Internship project report

Belguebli Rayane

Table of Contents

- **Abstract**
- **Chapter 1**
  - Introduction
- **Chapter 2**
  - What is CMMS ?
    - Examples of CMMS
      - CMMS comparaison
  - What are Maintenance Management Systems?
  - What is the django framework et and why use this for our web application ?
- **Chapter 3**
  - Requirements
    - Use Case
    - Non-Functional Requirements
  - External API Odoo
  - Database
  - Tests
- **Chapter 4**
- **Chapter 5**
  - Conclusion
- **References**

Abstract

Our project aims to create an innovative web platform that acts as a management centre for maintenance tasks. Data from a CMMS relating to tasks to be carried out, planned by managers, is retrieved and presented interactively to technicians via glasses equipped with AR technology. This information, integrated directly into their field of vision, gives technicians rapid access to instructions and task details, improving their efficiency and accuracy when working in the field. By using Django based on the MVT model, we are ensuring a robust and scalable architecture to support this complex integration between maintenance data and AR technology.

Chapter 1

Introduction

In the field of industrial maintenance, the efficient management of tasks and resources is crucial to ensuring the productivity and reliability of operations (Malta, Mendes, and Farinha 2021) [1]. Our project addresses this issue by proposing an innovative solution that exploits emerging technologies to facilitate the maintenance process. Inspired by recent advances on ICMMS (Fu et al. 2002) [2], we aim to design an integrated Django-based web platform to orchestrate coordination between managers and technicians, while integrating Augmented Reality (AR) features for an immersive experience.

This solution meets a growing need in the field of industrial maintenance, where the complexity of equipment and operations requires agile and efficient management. By combining the power of asset management with the ease of use of an intuitive web interface, our solution aims to streamline maintenance processes while providing an optimal user experience.

Our aim is to provide a versatile and adaptable platform that can be tailored to the specific needs of different industries and businesses. Through the integration of AR, technicians will be able to access contextual information in real time, improving their efficiency and accuracy when working in the field.

This project represents an exciting opportunity for an internship abroad, offering the chance to explore new technologies and contribute to innovative solutions in the field of industrial maintenance. By taking on this challenge, we aim to acquire new skills.

Chapter 2

What is CMMS ?

A Computerised Maintenance Management System (CMMS) is an essential tool for the efficient management of maintenance activities within an organisation. Whether in industry, services or public institutions, a CMMS provides a centralised platform where maintenance teams can effectively plan, execute and monitor their tasks.

One of the key benefits of a CMMS is its ability to integrate new technologies such as mobility and traceability applications. Using mobile devices such as smartphones or tablets, field technicians can access maintenance information in real time, enter data on the spot and receive instant instructions. This significantly improves the responsiveness and efficiency of the maintenance team, reducing unplanned downtime.

In addition, traceability of maintenance activities is a crucial element in ensuring regulatory compliance and optimising processes. Modern CMMSs offer advanced monitoring and reporting capabilities, enabling managers to track maintenance histories, analyse trends and make informed decisions to improve overall asset performance.

Studies and articles have been published to demonstrate the effectiveness of CMMS in different sectors. For example, research carried out by experts in maintenance management revealed that the implementation of a CMMS in an industrial company led to a significant reduction in downtime and maintenance costs, while improving equipment availability (Shankar, Singh, and Singh 2021) [3].

Similarly, another case study that highlighted the benefits of a CMMS in the utilities sector, showing how a municipality was able to optimise its public infrastructure maintenance operations through the use of an integrated CMMS solution (Kour et al. 2022) [4].

In summary, CMMS are essential tools for maintenance teams, providing efficient management of maintenance activities while taking advantage of new technologies to improve responsiveness and traceability. Studies and research articles support the positive impact of CMMS in different sectors, highlighting their value as a key element of asset management and preventive maintenance.

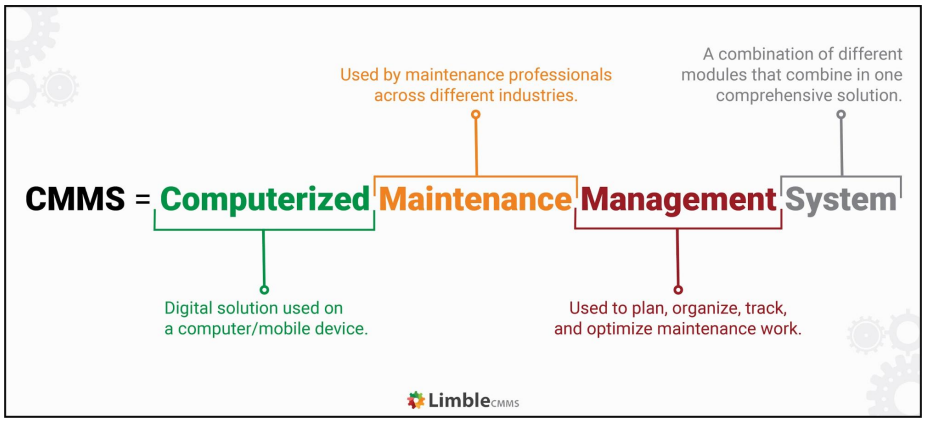

Figure 1: CMMS source(04/2024) : https://limblecmms.com/cmms/

Examples of CMMS

- **MaintainX** :

Figure 2: MAINTAINX source(04/2024) : https://www.prnewswire.com/news-releases/maintainx-raises-50m-with-12x-revenue-growth-from-boosting-productivity-of-industrial-and-frontline-workers-as-america-gets-back-to-work-301308326.html

MaintainX is a CMMS based on a mobile application and a web platform that aims to simplify the maintenance process for teams in the field.

It offers functionalities such as work order creation, asset management, spare parts inventory tracking, preventive maintenance task planning and report generation.

The mobile application allows technicians to receive real-time notifications, update the status of tasks and upload photos or notes to document interventions.

The web platform provides managers with a centralised dashboard where they can track the progress of jobs, analyse performance data and generate reports to optimise maintenance processes. source(04/2024)

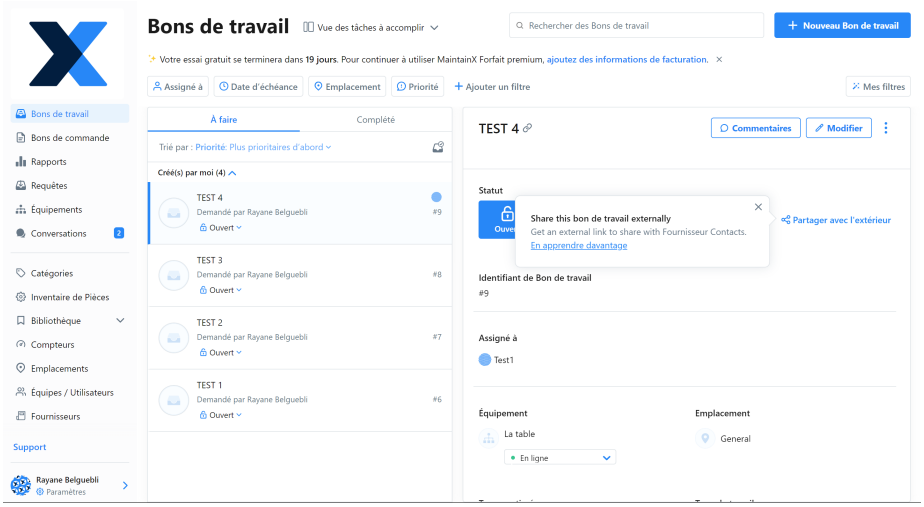

Figure 3: workorders

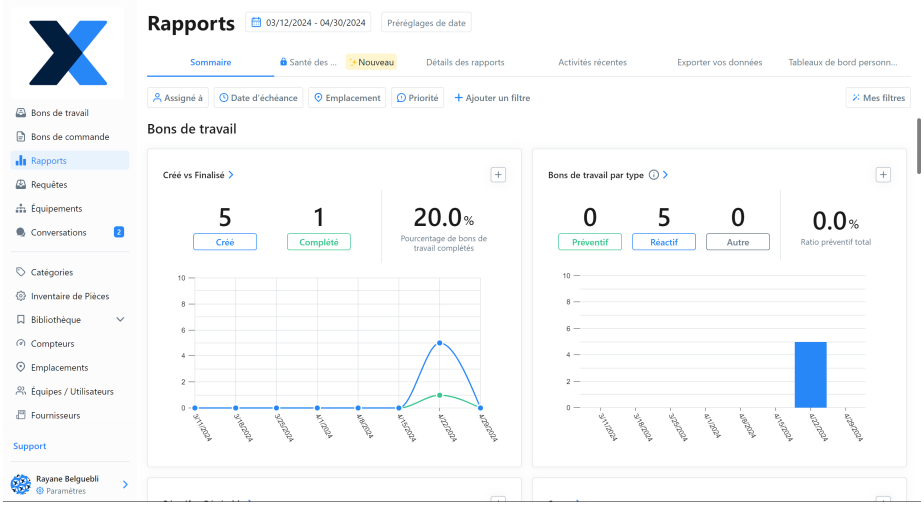

Figure 4: reports

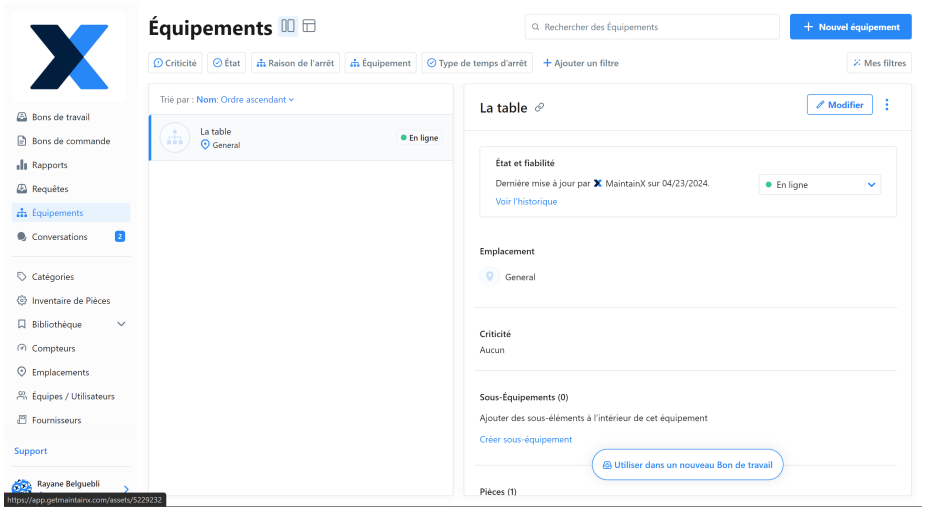

Figure 5: assets

- **Limble** :

Figure 6: LIMBLE source(04/2024) : https://www.prnewswire.com/news-releases/limble-announces-58m-series-b-funding-round-led-by-goldman-sachs-asset-management-bringing-total-valuation-to-450m-301857646.html

Limble is also a modern CMMS designed to simplify the management of maintenance activities and extend the life of assets.

It offers similar functionality to MaintainX, such as work order creation, scheduling, spares management and reporting.

Limble is distinguished by its user-friendly interface and advanced asset management features, such as equipment hierarchy modelling, warranty management and replacement planning.

It also incorporates predictive maintenance and performance dashboard capabilities to help users anticipate breakdowns and make informed decisions to optimise maintenance. source(04/2024)

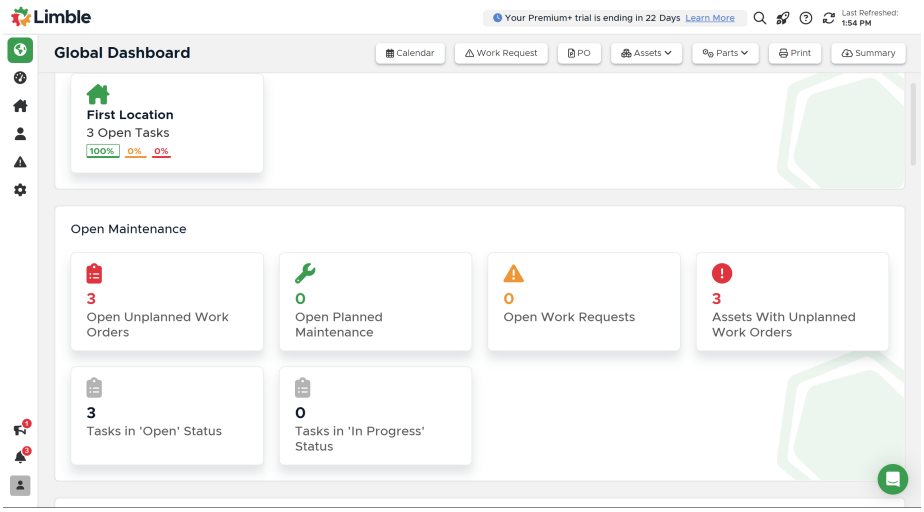

Figure 7: dashboard

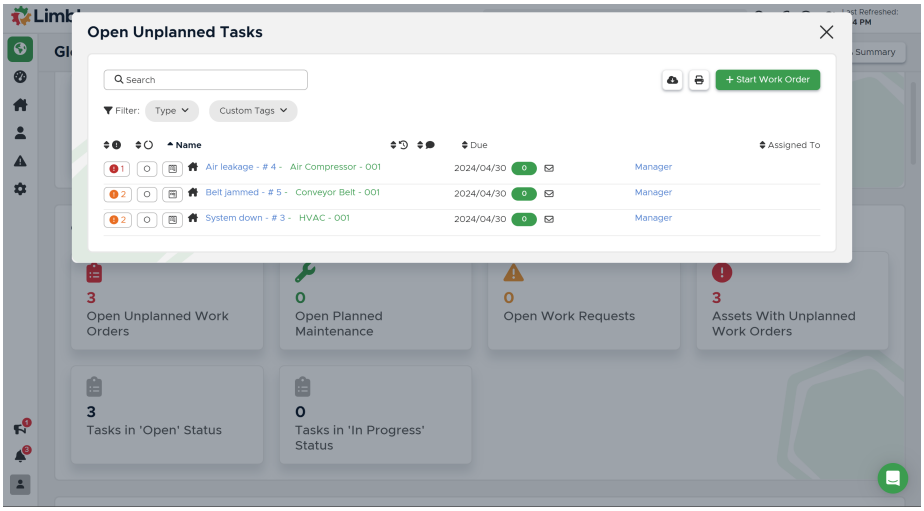

Figure 8: Tasks

- **Odoo Maintenance**

Figure 9: ODOO source(04/2024 : https://ubi.edu/odoo/)

Odoo is a suite of open source applications covering all your business needs: CRM, eCommerce, Accounting, Inventory, Point of Sale, Project Management, etc.

Odoo has a dedicated maintenance section that allows you to manage work orders and assets. Odoo Maintenance is perhaps less advanced than the previous CMMS but is sufficient to solve our problem and above all is free to download. source(04/2024)

Odoo is based on a 3-tier architecture:

- a PostgreSQL database server, which can contain several databases ;
- an application server containing the management objects, workflow engine, editing generator, etc. ;
- a presentation server that allows users to connect to OpenERP using any Web browser (with the Flash player installed for displaying graphics). This last server is not necessary if the user uses the native client, which does require installation on the user's workstation.

The server part is written in Python. The various building blocks are organised into modules. A module is a folder with a predefined structure containing Python code and XML files. A module defines the data structure, forms, reports, menus, procedures, workflow, etc. source(04/2024)

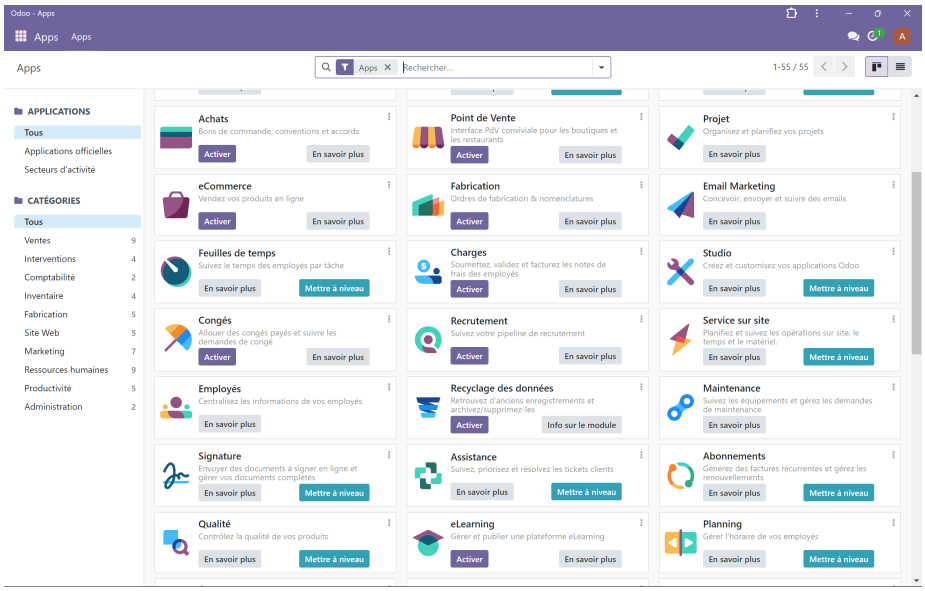

Figure 10: apps

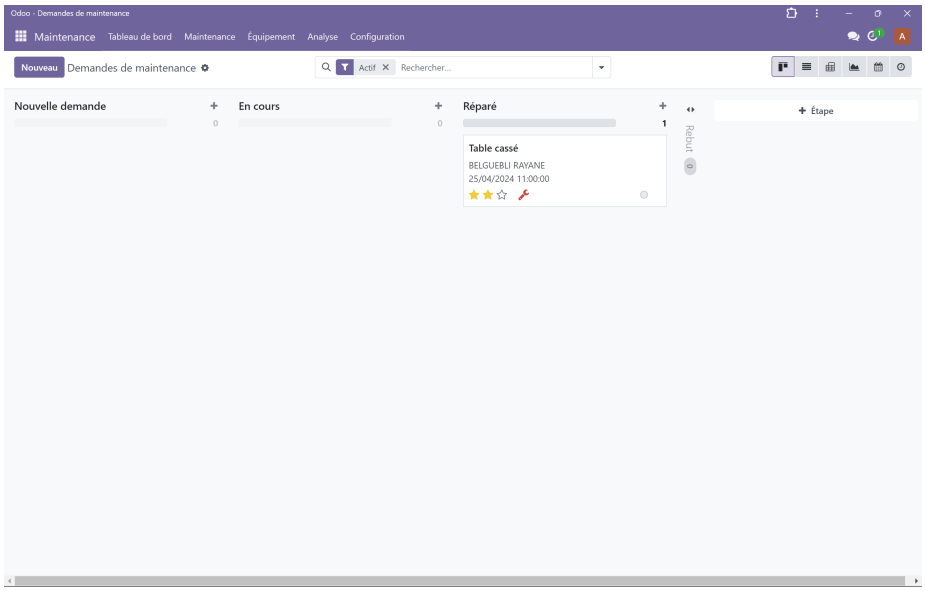

Figure 11: maintenace

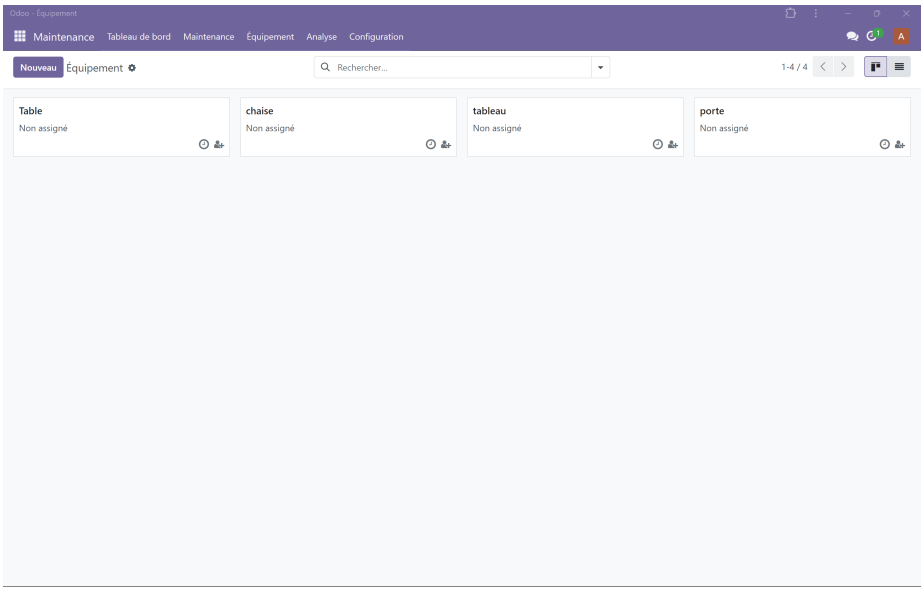

Figure 12: assets

CMMS comparison

	MaintainX	Limble	Odoo
Main features	work orders, digital checklists and real-time notifications.	work order, preventive maintenance, asset management and analytical dashboards.	work order management, preventive maintenance planning, asset tracking, spare parts stock management and report generation.
User interface	user-friendly mobile interface for fast, efficient communications.	intuitive interface, with simple navigation and clear menus.	user interface is renowned, ease of use and customisation.
Prices	paid enterprise version and online community version.	paid enterprise version and online community version.	paid enterprise version and open source community version.

	MaintainX	Limble	Odoo
Integration and customisation	MaintainX offers integrations with messaging tools and communication platforms such as Slack, as well as the ability to customise checklists and forms.	asset management systems, accounting software and other productivity tools.	other Odoo applications as well as with third-party software via additional modules.

What are Maintenance Management Systems?

Maintenance is a critical aspect of any manufacturing or industrial process. It ensures that equipment, machines and facilities operate at their optimum level, avoiding breakdowns and costly downtime. To achieve this, companies need to put in place a well-defined Maintenance Management System (MMS). An MMS is a holistic approach to maintenance management that involves the integration of people, processes and technology to optimise maintenance activities.

The main objective of an MMS is to improve equipment reliability, minimise downtime and increase productivity. It does this by providing a structured approach to maintenance planning, scheduling and execution. This helps maintenance teams to proactively identify and resolve potential problems before they become major issues. By leveraging data, analytics and technology, an MMS enables organisations to optimise maintenance processes, reduce costs and increase asset life. Here are some key points highlighting their importance and benefits :

- **Optimising maintenance operations** : MMS provides a centralised platform for planning, executing and monitoring maintenance activities. This enables efficient resource allocation, priority management and coordination of maintenance teams.
- **Reduced unplanned downtime** : By enabling preventive and predictive maintenance, MMS helps to identify and resolve problems before they become major breakdowns. This reduces unplanned downtime and minimises production interruptions.
- **Extended equipment life** : By providing regular, preventive maintenance, MMS helps to extend the life of equipment. By identifying and correcting potential problems at an early stage, MMS reduces wear and tear on assets.
- **Maximising asset availability** : By ensuring that equipment is well maintained and available when needed, MMS maximises asset availability. This enables organisations to maintain productivity and respond effectively to market demand.

In summary, MMS plays a crucial role in optimising maintenance operations, reducing downtime, extending equipment life and maximising asset availability. The above references illustrate the positive impact of MMS in various sectors and highlight their value as strategic tools for asset management and operational performance (Sorić 2024) [5].

What is the django framework et and why use this for our web application ?

Figure 13: DJANGO source(04/2024) : https://www.djangoproject.com/

In our project to create an innovative web platform that acts as a management centre for maintenance tasks, Django proved to be a wise choice for several key reasons :

- **MVC/MVT structure adapted to our architecture** : Django follows the MVT model, which is a variant of the MVC model. This architecture is particularly well-suited to our project because it allows us to clearly separate the different responsibilities of our application.
  - The Model represents data from the CMMS, such as maintenance task details, equipment information, schedules, etc.
  - The Template is responsible for presenting the data to technicians. In our case, this means generating content adapted to augmented reality for display on their glasses.
  - The View manages the business logic of our application, including the integration of data from the CMMS with augmented reality functionalities to provide technicians with precise, interactive instructions.
- **Managing complex data with Django's ORM** : Django's ORM (Object-Relational Mapping) is one of its most powerful and popular features. It greatly simplifies data manipulation by allowing developers to interact with the database using Python objects rather than direct SQL queries. Here are some key aspects of Django's ORM and why it's beneficial for our project:
  - **Database abstraction** : Django's ORM provides a high-level abstraction of the database, meaning developers don't need to worry

about the specific details of the underlying database (such as the type of database used or the SQL language).

This allows Django applications to be developed in a more portable way, as the code can be easily adapted to different types of database without having to change the business logic.

- **Using Python objects to represent data** : Database tables are represented by Python classes called "models" in Django. Each model defines the table's fields and relationships with other models.

  For example, in our project, we might have a Task model to represent the various maintenance tasks, with fields such as description, start date, end date, status, etc.

- **Easy data manipulation** : Once models are defined, CRUD (Create, Read, Update, Delete) operations can be performed using simple and intuitive methods on model objects.

  For example, to create a new task in our project, we would simply create an instance of the Task class, assign it the appropriate values for its fields, and then call the save() method to save it in the database.

- **Transparent management of relationships between tables** : Django's ORM makes it easy to define relationships between models, such as foreign keys and many-to-many relationships.

  For example, in our project, a task could be linked to a specific piece of equipment. This relationship can be easily defined in the Task model by adding a foreign key field that points to the Equipment model.

- **Protection against SQL injections** : Using Django's ORM provides built-in protection against SQL injections, as queries are securely generated using query parameters, preventing SQL injection attacks.

- **Flexible presentation of AR data** : Django has a flexible template system that allows dynamic content to be generated for a variety of devices, including augmented reality glasses.

  Templates can be customised to deliver instructions interactively, displaying details of maintenance tasks directly in the technicians' field of vision, improving their productivity and accuracy.

- **Security and scalability**: Security is a major concern, especially when it comes to handling sensitive data such as that associated with industrial maintenance. Django incorporates robust security features, such as protection against SQL injections and CSRF attacks, to ensure the security of user data.

  In addition, Django is known for its ability to manage large-scale web applications efficiently. Its scalability will enable our platform to grow

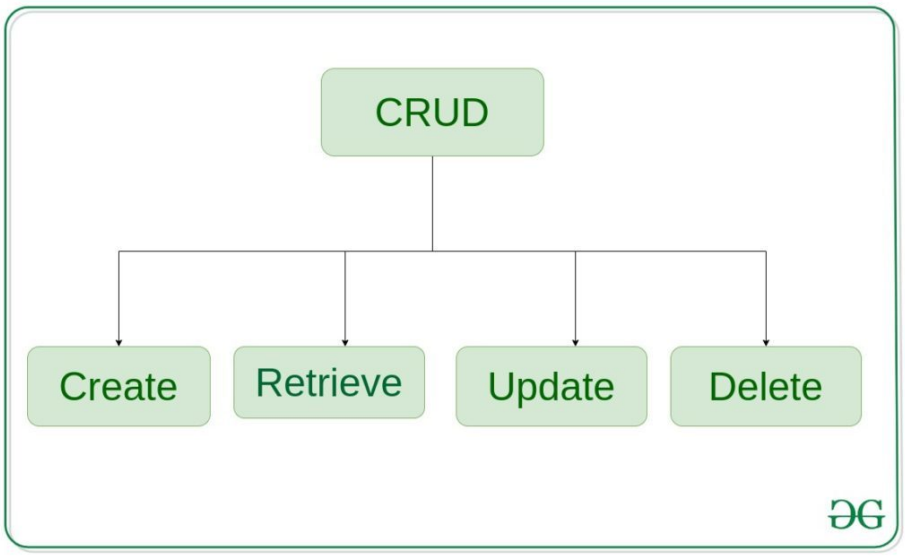

Figure 14: CRUD source(04/2024) : https://www.geeksforgeeks.org/django-crud-create-retrieve-update-delete-function-based-views/

with the addition of new features and larger volumes of data.

In short, Django offers a combination of powerful features, built-in security and flexibility that make it the ideal choice for our maintenance task management platform project incorporating augmented reality. Its MVC/MVT architecture, robust ORM and flexible templating system ensure that our solution can be implemented efficiently and scalably.

Chapter 3

Requirements

Use Case

Non-Functional Requirements

- Performance: The backend system should be highly performant and scalable to handle a large number of concurrent users and data requests. Optimize database queries and data processing to minimize response times and maintain overall application responsiveness.
- Security: Implement robust security measures to protect sensitive data, including encryption of data at rest and in transit.
- Maintainability: Employ modular and well-documented code to facilitate easy understanding, modification, and maintenance of the backend system.

External API Odoo

Connection

This part establishes the connection to the Odoo server using the provided URL, database name, username, and password.

```
url = <insert server URL>
db = <insert database name>
username = 'admin'
password = <insert password for your admin user (default: admin)>
```

GET

This example demonstrates how to retrieve data from Odoo, specifically from the 'res.users' model, using the 'search_read' method. It retrieves all records ('search_read' with an empty domain) and specifies which fields to include in the response.

```
models.execute_kw(db, uid, password, 'res.users', 'search_read', [[]], {'fields': []})
```

PATCH

This example shows how to update an existing record in the 'maintenance.request' model. It uses the 'write' method with the record ID (in this case, 33) and a dictionary containing the field(s) to update.

```
models.execute_kw(
        db, uid, password,
        'maintenance.request', 'write',
        [[33], {'schedule_date': formatted_now}]
    )
```

CREATE

Here, you're creating a new user in the 'res.users' model. You provide a dictionary with the necessary data for the new user, such as name, login, and password, and use the 'create' method to add it to the database.

```
new_user_data = {
    'name': "signup_username",
    'login': "signup_email",
    'password': "signup_password",
}
models.execute_kw(db, uid, password, 'res.users', 'create', [new_user_data])
```

DELETE

This example demonstrates how to delete a record from the 'res.users' model. It uses the 'unlink' method with the ID of the record to be deleted (in this case, 2).

```
models.execute_kw(db, uid, password, 'res.users', 'unlink', [[2]])
```

source (05/2024)

Database

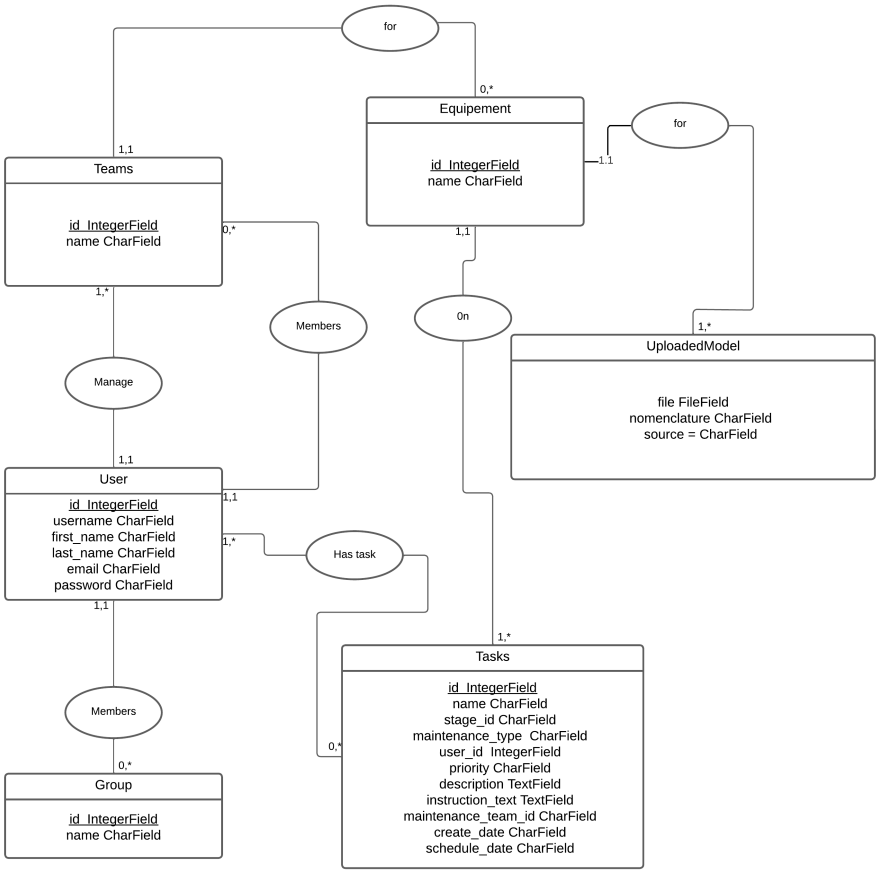

Figure 15: Database

Tests

General Information **Project:** Internship Project
Function: `user_login`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `user_login` method, which handles user authentication in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Login **Description:** Verify that a user can successfully log in with correct credentials.
Pre-condition: The user has valid login credentials.
Steps: - Send a POST request with valid `username` and `password`. - Authenticate the user. - Redirect to the home page if authentication is successful.

Expected Result: The user is logged in and redirected to the home page.
Actual Result: The user is logged in and redirected to the home page.
Status: Passed

Scenario 2: Unsuccessful Login with Incorrect Credentials **Description:** Verify that a user cannot log in with incorrect credentials.
Pre-condition: The user has invalid login credentials.
Steps: - Send a POST request with an incorrect `username` and/or `password`. - Attempt to authenticate the user. - Display an error message if authentication fails.

Expected Result: An error message "Incorrect username or password." is displayed.
Actual Result: An error message "Incorrect username or password." is displayed.
Status: Passed

Scenario 3: GET Request to Login Page **Description:** Verify that the login page is rendered when accessed with a GET request.
Pre-condition: None
Steps: - Send a GET request to the login URL. - Render the login page.

Expected Result: The login page is rendered.
Actual Result: The login page is rendered.
Status: Passed

Scenario 4: CSRF Protection **Description:** Verify that CSRF protection is enabled for the login form.
Pre-condition: None
Steps: - Send a POST request to the login URL without a CSRF token. - Attempt to process the login request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Summary of Results

- **Total Tests:** 4
- **Passed:** 4
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `user_login` function performs as expected in handling user authentication. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., rate limiting, account lockout) to ensure robustness.

General Information **Project:** Internship Project
Function: `add_manager`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `add_manager` method, which handles the addition of a new manager in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Manager Addition **Description:** Verify that an admin user can successfully add a new manager with valid details.
Pre-condition: The user has admin rights and the provided username does not exist.
Steps: - Send a POST request with `username`, `email`, and `password`. - Check if the user already exists. - Create a new user in Odoo. - Create a new user in Django. - Add the new user to the 'Manager' group. - Redirect to the home page with a success message.

Expected Result: The user is created and added to the 'Manager' group. A success message is displayed.
Actual Result: The user is created and added to the 'Manager' group. A success message is displayed.
Status: Passed

Scenario 2: Manager Addition with Existing Username **Description:** Verify that an error message is shown if the username already exists.
Pre-condition: The user has admin rights and the provided username already exists.
Steps: - Send a POST request with `username`, `email`, and `password`. - Check if the user already exists. - Display an error message if the username exists.

Expected Result: An error message "This user already exists." is displayed.
Actual Result: An error message "This user already exists." is displayed.
Status: Passed

Scenario 3: GET Request to Add Manager Page **Description:** Verify that the add manager page is rendered when accessed with a GET request.
Pre-condition: The user is authenticated and has admin rights.
Steps: - Send a GET request to the add manager URL. - Render the add manager page with the appropriate context.

Expected Result: The add manager page is rendered with the context.
Actual Result: The add manager page is rendered with the context.
Status: Passed

Scenario 4: CSRF Protection Description: Verify that CSRF protection is enabled for the add manager form.
Pre-condition: None
Steps: - Send a POST request to the add manager URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 5: Authorization Check Description: Verify that only users with admin rights can access the add manager functionality.
Pre-condition: The user is authenticated but does not have admin rights.
Steps: - Attempt to access the add manager URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling Description: Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: The user has admin rights, and Odoo integration is functional.
Steps: - Send a POST request with `username`, `email`, and `password`. - Simulate an error during Odoo user creation. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `add_manager` function performs as expected in handling the addition of a new manager. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship Project **Function:** `add_technician`
Date: May 30, 2024
Tested by: Belguebli Rayane

Summary The objective of this test was to verify the functionality of the `add_technician` method, which handles the addition of a new technician in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Technician Addition **Description:** Verify that a manager can successfully add a new technician with valid details.
Pre-condition: The user has manager rights and the provided username does not exist.
Steps: - Send a POST request with `username`, `email`, and `password`. - Check if the user already exists. - Create a new user in Odoo. - Create a new user in Django. - Add the new user to the 'Technician' group. - Add the new user to the manager's team. - Redirect to the home page with a success message.

Expected Result: The user is created, added to the 'Technician' group, and added to the manager's team. A success message is displayed.
Actual Result: The user is created, added to the 'Technician' group, and added to the manager's team. A success message is displayed.
Status: Passed

Scenario 2: Technician Addition with Existing Username **Description:** Verify that an error message is shown if the username already exists.
Pre-condition: The user has manager rights and the provided username already exists.
Steps: - Send a POST request with `username`, `email`, and `password`. - Check if the user already exists. - Display an error message if the username exists.

Expected Result: An error message "This user already exists." is displayed.
Actual Result: An error message "This user already exists." is displayed.
Status: Passed

Scenario 3: GET Request to Add Technician Page **Description:** Verify that the add technician page is rendered when accessed with a GET request.
Pre-condition: The user is authenticated and has manager rights.
Steps: - Send a GET request to the add technician URL. - Render the add technician page with the appropriate context.

Expected Result: The add technician page is rendered with the context.
Actual Result: The add technician page is rendered with the context.
Status: Passed

Scenario 4: CSRF Protection **Description:** Verify that CSRF protection is enabled for the add technician form.
Pre-condition: None
Steps: - Send a POST request to the add technician URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 5: Authorization Check **Description:** Verify that only users with manager rights can access the add technician functionality.
Pre-condition: The user is authenticated but does not have manager rights.
Steps: - Attempt to access the add technician URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling **Description:** Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: The user has manager rights, and Odoo integration is functional.
Steps: - Send a POST request with `username`, `email`, and `password`. - Simulate an error during Odoo user creation. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `add_technician` function performs as expected in handling the addition of a new technician. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship Project
Function: `take_task`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `take_task` method, which allows a technician to take a task and update its status in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Task Takeover **Description:** Verify that a technician can successfully take a task with valid details.
Pre-condition: The user is a technician and the provided task ID and user ID exist.
Steps: - Send a POST request with `taskId` and `userId`. - Retrieve the user and task from the database. - Update the task in Odoo. - Add the task to the technician's list of tasks. - Redirect to the home page with a success message.

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

Scenario 2: Task Takeover with Non-existent User **Description:** Verify that an error is handled when the user ID does not exist.
Pre-condition: The provided user ID does not exist.
Steps: - Send a POST request with `taskId` and `userId`. - Attempt to retrieve the user from the database. - Handle the `User.DoesNotExist` exception.

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

Scenario 3: Task Takeover with Non-existent Task **Description:** Verify that an error is handled when the task ID does not exist.
Pre-condition: The provided task ID does not exist.
Steps: - Send a POST request with `taskId` and `userId`. - Attempt to retrieve the task from the database. - Handle the `Task.DoesNotExist` exception.

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

Scenario 4: GET Request to Take Task Endpoint Description: Verify that the take task endpoint does not allow GET requests.
Pre-condition: None
Steps: - Send a GET request to the take task URL. - Ensure the request is not processed.

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

Scenario 5: Authorization Check Description: Verify that only users with technician rights can access the take task functionality.
Pre-condition: The user is authenticated but does not have technician rights.
Steps: - Attempt to access the take task URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling Description: Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: The user has technician rights, and Odoo integration is functional.
Steps: - Send a POST request with `taskId` and `userId`. - Simulate an error during the Odoo task update. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `take_task` function performs as expected in handling the task takeover by a technician. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship Project
Function: `update_task`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `update_task` method, which allows a technician to update the stage of a task in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Task Update **Description:** Verify that a technician can successfully update the stage of a task with valid details.
Pre-condition: The user is a technician, and both the user and task exist.
Steps: - Send a POST request with `taskId`, `userId`, and `stageId`. - Retrieve the user and task based on the provided IDs. - Update the task in Odoo to change its stage. - Remove the task from the technician's MyTasks. - Redirect to the My Tasks page.

Expected Result: The task stage is updated in Odoo, removed from the technician's MyTasks, and a success message is displayed.
Actual Result: The task stage is updated in Odoo, removed from the technician's MyTasks, and a success message is displayed.
Status: Passed

Scenario 2: Task Update with Non-existent User **Description:** Verify that an appropriate message is shown if the user does not exist.
Pre-condition: The user ID provided does not correspond to an existing user.
Steps: - Send a POST request with `taskId`, `userId`, and `stageId`. - Attempt to retrieve the user based on the provided ID. - Display an error message if the user does not exist.

Expected Result: An error message "The user does not exist." is displayed.
Actual Result: An error message "The user does not exist." is displayed.
Status: Passed

Scenario 3: Task Update with Non-existent Task **Description:** Verify that an appropriate message is shown if the task does not exist.
Pre-condition: The task ID provided does not correspond to an existing task.
Steps: - Send a POST request with `taskId`, `userId`, and `stageId`. - Attempt to retrieve the task based on the provided ID. - Display an error message if the task does not exist.

Expected Result: An error message “The task does not exist.” is displayed.
Actual Result: An error message “The task does not exist.” is displayed.
Status: Passed

Scenario 4: GET Request to Update Task Page Description: Verify that the update task functionality is not accessible via a GET request.
Pre-condition: The user is authenticated and has technician rights.
Steps: - Send a GET request to the update task URL. - Attempt to access the functionality.

Expected Result: The GET request is not processed, and no action is taken.
Actual Result: The GET request is not processed, and no action is taken.
Status: Passed

Scenario 5: CSRF Protection Description: Verify that CSRF protection is enabled for the update task form.
Pre-condition: None
Steps: - Send a POST request to the update task URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 6: Authorization Check Description: Verify that only users with technician rights can access the update task functionality.
Pre-condition: The user is authenticated but does not have technician rights.
Steps: - Attempt to access the update task URL. - Check for authorization.

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

Scenario 7: Odoo Integration Error Handling Description: Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: The user has technician rights, and Odoo integration is functional.
Steps: - Send a POST request with `taskId`, `userId`, and `stageId`. - Simulate an error during Odoo task update. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 7
- **Passed:** 7
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `update_task` function performs as expected in allowing a technician to update the stage of a task. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship Project
Function: `edit_member`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `edit_member` method, which allows an admin or manager to edit user details in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Member Edit by Admin **Description:** Verify that an admin can successfully edit a member's details.
Pre-condition: The user is authenticated as an admin, and the member ID corresponds to an existing user.
Steps: - Send a POST request with updated member details. - Update the user details in Odoo. - Redirect to the appropriate page (manager members or technician members).

Expected Result: The user details are updated in Odoo and Django, and the admin is redirected to the appropriate page with a success message.
Actual Result: The user details are updated in Odoo and Django, and the admin is redirected to the appropriate page with a success message.
Status: Passed

Scenario 2: Successful Member Edit by Manager **Description:** Verify that a manager can successfully edit a member's details.
Pre-condition: The user is authenticated as a manager, and the member ID corresponds to an existing user.
Steps: - Send a POST request with updated member details. - Update the user details in Odoo. - Redirect to the technician members page.

Expected Result: The user details are updated in Odoo and Django, and the manager is redirected to the technician members page with a success message.
Actual Result: The user details are updated in Odoo and Django, and the manager is redirected to the technician members page with a success message.
Status: Passed

Scenario 3: GET Request to Edit Member Page **Description:** Verify that the edit member functionality is not accessible via a GET request.
Pre-condition: The user is authenticated as an admin or manager.
Steps: - Send a GET request to the edit member URL. - Attempt to access the functionality.

Expected Result: The GET request is not processed, and no action is taken.
Actual Result: The GET request is not processed, and no action is taken.
Status: Passed

Scenario 4: CSRF Protection **Description:** Verify that CSRF protection is enabled for the edit member form.
Pre-condition: None
Steps: - Send a POST request to the edit member URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 5: Authorization Check **Description:** Verify that only admins or managers can access the edit member functionality.
Pre-condition: The user is authenticated but does not have admin or manager rights.
Steps: - Attempt to access the edit member URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling **Description:** Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: Odoo integration is functional.
Steps: - Send a POST request with updated member details. - Simulate an error during Odoo user update. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `edit_member` function performs as expected in allowing an admin or manager to edit user details. All tested scenarios passed successfully. It is recommended to conduct further testing

under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship project **Function:** `add_to_manager_team`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `add_to_manager_team` method, which allows a manager to add a user to their team in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Addition to Manager's Team **Description:** Verify that a user can be successfully added to the manager's team.
Pre-condition: The user is authenticated as a manager, and the user ID corresponds to an existing user.
Steps: - Send a POST request with the user ID. - Retrieve the manager's team. - Add the user to the manager's team. - Update the team in Odoo. - Redirect to the users without a team page.

Expected Result: The user is added to the manager's team in Odoo and Django, and a success message is displayed.
Actual Result: The user is added to the manager's team in Odoo and Django, and a success message is displayed.
Status: Passed

Scenario 2: No Manager's Team Found **Description:** Verify that an appropriate message is shown if the manager's team does not exist.
Pre-condition: The user is authenticated as a manager, and the manager's team does not exist.
Steps: - Send a POST request with the user ID. - Attempt to retrieve the manager's team. - Display an error message if the team does not exist.

Expected Result: An error message "Unable to find your team." is displayed.
Actual Result: An error message "Unable to find your team." is displayed.
Status: Passed

Scenario 3: GET Request to Add to Manager's Team Page **Description:** Verify that the add to manager's team functionality is not accessible via a GET request.
Pre-condition: The user is authenticated as a manager.
Steps: - Send a GET request to the add to manager's team URL. - Attempt to access the functionality.

Expected Result: The GET request is not processed, and no action is taken.
Actual Result: The GET request is not processed, and no action is taken.
Status: Passed

Scenario 4: CSRF Protection **Description:** Verify that CSRF protection is enabled for the add to manager's team form.
Pre-condition: None
Steps: - Send a POST request to the add to manager's team URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 5: Authorization Check **Description:** Verify that only managers can access the add to manager's team functionality.
Pre-condition: The user is authenticated but does not have manager rights.
Steps: - Attempt to access the add to manager's team URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling **Description:** Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: Odoo integration is functional.
Steps: - Send a POST request with the user ID. - Simulate an error during Odoo team update. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `add_to_manager_team` function performs as expected in allowing a manager to add a user to their team. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., different user roles, network issues) to ensure robustness.

General Information **Project:** Internship Project
Function: `upload_model`
Date: May 30, 2024
Tested by: Rayane Belguebli

Summary The objective of this test was to verify the functionality of the `upload_model` method, which allows a manager to upload a model in a Django application. The tests were conducted to ensure that the method performs as expected under various scenarios. Most tests were successful, but some issues were identified.

Test Scenarios

Scenario 1: Successful Model Upload **Description:** Verify that a manager can successfully upload a model with valid details.
Pre-condition: The user is authenticated as a manager.
Steps: - Send a POST request with valid model details. - Save the model in the database. - Redirect to the home page with a success message.

Expected Result: The model is uploaded successfully, and a success message is displayed.
Actual Result: The model is uploaded successfully, and a success message is displayed.
Status: Passed

Scenario 2: Invalid Form Submission **Description:** Verify that an appropriate message is shown if the form submission is invalid.
Pre-condition: The user is authenticated as a manager.
Steps: - Send a POST request with invalid model details. - Display an error message due to the invalid form submission.

Expected Result: An error message "The form is not valid. Please correct the errors." is displayed.
Actual Result: An error message "The form is not valid. Please correct the errors." is displayed.
Status: Passed

Scenario 3: GET Request to Upload Model Page **Description:** Verify that the upload model functionality is accessible via a GET request.
Pre-condition: None
Steps: - Send a GET request to the upload model URL. - Attempt to access the functionality.

Expected Result: The GET request is processed, and the upload model page is displayed with an empty form.
Actual Result: The GET request is processed, and the upload model page is

displayed with an empty form.
Status: Passed

Scenario 4: CSRF Protection **Description:** Verify that CSRF protection is enabled for the upload model form.
Pre-condition: None
Steps: - Send a POST request to the upload model URL without a CSRF token. - Attempt to process the request.

Expected Result: The request is rejected due to missing CSRF token.
Actual Result: The request is rejected due to missing CSRF token.
Status: Passed

Scenario 5: Authorization Check **Description:** Verify that only managers can access the upload model functionality.
Pre-condition: The user is authenticated but does not have manager rights.
Steps: - Attempt to access the upload model URL. - Check for authorization.

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

Scenario 6: Odoo Integration Error Handling **Description:** Verify that an error message is shown if there is an issue with Odoo integration.
Pre-condition: Odoo integration is functional.
Steps: - Send a POST request with valid model details. - Simulate an error during model upload. - Catch the error and display an appropriate message.

Expected Result: An error message detailing the Odoo error is displayed.
Actual Result: An error message detailing the Odoo error is displayed.
Status: Passed

Summary of Results

- **Total Tests:** 6
- **Passed:** 6
- **Failed:** 0
- **Blocked:** 0

Conclusions and Recommendations The `upload_model` function performs as expected in allowing a manager to upload a model. All tested scenarios passed successfully. It is recommended to conduct further testing under different conditions (e.g., large file uploads, network issues) to ensure robustness.

Chapter 4

Odoo Deployment Guide

This guide outlines the steps to deploy Odoo 15 on a Linux server using Python virtual environments, systemd for service management, and Nginx as a reverse proxy.

Create Odoo Directory and Clone Repository First, create a directory for Odoo and clone the Odoo 15 repository from GitHub.

```
sudo mkdir /opt/odoo
sudo git clone https://www.github.com/odoo/odoo --depth 1 --branch 15.0
--single-branch /opt/odoo/odoo15
```

Set Up Python Virtual Environment Navigate to the cloned Odoo directory, create a Python virtual environment, and activate it.

```
cd /opt/odoo/odoo15
python3 -m venv venv
source venv/bin/activate
```

Install Odoo Dependencies Install the necessary dependencies listed in the `requirements.txt` file.

```
pip install -r requirements.txt
```

Configure Odoo Create and edit the Odoo configuration file to set up database credentials, paths, and other options.

```
sudo nano /etc/odoo.conf
```

Add the following content to the configuration file:

```
[options]
admin_passwd = admin
db_host = False
db_port = False
db_user = odoo
db_password = odoo
addons_path = /opt/odoo/odoo15/addons
xmlrpc_port = 8002
proxy_mode = True
```

Create Systemd Service for Odoo Create a systemd service file to manage the Odoo service.

```
sudo nano /etc/systemd/system/odoo.service
```

Add the following content to the service file:

```
[Unit]
Description=Odoo
Requires=postgresql.service
After=network.target postgresql.service

[Service]
Type=simple
User=odoo
Group=odoo
ExecStart=/opt/odoo/odoo15/venv/bin/python3 /opt/odoo/odoo15/odoo-bin -c /etc/odoo.conf
WorkingDirectory=/opt/odoo/odoo15
StandardOutput=journal+console

[Install]
WantedBy=default.target
```

Reload Systemd and Start Odoo Service Reload systemd to apply the new service, start the Odoo service, and enable it to start on boot.

```
sudo systemctl daemon-reload
sudo systemctl start odoo15
sudo systemctl enable odoo15
```

Configure Nginx as a Reverse Proxy Create an Nginx configuration file for Odoo.

```
sudo nano /etc/nginx/sites-available/odoo.conf
```

Add the following content to the configuration file:

```
server {
    listen 80;
    server_name rcm.esac.pt;

    access_log /var/log/nginx/odoo.access.log;
    error_log /var/log/nginx/odoo.error.log;

    location / {
        proxy_pass http://127.0.0.1:8002;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /longpolling/ {
        proxy_pass http://127.0.0.1:8072;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Cache static files
    location ~* /web/static/ {
        proxy_cache_valid 200 60m;
        proxy_buffering on;
        expires 864000;
        proxy_pass http://127.0.0.1:8002;
    }
}
```

Enable Nginx Configuration and Reload Nginx Create a symbolic link to enable the Nginx configuration, test the Nginx configuration for syntax errors, and reload Nginx.

```
sudo ln -s /etc/nginx/sites-available/odoo.conf /etc/nginx/sites-enabled/
```

```
sudo nginx -t
```

```
sudo systemctl reload nginx
```

This completes the Odoo 15 deployment process, with Odoo running as a systemd service and Nginx configured as a reverse proxy.

Chapter 5

Conclusion

In conclusion, our project represents a significant step forward in the field of industrial maintenance management. By leveraging Django's robust and scalable architecture, we have created a versatile web platform that seamlessly integrates with existing CMMS systems and augments technician workflows with AR technology.

Through the development of this platform, we have addressed the pressing need for efficient task and resource management in industrial settings. By providing managers with a centralized hub for planning and coordination, and technicians with real-time access to contextual information via AR glasses, we have significantly enhanced the efficiency and accuracy of maintenance operations.

Furthermore, our project offers immense potential for customization and adaptation to various industries and business requirements. The modular nature of our solution allows for easy integration with different CMMS systems and AR devices, ensuring flexibility and scalability as technology evolves.

Overall, this project has been a rewarding journey, allowing us to explore new technologies, contribute to innovative solutions, and acquire valuable skills in software development, project management, and industrial maintenance. We are excited about the impact our platform will have on improving maintenance processes and look forward to future opportunities for innovation and growth in this dynamic field

References

Fu, Chuang, Lu-Qing Ye, Yuan-Chu Cheng, Yong-Qian Liu, and Benoi Iung. 2002. "MAS-Based Model of Intelligent Control-Maintenance-Management System (ICMMS) and Its Application" 1: 376–80.

Kour, Ravdeep, Miguel Castaño, Ramin Karim, Amit Patwardhan, Manish Kumar, and Rikard Granström. 2022. "A Human-Centric Model for Sustainable Asset Management in Railway: A Case Study." *Sustainability* 14 (2): 936.

Malta, Ana, Mateus Mendes, and Torres Farinha. 2021. "Augmented Reality Maintenance Assistant Using Yolov5." *Applied Sciences* 11 (11): 4758.

Shankar, Lakshmi, Chandan Deep Singh, and Ranjit Singh. 2021. "Impact of Implementation of CMMS for Enhancing the Performance of Manufacturing Industries." *International Journal of System Assurance Engineering and Management*, 1–22.

Sorić, Dario. 2024. "How to Optimize Operations and Maintenance for Success." *Maintenance World*.