ProductionOptimiser - Design Description

1. Project Background

The project was proposed by MITC, main customers being Vladana Celebic, John Moberg, Akshay Goyal and Nils Erlands. As an answer to modern manufacturing's needs for production flow optimization, MITC has developed an AI-drive production flow optimizer. By inputting data on different machines and products, the tool calculates the most optimal production sequence. This solution is currently based on data input through Excel files, making it easy to use and efficient compared to more complex alternatives. ProductionOptimiser would provide MITC with a web platform that utilizes MITC's developed algorithms.

2. High level description of the system

The project aims to develop a scalable web platform that would utilize MITC's algorithms. Therefore, there is a strong emphasis on the platform's extensibility as new algorithms and tools are being developed. Users need to be able to log into the platform, upload their production data, and access optimized production sequences. Furthermore, user's data needs to maintain confidentiality and integrity throughout the optimization process. A standalone API independent from the web interface that has access to the optimization algorithms needs to be developed. The API also needs to be secured, using the same credentials as the web interface. This puts a minimum requirement of two separate components to be developed, a web interface and an API that utilizes optimization algorithms while also being able to fetch past optimized production sequences. There also needs to be a suitable mechanism for storing past optimized production sequences and input data related to them. To summarize, the system would need to consist of a database for storage, a web interface for displaying optimization results and data and an API that is able to utilize optimization algorithms, assuming the optimization algorithm is either part of the API or is executed in an environment where it is invokable by the API logic and retrieve data from the database. For a more comprehensive list of requirements, refer to the Requirements Definition document.

3. System Overview

The platform runs in a web context i.e., exclusively uses HTTPS for communication with users and between components. Every component, excluding the web interface

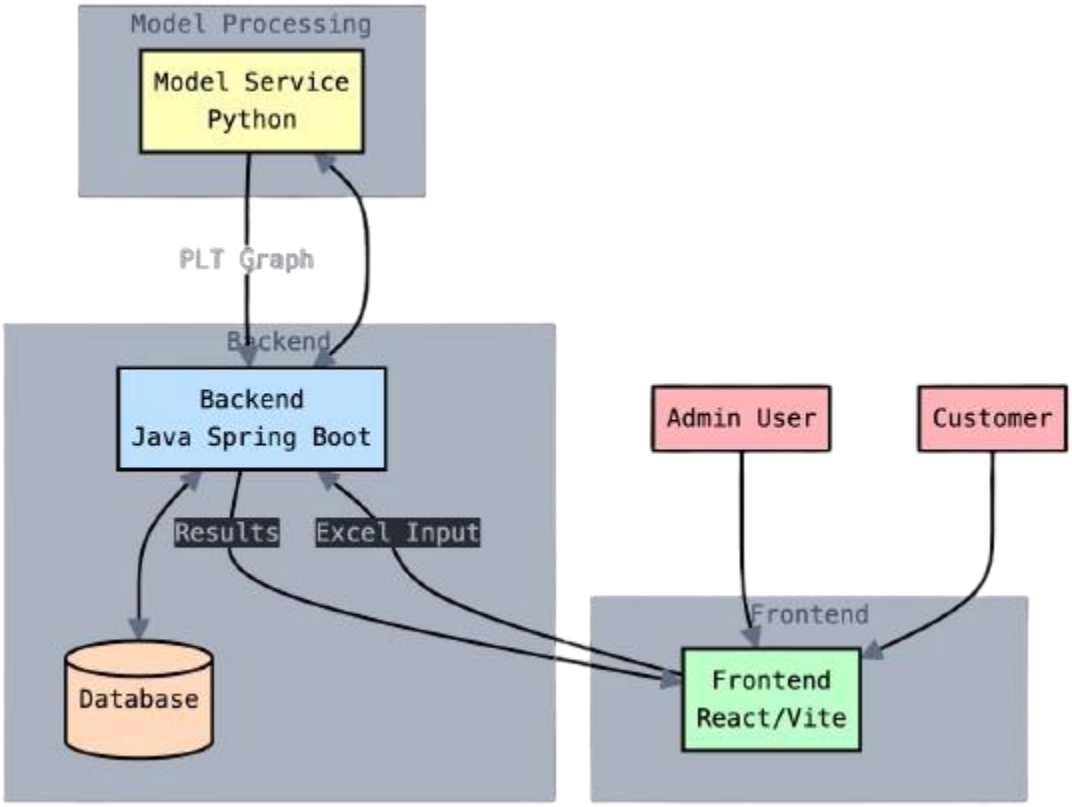
application, of the system runs in a UNIX based Docker container, allowing them to be easily moved and deployed in the cloud. Backend API uses Java Runtime Environment to run Tomcat server that is used as part of Spring Framework. Python optimization algorithms use Python 3.10 or higher to run.

4. Software architecture

4.1. Architecture:

Components of the architecture:

- Web Interface – React, Vite
- Backend REST API for fetching optimization results and utilizing optimization algorithms – Spring Boot Framework, Java
- Database – PostgreSQL
- 1..N optimization algorithms and other tools exposed as REST API microservices – Python, Fast API



Components communicate with each other via HTTPS protocol. API and models send data in JSON format. Frontend displays this JSON data in a more readable format.

4. 2. Technical considerations for architectural design

Due to the explicit requirement to be able to access optimization models and past data independent from the web interface the decision was made to separate the API and web platform as opposed to it being a single application that renders view on the server. The web interface instead requests data from API via HTTP method like GET, POST, PUT and delete. This way, even if there is no web interface, users can access past optimized production sequences and request optimization runs from algorithms via curl, Postman or any other tool that offers HTTP communication.

Another important consideration was how to include the optimization algorithms, the options being integrating them in the API or have them as standalone services. We decided to opt for the microservice methodology to enable extending the platform with other algorithms and tools. Specifically, each potential MITC tool is wrapped inside of an API that is deployed separately and functions as a service, being invokable via HTTP. Having microservices deployed separately gives us the option to, for example, use service discovery to dynamically add new algorithms into the platform, satisfies the requirement to be able to add new tools into the system from the GUI and makes the platform movable and scalable by allowing replication of services, starting services on demand as well as load balancing.

4.3. Component responsibilities and requirements mapping

4.3.1 Frontend/web interface

Responsibilities:

- Handles user interfaces for both Admin and Regular users
- Ensures responsive and intuitive design across devices
- Communicates with backend for all user interactions
- Displays optimized production sequences in a readable format
- Provides interface for user to choose which tool the user uses
- Provides interface for admins to add new MITC tools the platform will use
- Provides interface to upload production data used for optimization
- Provides users the ability to download past optimized production sequences and production input data that was used for the specific optimization

- Provides interface for admins to add new users and manage tools users can access

Technologies – React, Shadc, Vite

4.3.2 Backend REST API

Responsibilities:

- Handles user authentication and authorization via JWT tokens
- Call optimization algorithm services
- Persists optimization inputs and results into database
- Fetches past optimized production sequences and sends them as response to HTTP request
- Enables creation of new users
- Handles persistence of optimization models and other tools
- Enables export of previous input data and results in Excel
- Enables addition of new services and tools into the platform
- Enables management of available tools to user

Technologies – Spring Framework, Java

4.3.3 Optimization algorithm microservice

Responsibilities:

- Exposes optimization algorithms via a REST API
- Runs MITC's production sequence optimization algorithm
- Provides resulting sequences and graphs in JSON format

Technologies – Python, Fast API

4.3.4 Postgres relational database

Responsibilities:

- Store user credentials and data

- Stores past input data
- Stores past optimized production sequences and graphs

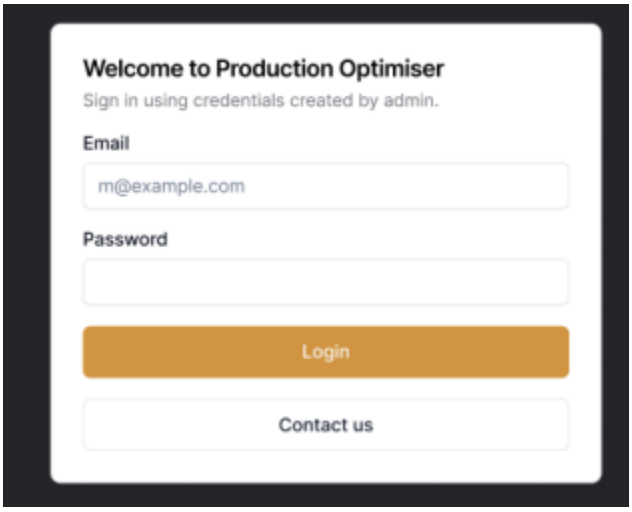
Technologies – Postgres

4.4. Security and access control

The platform supposes 2 roles, user and administrator. Administrators have all the functionality normal users have with the addition of being able to invite new users onto the platform as well as manage which tools and algorithms normal users can access. Authentication is done through JWT tokens configured with Spring Security provided inside Spring Framework. The REST API authorizes requests only from users who have successfully been authenticated i.e., have a valid Bearer Token as part of their request. As for algorithm services, they are protected with CORS rules allowing only requests from the central API to be authorized, all others are forbidden.

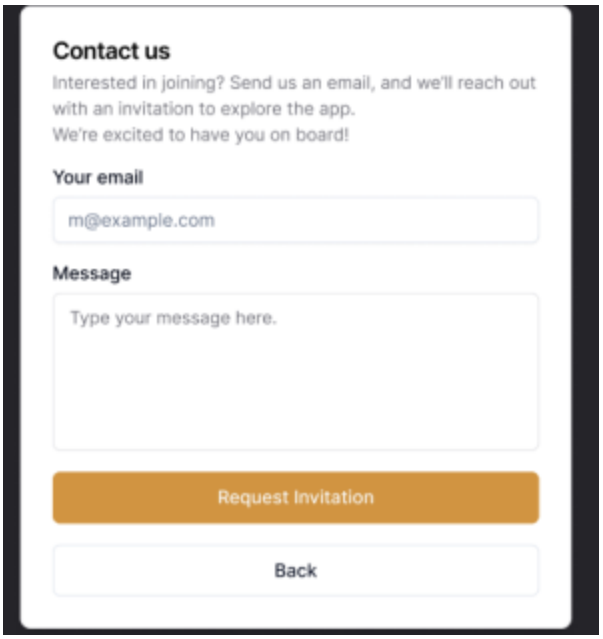
5. Graphical user interface

5.1. Login screen



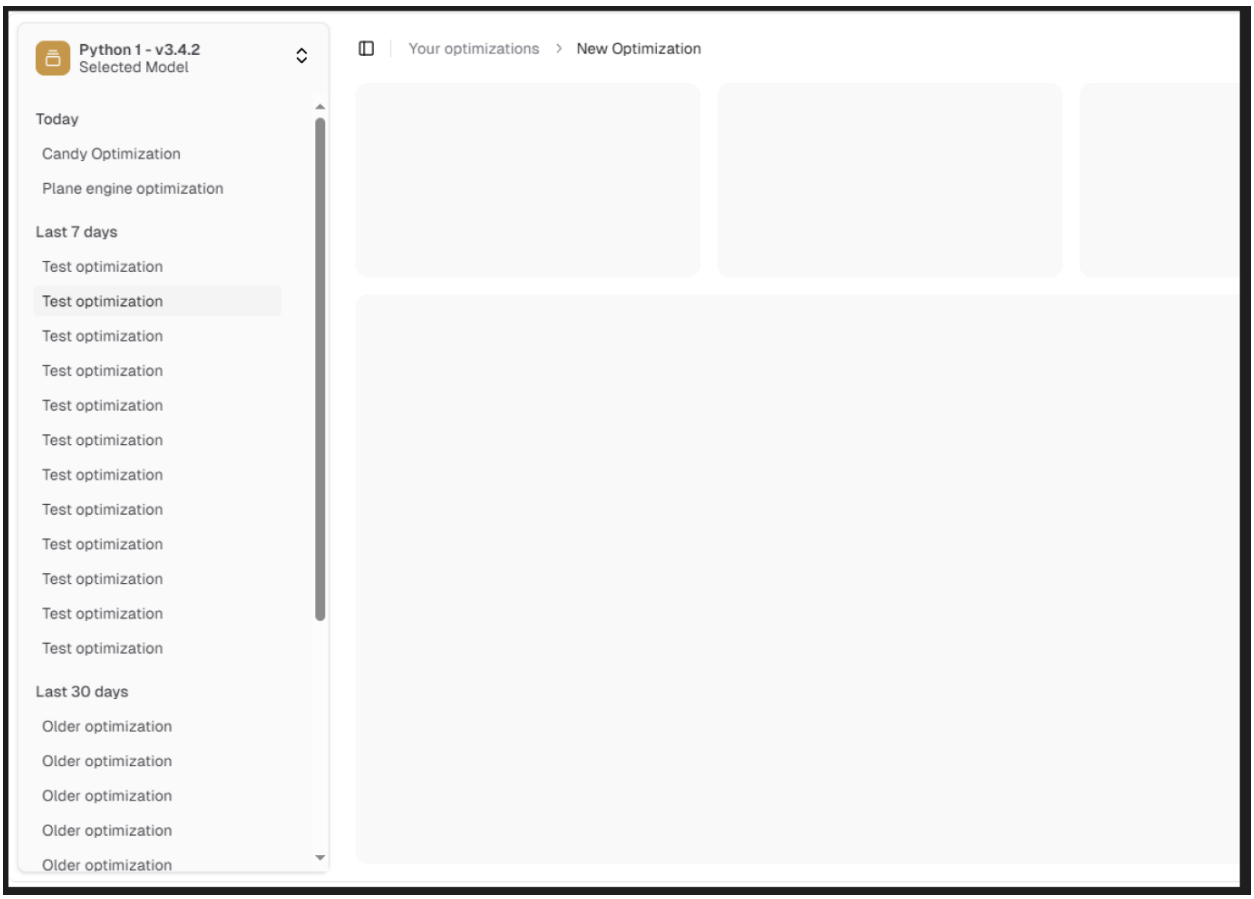
Login screen enables users to log into the platform, receive a valid JWT Token and access the platform

5.2. Registration/permission inquiry form



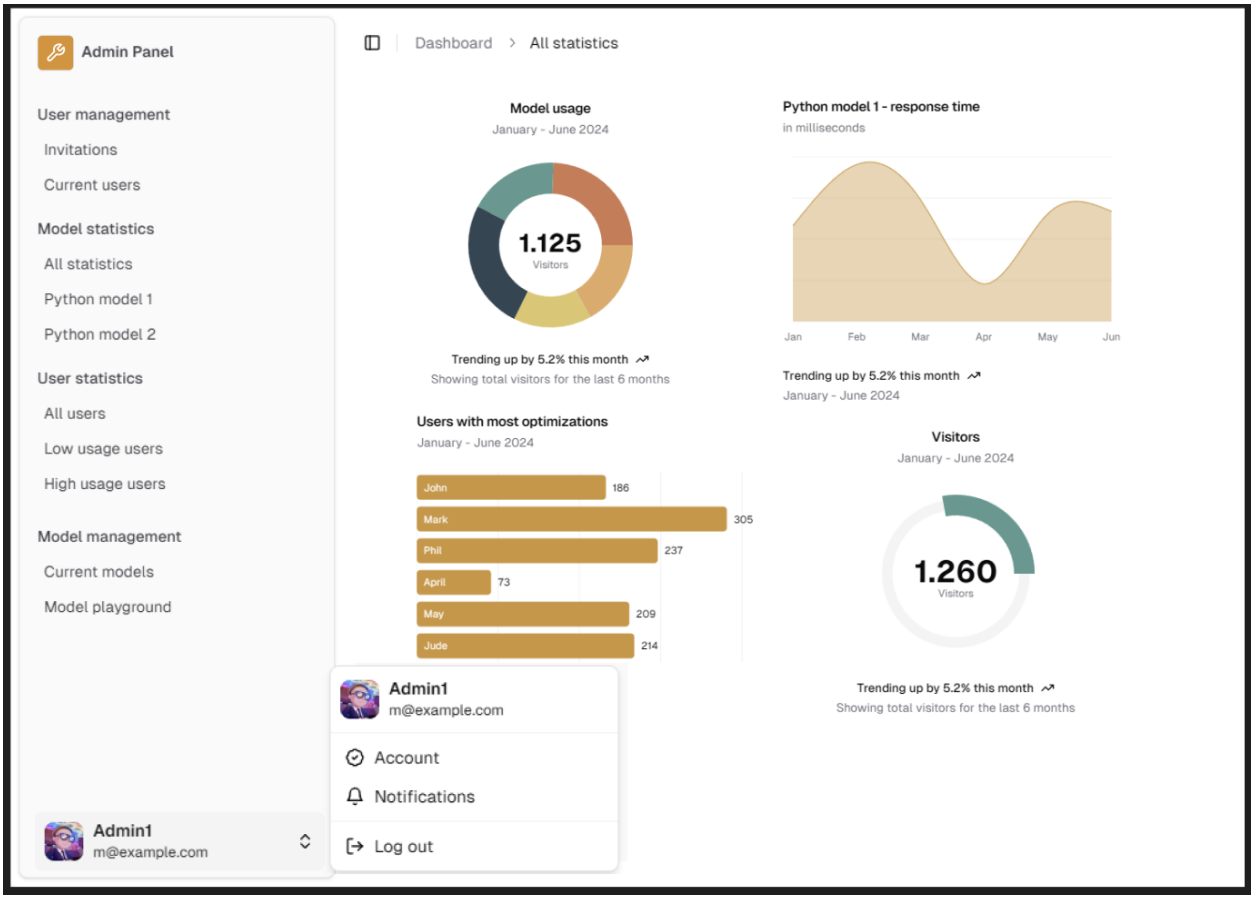
This screen provides users with a way to request access to the platform, sends request to administrators to add users credentials into the platform

5.3. Optimizations and history view



Main screen that shows optimization history, production sequences

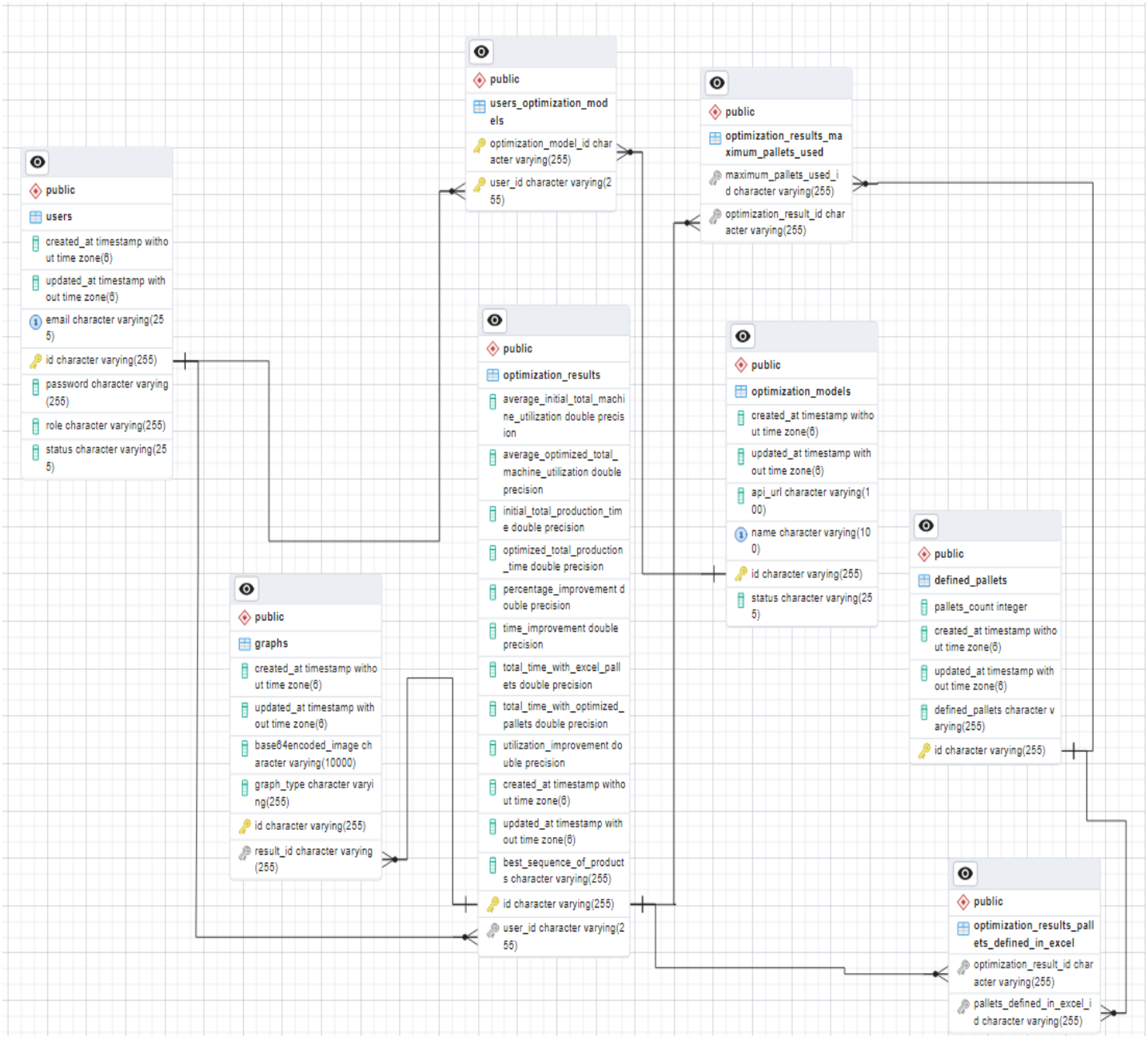
5.4. Model metrics and statistics



Metric and statistic are being tracked per model, showing usage, response time, users and other statistics

6. Detailed software design

6.1 Database design



Entities:

All entities share 3 columns, id that is represented as varchar(255) data type, and two timestamp fields created_at and updated_at.

User – stores details and credentials about user, specifically email, hashed password, role and status of his account.

Graph – represents the graph that is generated by production optimization algorithm. It can be OCCUPANCY_GRAPH, MACHINE_UTILIZATION, PRODUCT_FLOW and PALLET_GRAPH. The base64encoded_image represents the image of the graph that is converted to bytes and encoded with the Base64 algorithm.

Result – stores the output of the production optimization algorithm.

Defined pallets – stores information about number of pallets for each product group

Optimization result pallets defined in excel – join table for result and defined pallets, representing pallets that were defined in the excel production input data

Optimization result maximum pallets used – join table for result and defined pallets, representing maximum pallets used after optimization process.

Optimization model – represents the optimization model that is defined with its URL where it is deployed and status being ACTIVE or RETIRED

Users optimization models – join table that maps users to models they have access to