Factory Machinery Status & Repair Tracking System

EEN1037 Web Application Development - Group Project
Project Structure

Author: Alexandre Cathalifaud

March 2025

Group members

- **Maneesh Shankar**
- **Osgar Sullivan**
- **Kaushal Niranjan Agrawal**
- **Paul Hopkins**
- **Mehmet Battal**
- **Om Prakash Vengesanam Rajalingam Raja**
- **Jakub Wojtowicz**
- **Alexandre Cathalifaud**

Introduction

The aim of the **Factory Machinery Status & Repair Tracking System** project is to develop a web-based application that will enable real-time monitoring of the status of a factory's machinery, as well as management of the associated repairs. This platform will provide managers, technicians and repairers with an intuitive interface for efficiently supervising and organizing maintenance operations. The project is based on a robust architecture using **Django** for the backend and a modern frontend based on **HTML, CSS and JavaScript**.

General Architecture

The project architecture follows a client-server model where the backend, developed with **Django and Django REST Framework**, manages the business logic and the relational database (PostgreSQL). Unlike a pure API REST approach, the backend not only provides API endpoints, but also directly serves the frontend HTML pages by dynamically generating views via Django Templates.

The interaction between components follows a precise pattern:

- The frontend can be served directly by Django (backend) via integrated HTML templates.
- The backend manages a REST API that enables dynamic interaction with the database.
- Users access web pages dynamically generated by Django and perform actions that trigger requests to the API for data updates.
- The database stores all information on machines, breakdowns and users.

Team role organization

The project team is structured around different roles to ensure efficient management and a balanced distribution of responsibilities:

- **Project management, business requirements analysis and/or Web application structure design**: Alexandre Cathalifaud
- **Front-end development**: Maneesh Shankar
- **Client-side interactive features design**: Osgar Sullivan
- **Server-side development**: Kaushal Niranjan Agrawal
- **REST API development**: Paul Hopkins
- **Database design and connectivity**: Mehmet Battal
- **Testing and quality assurance**: Om Prakash Vengesanam Rajalingam Raja
- **Cloud deployment, maintenance, and scalability**: Alexandre Cathalifaud
- **Code repository management, and installation**: Alexandre Cathalifaud
- **Project documentation manager**: Jakub Wojtowicz
- **Project demonstration** : Jakub Wojtowicz

Key features

The application integrates a number of essential functions to guarantee optimal monitoring of machines and repairs.

Authentication and User Management

User authentication is handled by Django's session management system for HTML views, and by **Token Authentication** for the REST API. Each user must log in with a username and password. Django automatically manages user sessions and associated permissions for the web part, while the REST API is protected by a token system.

The following roles are defined:

- **Manager**: overall system supervision and assignment management.
- **Technician**: fault reporting and machine status updates.
- **Repair**: intervention on repairs and closure of trouble tickets.
- **View-only**: consultation of information without being able to modify it.

For users accessing the REST API, authentication is based on **Token Authentication**. Each user must generate a unique authentication token in order to interact with the API.

Machine Management

Each machine is referenced in the database and can be added, modified or deleted by an authorized user. Its status can be defined as **OK**, **Warning** or **Fault**. Warning notes can be added to signal anomalies before a major problem occurs. Machines can also be assigned to specific technicians and repairers, making it easier to track interventions.

Troubleshooting and repairs

When a machine encounters a malfunction, a **fault ticket** is generated, containing details of the nature of the problem, as well as images if necessary. Repairers can then add notes to document their intervention. Once the repair has been completed, the ticket is closed and the machine's status is updated.

Dashboard and Data Visualization

The system features a dashboard providing an overview of machine status. Interactive graphs, generated via **Chart.js** or **D3.js**, facilitate analysis of trends and recurring incidents. Managers can also export detailed reports in **CSV** or **PDF** format.

Deployment and Development Environment

To ensure efficient and reproducible deployment, the application is based on **Docker**. Since the frontend is served directly by Django, a single **Dockerfile** is required for the backend,

encompassing management of HTML views and the REST API. A **docker-compose.yml** file is used to orchestrate all services, including the database and REST API.

Development follows a rigorous GitHub-based process. Code is versioned using a structured branching system (`main`, `dev`, `feature/*`). Tests are automated via **GitHub Actions** to guarantee code quality and prevent regressions.

Documentation and Deliverables

Full documentation is provided to ensure understanding and maintenance of the project. It includes:

- Weekly meeting minutes.
- An `README` file detailing installation and operating instructions.
- In-depth documentation on architecture and technical choices.
- A user manual for end users.
- A demonstration in the form of a screencast or annotated screenshots.
- An individual report from each member describing his or her contributions.