Internship Report

Instituto Superior de Engenharia de Coimbra

Julien Bouin

2024-05-31

Contents

Chapter 1

1.1 Abstract

The completion of a three-month internship at the Instituto Superior de Engenharia de Coimbra (ISEC) in Portugal, as part of the Erasmus program, has been a rewarding and formative experience for me. Collaborating with my colleague Rayane Belguebli and under the guidance of our supervisor, Mateus Mendes, our mission was to develop a full-stack website with tight integration between the backend and frontend. This project, based on Django, aims to connect a Computerized Maintenance Management System (CMMS) to an Augmented Reality (AR) interface, thereby facilitating maintenance operations management. My primary role was focused on frontend development of the application.

Additionally, I also contributed to the development of mobile applications enabling maintenance technicians to update their interventions directly from their phones, even in the absence of Wi-Fi connectivity.

This internship represents a unique opportunity to explore the synergies between CMMS and AR technologies while deepening my understanding of frontend development. By concentrating my efforts on the frontend, I aimed to bridge the gap between backend infrastructure and user interfaces. Contributing to a tangible project, as described in the referenced article, has bolstered my motivation by emphasizing the relevance and significance of our work.

1.2 List of Figures

1.3 List of Acronyms

- **AR**: Augmented Reality
- **ORM**: Object-Relational Mapping
- **MVT**: Model-View-Template
- **CMMS**: Computerized Maintenance Management Systems
- **API**: Application Programming Interface

Chapter 2

Introduction

At the end of my two years at IUT A of the University of Lille, I undertook a three-month internship to validate my diploma. I chose to do this internship abroad through the Erasmus program, which led me to Instituto Superior de Engenharia de Coimbra (ISEC) in Portugal. I completed this project with my colleague Rayane Belguebli, and we were supervised by our tutor, Mateus Mendes.

We were tasked with developing a full-stack website requiring close integration between the backend and frontend. Our application, developed with Django, will be connected to a Computer-Aided Maintenance Management System (CMMS) to effectively manage maintenance operations. The frontend, also built with Django, will serve augmented reality (AR), an immersive technology that enhances user experience by providing contextual information in real time. Our project is directly related to the work presented in the article **(Ana, Mateus, and Torres 2021)** [**1**]. Our goal is to connect the CMMS to AR, so that technicians can directly see what they need to do on our site and then visualize it through AR; therefore, we will be responsible for the server and the database. My role primarily focuses on frontend development.

Additionally, I also worked on the development of mobile applications allowing maintenance technicians to update their interventions directly from their phones, even without Wi-Fi connection.

This internship offers a unique opportunity to explore the synergies between CMMS and AR technologies, and to deepen my understanding of frontend development. By primarily focusing on frontend development, this internship aims to bridge the gap between backend infrastructure and user interfaces. Furthermore, contributing to a project with real-world applications, as demonstrated in **(Ana, Mateus, and Torres 2021)** [**1**], is a great source of motivation, highlighting the relevance and importance of our work.

Chapter 3

3.1 What is Django?

Django is an open-source web framework written in Python, designed to simplify the development of complex and scalable web applications. Launched in 2005, Django is maintained by an active community of developers and is widely used in the industry to create a variety of web applications, ranging from simple websites to complex web platforms.

Key features of Django:

- **Full-stack Web Framework**: Django is a full-stack web framework, providing all the necessary tools to develop both the frontend and backend of a web application, including database management, business logic, URL routing, and form handling .
- **MVT Architecture (Model-View-Template)**: Unlike the MVC model, Django follows the MVT architecture where models represent application data, templates are responsible for presentation, and views contain the processing logic .

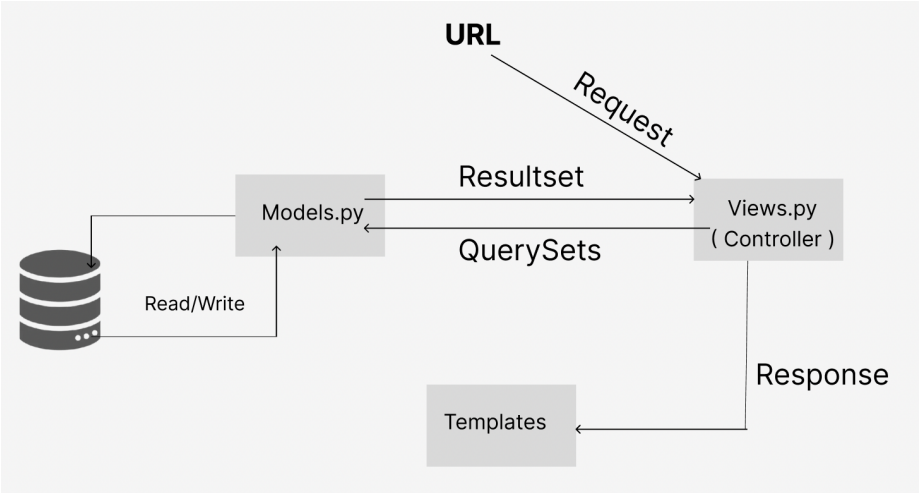

Figure 1 - Explanation MVT

- **ORM (Object-Relational Mapping)**: An ORM is a programming technique that allows mapping objects from an object-oriented programming language (such as Python, Java, etc.) to data stored in a relational database . Rather than writing SQL queries directly, the ORM allows developers to interact with the database using familiar objects and methods, thus facilitating application development and maintenance. In summary, an ORM abstracts the complexity of the relational database, allowing developers to work with data in a more object-oriented way.
- **URL Routing System**: Django uses a URL routing system that maps URLs to corresponding views, providing clean and efficient route management in the application .
- **Built-in Security**: Django comes with many built-in security features, such as protection against common security vulnerabilities like SQL injection attacks, cross-site scripting (XSS) attacks, and bypassing form validation.
- **Automatic Administration**: Django provides an automatically generated admin interface from the data models, allowing administrators to manage site content without needing to write specific code for it .
- **Batteries Included**: Django follows the “batteries included” principle, meaning it provides many ready-to-use features like user authentication, session management, pagination, static file management, and more .
- **Extensibility**: Django is highly extensible, allowing developers to add custom features to their application by using third-party packages or writing their own custom code .

3.2 What is CMMS (Computerized Maintenance Management Systems)?

Computerized maintenance management systems (CMMS) are sophisticated software solutions deployed by organizations to effectively streamline their maintenance operations. These systems form the backbone of maintenance management, orchestrating various tasks such as planning, organizing, tracking and managing maintenance activities across the organization.

At the heart of a CMMS is a robust and comprehensive database, which acts as a centralized repository of vital information relating to the organization's maintenance operations. This database stores a wealth of data, including detailed equipment specifications, maintenance schedules, historical maintenance records, spare parts inventory and work order histories.

One of the key benefits of CMMS is its ability to empower maintenance technicians by providing them with instant access to relevant information needed to perform their tasks accurately. Using the CMMS interface, technicians can quickly identify equipment requiring maintenance, view detailed maintenance procedures, and determine the availability of required spare parts. This streamlined access to information not only improves the efficiency of maintenance activities, but also helps minimize downtime by ensuring timely maintenance interventions.

In addition, CMMS plays a crucial role in facilitating informed decision-making at the managerial level. By analyzing the data stored in the CMMS database, managers can gain valuable insights into the performance of their maintenance operations. They can track key performance indicators, identify trends and forecast maintenance needs, enabling them to make data-driven decisions regarding resource allocation, budgeting and strategic planning. For example, managers can calculate the financial implications of repairing machine breakdowns versus preventative maintenance measures, thereby optimizing maintenance budgets and improving operational efficiency.

In summary, CMMS is a powerful tool for effectively managing a company's maintenance resources. By centralizing maintenance-related information, streamlining processes and providing valuable insights, CMMS enables organizations to optimize their maintenance operations, minimize downtime, reduce maintenance costs and, in turn, ultimately drive overall organizational performance and market competitiveness. But although it is very effective and useful for businesses it nevertheless remains very complex, as this article explains **(Mather 2002)** [**2**]

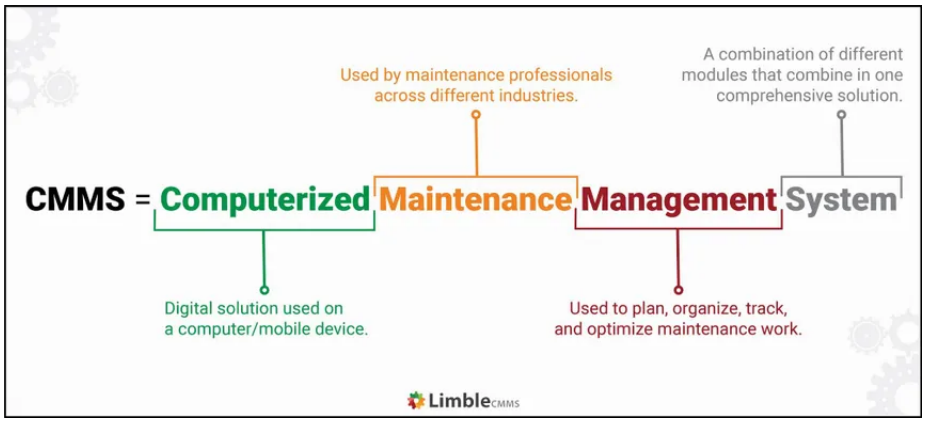

Figure 2 - Explanation CMMS Source : limblecmms.com/ZGl1bV9WMl8xOTgw/JUUyJTgwJThBJ Last search : 24/04/2024

3.3 What is Augmented Reality?

Augmented Reality (AR) is a revolutionary technology that overlays virtual elements such as images, videos, or 3D models onto the real world, typically through an electronic device such as a smartphone, tablet, or smart glasses. Unlike Virtual Reality (VR), which creates an entirely virtual environment, AR enhances and improves the real environment by adding digital information in real time.

Augmented reality applications can take various forms, offering a wide range of uses. They can serve as product visualization tools, allowing consumers to virtually try out products before purchasing them. Additionally, they can be used in interactive games that blend the real world and the virtual world, providing immersive and innovative gaming experiences. In the field of education, augmented reality can be used as an interactive learning tool, allowing learners to explore subjects in a more engaging and practical manner. Similarly, in the field of assistance, it can provide step-by-step guides to help users perform complex tasks, such as repairing electronic devices or navigating unfamiliar environments.

In summary, augmented reality offers significant potential to transform our interaction with the world around us, providing rich and immersive experiences in many areas, ranging from entertainment to education to commerce and industry.

Figure 3 - AR Illustration Source : gettyimages.fi/ZGl1bV9WMl8xOTgw/JUUyJTgwJThBJ Last search : 24/04/2024

3.4 Integration of Augmented Reality into Maintenance

The integration of Augmented Reality (AR) into maintenance operations signifies a remarkable advancement in optimizing industrial processes. By harmonizing the sophisticated functionalities of Computerized Maintenance Management Systems (CMMS) with the immersive capabilities of augmented reality, companies can curtail unplanned downtime and enhance overall operational efficiency. Through AR, maintenance technicians gain access to real-time contextual information seamlessly integrated into their field of vision, empowering them to conduct inspections and repairs with unprecedented precision. For instance, utilizing augmented reality glasses or headsets, a technician can effortlessly visualize critical data such as electrical schematics, technical specifications, or repair instructions overlaid onto the actual equipment, thereby diminishing the reliance on traditional paper manuals or digital documents on laptops and enabling faster and more precise interventions. Additionally, AR facilitates interactive step-by-step guides for intricate maintenance procedures, mitigating the risk of human errors and elevating the quality of repairs. By seamlessly integrating augmented reality into the existing framework of maintenance management systems, companies not only modernize their maintenance processes but also ignite innovation and fortify their competitive stance in the market.

A tangible application of AR in maintenance is illustrated in the article by **(Henderson and Feiner 2010)** [**3**]. They devised a prototype aimed at assisting military mechanics in executing routine maintenance tasks within an armored vehicle turret. The prototype employs a head-mounted display to augment a mechanic's natural view with text, labels, arrows, and animated sequences, tailored to enhance comprehension, localization, and task execution. A qualitative survey conducted revealed that mechanics found the augmented reality conditions intuitive and satisfactory for the tested task sequence.

Expanding on this momentum, a recent empirical study by **(Fiorentino et al. 2014)** [**4**] was conducted to assess the efficacy of technical maintenance aided by interactive instructions in augmented reality. This study introduces an innovative methodology incorporating augmented visualization on a large screen in conjunction with a fusion of multiple fixed and mobile cameras, leveraging commercially available solutions. During the experiment, 14 participants undertook a series of four maintenance tasks centered on manual inspections of a motorcycle engine. With augmented instructions providing support through visual labels, 3D virtual models, and animations, participants executed tasks such as tool selection, bolt removal, and part disassembly. Comparisons were drawn between traditional paper textbooks and augmented instructions. Results of rigorous statistical analyses unequivocally demonstrated that augmented instructions significantly reduced overall execution time and participant error rates, thereby underscoring the transformative potential of AR in optimizing maintenance processes.

To be even more concrete, we're going to look at a method of implementing augmented reality using Python . We're going to take a look at an Open CV Python library. This library was originally developed by Intel, which specializes in real-time image processing. The robotics company Willow Garage, followed by ItSeez, have since supported this library. The library provides a wide range of functions for creating programs from raw data to basic graphical interfaces. The software can also be used to develop facial recognition methods.

Figure 4 - Exemple d'utilisation d'openCV Source : viso.ai/ZGl1bV9WMl8xOTgw/JUUyJTgwJThBJ Last search : 26/04/2024

Chapter 4

Web Application

This chapter delves into the detailed process of developing our web application, from defining the requirements to practical implementation. It highlights the key stages of development, including requirement analysis, user interface design, and feature implementation.

4.1 Requirement Definition with Use Cases

The first step in development involved defining the functional requirements of the system using use cases. These allowed us to model the interactions between users and the system, identifying the main features and associated workflows.

4.1.1 Use Cases

4.1.2 Description of Use Cases

Login:

- **Actor:** Technician
- **Objective:** Log in to access features.
- **Preconditions:** On-site, connected to the Internet, and have an account.
- **Postconditions:** Logged in and can access features.
- **Steps:**
  1. Click on login.
  2. Enter required information.
  3. Click login.
  4. See success message.
  5. Logged in.

Logout:

- **Actor:** Technician

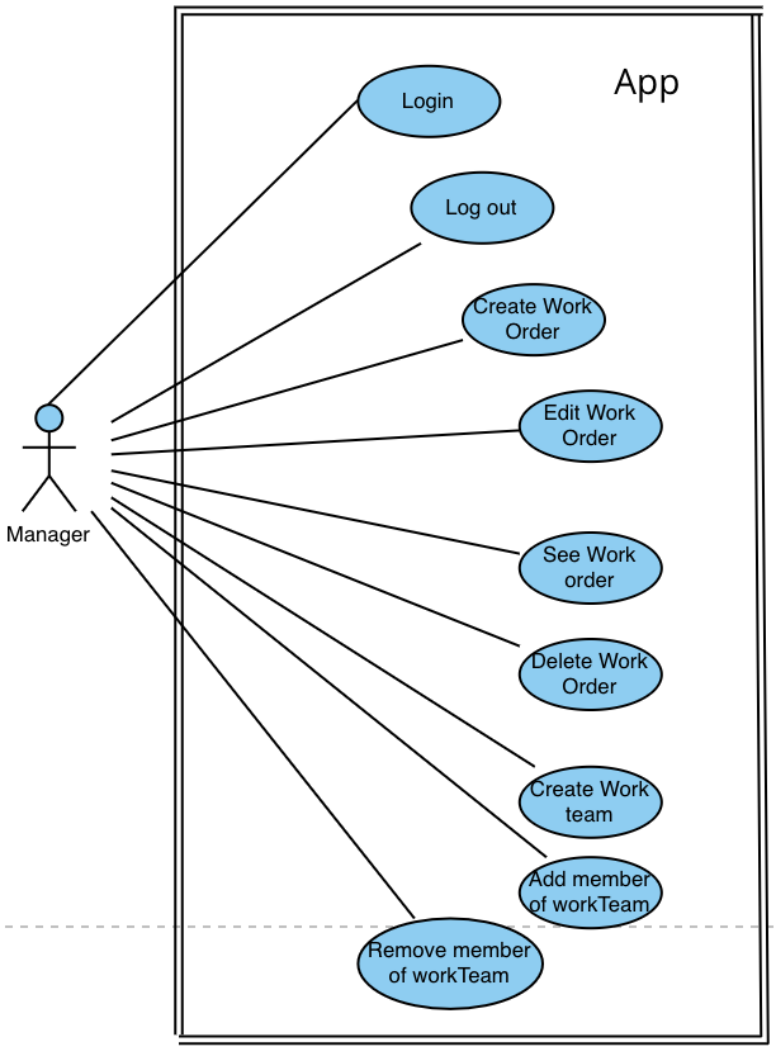

Figure 4.1: Use case for Technician

- **Objective:** Log out.
- **Preconditions:** On-site and logged in.
- **Postconditions:** Logged out.
- **Steps:**
  1. Click logout.
  2. Confirm logout.
  3. See success message.
  4. Logged out.

View Work Orders:

- **Actor:** Technician
- **Objective:** View work orders.
- **Preconditions:** On-site and logged in.
- **Postconditions:** View available work orders.
- **Steps:**
  1. Click work orders link.
  2. See list of work orders.
  3. View work orders.

Select Work Order:

- **Actor:** Technician
- **Objective:** Choose a work order.
- **Preconditions:** On-site, logged in, and on work orders list page.
- **Postconditions:** Selected work order, removed from list.
- **Steps:**
  1. Click desired work order.
  2. Click “Choose this work order”.
  3. Work order associated.

Validate Work Order:

- **Actor:** Technician
- **Objective:** Validate a work order.
- **Preconditions:** On-site, logged in, and selected a work order.
- **Postconditions:** Validated work order, no longer in list.
- **Steps:**
  1. Click “View my work orders”.
  2. See list of work orders.
  3. Click work order to validate.
  4. Click “Validate work order”.
  5. See success message.

Add Media to Work Order:

- **Actor:** Technician
- **Objective:** Add media to a work order.
- **Preconditions:** On-site, logged in, and selected a work order not yet validated.

- **Postconditions:** Media added to work order, viewable by others.
- **Steps:**
  1. Click “View my work orders”.
  2. See list of work orders.
  3. Select work order.
  4. Click “Add Media”.
  5. Upload media.
  6. Submit.
  7. See success message.

Alternative Paths: Display error message if uploaded media is incorrect.

4.2 User Interface Design with Mockups

Next, I created mockups to visually represent the user interface. These mockups were invaluable in clarifying design expectations and facilitating faster, more organized graphical interface development.

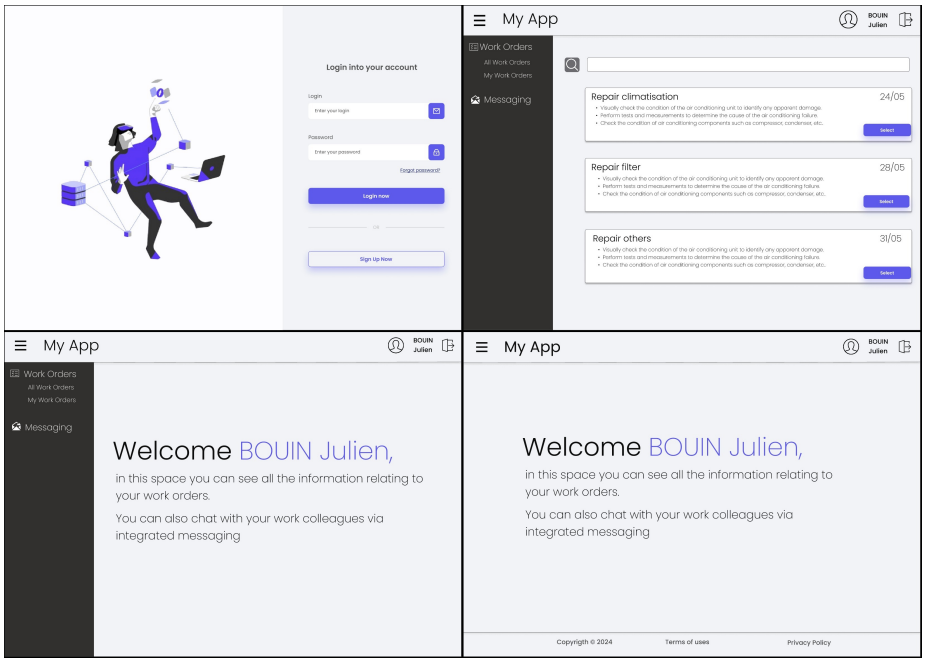

Figure 4.2: Mockup

4.3 Implementation

4.3.1 Frontend Development Using Django Templates

I embarked on frontend development of the application using Django templates. The goal was to create a user-friendly interface that was both functional and aesthetically pleasing. Here's a detailed summary of the work I accomplished:

To ensure well-structured and maintainable CSS, I learned and applied the BEM (Block, Element, Modifier) methodology. BEM is a naming convention for HTML and CSS classes that helps write clean and manageable code. It divides the interface into independent blocks, with elements as child components and modifiers to signify variations. This approach made the CSS more readable and easier to debug.

Next, one of the main requirements was to ensure the application was responsive, meaning it worked seamlessly on various devices and screen sizes. I used a combination of CSS Grid and Flexbox to create fluid layouts. Media queries were used to adjust the design for mobile, tablet, and desktop views, ensuring a consistent user experience. CSS Grid was used for creating complex, multidimensional layouts, while Flexbox was used for simpler, one-dimensional layouts. Media queries allowed applying different CSS rules based on device characteristics, such as screen width.

Django's template system allows powerful features such as template inheritance, which I utilized to maintain consistent layout across different pages. By creating a base template including common elements like header, footer, and navigation menu, I could reuse these components across multiple pages, reducing redundancy and simplifying maintenance. Inside the base template, block tags were used to define sections that child templates could override, providing flexibility to customize specific pages while maintaining a consistent overall structure.

The frontend needed to display dynamic data retrieved from the backend. Using Django's template language, I integrated data into the templates by passing context variables from views. This included displaying lists of work orders, user information, and other dynamic content. Context variables were passed from views to templates, allowing templates to access and display dynamic data. Template tags and filters were used to manipulate and display data in the templates, such as {% for %} loops for iterating over data and {% if %} statements for conditional rendering.

To enhance user interaction, I implemented various forms using Django's form handling capabilities. This included login and registration forms, work order submission forms, and feedback forms. Client-side validation was added to improve the user experience, providing immediate feedback on form inputs. Django forms were used to define the structure and validation rules for each form. By using Django's built-in form classes, I could quickly create and validate forms. Client-side validation was implemented using JavaScript to provide immediate

feedback on form inputs, such as displaying an error message immediately if a required field was left empty, without needing to reload the page.

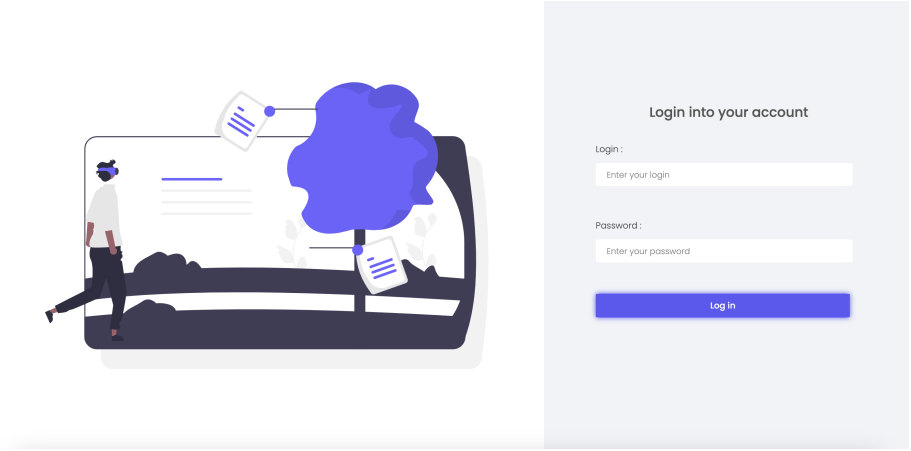

Figure 6 - Login Page

4.3.2 API Development with Django REST Framework

The second part of my internship focused on creating a RESTful API using Django REST Framework (DRF). This API aimed to manage data exchanges between the frontend and backend, as well as facilitate integrations with other systems. This step was crucial for the continuation of my work.

Firstly, it was crucial to understand what a REST API is and how to develop one. A RESTful API (Representational State Transfer) is an architectural style for designing networked applications. It relies on a stateless client-server communication protocol, typically HTTP. RESTful APIs allow interacting with application data via standard HTTP methods such as GET, POST, PUT, and DELETE.

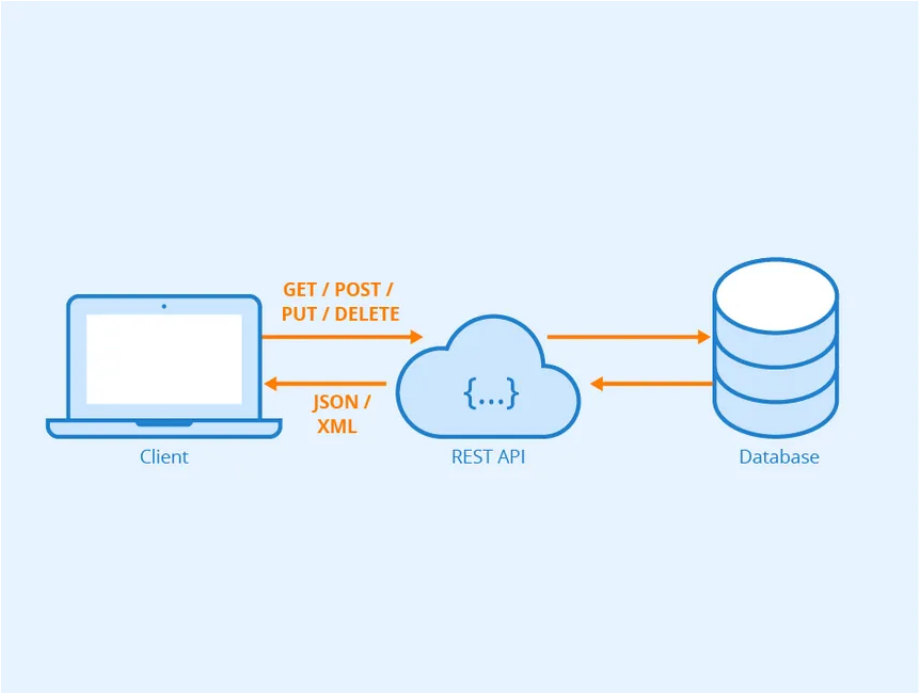

Figure 5 - API REST using Source: https://www.astera.com/fr/type/blog/rest-api-definition/ Last search: 21/05/2024

To set up this REST API using Django, I chose to use a Django framework called Django REST Framework, which allows for quicker and easier creation of REST APIs. I downloaded this framework.

Next, I created several API endpoints to handle different aspects of the application, such as user authentication and work order management. Each endpoint was defined in a dedicated view, using DRF's class-based views for better organization and scalability. These views were then associated with URL patterns in our Django app's urls.py file.

For data serialization, I used DRF serializers to convert model instances into JSON data and vice versa. This was crucial for data exchange between the frontend and backend. ModelSerializers simplified the conversion of model instances into JSON, and in some cases, custom serializers were created to handle more complex data transformations or validations.

Finally, to secure access to the API, I implemented token-based authentication using DRF's built-in token authentication system. Several token systems are available via DRF, but I decided to use Token BASIC as it was the most suitable in this situation. Each user received a unique token after successful login, which had to be included in the headers of subsequent API requests. This ensured that only authenticated users could access protected endpoints.

In summary, this part of my internship allowed me to develop a comprehensive RESTful API via Django, from the initial configuration of DRF to the creation of secure endpoints and data serialization management. This strengthened my backend development skills using Django.

4.3.3 Test Cases

After finishing the development of our website, we performed test cases to summarize what we have done and what is possible to do in our application. Here, we focused only on the use cases that directly change the data of our website.

4.3.3.1 Test Scenarios - Function: `take_task`

4.3.3.1.1 Scenario 1: Successful Task Takeover **Description:** Verify that a technician can successfully take a task with valid details. **Expected Result:** The task is updated, added to the technician's list, and a success message is displayed. **Actual Result:** The task is updated, added to the technician's list, and a success message is displayed. **Status:** Passed

4.3.3.1.2 Scenario 2: Task Takeover with Non-existent User **Description:** Verify that an error is handled when the user ID does not exist. **Expected Result:** An error message "The user or task does not exist." is displayed. **Actual Result:** An error message "The user or task does not exist." is displayed. **Status:** Passed

4.3.3.1.3 Scenario 3: Task Takeover with Non-existent Task **Description:** Verify that an error is handled when the task ID does not exist. **Expected Result:** An error message "The user or task does not exist." is displayed. **Actual Result:** An error message "The user or task does not exist." is displayed. **Status:** Passed

4.3.3.1.4 Scenario 4: GET Request to Take Task Endpoint **Description:** Verify that the take task endpoint does not allow GET requests. **Expected Result:** The request is ignored or an appropriate response is given. **Actual Result:** The request is ignored or an appropriate response is given. **Status:** Passed

4.3.3.1.5 Scenario 5: Authorization Check **Description:** Verify that only users with technician rights can access the take task functionality. **Expected Result:** Access is denied, and the user is redirected or shown an error message. **Actual Result:** Access is denied, and the user is redirected or shown an error message. **Status:** Passed

4.3.3.1.6 Scenario 6: Odoo Integration Error Handling Description: Verify that an error message is shown if there is an issue with Odoo integration. **Expected Result:** An error message detailing the Odoo error is displayed. **Actual Result:** An error message detailing the

Chapter 5

Mobile Application

5.1 Development of a mobile application

5.1.1 Why a mobile application?

Following the development of a website to address the aforementioned needs, my supervisor proposed creating a mobile application mirroring the website's functionalities. This would enable technicians to directly access it via their phones. Technicians can log in, view their work orders, select them, and add notes similar to the website. However, the significant advantage of this application is its offline functionality. Technicians can update their work orders even without an internet connection. They can add notes, change work order states, and upon reconnecting, all changes will synchronize with our application.

5.1.2 Choice of Language

Initially, with little knowledge about mobile applications, I conducted thorough research upon realizing the need to develop one. Two standout technologies emerged: React Native and Flutter.

React Native, an open-source framework by Facebook, and Flutter, by Google, support cross-platform development for both Android and iOS. Further research was conducted to determine the most suitable choice.

After evaluation, React Native emerged as the preferred option. Its utilization of JavaScript, a familiar language, was advantageous for me.

5.1.3 Use Cases

For this application, the use cases remained consistent with our web application. The primary goal was to provide technicians with identical functionalities on their mobile devices as on the website.

5.1.4 Development with React Native

The development of the mobile application was an enriching journey, where I delved into React Native's intricacies while addressing project-specific challenges. Here's an account of my experiences throughout the development process:

5.1.4.1 Project Structuring

Initiating the mobile application development, my first step was meticulously organizing the project structure. Drawing inspiration from React Native community best practices, I divided the code into distinct components, views, and functionalities. This structuring facilitated clear project oversight and seamless navigation between code segments.

5.1.4.2 Creation of Reusable Components

Subsequently, emphasis was placed on crafting reusable components. Recognizing code reusability's pivotal role in maintaining clean and efficient code, I developed components for common UI elements like buttons, forms, and lists. This approach optimized time and effort by averting code duplication.

5.1.4.3 State Management

Effectively managing the application state emerged as a crucial lesson during development. Leveraging React's state hooks and contexts, I ensured consistent data sharing among various components. This fostered a smooth and responsive user experience, even during concurrent interactions with multiple components.

5.1.4.4 Smooth Navigation

Establishing seamless navigation between application screens posed an exciting challenge. Utilizing Expo Router, I structured a clear navigation hierarchy and defined fluid transitions between different views. This library provided the requisite flexibility to design an intuitive and ergonomic user experience.

5.1.4.5 Offline Capability

One of the application's most intriguing features was its offline functionality. I designed the application to locally store necessary data and synchronize modifications with the backend upon internet reconnection. This feature proved invaluable in scenarios with limited or unstable internet connectivity, crucial for technicians to modify work orders during maintenance operations.

5.1.4.6 Seamless API Integration

A gratifying aspect of my work was ensuring seamless integration between the mobile application and the existing backend. By utilizing the same API endpoints as the web application, I ensured a consistent user experience. Technicians could

access the same data and functionalities across both web and mobile platforms. This compatibility bolstered our commitment to delivering a uniform and fluid user experience across devices.

Chapter 6

Conclusion

My internship experience at the Instituto Superior de Engenharia de Coimbra (ISEC) has been a period filled with discoveries and learning. By working on the development of a full-stack website and mobile applications integrating CMMS and augmented reality technologies, I have deepened my skills in both frontend and backend development.

Collaborating with my colleague Rayane Belguebli and our supervisor, Mateus Mendes, was essential for the success of this project. Together, we overcame technical challenges and made progress towards achieving our goals.

This internship has also provided me with a better understanding of the importance of synergies between different technologies and their impact on the efficiency of maintenance processes. The integration between CMMS and augmented reality offers promising prospects for the future of maintenance operations management.

In conclusion, this experience has not only allowed me to acquire new technical skills but has also provided me with a deeper insight into the challenges and opportunities in the field of software development applied to industrial maintenance. I am grateful for this opportunity and confident that the knowledge gained during this internship will be valuable for my future professional journey.

Chapter 7

References

Ana, Malta, Mendes Mateus, and Farinha Torres. 2021. "Augmented Reality Maintenance Assistant Using YOLOv5 [j]." *Applied Sciences* 11 (11): 4758–58.

Fiorentino, Michele, Antonio E Uva, Michele Gattullo, Saverio Debernardis, and Giuseppe Monno. 2014. "Augmented Reality on Large Screen for Interactive Maintenance Instructions." *Computers in Industry* 65 (2): 270–78.

Henderson, Steven, and Steven Feiner. 2010. "Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair." *IEEE Transactions on Visualization and Computer Graphics* 17 (10): 1355–68.

Mather, Daryl. 2002. *CMMS: A Timesaving Implementation Process*. CRC Press.