DCU EEN1037 2025 - Group Project: Factory Machinery Status & Repair Tracking System
Group G

Team member:

Name	Email
Mohammed AL shuaili	mohammed.alshuaili2@mail.dcu.ie
Ronghui Lin	ronghui.lin2@mail.dcu.ie
Vennela Malavika Bada	vennelamalavika.bada2@mail.dcu.ie
Ravi Peddi	ravi.peddi2@mail.dcu.ie
Dhanush Ramesh	dhanush.ramesh3@mail.dcu.ie
Rishabh Yadav	rishabh.yadav2@mail.dcu.ie
Vignesh Sivaprakasam	vignesh.sivaprakasam2@mail.dcu.ie
Sahana Sankaralingam	sahana.sankaralingamselvi2@mail.dcu.ie

Module: Web Application Development

Date : 13/04/2025

Table of content

1. Introduction (Project Overview)

The Factory Machinery Status & Repair Tracking System is designed for industrial manufacturing environments, specifically targeting high-precision production facilities like ACME Manufacturing Corp., which specializes in Printed Circuit Board (PCB) manufacturing. The application operates in a domain where real-time equipment monitoring, maintenance coordination, and operational efficiency are critical. It addresses challenges such as machinery downtime, repair workflow management, and status visibility across complex production lines. By integrating with factory operations, the system bridges the gap between physical machinery and digital tracking, enabling seamless communication between technicians, repair teams, and management.

Key industry-specific elements include:

- PCB Manufacturing Machinery: CNC drilling machines, lamination presses, soldering systems, and automated optical inspection (AOI) equipment.

- Operational Requirements: Strict adherence to safety protocols, minimal downtime, and compliance with industrial automation standards.

Purpose & Target Audience:

The Factory Machinery Status & Repair Tracking System is designed to centralize real-time monitoring and repair workflows for industrial machinery, enabling technicians to log faults, repair teams to resolve issues efficiently, and managers to oversee equipment health via dashboards and reports. Targeting ACME Manufacturing Corp., the system serves Technicians (fault reporting), Repair Personnel (case resolution), and Managers (analytics/assignments), while supporting integration with external IoT systems via REST APIs. By streamlining maintenance operations and reducing downtime, the platform enhances productivity across PCB production lines.

2. Technical Requirements Gathering

Functional Requirements:

2.1 User Registration

Name	Employee Registration
Description	Allows the IT staff to register new employee to use the application
Criticality	Critical - employee to need to registered in order to access their role
Technical Issues	Employee information must be secure, so the database needs to be protected.

Dependencies with other Requirements	

2.2 User Login

Name	User Login
Description	Allows users with existing username and password to log in.
Criticality	High - (users will not be able to see machine status or ticket status without logging in.)
Technical Issues	User information must be secure, so the database needs to be protected.
Dependencies with other Requirements	User Registration

2.3 Machine Status Tracking

Name	Machine Status Tracking
Description	Dashboard that gives an overview of the status of machines for logged in users
Criticality	It is a core feature of the webapp so we will need to ensure that this feature is working as intended.
Technical Issues	Dependent on database and API design.
Dependencies with other Requirements	User Registration and User Login

2.4 Employee Management

Name	Employee Management
Description	Administrators will be able to manage employee accounts from this page
Criticality	High - Allows for manual account creation and deletion as well as management
Technical Issues	Database design and API
Dependencies with other Requirements	User Login

2.5 Ticket Submission

Name	Ticket Submission
Description	Allows for user submission of tickets related to the service.

Criticality	High - In order for users to report faults or issues that cannot be resolved by themselves
Technical Issues	Will need to save attachments
Dependencies with other Requirements	Ticket Tracking

2.6 Ticket Tracking

Name	Ticket Tracking
Description	Used to see status of tickets overall
Criticality	High - Would be the only way of resolving issues reported by issues.
Technical Issues	Linked with ticket submission, requires submission checking
Dependencies with other Requirements	Ticket Submission

2.7 Ticket Comments

Name	Ticket Commenting
Description	Allows users to add comments to tickets for clarification or communication.
Criticality	High - Only way to communicate for tickets
Technical Issues	Tracking and viewing media files (if attached)
Dependencies with other Requirements	Ticket Submission, Ticket Tracking

3. Technology Selection and Functionalities

Static Pages & Navigation

- **Static pages** provide essential non-interactive content that users need to understand the company, its services, policies, and other foundational information. These pages serve as the core informational sections of the website, allowing users to navigate easily between sections such as company details, services, and contact information.

- The **navigation menu** is an integral part of the static pages. It allows users to easily navigate between the different sections of the application. The navigation menu dynamically adjusts based on whether the user is logged in,

showing options like **Submit Ticket** and **My Tickets** to authenticated users.

- **Key Files**:
    - **about.html**: Contains information about the company, including its history, values, and mission.
    - **contact.html**: A page where users can find contact information or submit a contact form.
    - **faq.html**: Frequently asked questions and their answers, providing clarity on common inquiries about the system.
    - **policy.html**: Contains the privacy policy and terms of service, ensuring legal compliance.
    - **catalog.html**: A catalog listing the company's products or services, depending on the context.
    - **services.html**: A detailed page outlining the services provided by the company and how to use the system.

Menu Bar with Consistent Navigation (Dynamic Menu)

- The **menu bar** provides a consistent and intuitive navigation experience across all pages. The dynamic nature of the menu allows it to adjust based on the authentication status of the user.
- For instance, when a user is logged in, the menu may display options like **Submit Ticket** and **My Tickets**. If the user is a **Manager**, additional menu options like **Create User** or **Manage Tickets** will appear.
- This ensures that users only see the relevant options available to their role, enhancing the user experience and maintaining security by restricting access based on roles.

Client-Side JavaScript

- **JavaScript** is used to enhance the interactivity of the web application, providing a seamless experience for users. It runs on the client-side, meaning it interacts directly with the user without needing a round-trip request to the server.

- Key functionalities that JavaScript handles include form validation, event handling, dynamic content updates, and animations, all of which contribute to a more dynamic and user-friendly interface.

Key Functionalities:

- **Form Validation**: Ensures that data entered by the user is valid before submission, preventing errors and ensuring data integrity.
- **Event Handling**: Manages interactions such as clicks, form submissions, or hover actions. For example, when a user submits a ticket or interacts with the dashboard, JavaScript triggers the necessary actions.
- **Dynamic Content**: Dynamically updates content on the page. For instance, it can show real-time ticket statuses, filter machinery data, or display charts that represent machine performance or fault statuses.
- **Animations**: The **AOS (Animate On Scroll)** library is used to animate elements on the page as users scroll, making the user interface more engaging and visually appealing.

Dynamic Server-Side Pages

- **Dynamic pages** are essential for interacting with the database and displaying real-time data based on user interactions. These pages fetch and display dynamic content, ensuring that the application always shows the most up-to-date information.

Key Pages:

- **index.html**: The home page, which adapts its content based on the user's role. For example, a **Manager** may see an overview of the entire system's machinery and performance metrics, while a **Technician** sees only their assigned tickets.
- **machine_status.html**: Displays the operational status of different machines. It provides a real-time update on the health of machinery, including warnings, faults, and repairs.
- **ticketdash.html**: The ticket dashboard where users can manage and view their fault tickets. Technicians can see their open tickets, and Managers can assign and review

tickets.

- **submit_ticket.html**: Allows users or technicians to submit fault tickets for machinery issues. This page captures information like machine status, fault description, and priority.

Tables generation and key Files:

- **models.py**: Defines the data models for the system, including the database schema for **Machinery**, **Faults**, **Warnings**, and **Users**. These models define the structure and relationships between the data stored in the database.
    - **Examples**:
        - **Machine**: Stores data about each machine, such as its name, model, and operational status.
        - **FaultCase**: Represents fault tickets created for machinery, including descriptions, priority, and status.
        - **User**: Manages user accounts, including role-based access (Technician, Manager, etc.).
- **serializers.py**: Converts complex data types (like database querysets) into JSON format for API responses. It ensures that data can be sent efficiently to the frontend.
- **views.py**: Handles logic for retrieving and displaying data. Views fetch data from the database and pass it to templates or APIs to render dynamic content.
- **api_views.py**: Manages the REST API endpoints, allowing external systems (e.g., IoT devices) to interact with the application, such as submitting fault data or retrieving machine status.

User Authentication

- **Why?**
    - **User authentication** is critical for ensuring that only authorized users can access specific areas of the application based on their roles. It also protects

sensitive data and ensures proper access control for different functionalities.

- **Roles**:
    - **Technician**: Can create and manage fault tickets for assigned machinery.
    - **Repair Personnel**: Can update the status of fault tickets and add repair notes.
    - **Manager**: Has full control over the system, including user management, assigning roles, and generating reports.
    - **View-only**: Can only view machine statuses and reports without the ability to modify any data.
- **Key Files**:
    - **signin.html and signup.html**: Provide login and registration forms for user authentication.
    - **urls.py**: Contains the URL routes for login and registration actions.
    - **views.py**: Contains the views responsible for handling login, registration, and logout functionality.

Docker Setup

- **Docker** is used for containerizing the application, ensuring consistency across different environments and simplifying deployment. Docker allows the application, its dependencies, and the database to run in isolated containers, making it easier to deploy on different systems without configuration issues.

Key Files:

- **Dockerfile**: Defines the instructions for building the Docker image for the Django application. It installs necessary dependencies, sets environment variables, and prepares the application for deployment.
- **docker-compose.yml**: Manages multiple Docker containers, including the Django application and the PostgreSQL database. It makes it easy to bring up all required

services with a single command.

- **init.sql and db_backup.sql**: Provide the initial database schema and sample data to set up the application's database when the Docker container is started.

Key Functionalities:

- **Automated SQL Migrations**: Docker ensures that database migrations are automatically applied when the container starts, keeping the database schema up to date.
- **Environment Variables**: Sensitive data like database URLs, credentials, and API keys are stored in Docker environment variables for added security and flexibility.
- **Persistent Storage**: Docker volumes are used to store data such as database records and uploaded files. This ensures data persistence even when the Docker container is stopped or restarted.

4. Code Management Decisions (repo management)

Version Control: GitHub

Effective code management and version control are vital in software development for team collaboration and productivity and Github is the dominant software in this case. It is arguably the industry standard, particularly with the open-source community. We chose github as it has excellent features, with pull requests, inline commenting, review approvals, status checks and issue tracking all inbuilt to github.

It also has an extensive integration ecosystem, it provides native Continuous Integration/ Continuous Deployment (CI/CD) which enables automation of build, test and deployment pipelines, it is also cloud hosted, which provides us with high availability and allows us to access our code anywhere, the UI is also user friendly and allows us to search the code itself and view the history of our commits.

Commit Guidelines:

Commit atomicity: Each commit should represent a small logical change (like fixing a specific bug).
Commit Messages: These messages need to be clear, it should give a quick overview of what the change is and why.
Frequent Commits: Each Commit should also keep the codebase in a functional state (especially to main). This will help everyone as it will prevent us from being affected by a broken state, also commit work in progress features to local branches if possible before creating a push request to merge to main.

5. Roles & Responsibilities

Team Member Names

Mohammed
Ronghui
Vennela
Ravi
Sahana
Dhanush
Vignesh
Rishabh

Final Assigned Roles (Backend, Frontend, Testing, Documentation, etc.)

Meeting Manager: Mohammed
Front-end: Sahana, Mohammed, Ravi, Vennela, Ronghui
Backend: Vignesh, Dhanush
Database design: Dhanush,Vignesh
Client Side Interactive Design: Sahana, Mohammed
Testing & QA
- Adhoc testing: Shared
- User Testing: Shared

Documentation: Shared
- Minutes: Ronghui

REST-API Development + Documentation: Rishabh, Vignesh
Code Repository Management: Ronghui

6. Meeting Minutes

Week 9 14/03/2025 (Kick off meeting)

- Assigning initial group roles
  Meeting Manager: Mohammed
  Front-end: Sahana, Mohammed, Ravi, Vennela, Ronghui
  Backend: Vignesh, Dhanush, Ronghui
  Database design: Dhanush
  Client Side Interactive Design: Sahana, Mohammed
  Testing & QA
  - Adhoc testing: Shared
  - User Testing: Vennela

  Documentation: Shared
  - Minutes: Ronghui

  REST-API Development + Documentation: Rishabh, Vignesh
  Code Repository Management/Maintenance: Shared

- Discussed initial requirements for project
  Created github repo, shared meeting documentation.

(https://github.com/Ronghui-Lin/ee1037-groupg)
No Cloud Deployment (local docker deployment)

- Initial Design Plan
  7 Static, 7 Dynamic pages
  Discussed role based access & database design (Postgres)
  REST-API (potentially use Express)
  Discuss more details on what is on each page next meeting.

  -Next meeting will decide on overall design of each page, layout then discuss the initial API requirements based on the information required from each page.

-Final project documentation write up will be shared between everyone.

Week 10 18/03/2025

- Discussed the initial details of each static and dynamic page.
  Decided on front end requirements.
  Static Pages
    - Home Page
    - About Us
    - Contact Us
    - Privacy Policy
    - FAQ
    - Machinery Catalog
    - Services Page(Read Me)

  Dynamic Pages
    - Role–Based Dashboard
    - Machine Status & Details
    - Fault Case Management
    - Warning Management
    - User Management
    - API Test interface
    - Ticket Creation, Logging & Editing

- The Frontend team will have a meeting later to decide on the overall design and theme of the web application and have initial design plans to present at the next meeting.
- Backend team will create rough numbers and plans for integrating the API and the required database tables to be presented at the next meeting.
- Work splitting and possible conflicts between the two plans will be resolved at next meeting before active development begins

Week 11 24/03/2025

- Dashboard Parallax Idea
- Discussed plans created by frontend and backend team
    - Resolved any conflicts and any missing information required by each team
- Development starts on frontend
- Logo Discussion
- Backend team to start implementing the rough backend

Week 12 04/04/2025

- Original Meeting scheduled 01/04/2025, conflict in scheduling, pushed to 04/04/2025
- Finishing off dynamic pages on frontend
- Backend largely complete, just waiting to link between frontend and backend
- Finalising the specifics in what “machines” are being tracked and what is manufactured

- Role Based Details:
    - Administrator
        - Access to ALL pages, except user management
        - Can also move reassign tickets
    - IT Staff
        - Cannot reassign tickets, access user management
        - Can access everything else
    - Developers/Technicians
        - Access assigned tickets, modify status of the ticket
    - User
        - Create Faults/tickets, View assigned tickets

Week 13 10/04/2025

- Finishing touches on the web application
- Few remaining tasks divided up
- Rishabh: Continue working on about us page to include company story, pictures and more details on the products.
- MalavikaVennela: Change the service page to talk about the company's services in general and how to use the website and functionalities of the web application.
- Dhanush: Finish the backend for role based access
- Ravi, Sahana, Dhanush, Rishabh: Frontend for role based page
- Ronghui: Double check links in URL and VIEWS, add additional comments where required, testing and QA on the ticketing system
- Mohammed: Recording the demonstration video
- Malavika Vennela, Ravi work on documentation.
- Collectively work on authentication and display, change the nav bar depending on the user.

7. Installation & Setup Instructions

Prerequisites:

Make sure the following software is installed on your system:

1. **Docker**:
    - Docker is required to run the application. You can install Docker by following the instructions here.
2. **Docker Compose**:

- Docker Compose is used to define and run multi-container Docker applications. It should be included when you install Docker, but you can verify it by running:

  docker-compose --version

3. **Python**:

   - Although Docker handles the Python environment, you will need **Python** installed to manage dependencies locally (for your virtual environment or local testing).

   - **Django** and other dependencies will be installed through Docker, so no need to install them manually.

   Step-by-Step Installation Guide:

   - **Clone the repository**: First, clone the repository to your local machine:

     git clone <repo_url>

1. Replace <repo_url> with the actual URL of the project repository.

   - **Set up a virtual environment** (optional): If you want to set up a virtual environment for development (not required for Docker), follow these steps:

     cd <project_directory>
   - python -m venv venv
   - source venv/bin/activate # On macOS/Linux
   - venv\Scripts\activate # On Windows
   - pip install -r requirements.txt # Install dependencies locally
   - Python manage.py makemigrations
   - Python manage.py migrate
   - Python manage.py runserver
   - Note: Ensure your database settings in mysite/settings.py use localhost as the host.

2. However, since the project is **dockerized**, these steps may not be necessary for running the project via Docker.

3. **Install Docker dependencies**:

   - The required Docker services and configurations are defined in the docker-compose.yml file, so you only need Docker to run the project.

4. **Clean up Docker containers (if necessary)**:

   - Before building the Docker containers, you might want to ensure that old volumes and containers are removed to avoid conflicts:

     docker-compose down --volumes --remove-orphans

-

5. **Build the Docker container**:
    - Now, build and start the Docker containers:

      docker-compose up --build
    - This command will pull any necessary Docker images, build the project, and start the containers.

6. **Access the application**:
    - Once Docker finishes building the containers, you can access the application at http://localhost:8000/ or the provided URL in your browser.

Running the Application Using Docker:

- **Bring down the Docker containers** (if you need to reset or shut down the application):

  docker-compose down --volumes --remove-orphans

7. **Start or rebuild the application**: To build and start the containers (if not already running),

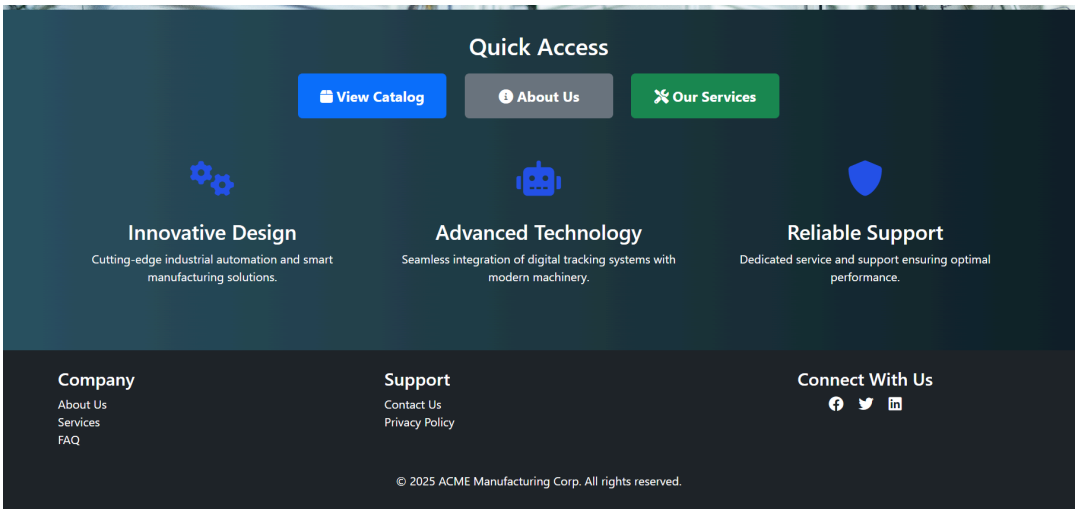
   docker-compose up --build

8. Walkthrough of Key Functionalities (explanation of all pages)

Home Page Overview

The Home Page for ACME Manufacturing Corp is designed to introduce the company and highlight key features of their services. It is built with Bootstrap for responsiveness and Font Awesome for icons. The page includes several key sections such as navigation, hero section, quick links, features, and footer. The page is interactive, responsive, and visually appealing, incorporating animations using **AOS**.

The navigation bar includes menu items like Home, Catalog, Machinery Status, Employee Management, and Ticket Dashboard, with additional options such as Submit Ticket and My Tickets appearing when the user is authenticated. A Sign In/Sign Out button adjusts based on login status, while superusers/managers get a Create User link. The hero section features a full-width introduction with the title "Welcome to ACME Manufacturing Corp.," a subtitle about innovative automation, and a Learn More button linking to the About Us page. The background image of an engineer is dynamically loaded for high resolution. Quick Links provide animated access to key pages like the catalog and services, while the Features section highlights innovative design, advanced technology, and reliable support, each with an icon and description.

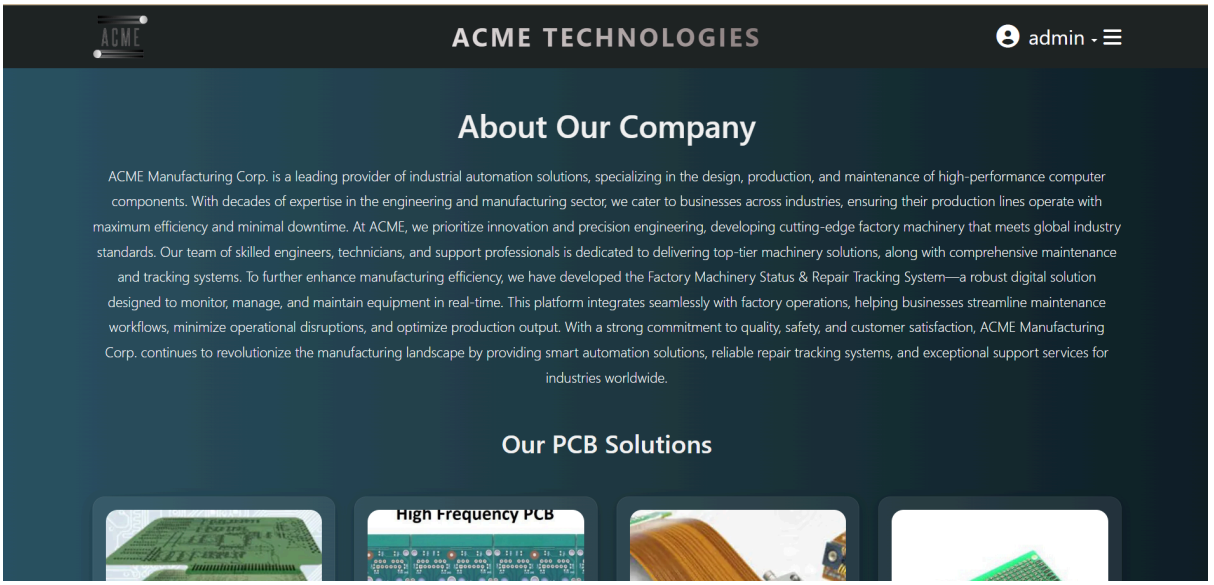


The footer contains company, support, and social media links, along with copyright information. JavaScript enhances the page with scroll animations (AOS) and dynamic image loading. Stylesheets and scripts include custom CSS, Bootstrap for responsiveness, AOS for animations, and custom JavaScript for the hero section’s background image. The design ensures an engaging, interactive experience with smooth animations and high-quality visuals.

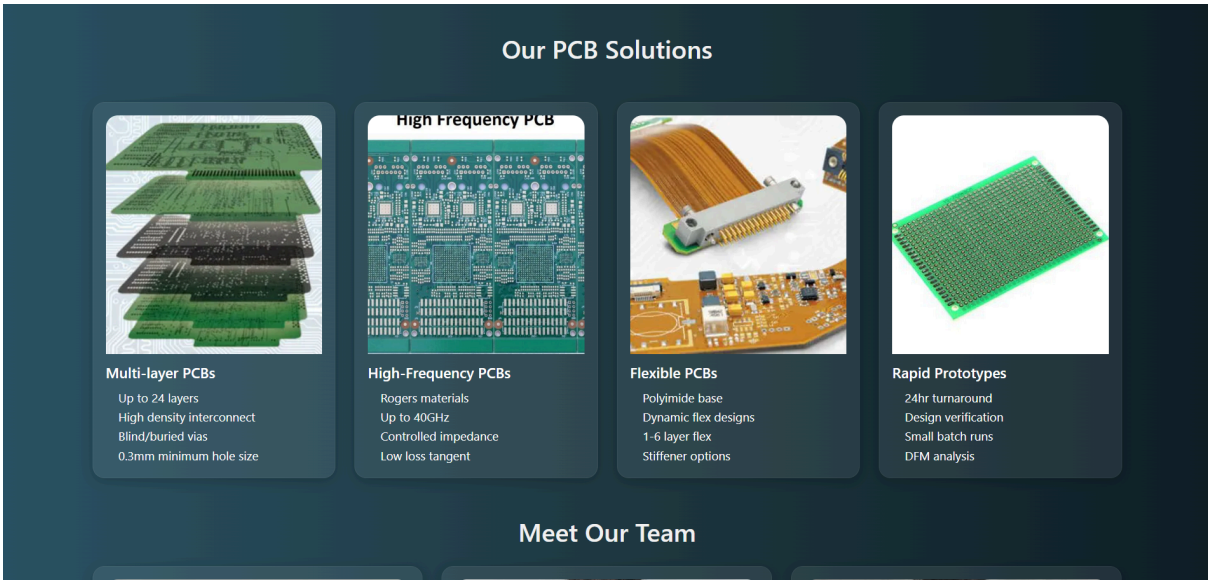
About us page:

The "About Us" page for ACME Manufacturing Corp. introduces the company and showcases its services, products, and team. Built using Bootstrap for responsiveness and Font Awesome for icons, the page is divided into three main sections:

1. **Company Overview**: A detailed description of ACME's expertise in industrial automation, highlighting its innovative machinery solutions and the Factory Machinery Status & Repair Tracking System designed to improve manufacturing efficiency.

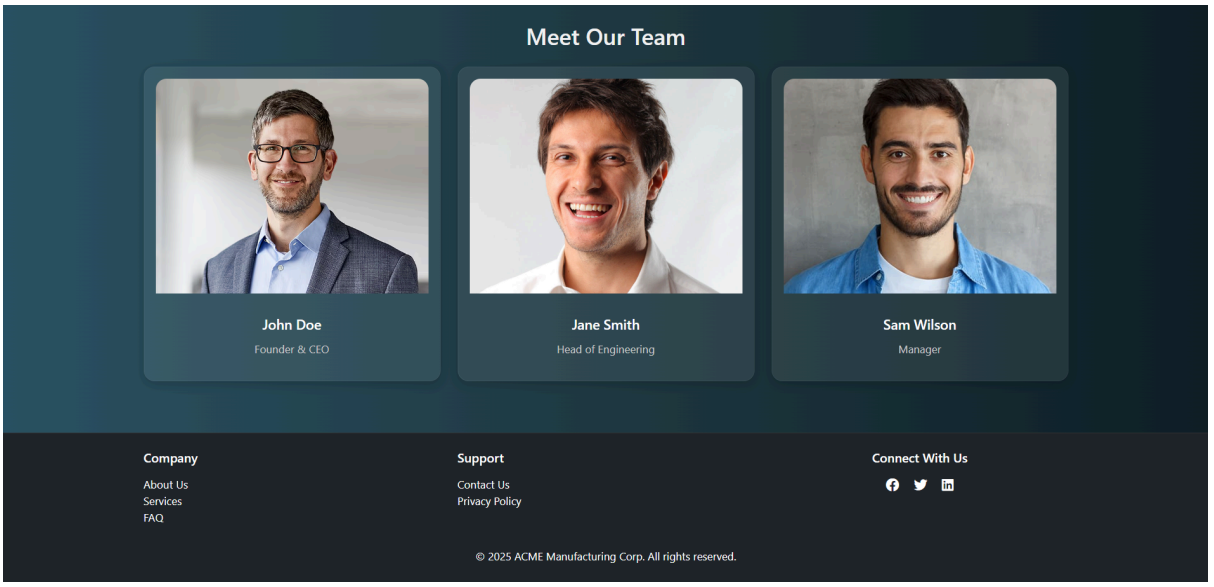


2. **PCB Solutions**: This section displays ACME's key PCB products like Multi-layer, High-Frequency, Flexible, and Rapid Prototypes. Each product is shown in responsive cards with specifications to help users understand the offerings.



3.

4. **Meet Our Team**: Profiles of key team members, including the Founder & CEO, Head of Engineering, and Manager, are presented in a visually appealing layout, building trust with visitors.

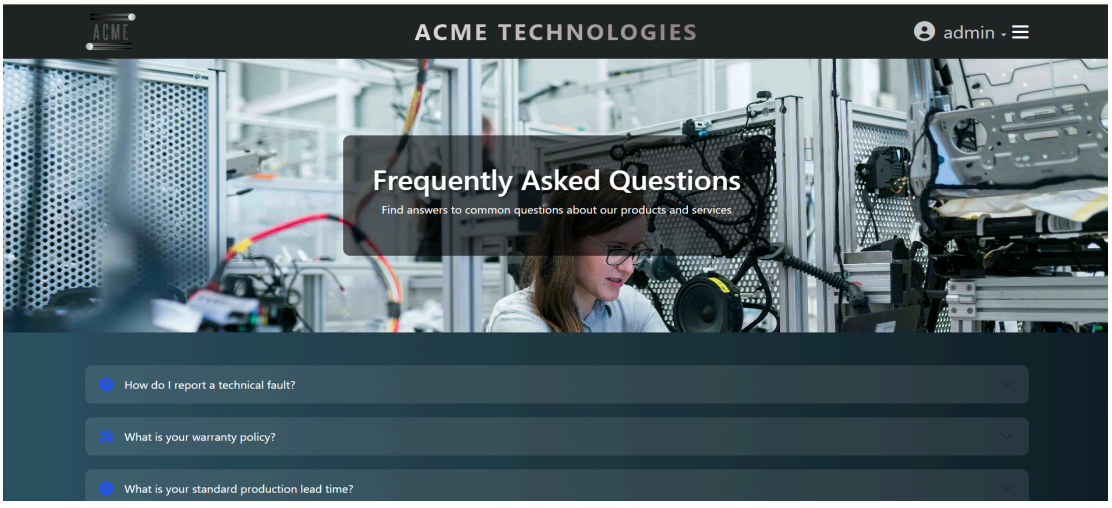


The page is enhanced with interactive elements, including a dynamic navigation bar, smooth hover effects on product and team cards, and AOS (Animate On Scroll) animations for a modern user experience. The footer includes links to important company pages and social media icons for easy connection. Overall, the page provides a clean, engaging, and informative experience for users while showcasing ACME Manufacturing Corp.'s expertise and leadership.

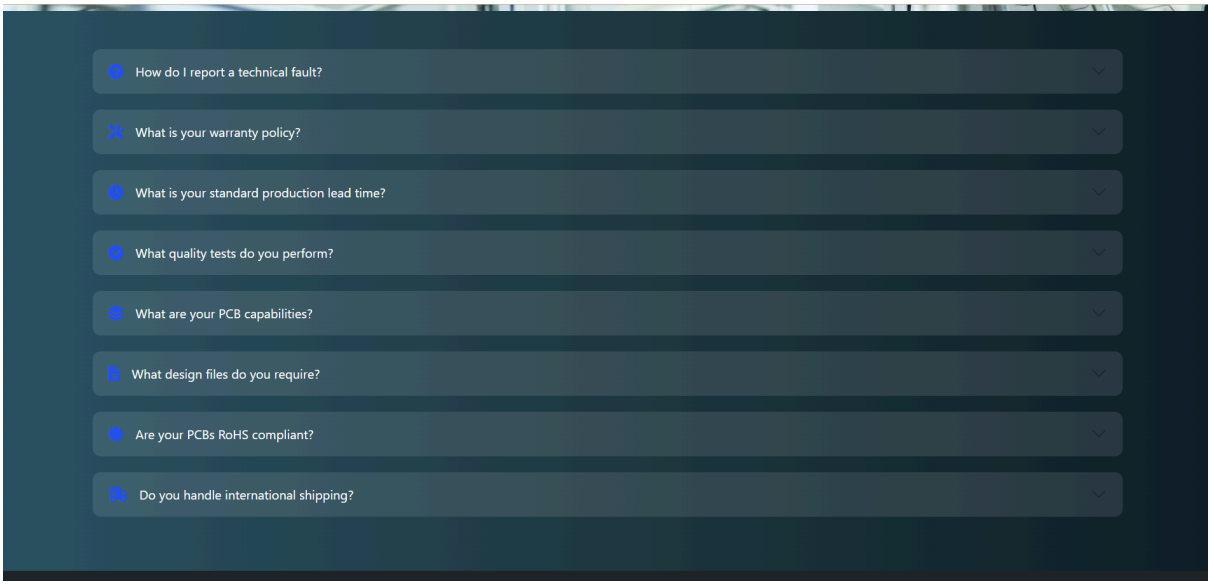
FAQ page:

The "FAQ" page for ACME Manufacturing Corp. is designed to provide answers to common questions about their products and services. Built with Bootstrap for responsiveness and Font Awesome for icons, the page features a clean, interactive accordion layout for easy navigation.

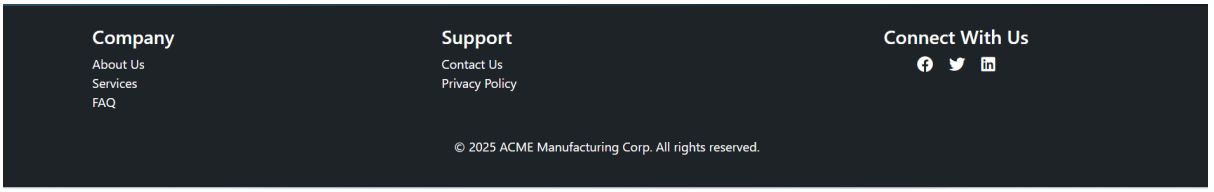
1. **Hero Section**: A visually engaging section with a background image of an engineer, introducing the FAQ topic with a transparent overlay and centered text.



2. **FAQ Accordion**: Key questions are displayed with expandable answers, covering topics such as reporting technical faults, warranty policies, production lead times, PCB capabilities, and international shipping. Each question features an icon for better user interaction.



3. **Footer**: Includes links to important company pages and social media icons for easy connection.

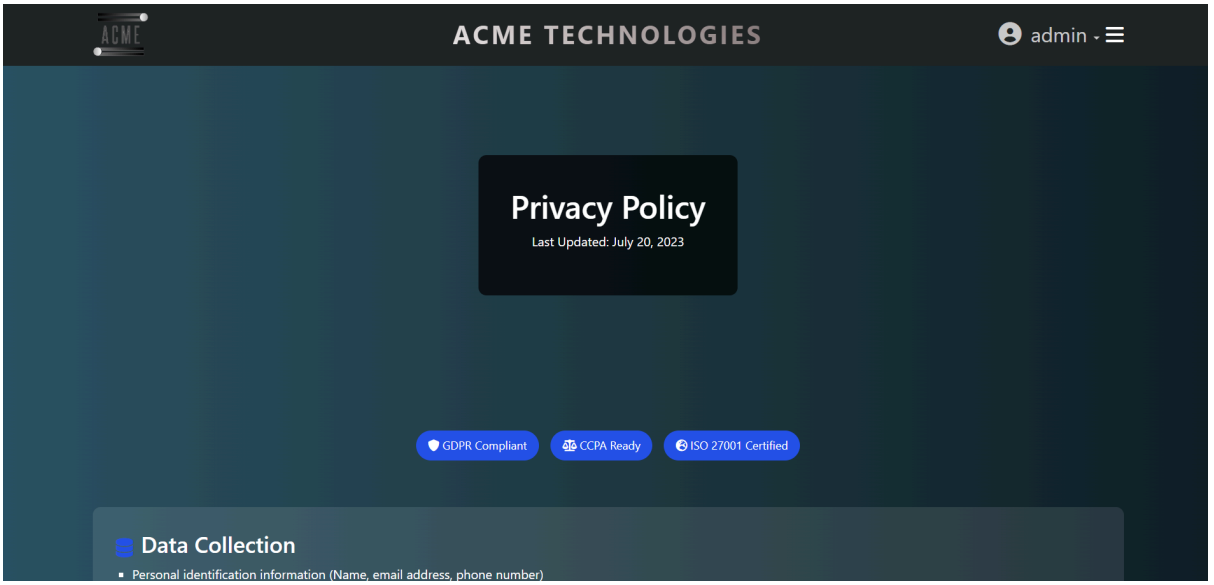


The page is styled with a dark gradient background and animations powered by AOS (Animate On Scroll), creating a smooth, modern user experience.

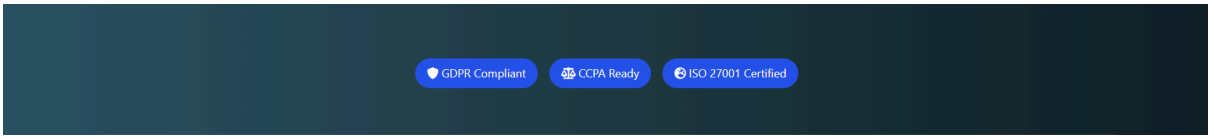
Policy privacy page:

The "Privacy Policy" page for ACME Manufacturing Corp. outlines the company's approach to data collection, usage, sharing, and security measures. It is designed with Bootstrap for responsiveness and includes Font Awesome icons for visual clarity.

1. **Hero Section**: The page begins with an engaging hero section that features a background image and an overlay with the title "Privacy Policy" and the last updated date.

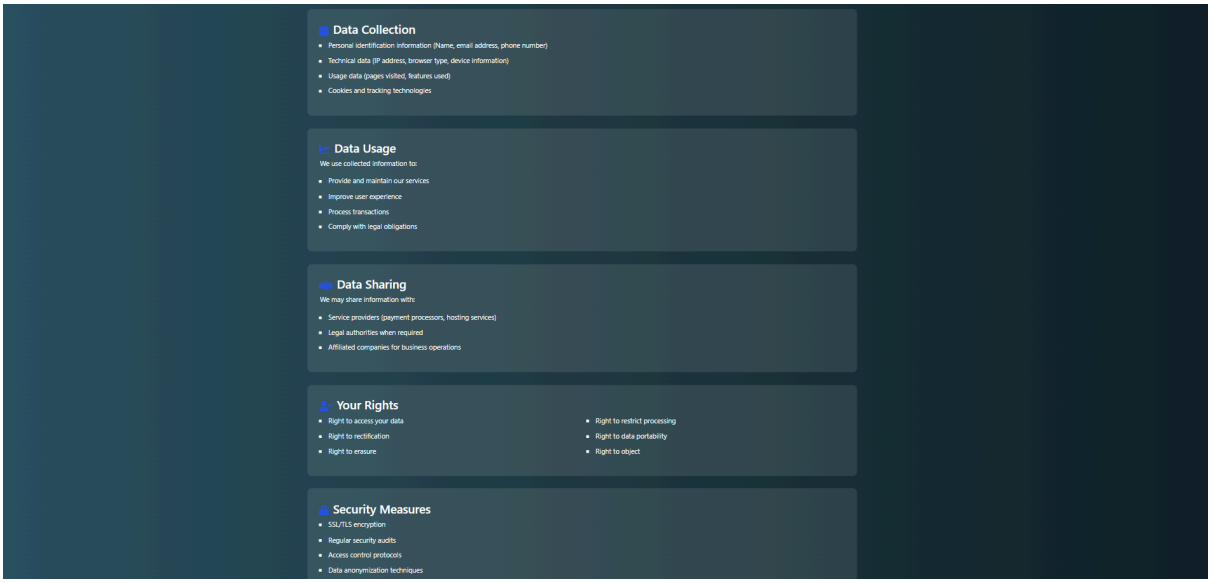


2. **Compliance Badges**: The page highlights ACME's commitment to privacy regulations by displaying badges for GDPR compliance, CCPA readiness, and ISO 27001 certification, making users feel assured of the company's security standards.

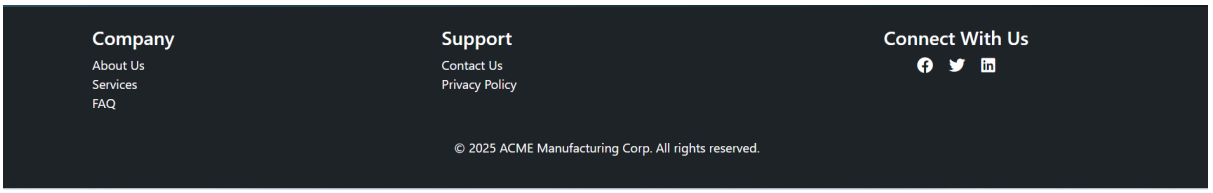


3. **Policy Sections**:

    - **Data Collection**: Lists types of data collected, including personal, technical, usage data, and cookies.
    - **Data Usage**: Explains how collected data is used for service provision, user experience improvement, transaction processing, and legal compliance.
    - **Data Sharing**: Details who the data may be shared with, such as service providers and legal authorities.
    - **User Rights**: Outlines users' rights, including access to data, rectification, erasure, and data portability.
    - **Security Measures**: Describes the company's security protocols, including SSL/TLS encryption and data anonymization techniques.
    - **Contact Information**: Provides contact details for privacy concerns and data requests, including email and phone.



4. **Footer**: The footer includes links to the company's important pages and social media icons for easy connection.



The page is styled with a dark gradient background and interactive hover effects, ensuring a visually appealing and easy-to-navigate experience.

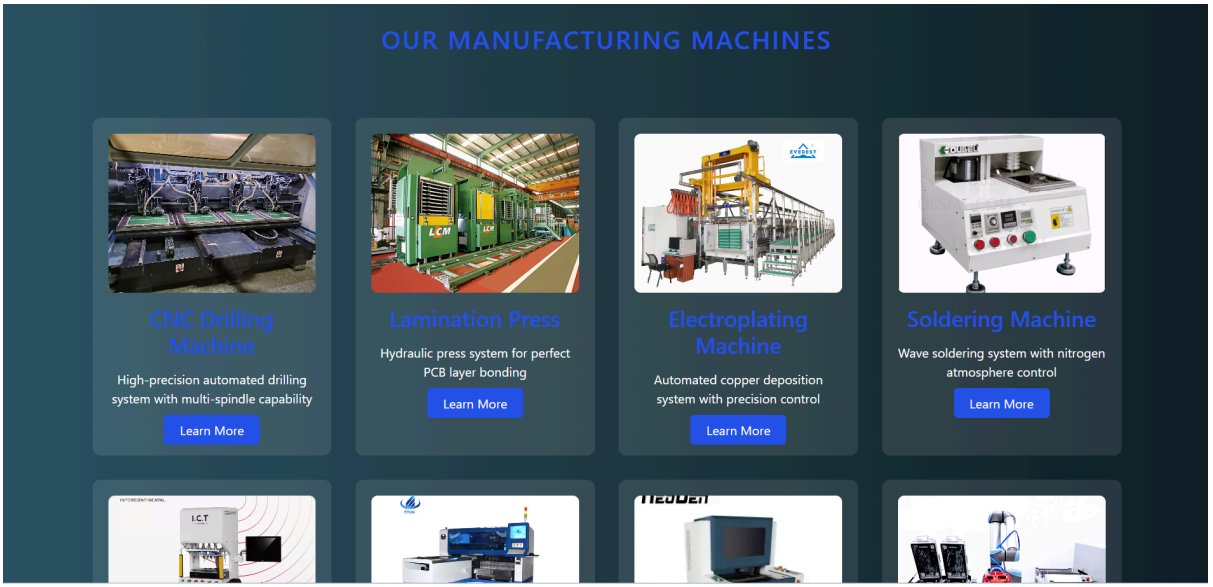
Catalog page:

The "Industrial Machinery Catalog" page for ACME Manufacturing Corp. presents a detailed catalog of machinery for PCB manufacturing, designed to offer both technical specifications and interactive features for users.

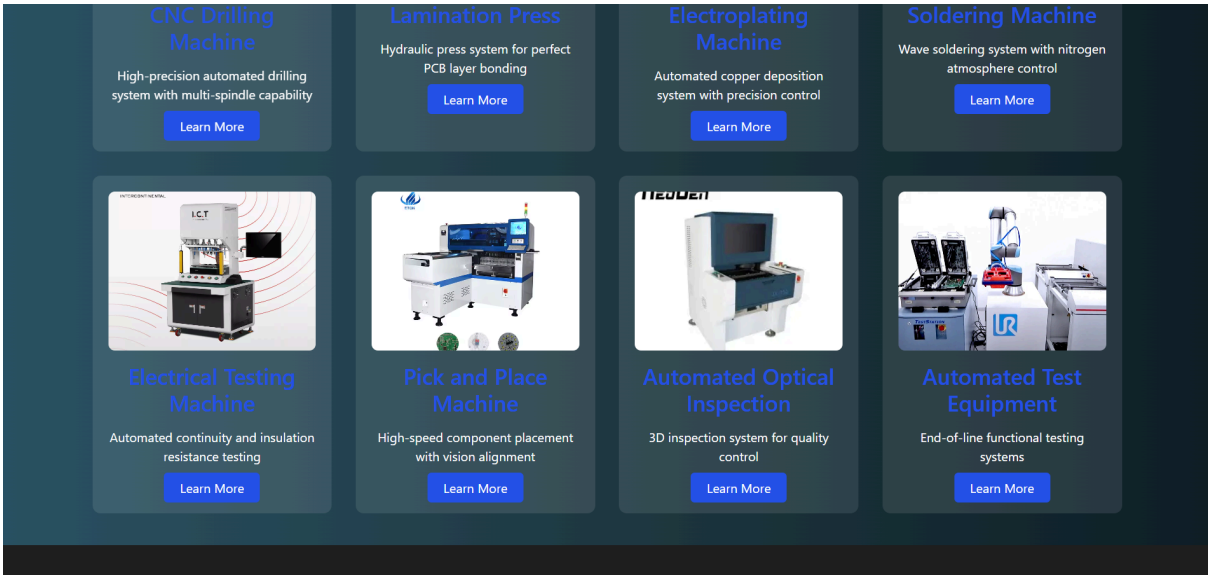
1. **Hero Section**: The page begins with a hero section featuring a background image of a factory, with an overlay that introduces the catalog and emphasizes the premium nature of the machinery offered.

2. **Product Grid**: The catalog is displayed in a responsive grid layout, featuring a variety of machinery cards, each with an image, short description, and a "Learn More" button. Products showcased include:

    - **CNC Drilling Machine**: High-precision automated drilling system.
    - **Lamination Press**: Hydraulic press for PCB layer bonding.
    - **Electroplating Machine**: Precision copper deposition system.
    - **Soldering Machine**: Wave soldering system with nitrogen control.

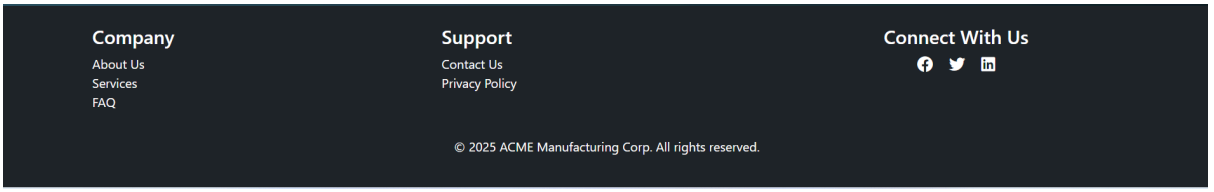


- **Electrical Testing Machine**: Automated PCB quality assurance.
- **Pick and Place Machine**: High-speed component placement system.
- **AOI Machine**: 3D optical inspection for quality control.
- **Automated Test Equipment**: End-of-line functional testing system.



3. **Machine Details Modal**: Each product card has a "Learn More" button that opens a modal with detailed machine specifications, key features, and images. The modal also includes a description of the machine and its technical specifications, giving users a deeper understanding of each product.

4. **Footer**: The footer includes navigation links to important company pages like "About Us," "Services," and "FAQ," as well as social media icons for user engagement.

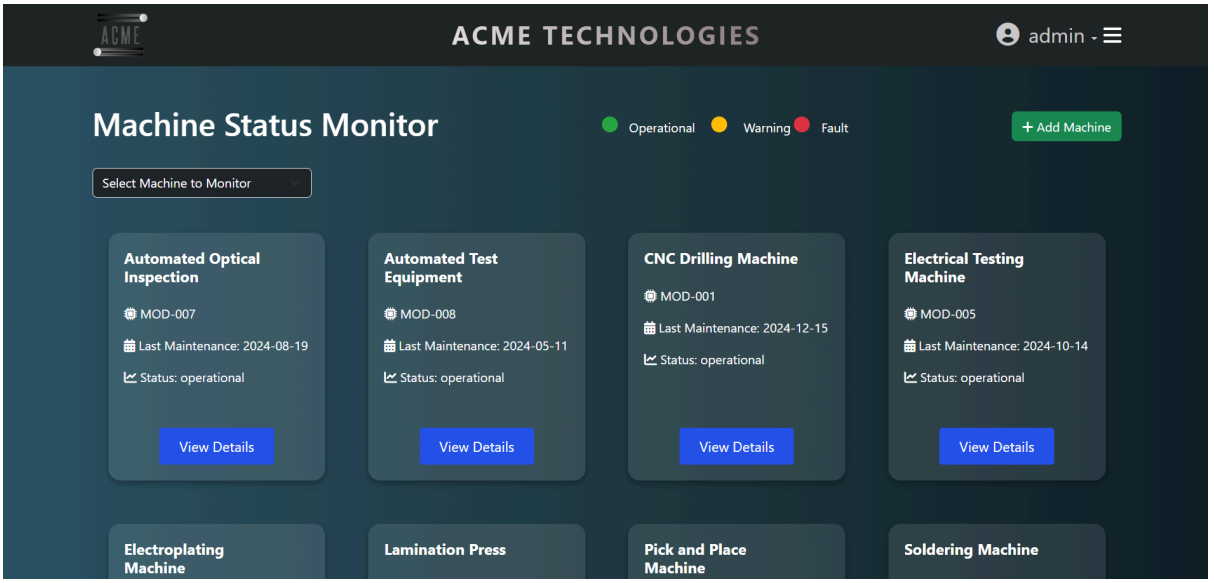


The page uses custom CSS for a modern design, with hover effects on product cards and modal pop-ups for an interactive experience. It’s mobile-responsive and easy to navigate, offering users a smooth, detailed exploration of ACME’s machinery offerings.

Machinery status page:

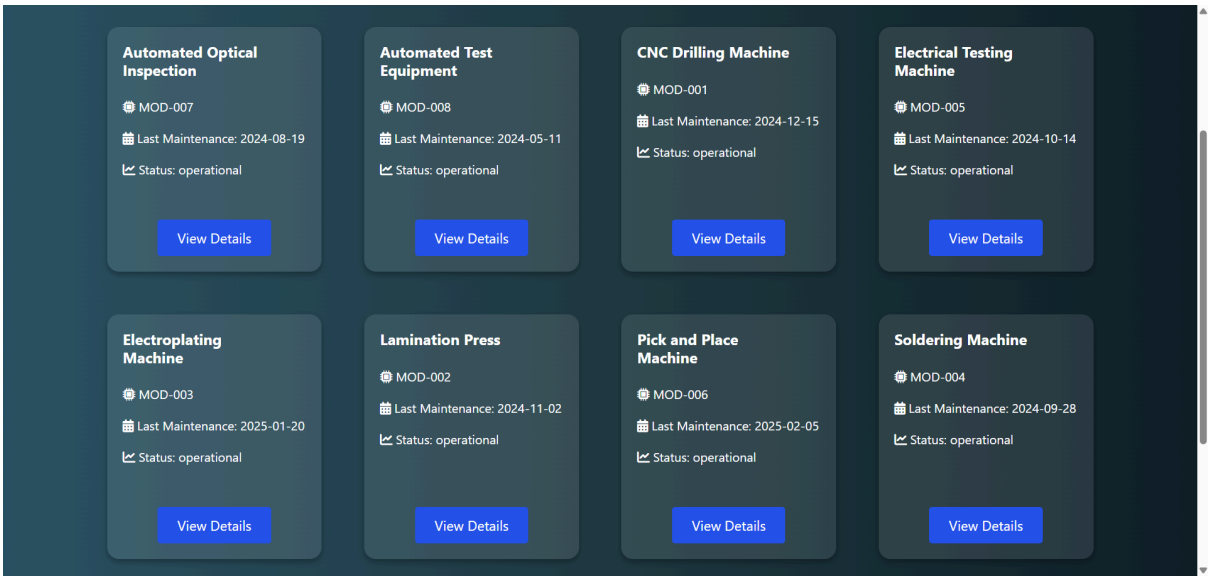
The "Machine Status Monitor" page for ACME Manufacturing Corp. allows users to track the real-time operational status of machinery in their manufacturing facility. The page is designed for an interactive and efficient user experience, leveraging Bootstrap for responsiveness and Font Awesome for icons.

1. **Header**: The page includes a clear and engaging header with the title "Machine Status Monitor" and a status legend to easily identify machine states: "Operational," "Warning," and "Fault."

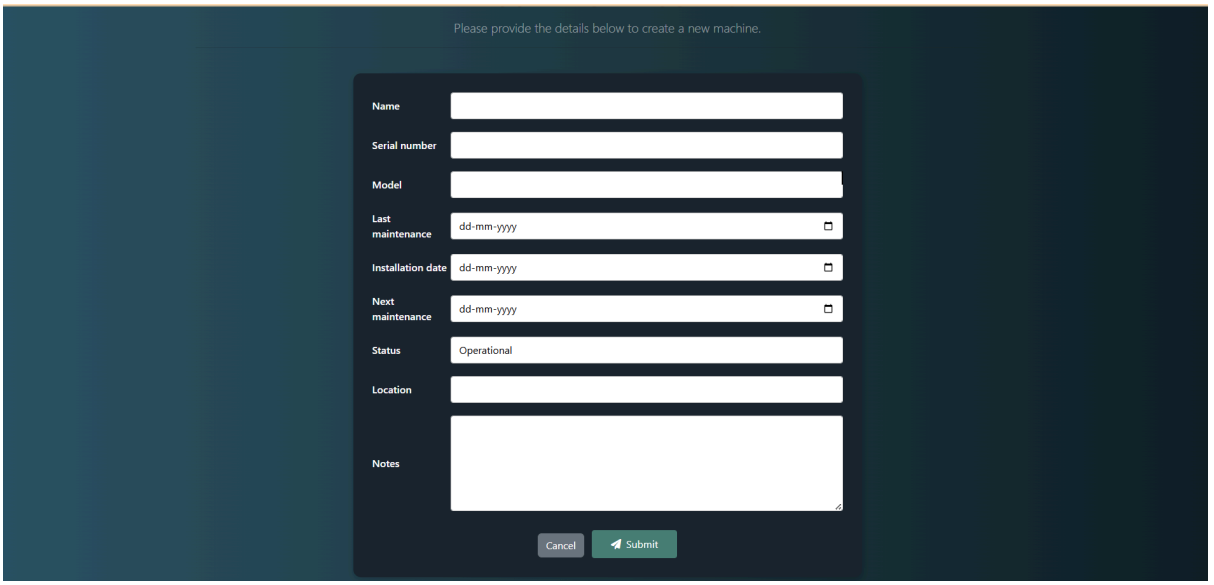


2. **Machine Filter**: Users can select a machine from a dropdown list to monitor its status. The list is dynamically populated with machine names and serial numbers.

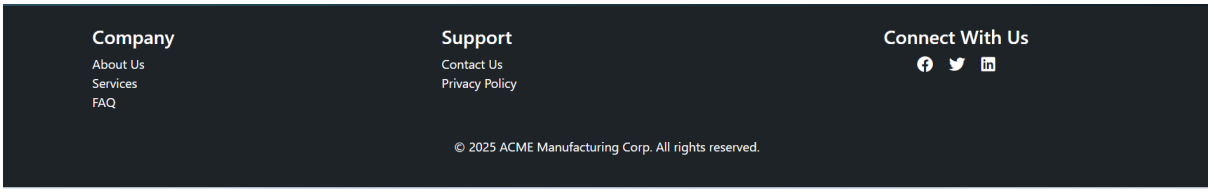
3. **Real-time Status Grid**: The machine status grid dynamically displays the operational status of machines, which is updated every minute to reflect real-time data. Cards with machine information are dynamically inserted based on the selected filter.



4. **Add Machine Button**: A prominent "Add Machine" button allows users to quickly add new machines for monitoring.



5. **Footer**: The footer includes links to the company's important pages, including "About Us," "Services," and "FAQ," as well as social media icons for connection.



6. **Auto-refresh**: The page automatically refreshes every minute to ensure the status grid stays up to date.

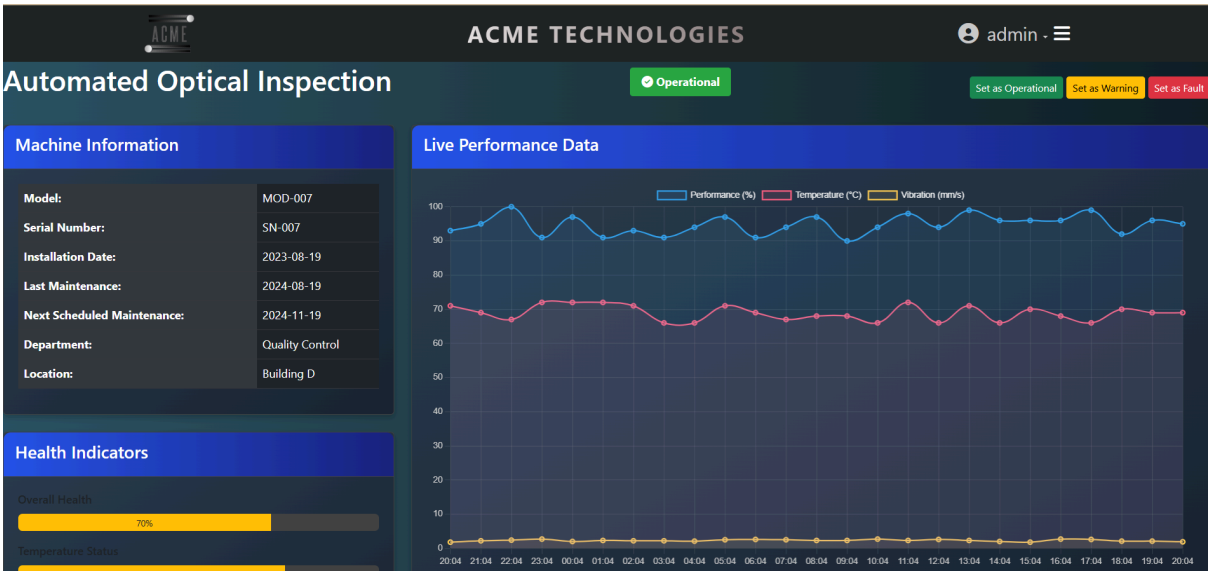
This page provides a simple, efficient interface to monitor the health and status of machinery, making it easy for users to stay informed on operational conditions.

Machinery detail page:

The "Machine Detail" page for ACME Manufacturing Corp. provides a comprehensive view of individual machine information, including operational status, health indicators, live performance data, and maintenance history. It allows users to monitor machine conditions and perform actions like status updates or ticket submissions.

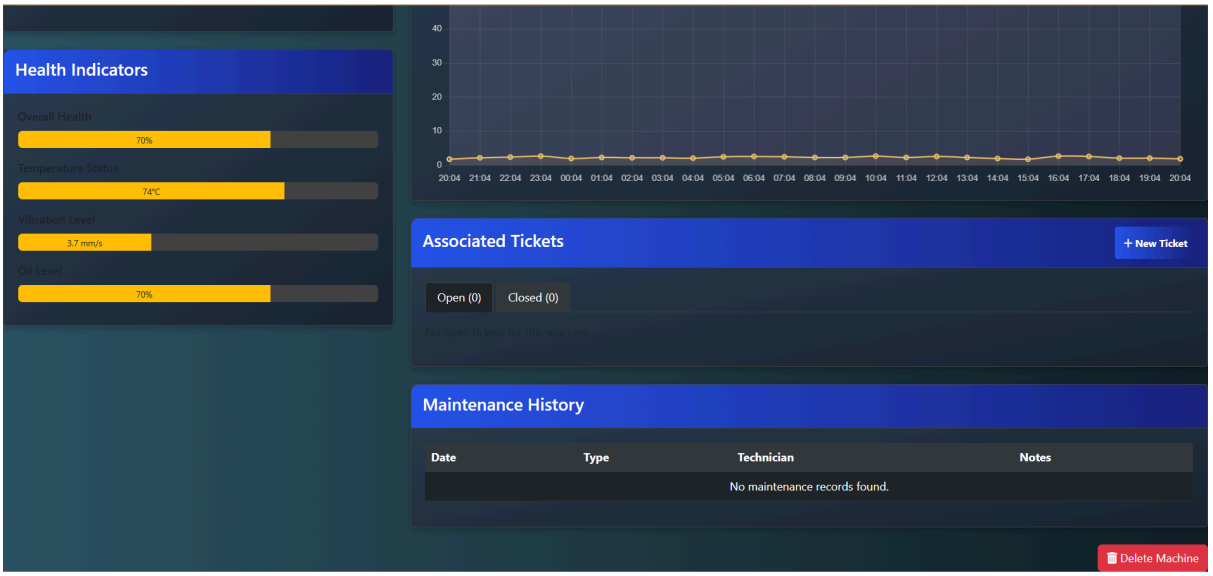
1. **Machine Header**:
    - The header displays the machine's name and current operational status, indicated by a badge that changes based on the status (Operational, Warning, or Fault).
    - Users can change the machine status using buttons that trigger an AJAX call to update the status dynamically.
2. **Machine Information**:

- This section presents essential details about the machine, including model, serial number, installation date, maintenance history, department, and location in a table format.



3. **Health Indicators**:

   - The machine's overall health, temperature, vibration, and oil levels are displayed using progress bars that visually represent their current status with different colors. These values are dynamically updated and reflect the machine's real-time condition.

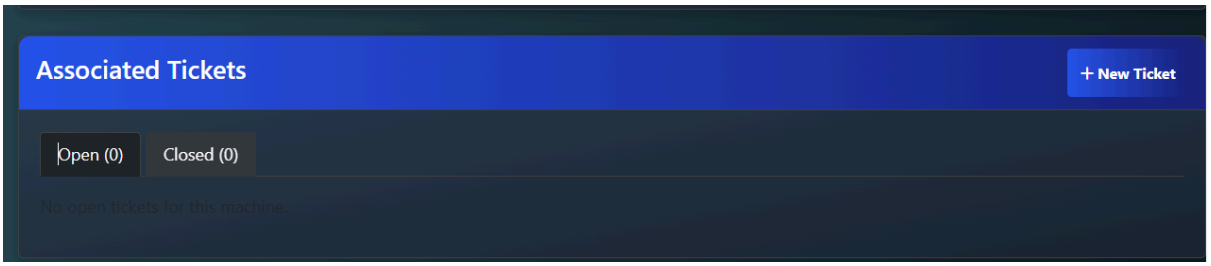


4. **Live Performance Data**:

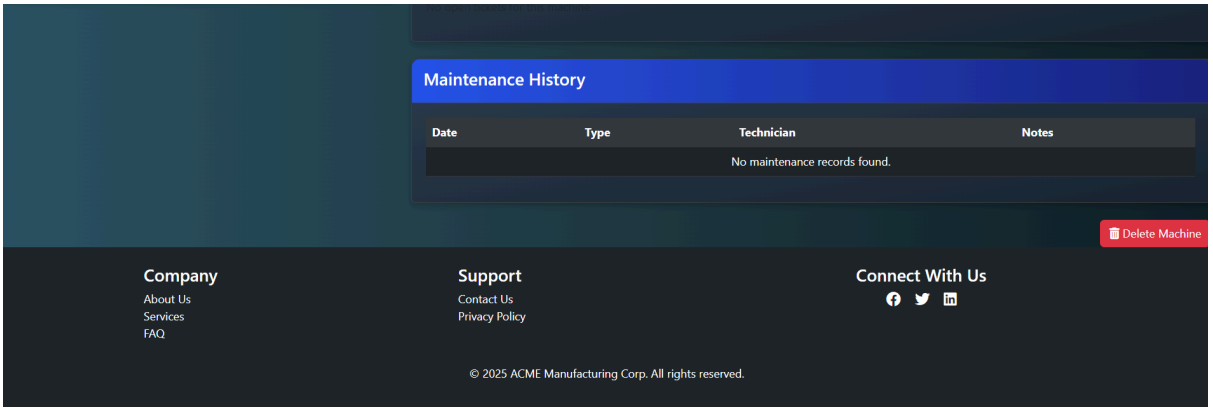
   - A performance chart (using Chart.js) shows real-time data for performance, temperature, and vibration. The chart updates automatically with new data fetched via

an API, providing live monitoring.

5. **Associated Tickets**:
    - This section displays open and closed tickets related to the machine. It uses tabs to switch between the two categories, and users can view ticket details or create new tickets.



6. **Maintenance History**:
    - A table lists the machine's maintenance history, showing the date, type of maintenance, technician, and notes for each event.
7. **Delete Machine Option**:
    - At the bottom, users can delete the machine from the system by submitting a confirmation form.



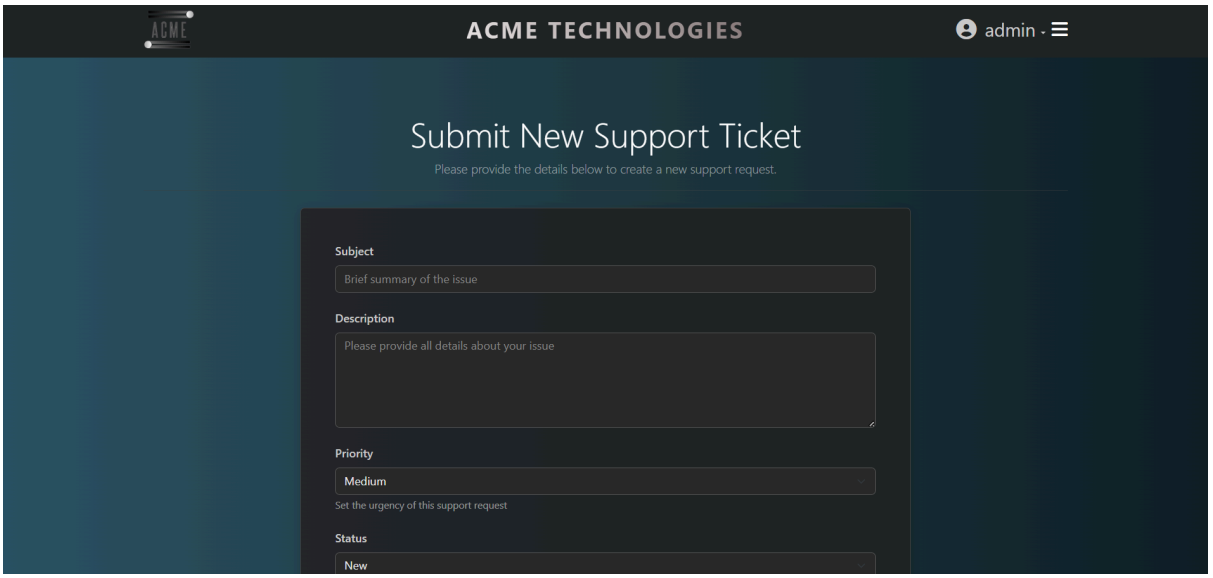
Interactive Elements:

- **Status Update**: The status update buttons trigger AJAX requests that dynamically change the machine's status on the page without reloading.
- **Performance Chart**: The chart visualizes real-time performance, with data fetched from the backend and displayed in a line graph.
- **Ticket Management**: Users can view open and closed tickets and quickly add new ones.

This page is designed for real-time monitoring and efficient management of machine performance and issues, making it an essential tool for ACME's machine tracking system.

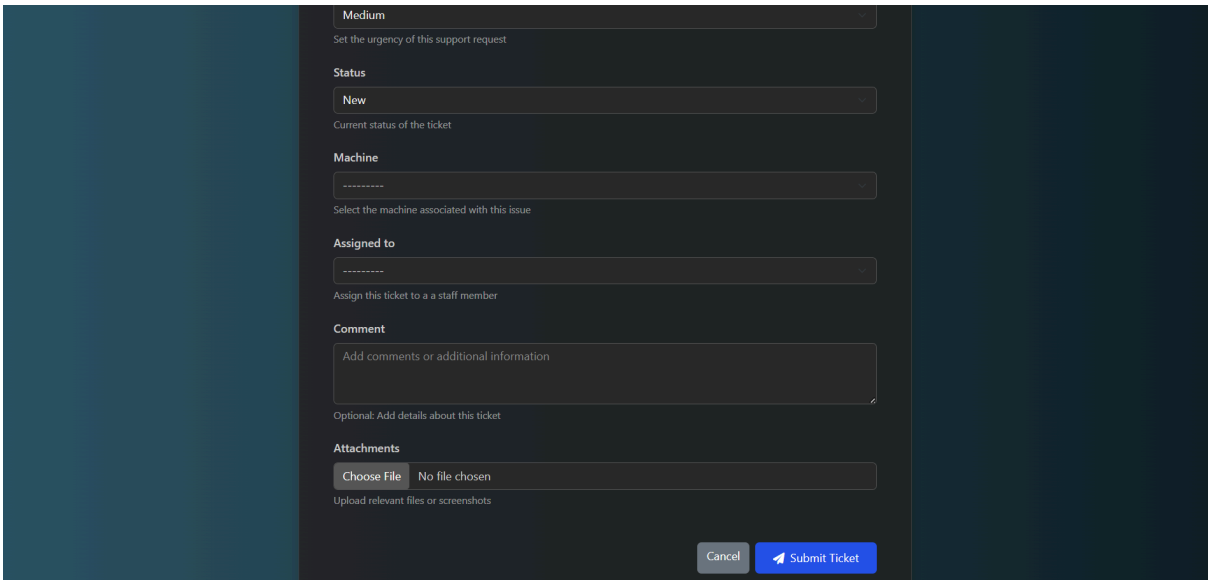
Submit ticket page:

The "Submit Ticket" page for ACME Manufacturing Corp. provides a user-friendly interface for submitting support tickets. This page allows users to report issues, request assistance, and manage their tickets within the system.

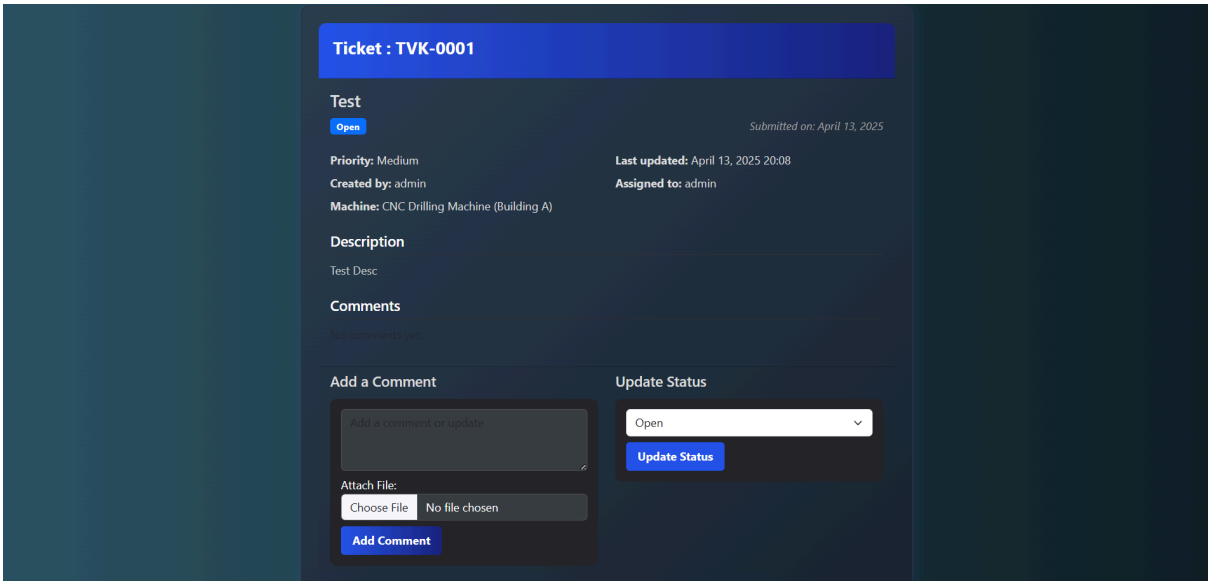


1. **Header**:
    - The page features a prominent header with the title "Submit New Support Ticket" (or a custom title) and a brief description of the purpose of the page. It sets the tone for the user to fill in the necessary details.
2. **Form**:
    - The form allows users to submit their ticket details. Each field in the form is rendered dynamically using Django template tags. If there are any validation errors or help text, they are displayed below the respective fields.
    - **Error Handling**: Errors are displayed as invalid feedback with an icon and message, helping users correct their input.
    - **CSRF Protection**: Django's {% csrf_token %} tag ensures security against cross-site request forgery attacks.
3. **Buttons**:
    - The page includes two main buttons:

- **Cancel**: Redirects the user to the ticket dashboard without submitting.
- **Submit Ticket**: Submits the ticket with the entered details.



4. **Message Display**:
    - If there are any messages (success or error) from Django's message framework, they are displayed at the top of the form in alert boxes.



5. **Footer**:
    - The footer contains links to essential company pages like "About Us," "Services," "FAQ," and "Privacy Policy."

- Social media icons are included to encourage engagement.
- The footer includes a copyright notice with the current year, dynamically generated using Django's {% now "Y" %} tag.

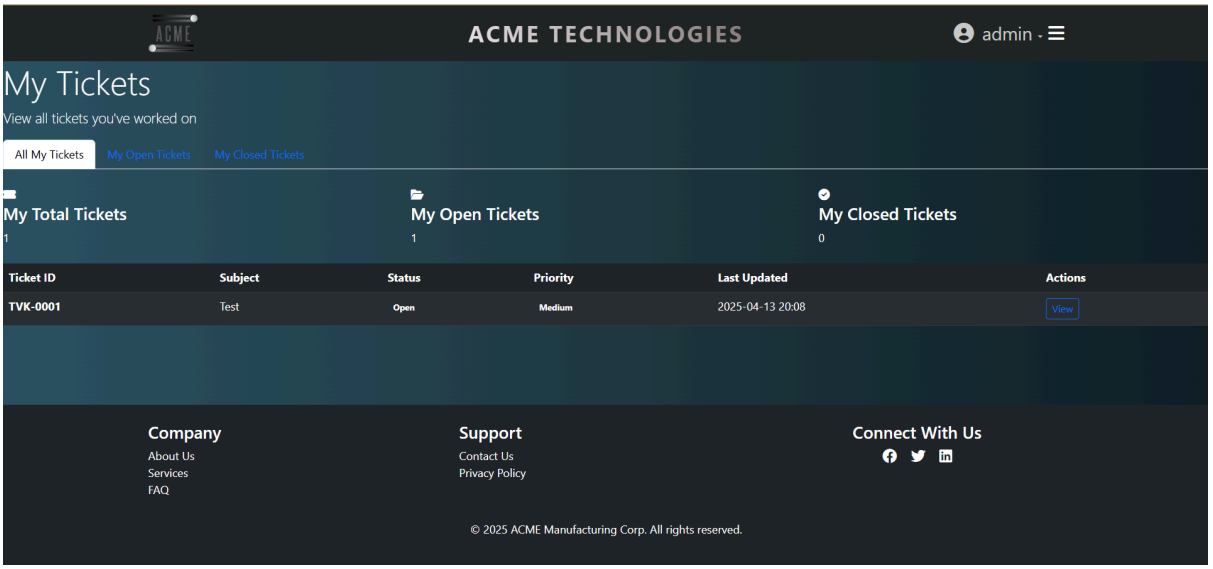
Design:

- The page uses Bootstrap for layout and styling, with responsive design to ensure accessibility on mobile devices.
- The form is presented in a clean, professional style with clear spacing and alignment for a better user experience.

This page is essential for users to report technical issues or request assistance, integrating seamlessly into ACME Manufacturing Corp.'s support system.

My tickets page:

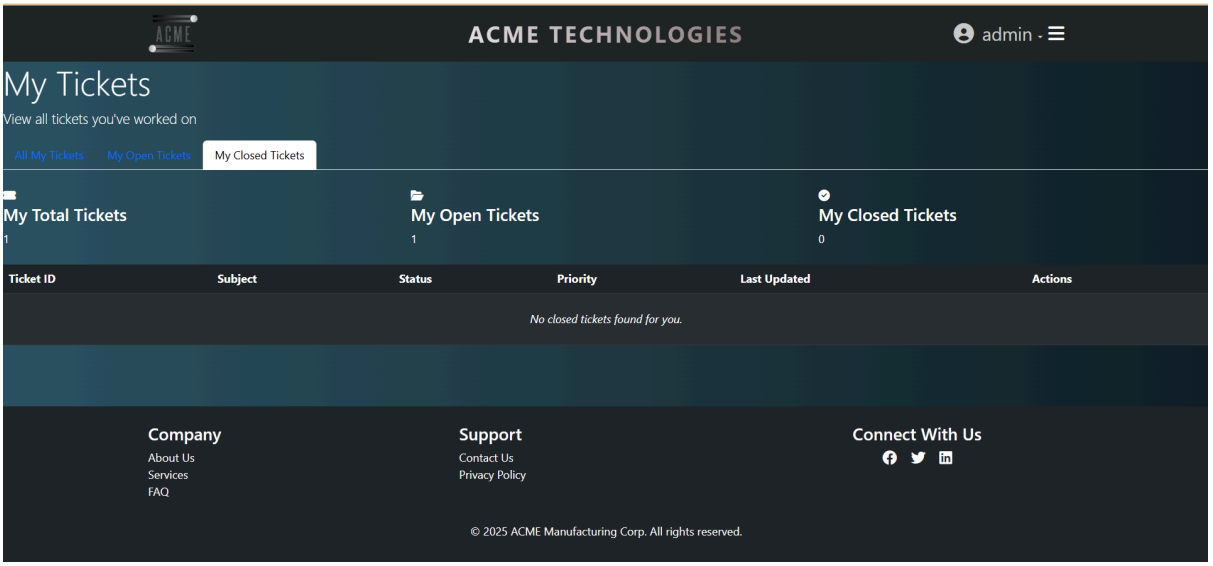
The "My Tickets" page for ACME Manufacturing Corp. provides users with a detailed overview of the tickets they've worked on, allowing them to track the status and progress of their support requests.



1. **Page Header**:
    - The page header includes a title ("My Tickets") and a brief description inviting users to view all the tickets they've worked on.
2. **Tab Navigation**:
    - Three tabs allow users to filter and view their tickets by category:
        - **All My Tickets**: Displays all tickets.

- **My Open Tickets**: Filters and displays open tickets.
- **My Closed Tickets**: Filters and displays closed tickets.

- Each tab is dynamically styled based on the current_view variable to highlight the active category.



3. **Summary Cards Section**:
   - This section presents three summary cards that provide quick insights:
     - **My Total Tickets**: Displays the total number of tickets.
     - **My Open Tickets**: Shows the count of open tickets.
     - **My Closed Tickets**: Displays the count of closed tickets.
   - Each card includes an icon and a number indicating the count.
4. **Table Content**:
   - The table lists all tickets based on the selected tab (All, Open, or Closed), displaying the following information for each ticket:
     - **Ticket ID**
     - **Subject** (with truncation for long text)
     - **Status** (styled with badges indicating Open, In Progress, Closed, etc.)

        - **Priority** (with badges for High, Medium, Low priority)
        - **Last Updated** (formatted as a date and time)
        - **Actions** (a "View" button that leads to the ticket detail page)
    - If no tickets are found in the selected category, a message is displayed in the table.

5. **Footer**:
    - The footer provides links to essential company pages and social media profiles for user engagement.

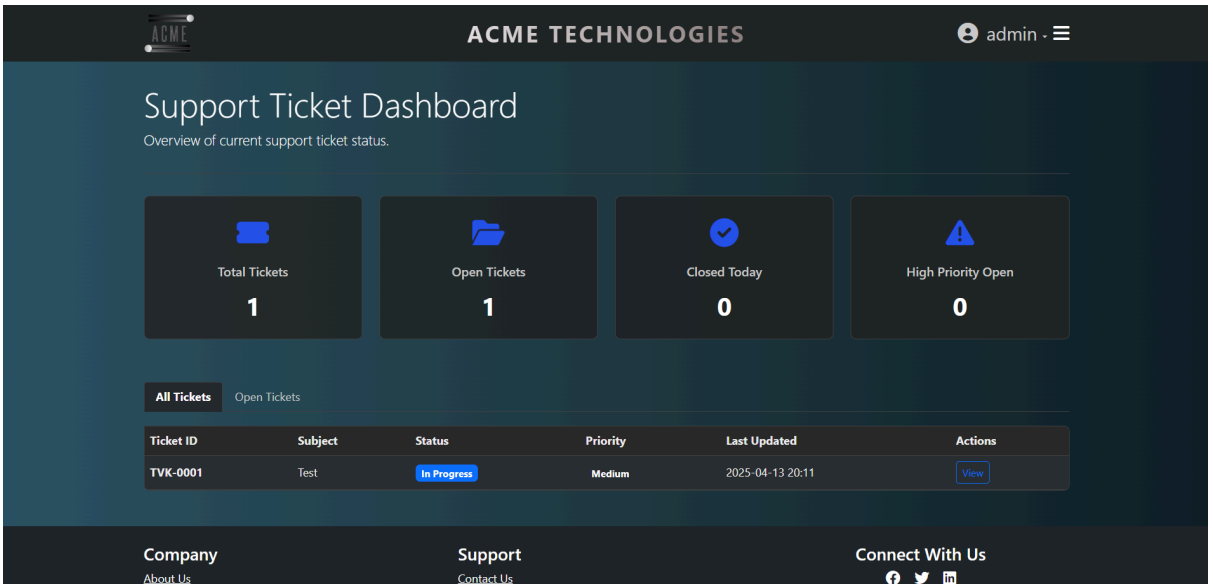
Design:

- The page uses Bootstrap for a clean, responsive layout, ensuring accessibility on both desktop and mobile devices.
- Dynamic tab switching helps users quickly navigate between different ticket categories.
- The use of badges for status and priority makes it easy to visually distinguish between different ticket states.

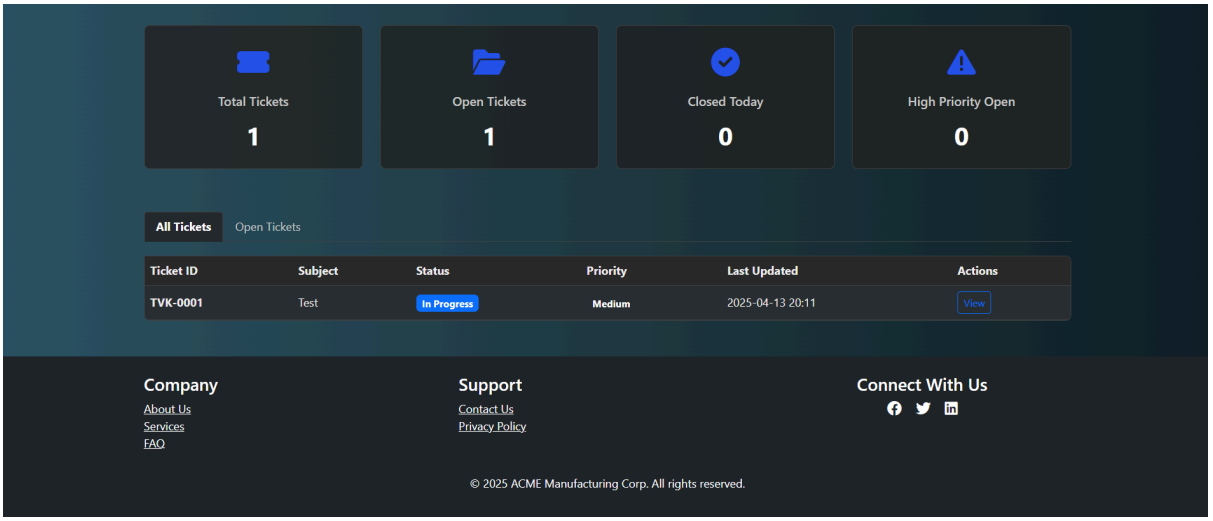
This page is an essential part of ACME Manufacturing Corp.'s ticketing system

Ticket dashboard page:

The "Ticket Dashboard" page for ACME Manufacturing Corp. provides an overview of support tickets and their statuses, allowing users to track and manage their tickets in a centralized dashboard.



1. **Page Header**:
    - The header prominently displays the page title "Support Ticket Dashboard" and a brief description, helping users understand the purpose of the page.
2. **Summary Cards Section**:
    - The dashboard includes four summary cards that provide quick insights into ticket statistics:
        - **Total Tickets**: Displays the total number of tickets.
        - **Open Tickets**: Shows the count of tickets that are currently open.
        - **Closed Today**: Displays the number of tickets closed on the current day.
        - **High Priority Open**: Highlights the number of high-priority open tickets.
3. **Tab Navigation**:
    - Tabs allow users to filter and view tickets based on their status:
        - **All Tickets**: Displays all tickets.
        - **Open Tickets**: Filters and shows open tickets.
    - Each tab is dynamically styled based on the current_view variable, making it easy to identify the active view.
4. **Ticket Table**:
    - The table displays a list of tickets with relevant details, including:
        - **Ticket ID**: The unique identifier for the ticket.
        - **Subject**: The subject of the ticket, truncated for readability.
        - **Status**: The current status of the ticket, with color-coded badges (e.g., Open, In Progress, Closed).
        - **Priority**: The priority level of the ticket (e.g., High, Medium, Low), also color-coded.
        - **Last Updated**: The timestamp of the last update.
        - **Actions**: A button to view the detailed ticket.



5. **Empty State**:
    - If no tickets are found for the selected view, a message is displayed in the table informing the user that no tickets are available.
6. **Footer**:
    - The footer includes links to key company pages like "About Us," "Services," "FAQ," and "Privacy Policy."
    - Social media icons are included to facilitate engagement with the company.

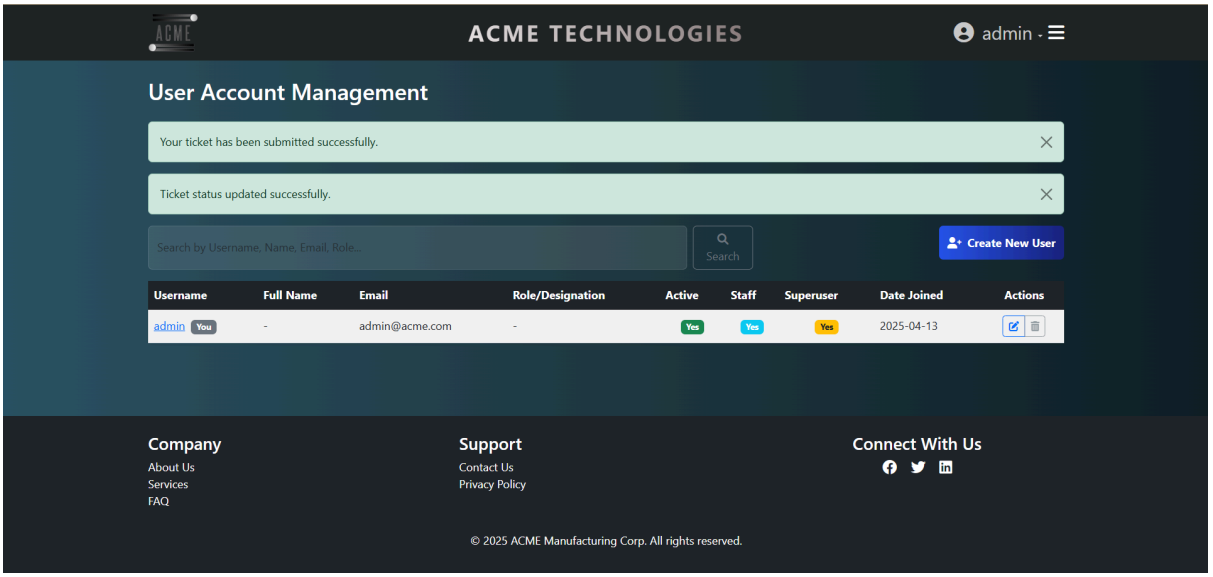
Design:

- The page is built using Bootstrap for responsiveness, ensuring that it looks good on both desktop and mobile devices.
- The tab navigation makes it easy for users to switch between different ticket views.
- The use of badges for ticket status and priority provides clear visual cues, enhancing user experience and efficiency in navigating the ticket data.

This page serves as a comprehensive, interactive interface for users to monitor and manage their support tickets efficiently.

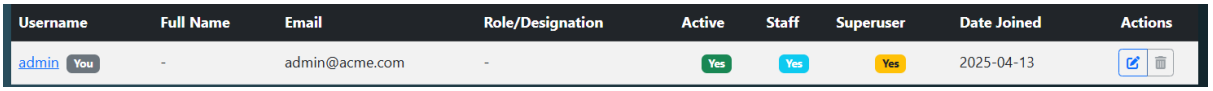
User Account Management:

The "User Account Management" page for ACME Manufacturing Corp. enables administrators to manage user accounts, including creating new users, editing existing accounts, and deleting users.



1. **Page Header**:
    - The header section includes a title "User Account Management" and a brief description ("Overview of current user accounts"). It provides clarity about the purpose of the page.
2. **Search Form**:
    - The page includes a search bar that allows administrators to search for users by username, name, email, or role. It uses a GET request to submit the search query, and the query is displayed in the input field. If a search is active, a "Clear" button is displayed to reset the search.
3. **Create User Button**:
    - A "Create New User" button is displayed to allow administrators to add new users. Clicking this button redirects to the user signup or creation page.
4. **User List Table**:
    - The main table displays a list of users, with columns showing:
        - **Username** (links to the edit user page and highlights "You" if the logged-in user is being viewed)
        - **Full Name** and **Email**
        - **Role/Designation** (accessed through the user's profile)
        - **Active**, **Staff**, and **Superuser** status (displayed with badges indicating Yes/No)

- **Date Joined** (formatted as YYYY-MM-DD)
- **Actions**: Includes edit and delete buttons for each user, with delete functionality disabled for the logged-in user.



5. **Status Badges**:
    - Badges are used to indicate whether the user is active, a staff member, or a superuser, with color coding for easy identification (e.g., green for active, red for inactive, etc.).
6. **Action Buttons**:
    - **Edit Button**: Links to the user edit page for modifying user details.
    - **Delete Button**: Allows administrators to delete users, with a confirmation prompt. The delete button is disabled for the logged-in user to prevent accidental self-deletion.
7. **Empty State**:
    - If no users match the search criteria or if no users are available, a message is displayed informing the administrator that no users were found. It also suggests creating a new user.
8. **Responsive Layout**:
    - The page uses Bootstrap for a responsive design, ensuring that it is optimized for various screen sizes (desktop, tablet, mobile).

Design:

- The table uses the table-striped and table-hover Bootstrap classes to improve readability, with alternating row colors and hover effects.
- The summary cards give quick insights into the total number of users, open tickets, and high-priority tickets.
- Actions like editing and deleting users are presented with clear icons and tooltips for better user experience.

This page provides a comprehensive view of user management, allowing administrators to easily navigate, search, edit, and delete users within the system. It is an essential tool for maintaining and organizing user accounts in ACME Manufacturing Corp's platform.

9. Database Schema

The database schema is intended to provide a web-based machine management and ticketing system that was mainly constructed with the Django framework. Fundamentally, it combines user authentication and permissions with two primary features: issue ticketing and industrial equipment maintenance tracking.

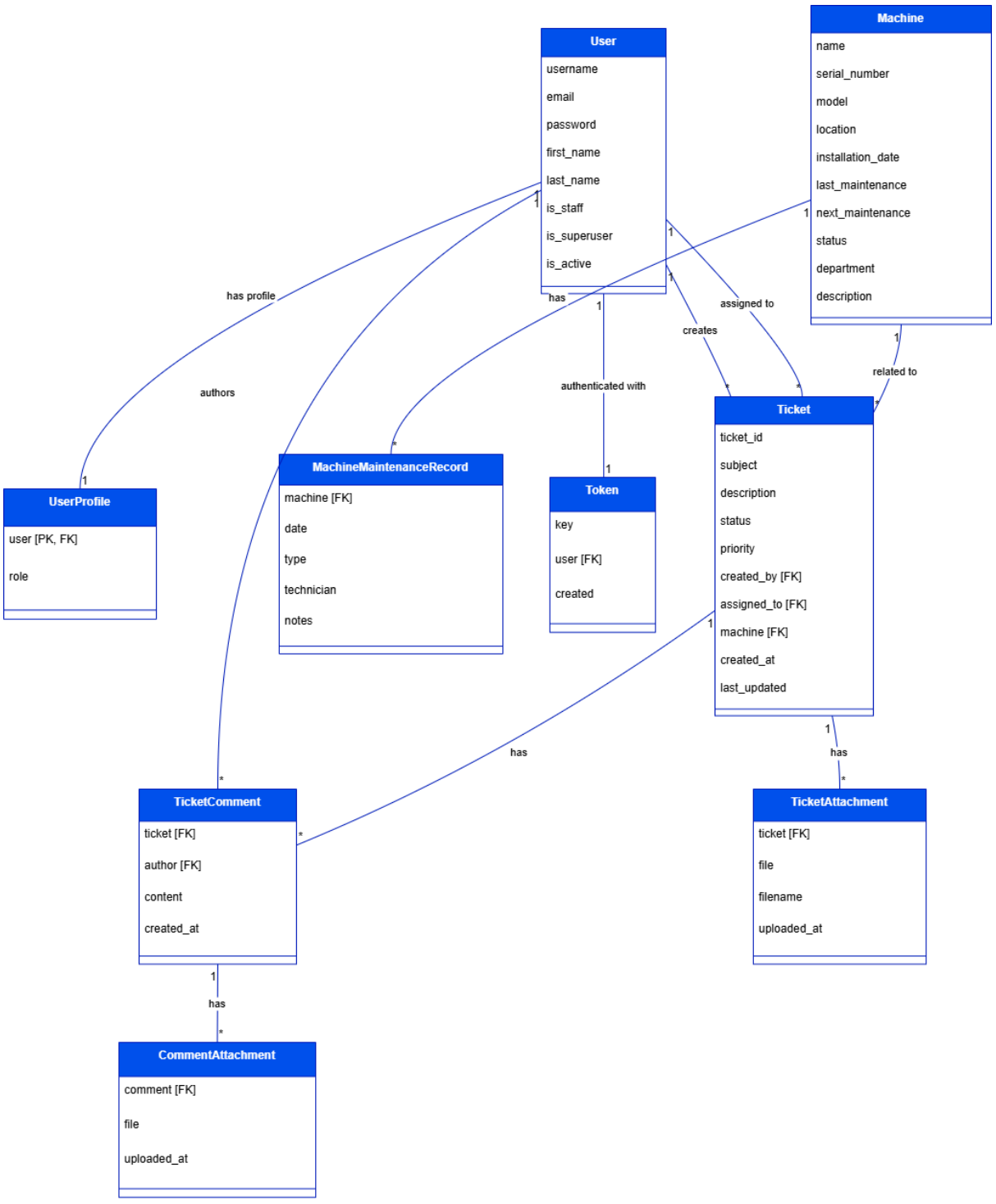
A number of application-specific tables are included in the schema, including acme_machine, which contains details on machines, including their name, model, serial number, status, and maintenance dates. acme_machinemaintenancerecord, which records the dates, people involved, and notes of each machine's maintenance history, is a useful addition to this.

The acme_ticket table keeps track of each ticket, recording information such as the title, description, machine involved, originator, status, and timestamps. Each ticket can have attachments and comments associated with it using the acme_ticketattachment and acme_ticketcomment tables, respectively. Furthermore, files can be directly attached to comments with acme_commentattachment, facilitating more thorough issue tracking and documentation.

Django's internal authentication mechanism handles user administration. The auth_user table contains profile information and user credentials. Auth_group, auth_group_permissions, auth_user_groups, and auth_user_user_permissions can be used to group users and provide them particular capabilities. In order to facilitate safe API interactions, token-based authentication is also provided using the authtoken_token table.

Tables like django_admin_log, django_content_type, django_migrations, and django_session are also built to manage the internal administrative and operational operations of the Django project. By referring to content types, migration history, and sessions, respectively, these control the logging of admin operations.

All things considered, this plan logically offers a cooperative setting for monitoring machine health, recording maintenance tasks, monitoring user-submitted problems, and executing secure user access. The full scheme is outlined below:



User Management Tables

auth_user: Stores the user account details such as username, password (hashed), email, and other profile details. It is the core table for all system registered users.

auth_user_groups: Links users to groups. Groups allow handling permissions for a group of users as a whole.

auth_user_user_permissions: Links individual users to specific permissions directly (not via group).

auth_group: Defines user groups (e.g., admin, technician, viewer), to which users can be assigned.

auth_group_permissions: Associates groups with permissions in order to deliver role-based permissioning.

auth_permission: Tracks a list of all permissions accessible within the system (e.g., add machine, delete ticket). Model-dependent.

authtoken_token: Used to save authentication tokens for users (to be used for API access or token-based logins).

TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME
public	auth_user	id	4	serial
public	auth_user	password	12	varchar
public	auth_user	last_login	93	timestamptz
public	auth_user	is_superuser	-7	bool
public	auth_user	username	12	varchar
public	auth_user	first_name	12	varchar
public	auth_user	last_name	12	varchar
public	auth_user	email	12	varchar
public	auth_user	is_staff	-7	bool
public	auth_user	is_active	-7	bool
public	auth_user	date_joined	93	timestamptz

Machine and Maintenance Tables

acme_machine: Stores details about plant machines, including name, model, serial number, where they are located, date installed/maintained, and description.

TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME
public	acme_machine	id	4	serial
public	acme_machine	name	12	varchar
public	acme_machine	model_number	12	varchar
public	acme_machine	serial_number	12	varchar
public	acme_machine	location	12	varchar
public	acme_machine	purchase_date	91	date
public	acme_machine	last_maintenance	91	date
public	acme_machine	status	12	varchar
public	acme_machine	notes	12	text

acme_machinemaintenancerecord: Stores maintenance records per machine, like date and description of work, all of which are tied to a machine.

Ticketing System Tables

acme_ticket: Holds support or maintenance tickets related to machines. Holds information such as the issue, status, priority, affected machine, and user who opened the ticket.

TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME
public	acme_ticket	id	-5	int8
public	acme_ticket	subject	12	varchar
public	acme_ticket	description	12	text
public	acme_ticket	status	12	varchar
public	acme_ticket	priority	12	varchar
public	acme_ticket	created_at	93	timestamptz
public	acme_ticket	last_updated	93	timestamptz
public	acme_ticket	assigned_to_id	4	int4
public	acme_ticket	created_by_id	4	int4
public	acme_ticket	ticket_id	12	varchar

me_ticketcomment: Stores comments that are added to tickets, usually for status or discussion. Related to the user who posted it and the related ticket.

acme_ticketattachment: Holds file attachments that are added to a ticket (e.g., error screenshots or reports).

acme_commentattachment: Holds files attached to individual comments, allowing users to add evidence or context to their notes.

Django Internal System Tables

django_admin_log: Logs all administrative actions (e.g., object deletion or editing) performed through Django's admin interface. Assist with auditing.

django_content_type: Links models to a generic content type system in Django so that numerous models can share the same relationship with permissions and logs.

django_migrations: Records changes in database schema over time. All migrations run are logged here so they won't be repeated.

django_session: It holds session data for logged-in users, marking their activity and state across requests.

Docker Installation:

Docker is a platform for developing, shipping, and running applications. Docker uses containerization technology to package applications and their dependencies into a container that ensures the application runs the same everywhere. Docker implementation in your project involves numerous components and processes that facilitate building, deploying, and managing the application in sandboxed environments. The following is a detailed explanation of how Docker is implemented and used in the project.

1. **Docker Basics**

Docker containers are lightweight and portable, providing a consistent environment across the development lifecycle. Containers run directly on the kernel of the host machine but are isolated per

container with their own file system, network, and processes. Docker uses images, read-only templates to create containers. The images are built using Dockerfiles, which define the installation and setup of the application and its dependencies.

Docker Components:

- Docker Engine: The execution environment that runs and manages containers.
- Docker Image: A blueprint to create containers. It has all the dependencies the application needs to run (code, libraries, environment variables).
- Docker Container: A runtime instance of a Docker image. It is isolated and runs an application or service.

Dockerfile: A text file with instructions to build a Docker image.

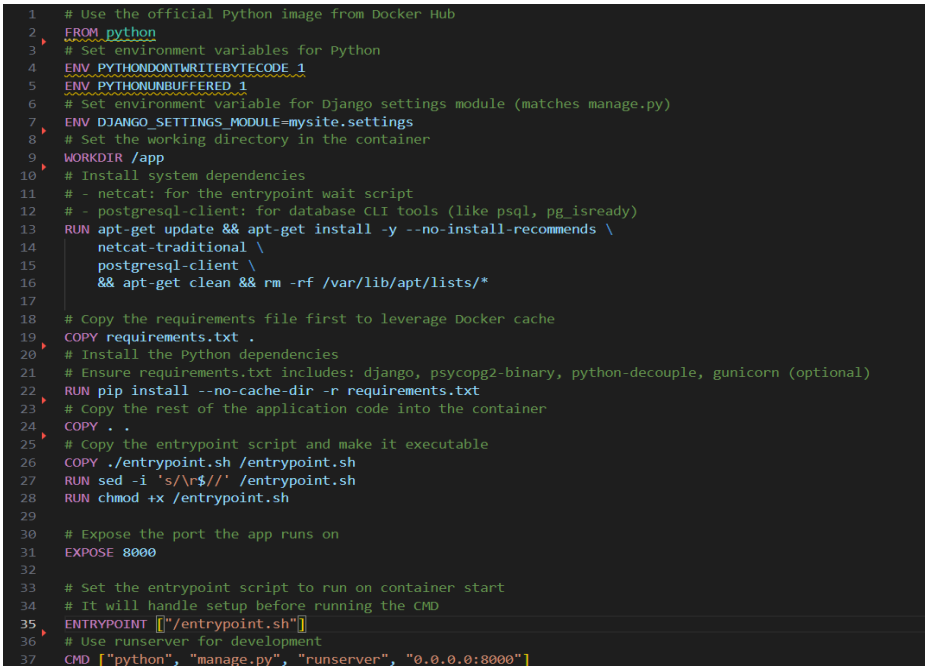
2. Configuring Docker for the Application

Docker configuration in your project primarily involves defining a Dockerfile, which contains instructions to create a container image with the required application environment.

Steps for Docker Setup:

Install Docker: Docker is set up on the local machine or server (in case it's the deployment scenario). It is used to build images, run containers, and manage the entire lifecycle.

Create a Dockerfile: A Dockerfile contains the instructions to build the Docker image for the app. In a web application, this file defines the base image (for example, Python or Node.js), installs dependencies, and sets environment variables to run the app.



```
# Use the official Python image from Docker Hub
FROM python
# Set environment variables for Python
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
# Set environment variable for Django settings module (matches manage.py)
ENV DJANGO_SETTINGS_MODULE=mysite.settings
# Set the working directory in the container
WORKDIR /app
# Install system dependencies
# - netcat: for the entrypoint wait script
# - postgresql-client: for database CLI tools (like psql, pg_isready)
RUN apt-get update && apt-get install -y --no-install-recommends \
    netcat-traditional \
    postgresql-client \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# Copy the requirements file first to leverage Docker cache
COPY requirements.txt .
# Install the Python dependencies
# Ensure requirements.txt includes: django, psycopg2-binary, python-decouple, gunicorn (optional)
RUN pip install --no-cache-dir -r requirements.txt
# Copy the rest of the application code into the container
COPY . .
# Copy the entrypoint script and make it executable
COPY ./entrypoint.sh /entrypoint.sh
RUN sed -i 's/\r$//' /entrypoint.sh
RUN chmod +x /entrypoint.sh

# Expose the port the app runs on
EXPOSE 8000

# Set the entrypoint script to run on container start
# It will handle setup before running the CMD
ENTRYPOINT ["/entrypoint.sh"]
# Use runserver for development
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Docker Compose: Docker Compose is used in a multi-container application. Docker Compose simplifies the management of various services (e.g., database, web server, etc.) that make up the application using a docker-compose.yml file. This specifies services, networks, and volumes.

Example docker-compose.yml file

This setup creates two services:

Web: A service to run the web application (e.g., Django) with a Python image, exposing port 8000.

DB: A service to execute a PostgreSQL database container with required environment variables (like POSTGRES_USER, POSTGRES_PASSWORD, and POSTGRES_DB).

3. Creating Docker Images

After having docker-compose.yml and Dockerfile ready, the second step is to construct the Docker image.

Build the Docker Image: The command for building a Docker image from the Dockerfile is docker build.

docker build -t acme

Build and Run with Docker Compose: If you're using Docker Compose, you can build and run the containers with a single command:

docker-compose up --build

This command does several things:

- It builds images for all services defined in the docker-compose.yml file.
- It creates and starts containers for all services.
- It allows the web service to communicate with the database service through Docker's internal networking.

4. Running Containers

Once the containers are built, you can start the services using the docker run command (one-off containers) or docker-compose up (for orchestrating multiple containers that work together with each other).

Run the Web Application: After building the web application image, run it in a container: docker run -p 8000:8000 acme
Run with Docker Compose: If running with Docker Compose, this command boots all of the services specified in the docker-compose.yml file:

docker-compose up

The web application can now be accessed by visiting http://localhost:8000, with the PostgreSQL database in the background.

5. Managing Database with Docker

In this project, the PostgreSQL database is managed as a Docker service using the db service available in the docker-compose.yml file. The database is populated with required tables and data when the container is started for the first time.

Database Volume: A volume (db-data) is declared in the docker-compose.yml file to store the database data even when the container is stopped or removed. This is done to prevent data loss when the container is restarted.

Database Initialization: On initial startup of the database container, it can also initialize the database schema automatically by using SQL scripts or migrations. These scripts usually reside in the Docker image or are mounted on the container as volumes.

6. Database Migrations

Docker also simplifies running database migrations. After you have the container running and the database online, you can run Django migrations inside the container to set up the database schema.

Run Migrations in a Docker Container: Run the following command to run Django migrations inside the web container:

docker-compose run web python manage.py migrate

This will apply the database schema and create the necessary tables in the PostgreSQL container.

7. Log and Persistent Data Management

Docker provides tools for log and persistent data management:

Docker Logs: The logs for each running container are accessible with:

docker logs e8ed66a4712c
Database Backups: You can backup the database by dumping the data from the PostgreSQL container:

docker exec -te8ed66a4712c pg_dumpall -c -U postgres > dump.sql

Utilization of Docker in this project helps to make the development, testing, and deployment process easier by providing isolated environments for the database and web application. Docker guarantees that all dependencies, configurations, and services required by the application are packaged into containers, hence easier to handle and scale the application across different environments. With Docker Compose, the management of multiple services (such as a web server and database) is simpler, and it simplifies the integration of these components smoothly. The database can be managed effectively with Docker volumes so that data persists across container restarts and migrations are applied consistently. Docker simplifies the entire process of deployment, and web applications can be created and deployed in a consistent, reproducible manner.

10. Testing, validation and Encountered challenges:

Front end testing and validation (Css files /js files functionalities):

Navbar.html: Test responsive menu, auth-dependent links (Sign In/Out), and role-based admin access.

Catalog.html: Check modal functionality for machine details, image loading, and responsive product grid.

Contact.html: Test form validation (email, name rules), submission success, and error message clarity.

FAQ.html: Verify accordion expand/collapse, table rendering, and content accuracy.

Index.html: Test hero image loading, AOS animations, and Quick Links navigation.

Machine_detail.html: Validate real-time charts, status change buttons, and maintenance history display.

Machine_status.html: Test live status updates, filter functionality, and "Add Machine" button flow.

Policy.html: Check compliance badge visibility, responsive layout, and content formatting.

Profile.html: Validate user data display, role badges, and profile completeness.

Services.html: Test service card layouts, icon visibility, and responsive grid behavior.

Backend testing and validation (correct links and error handling):

a. Ticketing system testing:

Testing Valid Inputs

1. **Submit with Minimum Required Fields:**
    - **Input:**
        - Subject: "CNC Drilling Machine not working"
        - Description: "The CNC Drilling Machine is not responding. Tried restarting it."
        - Priority: Medium
        - Status: New
        - Machine: Any valid machine
        - Assigned to: test
        - Comment: (blank)
        - Attachments: (blank)
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Ticket submitted successfully. User to be redirected to a ticket view page.
    - **Actual Result**: Ticket submitted successfully. User redirected to a ticket view page.
2. **Submit with All Fields Filled:**
    - **Input:**
        - Subject: "Software Crash Issue"
        - Description: "The application consistently crashes. See attached screenshot."
        - Priority: High
        - Status: New
        - Machine: Any valid machine
        - Assigned to: Any valid user
        - Comment: "Issue occurs for multiple users on different machines."
        - Attachments: Browse and select any valid file
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Ticket submitted successfully with all details, including the comment and attachment.
    - **Actual Result**: Ticket submitted successfully. User redirected to a ticket view attachment missing **[Bug now fixed]**

3. **Submit with Different Priority Levels:**
    - **Input:** Fill required fields. Select "Low" priority.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Success. Verify tickets are created with "Low" priority.
    - **(Repeated for other priority levels.)**
    - **Actual Result:** Ticket submission success, all priority levels show up.
4. **Submit with Long Description:**
    - **Input:** Fill required fields. Enter a very long, detailed description
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Success. Verify the full description is saved correctly.
    - **Actual Result**: Ticket submitted successfully. Description is saved correctly (1000 words)
5. **Submit with Special Characters in Text Fields:**
    - **Input:** Use special characters (~!@#$%^&*()_+=-{}[]|\\:";'<>,.?/) in Subject, Description, and Comment.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Success. Verify characters are handled correctly (displayed as entered, not causing errors or security issues like XSS).
    - **Actual Result**: All characters displayed as entered, no errors or issues.
6. **Submit with Large Attachment:**
    - **Input:** Fill required fields. Try to attach a large file (tested 4gb)
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Submission succeeds.
    - **Actual Result:** Submission succeeds.

Error Handling - Invalid/Missing Inputs

1. **Submit with Empty Subject:**
    - **Input:** Leave Subject blank. Fill other required fields.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Submission fails. An error message is displayed indicating the Subject field is required.
    - **Actual Result**: Submission fails, error message appears indicating subject is required.
2. **Submit with Empty Description:**
    - **Input:** Leave Description blank. Fill other required fields.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Submission fails. An error message is displayed indicating the Description field is required
    - **Actual Result**: Submission fails, error message appears indicating description is required.
3. **Submit with Invalid Attachment Type:**

    - **Input:** Fill required fields. Try to attach a disallowed file type (e.g., .exe, .bat, .sh).
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Submission fails. An error message indicates the file type is not allowed.
    - **Actual Result:** All attachments are accepted, but certain ones will not be shown to the user.
4. **Submit with Multiple Required Fields Missing:**
    - **Input:** Leave Subject and Description blank.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Submission fails. Error messages are displayed for *both* missing fields.
    - **Actual Result:** Submission fails. Error messages are displayed for *both* missing fields.

Boundary and Edge Case Test Cases

1. **Subject Exceeding Maximum Length:**
    - **Input:** Enter text exceeding the maximum allowed character limit for Subject (test if typing stops). Fill other required fields.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Input field prevents exceeding the limit.
    - **Actual Result:** Input field prevents exceeding the limit.
2. **Description Maximum Length:**
    - **Input:** Enter text exactly at the maximum allowed character limit for the Description field. Fill other required fields.
    - **Action:** Click "Submit Ticket"
    - **Expected Result:** Success.
    - **Actual Result:** Success, no maximum limit hit at 1000 words of Lorem Ipsum
3. **Using the "Cancel" Button:**
    - **Input:** Fill in some or all fields.
    - **Action:** Click "Cancel"
    - **Expected Result:** Form data is cleared/discarded.No ticket is created.
    - **Actual Result:** Form data is cleared/discarded.No ticket is created.
4. **Submit After Validation Failure and Correction:**
    - **Input:** Leave Subject blank, fill Description.
    - **Action:** Click "Submit Ticket" (Expect validation error).
    - **Input:** Fill in the Subject field correctly.
    - **Action:** Click "Submit Ticket" again.
    - **Expected Result:** Success. The form submits correctly after the validation error is fixed.
    - **Actual Result:** Success. The form submits correctly after the validation error is fixed.

Usability/UI Test Cases

1. **Placeholder Text:**
    - **Action:** Observe the form on load.
    - **Expected Result:** Placeholder text ("Brief summary...", "Please provide all details...", "Add comments...") is visible in the respective fields and disappears when the user starts typing.
    - **Actual Result:** All placeholder text appears.
2. **Default Values:**
    - **Action:** Observe the form on load.
    - **Expected Result:** Priority defaults to "Medium". Status defaults to "New". Machine/Assigned to default to "-------". Attachment shows "No file selected.".
    - **Actual Result:** All default values appear.
3. **Dropdown Functionality:**
    - **Action:** Click on each dropdown (Priority, Status, Machine, Assigned to).
    - **Expected Result:** A list of options appears for each dropdown. Selecting an option updates the displayed value.
    - **Actual Result:** All dropdown values appear.

B. User management/ profile testing

Testing Valid Inputs

1. Create User with All Required Fields:

- Input: Username, email, first/last name, role (e.g., "technical_staff"), password, confirm password.
- Action: Click "Create Account."
- Expected: User created. Redirect to account management page. Role/staff/superuser status saved correctly.
- Actual: User created. Redirect successfully.

2. Create User with Staff/Superuser Access:

- Input: Fill required fields + check "Staff Access" or "Superuser Access."
- Action: Click "Create Account."
- Expected: User created with correct permissions.
- Actual: Staff/superuser permissions applied.

3. Create User with Different Roles:

- Input: Test all role options ("user," "technical_staff," "manager," etc.).
- Action: Submit form.
- Expected: Role saved to user profile.

- Actual: Roles persist in the database.

4. Password Complexity Validation:

- Input: Password meets requirements (e.g., 8+ characters, mix of letters/numbers).
- Action: Submit form.
- Expected: Success.
- Actual: Account created.

Error Handling - Invalid/Missing Inputs

1. Missing Required Fields (e.g., username, email):

- Input: Omit username or email.
- Action: Submit form.
- Expected: Error messages for missing fields.
- Actual: Form rejects submission, displays errors.

2. Invalid Email Format:

- Input: Enter "invalid-email."
- Action: Submit form.
- Expected: Error: "Enter a valid email address."
- Actual: Validation fails.

3. Password Mismatch:

- Input: Enter mismatched passwords.
- Action: Submit form.
- Expected: Error: "Passwords do not match."
- Actual: Form rejects submission.

4. Duplicate Username/Email:

- Input: Use an existing username/email.
- Action: Submit form.
- Expected: Error: "Username/Email already exists."
- Actual: Submission blocked.

Boundary and Edge Cases

1. Long Username/Email:

- Input: Username/email at maximum character limit (e.g., 150 chars).
- Action: Submit form.
- Expected: Success.

- Actual: Data saved correctly.

2. Special Characters in Fields:

- Input: Use !@#$%^&*() in username/first name.
- Action: Submit form.
- Expected: Success. Data sanitized/displayed safely.
- Actual: No XSS/SQL injection issues.

3. Cancel Button Functionality:

- Input: Fill fields, click "Cancel."
- Action: Navigate back to account management.
- Expected: Form discarded. No user created.
- Actual: Redirect without saving.

Account Management Page Testing

1. Search Functionality:

- Input: Search by username/email/role.
- Action: Enter query, click "Search."
- Expected: Relevant users displayed.
- Actual: Filters results correctly.

2. Edit User Details:

- Input: Click "Edit" on a user. Update role/name.
- Action: Save changes.
- Expected: Data updated in table.
- Actual: Changes persist after refresh.

3. Delete User (Non-Self):

- Input: Click "Delete" on another user. Confirm.
- Action: Submit deletion.
- Expected: User removed from table.
- Actual: User deleted from database.

4. Delete Own Account Attempt:

- Input: Click "Delete" on your own account.
- Action: Confirm.
- Expected: Button disabled. Error: "Cannot delete your own account."
- Actual: Deletion blocked.

Usability/Security Tests

1. Authorization Checks:

- Action: Non-superuser accesses account management page.
- Expected: Redirect/access denied.
- Actual: Page inaccessible.

2. Form Navigation:

- Action: Tab through fields.
- Expected: Logical order (username → email → password).
- Actual: Correct tab sequence.

3. Error Message Clarity:

- Action: Submit invalid form.
- Expected: Specific errors highlighted (e.g., "Username is required").
- Actual: Messages guide users to fix issues.

11. Conclusion:

The Factory Machinery Status & Repair Tracking System is designed to improve the operational efficiency of high-precision manufacturing environments. By centralizing the tracking of machinery status and streamlining the repair process, the system enables technicians, repair teams, and managers to work more effectively together. It ensures that machinery downtime is minimized, repair workflows are efficiently managed, and managers have access to detailed reports and analytics that can guide decision-making.

This project, deployed using Docker, provides a consistent and reliable environment for running the application across different platforms. Dockerization ensures that the system can be easily deployed, tested, and maintained without dependency conflicts, improving the overall robustness and scalability of the application.