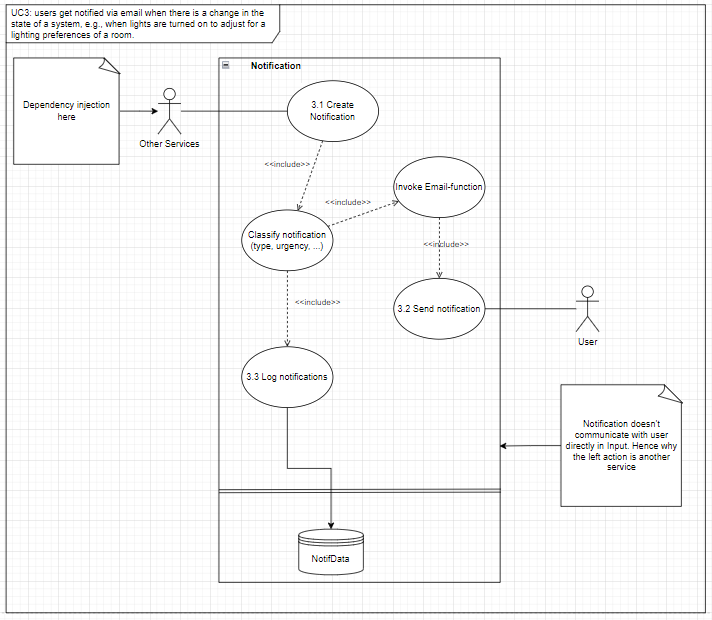
### Use Case Diagram(s)



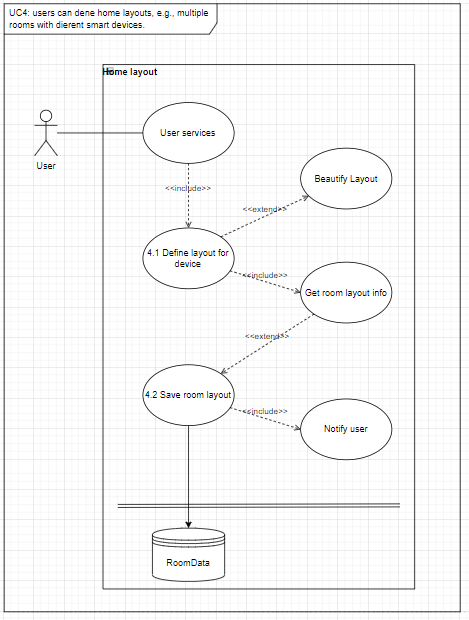
*Figure 1: UC1, users can monitor their smart home through the system*



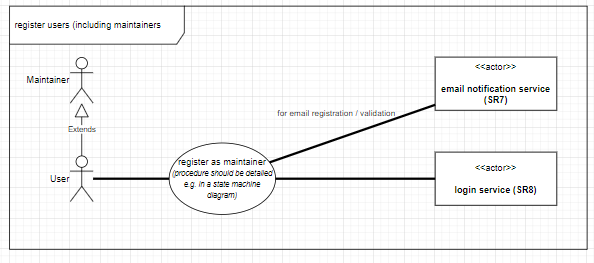
*Figure 2: UC2, user can control their smart home by setting preferences*



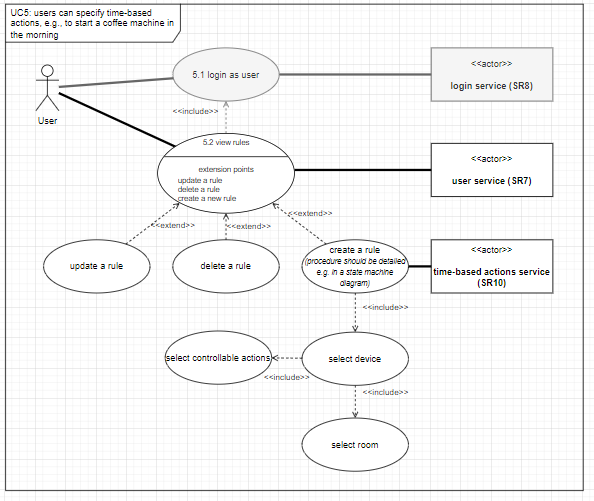
*Figure 3: UC3, user get notified via email when there is a change in the state of a system*



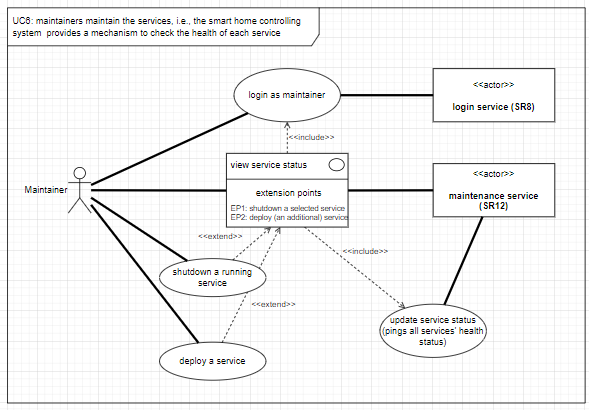
*Figure 4: UC4, user can define home layouts*



*Figure 5: use case, user can register in the System*



*Figure 6: UC5, user can specify time-based actions*



*Figure 7: UC6, maintainers maintain the services*

### Use Case Descriptions

|  |  |
| --- | --- |
| **Use Case:** | User can monitor their smart home through a system |
| **Use Case ID:** | Monitor system (UC1) |
| **Actor(s):** | User, Database |
| **Brief Description:** | User can create, save and monitor sensor data |
| **Pre-Conditions:** | User must be logged in; Sensor must be installed |
| **Post-Conditions:** | User can successfully monitor the data |
| **Main Success Scenario:** | 1. select monitor button 2. get sensor values from controller 3. show data in UI |
| **Extensions:** |  |
| **Priority:** | high |
| **Performance Target:** | The System should be able to register at least 100 sensors |
| **Issues:** | Is there any limit of sensors? |
|  |  |
| **Use Case:** | User can create Rules and Preferences for temperature |
| **Use Case ID:** | tempControl System (UC2) |
| **Actor(s):** | User, Database |
| **Brief Description:** | User can create and save rules and preferences for temperature |
| **Pre-Conditions:** | User must be logged in; Smart Home Devices must be installed; Rooms must be declared |
| **Post-Conditions:** | User can successfully create his preferences |
| **Main Success Scenario:** | 1. select room 2. select temperature settings 3. create a rule for temperature 4. choose temperature value 5. set temperature preferences 6. save temperature rule |
| **Extensions:** |  |
| **Priority:** | high |
| **Performance Target:** | The System should be able to set at least one Rule per heating element |
| **Issues:** |  |
|  |  |
| **Use Case:** | User can create Rules and Preferences for lights |
| **Use Case ID:** | lightControl System (UC2) |
| **Actor(s):** | User, Database |
| **Brief Description:** | User can create and save rules and preferences for lights |
| **Pre-Conditions:** | User must be logged in; Smart Home Devices must be installed; Rooms must be declared |
| **Post-Conditions:** | User can successfully create his preferences |
| **Main Success Scenario:** | 1. select room 2. select lights settings 3. create a rule for lights 4. choose light to dimm 5. set dimming percentage 6. save rule for light |
| **Extensions:** |  |
| **Priority:** | high |
| **Performance Target:** | The System should be able to set at least one Rule per light element |
| **Issues:** |  |

|  |  |
| --- | --- |
|  | Create Notification |
| **Use Case ID:** | Notification 3.1 |
| **Actor(s):** | Other Services |
| **Brief Description:** | Creates the notification for each user activity |
| **Pre-Conditions:** | User has to interact with the system and there needs to be an email defined |
| **Post-Conditions:** | None |
| **Main Success Scenario:** | Notification is created |
| **Extensions:** | None |
| **Priority:** | High |
| **Performance Target:** | Creation of notification |
| **Issues:** | None |

|  |  |
| --- | --- |
|  | Send email notification |
| **Use Case ID:** | Email Notification 3.2 |
| **Actor(s):** | Notification creation |
| **Brief Description:** | Sends the notification via email |
| **Pre-Conditions:** | None |
| **Post-Conditions:** | Make sure the email is sent |
| **Main Success Scenario:** | Email is sent to user |
| **Extensions:** | None |
| **Priority:** | High |
| **Performance Target:** | The email is sent to notify the user |
| **Issues:** | Make sure there’s internet connection when the email is trying to be sent. |

|  |  |
| --- | --- |
|  | Store notifications |
| **Use Case ID:** | Store notifications 3.3 |
| **Actor(s):** | Created notifications |
| **Brief Description:** | Saves the notification time, which device is taking the notification in the database. |
| **Pre-Conditions:** | Notification needs to be created successfully |
| **Post-Conditions:** | None |
| **Main Success Scenario:** | Notification is created |
| **Extensions:** | None |
| **Priority:** | High |
| **Performance Target:** | Storing the notification in the database |
| **Issues:** | Database connection is done successfully |

|  |  |
| --- | --- |
|  | Define layout for each device |
| **Use Case ID:** | Room Layout 4.1 |
| **Actor(s):** | User |
| **Brief Description:** | Defines the room layout for each device |
| **Pre-Conditions:** | Device needs to know the home layout |
| **Post-Conditions:** | Get the new room layout information |
| **Main Success Scenario:** | Room layout is changed |
| **Extensions:** | Beautify Layout |
| **Priority:** | High |
| **Performance Target:** | Allows the user to determine the room layout |
| **Issues:** | Make sure the room is defined by 1 device. |

|  |  |
| --- | --- |
|  | Save layout |
| **Use Case ID:** | Save Layout 4.2 |
| **Actor(s):** | Room layout creation |
| **Brief Description:** | Saves the room layout in the database |
| **Pre-Conditions:** | Get the information about the new room layout from 4.1 |
| **Post-Conditions:** | Make sure to notify the user once the layout is saved |
| **Main Success Scenario:** | Email is sent to user |
| **Extensions:** | None |
| **Priority:** | Low-Medium (I don’t think this should be mandatory) |
| **Performance Target:** | Create a new layout for your house |
| **Issues:** | Make sure the layout is saved in the database. |

## Logical View

The logical view must at least include UML class diagrams (for example, see Figure 2) to describe the structure of your design. Try to omit unnecessary detail (e.g., getter and setter operations), and focus on giving a good structural overview. You may also use multiple levels of details (zoom levels), or present multiple class diagrams showing parts of the overall solution that are (mostly) independent from each other. For organizational reasons and/or for documenting other important design choices in your logical view, you may also use package diagrams, object diagrams, state charts, composite structure diagrams.

Please consider:

* The UML class diagrams of your domain layer must consist of at least 10 classes and additionally 2 classes per team member.
* Every DDD building block must be used at least once. Use UML facilities like UML comments, or custom stereotypes (e.g., see Figure 3) to document the applied building blocks in your UML diagrams.
* If your domain layer does not meet these requirements, identify more functional requirements, explore the domain more deeply, and/or consider additional use-cases in agreement with your supervisor.

Diagram, engineering drawing, schematic

Description automatically generated*Figure 8: Class Diagram of our Smart Home Controlling System, the full svg file is in the master branch*

## Process View

For process decomposition use sequence (e.g., Figure 4) and communication diagrams. Optionally use, Activity (e.g., Figure 5), Timing, Interaction Overview, and/or Package diagrams.

Chart, box and whisker chart

Description automatically generated

*Figure 9: Notification Sequence Diagram*

Chart, box and whisker chart

Description automatically generated

*Figure 10: Home Layout Sequence Diagram*

Chart

Description automatically generated

*Figure 11: Register + Login Sequence Diagram*

*A picture containing graphical user interface

Description automatically generated*

*Figure 12: Time-Base-Action Sequence Diagram*

*Chart, box and whisker chart

Description automatically generated*

*Figure 13: create and monitor sensor values Sequence Diagram*

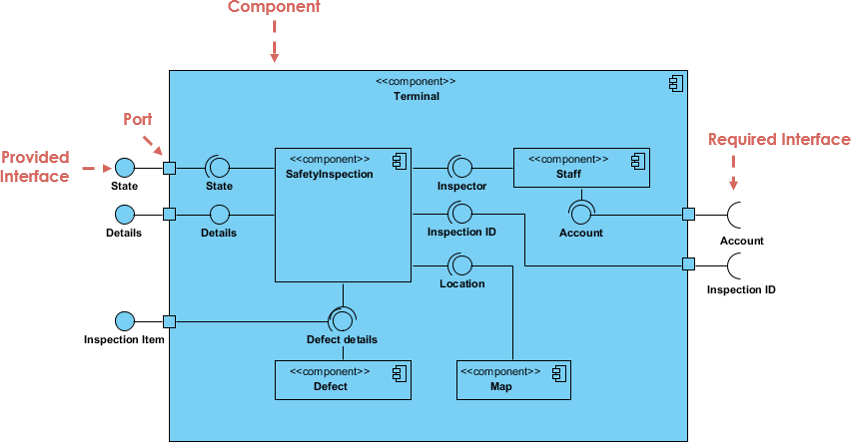
*Chart

Description automatically generated*

*Figure 14: create preferences for temperature and lights*

## Development View

For implementation and subsystem decomposition use Component diagrams (e.g., Figure 6). Optionally use Package diagrams for organizing the diagrams.

  
Figure 5: Sample Component Diagram. TODO: remove in final submission/replace by your own diagram(s)!

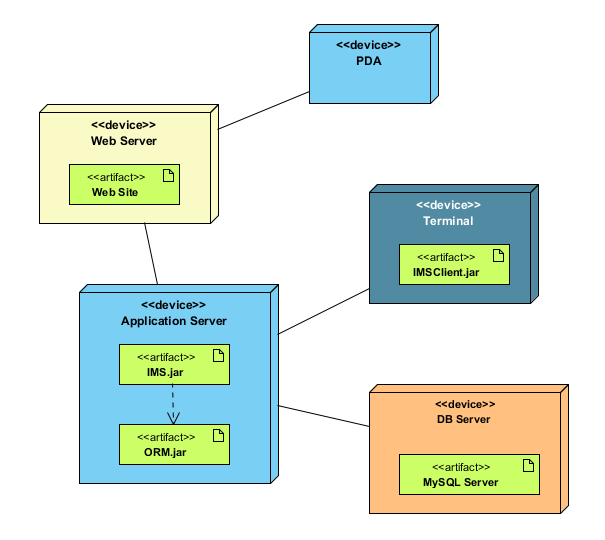
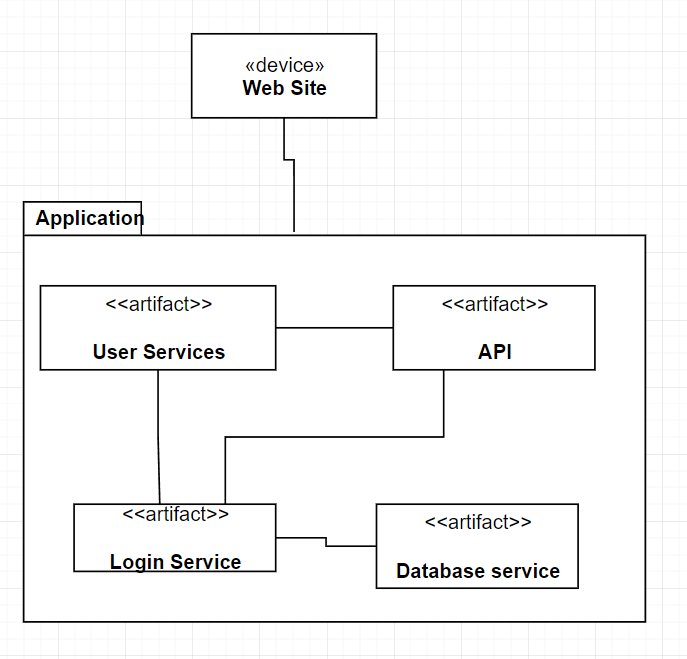
**Sample Diagram**

## Physical View

The Physical View includes the architectural design of how our system will be connected in the real world. We have decided to follow a simple approach for the longevity of our system, and how it will run.

First the application is made out of it’s microservices, each providing different services such as

* User services (monitoring system, home layout service, logging in, time based service, notification system etc)
* UI which the user will interact with (simple website, potentially built using ASPNET)
* Database system which again will most likely be SQL

  
Figure 6: Sample Deployment Diagram. TODO: remove in final submission/replace by your own diagram(s)!

*Figure 16: Deployment Diagram of the System*

# **Team Contribution and Continuous development method**

## Project Tasks and Schedule

As a comparatively compact design and development team of three people, we decided to go for a flexible project structure and team coordination approach. We organized our communication using a Discord server and utilized shared Google Drive integrated draw.io to design our models and have collaborative discussions on the design decisions. The report was written using Microsoft Word with its OneDrive share functionality for simultaneous writing and editing.

As we were tasked to approach the project using the 4+1 architectural design tools, we followed the process accordingly. First, we split the 6 main use-cases as specified in the assignment sheet into “2-2-2”, meaning each of us was tasked with the main responsibility for two use-cases. The first step here was to design the use-case diagrams that represented the vision each of us had for our use cases, and then in the logical view to imagine the required structural basis for the realization of such defined use cases. An additional tool we utilized in imagining the scenario view were sketches or “wireframes” of the user interaction to discover the required elements and components. In the second step we had to discuss the proposed ideas and had to integrate them especially on the logical level. We noticed that the very same starting point (info from the assignment sheet) had led to some different imaginations of how the processes would work. Examples for this are the conception of the home layout, or the interaction between devices and rooms. To synchronize these ideas, we held discussions on the benefits and drawbacks of the different proposals and consequently developed a common vision.

After having gained structural and conceptual clarity we then approached the remaining 4+1 views – process, development, and deployment – more efficiently and built on a common base.

Before writing this summarizing report we also attempted various options of making meeting notes. We, however, found that it was a rather time-consuming task with limited benefit, as the project management side of it was rather transparent to us due to the compact team size. We found the complexity to be rather of a technical or conceptual nature than of a project management one.



Figure 17: project schedule towards the “Design” submission

## Continuous Integration, Delivery and Deployment Plan

Explain how you continuously build, test, and deploy the iterative changes of your code, i.e., how you automate integration, delivery, deployment activities and describe – with the help of diagram(s) – your CI/CD workflow. Define the review/approval procedure among you.

## Distribution of Work and Efforts

Report how you distributed the overall work among team members and how much time was spent by each team member on the tasks.

A planned distribution of tasks among team-members, which, in case of approval by your supervisor, will be worked on individually for FINAL. Please indicate in your DESIGN deliverable which SRs will be realized by each student for FINAL and give an estimate of effort planned.

Contribution of Member 1:

Contributed on all the views for the 4 + 1 model mainly on Home layout service, Notification Service and Privilege Access Service. Developed the initial context map for an idea of how the system should function as a whole.

**Home layout Service**

Designed the home layout system as below

For the home layout design I decided to create a matrix like home access class. - Each room will have a place in the matrix of the entire house, for example

0 0 1 1 2 2 2

0 0 1 1 0 2 2

3 3 1 1 0 2 2

0 -> Represents empty spaces (walls) which aren't part of the house The rest of the numbers represent rooms in the house - There will be a second matrix which handles devices controlling rooms. For example

0 0 1 1 2 2 2

0 0 1 1 0 2 2

1 1 1 1 0 2 2

Each 0 again represents no devices - Each other number represents the device ID that is controlling room. This design allows us to create a map like interface for the user of the smart device and easy interactions with the design. Each room can have their own devices which will have their own ID and can be accessed from the device that is controlling the room.  
  
**Notification Service**

Defined the interfaces for notifications and the notification creation. A notification will be created for every single action that happens within the system and each notification will be saved in the database.   
Even if the system is restarted, the notifications will be safe inside the database and the system can progress where it left off by checking the database for any ongoing notification.

**Privilege Access System**

Concepted the log in system for the system. Users can login using their email/username and password. After logging in the user gets their privilege which is either a maintainer or a normal user. The maintainer gets access to the Health Check System, and the normal user gets access to the User Services such as Home layout service, Monitoring system etc.

Contribution of Member 2:

Contribution of Member 3:

Contributed on all the view for the 4 + 1 model, mainly on the creation, monitoring, and preferences Use Cases

# **How-to / mock-up documentation**

Privilege access system mockup

*Ein Bild, das Text enthält.

Automatisch generierte Beschreibung*

*Figure 18: Code template example of our logical structure*

The mockup was made to test the authentication system without any tokens, just a simple username/email password authentication.   
All team members worked on it, taking into consideration how this system would work as a microservice, the API Endpoints, the database implementation and how the code should have been organized.

The expected output of this mockup was the authentication of a user when the right username and password were given and error checking when the initials where incorrect.

1. <https://en.wikipedia.org/wiki/Requirement#Characteristics_of_good_requirements> [↑](#footnote-ref-2)