CSE 3105 / CSE 3137

OBJECT-ORIENTED ANALYSIS AND DESIGN

FALL 2022

COURSE PROJECT: *SafeHomeSystem*

System Design Document

Group 21

Yaren Mamuk - 190315021

Uğurcan Çırak - 190315051

Merthan Erler - 190315033

Sefa Altınok - 190315020

Hurşit İçke - 190315068

Elif Merve Arslan – 180315022

28 December 2022

Table of Contents

1 Introduction

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements. Software architecture exposes the structure of a system while hiding the implementation details. Architecture also focuses on how the elements and components within a system interact with one other. Software design delves deeper into the implementation details of the system. The aim of the project we are developing is to ensure security and ensure the fast operation of the system. The rapid operation of the system will make it easier for us to intervene in adverse situations. Thanks to the system, we can provide security from different angles. We can do multiple operations at the same time. While designing the system, we gave importance to ease of use. The working principle of our system is based on interaction with each other.

1.1 Purpose of the System

1.2 Design goals

Trade-offs	**Rationable**
Space vs **speed** (High priority)	If the system software does meet response time and throughput requirements, does not more memory can be expended to speed up the system software. If the system is not as fast as possible, low memory consumption means nothing. The program needs to be as fast as possible because it is an emergency application.
Quality vs Appearance (High priority)	If the quality is sufficient for this system, there is no need to change the appearance and spend a lot of costs.
Functionality vs **Usability** (Medium priority)	If we add a lot of functions, it makes the system more complicated to use. If it is more convenient and easier to use, the user will not be forced.
Cost vs Robustness	A low-cost system does not check for errors when the user is entering wrong data
Rapid Development vs **Functionality** (Medium priority)	If the required functions will have been implemented properly, development time takes more time than we expected.
Cost vs Realiability	The cost is high to maximize the security of the system. Details increase the cost and increase the robustness.

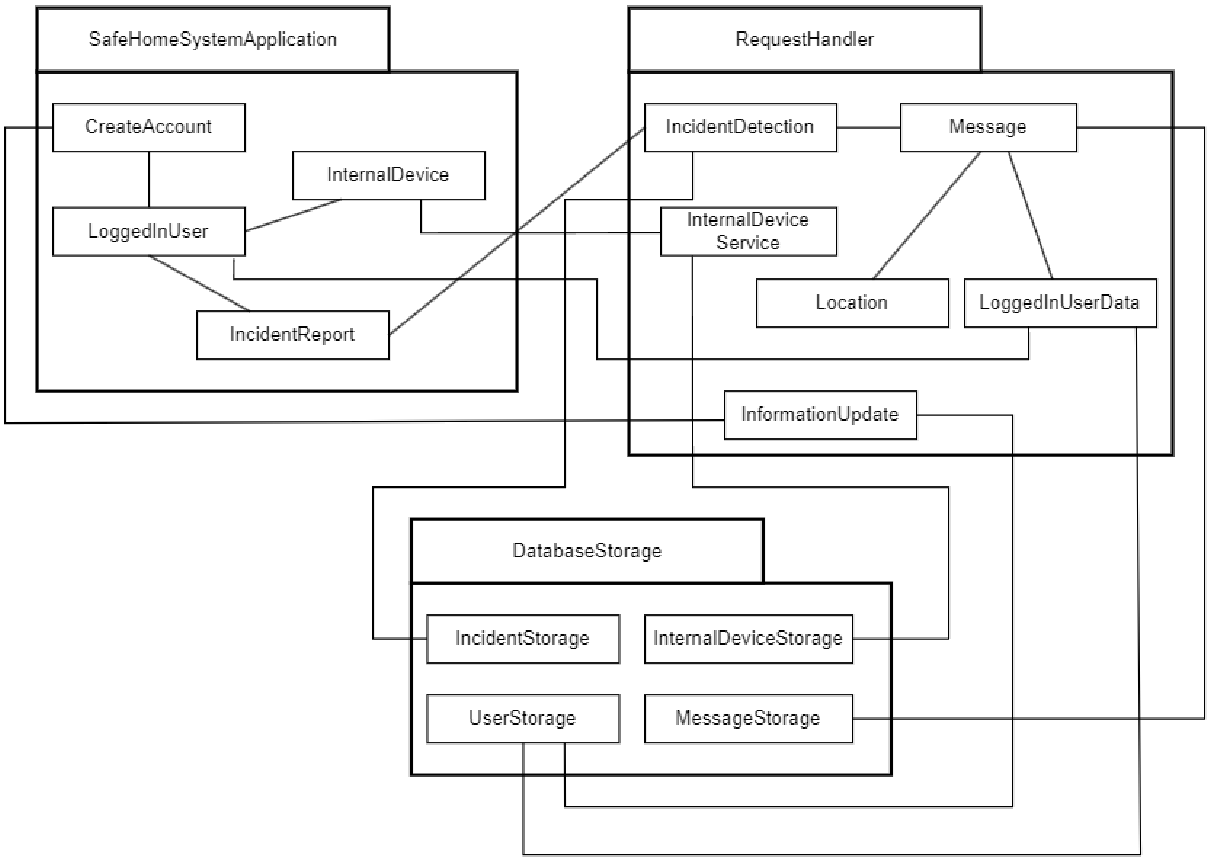
2 Current Software Architecture

The Motion Detection Security Camera Project is like the one we are developing. It is a project about security cameras in motion detection security camera project. The security camera can record video. And when the camera detects a negative situation, it can send SMS or email. We have a similar unit situation in our project. We can monitor the security camera through the system. In a negative situation, the system can send a message to the homeowner.

Utilizing UML and Patterns for Safety Critical Systems discusses methods of object-oriented analysis and design in UML suitable for the specific needs developing safety critical software systems, and to which degree safety can be related to components. The aim of our project is to keep security at the highest level. The system we build through the software used is safe thanks to sensors, locking the doors and cameras.

3 Proposed Software Architecture

3.1 Subsystem decomposition



We designed our program using the client-server architecture. Because our application is a mobile emergency application and the things that the user can do are limited. The user will make requests to the server for specific operations, and the server will return an output to the user after evaluating these requests. Also, our application needs to be as fast as possible because it is an emergency application.

We have divided the system we designed into 3 subsystems. These 3 subsystems are **SafeHomeSystemApplication**, **RequestHandler** and **DatabaseStorage**. **SafeHomeSystemApplication** is the subsystem that the user will make requests to the server, and this subsystem is running on the user's phone (client layer). The **RequestHandler** and **DatabaseStorage** subsystems are subsystems that evaluate requests from the client layer and provide appropriate answers to them.

There are 4 classes in the **SafeHomeSystemApplication** subsystem. These classes are **CreateAccount**, **LoggedInUser**, **InternalDevice** and **IncidentReport**.

- The **CreateAccount** class allows the user to register to the application.
- The **LoggedInUser** class contains the ID, password and online status of the user who has entered the application. It forms the basis of what the user can do in the application.
- The **InternalDevice** class shows the status of the specific/chosen camera, door, or sensor in the system of the user who has logged in to the application, and the user can monitor the cameras.
- The **IncidentReport** class allows the user to create a case record directly through the application.

The user performs these operations from his/her own phone. When the user starts performing operations, the user makes a ’request' and this request is sent to the **RequestHandler** subsystem.

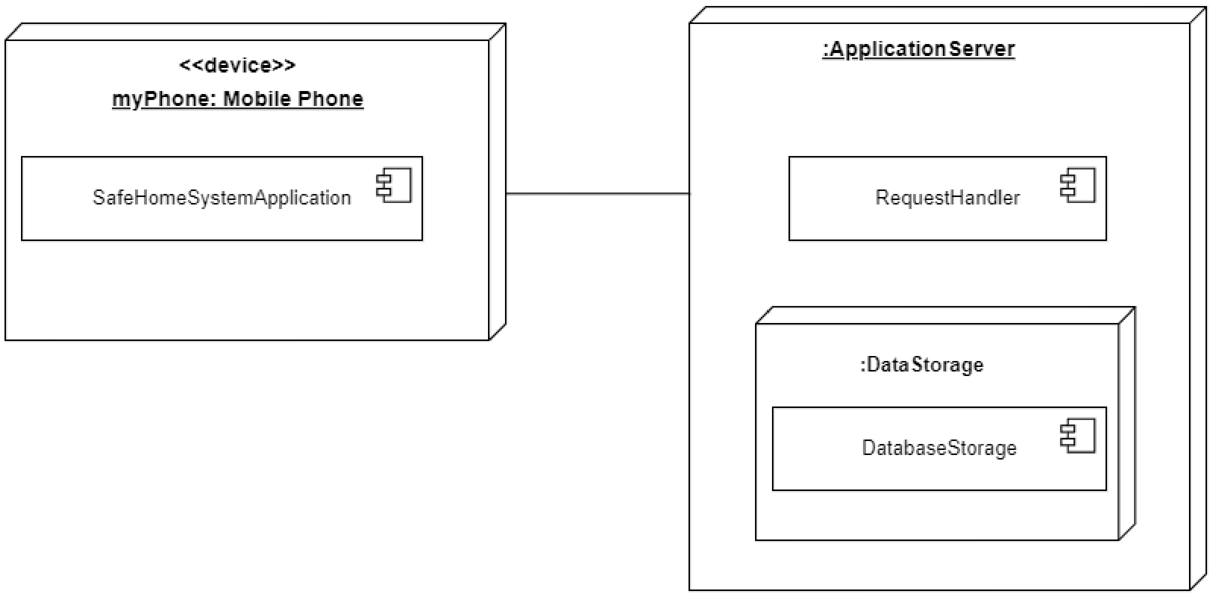
The **RequestHandler** subsystem contains the **IncidentDetection**, **Message**, **InternalDeviceService**, **InformationUpdate**, **Location**, **LoggedInUserData** classes.

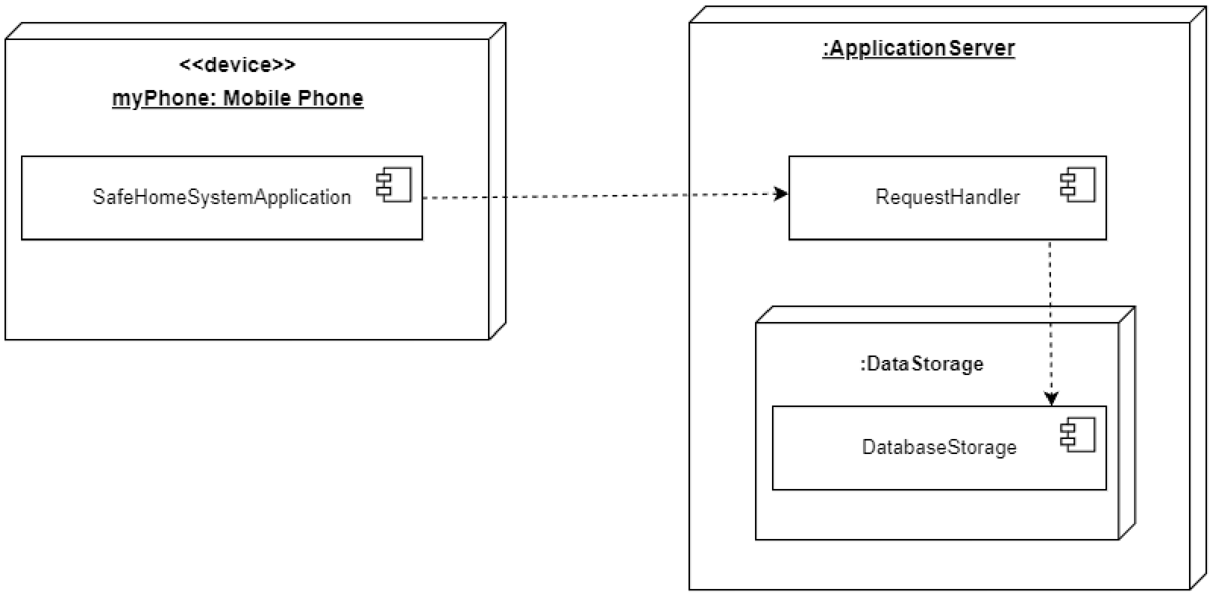
- **IncidentDetection** is responsible for evaluating the parameters from the **IncidentReport** class in the SafeHomeApplication subsystem.
- **InternalDeviceService** is responsible for fetching the desired internal device data from the **InternalDevice** class from the **DatabaseStorage** subsystem and forwarding it to the user.
- The **InformationUpdate** class transmits the account information to the **DatabaseStorage** subsystem when the user creates a new account or changes his/her information.
- The **LoggedInUserData class** is responsible for retrieving all information of the user who has logged in to the application from the database.
- A **Message** represents a **Location**, **IncidentDetection**, and its related **LoggedInUserData**. It is responsible for giving information to security officers, and users.

The **DatabaseStorage** subsystem has the **IncidentStorage**, **MessageStorage**, **InternalDeviceStorage**, and **UserStorage** classes.

- **IncidentStorage** stores cases reported by the user or detected by internal devices.
- **MessageStorage** stores **Messages** that are generated and handled in the **RequestHandler** subsystem and then sent to it.
- **InternalDeviceStorage** stores the properties of all cameras, sensors and doors that are in the system.
- **UserStorage**, stores the user's information.

3.2 Hardware/software mapping

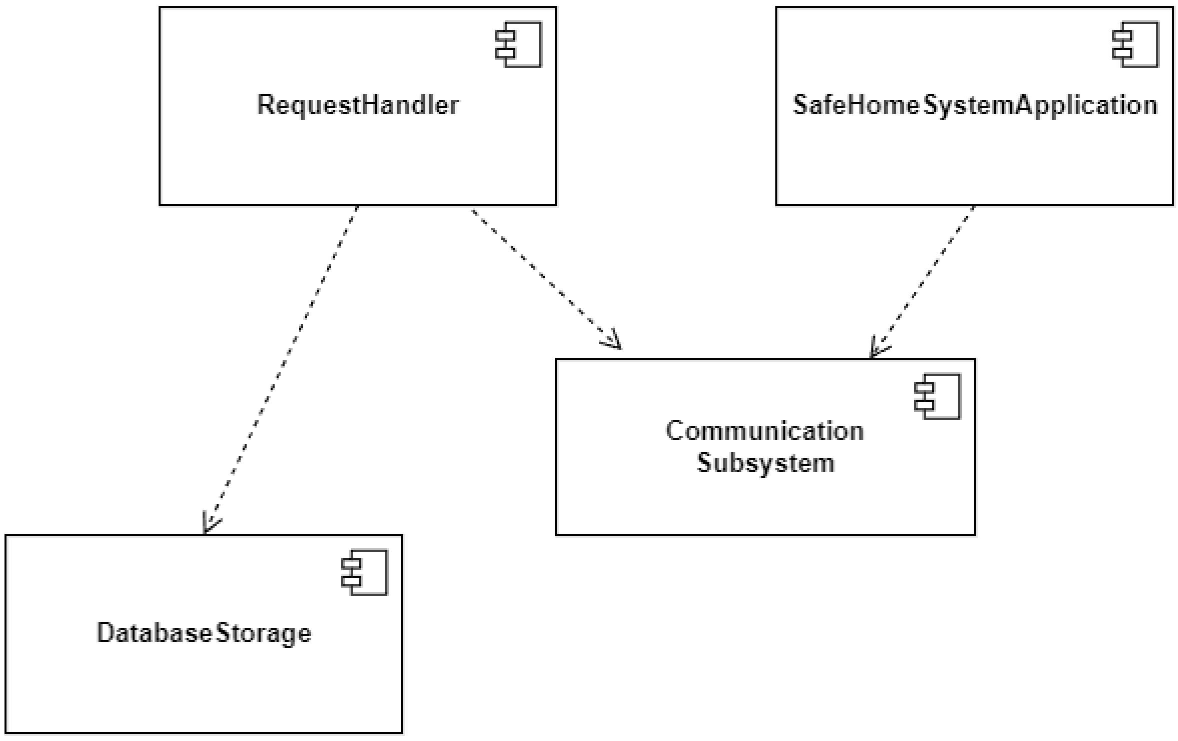




Since the SafeHomeSystemApplication subsystem performs the requests made by the user, users need to work on their mobile devices. Therefore, we created a **Mobile Phone node,** and the node contains SafeHomeSystemApplication subsystem. The **ApplicationServer node** represents the server part of our program. We put the RequestHandler subsystem in the **ApplicationServer node** because it evaluates the requests coming from the SafeHomeSystemApplication subsystem and creates the appropriate outputs. We also added the **DataStorage node** to the **ApplicationServer node** to access the database. **DataStorage node** contains DatabaseStorage subsystem. The reason for this is to get to the **DatabaseStorage** subsystem faster. However, because the **Data Storage node** is in the **ApplicationServer node**, it increases the cost of the program.

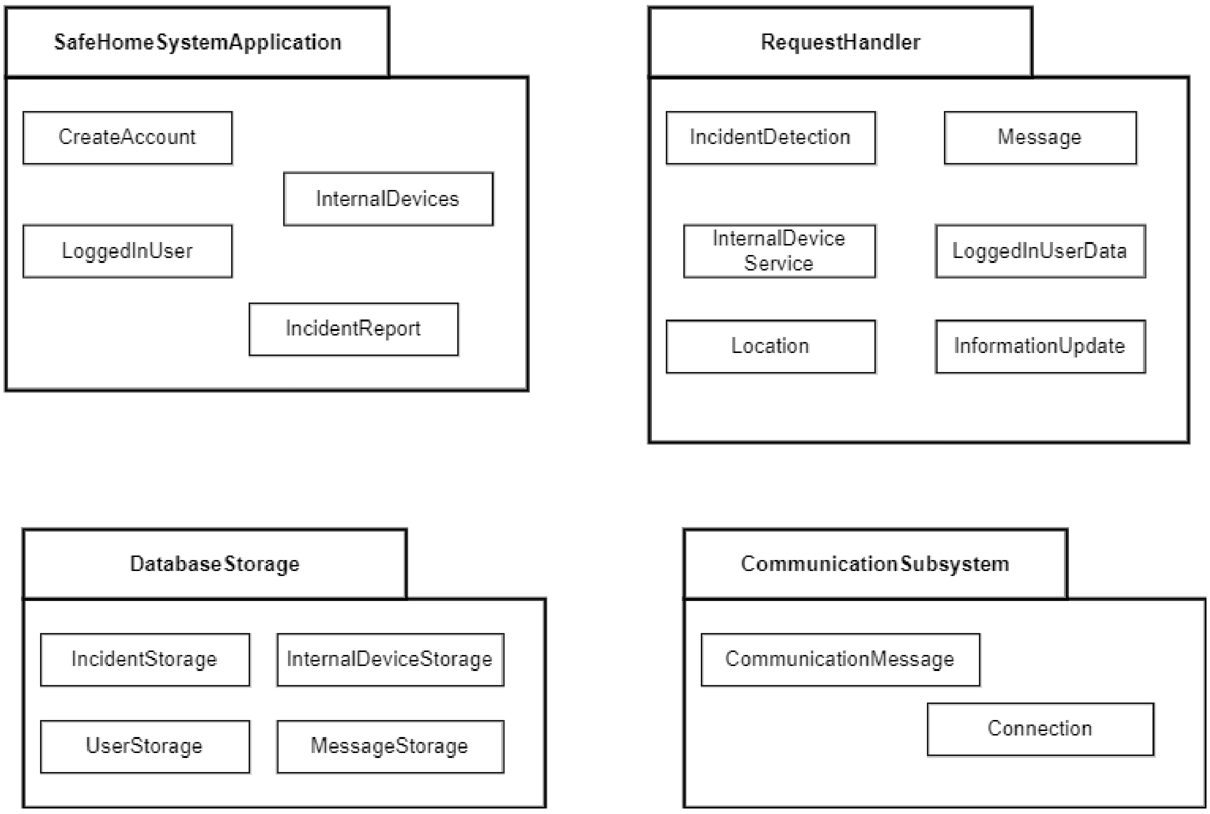
3.3 Persistent data management

We store CreateAccount information, information about all cameras, doors and sensors in the house, information about an incident report when the incident is reported by user or detected by sensors and cameras, and messages to be sent to the user and security officers in a relational database. All this information is stored in the database after being evaluated in the **RequestHandler** subsystem in accordance with the requests made by the user.

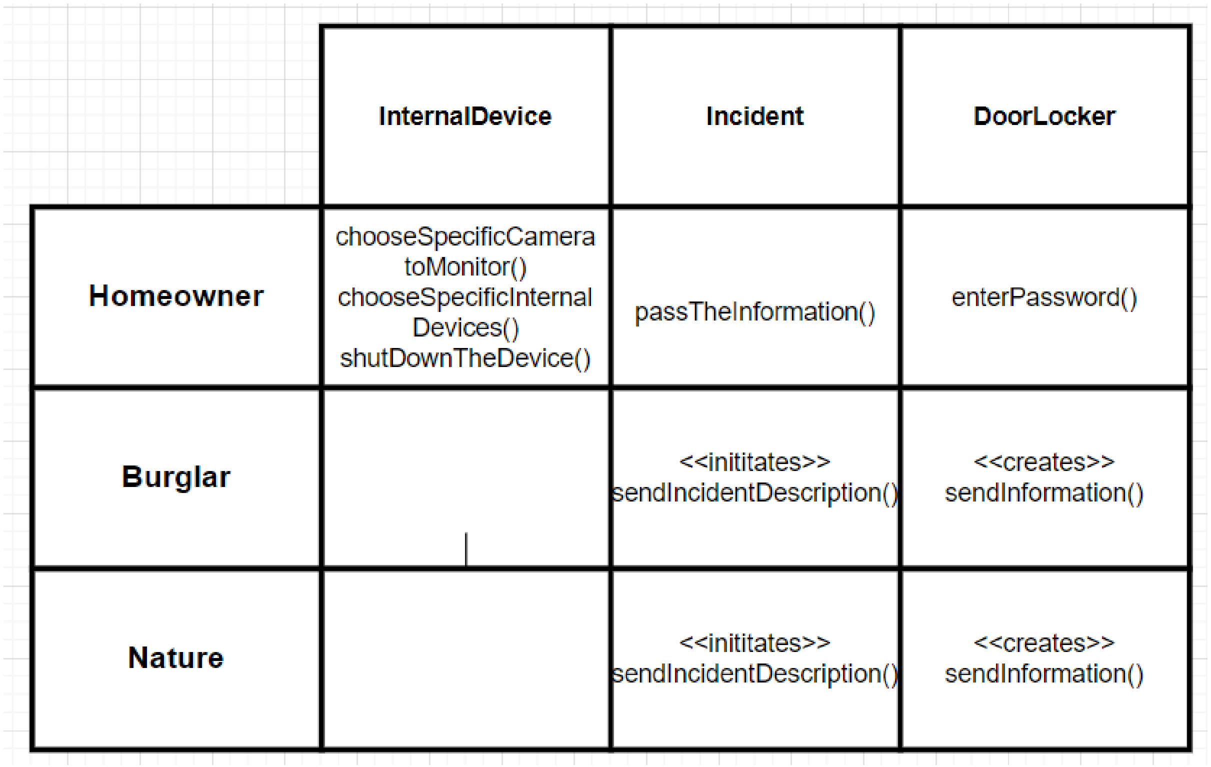


We have shown the subsystems that we use permanently in our system. In fact, our system keeps all subsystems permanently because this data can be used and analyzed in the future. Here, the **RequestHandler** subsystem evaluates the requests from the **SafeHomeApplication** subsystem and prepares appropriate outputs for them. The **DatabaseStorage** subsystem stores user-related information, records of events/incidents, information about all internal devices (sensors, cameras, and doors),and messages including exact location and description of the incident.

The **CommunicationSubsystem** is responsible for transporting objects from the **SafeHomeSystemApplication** subsystem to **RequestHandler** subsystem with the CommunicationMessage and Connection classes.



3.4 Access control and security



	InternalDevice	Incident	DoorLocker
Homeowner	chooseSpecificCamera toMonitor() chooseSpecificInternal Devices() shutDownTheDevice()	passTheInformation()	enterPassword()
Burglar		<<inititates>> sendIncidentDescription()	<<creates>> sendInformation()
Nature		<<inititates>> sendIncidentDescription()	<<creates>> sendInformation()

Homeowner uses chooseSpecificCameraToMonitor(), chooseSpecificInternalDevices(), shutDownTheDevice() operations for reach InternalDevice class. After the homeowner reach to InternalDevice class, he/she will be able to open camera operation, active/inactive the sensor.

The homeowner uses passTheInformation operation for reach to Incident class. Incident class give informes about incident type. (For example, nature, burglary, user knowledge)

When burglar enters to home, the system close down all doors inside of home. Homeowner have to login password so with enterPassword() operation to reach DoorLocked

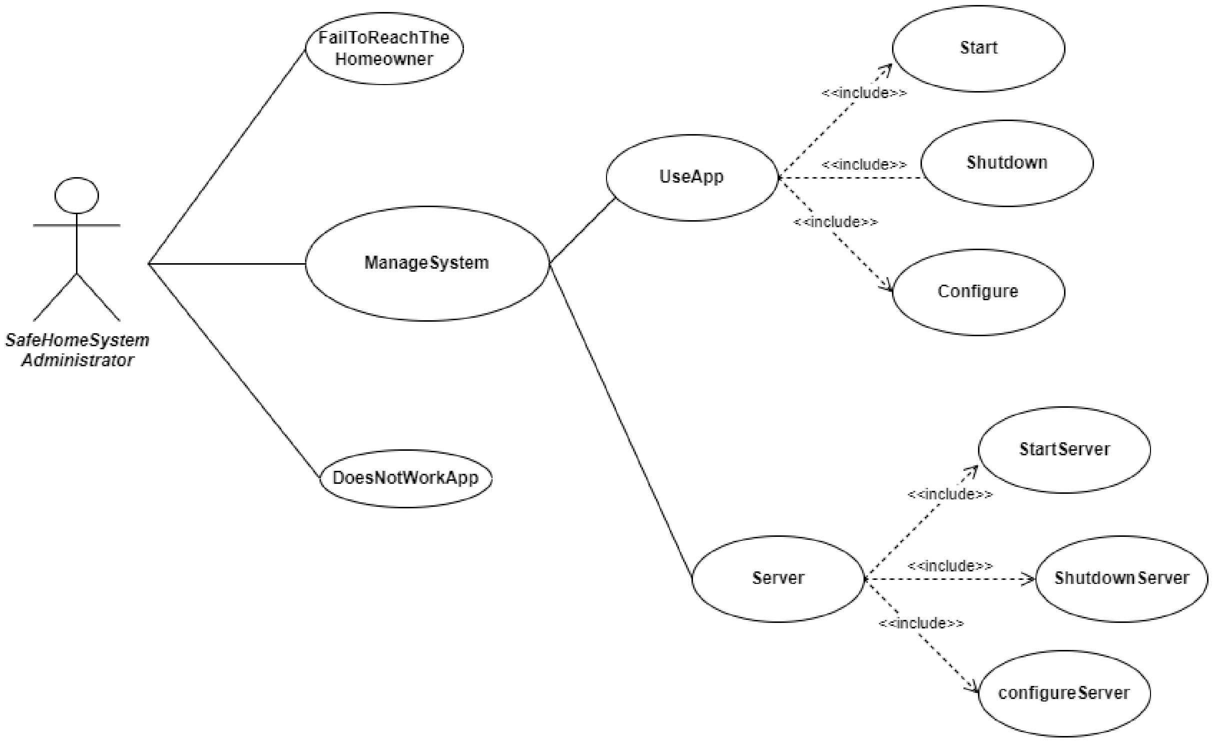
Burglar initiates Incident class, so the system get notification and send that to homeowner.

Since the burglar triggers the system, the system automatically creates a locking operation for all doors. The system receives a notification after this event and sends this notification to the homeowner.

Nature initiates Incident class. The system get notification and that send homeowner.

It causes the doors to lock as it triggers the system. That's why it provides access to the DoorLocker class. The homeowner sends a notification with the sendInformation() operation.

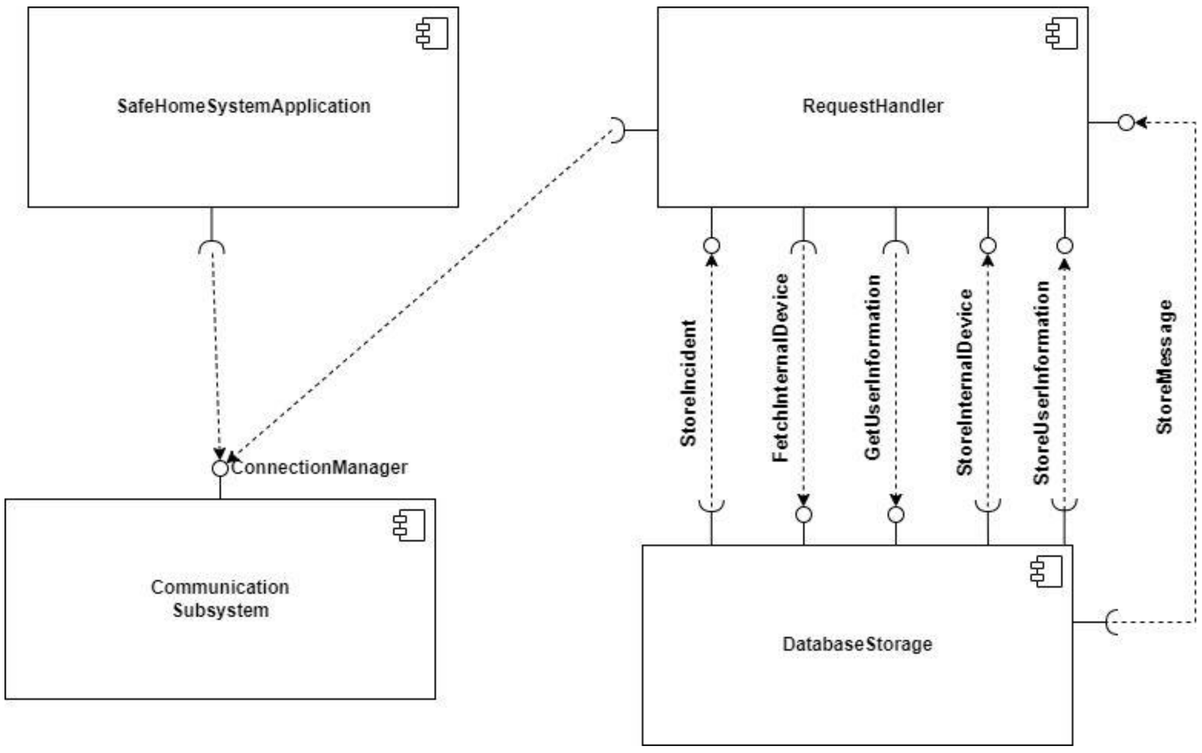
3.5 Boundary conditions



When the administrator cannot reach the homeowner, he tries to reach the homeowner by calling directly instead of password verification. If it still can't reach it, it sends a notification to the security. The security house that receives this notification goes to check. When the application is not running, the Administrator informs the user about this situation and sends a fault notification to the technical service.

The Administrator provides the users with access to the application and the server and enables them to perform operations on them. The user can start, configure, shut down the server and application over the system.

4 Subsystem Services



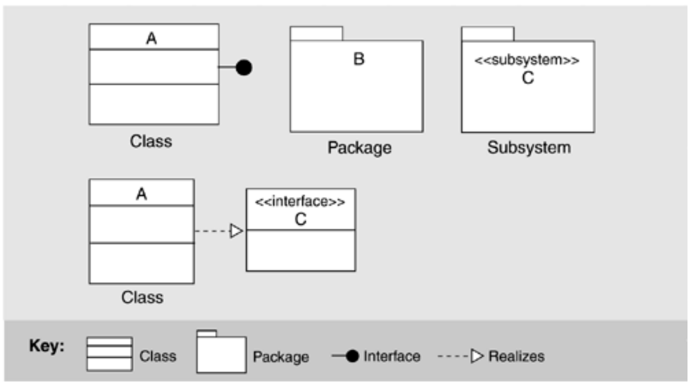
The lollipop diagram (LD) gives the service (lollipop) provided by a component to the component (socket) that requests that service. Just like bar charts, lollipop charts are used to link between different items or categories.

A UML interface describes a group of operations used or created by UML components.

•There are two types of interfaces: provided and required interfaces.

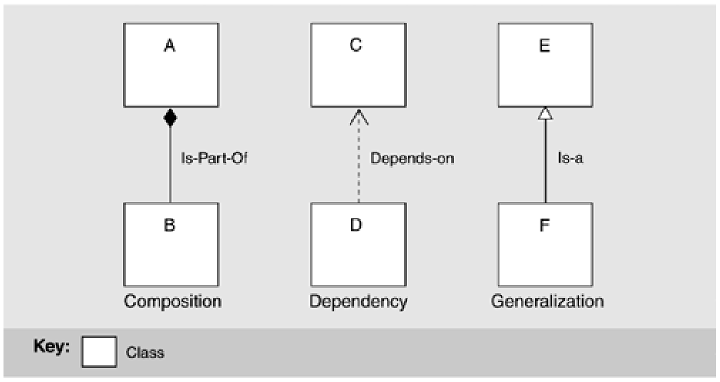
•A provided interface is modeled using the lollipop notation.

•A required interface is modeled using the socket notation.



UML provides a variety of constructs to represent different kinds of modules. UML has a class construct, which is the object-oriented specialization of a module. Packages can be used in cases where grouping of functionality is important, such as to represent layers and classes. The subsystem construct can be used if a specification of interface and behavior is required.

There are three type of relation notations in UML.



If we come to our project, **CommunicationSubsystem**; It provides **ConnectionManager** service to **SafeHomeSystemApplication** and **RequestHandler**. Also, the service depends on the **CommunicationSubsystem** due to the **SafeHomeSystemApplication** and **RequestHandler** relation.

RequestHandler provides **StoreIncident, StoreInternalDevice, StoreUserInformation, StoreMessage** services to **DatabaseStorage**. In these services **DatabaseStorage** depends on **RequestHandler. DatabaseStorage** provides **FetchInternalnalDevices, GetUserInformation** services to **RequestHandler**. And in these services, **RequestHandler** depends on **DatabaseStorage.**

5 Glossary

- The **CreateAccount** class allows the user to register to the application.
- The **LoggedInUser** class contains the ID, password and online status of the user who has entered the application. It forms the basis of what the user can do in the application.
- The **InternalDevice** class shows the status of the specific/chosen camera, door, or sensor in the system of the user who has logged in to the application, and the user can monitor the cameras.
- The **IncidentReport** class allows the user to create a case record directly through the application.
- **IncidentDetection** is responsible for evaluating the parameters from the **IncidentReport** class in the SafeHomeApplication subsystem.
- **InternalDeviceService** is responsible for fetching the desired internal device data from the **InternalDevice** class from the **DatabaseStorage** subsystem and forwarding it to the user.
- The **InformationUpdate** class transmits the account information to the **DatabaseStorage** subsystem when the user creates a new account or changes his/her information.
- The **LoggedInUserData class** is responsible for retrieving all information of the user who has logged in to the application from the database.
- A **Message** represents a **Location**, **IncidentDetection**, and its related **LoggedInUserData**. It is responsible for giving information to security officers, and users.
- **IncidentStorage** stores cases reported by the user or detected by internal devices.
- **MessageStorage** stores **Messages** that are generated and handled in the **RequestHandler** subsystem and then sent to it.
- **InternalDeviceStorage** stores the properties of all cameras, sensors and doors that are in the system.

- **UserStorage**, stores the user's information.
- **DataStorage node** contains DatabaseStorage subsystem
- **RequestHandler** subsystem evaluates the requests from the **SafeHomeApplication** subsystem and prepares appropriate outputs for them.
- The **CommunicationSubsystem** is responsible for transporting objects from the **SafeHomeSystemApplication** subsystem to **RequestHandler** subsystem with the CommunicationMessage and Connection classes

6 References

abd haidan.pdf (mu.edu.sa)

(PDF) Utilizing UML and patterns for safety critical systems (researchgate.net)

7 Appendix

- Annex – I: Distribution of Work
- Annex – II: Meeting Minutes

Annex – I

Distribution of Work

Merthan Erler, Yaren Mamuk, and Uğurcan Çırak did Current Software Architecture, Subsystem decomposition, Hardware/software mapping, Persistent data management.

Hurşit İçke and Sefa Altınok did Access control and security section.

Yaren Mamuk wrote the introduction, Current Software Architecture and Glossary parts.

Uğurcan Çırak wrote the explanations for 3.4 Access control and security and 3.5 Boundary conditions sections.

Merthan Erler did Subsystem Services.

Sefa Altınok wrote Subsystem Services

Annex – II

Meeting Minutes

<Copy this sheet as much as you need (for each meeting)>

Date:	1712/2022
Location:	Discord meeting
Duration:	3 hours
Participants:	Merthan Erler, Yaren Mamuk, Uğurcan Çırak

Completion of missing parts of the project

Current Software Architecture, Subsystem decomposition, Hardware/software mapping, Persistent data management have been completed. We made the necessary drawings from the draw.io page. Necessary explanations have been written.

Annex – II

Meeting Minutes

<Copy this sheet as much as you need (for each meeting)>

Date:	24/12/2022
Location:	Discord meeting
Duration:	4 hours
Participants:	Merthan Erler, Yaren Mamuk, Uğurcan Çırak, Sefa Altınok, Hurşit İçke

Completion of missing parts of the project

We have completed the missing parts in the homework. We corrected the wrong places in the assassignment. Everyone talked about the places they did in the homework and we completed the homework.