ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA
CESENA CAMPUS

MSc in Engineering and Computer Science

Development of a System for Monitoring the Energy Consumption of a Renewable Energy Community

Thesis in:
Concurrent and Distributed Programming
(IT: Programmazione Concorrente e Distribuita)

Supervisor
Prof. Alessandro Ricci

Candidate
Riccardo Omiccioli

Co-supervisors
Ing. Andrea Diotallevi
Ing. Marco Diotallevi

IV Graduation Session
Academic Year 2023-2024

Abstract

Economic growth caused the world's energy demand to drastically rise in the course of the last decades, and is projected to rise in the future. With this increasing demand, the introduction of new energy sources is becoming increasingly important. However, the creation and building of new infrastructure is not straightforward, and it would take decades to accomplish.

In this context, Renewable Energy Communities are a novel concept that aims at supporting the energy transition to renewable sources, serving as middle entities between the consumer and the supplier. Renewable Energy Communities allow users to share energy locally through a decentralized model and offer a method to lower energy transportation costs and wastes, while encouraging the use and creation of new renewable energy sources with state-backed incentives. Furthermore, direct connections from the main power plants to the final consumers can be inefficient and susceptible to moments of high demand. There have been previous initiatives on this topic, which primarily focused on a Renewable Energy Community's management. However, raising awareness among Renewable Energy Community's members could prove essential for optimizing the community's effectiveness.

This project seeks to improve the local communities' general experience, incorporating a tool to monitor the overall consumption of energy, as well as receiving alerts when a localized shared consumption opportunity presents. The system employs a low-cost Internet of Things device, connected to each member's electrical system to measure each member's production and consumption in order to reach this goal. The obtained data is analyzed to determine desired or unwanted behaviors, and to deliver a summary of the community's current state. Additionally, a mobile application is provided, acting as a graphical user interface for receiving system notifications and an overview of the community state.

Contents

List of Figures

List of Listings

Introduction

This thesis describes the development of a project targeted to the monitoring of a REC[1], a recent concept designed to promote sustainable collaboration in the consumption and production of renewable energy. World energy consumption has been steadily increasing over the past decades, with no signs of slowing down. Continuing to use traditional energy sources, which are mainly based on fossil fuels, will not be sustainable in the long term due to the environmental impact and the limited availability of these resources[MH12]. Since large production plants are unlikely to be replaced with better alternatives in the short term, the introduction of new renewable energy sources is becoming increasingly important. In the context of the energy transition to renewable energy sources, Renewable Energy Communities (RECs) are a useful instrument that can help ease the transition by sharing resources locally [LHv20]. The ultimate goal of this project is to develop a comprehensive system that includes energy value monitoring, community management, and active user engagement to increase system efficiency. Every consideration thereafter depends on the regulations and guidelines that are specifically tailored to Italian communities, or "Comunità Energetiche Rinnovabili (CER)", which are the project's main focus. However, the idea of REC is not limited to Italy; it is also found in other nations, with each having its own rules and regulations.

Structure of the Thesis This document is structured in chapters that reflect the main phases of the development cycle of a generic project. Although there are different methodologies for development, almost every approach used for an engineering project requires analyzing the problem, formally defining what will be realized, designing a solution that meets the needs, and finally implementing the

[1] `www.e-distribuzione.it`

solution, validating the choices made throughout the process. Following this structure, the document starts with a detailed analysis of the domain relative to the project, highlighting non-trivial concepts that emerge during the first approach with the project environment. Subsequently, the requirements that the project must meet are formally defined, using different methodologies to identify the functionalities and characteristics that the system must provide. After defining what will be done, the design phase will discuss how the system will realize the desired solution. Finally, the implementation process will be described, documenting the methodologies used to provide a high-quality solution and analyzing the results obtained, outlining possible improvements and future developments.

Chapter 1

Domain Analysis

The initial phase of domain analysis is a fundamental activity that allows to study and understand the context in which the project is inserted [Eva04]. This process aims to identify the entities, concepts, activities, and relationships that define the project environment. The ultimate goal of this phase is to gather all the necessary knowledge of the external context, familiarizing with concepts and dynamics that could initially be unknown to the people involved in the project. The domain analysis begins by considering the project idea and expanding it through a series of questions, answers, and information collected during the exploration of the project environment. Subsequently, the characteristics of the project are compared with those of existing solutions, in order to identify the innovative elements that the project seeks to introduce. Finally, all the terms and key concepts encountered during the analysis are collected, providing a shared and ubiquitous language to avoid misunderstandings in the communications that will take place during the project's life cycle.

1.1 Project Idea and Initial Analysis

The project was born with the aim of creating a system to measure and consult the consumption and production of electrical energy within a REC. Before proceeding with the analysis of the domain, it is necessary to define the concept of REC. REC are legal entities formed by members who can be private citizens, companies,

public entities, or small and medium-sized enterprises. The members of a REC voluntarily join together to virtually share the energy produced by one or more participants. The shared energy, characterized by self-consumption within a REC, is subject to incentives granted by the state to promote localized production and consumption of electrical energy and the generation of electrical energy through renewable sources.

The idea is to measure the consumption and production of each member, for example, through Internet of Thing (IoT) devices connected to the electrical system of each member. All measurements will then be stored and analyzed to provide a real-time overview of the energy situation within the community. The members of a community holds a key role in the success of a REC, as they are the ones who consume the energy produced and therefore must be actively involved in the management of the community. The final goal is to realize a tool that members can use not only to monitor consumption, but also to notify consumers of the best times for consumption.

Question and Answers Session Following the definition of the project idea, a questions and answers session was conducted to better understand the context of the project. Here are some of the most interesting aspects that emerged during the session to better understand some peculiar aspects of the REC domain.

Question: Do the members of a REC have to be in the same geographical area?

Answer: Yes, the members must be under the same primary substation. This constraint arises from the fact that the infrastructure present in the primary substations is used to locally share electrical energy, partially alleviating the load on the national electrical grid.

Question: What does it mean that the produced energy is shared virtually?

Answer: The electrical energy produced is exchanged through the primary substation. There is no direct electrical connection between the producer and the consumer.

Question: How can electrical energy be produced? Does it necessarily have to be solar energy?

Answer: No, within a community it is possible to produce energy using any renewable energy source. In fact, some communities have introduced small wind

turbines plants. However, usually most of the energy is produced using solar panels.

Question: What types of members are there within a REC?

Answer: Members can be of three types: consumers, producers, and self-consumers. Consumers and producers are users who can only consume or produce electrical energy. Self-consumers are members who have an electrical system suitable for direct self-consumption.

Question: Why is it particularly important to involve the consumers of a REC?

Answer: Incentives and lower operational costs are obtained only when members virtually exchange energy. In other words, a member who consumes electrical energy must do so while energy is being produced within the community to obtain the benefits provided.

General information on RECs RECs represent only one of the possible configurations provided for the self-consumption of electrical energy. Two other configurations that are quite common and relevant in the context of renewable energy consumption are individual remote self-consumers (IT: "autoconsumatori individuali a distanza") and groups of self-consumers (IT: "gruppi di autoconsumatori"). Unlike RECs, individual remote self-consumers are final customers who produce and directly consume their electrical energy. Groups of self-consumers, although similar to RECs, have some important differences, such as the constraint of belonging to the same building. In addition, large companies can also participate in groups of self-consumers, which is not allowed for RECs.

Of these configurations, RECs represent the most flexible solution; however, they are subject to some constraints imposed by current regulations. As a first aspect, it is specified that RECs must be formed by a minimum of two members, who can be private citizens, entities, or companies, excluding large companies. All the plants of a REC must also be connected to the same primary substation, have a power of less than 1 MW, and have been put into operation after December 16th, 2021. Each production or consumption plant must belong to one and only one REC, even if it is specified that the same individual can participate in the same REC with multiple, distinct facilities.

Incentives and benefits for RECs The main incentive for the creation of a REC is the possibility for users to obtain a series of economic benefits. Each REC receives a series of incentives for the self-consumed electrical energy, produced and consumed at the same time by the members of the community. These incentives are granted to promote the use of renewable sources and reduce the load on the national electrical grid, using the infrastructure present in the primary substation to connect a producer with the relative consumer. Each hour, the calculation of the energy produced and consumed within the community is automatically calculated and reported by the Gestore dei Servizi Energetici (GSE).

The incentives are composed of an incentive tariff and valorization fee granted by Autorità di Regolazione per Energia Reti e Ambiente (ARERA). The incentive tariff is composed of a fixed part, which depends on the size of the production plant, and a variable part that depends on the market value of electrical energy. The ARERA fee is recalculated annually, remaining fixed within each period. Moreover, for regions in central and northern Italy, additional bonuses are provided depending on the region of the community. The following table briefly reports the details of the calculation of the incentive tariff for each plant, which will be useful to reference during the calculation of the incentives.

Plant power	Incentivizing tariff
power $<$ 200 kW	80€/MWh + (0 $\rightarrow$ 40) €/MWh
200 kW $<$ power $<$ 600 kW	70€/MWh + (0 $\rightarrow$ 40) €/MWh
power $>$ 600 kW	60€/MWh + (0 $\rightarrow$ 40) €/MWh

Table 1.1: Table of incentivizing tariff values

Regional bonuses, provided to compensate for the lower solar radiation in the central and northern regions are reported in the following Table 1.2. Receiving capital contributions from the National Recovery and Resilience Plan (NRRP) for the realization of new plants will result in a proportional reduction in the incentivizing tariff. The total incentive obtained for a production plant is therefore calculated as:

$$(T_{fix} + T_{var}) * C + B + V$$

Bonus	Region
4€/MWh	Lazio, Marche, Toscana, Umbria, Abruzzo
10€/MWh	Emilia-Romagna, Friuli-Venezia Giulia, Liguria, Lombardia, Piemonte, Trentino-Alto Adige, Valle d'Aosta, Veneto

Table 1.2: Table of regional bonus values

Where T is the incentive tariff, C the reduction factor caused by the receipt of capital contributions from the NRRP, B the regional bonus, and V the ARERA valorization fee. The following calculation allows to obtain a rough estimate of the incentives that a member of a REC can obtain, that can span approximately from 0.07€/kWh up to a maximum of 0.14€/kWh.

Electricity Meter A key factor that has allowed RECs to develop in recent years is the transition towards second-generation electronic meters. This new generation of meters began to spread following the Deliberation 646/2016/R/eel of November 2016, in which the Autorità per l'Energia Elettrica il Gas e il Sistema Idrico (AEEGSI) set the regulations for the commissioning of these new meters over a span of 15 years. Compared to first-generation meters, these new meters offer a series of important advantages, including:

- detailed monitoring of energy consumed, produced, and fed into the electrical grid;
- real-time reading of the electrical power for a facility;
- bidirectional communication between the meter and electricity distributors;
- integration with IoT devices through the Chain 2 protocol for direct data reading on the meter.

These new electronic meters provide a series of fundamental advantages without which RECs would not have been able to easily spread. The new second-generation meters are divided into three categories based on the maximum power allowed for the plant in which they are installed:

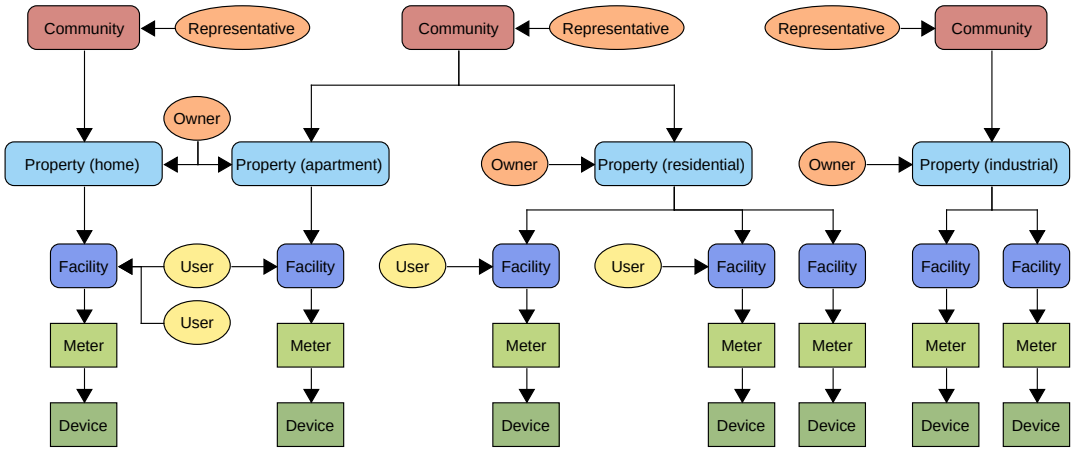


Figure 1.1: General structure of a REC

- Single-phase GEMIS, for plants with a maximum power of 3 kW. Typically used in homes, shops, and small businesses, is one of the most common meters;

- Three-phase GETIS, for plants with a maximum power of 30 kW. Used in medium-sized plants such as companies or public buildings;

- Three-phase GESIS semi-direct, which uses current transformers for semi-direct measurement. This meter is used in large plants with a power greater than 30 kW. For each plant, a constant transformation value K is defined, which is the ratio of reduction between the current used by the plant and that read by the meter. Some examples of values for the transformation constant are 125/5 and 300/5, which provide a reduction factor of 25 and 60, respectively.

General Structure The general structure of a REC can be summarized into a tree structure, as shown in Figure 1.1. This schema represents a very primitive structure of how a REC is generally organized.

1.2 State of the Art

The analysis of the state of the art is a fundamental activity that allows to understand the context in which the project is inserted. Analyzing the solutions present on the market is useful to frame the project context in order to evaluate the novelties introduced by the proposed project.

MAPS Digital Energy Community MAPS Group[1] is a company that operates in the field of Big Data applied to healthcare, energy, and environmental social and governance sectors. It offers a cloud software to simulate and forecast the economic performance of a REC, managing the onboarding of new members and the collection of adhesions.

This system also provides the possibility, thanks to third-party devices, to measure the consumption and production of electrical energy. These devices used for the measurement, are inserted into a power outlet and communicate directly with the electronic meter through the Chain2 protocol to obtain the necessary data on the energy situation of the facility.

Different software modules are available, each designed for a specific purpose, such as the preliminary simulation of a REC, the collection and management of adhesions, and the monitoring and operational management[2] of a REC. A mobile app is also available that allows access to numerous functions and, above all, to actively engage the members of the REC by providing useful suggestions to optimize the consumption habits of electrical energy. AI algorithms are also available to analyze the situation of a REC and suggest an optimal use of resources, improving the overall efficiency of the system. The software also offers additional customization possibilities[3] by providing Representational State Transfer (REST) Application Programming Interface (API) and Message Queuing Telemetry Transport (MQTT) protocols to integrate other external systems. However, no detailed information is publicly available on the system functionalities, especially for the energy data collection and analysis part that seems to rely on third-party solutions.

[1] `energy.mapsgroup.it`
[2] `energy.mapsgroup.it/energy-community-manager`
[3] `energy.mapsgroup.it/faq`

SmartDHOME SmartDHOME[4] is a company that offers a series of IoT devices designed for both energy communities and renewable energy production in general.

In addition to meters that monitor the production or consumption of energy, some Z-Wave interfaces are available for ModBus devices and systems for acquiring data from photovoltaic strings. The IoT meters available are designed to be installed in series with the electronic meter on a DIN rail and transmit the measurements detected through the Narrowband Internet of Things (NB-IoT) or Second Generation (2G) network.

The software realized mainly consists of a platform called Contact Pro CER, designed to meet various management needs of a REC. This platform is aimed primarily at the representatives of a REC and offers various functionalities, in addition to real-time monitoring. The platform allows, for example, to manage the members and the accounting of the REC and to view the environmental impact, making the administrators aware of the ecological benefits obtained. The system also implements AI algorithms to analyze and improve the efficiency of a REC.

For the members, a separate app is available that allows real-time control of consumption or production and receive personalized advice to optimize the use of electrical energy.

CloE Energy Team CloE Energy Team[5] is a company that realized an app called "Comunità Energetiche" designed to monitor and manage REC. The app is specifically designed to be used by the managers of the communities to evaluate the efficiency and operate through a series of tools that simplify the management activities of a community. Through the application, it is possible to obtain a summary of various information, such as the energy performance of a community, the economic incentives obtained, and the environmental impact generated. This application also implements functionalities to simplify the onboarding of new members and the company provides technical assistance during the configuration phase.

[4] `www.smartdhome.com`
[5] `www.cloe-energy-team.it`

MyCER This project[6] is developed by Higeco Energy, a startup belonging to the Higeco group, created to address issues in the management of a REC. Since the beginning of the project, in January 2023, more than 35 RECs have been created using this system.

The project offers a somewhat complete solution that includes IoT measurement devices and software for the management and administration of a REC. The IoT meters realized for the project use the Chain2 protocol to communicate directly with the electronic meter and obtain the energy consumption and production values.

A platform called "MyCER" is available to meet the needs of administrators and users, allowing the monitoring and management of the community. The functionalities available include monitoring of energy consumption and production, the creation of personalized rules for the redistribution of incentives obtained, and the visualization of historical data. Each member can consult the history of their energy profile and obtain an estimate of the incentives that will be conferred to them. The system also integrates with energy storage systems, electric vehicle charging stations, and the management of large energy loads. However, analyzing the publicly available information, there do not seem to be functions for direct engagement with the members of the REC.

SUN4U SUN4U[7] is a project that began in 2023 in the province of Rome, with a planned duration of 24 months. The main focus is on the organization and creation of a REC, providing useful tools to facilitate the organization and coordination between members. This project focuses on identifying other users in the same area who may be interested in forming a community, providing also a simulation and forecasting tool to estimate the possible benefits obtained from the creation of a specific REC. Despite the functionalities provided, this project does not include energy monitoring elements or direct engagement with the members of the community.

[6] `www.mycer.it`
[7] `sun4u.it`

FlexyGrid FlexyGrid[8] is a project that aims to provide a platform for the management of REC through a web platform. The project has obtained funding from the European Union Horizon 2020 program, created to promote sustainable initiatives.

The system promises to monitor the performance of a community, consumption and production, and the convenience of self-consumption considering external factors such as the cost of different hourly rates. Although IoT measurement devices and AI algorithms are mentioned, the information on these aspects is scarce, indicating that the project is probably still in development. A hybrid architecture is proposed, which includes traditional database elements with novelty blockchain technologies and IoT devices for data collection.

NRG2PEERS This project[9] was developed during research activities at the university Politecnico di Milano. Unlike the other projects considered, it focuses on supporting REC through the implementation of two tools aimed at the social aspects and preliminary evaluation of a community constitution. The project therefore proposes a complementary approach compared to the other projects, focusing on support and social features rather than monitoring and managing an established REC.

The first tool, the Readiness Level Indicator Tool, is a software that allows to evaluate the readiness level of the members to verify that the requirements for a successful community are met. The second tool, called Advisori App, is an interactive platform with social elements that offers references to useful resources and allows users to interact with each other to share success stories or clarify any doubts by sharing their knowledge.

City Green Light This project[10] was started in 2023 by City Green Light in collaboration with Sidora and realizes an open-data platform called "OpenCER". This platform aims to facilitate the monitoring and analysis of energy consumption through the use of Artificial Intelligence (AI) algorithms to optimize the

[8] `it.flexygrid.com`
[9] `www.behavioralchange4sustainability.polimi.it/catalogo/nrg2peers`
[10] `citygreenlight.com`

performance of a REC. This project also introduces gamification elements to further engage the members while promoting virtuous behaviors aimed at optimizing energy consumption. In the available information, some aspects of energy data acquisition and analysis through measurement devices and a cloud architecture are also mentioned. The platform also provides services for the design and realization of photovoltaic systems for electricity generation, as well as technical and administrative assistance for the establishment of a REC. Despite the numerous features listed, the project lacks detailed public information, and it is not specified how active user engagement is intended to be integrated, possibly indicating that the system is still under development.

Bryo Bryo is a company that has developed an app[11] to support the creation and preliminary evaluation of a REC. This app allows users to fill out a questionnaire to evaluate the suitability of joining or creating a REC. All other monitoring, management and interaction functionalities are absent, and the project realizes a solution complementary to that proposed by this project.

State of the Art summary The following tables, shown in Figure 1.2 and Figure 1.3 compare some key features of the different projects considered in the state of the art analysis, related to the REC domain. The first row of each table indicates the features that this project aims to realize, while the rows labeled with the letters from A to I refer to the above-mentioned projects.

The table in Figure 1.2 compares the characteristics of the energy measurement devices. Of the projects considered, only some provide devices capable of directly measuring the amount of energy consumed or produced. It is important to notice that the effective amount of energy produced or consumed is also measured and made available to the members of the community by GSE; however, the data is not meant to be used for real-time monitoring. This project aims to create a metering device that is easy to install and uses the Chain2 protocol to retrieve the measurements directly from the new generation of electronic meters. It is also expected that the cost of such a device will be far lower than that of the devices used in the other projects, according to the pricing information available online.

[11] `comunitaenergetica.bryo-spa.it`

	Availability	Measurements	Format	Connection	OTA Updates	Cost
	First party	Chain2	Plug	WiFi + BT 5	✔	$
A	Third party	Chain2	Plug	WiFi + BT 4.2	✔	$$$
B	First party	Direct	DIN Rail	NB-IoT + 2G	?	$$
C	✘					
D	First party	Chain2	DIN Rail	Modbus (RS485)	✘	?
E	✘					
F	✘					
G	✘					
H	✘					
I	✘					

Figure 1.2: State of the Art comparison table for devices

	Simulation	Users Management	Incentives Policies	Real-time Monitoring	Efficiency Advices	Real-time Engagement	AI Algorithms
	✘	✔	✔	✔	✔	✔	✘
A	✔	✔	✔	✔	✔	✔	✔
B	✘	✔	✔	✔	✔	✔	✔
C	✘	✔	✔	✘	✘	✘	✘
D	✘	✔	✔	✔	✘	✘	✘
E	✔	✔	✘	✘	✘	✘	✘
F	✘	✔	✔	✘	✘	✘	✔
G	✔	✘	✘	✘	✘	✘	✘
H	✔	✔	✘	✘	✔	✘	✔
I	✔	✘	✘	✘	✘	✘	✘

Figure 1.3: State of the Art comparison table for software platforms

The table shown in Figure 1.3 provides a summary of the software functionalities provided by the various projects considered. From the point of view of the platforms made available, this project seeks to realize, at least in part, some functionalities similar to those already implemented. However, this project focuses primarily on direct monitoring and interaction with the user, providing limited management functionalities for a REC, while the other projects are mainly aimed at facilitating the management of the community.

1.3 Ubiquitous Language

The domain analysis phase has allowed to identify the main concepts and entities that characterize the REC domain. The *Ubiquitous Language* constitutes the vocabulary of all the terms used within the project to ensure a clear and shared vision of all the relevant terms. The purpose is to avoid ambiguities and misunderstandings that could arise when using the same words to indicate different concepts. Conversely, different words could be used to indicate the same concept, and the ubiquitous language collects such synonyms into a single shared term. The following Table 1.3 reports the terms used within the project, indicating the term, the corresponding translation in Italian, and a brief definition of the concept.

Italian Term	English Term	Definition
Comunità energetica rinnovabile, Comunità	Community	A community that aggregates producers and consumers who share the renewable electrical energy produced by one or more members
Membro della comunità energetica, Membro	Member	A subject who participates in at least one energy community
Utente	User	A generic person who interacts with the system after authentication

Referente	Representative	A subject who plays the role of legal representative for the energy community
Proprietario	Owner	A subject to whom the electrical supply contract for one or more facilities located in a generic building
Energia elettrica, Energia	Energy	Measure of the quantity of electrical energy consumed, produced, or stored
Impianto	Facility	A generic plant for which an electrical energy supply or production contract is provided
Contatore	Meter	An electronic meter installed by the electrical supplier that manages and measures the electrical energy for a facility
Sistema	System	The overall set of devices, software, and other technologies realized in a project
Dispositivo, Misuratore	Device	An electronic device that measures the quantity of electrical energy consumed and eventually produced by a facility
Applicazione, Portale	App	Software used to enable people to interact with the system in all its functionalities
Tariffa incentivante	Incentive tariff	A tariff applied to incentivize the production of renewable energy
Contributo di valorizzazione	Valorization fee	An economic contribution recognized to the REC to incentivize them to share locally produced renewable energy without burdening the national energy grid
Contributo in conto capitale	Capital grant	Financing aimed at the realization of new plants for the production of renewable energy

Interazione, Suggerimento, Notifica	Engagement	A generic data sent by the system to a user to communicate relevant information
Autoconsumo	Self-consumption	Consumption of electrical energy that occurs at the same time as electrical energy is produced within the REC
Registrare	Sign Up	Create a user profile in the system by providing required credentials
Accedere	Sign In	Authentication procedure to access the functionalities provided by the system by providing the user credentials
Monitorare un impianto	Monitor Facility	Consult the information related to a facility
Ricevere notifiche	Receive notifications	Receive a notification through some communication channel about the status of the system
Gestire un impianto	Manage Facility	Add or remove facilities from the system
Modificare un impianto	Update Facility	Update relevant information of a facility. Add or remove other users to a facility
Gestire un dispositivo	Manage device	Add or remove a device within a facility
Gestire una comunità	Manage community	Create or remove a community
Modificare una comunità	Update community	Add or remove facilities from a community
Misurare	Measure	Detect the quantity of electrical energy consumed or produced by a facility
Inviare misurazioni	Send measurements	Send the data detected by a device to the system

Chapter 2

Requirements

After the completion of the domain analysis phase, it is necessary to identify and formalize what needs to be done in terms of functionalities and characteristics that the system must provide. This phase is known as requirements analysis and is fundamental to collect in a structured way what will be implemented and act as a guide during the subsequent design, implementation, and validation.

This project follows a Domain Driven Design (DDD) approach to requirements analysis, using different methodologies to identify requirements under different perspectives. Using DDD methodologies allows for a requirement analysis that closely align with the business domain and the needs of the stakeholders involved in the project. This analysis will start by identifying user stories and scenarios that will be further refined into use cases to provide a more detailed view of the expected behavior of the system from the point of view of the actors involved. After this initial phase, a domain storytelling session and mock-ups are used to validate the requirements and provide visual feedback on the expected functionalities. Finally, the requirements are divided into functional, non-functional, and implementation requirements and subdomains are identified to classify requirements based on their role and importance in the scope of the project.

2.1 User Stories

User stories are short descriptions of a system feature, as seen from the user's perspective. As the name suggests, these stories mock a user's interaction with the system and are useful to understand the needs of different user roles and why a specific functionality could prove valuable for that user.

- As a User, I want the ability to sign up, so that I can add a new user to the system.
- As a User, I want the ability to sign in, so that I can access the system and its features.
- As a User, I want to be able to monitor a facility, so that I can get information about energy production or consumption.
- As a User, I want to receive notifications, so that I can be informed about some relevant information of the system.
- As a Owner, I want to be able to manage my facilities, so that I can add or remove a facility from the system.
- As a Owner, I want to be able to update my facilities, so that I can update the information of the facility and add or remove other users to the facility.
- As a Owner, I want to be able to manage my devices, so that I can add or remove a device within a facility.
- As a Representative, I want to be able to manage a community, so that I can add or remove a community.
- As a Representative, I want to be able to update a community, so that I can add or remove facilities from a community.

2.2 User Scenarios

User scenarios are a more detailed view of the user stories, providing a step-by-step description of the actions that a user can perform within the system. These

scenarios are expressed in terms of **Given**, **When**, and **Then** to indicate the initial conditions, the action performed by a subject, and the expected result.

Given a user And some credentials provided by the user in the form of email and password **When** the user sign up to the system **Then** then a new unverified user is created And a confirmation code is sent to the specified user email

Given a user And the confirmation code received by the user **When** the user completes the registration by entering the code **Then** the user is confirmed

Given a user And some credentials provided by the user in the form of email and password **When** the user sign in **Then** the user is signed in

Given a user And the user is signed in And the user has access to at least one facility **When** the user access the monitoring page **Then** all production and consumption information are shown for the user's facilities

Given a user And the user is signed in And the user has access to at least one facility And the facility is in a community **When** an excess of energy production in the community is detected

Then a notification is sent to the user

Given a user
And the user is signed in
When the user adds a new facility
Then the facility is added to the user's facilities

Given a user
And the user is signed in
And the user is the owner of a facility
When the user removes a facility
Then the facility is removed from the user's facilities
And all devices associated with the facility are removed

Given a user
And the user is signed in
And the user is the owner of a facility
When the user updates the facility
Then facility information is updated

Given a user
And the user is signed in
And the user is the owner of a facility
When the user adds a new device
Then the device is added as the facility device

Given a user
And the user is signed in
And the user is the owner of a facility
And the facility has a device
When the user removes a device

Then the device is removed from the facility

Given a user
And the user is signed in
When the user creates a new community
Then the community is added to the user's communities

Given a user
And the user is signed in
And the user is the representative of a community
When the user updates the community
Then the community information is updated

Given a user
And the user is signed in
And the user is the representative of a community
When the user adds a facility to the community
Then the facility is added to the community

Given a user
And the user is signed in
And the user is the representative of a community
When the user removes a facility to the community
Then the facility is removed from the community

Given a device
And the device is added to a facility
When the device measures the energy
Then the device sends consumption information

2.3 Expected Use Cases

In the context of a software development project, use cases are a fundamental tool to define the functionalities that the system must provide viewed from the perspective of the actors involved [JC23]. The following use cases help to summarize the expected behavior of the system and provide a structured view of the functionalities that the system must provide, each with an associated actor.

Title: Sign Up
Primary Actor: User
Goal: A user signs up to the system by providing the required credentials
Scope: System
Level: User Goal
Precondition: The user is not yet signed up into the system
Minimal Guarantee: The system records the request
Success Guarantee: The user is signed up into the system
Main Success Scenario:

1. The user accesses the sign up page
2. The user provides the required credentials
3. The user confirms the sign up
4. The system sends an email with a confirmation code
5. The user completes the sign up by entering the received code
6. The user is signed up into the system

Extensions:

1. The user does not provide the required data: the system shows a specific error message
2. The user does not complete the sign up: the system sends the code again after a new request

Title: Sign In
Primary Actor: User
Goal: A user signs in to the system with the provided credentials
Scope: System

Level: User Goal
Precondition: The user is signed up into the system
Minimal Guarantee: The system records the request
Success Guarantee: The user is authenticated to the system
Main Success Scenario:

1. The user accesses the sign in page
2. The user provides the required credentials
3. The user confirms the sign in
4. The system authenticates the user

Extensions:

1. The user does not provide the required data: the system shows a specific error message

Title: Monitor Facility
Primary Actor: User
Goal: A user monitors one or more facilities to obtain information on energy production or consumption
Scope: System
Level: User Goal
Precondition: The user is signed in into the system
Success Guarantee: The user views the requested information
Main Success Scenario:

1. The user accesses the monitoring page
2. The user selects the facility to monitor
3. The system shows the requested information

Extensions:

1. The user does not have access to any facility: the system shows an error message

Title: Receive Notifications
Primary Actor: User

Goal: A user receives a notification in case of excess energy production within the community
Scope: System
Level: User Goal
Precondition: The user is signed in into the system
Success Guarantee: The user receives the notification
Main Success Scenario:

1. The system detects an excess of energy production within the community
2. The system sends a notification to the user
3. The user views the notification

Extensions: None

Title: Add Facility
Primary Actor: User
Goal: A user adds a new facility
Scope: System
Level: User Goal
Precondition: The user is signed in into the system
Success Guarantee: The facility is added correctly
Main Success Scenario:

1. The user accesses the facility management page
2. The user provides the required data
3. The user confirms the addition
4. The system adds the facility
5. The user views the added facility

Extensions:

1. The user does not provide the required data: the system shows an error message asking the user to correct the data

Title: Delete Facility
Primary Actor: Owner
Goal: An owner removes a facility

Scope: System
Level: User Goal
Precondition: The user is signed in into the system and is the owner of the facility to remove
Success Guarantee: The facility is removed correctly
Main Success Scenario:

1. The user accesses the facility management page
2. The user selects the facility to remove
3. The system asks for confirmation
4. The user confirms the removal
5. The system removes the facility
6. The system removes the device associated with the facility if present

Extensions:

1. The user does not confirm the removal: the system cancels the operation

Title: Update Facility
Primary Actor: Owner
Goal: An owner updates the information of a facility
Scope: System
Level: User Goal
Precondition: The user is signed in into the system and is the owner of the facility to update
Success Guarantee: The facility is updated correctly
Main Success Scenario:

1. The user accesses the facility management page
2. The user selects the facility to update
3. The user provides the required data
4. The user confirms the update
5. The system updates the facility

Extensions:

1. The user does not confirm the update: the system cancels the operation

Title: Create Device
Primary Actor: Owner
Goal: An owner adds a new device to a facility
Scope: System
Level: User Goal
Precondition: The user is signed in into the system and is the owner of the facility where to add the device
Success Guarantee: The device is added correctly
Main Success Scenario:

1. The user installs the device
2. The user accesses the facility management page
3. The user selects the facility where to add the device
4. The user starts the device association procedure
5. The system associates the device to the facility

Extensions:

1. The procedure fails: the system shows an error message

Title: Delete Device
Primary Actor: Owner
Goal: An owner removes a device from a facility
Scope: System
Level: User Goal
Precondition: The user is signed in into the system and is the owner of the facility where to remove the device
Success Guarantee: The device is removed correctly

Main Success Scenario:

1. The user accesses the facility management page
2. The user selects the facility where to remove the device
3. The user starts the device removal procedure
4. The system removes the device from the facility

Extensions:

1. The procedure fails: the system shows an error message and cancels the operation

Title: Create Community
Primary Actor: User
Goal: A user creates a new community
Scope: System
Level: User Goal
Precondition: The user is signed in into the system
Success Guarantee: The community is created correctly
Main Success Scenario:

1. The user accesses the community management page
2. The user provides the required data
3. The user confirms the addition
4. The system adds the community
5. The user views the added community

Extensions:

1. The user does not provide the required data: the system shows an error message asking the user to correct the data

Title: Delete Community
Primary Actor: Representative
Goal: A representative removes a community
Scope: System
Level: User Goal

Precondition: The user is signed in into the system and is the representative of the community to remove
Success Guarantee: The community is removed correctly
Main Success Scenario:

1. The user accesses the community management page
2. The user selects the community to remove
3. The system asks for confirmation
4. The user confirms the removal
5. The system removes the community

Extensions:

1. The user does not confirm the removal: the system cancels the operation
2. The community has associated facilities: the system shows a warning message

Title: Update Community
Primary Actor: Representative
Goal: A representative updates the information of a community
Scope: System
Level: User Goal
Precondition: The user is signed in into the system and is the representative of the community to update
Success Guarantee: The community is updated correctly
Main Success Scenario:

1. The user accesses the community management page
2. The user selects the community to update
3. The user provides the required data
4. The user confirms the update
5. The system updates the community

Extensions:

1. The user does not confirm the update: the system cancels the operation

Title: Measure
Primary Actor: Device
Goal: A device measures the quantity of electrical energy consumed or produced in a facility
Scope: System
Level: User Goal
Precondition: The device is installed and working
Success Guarantee: The device detects the data correctly
Main Success Scenario:

1. The device detects the electrical energy data
2. The device saves the detected data locally

Extensions:

1. The device is not able to detect the data: the device shows an error

Title: Send Measurements
Primary Actor: Device
Goal: A device sends the measures data to the system
Scope: System
Level: User Goal
Precondition: The device is installed and working
Success Guarantee: The system receives the data correctly
Main Success Scenario:

1. The device sends the measures to the system
2. The system receives the data
3. The system saves the received data

Extensions:

1. The device is not able to send the data: the device shows an error and retries later

Following the first phase of analysis, different roles that users can assume begin to emerge, each with different responsibilities. In Figure 2.1, it is possible to notice, in addition to a generic user, Owners and Representatives roles that respectively

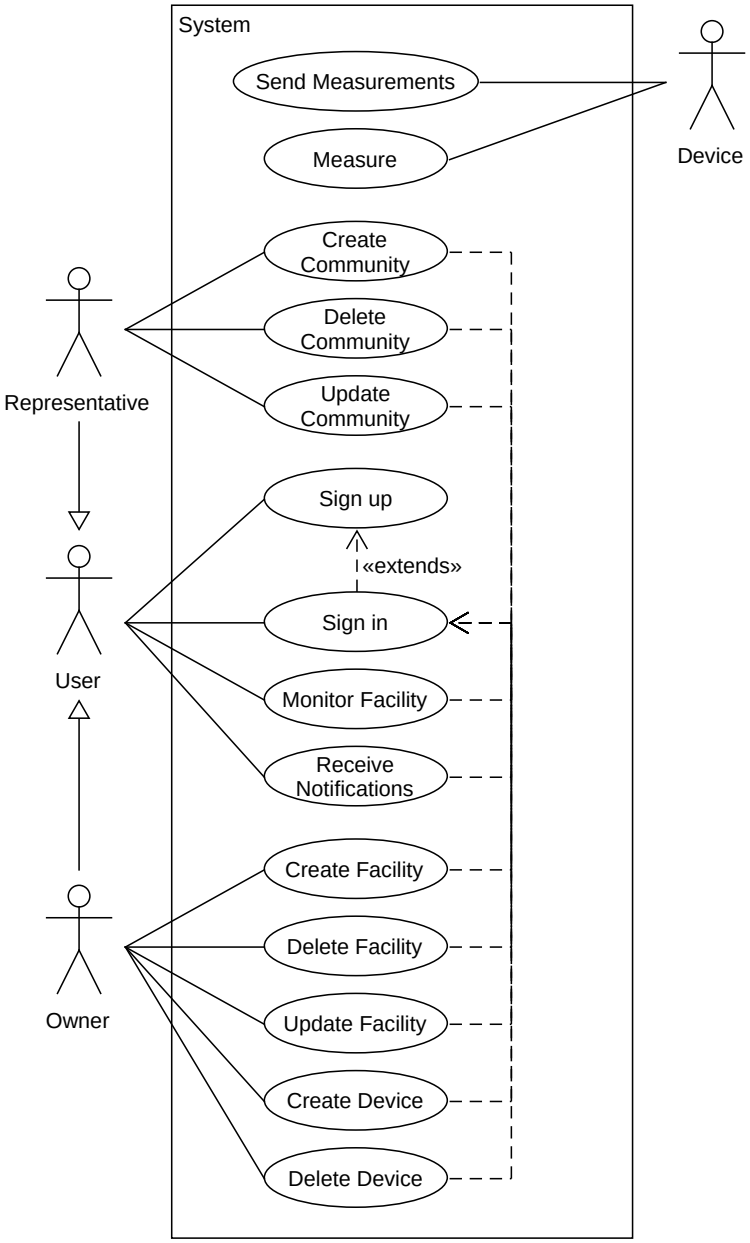


Figure 2.1: Use case diagram of the system

represent the owner of a facility and the representative of a REC. These roles are all specializations of a generic User, that is, a user who registers to the system, logs in, monitors one or more facilities, and receives some relevant notifications. All operations provided for a User require the user to sign in before accessing the available functionalities. In Figure 2.1, a device is also shown as an actor of the system which performs measurements and sends the detected data.

2.4 Domain Storytelling

The Domain Storytelling is a technique that helps to reduce misunderstandings during the analysis phase, representing some meaningful scenarios in the form of stories that illustrate graphically the interactions between the different entities of the system. This approach allows validating in an intuitive way the information collected up to the moment, making it understandable to all persons involved in the project. For this purpose, different scenarios have been developed, focusing on those considered most significant.

In the story shown in Figure 2.2, the management of a community is illustrated, that is the creation of the community itself and the addition of facilities to it. Moreover, the owners of the facilities can add other users to a facility after obtaining some information about them, in order to allow more people to monitor the energy consumption data.

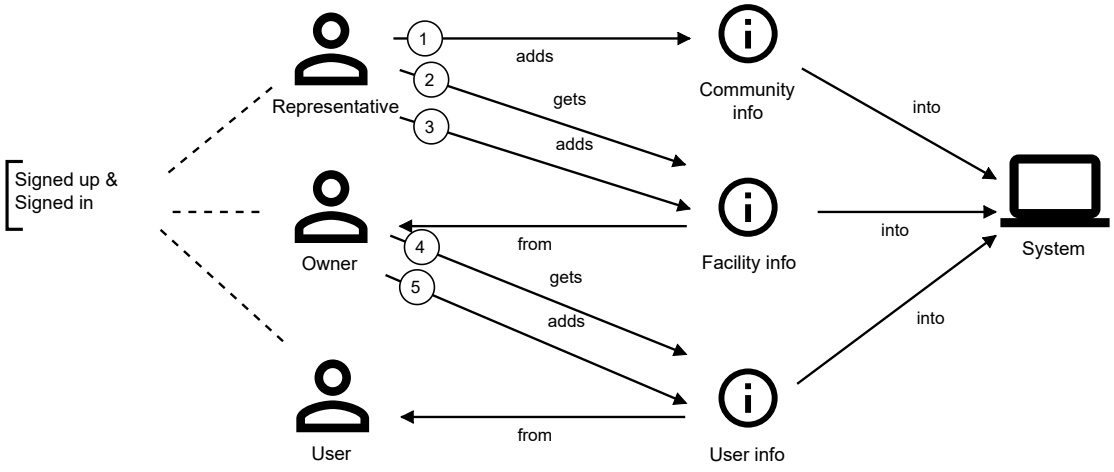


Figure 2.2: User story for community creation and update

The scenario shown in Figure 2.3 highlights the association of a measuring device to a facility. In particular, a user who is signed up and signed in the system, proceeds to add a device through an interface that allows to obtain information about the device's identity to be added to the system to complete the association.

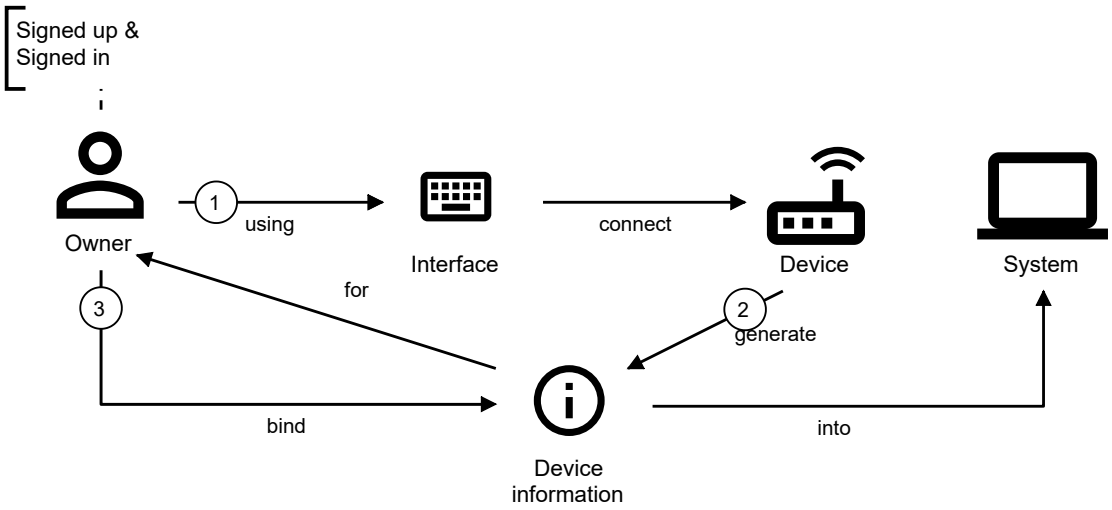


Figure 2.3: User story for device installation

The story in Figure 2.4 shows the scenario in which a device measures the energy data and sends it to the system. The device, after reading the energy data from a meter, sends the data to the system, which receives and saves it. After the data is saved, the system can provide the data to the users.

Finally, the scenario shown in Figure 2.5 illustrates the notification functionality provided to notify users when excess energy is available. After the system analyzes the data sent by the devices, it can detect an excess of energy production within a community. In this case, the system sends a notification to the users who have access to the facilities in that community.

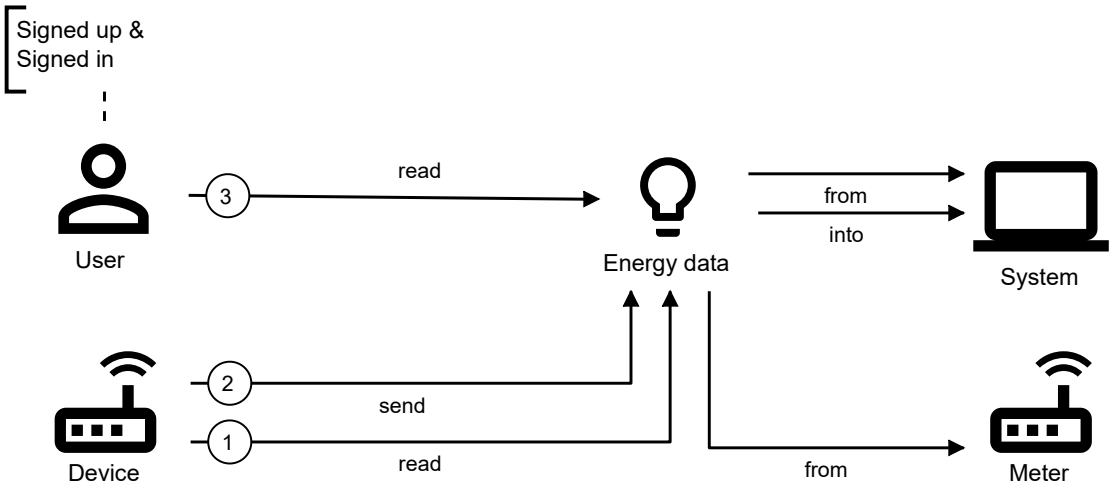


Figure 2.4: User story for energy data reading and monitoring

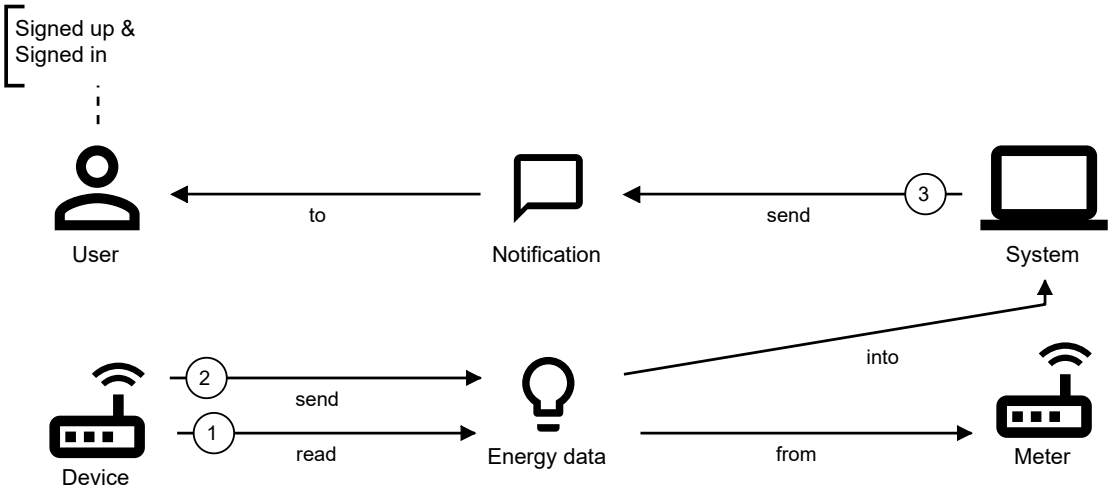


Figure 2.5: User story for user notifications when excess energy is available

2.5 Functional Requirements

Functional requirements detail the features that the system must provide to meet the needs that the project intends to achieve. Those requirements are expressed in terms of a numbered list, organizing the functionalities in a structured way, and providing a guide to verify the complete implementation of all the features.

1. user sign up
   - 1.1. sign up providing:
     - mandatory: email, password, verification code
   - 1.2. show error message if sign up procedure fails
   - 1.3. verify a new user with a validation code sent via sign-up email
2. user sign in
   - 2.1. sign in providing:
     - mandatory: email, password
   - 2.2. show error message if sign in procedure fails
   - 2.3. sign out
3. monitor facility
   - 3.1. view the facility status of user's facilities
     - power consumed
     - power produced
   - 3.2. view the REC status of user's communities
     - power consumed
     - power produced
     - excess power available
     - electric energy consumed in the last 24h
     - electric energy produced in the last 24h
4. reveice notifications

4.1. receive notifications when excess energy is available

5. manage a facility
   5.1. add a facility
   5.2. remove a facility
   5.3. show error message if facility management fails
   5.4. show error message if user is not authorized to manage the facility
6. update a facility
   6.1. add and remove users to a facility
   6.2. change facility settings
   6.3. show error message if facility update fails
   6.4. show error message if user is not authorized to update the facility
7. manage a device
   7.1. add a device to a facility
   7.2. remove a device from a facility
   7.3. show error message if device management fails
   7.4. show error message if user is not authorized to manage the device for the facility
8. manage a REC
   8.1. add a REC
   8.2. remove a REC
   8.3. show error message if REC management fails
   8.4. show error message if user is not authorized to manage the REC
9. update a REC
   9.1. add and remove facilities to a REC

9.2. show error message if REC update fails

9.3. show error message if user is not authorized to update the REC

10. measure energy data

10.1. read energy data from a device

10.2. send energy data to the system

10.3. show error if energy data measurement fails

10.4. show error if energy data sending fails

2.6 Non-Functional Requirements

Non-functional requirements detail aspects related to the quality of the system to be implemented, rather than features to be added. However, these requirements are crucial to the success of the project, as they influence the final quality of the system. These requirements specify elements related, for example, to performance, reliability, scalability, and usability of the system.

For each non-functional requirement, a quality attribute scenario is provided, which describes how to measure the system's behavior in specific situations in order to verify the compliance with the requirement. The scenario also includes the source of the stimulus, the type of stimulus, the artifact involved, the environment in which the stimulus occurs, the expected response, and how to measure the response.

1. **Performance Scenario**:

   - **Source of Stimulus**: user
   - **Stimulus**: initiate a request
   - **Artifact**: system
   - **Environment**: normal operation
   - **Response**: the request is processed
   - **Response Measure**: average response time is less than 2 seconds

2. **Compatibility Scenario**:

    - **Source of Stimulus**: user
    - **Stimulus**: installs the application on a device
    - **Artifact**: user application
    - **Environment**: normal operation
    - **Response**: the application successfully installs and operates as intended
    - **Response Measure**: the application is compatible with all devices running Android 11 or later

3. **Modifiability Scenario**:

    - **Source of Stimulus**: developer
    - **Stimulus**: changes the system code
    - **Artifact**: codebase
    - **Environment**: normal operation
    - **Response**: the update is automatically deployed
    - **Response Measure**: the system is updated and deployed in less than 10 minutes

4. **Availability Scenario**:

    - **Source of Stimulus**: user
    - **Stimulus**: use the application
    - **Artifact**: system
    - **Environment**: normal operation
    - **Response**: the system is highly available
    - **Response Measure**: the system is available at lest 99% of the time

5. **Usability Scenario**:

- **Source of Stimulus**: user
- **Stimulus**: use the application
- **Artifact**: user application
- **Environment**: normal operation
- **Response**: the application is easy to use
- **Response Measure**: all the application functionalities can be completed successfully in less than 1 minute

6. **Accessibility Scenario**:

   - **Source of Stimulus**: user with color blindness
   - **Stimulus**: use the application
   - **Artifact**: user interface
   - **Environment**: normal operation
   - **Response**: the interface is accessible
   - **Response Measure**: the interface is accessible according to WCAG AAA for text and main graphical elements

2.7 Implementation Requirements

Implementation requirements define the tools and technologies with which the system must be implemented. Defining these aspects is necessary, for example, to plan the use of some innovative technologies crucial for the success of the project, or to respect any business or pre-existing system constraints. These requirements are essential to ensure that the system is developed in a way that meets the needs of the project.

- The system must be implemented using a serverless microservices architecture to make the system as modular and scalable as possible;
- The mobile application that acts as the system dashboard must be developed using React Native so that it can be distributed on multiple platforms if a need arises;

- The measuring device must use an ESP32-based System on Chip (SoC) due to the low costs, reliability, well-documented development framework, and integrated Wi-Fi module;
- The source code and essential resources must be managed through the Git Distributed Version Control System (DVCS), using GitHub repository hosting;
- The workflow adopted during the development of the software components should be inspired by the model provided by Gitflow;
- To identify successive versions of the released software, the convention provided by Semantic Versioning 2.0.0[1] must be used.

2.8 Prototypes

The realization of prototypes in the early stages of the project allows evaluating in advance some aspects and obtain validation early on. These artifacts should be considered disposable and serve only as a guide for the subsequent development phases.

For this project, mock-ups have been created for the user interface used to interact with the system. The most important screens are shown in Figure 2.6, which includes the sign-up, monitoring, and community management pages. This tool not only allows to validate some aspects emerged during the domain analysis but also provides an intuitive way to visualize the elements expected for the system. Simulating user interactions with fictitious screens is extremely useful to identify some aspects to be deepened in the subsequent phases. Even if a general style is presented for the interface, it is more important to identify the screens and the components expected for each of them rather than focusing on the look and feel of the proposed interface.

Figure 2.7 shows a render of the device that is then realized as a prototype to show stakeholders the expected outcome for the measuring device.

[1] `semver.org`

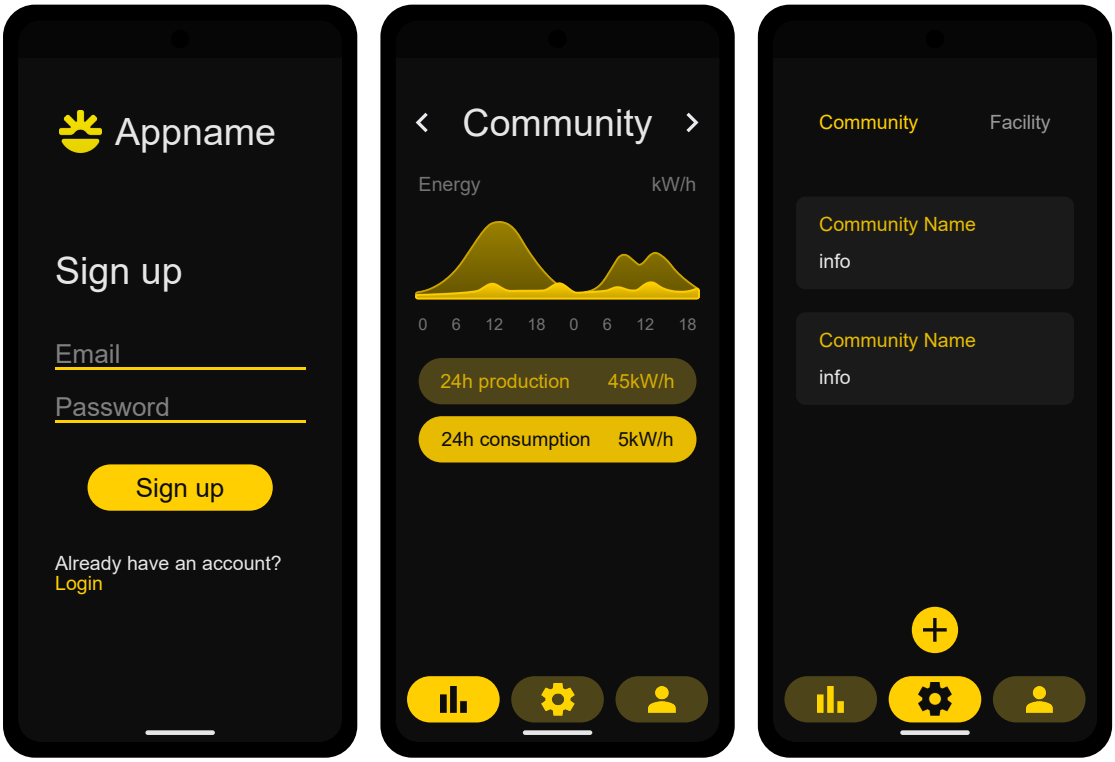


Figure 2.6: Mockups of the user interface showing the main features of the system

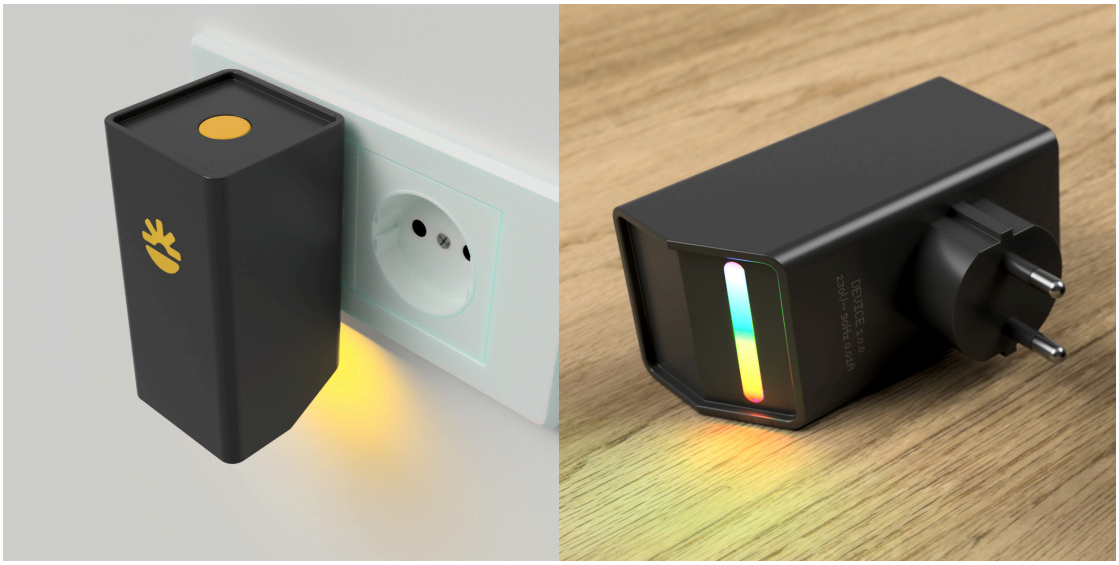


Figure 2.7: Render of the device

2.9 Subdomains

The subdomains divide and organize the parts that make up the project, making the structure of the system clearer. Each subdomain represents a component of the overall domain, with its own functionalities, purpose, and distinctive task. This division allows to address the problem in a modular way, focusing on each subdomain independently and assigning to each of them a different level of complexity.

- **Energy Data**: manages the data related to electrical energy measurements. It includes the addition and storage of data in the system and the analysis activities on them.
- **User**: deals with the management of the system users, that is their registration and authentication, and management of active sessions and security aspects.
- **Community Management**: manages all aspects related to an energy community, that is the management and updating of communities, facilities, and devices.
- **Device**: includes all the functionalities provided for a device, that is the operations of measurement and verification of the correct functioning of the devices.
- **Engagement**: analyzes the current state of the system and provides suggestions to users to improve the performance of the community.

The identified subdomains have been classified according to their complexity and the business value they bring to the project. Figure 2.8 shows how the subdomains are positioned with respect to the business value and complexity factors of the domain model. The business value refers to the value that distinguishes a particular aspect of the project from others already existing, that is, it indicates what makes the project unique and innovative compared to those already existing. The complexity of the domain model indicates the complexity of the system in terms of the functionalities provided by a specific component. In particular, according to the core domain chart of Figure 2.8, the various subdomains have been arranged in the graph according to the following considerations:

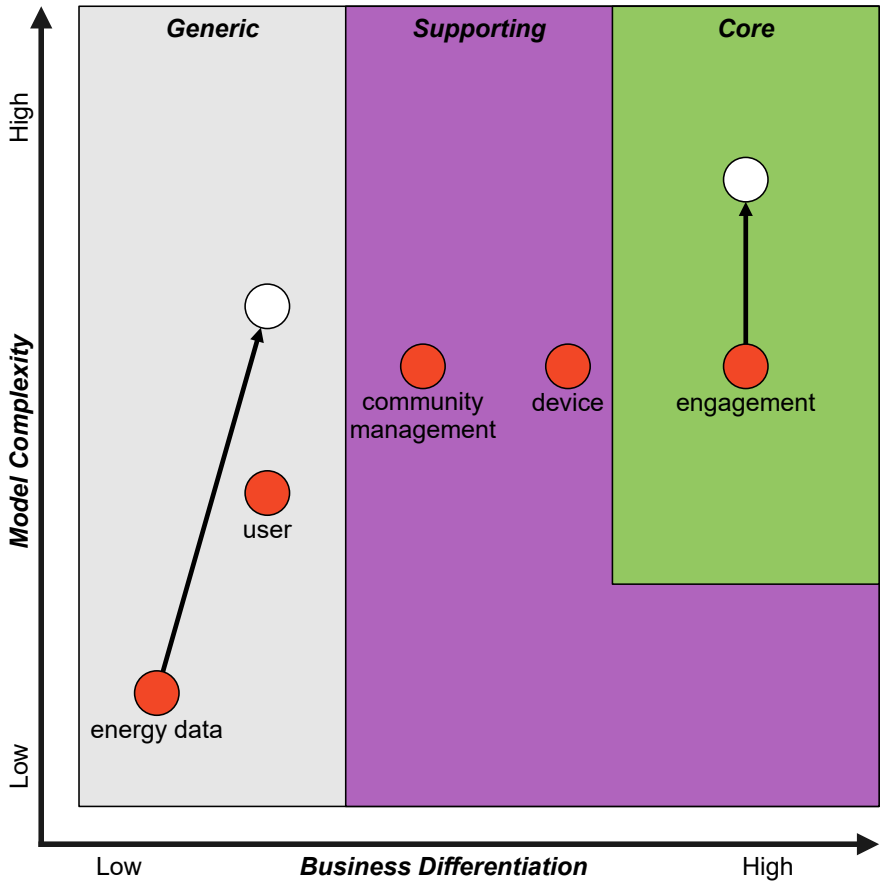


Figure 2.8: Core Domain Chart

- **Energy Data**: includes the addition of energy data to the system, as well as reading and analysis operations on them. This subdomain is characterized by a low complexity of the model and low business value, as there are no innovative aspects brought by this project. Given the simplicity of the subdomain, it is considered a generic domain. However, some future developments could significantly increase the complexity of this subdomain, while also slightly increasing its business value. For example, through AI tools, energy data could be used to make estimates on consumption and production, anticipating the overall state of the community based on observed behaviors in the past. These data could be used to further optimize consumption and receive a greater amount of incentives. However, even considering these possible future developments, this subdomain remains of a generic type, as the expected developments would directly use existing technologies. Furthermore, there are already other projects that provide these expected functionalities.

- **User**: this subdomain is classified as generic, as it uses already implemented technologies without introducing innovative elements or personalized functionalities. These functionalities include user registration and authentication and the control of the user sessions that occur during the interaction with the system.

- **Community Management**: this subdomain deals with the management of communities, facilities, and devices. Even if it does not introduce particularly innovative aspects compared to other projects, it is still considered a supporting domain, as it implements an important component for the overall functioning of the system. This component implements and manages the structure of a community as observed in the domain analysis, and the parts related to this subdomain will have to be implemented without reusing existing solutions as it will have to be customized to adapt to the project's needs.

- **Device**: includes the management of devices and all the functionalities necessary to measure, send data, and control the correct functioning of the devices. This subdomain is considered a supporting domain, as it provides

an important element for the operation of the system, without introducing particularly innovative aspects compared to other projects.

- **Engagement**: constitutes one of the fundamental aspects of the project, and for this reason, it is considered a core domain. This subdomain presents innovative business aspects compared to other projects considered and deals with the analysis of the system and the mechanisms that interact with users to make them an active component of the system. This subdomain is responsible for notifying users in real-time of certain conditions based on the current state of the system, an aspect that is often completely absent or only partially implemented. Even if the functionalities provided for this project are not trivial, some future developments could be expected to further improve the quality of the information sent to users, for example, using AI algorithms to predict future system states ahead of time.

Chapter 3

Design

The design phase of the project elaborates on the results obtained during the analysis phase, in order to define how the system will be implemented. Here the architecture of the system is defined, the components that make up the system are further detailed, and relevant decisions are made to ensure a solution that meets the requirements and constraints of the project.

While the system includes different components, such as a mobile application, and a measuring device, the focus of this project resides primarily on the service component. Each identified subdomain represents a modular and independent component that divides the overall business domain into distinct parts, each with its objectives and characteristics. In this project, each subdomain is associated with a bounded context, that is, a distinct scope in which the entities appear and specific rules of the domain are valid. Each microservice will realize one of the bounded contexts, encapsulating the domain model specific to that context. Figure 3.1 illustrates how each subdomain is mapped and realized by a different bounded context, within which a specific domain model is defined and enforced. Each domain model represents, at a high level of abstraction, the concepts, entities, and relationships relevant inside a particular bounded context.

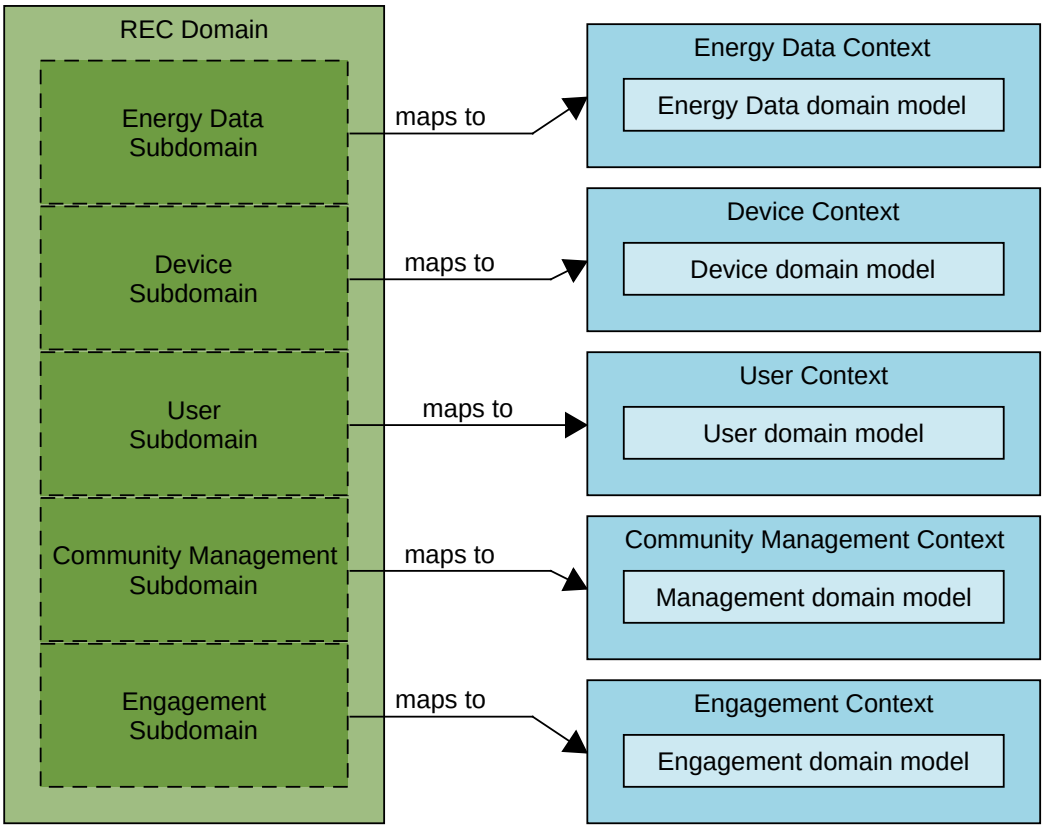


Figure 3.1: Mapping of subdomains to contexts and services

3.1 Domain Model

The domain model represents the structure of the domain, that is, the entities and the relationships between them. The starting point to create the domain model is obtained from the terms and subjects collected in the User Stories and User Scenarios during the previous analysis phase. These terms have been highlighted in the sections Section 2.1 and Section 2.2, respectively, using blue for the subjects and green for the terms and actions. The following figures show the domain models for each subdomain identified during the analysis.

The domain model for the Energy Data subdomain, shown in Figure 3.2, is minimal and only reports the monitoring by users of the energy data of the consumption and production detected previously and stored in the system.

The Device domain model, shown in Figure 3.3, illustrates how a device reads energy data from a meter and sends it to the system. This model highlights the independence of the device from the rest of the system, as it only needs to know how to send the data to the system without having to know the structure of the community.

Figure 3.4 shows the domain model for the User subdomain, which includes the sign up, confirmation, and sign in operations. The model highlights the presence of the email and password values, which are necessary for the registration and access operations.

The domain model for the Community Management subdomain is shown in Figure 3.5 includes the management of communities, facilities, and devices. The model highlights the relationships between the entities, such as the association of a device with a facility, and the presence of the Owner and Representative roles. The

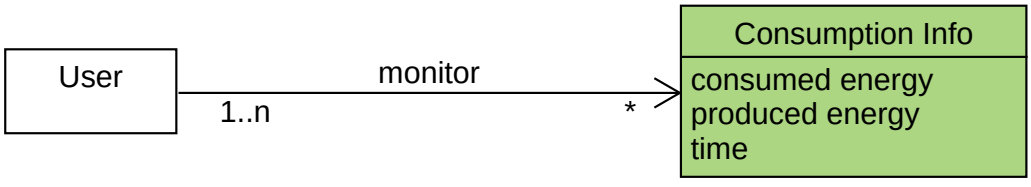


Figure 3.2: Energy Data domain model

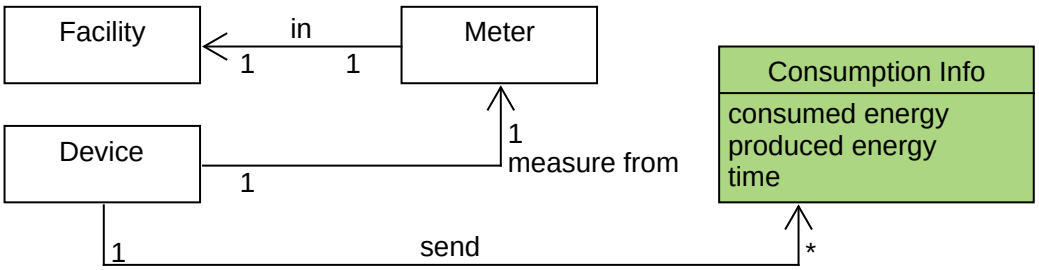


Figure 3.3: Device domain model

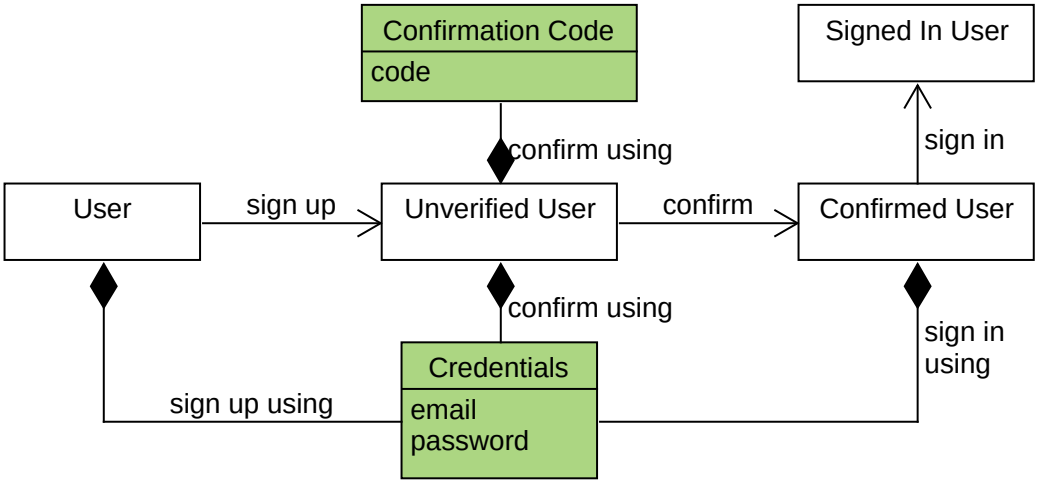


Figure 3.4: User domain model

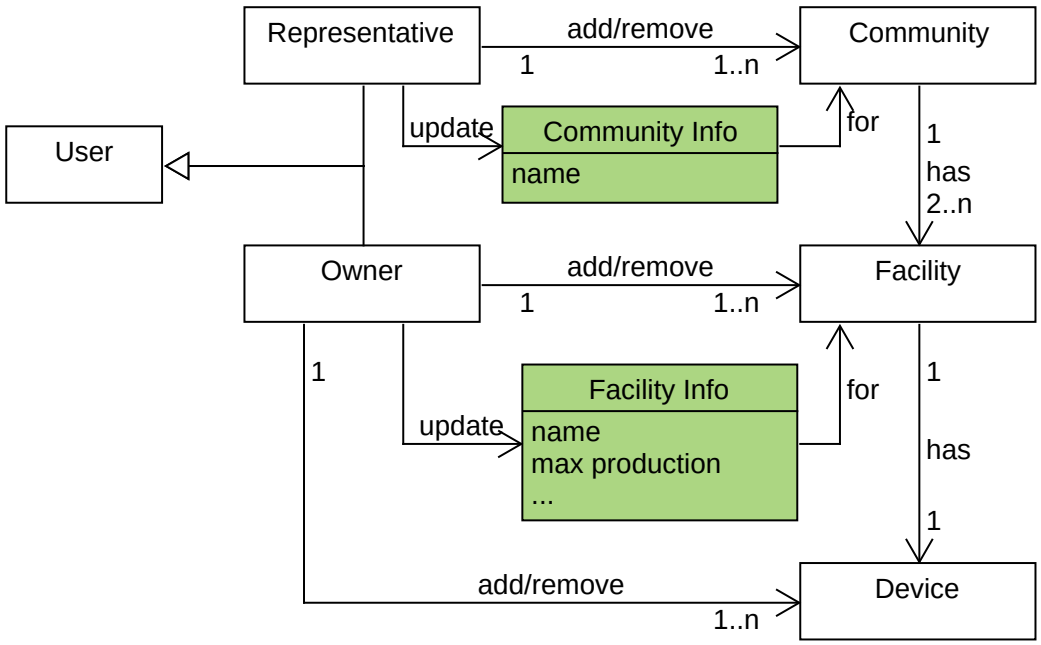


Figure 3.5: Community Management domain model

Owner is responsible for managing and updating the facilities and devices, while the Representative is responsible for managing and updating the communities.

Finally, Figure 3.6 shows the domain model for the Engagement subdomain, which includes the analysis of the system and the mechanisms that interact with users through notifications. This subdomain ensures that users are notified in real-time of certain conditions based on the current state of the system.

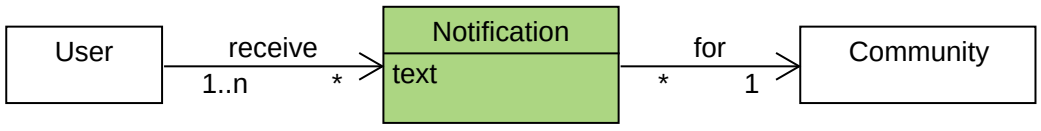


Figure 3.6: Engagement domain model

3.2 System Operations

Once the domain model is defined, the system operations are identified, which represent the interactions and relationships between the different objects of the domain. These operations describe the behavior of the system and the interactions between the entities, defining the functionalities that the system must provide. Usually, these operations are related to the requests that a user can make to the system through the user interface. The operations are defined at a high level of abstraction and are divided into two categories: commands and queries.

- **Commands**: operations that modify the state of the system, such as creating, updating, or deleting an entity.
- **Queries**: operations that return information about the state of the system, such as reading the data of an entity.

Operation	signUp(email, password)
Returns	-
Preconditions	There is no user with the same email
Postconditions	A new unconfirmed user is added to the system
Operation	confirmSignUp(email, code)
Returns	-
Preconditions	There is an unconfirmed user with the same email
Postconditions	The user is confirmed and can sign in
Operation	signIn(email, password)
Returns	-
Preconditions	The user is signed up and confirmed
Postconditions	The user is signed in
Operation	signOut()
Returns	-
Preconditions	The user is signed in
Postconditions	The user is signed out
Operation	receiveNotification()
Returns	message

Preconditions	The user monitors a facility inside the community
Postconditions	The user receives a notification
Operation	addFacility(facilityInfo)
Returns	facilityId and facility info
Preconditions	The user is signed in
Postconditions	A new facility is added to the system
Operation	removeFacility(facilityId)
Returns	-
Preconditions	The user is signed in and the user is the owner of the facility
Postconditions	The facility is removed from the system
Operation	updateFacility(facilityId, facilityInfo)
Returns	-
Preconditions	The user is signed in and the user is the owner of the facility
Postconditions	The facility is updated with the new information
Operation	addDevice(facilityId, deviceId)
Returns	-
Preconditions	The user is signed in and the user is the owner of the facility
Postconditions	A new device is added to the facility
Operation	removeDevice(deviceId)
Returns	-
Preconditions	The user is signed in and the device is in a facility that belongs to the user
Postconditions	The device is removed from the facility
Operation	addCommunity(communityInfo)
Returns	communityId
Preconditions	The user is signed in
Postconditions	A new community is added to the user
Operation	removeCommunity(communityId)
Returns	-

Preconditions	The user is signed in and the community belongs to the user
Postconditions	The community is removed from the user
Operation	updateCommunity(communityId, communityInfo)
Returns	-
Preconditions	The user is signed in and the community belongs to the user
Postconditions	The community is updated with the new information
Operation	getConsumption(facilityId)
Returns	consumptionInfo
Preconditions	The user is signed in and the facility can be monitored by the user
Postconditions	The user receives the consumption information
Operation	measure()
Returns	energyData
Preconditions	The device is connected to a meter
Postconditions	The energy data is measured
Operation	sendEnergyData(deviceId, energyData)
Returns	-
Preconditions	The device is in a facility
Postconditions	The energy data is sent to the system

3.3 Bounded Context

An additional step in the design of the system is the definition of bounded contexts, that is, the delimited contexts in which the domain model is valid. Each bounded context represents an area of the domain in which the model is consistent and coherent, and the specific ubiquitous language valid within that context is defined.

In the case of this project, each bounded context is associated with one and only one subdomain, but usually, a bounded context can be associated with multiple subdomains. The following tables show the name, a brief description, the associated subdomain, the specific ubiquitous language for that context, and, if

present, some business decisions related to the context. Regarding the ubiquitous language of each context, if there are variations or new terms specific to the context, these are reported in detail, while for terms already defined previously, reference is made to those identified in Table 1.3.

Name: Energy Data	
Description: This is the context of energy data measurements, where consumption and production values are stored and monitored by users	
Subdomain: Energy Data (generic)	
Term	**Definition**
User	An authenticated user that has access to at least a facility
Business decisions: -	

Table 3.2: Energy data bounded context

Name: User	
Description: The context where new users are added or authenticated in the system	
Subdomain: User (generic)	
Term	**Definition**
User	A generic person interacting with the system before signing up
Unverified user	A user that has signed up but is not yet confirmed
Confirmed user	A user that has completed the sign up procedure and is confirmed
Signed in user	Same definition as User in Table 1.3
Business decisions: Required credentials are email and password. Confirmation requires email and a generated confirmation code	

Table 3.3: User bounded context

Name: Community Management

Description: The context of the community management, from the community itself to facilities and bounded devices	
Subdomain: Community Management (supporting)	
Term	**Definition**
User	Same definition as User in Table 1.3
Representative	Same definition as Representative in Table 1.3
Owner	Same definition as Owner in Table 1.3
Community	Same definition as Community in Table 1.3
Facility	Same definition as Facility in Table 1.3
Device	Same definition as Device in Table 1.3
Business decisions: Only the representative for a community can change the community information. Only the owner of a facility can change information for that facility	

Table 3.4: Community Management bounded context

Name: Device	
Description: The context for device management and measurement of electric energy	
Subdomain: Device (supporting)	
Term	**Definition**
Device	Same definition as Device in Table 1.3
Meter	Same definition as Meter in Table 1.3
Facility	Same definition as Facility in Table 1.3
Business decisions: The device is not directly aware of the facility where the device is installed. Consumption information sent to the system includes the identifier of the device	

Table 3.5: Device bounded context

Name: Engagement

Description: The context for the notification of users	
Subdomain: Engagement (core)	
Term	**Definition**
User	Same as User in Table 1.3
Notification	Same as Engagement in Table 1.3
Community	Same as Community in Table 1.3
Business decisions: A notification is sent when there is an excess in energy production relative to the consumption value of the community	

Table 3.6: Engagement bounded context

The diagram in Figure 3.7 shows the different bounded contexts, each with its own domain model. In particular, the diagram highlights the relationships between the entities of the domain model in different contexts, showing when two entities in different contexts are related. However, the image in Figure 3.7 does not show the interactions between the different bounded contexts, limiting itself to show only the conceptual relationships between the entities of different contexts.

Figure 3.8 shows the map of the bounded contexts, highlighting the interactions between the different contexts. The map indicates how the different bounded contexts interact with each other, through the use of interfaces exposed between one context and another.

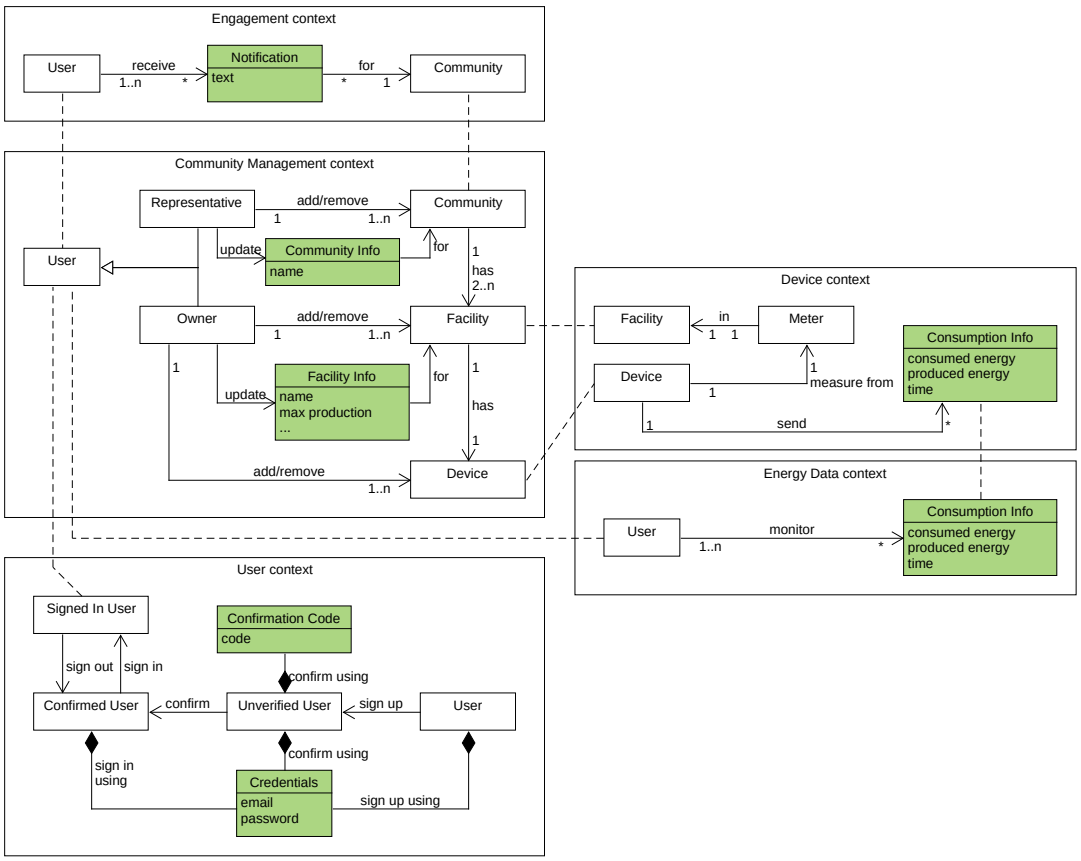


Figure 3.7: Bounded contexts relationships

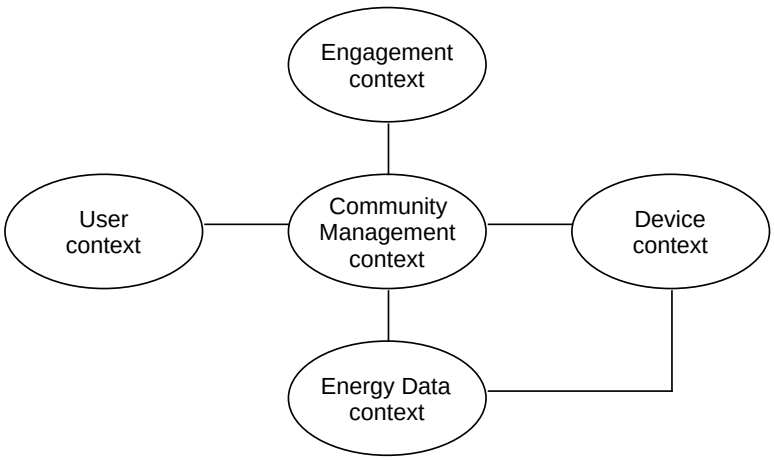


Figure 3.8: Bounded contexts map

3.4 General Architecture

What follows is a high-level overview of the system architecture, showing the various components identified and explaining the decisions made during the design of their architecture. As already mentioned in the implementation requirements, the chosen architecture for the system is a microservices architecture, in particular using a serverless approach. This choice was made to ensure maximum scalability and modularity of the system, as well as to reduce the management costs of the system[VGO+17]. In particular, by using services oriented to a serverless architecture, the management of resources is completely delegated to the service provider, avoiding having to manually manage the allocation of resources.

Figure 3.9 shows the view of the components and the system, providing a high-level view of the software components considered during the execution of the system. This view also shows the two "clients" of the system, the mobile application and the measuring device, which interact respectively via Hypertext Transfer Protocol (HTTP) and MQTT protocols with the system. Each microservice is designed as a different component, which is responsible for a specific area of functionality, following the structure proposed by the different subdomains identified. Each component can be composed of different subcomponents, depending on the needs of the specific microservice, while the interactions between the different components happens through the use of exposed interfaces. For example, the Energy Data microservice has a database component that stores and provides basic operations on the data, and an application logic component that manages the data at a higher level to provide the expected functionalities. As already stated, each component interacts with the others through ports that expose the functionalities of the component and the expected interactions are shown in Figure 3.9.

The designed architecture is illustrated in Figure 3.10, reporting the different microservices identified. The diagram reports the microservices for the management of users, the sending of measurements by a device, the management and analysis of energy data, the management of an energy community, and the notification of users. Each microservice will have its own exposed interface, which will be used for interactions with other services. Furthermore, some microservices will manage the data of their competence within a specific database in order to

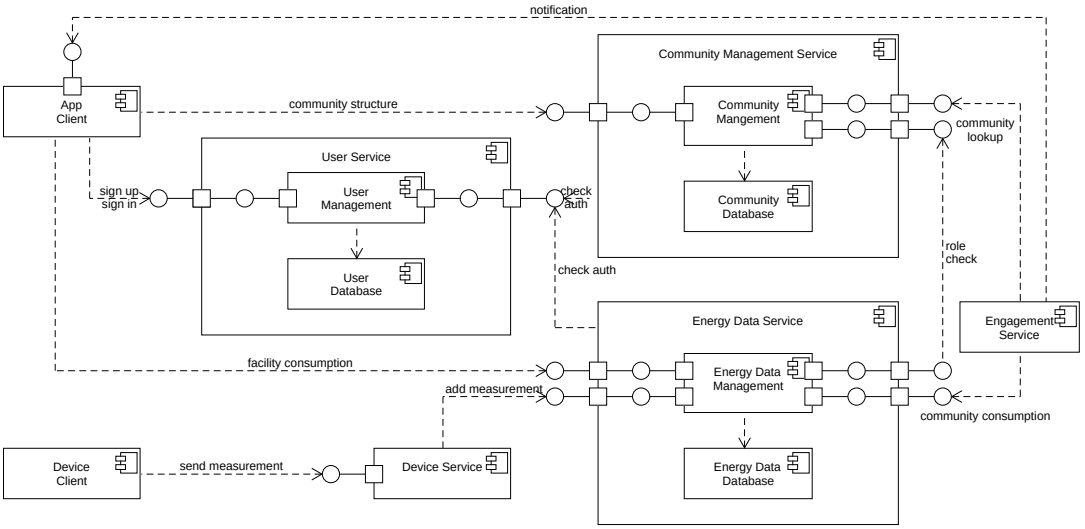


Figure 3.9: Component diagram of the system

guarantee the separation of data based on the functionality implemented.

Having defined the general architecture of the system, it is also possible to design observability and monitoring patterns to ensure the correct functioning of each microservice. The following Table 3.7 highlights the patterns expected for each microservice, which can differ depending on the specific needs and importance of the microservice.

Microservice	**Patterns**
User	-
Device	Logging, Tracing, Metrics
Energy Data	Logging, Distributed tracing, Exception tracking, Application metrics
Community Management	Logging, Distributed tracing, Exception tracking, Application metrics
Engagement	Logging, Tracing, Metrics

Table 3.7: Observability and monitoring patterns

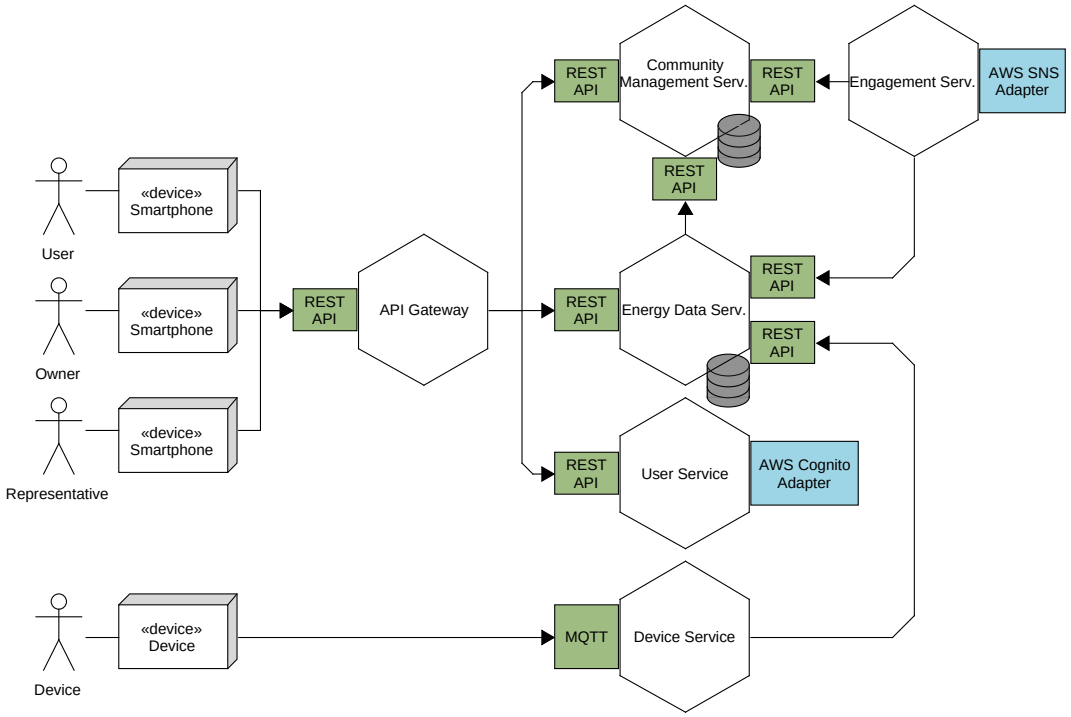


Figure 3.10: Designed microservices architecture

3.5 Detailed Architecture

The general architecture of the system has been defined, and the components have been identified, however, it is necessary to further detail the architecture of each component. The three main artifacts of the system are the Service, the App, and the Device, which respectively represent the application service, the mobile dashboard app, and the measuring device. The following sections analyze in detail each of the three components, justifying the design choices made and reporting the proposed architecture for each of them.

Service The Service component serves as the backend of the application and has the purpose of receiving, storing, and making available the data related to electrical measurements, as well as providing the necessary to manage users and the structure of a REC. Service is composed of five different microservices, each of which is responsible for a distinct functionality requiring different characteristics for its implementation.

Referring to the diagrams of Figure 3.10 and Figure 3.9, it can be noted the presence of numerous interfaces exposed by the different microservices, which are necessary to allow the interaction between them and with the outside. Most of the exposed interfaces are of the REST API type, which allows exposing the functionalities offered by the microservices in a simple and standardized way, using MQTT for the communication between the measuring device and the Device microservice. The following tables summarize the exposed interfaces, indicating the available methods and a brief description of the functionalities offered. For the Community Management and Energy Data microservices, the interfaces have been documented in more detail using the OpenAPI standard version 3.1.0[1]. The User microservice uses directly the interface implemented at Amazon Web Services (AWS) Cognito, the documentation of which is available in the documentation provided by AWS[2].

Table 3.8 and Table 3.9 show the exposed interfaces for the Community Management and Energy Data microservices, respectively. The type of these interfaces is REST API, which allows to expose the functionalities of the microservices in a

[1] `www.openapis.org`
[2] `docs.aws.amazon.com/cognito-user-identity-pools`

simple and standardized way.

Method	**Path**	**Description**
GET	/community	Get all communities
POST	/community	Add a new community
GET	/community/{communityId}	Get community info
PUT	/community/{communityId}	Update community info
DELETE	/community/{communityId}	Delete a community
GET	/community/{communityId} /device	Get all devices in a community
GET	/community/{communityId} /facility	Get all facilities in a community
PUT	/community/{communityId} /facility/{facilityId}	Add a facility to a community
DELETE	/community/{communityId} /facility/{facilityId}	Delete a facility from a community
POST	/facility	Add a new facility
GET	/facility/{facilityId}	Get facility info
PUT	/facility/{facilityId}	Update facility info
DELETE	/facility/{facilityId}	Delete a facility
GET	/facility/{facilityId} /device	Get device of a facility
PUT	/facility/{facilityId} /device/	Update device of a facility
GET	/facility/{facilityId} /community	Get community of a facility
GET	/facility/{facilityId} /user	Get all users of a facility
GET	/user/facility	Get all facilities of a user
GET	/user/community	Get all communities of a user
PUT	/user/{userId} /facility/{facilityId}	Update role of a user in a facility
DELETE	/user/{userId} /facility/{facilityId}	Remove a user from a facility

Table 3.8: Community Management microservice REST API

Method	Path	Description
POST	/energy-data/	Add new consumption information
GET	/energy-data/device/{deviceId}	Get consumption information of a device
GET	/energy-data/community/{communityId}	Get consumption information of a community

Table 3.9: Energy Data microservice REST API

Table 3.10 summarizes the exposed interface for the Device microservice, which uses the MQTT protocol to receive data from the measuring device. The API for the Device microservice is asynchronous and documented using the AsyncAPI standard version 3.0.0[3].

Action	Topic	Payload
PUBLISH	energy-data	{ deviceId: string, energyValue: number, timestamp: string }

Table 3.10: Device microservice Async API

The following diagrams show each microservice in terms of clean architecture, reporting on multiple levels of detail various aspects of each service. In particular, the highest level, that of the domain, summarizes the main entities, taking up concepts that had already been identified in the previous phases, while the application level reports the title of the relevant use cases for the service. Finally, the lowest level, which in this case includes both interfaces and infrastructure layers, provides more details on the technologies that have been chosen to implement the service.

Figure 3.11 shows the clean architecture for the Community Management microservice, reporting the entities and use cases related to the management of the structure of a REC. In the interfaces and infrastructure level, three components are reported that will be used to implement the service, namely API Gateway,

[3] `www.asyncapi.com`

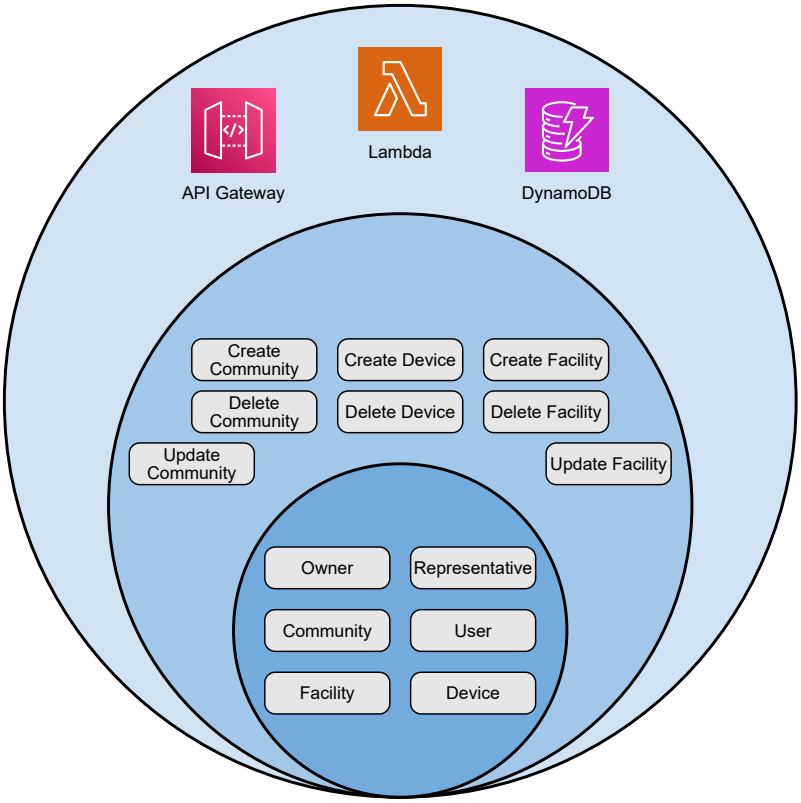
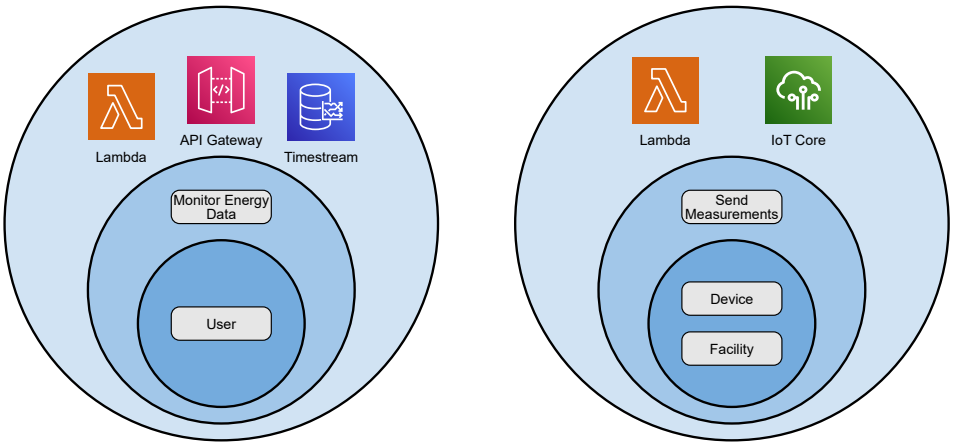


Figure 3.11: Clean Architecture for Community Management service

Lambda, and DynamoDB. These three components are used, respectively, to implement the REST API interface, execute serverless code to implement the application logic related to the management of a community, and store the data related to the structure of a REC. For the functionalities to be implemented in the Community Management service, API Gateway, Lambda, and DynamoDB represent a recommended choice to implement a serverless architecture that can offer scalability and high performance[Pat19].

Similarly, Figure 3.12a and Figure 3.12b show, respectively, the clean architecture for the Energy Data and Device microservices. In these diagrams, the entities and use cases related to the two microservices are reported, but different technologies are used compared to Community Management. In particular, for Energy Data, a time-dependent database is used through AWS Timestream, which unlike other types of databases allow storing and querying data based on their tempo-



(a) Clean Architecture for Energy Data service (b) Clean Architecture for Device service

rality. In addition to increasing performance, the use of Timestream allows to reduce costs, using for older and therefore less used data a different storage policy with lower costs at the expense of longer reading times[Win22]. Instead, for the Device microservice, AWS IoT Core is used, which includes various functionalities designed specifically for IoT applications. In addition to providing a highly performing MQTT broker for communication with devices, AWS IoT Core allows to manage the security and scalability of an IoT application.

Figure 3.13 summarizes the sequence of operations necessary to configure a device with AWS IoT Core, so that energy data can be sent securely from the Device to the Device service. The use of a claim certificate has been chosen to allow the device to authenticate itself initially with the service and perform the provisioning, as it was considered a good compromise between security and ease of implementation for this project. This procedure allows to communicate data securely, however, it is necessary to avoid that the claim certificate is compromised, as it could be used by a malicious actor to gain access to the service and compromise its correct functioning by inserting fake data and potentially reducing the availability of the service.

Figure 3.15b shows the clean architecture for the User microservice, which is responsible for managing the registration and authentication of users within the

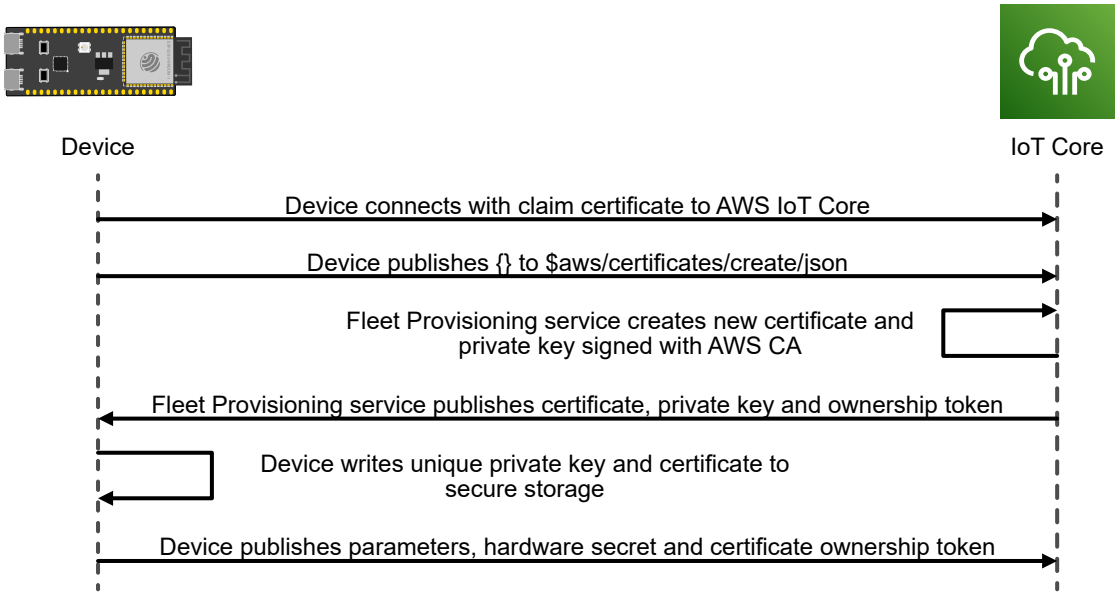


Figure 3.13: Overview of device fleet provisioning with claim certificate

system. In this case, the main entity is the user, which registers and authenticates with the system. At the interfaces and infrastructure level, AWS Cognito is used, as it provides a managed platform for storing users and registration and authentication procedures.

Figure 3.14 shows the authentication flow for a user within the system, using AWS Cognito. By using the Cognito platform, it is possible to obtain numerous advantages in terms of security, such as the use of zero knowledge authentication protocols, which offer greater security than traditional authentication methods[PPP+21]. Access to system resources is authenticated using JSON Web Token (JWT) tokens, which provide an authentication mechanism without having to maintain a server-side state, also using automatic token renewal to ensure greater security in accessing resources.

Finally, Figure 3.15a shows the clean architecture for the Engagement microservice, which is responsible for interacting with users through notifications. In this case, the main entity is the notification, which is sent to users when certain conditions are met. At the interfaces and infrastructure level, AWS SNS is used, as it provides a managed platform for sending notifications through various channels, such as Short Message Service (SMS), email, and push notifications. The use of

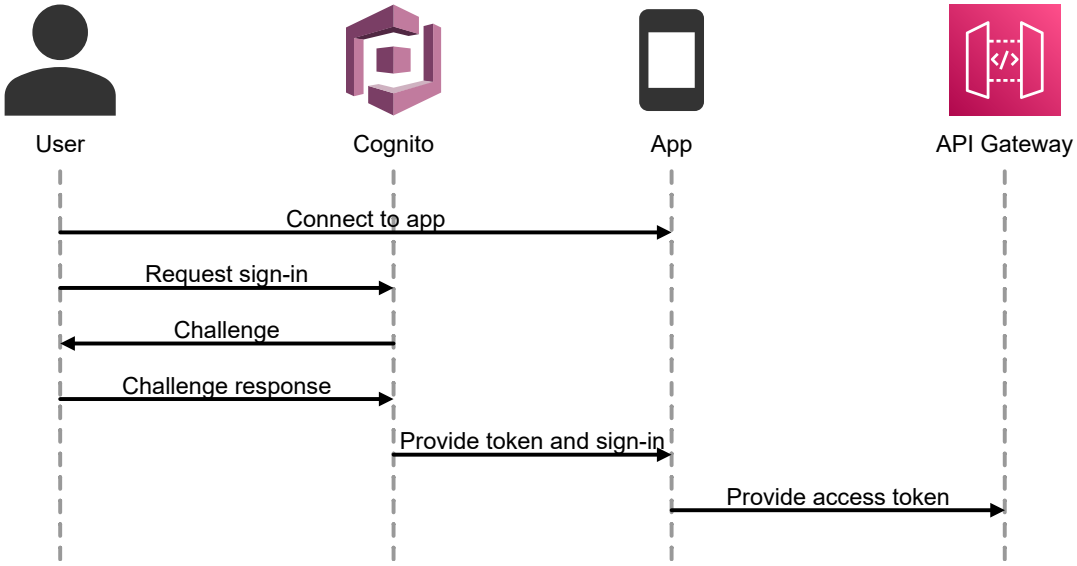
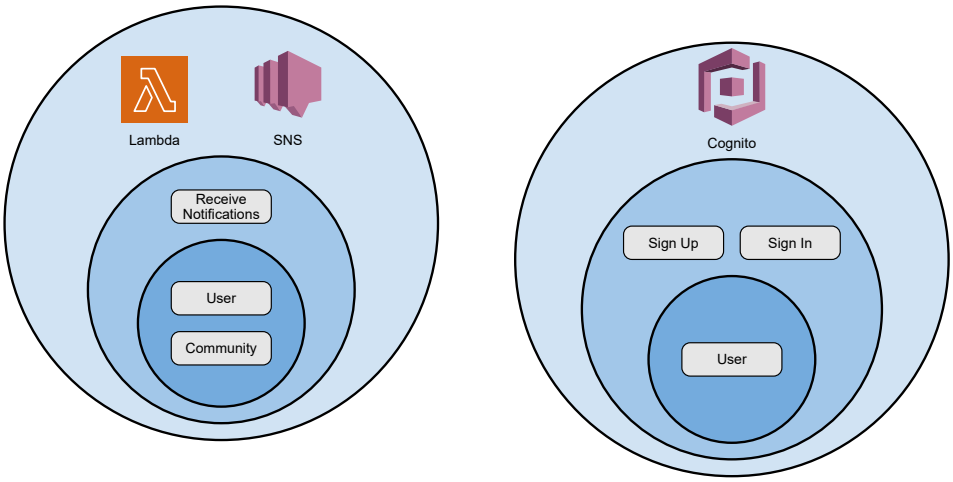


Figure 3.14: User authentication flow with AWS Cognito

SNS allows sending notifications in a simple and standardized way, using a single platform to manage the sending of notifications through different channels. This microservice also uses AWS Lambda to execute the application logic, which decides when to send notifications based on the data read from the other microservices.



(a) Clean Architecture for Engagement service

(b) Clean Architecture for User service

App The App component is designed to play the role of an interface between the user and the system, allowing to view the data and interact with the system. To develop this user interface, it was chosen to create a mobile application, as it allows greater flexibility in choosing the technologies to use to receive notifications. In particular, it was chosen to use React Native to develop the mobile application, as it allows to create a multi-platform mobile application with a single source code, an aspect that could reduce future costs for adapting the application to other platforms.

Device Device realizes the measuring device, which has the role of measuring and sending the data related to energy consumption and production of a facility to the system. An SoC belonging to the ESP32 family was chosen, given the high performance compared to competing microcontrollers, the presence of an integrated Wi-Fi module, and the low cost[MSV17]. In particular, it was chosen to use the ESP32-S3 module, one of the latest additions to the ESP32 family, which offers more powerful hardware and modern technologies compared to previous models[4]. In addition to the hardware advantages, the ESP32-S3 module offers extensive documentation and a feature-rich development framework, which could greatly simplify the implementation of the device software. The device is designed to be installed at a user's location, connected to the electricity meter to measure consumption and using the Wi-Fi module to send the data to the system.

3.6 Deployment

Figure 3.16 shows the deployment diagram of the system, summarizing at a high level of abstraction how the system is composed of different elements, each necessary for the correct functioning of the overall system. In this diagram, three main entities are shown, namely the mobile application, the measuring device, and the set of services that provide the required functionalities. The mobile application is designed to be simple and allow a user to interact and view the status of the system from their mobile device. This interface communicates with the services

[4] `docs.espressif.com`

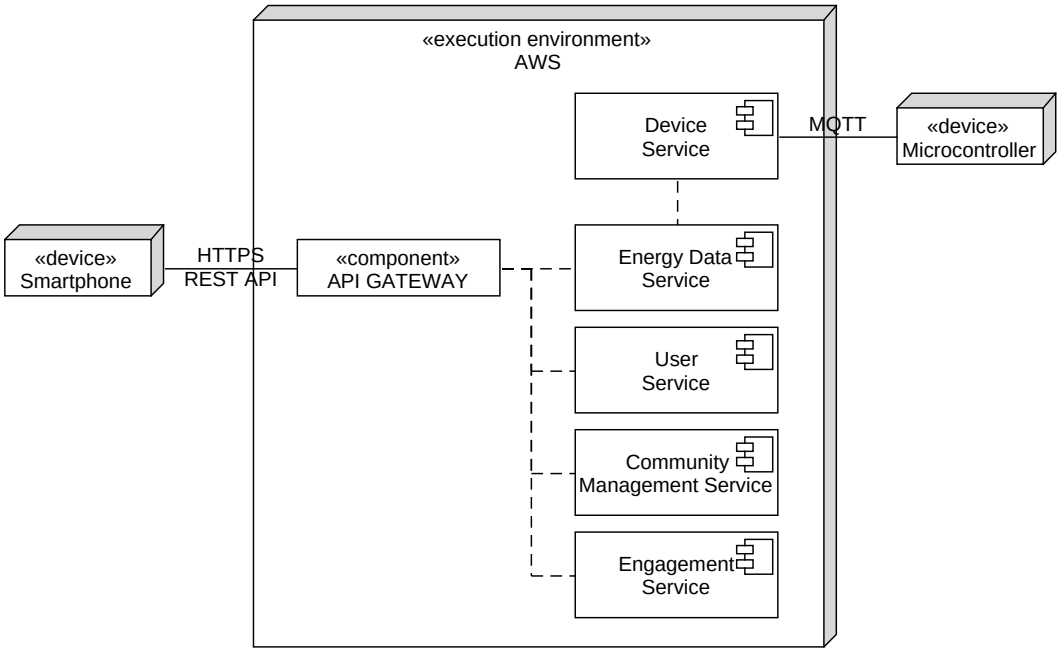


Figure 3.16: Deployment diagram

provided through the HTTP protocol using the REST APIs exposed by the different services. The measuring device is designed to be installed at a user's location and send the measurement data to the system through the MQTT protocol. Finally, the different services are executed within a serverless architecture, in this case using the services offered by AWS. For an external observer, the different services could be seen as if they were within the same execution context, even if in reality they will be executed in different execution contexts, an aspect that remains transparent from the user's point of view. However, by implementing a microservices architecture based on a serverless approach, each of them is designed to be modular and independent of the others, allowing to scale and manage them independently and automatically.

Chapter 4

Implementation

This chapter will report some relevant or non-trivial aspects encountered during the implementation phase of the system. Where necessary, detailed technical elements will be introduced to show some interesting aspects both regarding the software and hardware used. However, the main objective of this chapter is to show how the different components designed have been implemented and not to act as a guide to the installation or use of the system.

The identified requirements are considered to be stable and not subject to change, however, the implementation is carried out following an incremental approach, integrating Kanban methodologies. This decision was made to allow the implementation of each functionality incrementally with frequent releases, thanks to a division of the overall system into independent elements. Using an incremental approach allows to implement and deliver early parts of the system, allowing to have a partially working system that can be tested and analyzed earlier. A Kanban board was used to keep track of the progress of the various components, allowing to have a clearer overview of the remaining elements. The following are some of the tools that were used for the project documentation and activity management:

- Miro[1] is an online collaboration platform that allows to create diagrams, schemes, and organize activities. It was used especially during the analysis phase to visually present some concepts to domain experts. It was also used to keep track of the implemented work through a Kanban board.

[1] `miro.com`

- Egon.io[2] is an online platform for diagrams, it was used to create the domain storytelling diagrams.
- Umlet[3] is an open source tool for creating UML diagrams, it was used to create numerous diagrams included in this document.
- LaTeX[4] is a markup language for document creation, it was used to create this document. Unlike a traditional word processor, LaTeX allows to create structured documents more efficiently and to maintain a consistent formatting.
- Affinity Designer[5] is a vector graphics software, it was used to create some images and diagrams inserted in this document. This program was also chosen to create mock-ups of the mobile application.

4.1 Service

The following section will show the implementation details for the identified microservices, showing how they were developed and how they were configured to work within the system. Considering the domain model designed for the Community Management microservice, shown in Figure 3.5, the data schema for the DynamoDB database was designed, as shown in Figure 4.1. In this schema, the main entities of the subdomain are shown, namely Community, Facility, Device, and User, which include some relevant information and the relationships between them.

By combining the schema in Figure 4.1 with the access patterns identified in Section 3.2, it is possible to design the access pattern for the database. The access pattern is fundamental to define how the data will be read and written within the database, in order to guarantee optimal performance and scalability of the system. To implement the database for the Community Management microservice, DynamoDB was chosen, a NoSQL database service offered by AWS. DynamoDB

[2] `egon.io`
[3] `www.umlet.com`
[4] `www.latex-project.com`
[5] `affinity.serif.com`

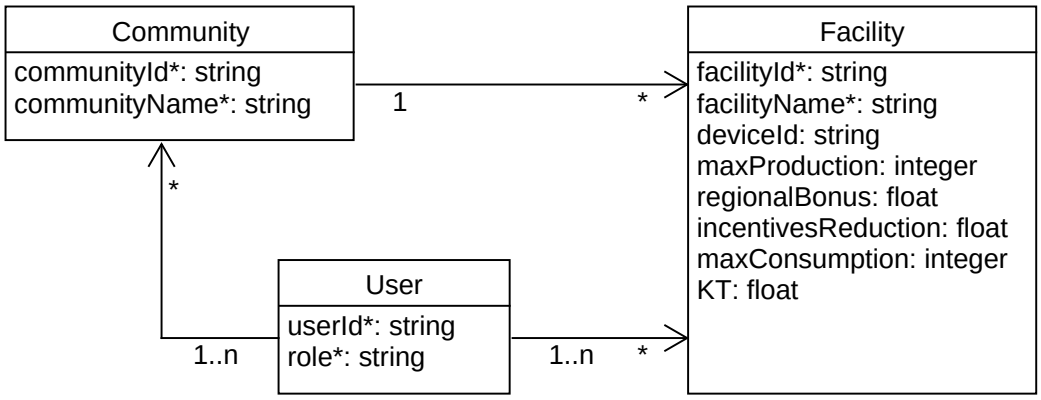


Figure 4.1: Model schema for Community Management service

is a fully managed, highly scalable, and versatile service, which allows to store and retrieve data in an efficient way with latencies of a few milliseconds. However, the use of a NoSQL database requires designing the database differently from a traditional database, as it does not support the classic join operations between tables and requires to design the database based on the queries that will be performed. DynamoDB distributes data across multiple physical partitions based on hashing functions performed on the Partition Key (PK), in order to guarantee scalability and high performance, an aspect that in the case of non-optimal design can cause hot partitions problems, that is, partitions that receive a very high load of requests compared to others. The analysis of the access pattern is therefore fundamental to identify which data will be frequently read and written, in order to design the tables considering these access criteria.

#	**Access pattern**	**Query by**
1	Get facility info by facility	facilityId
2	Get device by facility	facilityId
3	Get all facilities by community	communityId
4	Get community info by community	communityId
5	Get community by facility	facilityId
6	Get all facilities for a user	userId

7	Get all users for a facility	facilityId
8	Get all communities for a user	userId
9	Get user role in facility	userId, facilityId
10	Get user role in community	userId, communityId
11	Get all communities	
12	Get all devices for a community	communityId

The database for the Community Management microservice was designed following the Single Table Design pattern, which involves storing all data within a single table. This design pattern allows to access data very efficiently, however it requires careful design of the table. Figure 4.2 shows the table design for the Community Management microservice, where through the use of PK, Sort Key (SK), and Global Secondary Index (GSI) it is possible to access data efficiently according to the identified access patterns. Figure 4.3 shows an example of data for

Primary Key		Global Secondary Index		Attributes					
Partition key PK	Sort key SK	Partition key GSI_PK	Sort key GSI_SK						
facility#facilityId	INFO			facilityName	maxProduction	regionalBonus	incentivesReduction	maxConsumption	KT
	DEVICE			deviceId	...				
community#communityId	INFO	COMMUNITY	COMMUNITY	communityName	...				
	facility#facilityId	facility#facilityId	community#communityId						
user#userId	facility#facilityId	facility#facilityId	user#userId	role					
	community#communityId	community#communityId	user#userId	role					

Figure 4.2: Single Table Design for the Community Management service database

the Community Management microservice, in the case of a community with two facilities, each with a device associated and some users.

Primary Key		Global Secondary Index		Attributes					
Partition key PK	Sort key SK	Partition key GSI_PK	Sort key GSI_SK						
facility#001	INFO			"Apartment 1"	0	0.0	0.0	3000	1
	DEVICE			"001"	...				
facility#002	INFO			"Apartment 2"	10000	4.0	0.0	4500	1
	DEVICE			"002"					
community#001	INFO	COMMUNITY	COMMUNITY	"City community"	...				
	facility#001	facility#001	community#001						
	facility#002	facility#002	community#001						
user#001	facility#001	facility#001	user#001	OWNER					
	facility#002	facility#002	user#001	OWNER					
user#002	facility#002	facility#002	user#002	USER					
user#003	community#001	community#001	user#003	REPRESENTATIVE					

Figure 4.3: Example of data for the Community Management service database

#	**Access pattern**	**Query structure**
1	Get facility info by facility	PK=facility#facilityId SK="INFO"
2	Get device by facility	PK=facility#facilityId SK="DEVICE"
3	Get all facilities by community	PK=community#communityId SK=facility#
4	Get community info by community	PK=community#communityId SK="INFO"
5	Get community by facility	GSI_PK=facility#facilityId GSI_SK=community#
6	Get all facilities for a user	PK=user#userId SK=facility#
7	Get all users for a facility	GSI_PK=facility#facilityId GSI_SK=user#
8	Get all communities for a user	PK=user#userId SK=community#
9	Get user role in a facility	PK=user#userId SK=facility#facilityId
10	Get user role in community	PK=user#userId SK=community#communityId
11	Get all communities	GSI_PK="COMMUNITY"
12	Get all devices for a community	PK=community#communityId SK=device#

Each microservice is realized using an Infrastructure as Code (IaC) approach, using the Cloud Development Kit (CDK) framework to define the infrastructure and the application logic. The CDK framework allows to define the infrastructure using a programming language, in this case TypeScript, which allows to define the infrastructure in a more structured and maintainable way. The following code listing demonstrates a simplified example of a microservice that defines a simple REST API route using AWS Lambda, API Gateway, and DynamoDB.

Listing 4.1: Example of a microservice stack using AWS CDK

```
export class ExampleService extends cdk.Stack {
    constructor(scope: Construct, id: string, props?: cdk.StackProps) {
        super(scope, id, props);

        // Lambda
        const exampleLambda = new lambda.Function(this, `ExampleLambda`, {
            runtime: lambda.Runtime.NODEJS_18_X,
            code: lambda.Code.fromAsset('lambda/example'),
            handler: 'example.handler',
            environment: {...},
        });

        // DynamoDB
        const exampleTable = new dynamodb.Table(this, `ExampleTable`, {
            partitionKey: { name: 'PK', type: dynamodb.AttributeType.STRING },
            sortKey: { name: 'SK', type: dynamodb.AttributeType.STRING },
        });
        exampleTable.grantReadWriteData(exampleLambda);

        // API Gateway
        const api = new apigateway.RestApi(this, `ExampleApi`, {...});
        const example = api.root.addResource('example');
        example.addMethod('GET', new apigateway.LambdaIntegration(exampleLambda));
    }
}
```

4.2 App

The mobile application was developed using React Native, a framework that allows to create multi-platform mobile applications using a single source code. There are no significant implementation details to report, as the application realizes a simple prototype to show the functionalities of the system. The diagram reported in Figure 4.4, shows the structure of the screens implemented, as well as the main functionalities offered by each of them to the user.

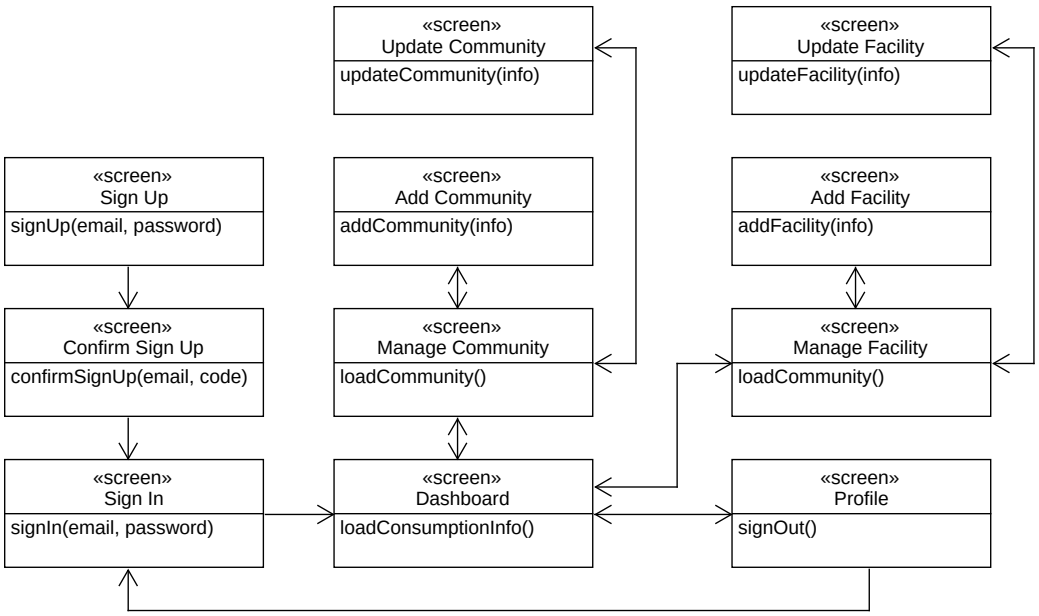


Figure 4.4: Implemented screens of the mobile application

4.3 Device

The measuring device is implemented using an ESP32-S3[6] module, using the ESP-IDF framework to develop the software. The final version of the device will use dedicated hardware to implement the Chain2 protocol and retrieve energy data directly from the electricity meter, while being connected to whichever electric socket available in a user's facility. This communication is possible by using the ST75MM[7] SoC, which allows power line communication with the electricity meter.

The prototype device realized to test the system uses a simple current sensor to measure the current flow in the lines of the facility, this requires installing a current clamp around the electrical cable after the electricity meter. Figure 4.5 shows the activity diagram for the device, which illustrates the main activities that the device performs during its operation. The diagram of the circuit realized for the prototype device is shown in Figure 4.6.

[6] www.espressif.com/en/products/socs/esp32-s3
[7] www.st.com/interfaces-and-transceivers/st75mm

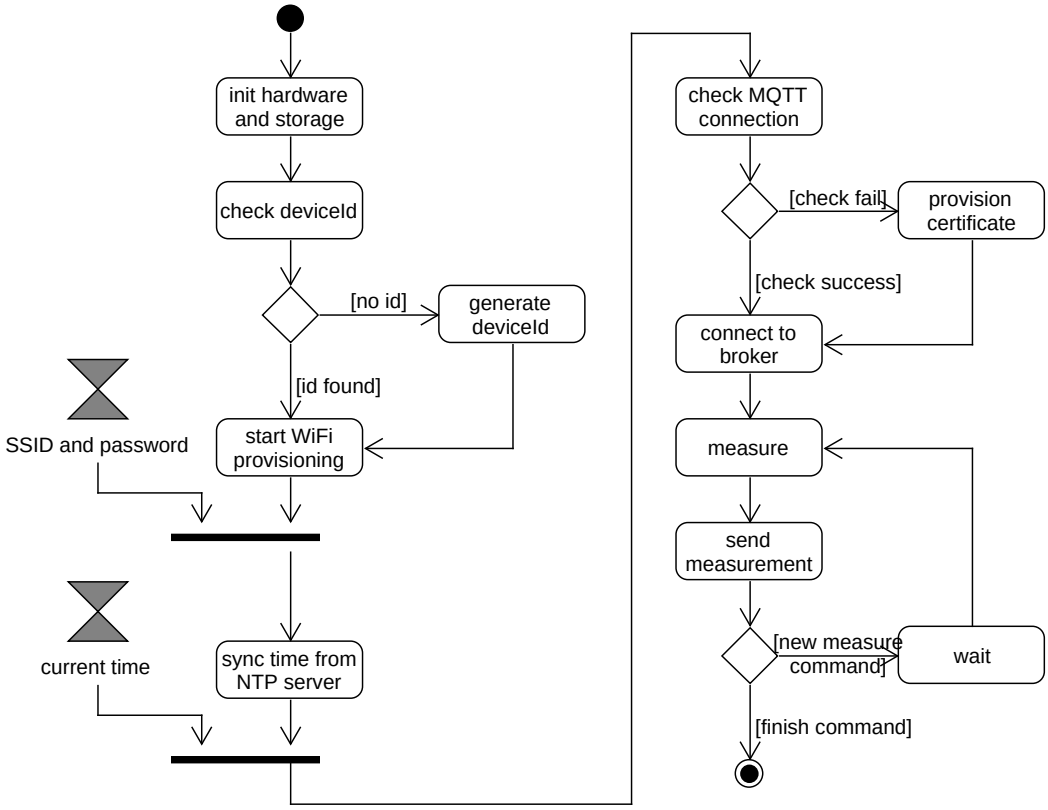


Figure 4.5: Activity diagram for the device

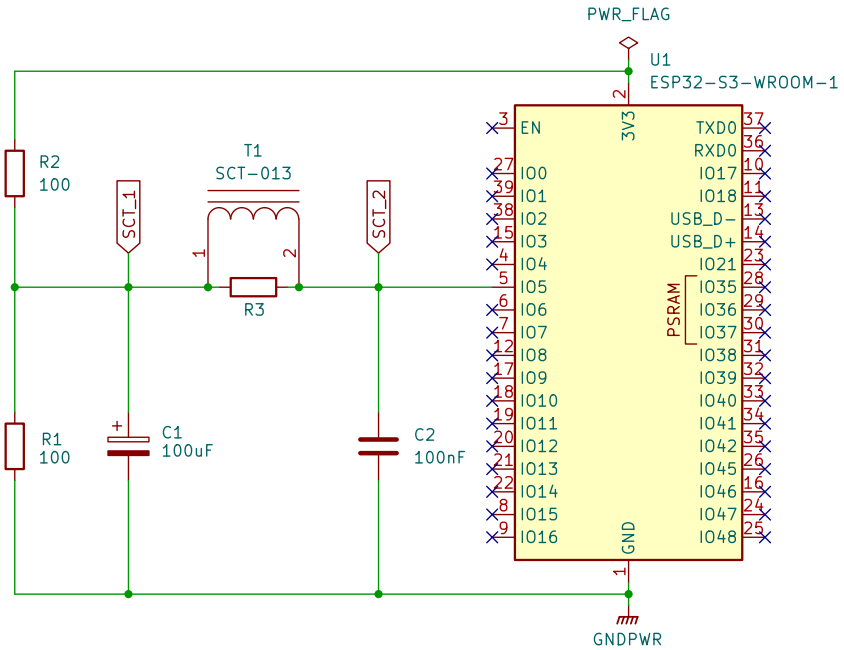


Figure 4.6: Schematic of the prototype device

4.4 DevOps

This project uses a DevOps [EGHS16] methodology to increase the quality of the system by making development and validation easier. DevOps techniques aim to eliminate the gap between code in development and code released, using a series of mechanisms to automate various release and integration processes. The two fundamental principles are in fact Continuous Integration (CI) and Continuous Delivery (CD), that is, the frequent integration and distribution of the software. This approach allows to frequently evaluate the complete system, significantly reducing potential integration problems and errors related to human factors. It also improves development efficiency by automating repetitive processes wherever possible, freeing the developer from this aspect and providing automatic and predictable mechanisms.

Workflow Before defining the CI and CD techniques to be used, it is useful to establish the project workflow, that is how the system development is managed, so that automation can be added to the identified processes. This project uses

a branching model strongly inspired by GitFlow, making some modifications to simplify it and better adapt it to the development context.

As in GitFlow, there are two main branches, one dedicated to production code called "main" and one dedicated to integrating code during development, called "dev". On these two main branches, commits are never made directly, and the code is integrated from other branches through merge commits. In fact, starting from the development branch, numerous branches dedicated to different features are created, which are then merged back into the development branch through a merge commit to integrate the developed features once they are considered complete. When it is considered that the code integrated on the development branch is ready for release, a merge to the main branch is made. In case of urgent problems in production, hotfixes are implemented in hotfix branches before being directly brought into the main branch. When a bug discovered is less urgent and affects a significant part of the code, which therefore affects the code of more than one feature, a bugfix branch is used. Very contained bugs that affect only one feature can be managed directly in the feature branch, considering them as an expansion of the feature itself.

The main differences with the model introduced by GitFlow are the absence of a specific branch for release, called release branch, where the features are evaluated before being merged with the production code. This aspect has been integrated directly into the development branch, considering the absence of a large team and the presence of continuous integration and deployment tools, significantly reducing the need for a dedicated branch. The second difference is the presence of bugfix branches, absent in GitFlow and introduced here to solve non-urgent problems that affect multiple features, maintaining more coherence and consequently a cleaner history of the source code.

The image in Figure 4.7 shows an example of the workflow used, indicating some of the typical scenarios that could occur during the development phase. In addition, the messages of the commits made will use the Conventional Commits 1.0.0 specification[8], using, in particular, the following tags: feat, fix, refactor, and docs.

[8] `www.conventionalcommits.org`

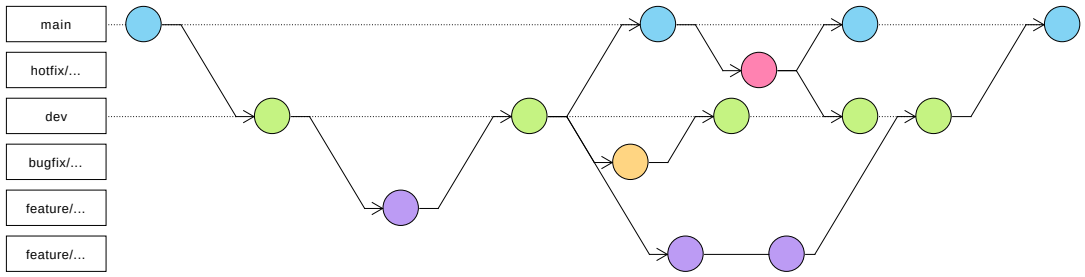


Figure 4.7: Example of the workflow used for this project

Versioning To version the code, the Semantic Versioning 2.0.0[9] rules are used, using numbers with the format MAJOR.MINOR.PATCH to indicate specific released versions of the software. The first released version is indicated with 0.1.0, while the first production release of the software is indicated with 1.0.0, indicating the end of the initial development period. When a new feature is released, the number of MINOR is increased, resetting the number of PATCH, which is reserved to indicate software updates aimed at fixing bugs without adding new features.

Continuous Integration & Deployment To implement CI and CD techniques, AWS CodePipeline was used to automatically deploy the CDK application. The system provides two pipelines for each microservice, one for the production environment and one for the development environment. A dedicated stage is created for each environment, which is activated by the detection of updates on the main and dev branches. In this way, it is possible to separate two separate environments to add and test changes during development without affecting the production environment.

Each microservice can be configured and deployed using the AWS Management Console, but a better solution is to implement the system with an IaC approach[Mor20]. In this way, the infrastructure is defined through code, ensuring automation and speed in the creation of the infrastructure, but also increasing the simplicity in managing aspects of scalability and flexibility. Other advantages of using IaC include managing through Version Control System (VCS) to track and undo changes if necessary, as well as providing automatic configuration validation

[9] `semver.org`

tools before deployment. It is also possible to automate multi-cloud deployments without the need to use different consoles for each service provider. Finally, consistency is guaranteed between different deployments, eliminating possible errors that could arise during manual configuration.

The code in Listing 4.2 shows a simplified example of a pipeline for a microservice. This pipeline automatically fetches the latest changes published to a GitHub repository branch, executes some build steps, and finally deploys the application to the AWS environment.

Listing 4.2: Example of a code pipeline for a microservice

```
export class ExamplePipeline extends cdk.Stack {
    constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
        super(scope, id, { env: { ... }, ...props });

        const examplePipeline = new CodePipeline(this, 'ExamplePipeline', {
            pipelineName: 'ExamplePipeline',
            synth: new ShellStep('Synth', {
            input: CodePipelineSource.gitHub(serviceConfig.repository, 'main', {
                authentication: cdk.SecretValue.secretsManager('github-token'),
            }),
            commands: ['npm install', 'npm run build', 'npx cdk synth'],
            }),
        });
        examplePipeline.addStage(new ServiceStage(this, 'ExampleStage', {
            env: { ... }, stageName: 'Prod',
        }));
    }
}

interface ServiceStageProps extends cdk.StageProps {
    stageName: string;
}

class ServiceStage extends cdk.Stage {
    constructor(scope: Construct, id: string, props: ServiceStageProps) {
        super(scope, id, props);

        new ExampleService(this, `ExampleService-${props.stageName}`, {
            stageName: props.stageName,
        });
    }
}
```

Chapter 5

Evaluation

This chapter will evaluate the system developed, focusing on the main aspects that have been implemented and the results obtained. The evaluation will be carried out by analyzing the system from different points of view, from the validation of single components to the integration of the system as a whole. Furthermore, the evaluation will consider observability patterns, which are used to monitor the system and identify possible problems or anomalies.

The validation of each microservice can be verified statically using the Synth tool, which is included in the CDK framework. This tool allows to verify the correctness of the infrastructure configuration before deploying it, intercepting configuration errors even before deployment.

For each microservice, CloudWatch provides all the necessary tools to view the logs generated during execution, allowing to identify problems or anomalies during development or execution of a service. In addition, CloudWatch allows to set alarms to monitor resources and generate notifications in case of problems, defining customized rules and thresholds for different metrics of a service. The metrics that can be monitored include, for example, the number of errors generated by a service, the response time of an API call, or the number of requests received by a service. In case the observed metric enters a trigger state, a notification is generated, which can be sent through different channels, such as email, SMS, or a Lambda function invocation. The following list shows the main channels that can be used to send notifications:

Service	Metric	Threshold	Action
Community Management	5XX Errors	1	Email
Community Management	4XX Errors	10	Email
Community Management	Latency	3000ms	Email
Community Management	Count	1000	Email
Energy Data	5XX Errors	1	Email
Energy Data	4XX Errors	10	Email
Energy Data	Latency	3000ms	Email
Energy Data	Count	1000	Email

Table 5.1: CloudWatch alarms for the implemented services, assuming a threshold reset period of 1 minute

- Email: sending a notification through a list of specified email addresses;
- SMS: sending a notification through an SMS message to the specified phone numbers;
- Lambda Function: executing a custom Lambda function;
- HTTP/HTTPS Endpoint: sending an HTTP/HTTPS request to a specified endpoint;
- SQS Queue: sending a message to an AWS SQS queue;
- Application: sending a notification to a mobile application through AWS SNS.

All these channels can be used to send notifications to different recipients, allowing to customize the notification system according to the needs of the system. Table 5.1 shows the alarms implemented for the different services.

Integration test To test the integration between the different services, AWS offers a tool called X-Ray, which allows to monitor the interaction flows between the different services. X-Ray allows to trace the requests made between the different services, showing the time taken for each operation and any calls made, allowing to identify any performance problems or integration errors. Although it is not considered a testing tool, X-Ray is still useful for identifying dependencies between

the various components of the system, whether internal or external, and identifying any runtime integration problems.

Component test Testing of the single components is carried out using Jest[1], a testing framework that allows to test JavaScript and TypeScript code. Jest allows to test single components, simulating the necessary resources and verifying the behavior of a component in an isolated way. The components that have been tested include mainly the Community Management service, for which tests have been implemented to verify the correct functioning of the main operations. The Listing 5.1 summarizes the tests implemented for this service.

Listing 5.1: Jest component tests for the Community Management microservice

```
PASS  test/jest/management/getUserCommunity.test.ts
 getUserCommunity
   * should return unauthorized if request is unauthorized
   * should return all communities of a user
   * should return an empty array if user has no communities
   * should return internal server error if an error occurs

PASS  test/jest/management/postFacility.test.ts
 postFacility
   * should return unauthorized if request is unauthorized
   * should return bad request if request body is null
   * should return bad request if facility name is null
   * should create a new facility

PASS  test/jest/management/facility.test.ts
 Facility
   * should create a new Facility object from query data
   * should create a new Facility object using the constructor
   * should create a new Facility with users

PASS  test/jest/management/getCommunityId.test.ts
 getCommunityId
   * should return unauthorized if request is unauthorized
   * should return bad request if userId or communityId is missing
   * should return forbidden if user does not have the required role
   * should return not found if community does not exist
   * should return an empty array if user has no communities
   * should return community data if community exists
   * should return internal server error if an exception occurs

PASS  test/jest/management/getUserFacility.test.ts
```

[1] jestjs.io

```
  getUserFacility
    * should return unauthorized if request is unauthorized
    * should return all facilities of a user
    * should return an empty array if user has no facilities
    * should return internal server error if an error occurs

 PASS  test/jest/management/community.test.ts
  Community
    * should create a new Community object
    * should create a new Community object from query data

 PASS  test/jest/management/postCommunity.test.ts
  postCommunity
    * should return unauthorized if request is unauthorized
    * should return bad request if request body is null
    * should return bad request if community name is null
    * should create a new community

 PASS  test/jest/management/getFacilityId.test.ts
  getFacilityId
    * should return unauthorized if request is unauthorized
    * should return bad request if userId or facilityId is missing
    * should return forbidden if user does not have the required role
    * should return not found if facility does not exist
    * should return an empty array if user has no facilities
    * should return facility data if facility exists
    * should return internal server error if an exception occurs

Test Suites: 8 passed, 8 total
Tests:       35 passed, 35 total
```

End-to-end test To test the system as a whole, end-to-end tests have been implemented through the definition of user journey test. User journey tests are acceptance tests that test the system as a whole, simulating the interaction of a user with the system. These tests are carried out with the entire system in operation, simulating an interaction with the system as a user would do.

Performance Tests The system's performance has been evaluated only on some components of the system, in particular for the Community Management and Energy Data microservices. To evaluate the system performance, k6[2] was used, an open-source tool to perform load tests and evaluate the performance of an

[2] k6.io

application. For the Community Management microservice, it was chosen to test the read and write operations of the information related to the entities of an energy community, in order to evaluate the performance of the system under load conditions. For the Energy Data microservice, it was chosen to test the read operations of the energy measurements sent previously by the devices, in order to evaluate the performance of the retrieval of the data. The load was simulated by using different virtual users that simulate the interaction with the system, performing read and write operations of the information concurrently. To perform the test, it was chosen to use a base load of 10 virtual users, with an additional variable load composed of 40 virtual users. The variable load of virtual users was gradually increased over 30 seconds and then returned to 0 over the next 30 seconds. The test performed has a total duration of 60 seconds. Listing 5.2 shows the scenario created during the performance tests.

Listing 5.2: k6 scenario for get items of a user operations

```
* constantLoad: 10 looping VUs for 1m0s (gracefulStop: 30s)
* rampingLoad: Up to 40 looping VUs for 1m0s over 2 stages (gracefulRampDown: 30s,
    gracefulStop: 30s)

running (0m22.3s), 39/50 VUs, 454 complete and 0 interrupted iterations
constantLoad   [=============>------------------------] 10 VUs     0m22.3s/1m0s
rampingLoad    [=============>------------------------] 29/40 VUs  0m22.3s/1m00.0s
```

Listing 5.3 shows the results obtained during the performance test. In particular, the results obtained show that the system is able to handle a load of 50 virtual users with an average response time of 157.35 ms and a 95th percentile response time of 252.3 ms. In addition, the number of operations successfully completed is 99.90%.

Listing 5.3: k6 results for the Community Management microservice

```
checks.....................: 99.90% 3157 out of 3160
data_received..............: 1.2 MB 19 kB/s
data_sent..................: 133 kB 2.2 kB/s
http_req_blocked...........: avg=1.18ms   min=52ns    med=202ns    max=45.14ms  p
    (90)=320ns    p(95)=417ns
http_req_connecting........: avg=565.76us min=0s      med=0s       max=23.89ms  p
    (90)=0s       p(95)=0s
http_req_duration..........: avg=157.35ms min=44.1ms  med=143.19ms max=618.42ms p
    (90)=217.07ms p(95)=252.3ms
```

```
  { expected_response:true }: avg=157.56ms min=75.58ms med=143.19ms max=618.42ms p
      (90)=217.08ms p(95)=252.34ms
http_req_failed.............: 0.18%  3 out of 1580
http_req_receiving..........: avg=123.29us min=5.76us  med=20.77us  max=41.32ms  p
    (90)=232.79us p(95)=428.6us
http_req_sending............: avg=23.17us  min=5.79us  med=21.06us  max=238.89us p
    (90)=34.78us  p(95)=38.71us
http_req_tls_handshaking....: avg=619.26us min=0s      med=0s       max=23.01ms  p
    (90)=0s       p(95)=0s
http_req_waiting............: avg=157.2ms  min=44.08ms med=143.03ms max=618.29ms p
    (90)=217.02ms p(95)=251.82ms
http_reqs...................: 1580   25.846283/s
iteration_duration..........: avg=1.15s    min=1.04s   med=1.14s    max=1.61s    p
    (90)=1.21s    p(95)=1.25s
iterations..................: 1580   25.846283/s
vus.........................: 4      min=4            max=50
vus_max.....................: 50     min=50           max=50
```

Similarly, Listing 5.4 shows the results obtained during the performance test for the Energy Data microservice. In this case, the results obtained show that the system is able to handle a load of 50 virtual users with an average response time of 187.11 ms and a 95th percentile response time of 250.81 ms. The number of operations successfully completed in this case is 99.83%.

Listing 5.4: k6 results for the Energy Data microservice

```
checks......................: 99.83% 1176 out of 1178
data_received...............: 1.8 MB 29 kB/s
data_sent...................: 61 kB  978 B/s
http_req_blocked............: avg=3.76ms   min=200ns    med=1.01us   max=50.63ms
    p(90)=1.63us   p(95)=43.09ms
http_req_connecting.........: avg=1.7ms    min=0s       med=0s       max=24.45ms
    p(90)=0s       p(95)=19.41ms
http_req_duration...........: avg=187.11ms min=89.29ms  med=162.88ms max=1.42s
    p(90)=222.07ms p(95)=250.81ms
  { expected_response:true }: avg=187.44ms min=120.09ms med=162.93ms max=1.42s
      p(90)=222.19ms p(95)=250.87ms
http_req_failed.............: 0.33%  2 out of 589
http_req_receiving..........: avg=181.6us  min=27.51us  med=118.31us max=3.61ms
    p(90)=430.27us p(95)=558.49us
http_req_sending............: avg=99.02us  min=19.58us  med=97.07us  max=358.87us
    p(90)=133.3us  p(95)=142.84us
http_req_tls_handshaking....: avg=2.03ms   min=0s       med=0s       max=29.55ms
    p(90)=0s       p(95)=23.06ms
http_req_waiting............: avg=186.83ms min=89.03ms  med=162.28ms max=1.42s
    p(90)=221.83ms p(95)=250.42ms
http_reqs...................: 589    9.509176/s
iteration_duration..........: avg=3.19s    min=3.12s    med=3.16s    max=4.47s
```

```
    p(90)=3.23s    p(95)=3.25s
iterations.................: 589    9.509176/s
vus........................: 12     min=11          max=50
vus_max....................: 50     min=50          max=50
```

Quality Attribute Scenarios The validation of the system also includes the verification of the different quality scenarios identified during the requirement analysis. For each of the different scenarios, reported in Section 2.6, it was verified that the system meets the requirements identified during the analysis. The quality scenarios identified include:

- **Performance**: the system must respond in less than 2 seconds for the operations performed by a user. In the two services with which a user interacts, Community Management and Energy Data, the average response time is less than 2 seconds, as indicated by the performance tests performed.

- **Compatibility**: the system must be usable on a numerous devices, ensuring high compatibility. The mobile application developed to interact with the system is compatible with Android operating systems that have an API level of 30 or higher, meeting the compatibility requirement. Devices with an API level greater or equal to 30 represent 82.2% of all devices currently in use according to API Level reports[3].

- **Modifiability**: the system must be easily modifiable, ensuring a quick deployment for the implemented changes. This aspect has been addressed for the services part of the system using dedicated pipelines for each service, which allow automatic deployment of the service following the publication of a new version of the code in the corresponding repository. Through AWS CodePipeline it is possible to verify the correct functioning of the deployment pipelines and the necessary times, allowing to verify the correctness of this scenario.

- **Availability**: the system must be available to the user continuously, ensuring an uptime of the system of 99% of the time. The implemented system

[3] `apilevels.com`

uses highly available AWS services, which provide uptime guarantees higher than the required value. In addition, for the Community Management and Energy Data services, the availability of the system was verified during a realistic usage scenario using load tests, obtaining results higher than those required.

- **Usability**: all system functionalities must be easily usable by the user, requiring a maximum time of 1 minute for the completion of each operation. This scenario was verified through user journey tests, which simulate the interaction of a user with the complete system, verifying the time required for the completion of each operation.

- **Accessibility**: this scenario requires that the system be accessible to color-blind users, in accordance with the guidelines expressed by WCAG. The mobile application, the only component of the system that requires direct interaction with the user, was verified using color contrast detection tools, allowing to measure the level of accessibility of the application. The color contrast table used for verification is shown in Table 5.2, where any contrast ratio greater than 7:1 is considered accessible for normal text, while anything above 4.5:1 is suitable for large text. Any graphical object or user interface component is considered compliant with the WCAG guidelines if it has a contrast ratio greater than 3:1.

Foreground	Background	Contrast	Compliance
#B3B3B3	#0D0D0D	9.26:1	AAA
#E6E6E6	#0D0D0D	15.57:1	AAA
#F2F2F2	#0D0D0D	17.36:1	AAA
#B3B3B3	#1A1A1A	8.3:1	AAA
#E6E6E6	#1A1A1A	13.94:1	AAA
#F2F2F2	#1A1A1A	15.54:1	AAA
#FFD500	#0D0D0D	13.66:1	AAA
#FFD500	#1A1A1A	12.24:1	AAA
#0D0D0D	#FFD500	13.66:1	AAA
#E5D1FA	#7317CF	5.38:1	AA*/AAA

Table 5.2: Accessibility color contrast table for the mobile application

Conclusions and Future Work

This document describes the development of a system for monitoring RECs that also includes a measuring device to gather energy data and a mobile application to interact with the system. Although similar systems are already being worked on or are currently in use in this new and growing field, the developed system differs from the others due to some unique qualities. This project aims to reduce operating expenses and meet the need for a solution that benefits the entire community, not only representatives.

The system that was designed is capable of meeting the identified requirements; therefore, the primary goals that were established have been accomplished. As stated in Chapter 5, the validation of the accomplishment of the objective was assessed from many perspectives.

The implemented system, however, might still be enhanced and expanded upon. The first step is to improve the mobile application since the current version is to be considered a prototype that only allows interaction with the system for the bare minimum of features. To complete the application, a variety of adjustments are required, particularly to increase the user experience. Additionally, in order to enable the application to be used on a larger number of devices, the future expansion of the mobile application to the iOS platform should be assessed.

Some improvements are also needed for the measuring device in order to increase its effectiveness and reliability. The project was validated using a prototype that uses a current sensor to take measurements, as shown in Figure 5.1, instead of using the Chain2 communication protocol to exchange data with the electronic meter. Since it necessitates specialized and in-depth knowledge of the protocol and the hardware components required for communication, implementing communication through the Chain2 protocol is outside the scope of this thesis. Based

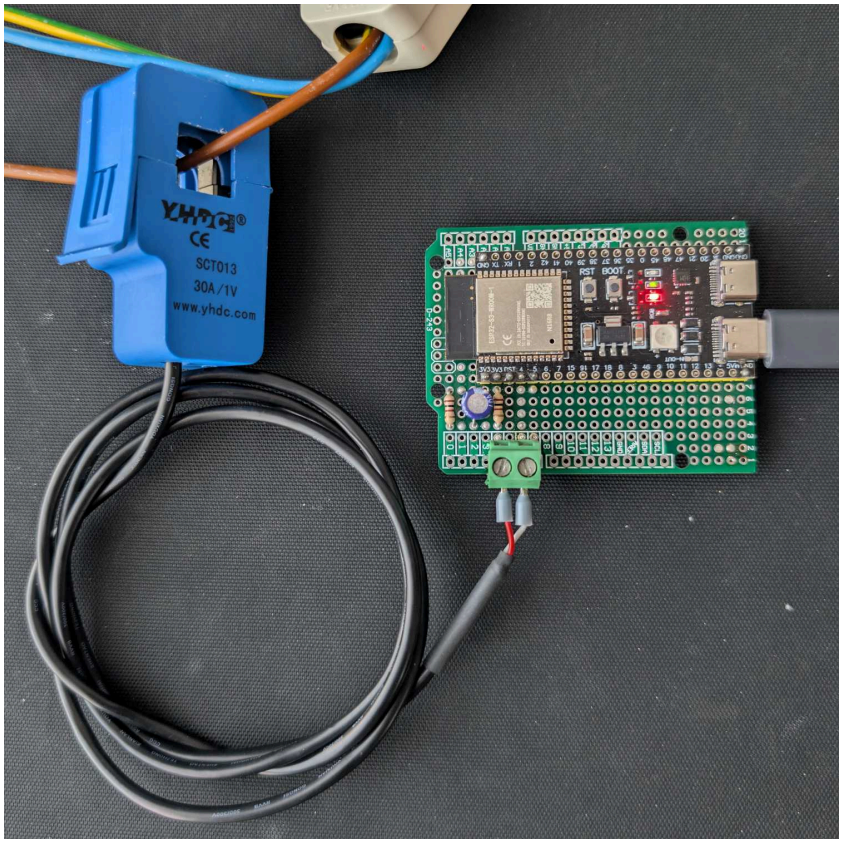


Figure 5.1: Photograph of the device prototype

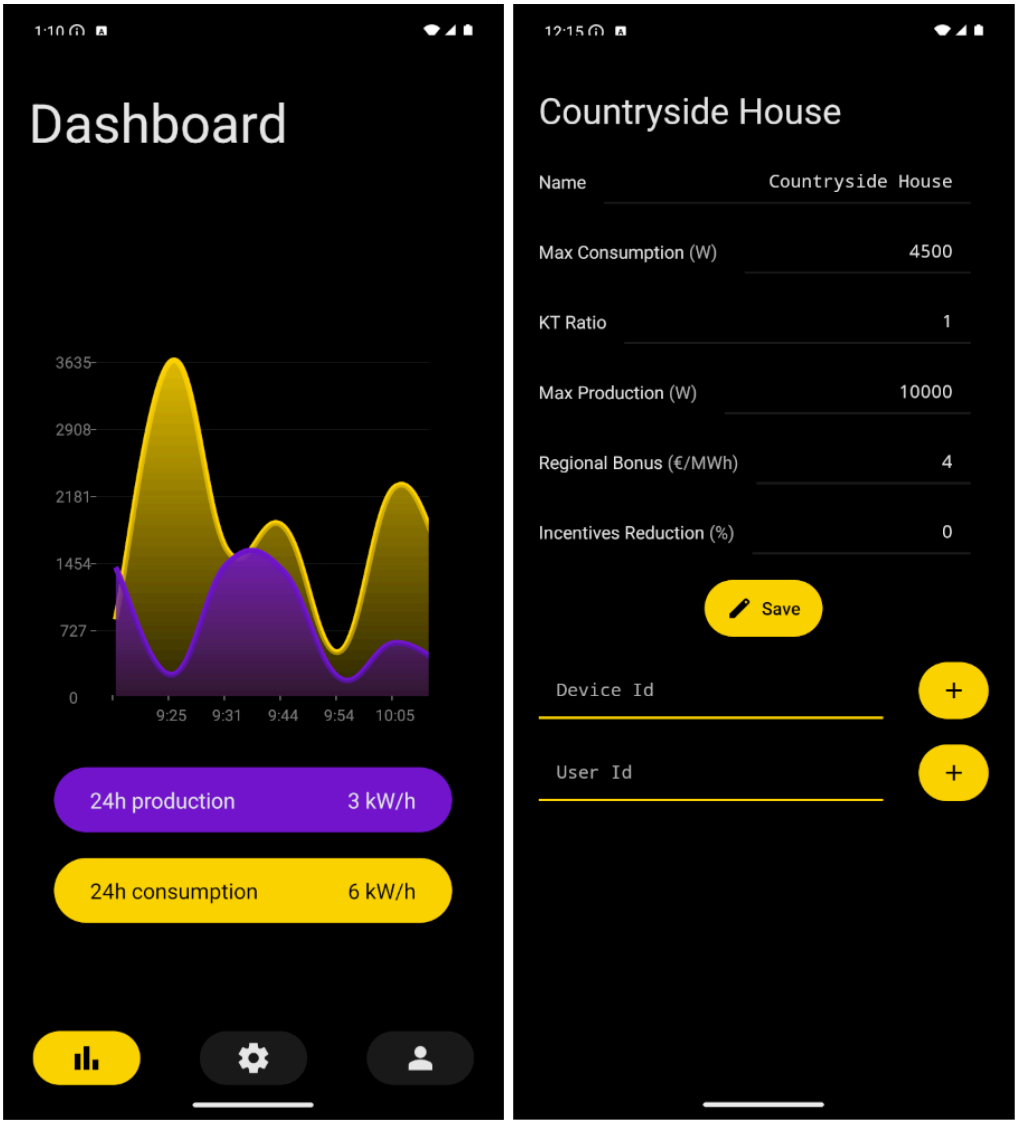
on the insights gained during the prototype development, an improved version that incorporates the Chain2 protocol is currently being worked on and will later undergo validation.

Instead, the implemented services, realized following a microservice architecture with a serverless approach, are the most comprehensive component of the system and might only need minimal adjustments to be prepared for a production environment. The designed services are built to be scalable and maintainable, with some initial tests conducted to assess the system's performance and reliability. However, further testing could prove extremely beneficial to fine-tune some configurations in order to ensure reliability under heavier load conditions.

A representative of a nearby REC was also consulted during the project, who expressed interest in the system and a wish to use it for a real-world REC monitoring scenario. In addition to validating the project's concepts, this meeting

provided feedback on the project's usefulness, which might be a valuable resource for the energy community's members. The system's compliance with the user's needs was judged by presenting the outcomes of the system and, specifically, the realized mobile application that a user would interact with, as seen in Figure 5.2.

Figure 2.8 partly reports upon some of the future developments for the system of broader scope in comparison to those that were just discussed. There definitely are better tools for analyzing consumption, which is one of the other features that may be interesting to include in a later version of the system. Other improvements could allow each user to monitor not just the production and consumption values but also to determine in real time the incentives that will be given out in the community, based on some criteria that the community representative can customize. To maximize community self-consumption, a second idea that might result in major advancements is the application of AI algorithms to forecast energy output and consumption with a high degree of accuracy.



(a) Screenshot of the app showing the dashboard

(b) Screenshot of the app showing the facility detail

Figure 5.2: Screenshots of the mobile application

Bibliography

[EGHS16] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016.

[Eva04] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[JC23] Ivar Jacobson and Alistair Cockburn. Use cases are essential: Use cases provide a proven method to capture and explain the requirements of a system in a concise and easily understood format. *Queue*, 21(5):66–86, November 2023.

[LHv20] J. Lowitzsch, C.E. Hoicka, and F.J. van Tulder. Renewable energy communities under the 2019 european clean energy package – governance model for the energy clusters of the future? *Renewable and Sustainable Energy Reviews*, 122:109489, 2020.

[MH12] Patrick Moriarty and Damon Honnery. What is the global potential for renewable energy? *Renewable and Sustainable Energy Reviews*, 16(1):244–252, 2012.

[Mor20] K. Morris. *Infrastructure as Code*. O'Reilly Media, 2020.

[MSV17] Alexander Maier, Andrew Sharp, and Yuriy Vagapov. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. In *2017 Internet Technologies and Applications (ITA)*, pages 143–148, 2017.

[Pat19] S. Patterson. *Learn AWS Serverless Computing: A beginner's guide to using AWS Lambda, Amazon API Gateway, and services from Amazon Web Services*. Packt Publishing, 2019.

[PPP+21] Adwait Pathak, Tejas Patil, Shubham Pawar, Piyush Raut, and Smita Khairnar. Secure authentication using zero knowledge proof. In *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pages 1–8, 2021.

[VGO+17] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, and Mery Lang. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS lambda architectures. *Serv. Oriented Comput. Appl.*, 11(2):233–247, 2017.

[Win22] Philip Winston. Time-series databases and amazon timestream. *IEEE Software*, 39(3):126–128, 2022.

Acknowledgements

I want to sincerely thank everyone who has helped me along this journey, supporting and encouraging me as I have grown into the person I am today. Although there are too many names to list everyone who has participated in this period of my life, it has been an immense privilege to be surrounded by so many wonderful people.

In particular, I want to express my appreciation to my family, which includes my father Aldo, mother Marinella, sister Sofia, and brother Lorenzo. They have always acted as a point of reference for me, supporting me in both my studies and personal life. They have provided me with everything I could have possibly asked for, offering me all the tools necessary to improve, grow, and fulfill my curiosity.

Additionally, I want to extend my gratitude to my girlfriend Nicole, who has been by my side for many years. She always had trust in me, and her love, patience, and support have been essential to me; helping me get through every obstacle I faced. I have shared countless moments of joy and happiness with her, always feeling loved and accepted for who I am, and knowing that I have a partner that I can always rely on.

I also had the great fortune to make plenty of awesome friends, who together make up the "Bewolla" group. They are genuinely amazing people, each with qualities and traits that make them unique and special. I will always be grateful to them for the moments we shared together, which I will always cherish.

Finally, I would like to thank my supervisor, Prof. Alessandro Ricci and my co-supervisors Ing. Andrea Diotallevi and Ing. Marco Diotallevi. Despite their numerous work commitments, they always found time to assist me, and their advice and support have been invaluable.