UMinho

Master's in Software Engineering
Information Technology Project

Group E07 - DTx
Maintenance Worker App (MWA)

Pedro Miguel
Castilho Martins
PG54146

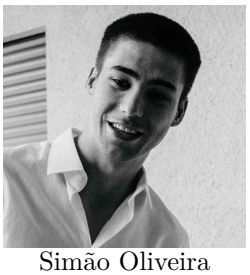
Ana Rita Santos
Poças
PG53645

João Bernardo
Teixeira Escudeiro
A96075

Orlando José da
Cunha Palmeira
PG54123

Miguel Ângelo Silva
Senra
PG54093

Miguel Silva
Pinto
PG54105



Simão Oliveira
Alvim Barroso
PG54236

Oriented By Manuel Alves and Ricardo Rodrigues from DTx

Contents

1 Purpose of the Project **2**
- 1.1 Summary . . . 2
- 1.2 Context . . . 2
  - 1.2.1 Definitions, acronyms, and abbreviations . . . 3
  - 1.2.2 Client, consumer and stakeholders . . . 3
- 1.3 Project Objectives . . . 5

2 Business Model **5**
- 2.1 Our Product . . . 5
- 2.2 Key Objective . . . 5
- 2.3 Customer Targets . . . 6
- 2.4 Customer Challenges . . . 6
- 2.5 Our Solution . . . 6
- 2.6 Our Value . . . 6
- 2.7 Our Pricing . . . 7
- 2.8 Our Message . . . 7
- 2.9 Go-to-Market Strategy . . . 7
- 2.10 Investment Required . . . 7
- 2.11 Growth Opportunity . . . 7

3 Requirements **7**
- 3.1 Functional Requirements . . . 8
- 3.2 Non Functional Requirements . . . 9

4 Product Design **9**
- 4.1 Domain Model . . . 9
- 4.2 Use Case Diagram . . . 10
- 4.3 BPMN Models . . . 11
- 4.4 State Machines . . . 12

5 System architecture **13**
- 5.1 Logical Database Model (Ticket Management) . . . 15
- 5.2 Real-Time Chat . . . 15
- 5.3 Notifications . . . 16
- 5.4 Frontend . . . 16

6 Conclusion and Future work **20**

A UI Mockups **21**

1. Purpose of the Project

1.1 Summary

This document results from the collaboration between the Collaborative Laboratory Association for Digital Transformation (DTx-CoLab) and the DST group, within the scope of the R2UT project, with the goal of developing a software solution that addresses the specific needs of modular construction. The platform, named R2UT, consists of an efficient and centralized management system focused on improving and optimizing the processes associated with modular construction, ensuring its effective implementation regardless of the available infrastructure and resources.

Integrated in R2UT is the Maintenance Worker App (MWA), a Progressive Web App (PWA) that is the central focus of this document, designed to simplify and modernize the way Maintenance Workers receive and manage their tasks. Through the MWA, Maintenance Workers can quickly and intuitively access all relevant information about their assigned tasks, such as the nature of tasks, required materials, deadlines, and the Maintenance Workers responsible for each activity. This system also allows Maintenance Workers to record detailed information about the work performed after completing tasks, such as the materials used and any difficulties found, promoting an integrated and comprehensive management of the maintenance process.

Throughout this document, all stages of MWA development will be detailed, from requirements definition, system modeling and architecture.

1.2 Context

In the past few years, there has been a growing integration of technology in all aspects of daily life. The growth of these types of technology connected to the Internet of Things (IoT) has led to an increasing need for specialists to repair such equipment. With the aim of optimizing the organization of these devices, many companies are offering alternatives to manage their smart devices, as seen with `Samsung` and its `SmartThings` application, which allows, for example, the washing machine to be started remotely or the air conditioning to be programmed according to the time of day.

Within the scope of the **R2UT** project, the aim is to create a set of systems that manage buildings with smart equipment, covering everything from customer management to a maintenance system. As part of this project, it is necessary to build the `Maintenance Worker App (MWA)`, for which we will be responsible and whose purpose is to support workers who repair equipment that requires such procedures. In this context of increasingly intelligent devices, the following challenges arise:

- **Equipments from various brands:** The heterogeneity of equipment, produced by a wide range of brands, results in devices with distinct characteristics and control actions, leading to additional complexity and consequently to time loss and inefficient management of these devices.
- **Difficulty in finding Maintenance Workers for repairs:** Given the increasing complexity of smart devices, more technical support is required, which sometimes demands various types of Maintenance Workers with different specializations.
- **Environmental sustainability:** By facilitating the standardization of the maintenance process of equipment, it helps to prolong its useful life and reduce consumerism.

The **R2UT** project aims to provide a comprehensive platform for managing and monitoring assets within a tenant-based infrastructure. In this section, we are going to showcase the

system and that our project is going to integrate, going into more detail about its entities and stakeholders.

1.2.1 Definitions, acronyms, and abbreviations

The following table outlines important terms, acronyms, and abbreviations that are relevant for understanding the overall structure and functionality of the system.

Term	Definition
PWA	A progressive web app (**PWA**) is an app that is built using web platform technologies but that provides a user experience like that of a platform-specific app.
Tenant	An organization aggregating a hierarchy of groups of **Assets**
Group of Assets	A collection of **Assets** and/or **Asset Groups**
Asset	• A building or a collection of buildings • Part of a building (room) • A collection of devices
Maintenance Worker	An individual, also referred to as a **Technician**, responsible for performing **tasks** related to the upkeep, repair, and efficient functioning of equipment or systems, as guided by assigned tickets and processes.
Issue	Represents a reported problem or concern within the system. Occasionally, the term **"Ticket"** is employed as a substitute for **"Issue"**.
IssueTask	Refers to a task that must be carried out by a **Maintenance Worker** and can be directly associated with an **Issue**. These tasks can range from routine inspections and preventative maintenance to reactive repairs and troubleshooting of equipment failures.
SAP	Refers to the service provided by DST that lists registered materials.
Device	A physical object equipped with sensors, actuators, or communication capabilities, enabling it to collect, process, or exchange data within an IoT network.
Chat	Stores conversations related to an **Issue**, including attached files.

Table 1.1: Definitions, acronyms, and abbreviations

1.2.2 Client, consumer and stakeholders

Throughout the development of this project, we identified the involved parts that will be active participants, including the client, consumers, and stakeholders. Each group has a clearly defined role and plays a critical part in ensuring the project's success.

- **Client:** Our client is the company **DTx-Colab**, which requires the development of an application to assist **Maintenance Workers** with equipment servicing.
- **Consumers:** Our product is designed for **Maintenance Workers** and it's a software tool to help organize tasks, list problems to be solved along with their locations, and optimize their time. This tool streamlines the maintenance process by providing features such as

listing materials, tracking time spent on tasks, and obtaining proof of service through the client's signature.

- **Other *stakeholders* on the R2UT context:**

  The following image represents the hierarchy of different stakeholders in a project, arranged by their level of permissions. It illustrates the roles and responsibilities of top-level administrators to final users, as well as their relationships and permission levels in the system. Bellow "Super Admin", all actors are bound to a Tenant. Different actors at the same level of permissions have different roles, so they do not have the exact same permissions.

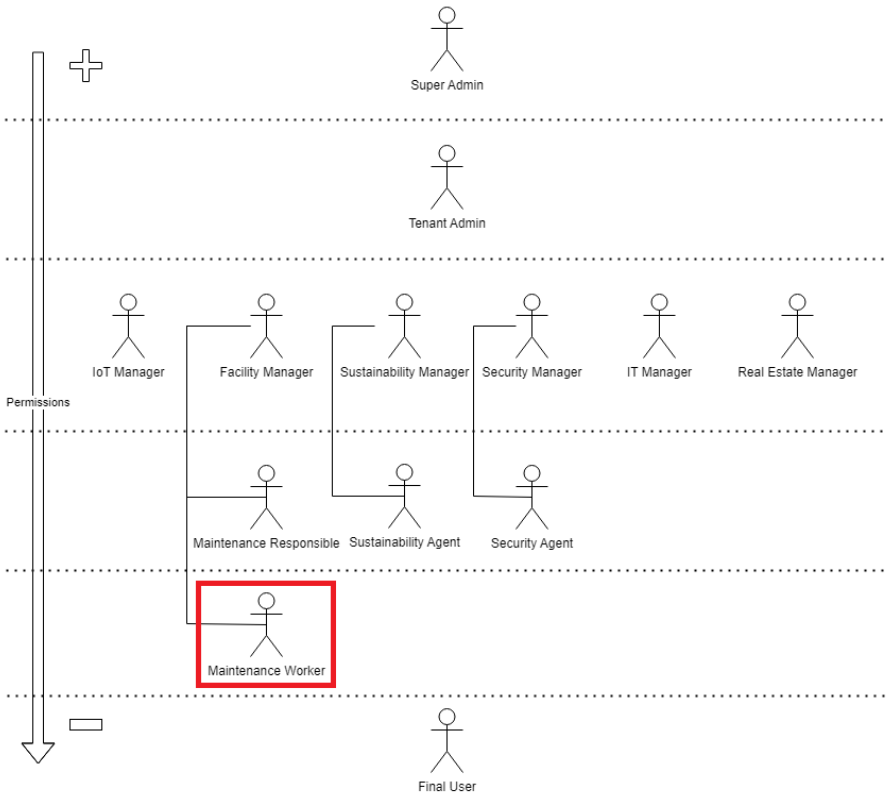


Figure 1.1: Actors hierarchy

  - **Super Admin**: Super user with access to all functionalities across all tenants.
  - **Tenant Admin**: Super user with access to all functionalities within a Tenant.
  - **IoT Manager**: Manages devices and IoT platform.
  - **Final User**: User of an asset. They are interested in a software tool that facilitates the maintenance work. They will interact with our system when they approve and acknowledge completion of the maintenance worker task when a client signature is required.
  - **Facility Manager**: The Facility Manager is responsible for approving tickets submitted to the system through our app, as well as other channels such as email or external systems. Additionally, the Facility Manager assigns tasks to Maintenance Workers and designates a Maintenance Responsible for each task, with the assignment process occurring on a separate platform.
  - **Maintenance Responsible**: The Maintenance Responsible can also create tasks and assign Maintenance Workers to them, with this process being carried out on a separate platform.

– **Maintenance Worker**: The Maintenance Worker is primarily responsible for executing a range of maintenance tasks across different asset groups, ensuring that each task is completed with its necessary requirements. This includes performing routine inspections, addressing repairs, and carrying out preventive maintenance to maintain the functionality of the devices.

It is important to emphasize that the application is intended for the **Maintenance Worker**, and for this reason, the remainder of this document is aimed at this actor.

1.3 Project Objectives

Within the project, this application is designed to streamline the repair processes for various types of equipment, serving as a valuable tool for maintenance workers. It aims to enable:

- **Standardization of the equipment maintenance process**: Ensure that, regardless of the Maintenance Worker's specialization, a consistent list of tasks is always presented for each *ticket*.
- **Providing visibility into assigned *Tickets***: Allow Maintenance Workers to view the *tickets* assigned to them, supporting task organization and prioritization.
- **Displaying tasks associated with a *Ticket***: Present a clear breakdown of tasks within each *ticket* that must be completed to resolve the *ticket* effectively.
- **Facilitating task management by Maintenance Workers**: Enable Maintenance Workers to update, edit, and mark tasks as completed, ensuring effective monitoring and progress tracking throughout the maintenance process.

2. Business Model

This section presents the strategic approach for delivering value with the Maintenance Worker App. It details how MWA addresses customer needs, supports sustainable maintenance practices, and creates revenue through targeted solutions. This section includes the app's core objectives, target customer profiles, pricing structure, and growth opportunities, providing a comprehensive overview of how MWA will operate and succeed in the market.

2.1 Our Product

The Maintenance Worker App is a Progressive Web App designed to simplify and modernize task management for maintenance teams, primarily in environments requiring consistent equipment upkeep. MWA optimizes the scheduling, tracking, and reporting of maintenance tasks, allowing workers to access task details, update status, and record work outcomes directly from a mobile device, promoting efficiency and traceability across all maintenance activities.

2.2 Key Objective

The primary objective is to provide enterprises with a centralized, efficient solution that facilitates the report of issues and the management of maintenance task, and with that improve operational productivity and sustainability by prolonging equipment lifespans through organized maintenance practices.

2.3 Customer Targets

Our app is targeted toward companies and organizations in industries such as:

- Facility management and real estate
- Manufacturing and industrial plants
- Hospitality and healthcare facilities
- Organizations with extensive IoT devices or smart infrastructures requiring regular maintenance

2.4 Customer Challenges

Customers face several challenges that make maintenance management complex and demanding. Some of this challenges are:

- **Diverse Equipment Brands:** Equipment from various manufacturers complicates the standardization of maintenance tasks.
- **Specialized Workforce Needs:** Complex and evolving smart technology requires specialized skills, making resource allocation challenging.
- **Sustainability Concerns:** Businesses need tools that support sustainable practices by extending the life of their assets and reducing waste.

2.5 Our Solution

MWA standardizes the workflow by enabling a uniform maintenance protocol regardless of equipment type or worker specialization. It offers:

- **Task Management:** Detailed tracking of tasks with the ability to log materials, track time, and record issues.
- **Communication Tools:** Integrated chat for real-time coordination among team members.
- **Comprehensive Reporting:** End-of-task reports, including materials used, time spent, and any encountered issues.
- **Sustainability Tracking:** Tools for capturing and analyzing data to promote sustainable practices in maintenance operations.

2.6 Our Value

MWA enhances productivity and optimizes resource usage by providing an intuitive and mobile-friendly interface for real-time task updates. This reduces downtime, improves asset management, and supports sustainability goals, positioning MWA as a cost-effective, scalable solution for enterprises.

2.7 Our Pricing

MWA is exclusively available to enterprise clients with customized pricing based on organizational needs. Our flexible model includes:

- **Custom Plans**: Options range from core maintenance tracking, comprehensive task management and analytics to machine learning models for unscheduled preventive maintenance and large language models for diagnostic assistance.
- **Volume Discounts**: Scalable pricing for clients with large facilities or workforces.
- **Long-Term Deals**: Reduced rates for extended partnerships, with options for priority support and training.

This approach ensures enterprises get maximum value aligned with their operational goals.

2.8 Our Message

"Streamline Your Maintenance Workflow. Improve Efficiency. Promote Sustainability." MWA is marketed as the go-to tool for businesses aiming to manage maintenance tasks with precision and support sustainable asset management.

2.9 Go-to-Market Strategy

- **Targeted Outreach**: Focused marketing to facility management, industrial operations, and smart-building industries.
- **Demonstrations**: Host virtual demos to showcase MWA's benefits and its alignment with sustainability goals.
- **Trial Offers**: Offer a free trial or pilot program to demonstrate value before full deployment.

2.10 Investment Required

An estimated initial investment of 500,000€ to cover development, marketing, and initial deployments, with additional funds allocated for scaling, including cloud hosting and data security infrastructure.

2.11 Growth Opportunity

With the global growth in IoT-enabled devices and smart infrastructure, MWA has potential to expand by adding integrations for predictive maintenance through IoT analytics, enhanced AI-driven support, and customizable reporting tailored to specific industries.

3. Requirements

This chapter provides an overview of the essential requirements for the project's development, defined in collaboration with the client and other stakeholders to ensure the system meets all functional and non-functional needs.

3.1 Functional Requirements

Requirement	Description
FR1	The System must have an authentication mechanism.
FR2	The User must be able to access the list of Tickets associated to him, where he has Tasks.
FR3	The System must display the Tickets to the User with a color coding that reflects their priority level.
FR4	Each Ticket contains at least one Task.
FR5	After the User selects a Ticket, the System must list the Tasks associated with the Ticket, ordered by their due date in ascending order (earliest first).
FR6	The User must be able to create a request for a Ticket in the System.
FR7	When creating a Ticket request, the User must be able to associate devices and photos relevant to the Ticket.
FR8	The User can view all Tasks associated with a Ticket, but can only edit the Tasks assigned to them.
FR9	The System must provide filters to allow Users to view only their Tasks or all Tasks, as well as to filter by open or closed Tasks.
FR10	A Task consists in a Facility Manager description, Maintenance Worker description, time slots, list of materials, list of devices, and optionally, photos.
FR11	The User must be able to view the descriptions, time slots, list of materials, list of devices, photos, start/stop the timer, and add or remove time slots.
FR12	The User must be able to edit the task's Maintenance Worker description.
FR13	The system must enable users to manage materials by providing features for selecting predefined materials from a service (SAP), manually adding unlisted materials, and editing associated materials for tasks.
FR14	The System must present a list of available devices in the Issue's Asset Group, provided by the IoT platform, allowing the User to select devices to associate with a Task and edit the devices associated with their Tasks.
FR15	The User must be able to mark their Tasks as completed.
FR16	The System must define and support the following possible states for Tasks: Pending, Assigned, In Progress, Pending Signatures, Resolution Validation, Failed Closed, and Success Closed.
FR17	Before completing a Task, the User must provide a description of the problem resolution.
FR18	The user shall not be allowed to edit a task once it has been marked as "closed".
FR19	Tasks that require a client signature must be signed by the client after the User completes and closes the Task.
FR20	A User assigned to a Ticket's Tasks must have the capability to communicate with all users who have access to that ticket through the chat feature.
FR21	The User must be able to upload and share files in the chat associated with a Ticket.
FR22	The User must be able to receive notifications when he is assigned to a Task.

Table 3.1: Functional requirements

3.2 Non Functional Requirements

Requirement	Description
NFR1	The MWA must be developed as a cross-platform app using PWA.
NFR2	The System's frontend must be developed using the Nuxt.js framework built on Vue.js.
NFR3	The System's database must be built using PostgreSQL.
NFR4	The System's logic must be developed using .NET 8.0.
NFR5	The application must be available 99% of the time.
NFR6	The System architecture must follow clean code architecture principles.
NFR7	The System must have an interface allowing for easy and intuitive navigation across all functionalities.

Table 3.2: Non Functional Requirements

4. Product Design

4.1 Domain Model

The Domain Model presents the main relationships and entities of the problem. This model is useful as it serves as a starting point for the system's design.

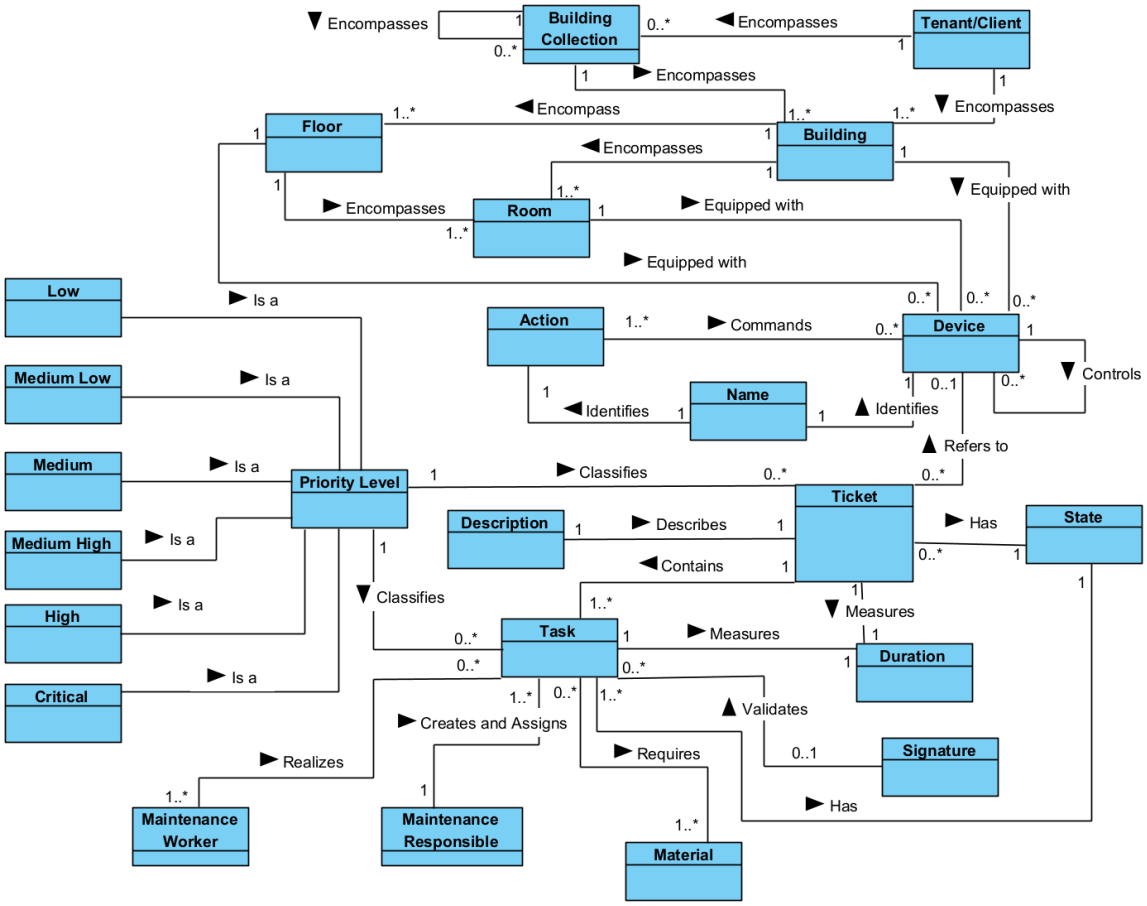


Figure 4.1: Domain Model

At the core of the model, there are two important entities: **Devices** and **Tickets**. Although tickets are often created to address device-related issues, they can also be issued independently to report anomalies that are not associated with any device. For example, a Ticket could be created to report an issue like "infiltration in the main corridor of Building C."

The devices in the system are distributed across various buildings which are part of larger collections. Each building may be subdivided into floors and rooms, providing an organized and hierarchical structure as required by the client. Instead of using specific entities for these spaces, they have been abstracted into a single entity called the **Asset Group**.

When a problem arises, a Ticket is issued to document the need for intervention, whether on a device or not. Tickets are classified according to a priority level, ranging from "Low" to "Critical", and may also include a detailed description of the issue, providing context for the problem. Based on the nature of the problem, specific tasks are then created to resolve it. These tasks outline the concrete actions necessary to resolve the problem and can involve mobilizing maintenance workers and materials.

Each Task is measured in terms of its required duration (estimated time required for completion) and may require a client's signature to validate the Task completion. The system tracks the status of each Ticket and Task through various stages, including "Assigned", "In Progress", "Pending Signatures", "Resolution Validation", "Success Closed" and "Failed Closed" ensuring that every step of the workflow is rigorously monitored.

4.2 Use Case Diagram

To better understand the context of the system, a use case diagram is presented in Figure 4.2. In this diagram, some of the main functionalities of the system, as well as its actors, will be detailed. It is also possible to identify the functionalities that each system actor will have access to.

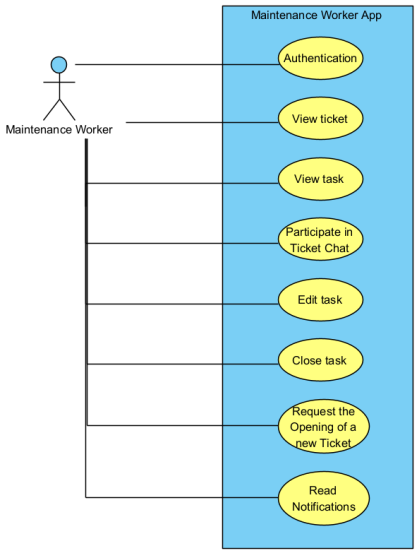


Figure 4.2: Use Case Diagram.

As it is mentioned in the Figure 4.2, the worker can perform authentication, which grants access to the app. After logging in, the worker can view the tickets, allowing him to see his assigned issues. In addition, the worker can view tasks related to these tickets. There is also an option to participate in an issue chat, enabling communication with other workers assigned to the issues' tasks to discuss and resolve its problems. The worker can also edit tasks, update task details, such as photos, materials and devices related to the task. Finally, once a task is

completed, the worker can close the task, marking it as resolved. If the Maintenance Worker wants to report a problem, he has the option of requesting a new Ticket, where he mentions its details and sends the request to the Facility Manager for its approval.

4.3 BPMN Models

At the beginning of the project, we considered using sequence diagrams as part of the development documentation. However, during the requirements gathering process, the company for which we are carrying out the project expressed a clear preference for using **BPMN** (Business Process Model and Notation), as they are more suitable for their work methodology and internal processes.

The following model describes the authentication process for a maintenance worker in a multi-tenancy system. In this context, clients will be able to store their own data, allowing access to be performed through an external authentication provider.

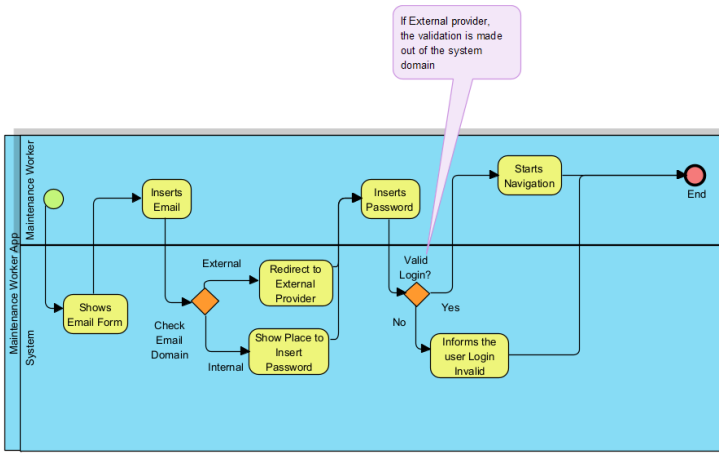


Figure 4.3: Login BPMN Diagram.

The following BPMN diagram illustrates the process of navigating through the app. Initially, the app verifies the user's authentication status. If the user is not authenticated, they are prompted to enter their credentials; upon successful validation, access to the app is granted. From this point, the user can select an issue, view the tasks associated with it, check notifications, or access the menu. Only one of these actions can be performed at a time.

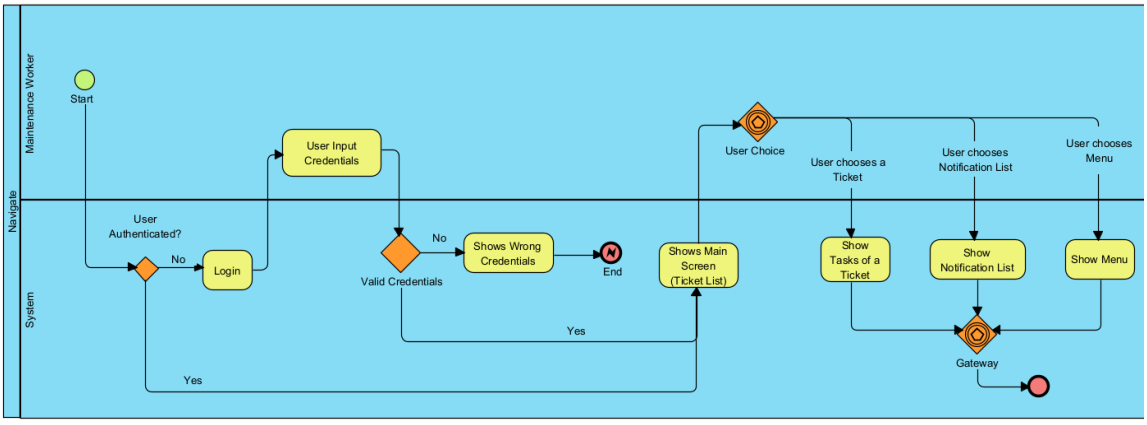


Figure 4.4: Navigate BPMN Diagram.

Figure 4.5 illustrates the typical workflow followed by a technician after successful authentication. Upon logging in, the user is presented with a list of tickets containing tasks assigned to them. By selecting a specific ticket, the user can view related tasks or engage in a chat with other collaborators associated with the ticket. The Maintenance Worker is restricted to editing only the tasks specifically assigned to them. For tasks not assigned to the user, but within an issue they are associated with, the user is granted view-only access. After selecting an accessible task, the user can update task details, start or stop the timer, and access additional task information. Finally, the user can close the task, documenting essential details such as materials used and obtaining any necessary signatures before completing the process.

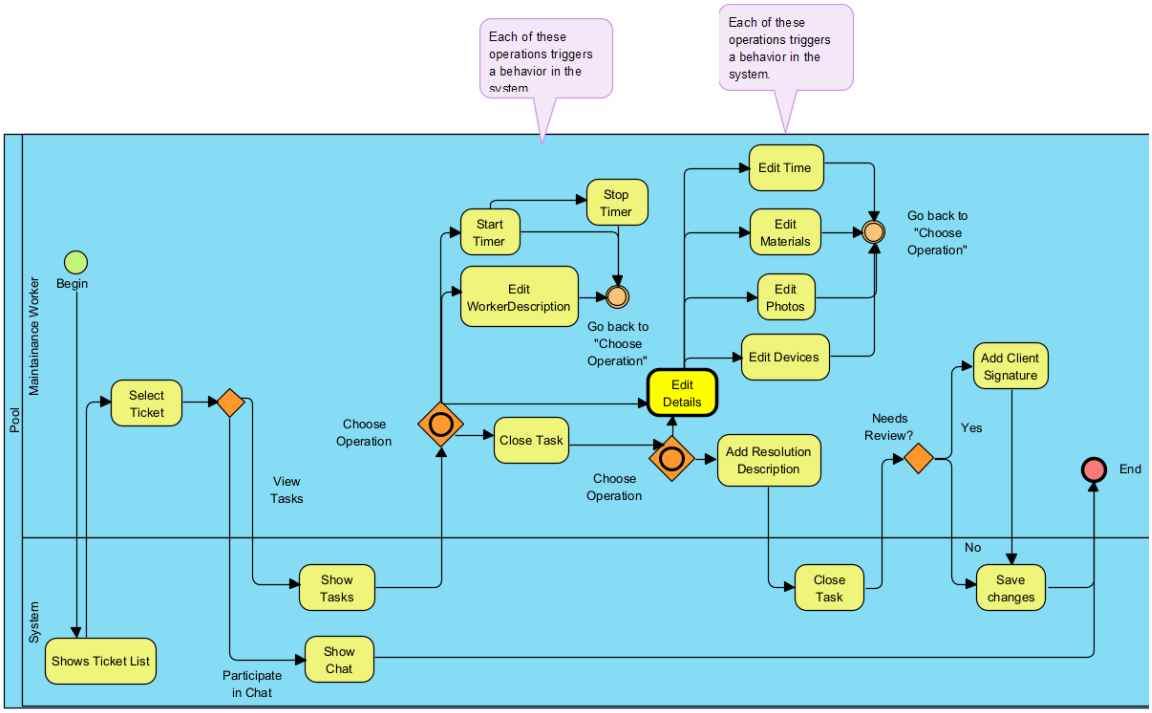


Figure 4.5: Normal Application Flow BPMN Diagram.

4.4 State Machines

The **Ticket** is the main entity to be addressed in this project, moving through various states throughout its life cycle. To illustrate this process, we have created the following diagram, which outlines each stage and its corresponding conditions.

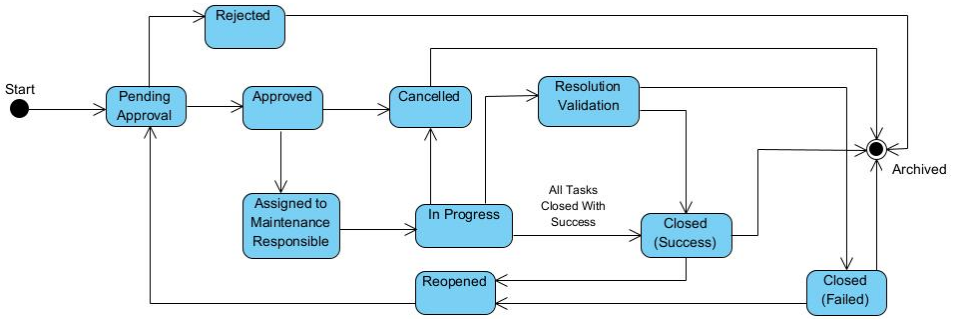


Figure 4.6: State machine for a Ticket.

At the start of an issue, the Ticket is in a "*pending approval*" state, which happens when

a worker submits a new issue request in our app. The Facility Manager then decides whether to approve or reject the Ticket. If approved, the Facility Manager assigns a Maintenance Responsible, moving the issue to the "*Assigned*" state. The issue moves to the "*In Progress*" state when one of the tasks is marked as "*In Progress*". The "*Resolution Validation*" status refers to a Ticket in which all tasks are closed. Once a Ticket enters this state, it is no longer listed in our app, as all associated tasks have been completed and no further attention from maintenance workers is required. In this state it is up to the Facility Manager to decide whether the Ticket is ultimately marked as "*Closed Successfully*" or "*Closed Failed*".

The following diagram describes the life cycle of a **Task**.

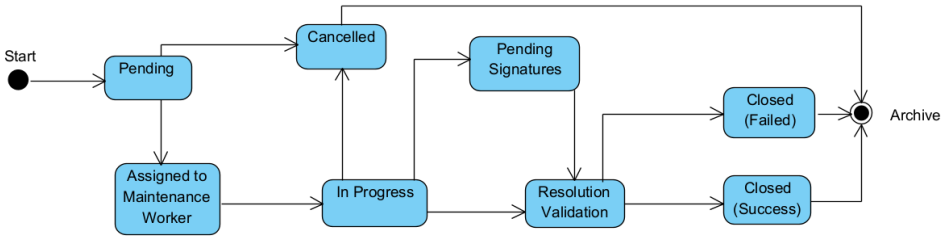


Figure 4.7: State machine for a Task.

The task lifecycle begins in the *"Pending"* state when created. When the Facility Manager or the Maintenance Responsible assigns a worker to the task, it transitions to the *"Assigned"* state. Once the worker enters a work time slot into the task, it moves to the *"In Progress"* state. At this point, the task can be closed, which advances it to the *"Resolution Validation"* state. However, if the task requires a client signature, closing it from the *"In Progress"* state moves it to the *"Pending Signatures"* state. The task will only transition to *"Resolution Validation"* after the client signature is provided.

In both the *"Pending Signatures"* and *"Resolution Validation"* states, the worker is no longer able to modify task details. Task approval or rejection is the sole responsibility of the Facility Manager or the Maintenance Responsible.

5. System architecture

The **R2UT** project employs a microservices architecture in which each service is responsible for a specific functionality. In the figure below, we can see the different services and how they interact with each other.

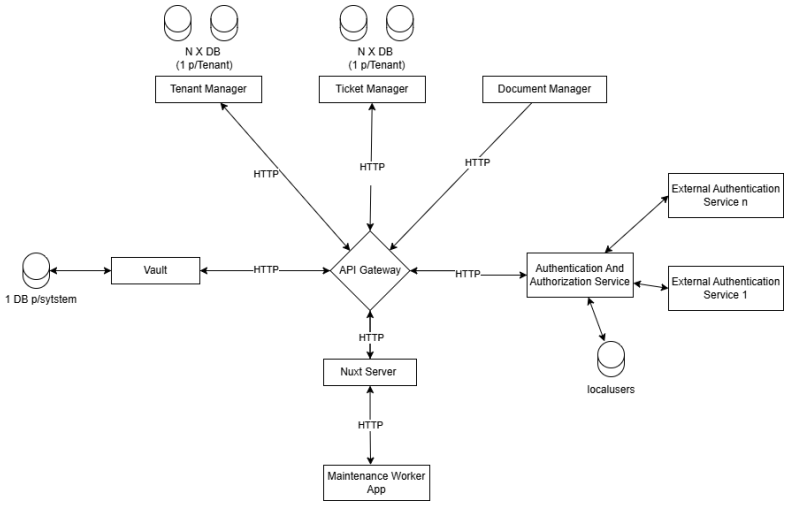


Figure 5.1: Micro-Services Architecture

As we can see in the figure above, the diagram illustrates a high-level architecture of our system with several components that interact with each other via HTTP requests.

- **Authentication and Authorization Service** - This service is responsible for managing user authentication and authorization. It integrates with an API Gateway via HTTP to authenticate and authorize incoming requests. The service supports local users, representing internal users stored within the system, as well as federated authentication through external service providers. Federated authentication allows users to authenticate using credentials managed by trusted external identity providers (e.g., Google, Microsoft), enabling seamless access without requiring separate credentials for each system. This functionality is implemented using **Keycloak**, a widely-used identity and access management solution. **Keycloak** was configured and is maintained by another team within the company, and our team solely consumes its endpoints to facilitate the authentication and authorization processes.

- **Tenant Manager** - The Tenant Manager microservice is responsible for managing Tenant-related information, including Asset Groups, Device Types, and Tags. In our use case, this service is exclusively utilized to obtain a list of tenants and their associated Asset Group structure. The service was developed and is maintained by another team within the company, and our team solely consumes its endpoints to access the required data.

- **Ticket Management** - This service handles all aspects of a Ticket and its associated Tasks, from creation to completion. For each tenant, a dedicated database is maintained, ensuring tenant-specific isolation of ticket and task data. The logic for selecting the appropriate database, to communicate with, is managed automatically by the Entity Framework repository. It communicates with its database to manage ticket and task data and utilizes the API Gateway via HTTP for authentication and authorization of ticket-related requests.

- **Document Manager** - The Document Manager service enables users to upload, download, and manage documents. It acts as a gateway to the SeaweedFS storage system, supporting operations like listing, retrieving, and deleting documents by name. Built using the Ocelot package, the service handles request routing, authentication, rate limiting, and caching. The Document Manager service is developed and maintained by a separate team within the company, and our application solely consumes the endpoints provided by this service to manage files in Tickets, Tasks and Chats.

- **Nuxt Server** - This is a front-end server responsible for delivering the user interface (UI)

for web clients. Communicates with the API Gateway via HTTP to handle requests from the client-side (browser or app) and interacts with other backend services as needed.

- **Maintenance Worker App** - Communicates with the Nuxt Server via HTTP to manage and handle maintenance tasks.
- **Vault** - It stores secrets and credentials related to our app. Stores the required "secrets" to configure and run the application.

5.1 Logical Database Model (Ticket Management)

This logical database model was designed to structure the data in a manner that supports the functional requirements of our application. This diagram provides a detailed view of the relationships, attributes, and entities used in the Ticket Management system.

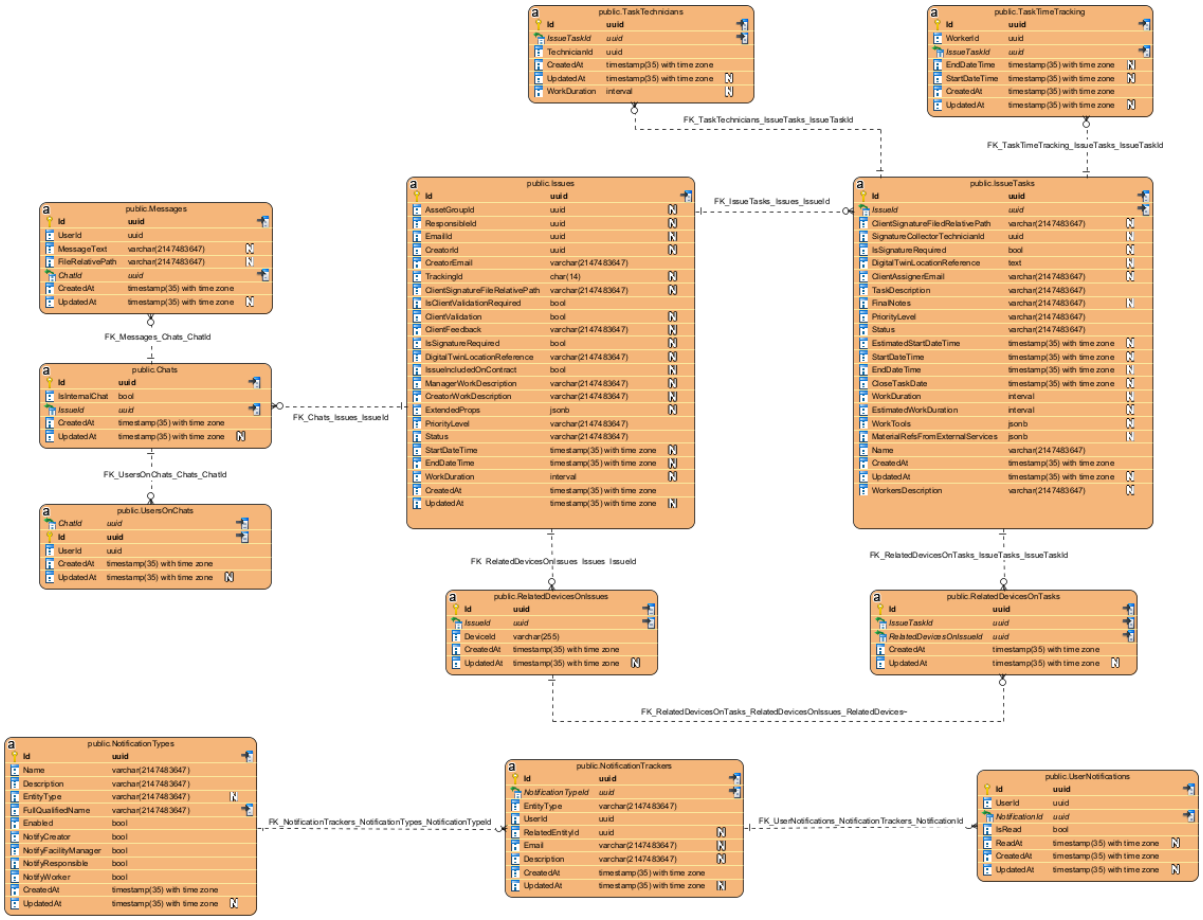


Figure 5.2: Logical database model

5.2 Real-Time Chat

As outlined by the company during the requirements elicitation phase, the development of a real-time chat feature was identified as a key need. This chat serves as the primary communication channel between stakeholders involved in a ticket, including the facility manager. It is designed to address scenarios where technicians require fast responses from colleagues or the facility manager to resolve issues efficiently.

The chat functionality was implemented using an observer pattern. Upon subscribing to the chat hub as we can see in Figure 5.3, users receive real-time updates for new messages. This ensures an instant and dynamic communication experience.

To achieve this real-time capability, Microsoft SignalR was used. SignalR is a powerful library that enables real-time web interactions by establishing seamless communication between the server and client. It optimizes performance through WebSockets where possible, while providing fallback mechanisms such as long polling to ensure robust functionality in diverse connection environments. This integration eliminates the need for the client to poll the server repeatedly, enabling instantaneous updates with minimal latency.

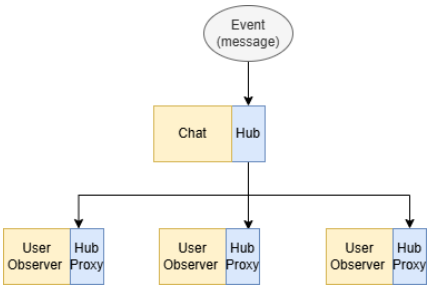


Figure 5.3: Chat Diagram

The chat also supports file and photo sharing, allowing users to attach and send a single file per message. These shared files are securely stored on a SeaweedFS server, ensuring efficient file management and retrieval. Additionally, the chat interface provides comprehensive information about all participants involved in the conversation.

5.3 Notifications

Similarly to the chat implementation, the notifications system operates using the same underlying mechanism. After logging in, the user subscribes to the notification service, which functions as a hub.

Through this subscription, the user receives real-time updates about relevant events. This implementation also relies on the observer pattern, where the user acts as an observer. Whenever a new notification is generated, it is sent to all subscribed users instantly, ensuring they remain informed about critical updates without the need for manual refreshes or repeated requests to the server.

5.4 Frontend

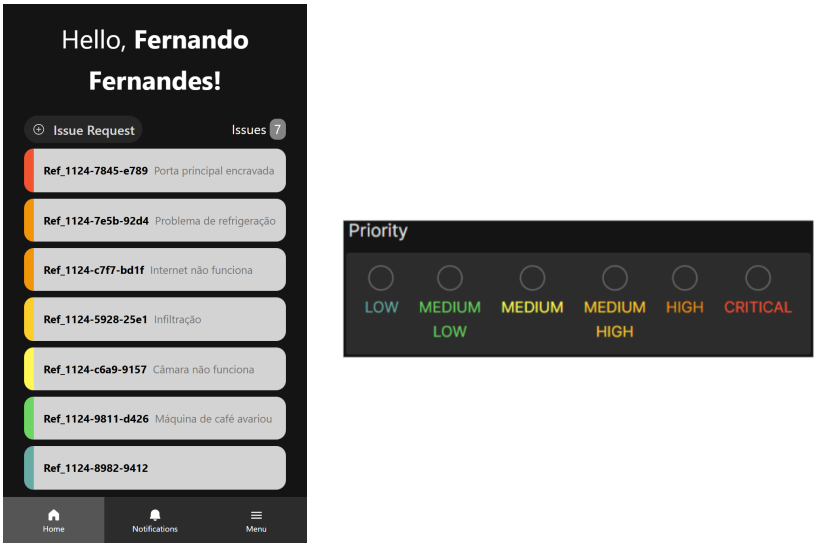
The frontend of the MWA application was implemented using Nuxt, a Vue framework, and is made up of two components: the client application and the Nuxt server.

The Nuxt server is responsible for providing the graphical interface pages to the client application and acts as an intermediary (a *backend for frontend*) that performs the necessary processing to coordinate communication between the client application and the system backend (the API Gateway).

As for the client application, it will simply consume the endpoints provided by the Nuxt server and is responsible for presenting data, information, and functionalities to the final user, the maintenance worker. The design and style of the app pages were implemented with Naive UI, a Vue component library.

Now, we present the application's user interface (UI), illustrating the design and visual organization of the main features and pages. The pages that are going to be presented are based on the mockups of the appendix A.

Upon logging in, the maintenance worker will first interact with the tickets page. On this page, the user can request the creation of a ticket by clicking the "Issue request" button or access a specific ticket by selecting one of the cards in the list. Each ticket card includes a colored strip indicating the ticket's priority, and the ticket list is sorted by priority, with the highest-priority tickets displayed first. The names presented in each ticket is a Tracking Id that DTx requested to be shown in this page.



(a) Tickets page (b) Priority color range

Figure 5.4: Tickets page

When the user intends to report an issue, they will request the creation of a Ticket that will be performed on the following page. On the figure 5.5, the three steps required to create a ticket are presented.

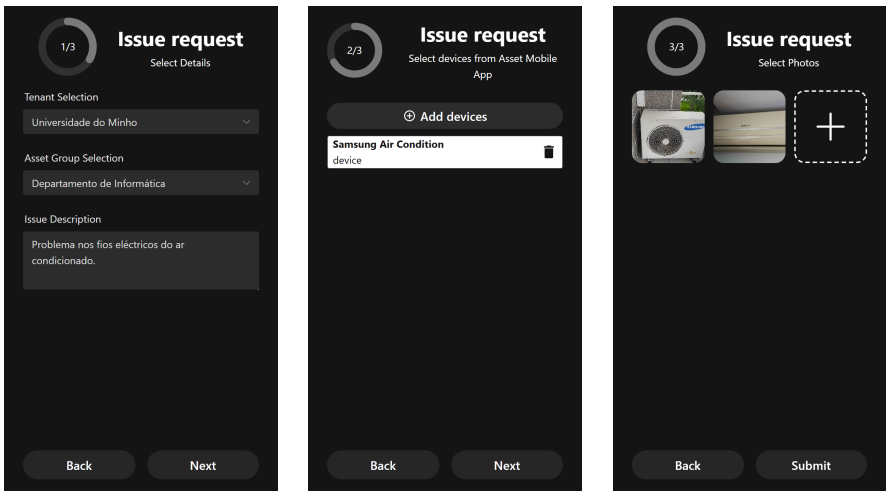


Figure 5.5: Issue request page

If the user needs to view a ticket and the tasks within it, the application provides the pages in the next figure.

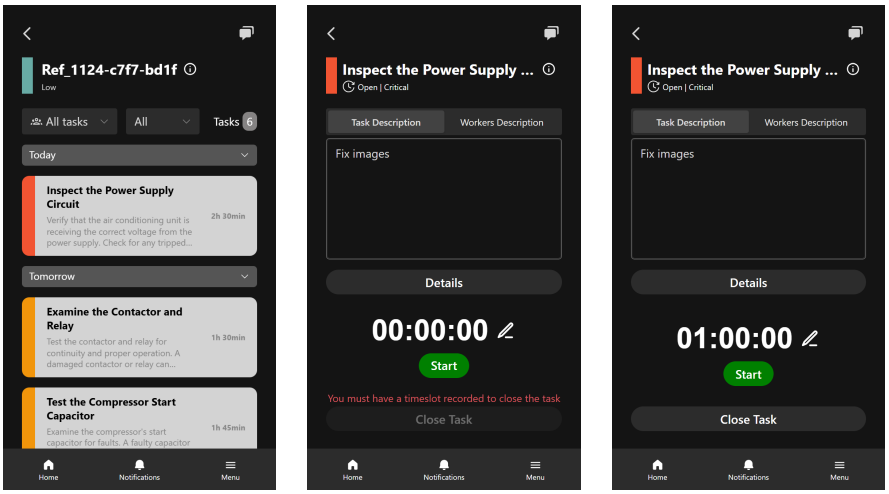


Figure 5.6: Issue page and Task page

The first page displays the details of a ticket, and the second page (two images in the right) shows what the user will see when he selects one of the ticket's tasks. As in Figure 5.4, the colored strips indicate priorities.

The two images to the right of the figure above display two tabs: "Task Description" which contains the description provided by the Facility Manager, and "Your Description" which contains the description provided by the Maintenance Worker, in case the issue was actually something else. Additionally, the "Close Task" button is only available once the user has initiated the task by registering work time slots, given the task can only be marked as resolved after being in progress.

The user will have the ability to modify certain details of their Task. For this purpose, the application will include the following page, where the user can update these details (photos, materials, and devices associated with the task).

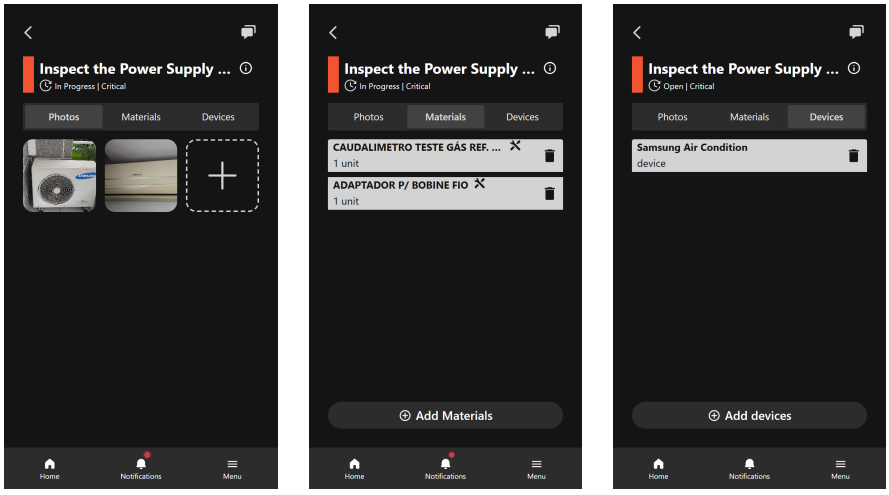


Figure 5.7: Task Details page

On the page above, the user can click on the "ℹ" icon to view additional details about the task (description, estimated start date, estimated end date, etc.).

To close a task, the user clicks on the "Close Task" button in the Figure 5.6, and then is redirected to the following page:

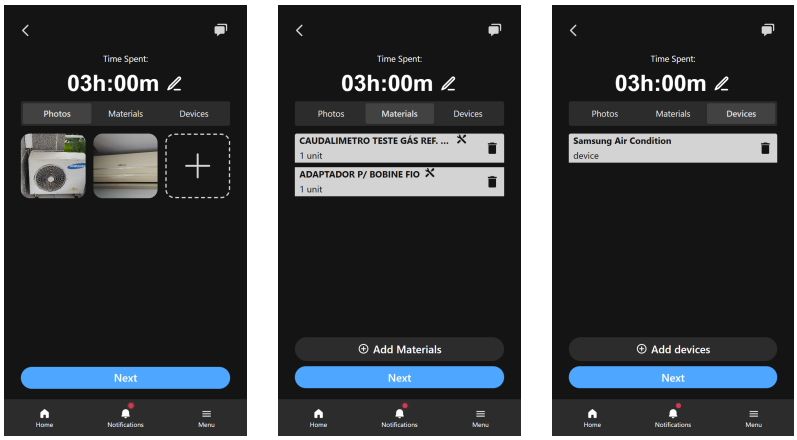


Figure 5.8: Close Task - Check details

As shown in the figure above, the user still has the opportunity to review and modify the task details before marking it as completed. After this step, the user will also need to describe how the task's problem was resolved and, if required, collect the signature of the client who requested the task execution.

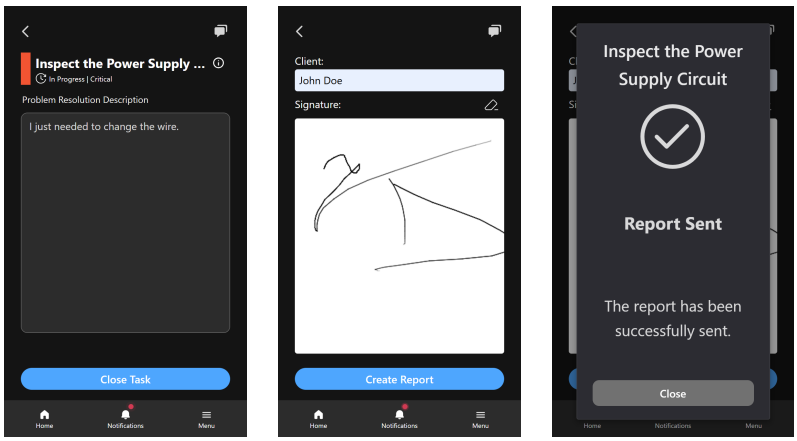


Figure 5.9: Close Task - Describe problem and Get signature

In the application, the user can also access notifications about the tickets they are working on, and for each ticket, they will have access to a chat to communicate with other users within that ticket. The notification and chat pages are shown below:

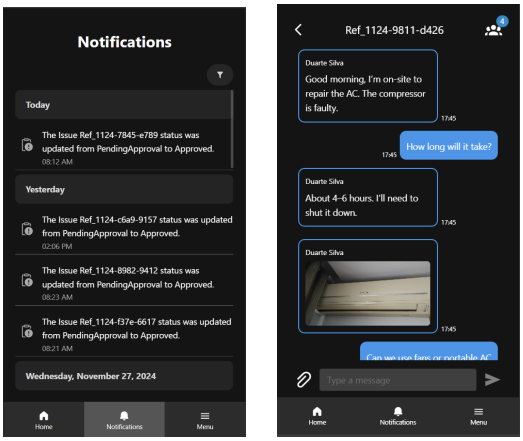


Figure 5.10: Notifications and Chat

6. Conclusion and Future work

In conclusion, we believe that the objectives set for this project have been successfully accomplished. The final product successfully meets the goals outlined at the project's inception, specifically aiming to streamline task management for maintenance workers within the IoT-enabled modular construction platform. By building a Progressive Web App (PWA), we have created a solution that enables workers to efficiently access task-related data, manage workflows, and address operational challenges in real-time.

This application not only meets the immediate needs of the R2UT project but also lays the groundwork for further enhancements in maintenance management. Moving forward, we plan to implement several enhancements that will extend the app's functionality and usability. Key features to be incorporated include:

1. **Task Assignment for Maintenance Workers:** We aim to integrate functionality currently available in another system within the platform, enabling maintenance workers to assign and allocate tasks directly through the app. This enhancement will improve coordination within teams and streamline task management.

2. **Support for Videos:** In addition to photo uploads, we will introduce the ability to attach videos as another media type. This will apply to both the chat feature and the task Photos section, providing users with a wider array of media for documentation and communication.

3. **Photo Editing Capabilities:** Future iterations of the app will allow users to edit photos during task execution. This feature will enable users to modify and annotate images after they have been uploaded, enhancing flexibility in managing task-related media.

4. **Enhanced Signature Screen:** We will refine the signature interface to allow users to sign over the entire mobile screen area, providing more flexibility and improving the overall user experience.

5. **Colorblind Mode for Task Prioritization:** A colorblind mode will be introduced to assist users who rely on alternative visual indicators to differentiate task priorities. This enhancement will reduce reliance on color coding and ensure accessibility for all users, regardless of visual impairments.

These planned features will further optimize the platform's functionality, enhance user interaction, and ensure greater accessibility across diverse user needs. As we move forward, we will continue collaborating with the client to integrate these features into the upcoming version 2.0 of the app.

A. UI Mockups

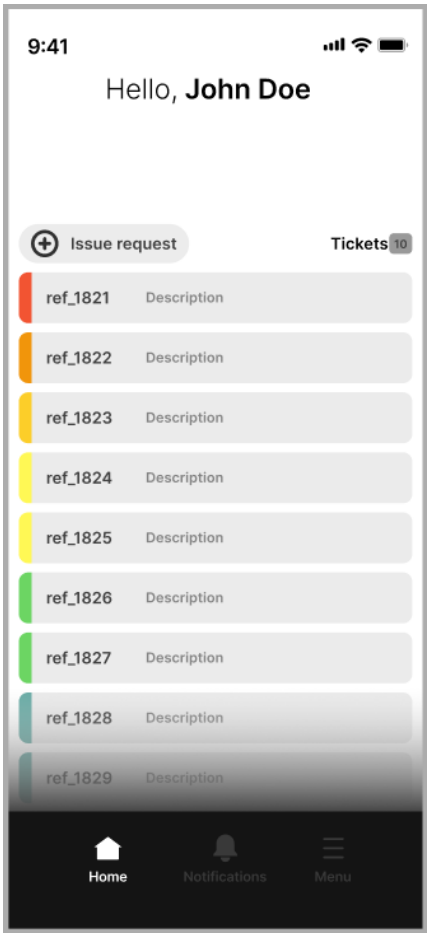


Figure A.1: Tickets page

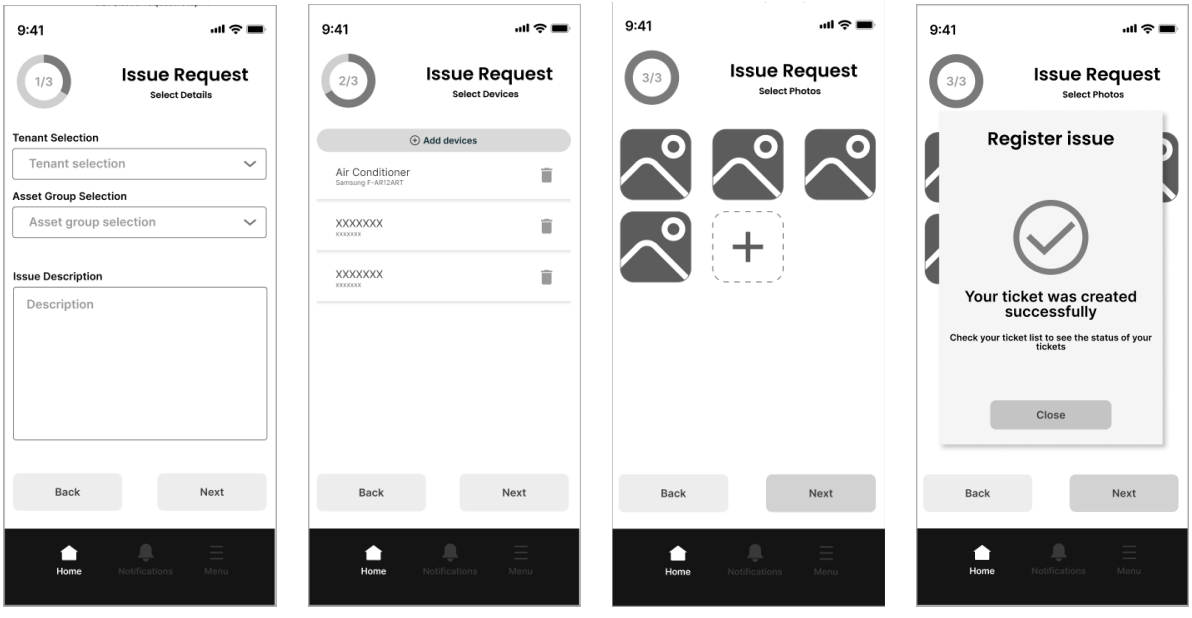


Figure A.2: Issue request page

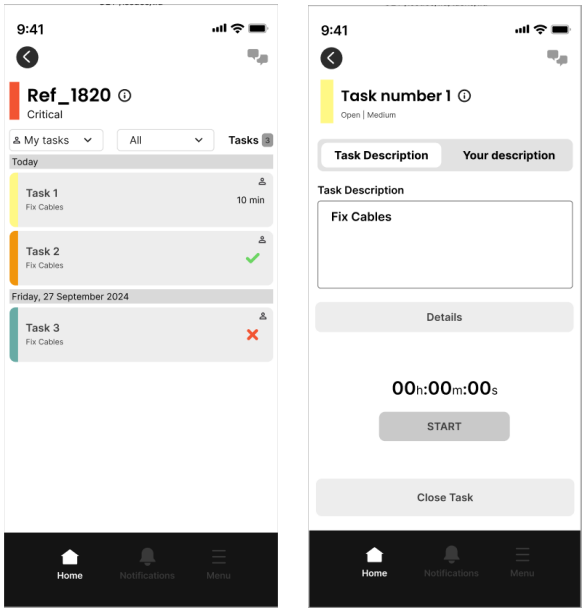


Figure A.3: Issue page and task page

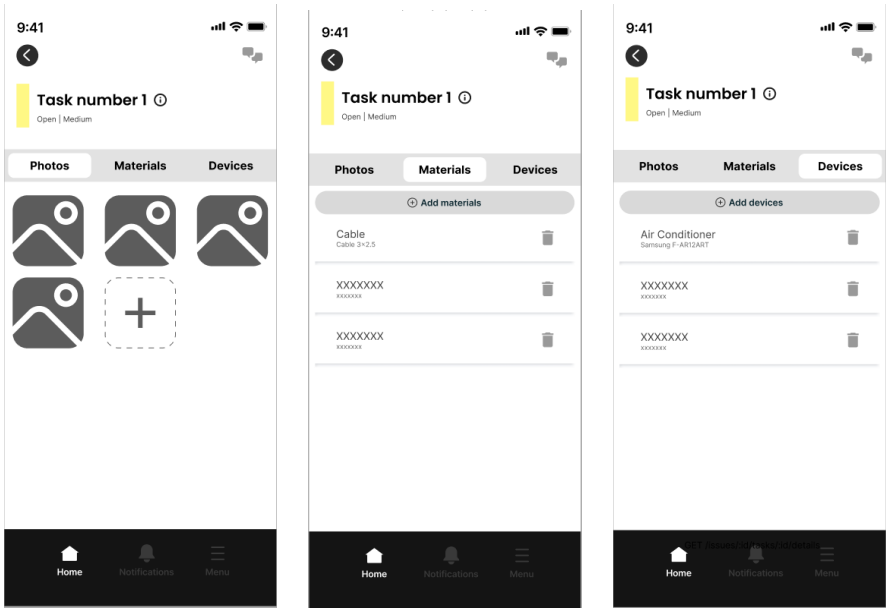


Figure A.4: Task details page

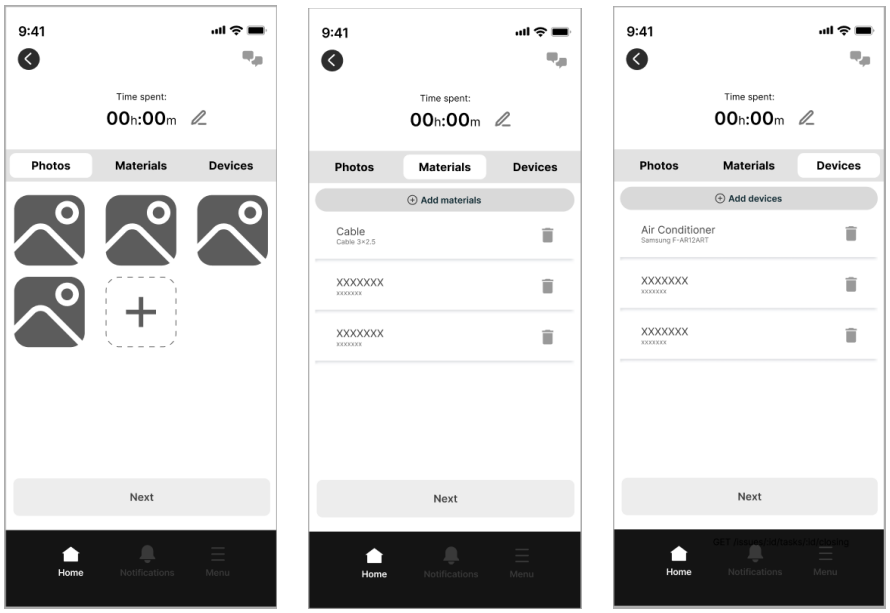


Figure A.5: Close Task - Check details

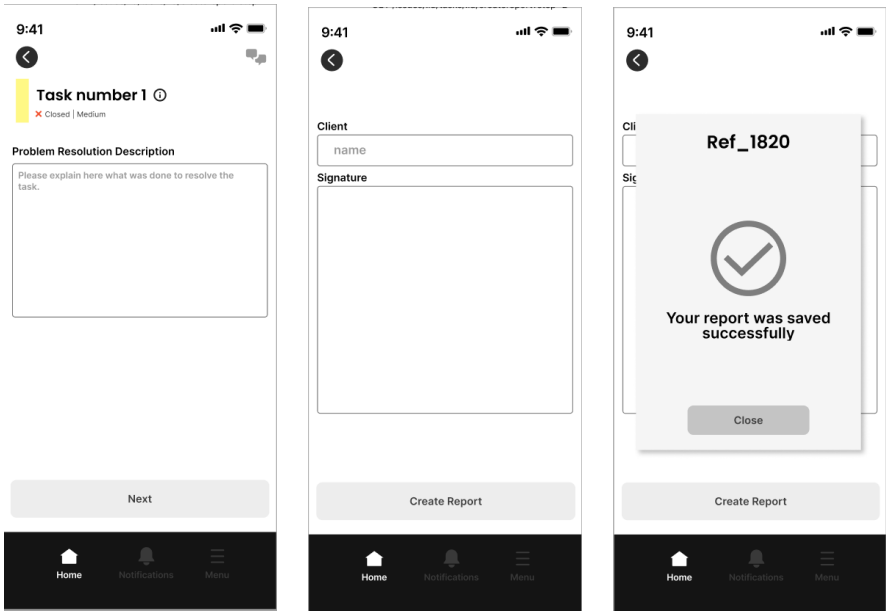


Figure A.6: Close Task - Describe problem and get signature

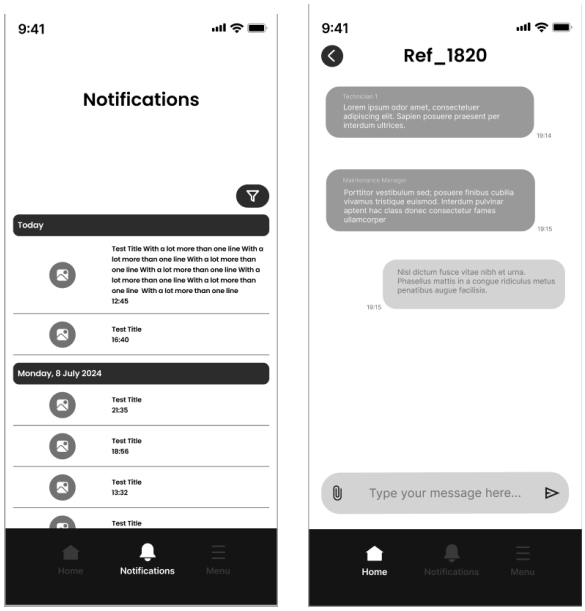


Figure A.7: Notifications and chat