



**POLITECNICO**  
MILANO 1863



UNIVERSITY OF ZAGREB

Faculty of Electrical  
Engineering and  
Computing

# **DSD**

## **Design Document for “Production Optimizer”**

**Authors:** Fran Hruza, Danko Čurlin, Georgs Lukass Rozins, Paula Šalković,  
Shreesh Kumar Jha, Giovanni Orciuolo, Samarth Bhatia

**Version:** 3

**Date:** 12.01.2025.

**Supervisors:** Elisabetta Di Nitto (POLIMI), Igor Čavrak (FER)

## Table of Contents

1. Project Background .....	3
2. High level description of the system.....	3
2.1 Core Functionalities .....	3
2.2 Requirements Coverage .....	4
3. System Overview .....	6
3.1 System Architecture and Interfaces .....	6
3.2 Communication and Integration .....	7
3.3 Deployment and Infrastructure .....	7
4. Software Architecture .....	8
4.1. Architecture: .....	8
4. 2. Technical Considerations for Architectural Design .....	8
4.3. Component Responsibilities .....	9
4.3.1 Frontend/web Interface .....	9
4.3.2 Backend REST API .....	9
4.3.3 Service Tools (Optimization Algorithms) .....	10
4.3.4 PostgreSQL Relational Database .....	10
4.4. Security and access control .....	11
5. Graphical user interface .....	12
6. Detailed software design .....	21
6.1 Database schema .....	21
6.2. Backend classes and methods .....	23
6.3.1 Service package detailed look .....	27
6.4. Class diagrams .....	31

# 1. Project Background

Current manufacturing industries always have faced and will face complexity in optimizing production processes, from resource allocation to scheduling. To address these challenges, MITC has developed an AI-driven Production Flow Optimizer. This tool processes data about machines and products to calculate the most efficient production sequences, offering tangible improvements in efficiency and cost-effectiveness.

The Optimizer operates using Excel data inputs with specific guidelines on the file's structure making it easy to use and efficient compared to more complex alternatives. However, to expand its accessibility and scalability, the goal was to make Production Flow Optimizer into a web-based platform, which utilizes MITC's already developed algorithms.

The project was proposed by MITC, main stakeholders include Vladana Celebic, John Moberg, Akshay Goyal, and Nils Erlands.

## 2. High level description of the system

The Production Flow Optimizer platform is a scalable web-based system. The platform will provide users with tools to upload production data, execute MITC's optimization algorithms, and visualize optimized production sequences. It emphasizes security, scalability, and extensibility to ensure its relevance with new algorithms introduced.

### 2.1 Core Functionalities

#### 1. User Authentication and Role Management:

- Secure login and registration mechanisms with role-based access control (e.g., administrators and regular users). (*Refer to FR1–FR4 in Section 2.1.1 of the Requirements Definition document.*)
- Administrators can manage user roles, permissions, and access to optimization tools. (*Refer to FR3 in Section 2.1.1. of RASD*)

#### 2. Data Management and Input Processing:

- Support for uploading production data in Excel format, with robust error handling and validation mechanisms. (*Refer to FR9 in Section 2.1.3. of RASD*)
- Storage of historical production data and results for trend analysis and comparison. (*Refer to FR12 in Section 2.1.3. of RASD*)

3. Service Tool Execution:

- Integration with MITC's proprietary optimization algorithms to process production data and generate optimized production sequences. (*Refer to FR5–FR7 in Section 2.1.2. of RASD*)
- Support for concurrent execution of multiple optimization runs. (*Refer to FR5–FR7 in Section 2.1.2. of RASD*)

4. Visualization and Analytics:

- Clear, interpretable visualizations of optimization results, including graphs and comparative analyses. (*Refer to FR10–FR11 in Section 2.1.3. of RASD*)
- Tools to track usage statistics, performance metrics, and service tool efficiency. (*Refer to FR13 in Section 2.1.4 and NFR6 in Section 2.2.2. of RASD*)

5. Extensibility and API Integration:

- A modular system architecture allowing the addition of new optimization algorithms and tools. (*Refer to FR5 and FR8 in Section 2.1.2. of RASD*)
- A RESTful API providing programmatic access to optimization functionalities and historical data for external integrations. (*Refer to FR8 in Section 2.1.2. of RASD*)

6. System Security and Reliability:

- End-to-end encryption for data security and compliance with privacy standards. (*Refer to NFR1–NFR4 in Section 2.2.1. of RASD*)
- High availability with error-handling mechanisms, automated backups, and monitoring. (*Refer to NFR7–NFR8 in Section 2.2.2. of RASD*)

## 2.2 Requirements Coverage

This system will implement all the functional and non-functional requirements detailed in the Requirements Definition document. These include but are not limited to:

- User Authentication and Management (*refer to Section 2.1.1 of the Requirements Definition document*).
- Service Tool Management (*refer to Section 2.1.2 of RASD*).
- File Operations (*refer to Section 2.1.3 of RASD*).
- Analytics and Monitoring (*refer to Section 2.1.4 of RASD*).

- Security Requirements (*refer to Section 2.2.1 of RASD*).
- Performance Requirements (*refer to Section 2.2.2 of RASD*).

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 invocable 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

### 3.1 System Architecture and Interfaces

The system architecture is designed for modularity and scalability, with distinct components for the frontend, backend, optimization services, and database. Key architectural elements include:

#### 1. Frontend

- Implemented using React and Vite.
- Handles user authentication and interactions for data uploads, results retrieval, and administrative functions.
- Communicates with the backend via HTTPS (*Refer to [FR1–FR4] in Section 2.1.1 and [FR10–FR11] in Section 2.1.3 of RASD*).

#### 2. Backend API

- Acts as the system's central hub, developed using Spring Boot.
- Provides RESTful endpoints for user management, optimization execution, and historical data retrieval (*Refer to [FR8] in Section 2.1.2 and [FR12] in Section 2.1.3 of RASD*).
- Integrates with external systems and optimization services through secure API calls.

#### 3. Optimization Services

- Implemented as standalone microservices using Python and FastAPI.
- Each service encapsulates a specific MITC optimization algorithm, ensuring scalability and extensibility (*Refer to Section 2.3.3 in RASD*).
- Processes inputs and returns outputs in JSON format to the backend.

#### 4. Database

- A PostgreSQL database securely stores user data, historical results, and metadata about optimization services (*Refer to [FR12] in Section 2.1.3 and [FR13] in Section 2.1.4 of RASD*).
- Ensures data consistency and availability for reporting and analytics.

## 3.2 Communication and Integration

All components communicate securely via HTTPS, ensuring data protection and compliance with security standards (*Refer to [NFR3] in Section 2.2.1 of RASD*). Key integration points include:

- **Frontend-Backend Communication:** The frontend interacts with the backend for all user operations, including data uploads, results retrieval, and user management.
- **Backend-Optimization Services Communication:** The backend dynamically invokes microservices for optimization tasks.
- **Database Integration:** Data is securely stored and retrieved using efficient queries, ensuring high performance under load.

## 3.3 Deployment and Infrastructure

The system is deployed in a cloud environment with containerized components (Docker) to enable horizontal scaling and maintainability (*Refer to Section 2.2.3 in RASD for Optimization Environment*). This architecture supports dynamic resource allocation, load balancing, and fault tolerance to ensure high availability (*Refer to [NFR7–NFR8] in Section 2.2.2 of RASD*).

- By separating its components into loosely coupled modules, the system achieves:
- **Scalability:** Easy addition of new optimization algorithms or tools.
- **Reliability:** Resilience through isolated services and robust error handling.
- **Extensibility:** Integration of new tools and features without disrupting existing workflows.

## 4. Software Architecture

### 4.1. Architecture:

Components of the architecture:

- Web Interface – React, Vite (*Refer to Section 2.7.1 of RASD*).
- Backend REST API for fetching optimization results and utilizing optimization algorithms – Spring Boot Framework, Java (*Refer to Section 2.7.2 of RASD*).
- Database – PostgreSQL
- Optimization Algorithms as Microservices – REST API services, implemented using Python and FastAPI, these microservices encapsulate MITC's service tools (*Refer to Section 2.7.3 and 2.6.4 of RASD*).

All components communicate via HTTPS, ensuring secure data transmission (*Refer to NFR3 in Section 2.6.1 of RASD*). Data is exchanged between the components in JSON format, with the frontend transforming it into readable and interactive visualizations.

### 4.2. Technical Considerations for Architectural Design

Due to the explicit requirement to allow access to service tools and past data independent of the web interface, the architecture separates the API and web platform. This decision ensures that users can programmatically interact with the platform using tools such as cURL, Postman, or other HTTP-based clients, even without accessing the web interface. The frontend interacts with the backend via HTTP methods like GET, POST, PUT, and DELETE, ensuring flexibility and usability across different access points (*Refer to Section 2.1.2 and 2.8.12 of RASD*).

Another critical consideration was the integration of optimization algorithms. The architecture adopts a microservice methodology, where each optimization algorithm is deployed as a standalone service. These MITC developed tools exposed as microservices will further be referred to as service tools, are wrapped within their own APIs and deployed independently. This approach enables:

- **Dynamic Service Discovery:** New algorithms can be seamlessly added to the platform without affecting existing functionality.
- **Extensibility:** The system supports the addition of new tools directly from the GUI.



- **Scalability:** Independent deployment of microservices allows for replication, on-demand service scaling, and load balancing.

This design satisfies the requirement for modularity and extensibility, allowing the platform to evolve as new tools and algorithms are developed. Furthermore, separating these components ensures that optimization services remain accessible, even in the absence of the web interface (*Refer to Section 2.6.4 of RASD*).

## 4.3. Component Responsibilities

### 4.3.1 Frontend/web Interface

Responsibilities:

- Provides user interfaces for both administrators and regular users
- Ensures responsive and intuitive design across devices
- Handles:
  - Uploading of input data (files or text) (*Refer to [FR9] in Section 2.1.3 of RASD*).
  - Downloading past results and inputs.
  - Visualization of optimized production sequences (*Refer to [FR10] in Section 2.1.3 of RASD*).
  - Administration tasks, such as managing user roles and adding new service tools (*Refer to [FR3] in Section 2.1.1 of RASD*).
- Communicates with the backend via HTTPS to send and receive data.

Technologies – React, Shadc, Vite.

### 4.3.2 Backend REST API

Responsibilities:

- Manages user authentication and authorization with JWT tokens (*Refer to FR1 and FR3 in Section 2.1.1 of RASD*).
- Provides endpoints for:
  - Invoking optimization algorithms (*Refer to [FR5] in Section 2.1.2 of RASD*).

- Managing service tools (adding, updating, retiring) (*Refer to FR6 in Section 2.1.2 of RASD*).
- Fetching historical data and exporting results (*Refer to [FR12] in Section 2.1.3 of RASD*).
- Tracking and reporting usage statistics (*Refer to [FR13] in Section 2.1.4 of RASD*).
- Acts as the intermediary between the frontend, database, and optimization services.

Technologies – Spring Framework, Java.

#### 4.3.3 Service Tools (Optimization Algorithms)

Responsibilities:

- Encapsulate MITC's optimization algorithms as independent microservices.
- Accept inputs (file, string, or image) and provide outputs in JSON format.
- Ensure modularity, enabling easy addition or modification of algorithms (*Refer to Section 2.6.4 of RASD*).

Technologies – Python, Fast API.

#### 4.3.4 PostgreSQL Relational Database

Responsibilities:

- Stores user credentials, service tool metadata, historical inputs, and outputs (*Refer to Section 3.1.1 of RASD*).
- Maintains user-service tool associations and tracks usage statistics (*Refer to [FR13] in Section 2.1.4 of RASD*).
- Ensures data consistency and supports reporting functionalities.

Technologies – PostgreSQL

## 4.4. Security and access control

The platform implements robust security measures to ensure data confidentiality, integrity, and availability:

### **Authentication and Authorization:**

- User authentication is managed via JWT tokens, configured using Spring Security (*Refer to [FR1] in Section 2.1.1 and [NFR1] in Section 2.6.1 of RASD*).
- Role-based access control (RBAC) defines two roles:
  - Administrator: Full access, including user and tool management.
  - Regular User: Restricted access to authorized tools and functionalities (*Refer to [FR7] in Section 2.1.2 of RASD*).

### **Service Tool Security:**

- Optimization services are protected by CORS policies, allowing only requests from the central backend (*Refer to Section 2.6.4 of RASD*).
- Each service tool validates input data to prevent malicious or erroneous requests.

### **Encryption:**

- All data transmissions are encrypted using SSL/TLS (*Refer to [NFR2] in Section 2.6.1 of RASD*).
- Sensitive data, such as passwords, is securely hashed and stored (*Refer to Section 3.3.1 of RASD*).

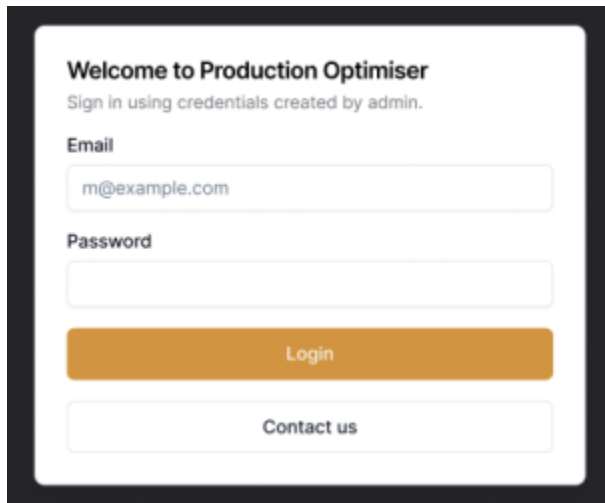
### **Audit Logs:**

All critical actions, such as user management and tool configuration, are logged for accountability and troubleshooting (*Refer to Section 2.6.4 of RASD*).

### **Session Management:**

- Automatic session timeout ensures inactive users are logged out (*Refer to [FR4] in Section 2.1.1 of RASD*).
- Secure token storage prevents unauthorized access.

## 5. Graphical user interface



**Welcome to Production Optimiser**  
Sign in using credentials created by admin.

**Email**

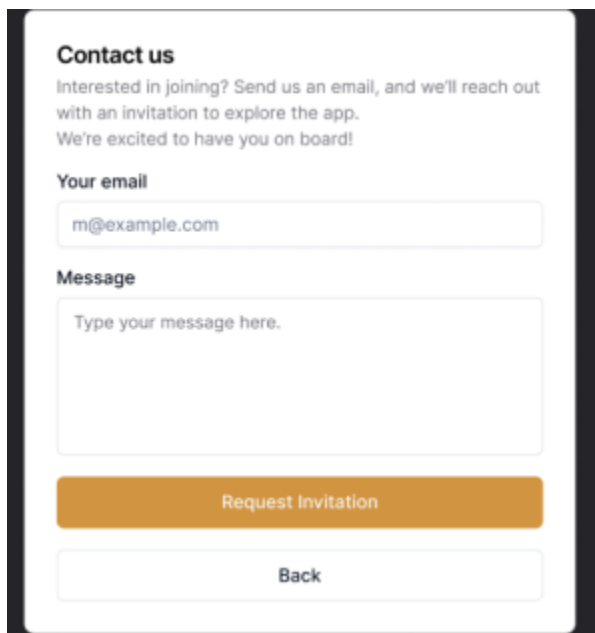
**Password**

**Login**

**Contact us**

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

Pressing on the Contact us screen:



**Contact us**  
Interested in joining? Send us an email, and we'll reach out with an invitation to explore the app. We're excited to have you on board!

**Your email**

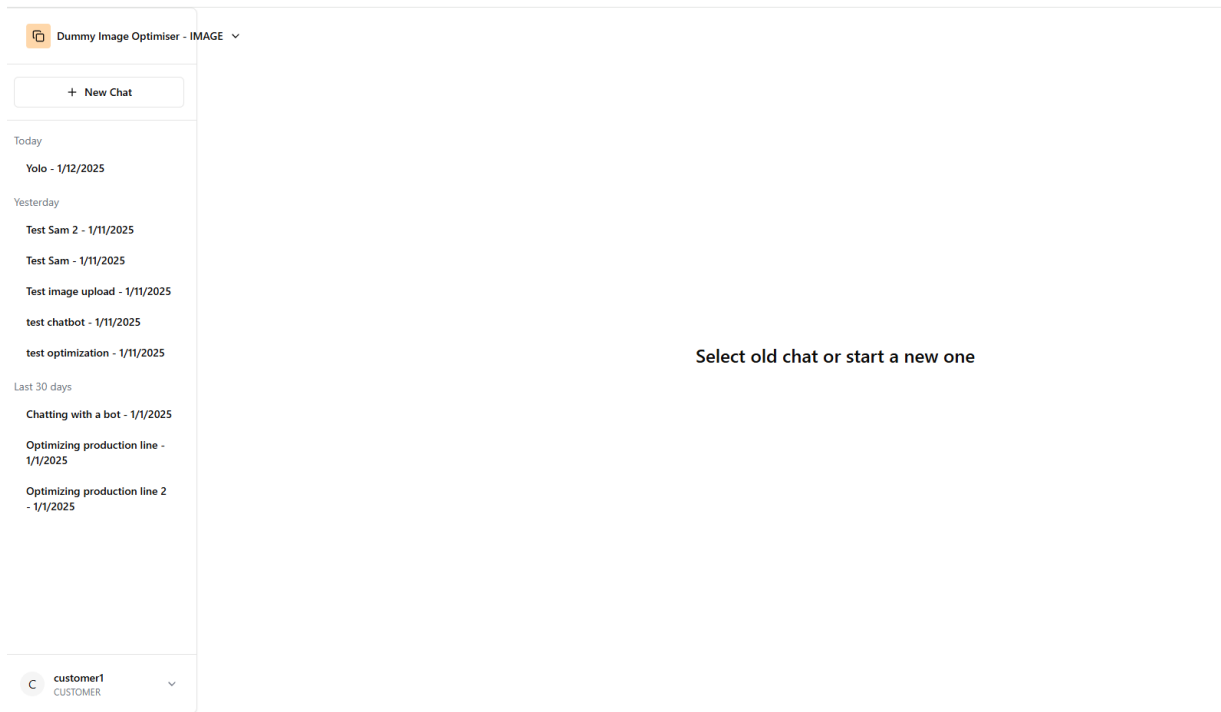
**Message**

**Request Invitation**

**Back**

This screen provides users with a way to request access to the platform, sends request to administrators to add user's credentials into the platform. By pressing the Back button, the user is brought back to the login screen.

Once user with role CUSTOMER has logged in, he will be greeted by the following screen:



On the left he can see previous usages of service tools. Clicking on one of them will bring him to the results screen, showing him the results of the service tool usage.

Above that there is a select list of available service tools and below it is the “New chat” button which takes user to screen where he can invoke the service tool by providing the appropriate input data depending on the service tool:

**New Service Execution**

Upload your file to invoke the selected service service tool.

**Service Tool**

Real AI Service - FILE

**Service Execution Name**

Enter Service Execution Name

**Input File**

Drop files to attach, or browse

Allowed file types: XLSX, XLS, CSV

Send

Uploading appropriate data and pressing the Send button invokes the service tool and after execution if finished, it takes the user to the results screen which varies based on the result the service tool provides. Here is an example of service tool that optimizes production sequences for various products and manufacturing data:

Input File

Download Input File

Product Flow

Download

Product flow through production steps and total time

Production Steps

- al wash
- cool 1
- cool 2
- cool 3
- cool 4
- cool 5
- dry
- fin kpress
- hoppingpress
- irradiation
- rinse
- vision control
- wash 1

Occupancy Graph

Download

Initial Sequence Occupancy

Optimized Sequence Occupancy

Total Time: 826.00 min

Machine Utilization

Download

Results

Time Improvement

304.50

Percentage Improvement

37.32

Best Sequence Of Products

h skibidi b skibidi f g skibidi i skibidi skibidi skibidi f e c h b e c c g h c d c d d h d d d e i g e c e i b e e skibidi skibidi c c c e e i b g b b f skibidi f f f h f i b g g d f g g f g skibidi d e h c h h b b i b f d i i i

Initial Total Production Time

816.00

Optimized Total Production Time

511.50

Average Initial Total Machine Utilization

31.34

Maximum Pallets Used

{ "d,e,f": 4, "g,h,i": 6, "skibidi,b,c": 10 }

Utilization Improvement

18.66

Pallets Defined In Excel:

{ "d,e,f": 10, "g,h,i": 10, "skibidi,b,c": 10 }

Total Time With Excel Pallets

511.50

Total Time With Optimized Pallets

511.50

Average Optimized Total Machine Utilization

50.00

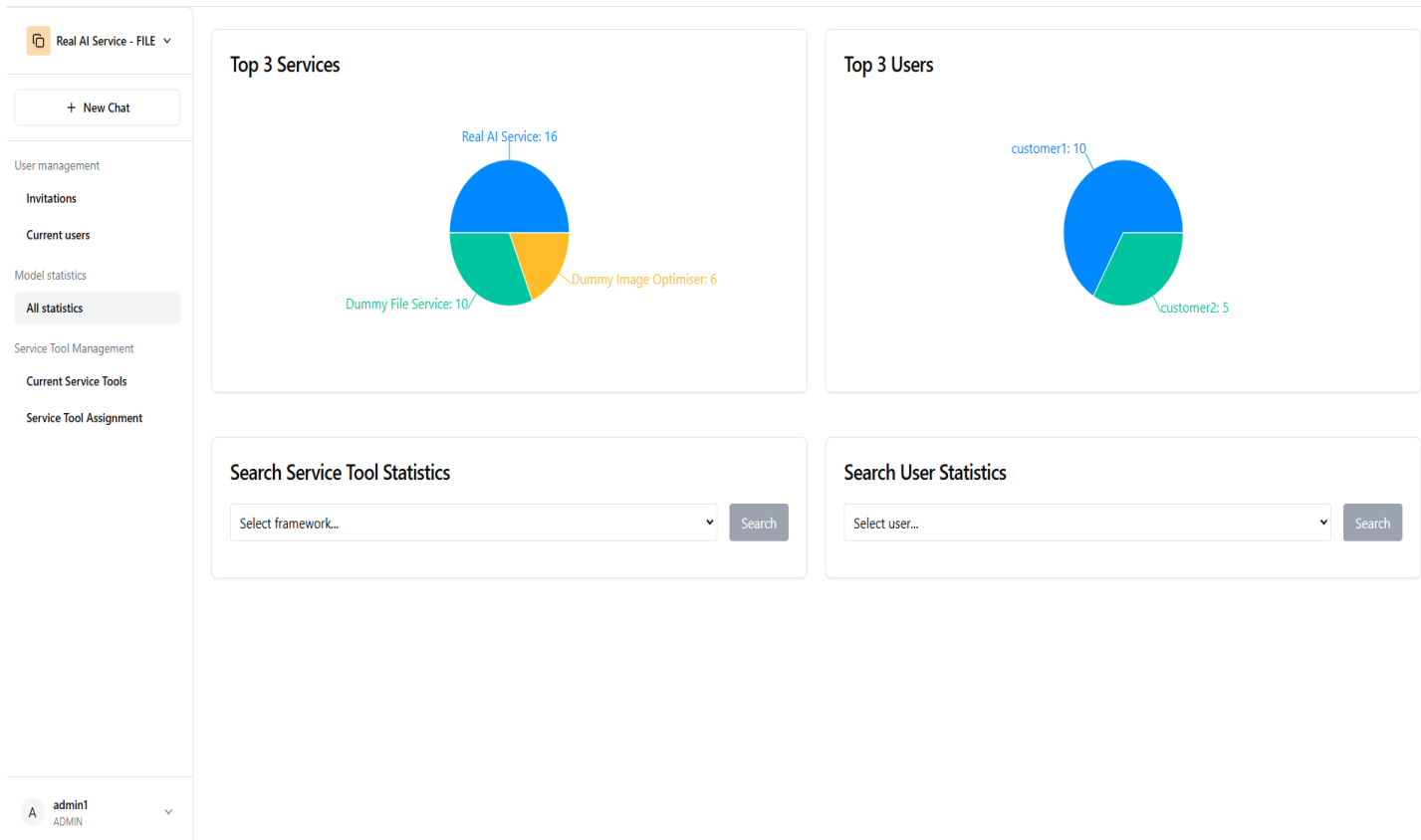
Raw Data

Download JSON

```
{  "d,e,t": 10,  "g,h,i": 10,  "skibidi,b,c": 10  },  "initial_total_production_time": 816,  "total_time_with_excel_pallets": 511.5,  "optimized_total_production_time": 511.5,  "total_time_with_optimized_pallets": 511.5,  "average_initial_total_machine_utilization": 31.34,  "average_optimized_total_machine_utilization": 50  },  "userId": "550e8400-e29b-41d4-a716-446655440002",  "inputFile": "UESDBAoAAAAAT16D1kAAAAAAAAAAAAAAAAAGAAAX3J1bHMvUESDBAoAAAAAT16D11qv7zG5gAAAEoCAALAAAAAX3J1bHMvLnJ1bHotksFKxDaQh181zH073RVEZLN7EWFvIvUBxmTALW0zIRm1+/ZGD2Jhf",  "inputString": null  }
```

In addition to visualizing the result of the service tool invocation, the result screen also has download buttons for downloading the input that was passed to the service tool, be it a file or some kind of text. Furthermore, if service tools produced images or files, the user can download those and the raw JSON file the service tool produced containing the result in JSON format.

Now we will look at the view users with ADMIN roles have after they logged in:



Administrator view shows some statistics like top 3 most used service tools and top 3 users based on the number of logins. The administrator also has access to all service tools and can invoke them as shown in previous screens. Pressing on the Invitations button bring him to the invitation screen:

Real AI Service - FILE

+ New Chat

User management

Invitations

Current users

Model statistics

All statistics

Service Tool Management

Current Service Tools

Service Tool Assignment

User Requests

Search by name or email...

Name	Email	Status	Created At	Actions
N/A	account.request1@mail.com	Pending	Invalid Date	<div>View Details</div> <div>Approve</div> <div>Deny</div>
N/A	account.request2@mail.com	Pending	Invalid Date	<div>View Details</div> <div>Approve</div> <div>Deny</div>
N/A	account.request3@mail.com	Pending	Invalid Date	<div>View Details</div> <div>Approve</div> <div>Deny</div>
N/A	account.request4@mail.com	Pending	Invalid Date	<div>View Details</div> <div>Approve</div> <div>Deny</div>

In the invitation screen, admins can see all requests from potential users for access to the platform. Administrator can approve or deny these requests and view more details about the request.

Here are the popup windows that appear when these options are pressed:

Request Details

Name

N/A

Email

account.request1@mail.com

Message

Please add me as a user.

Status

Pending

"View details" window



×

Approve user account.request1@mail.com

Password

Cancel

Approve

“Approve” window

×

Deny user account.request1@mail.com

Cancel

Deny

“Deny” window

Returning to the navbar in the left part of the screen, administrator can also see current users of the platform and filter them by email:

### Manage Users

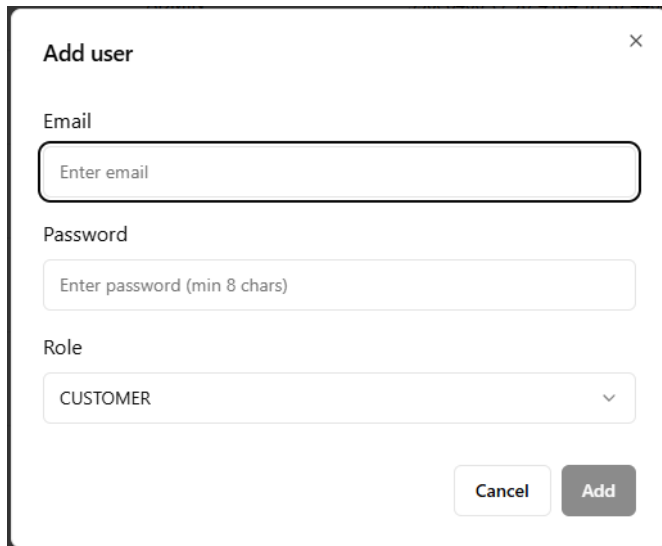
Filter emails...

Add User

<input type="checkbox"/>	Email	Status	Role	Id	Created At	Actions
<input type="checkbox"/>	admin1	ACTIVE	ADMIN	550e8400-e29b-41d4-a716-446655440000	Invalid Date	...
<input type="checkbox"/>	admin2	ACTIVE	ADMIN	550e8400-e29b-41d4-a716-446655440001	Invalid Date	...
<input type="checkbox"/>	customer1	ACTIVE	CUSTOMER	550e8400-e29b-41d4-a716-446655440002	Invalid Date	...
<input type="checkbox"/>	customer2	DELETED	CUSTOMER	550e8400-e29b-41d4-a716-446655440003	Invalid Date	...
<input type="checkbox"/>	test1@test1.com	ACTIVE	CUSTOMER	6d98f12f-ce13-4873-a2e8-71cf4c6a5d56	Invalid Date	...

Administrators can also add new users manually by pressing the “Add User” button.

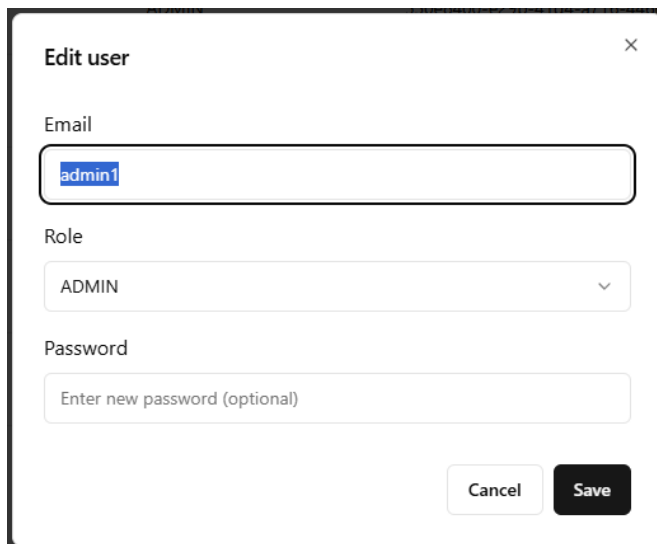
“Add User” button then shows following screen where he gets to input the new user’s credentials and assign a role to him which dictates the content the user has access to on the platform:



The "Add user" modal form contains the following elements:

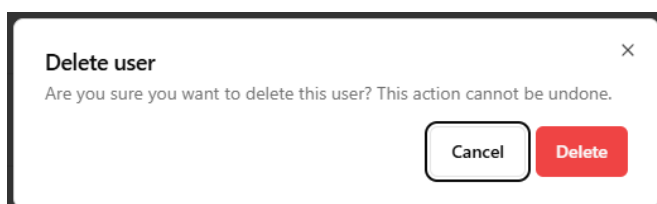
- Title:** Add user (with a close button 'x' in the top right corner).
- Email:** A text input field with the placeholder text "Enter email".
- Password:** A text input field with the placeholder text "Enter password (min 8 chars)".
- Role:** A dropdown menu currently showing "CUSTOMER" with a downward arrow.
- Buttons:** "Cancel" and "Add" buttons at the bottom right.

Administrator also has the option of deleting user from the platform or edit the credentials by pressing the three dots icon in the last “Actions” column of the view, having the following screens based on the chosen option:



The "Edit user" modal form contains the following elements:

- Title:** Edit user (with a close button 'x' in the top right corner).
- Email:** A text input field containing the text "admin1" with a blue selection highlight.
- Role:** A dropdown menu currently showing "ADMIN" with a downward arrow.
- Password:** A text input field with the placeholder text "Enter new password (optional)".
- Buttons:** "Cancel" and "Save" buttons at the bottom right.



The "Delete user" modal form contains the following elements:

- Title:** Delete user (with a close button 'x' in the top right corner).
- Message:** A warning message: "Are you sure you want to delete this user? This action cannot be undone."
- Buttons:** "Cancel" and "Delete" buttons at the bottom right.

Pressing on the “Current Service Tools” button, administrator can view all available service tools on the platform which administrator can filter by name:

Manage Models

<input type="checkbox"/>	Name	URL	Input Type	Id	Created At	Actions
<input type="checkbox"/>	Real AI Service	https://dsd-tool.commanderkowalski.uk/optimize	FILE	550e8400-e29b-41d4-a716-446655440004	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy File Service	https://dsd-dummy-tool.commanderkowalski.uk/service	FILE	550e8400-e29b-41d4-a716-446655440005	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy Chatbot	https://dsd-dummy-text-tool.commanderkowalski.uk/chatbot	STRING	550e8400-e29b-41d4-a716-446655440008	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy Image Tool 2	https://dsd-dummy-tool.commanderkowalski.uk/service	IMAGE	550e8400-e29b-41d4-a716-446655440007	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy Image Optimiser	https://dsd-dummy-tool.commanderkowalski.uk/service	IMAGE	550e8400-e29b-41d4-a716-446655440006	2025-01-11T21:14:45.858Z	...

Link Model

Previous

Next

The administrator also has the option of adding a new service tool to the platform using the “Link Model” button and filling out the following form:

Add Model

Name

URL

Input Type

Select input type

Link

Using the Service Tool Assignment screen that is accessible from the navbar to the left, administrator can search for users of the platform and grant them access to available service tools like so:

Service Tool Assignment

customer1

Search

customer1

550e8400-e29b-41d4-a716-446655440002

Role: CUSTOMER

Filter names...

<input type="checkbox"/>	Name	Url	Id	Created At	
<input type="checkbox"/>	Dummy Image Optimiser		550e8400-e29b-41d4-a716-446655440006	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy File Service		550e8400-e29b-41d4-a716-446655440005	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Real AI Service		550e8400-e29b-41d4-a716-446655440004	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy Chatbot		550e8400-e29b-41d4-a716-446655440008	2025-01-11T21:14:45.858Z	...
<input type="checkbox"/>	Dummy Image Tool 2		550e8400-e29b-41d4-a716-446655440007	2025-01-11T21:14:45.858Z	...

Assign Service Tool

Here, for example, administrator sees all service tools customer1 has access to and can revoke that access by pressing the three dots icon on the far right of the screen with the appropriate screen shown:

Remove Model

Are you sure you want to remove this model from the user?

Cancel

Remove

In addition, administrator can grant access to service tools using the “Assign Service Tool” button and filling out the following form:

Assign Service Tool to User

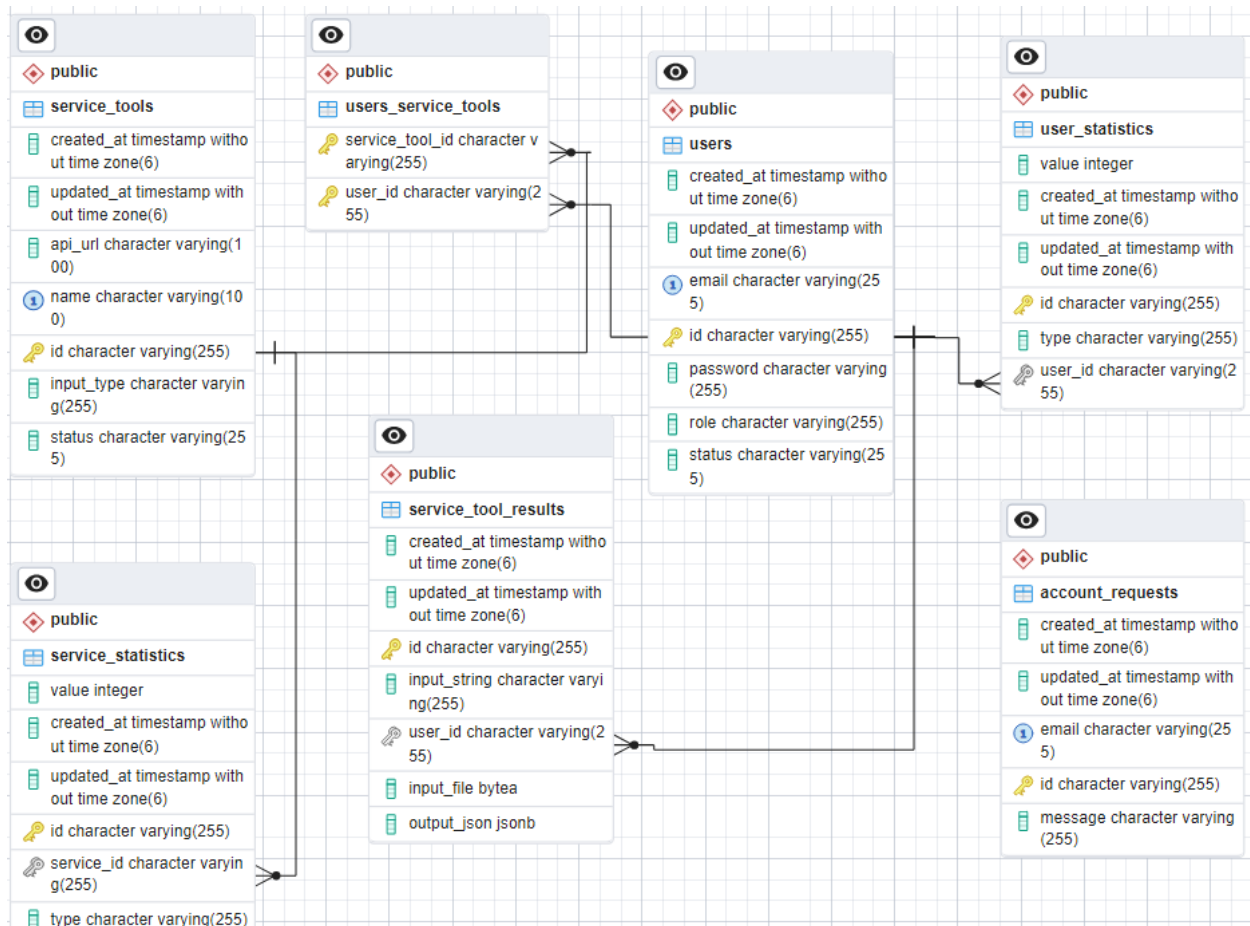
Search Service Tools...

No models found.

Cancel

## 6. Detailed software design

### 6.1 Database schema



## Entities

### Common attributes, excluding user\_service\_tools table:

- Id – automatically generated string, primary key
- created\_at – timestamp of creation of data
- updated\_at – timestamp of last update

### Service tool:

- api\_url – the URL the service tool is available at
- iame – name of service tool
- input\_type – type of input that service tool supports, must either be IMAGE, FILE or STRING
- Status – status of service tool, must either ACTIVE or RETIRED

**Account requests** – represents request sent by potential new users for their account to be created so they gain access to the platform:

- Email – new user's email
- Message – message the user wrote in request

**Service tool result** – represents the result of service tool invocation:

- Input\_string – string that was inputted to the service tool
- Input\_file – file that was inputted to the service tool, represented as byte array
- Output\_json – resulting JSON file of service tool usage, represented as JSON BLOB

**Service statistics** – represent the usage statistics of service tools:

- Type – type of metric, must be INVOCATION\_COUNT
- Value – number of invocations of service tool
- Service\_id – id of service the statistic is relevant to, foreign key for Service tool entity

**User statistics** – tracks login statistics of user

- Type – type of metric, must be LOGIN\_COUNT
- Value – number of logins
- User\_id – id of user, foreign key

#### **Users** – user credentials

- Email – user email
- Password – hashed password
- Role – can be CUSTOMER or ADMIN
- Status – status of users account, ACTIVE or DELETE

#### **Users\_service\_tools** – join table for users and service tools they have access to:

- User\_id – id of user, foreign key for user entity
- Service\_tool\_id – id of service tool user has access to, foreign key

## 6.2. Backend classes and methods

The backend API is comprised of following packages and their respective classes and interfaces:

#### **Services** – classes that execute the business logic of the application:

- AccountRequestService
- EmailService
- OptimizationModelService
- OptimizationResultService
- StatisticsService
- UserService
- Package impl – holds the concrete implementations of the interfaces specified in the package:
  - AccountRequestServiceImpl
  - EmailServiceImpl
  - OptimizationModelServiceImpl
  - OptimizationResultServiceImpl
  - StatisticsServiceImpl
  - UserServiceImpl

**Security** – contains packages related to security setup of the app:

- Package utils – security utility classes:
  - JwtUtil
  - RSAKeyUtil
- Package services:
  - UserDetailsServiceImpl
- Package filters:
  - JwtTokenFilter
  - UserAuthenticationFilter
- Package dtos:
  - AuthenticationResponseDTO
  - UserLoginDTO
- Package config:
  - SecurityConfig
  - UrlConstants

**Repositories** – classes that handle persistence and fetching of data from the database, they are Java Interfaces for which Spring Framework automatically generates implementation:

- AccountRequestRepository
- OptimizationModelRepository
- OptimizationResultRepository
- ServiceStatisticsRepository
- UserRepository
- UserStatisticsRepository

**Mappers** – mapper classes with static methods that map entities to data transfer object classes and vice versa:

- AccountRequestMapper
- MultipartFileResource
- OptimizationResultMapper
- UserMapper

**Exceptions** – exception classes:

- BadRequestException
- ErrorDetails
- ForbiddenException



**Enums** – enumeration classes

- GraphType
- InputType
- OptimizationModelStatus
- ServiceStatisticsType
- UserRole
- UserStatisticsType
- UserStatus

**Entities** – classes that represent entities that are part of the project domain and are mapped to database entities:

- BaseEntity
- AccountRequest
- OptimizationModel
- OptimizationResult
- ServiceStatistics
- User
- UserStatistics

**Dtos** – classes that represent data transfer objects

- AccountRequestDTO
- KeyValueDTO
- OptimizationModelDTO
- OptimizationResultDto
- UserDTO
- UserStatistics

**Controllers** – classes that handle HTTP requests:

- AccountRequestController
- OptimizationModelController
- OptimizationResultController
- StatisticsController
- TestController

**Config** – app config classes:

- Constants
- EmailSenderConfig
- RestControllerExceptionHandler
- RestTemplateConfig

### 6.3.1 Service package detailed look

The services package holds interfaces and classes that implement those interfaces and represent classes that handle the core of the business logic and are used by other classes that may need business logic to be executed and therefore deserve their own section. We will look at each of these interfaces and their methods and shortly describe what they do (when implemented by specific classes of course).

#### **AccountRequestService:**

- AccountRequestDTO **getAccountRequest**(String id) - method takes ID of the AccountRequest entity and fetches it from the database and returns it mapped as AccountRequestDTO class
- List<AccountRequestDTO> **getAccountRequests**() - method returns a list of AccountRequest entities present in the database mapped to AccountRequestDTO class
- AccountRequestDTO **createAccountRequest**(AccountRequestDTO accountRequestDTO) - method takes AccountRequestDTO object which contains fields needed to create AccountRequest entity and creates it and persist it into the database
- UserDTO **approveAccountRequest**(KeyValueDTO keyValueDTO) - method takes a keyValueDTO object that represents a key value pair, key being the primary key of AccountRequest entity and value being the password for User entity. It creates the User entity from email in AccountRequest entity and password in value field of keyValueDTO object and persists it in the database and returns the UserDTO object
- AccountRequestDTO **denyAccountRequest**(KeyValueDTO keyValueDTO) - method takes a keyValueDTO object that represents a key value pair, key being the primary key of AccountRequest entity and value being the password for User entity. Method fetches AccountRequest entity with id in key field of KeyValueDTO object from database and sends email notifications to email in AccountRequest entity email field saying the account request has been denied and deletes the aforementioned AccountRequest entity from database

#### **EmailService:**

- void **sendEmail**(String receiverMail, String subject, String body) - method takes email, subject and body strings and send email to the specified email address with the given content

- void **sendHtmlEmail**(String receiverMail, String subject, String body) - method takes email, subject and body strings and send email to the specified email address with the given content generate from a HTML template

**OptimizationModelService** – Note, naming of this class and its methods and related entities should be refactored to use the service tool naming specified in section 4.2 (so any mention of OptimizationModel should be replaced with ServiceTool):

- OptimizationModel **saveOptimizationModel**(OptimizationModelDTO optimizationModelDTO) - takes OptimizationModelDTO object containing information about service tool and create OptimizationModel entity with them and persists it into the database
- Optional<OptimizationModel> **findOptimizationModelById**(String id) - method takes ID and queries the database for OptimizationModel entity with that ID. Returns object of parametrized class Optional which can be empty if no entity with specified ID was found or contains object of type OptimizationModel if entity with that ID was found
- OptimizationModel **retireOptimizationModel**(String id) - method takes ID of OptimizationModel entity and if it exists, changes the status of the OptimizationModel to RETIRED, making it inaccessible
- List<OptimizationModel> **findAllOptimizationModels**() - method returns list of all service tools
- List<OptimizationModel> **findAllOptimizationModelsByUser**(User user) - takes in object of type User and returns all service tools the provided user has access to
- OptimizationModel **updateOptimizationModel**(String id, OptimizationModelDTO optimizationModelDTO) - method takes ID of service tool and OptimizationModelDTO object, finds the service tool in the database and updates it with fields present in optimizationModelDTO
- OptimizationResult **invokeOptimizationModel**(OptimizationModel service tool, MultipartFile inputFile, String inputString, User invoker) - method takes service tool, user and either a file or string as input, finds the service tool in database and, if the user provided has access to it, calls it and returns OptimizationResult object

#### **OptimizationResultService:**

- List<OptimizationResultDto> **getAllResults**(String userId) - method takes users ID and looks for all results related to user with given ID

- OptimizationResultDto **getResultById**(String resultId) - method takes ID of result and fetches it from database

### StatisticsService:

- List<ServiceStatistics> **getTop3MostInvokedServices**() - method gets the top 3 services ranked by how many invocations it had
- List<UserStatistics> **getTop3UsersByLoginCount**() - method fetches top 3 users with the greatest number of times logged in
- ServiceStatistics **getServiceStatisticsByServiceId**(String serviceId) - method takes ID of ServiceStatistics entity and fetches it from database
- List<UserStatistics> **getUserStatisticsByUserId**(String userId) - method takes users ID and fetches all statistics that are related to him

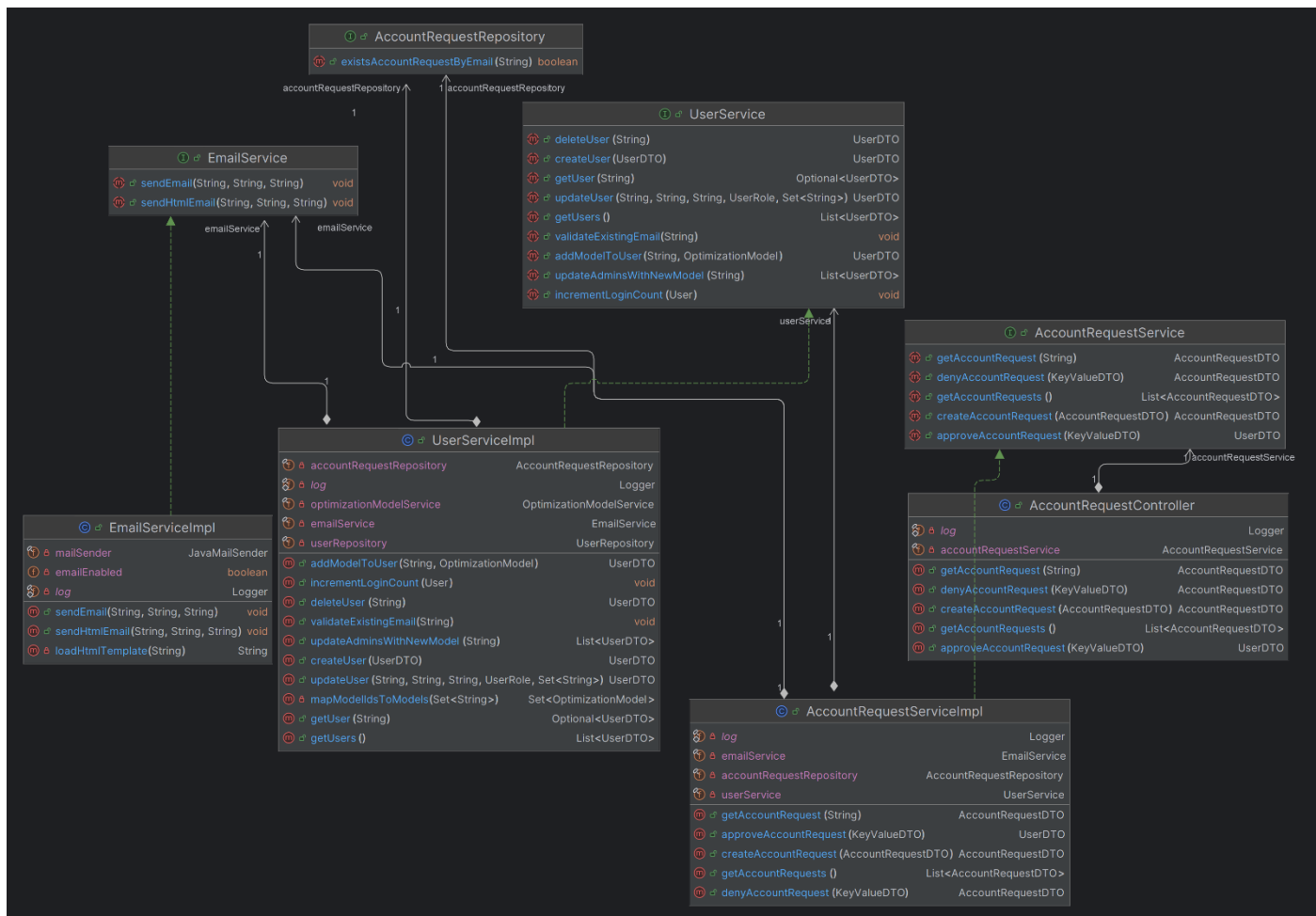
### UserService:

- UserDTO **createUser**(UserDTO userDTO) - method takes data about user in form of UserDTO and creates User entity from it and persists it in the database
- List<UserDTO> **getUsers**() - method fetches all users from database
- Optional<UserDTO> **getUser**(String id) - method takes ID and fetches user from database with that ID if user exists
- UserDTO **updateUser**(String id, String email, String password, UserRole role, Set<String> optimizationModelIds) - method takes user ID, email, password, role and set of service tool IDs and updates the information about user with specified ID with provided parameters, returns UserDTO object with updated user data
- UserDTO **deleteUser**(String id) - method takes user ID and deletes user with specified ID from database
- List<UserDTO> **updateAdminsWithNewModel**(String modelId) - method takes ID of a service tool, fetches OptimizationModel entity with specified ID from the database and associates it with all users having role ADMIN and persists that those associations in the database, returning list of all users that the service tool was associated with

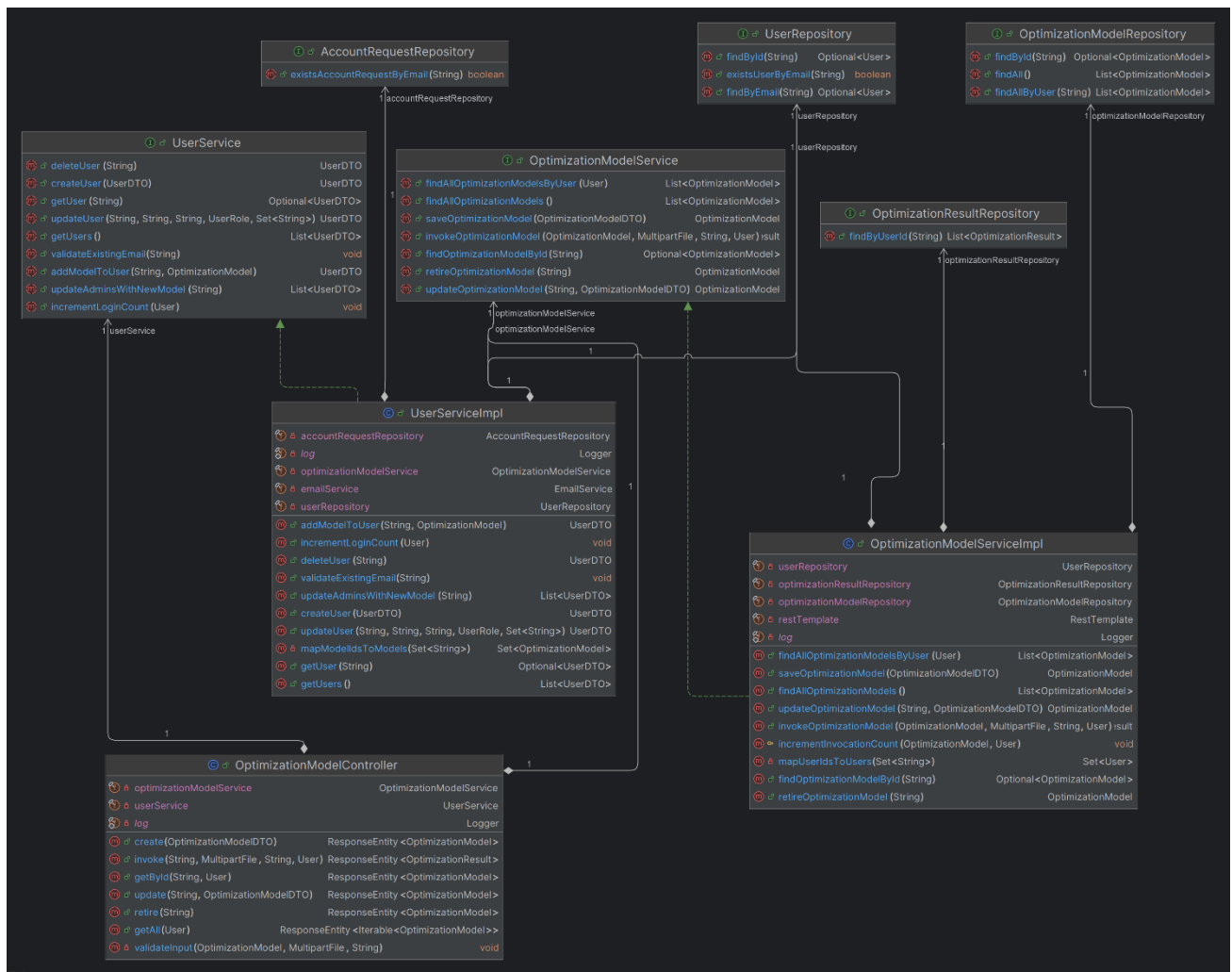
- UserDTO **addModelToUser**(String userId, OptimizationModel service tool) - takes ID of user and service tool to be associated with him, fetches the user from the database and adds service tool to list of service tools available to the user and persists that association in the database
- void **validateExistingEmail**(String email) - helper method to validate if given email already exists in the database
- void **incrementLoginCount**(User user) - method takes user entity, finds statistics entity associated with him, increments the login counter, and persist it to the database

## 6.4. Class diagrams

Because the backend has a lot of classes the class diagram would not be readable so we have decided to break the diagram into multiple diagrams where classes are grouped around a particular service or use case. Some utility classes that are available in Spring Framework were omitted such a logger classes and automatically generated classes that implement repository interfaces.

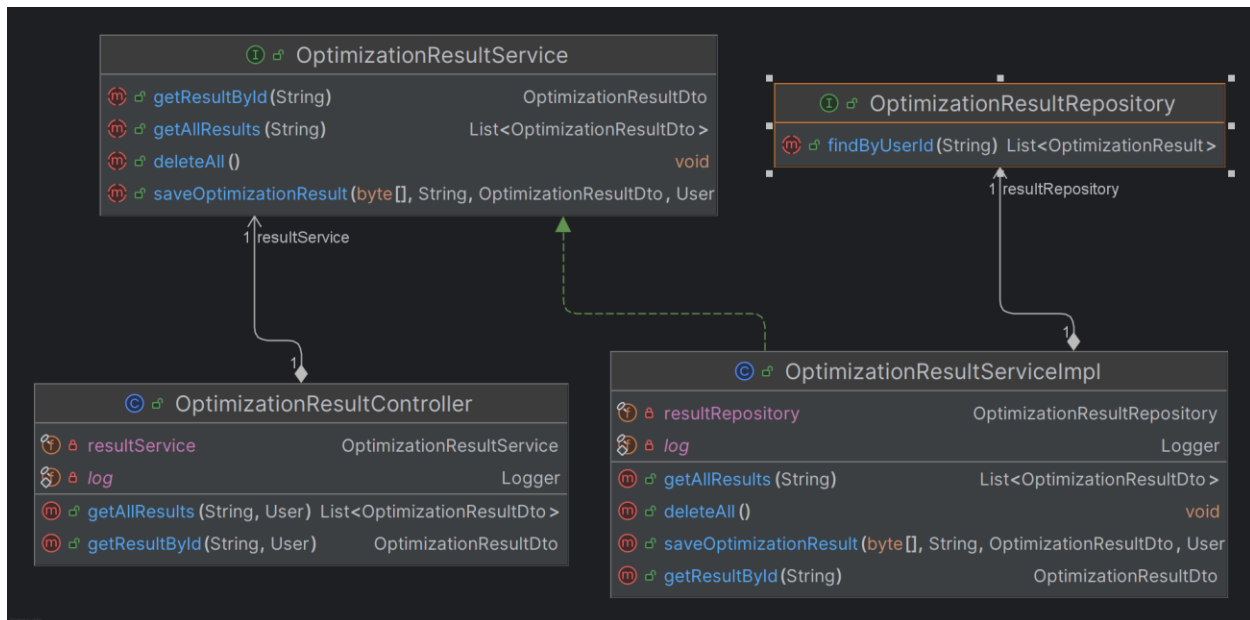


Classes that execute account request logic

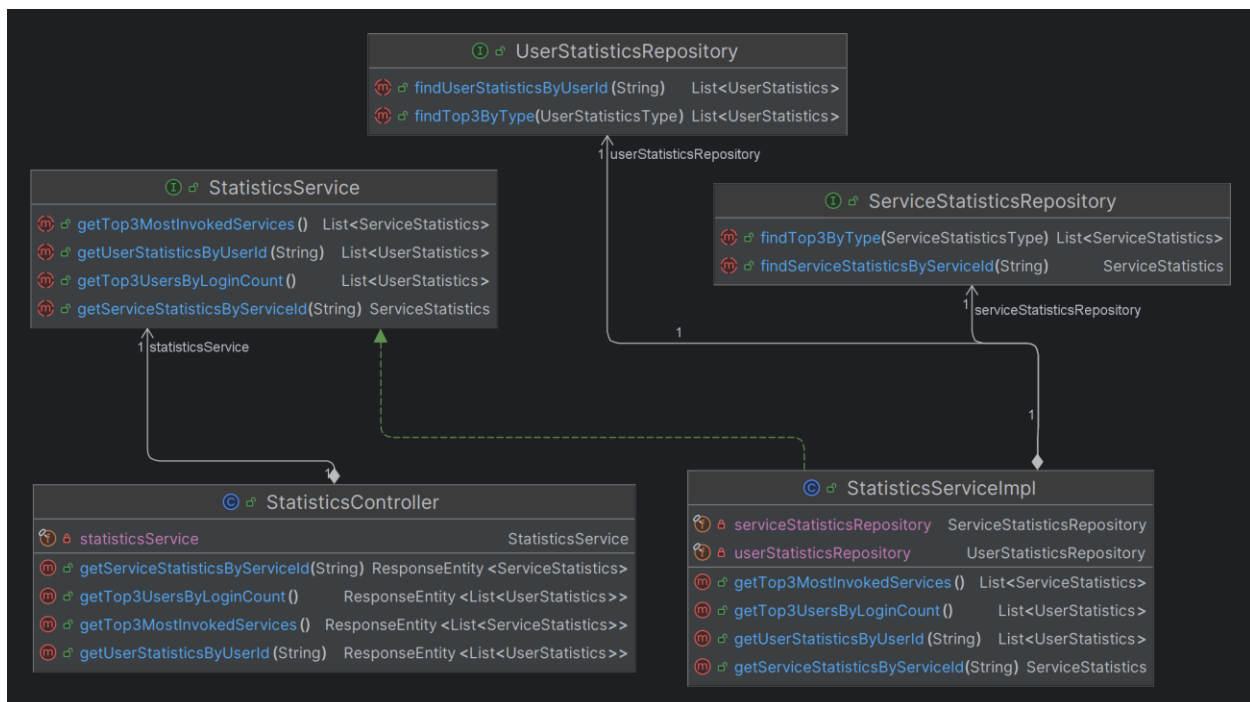


Service tool logic classes

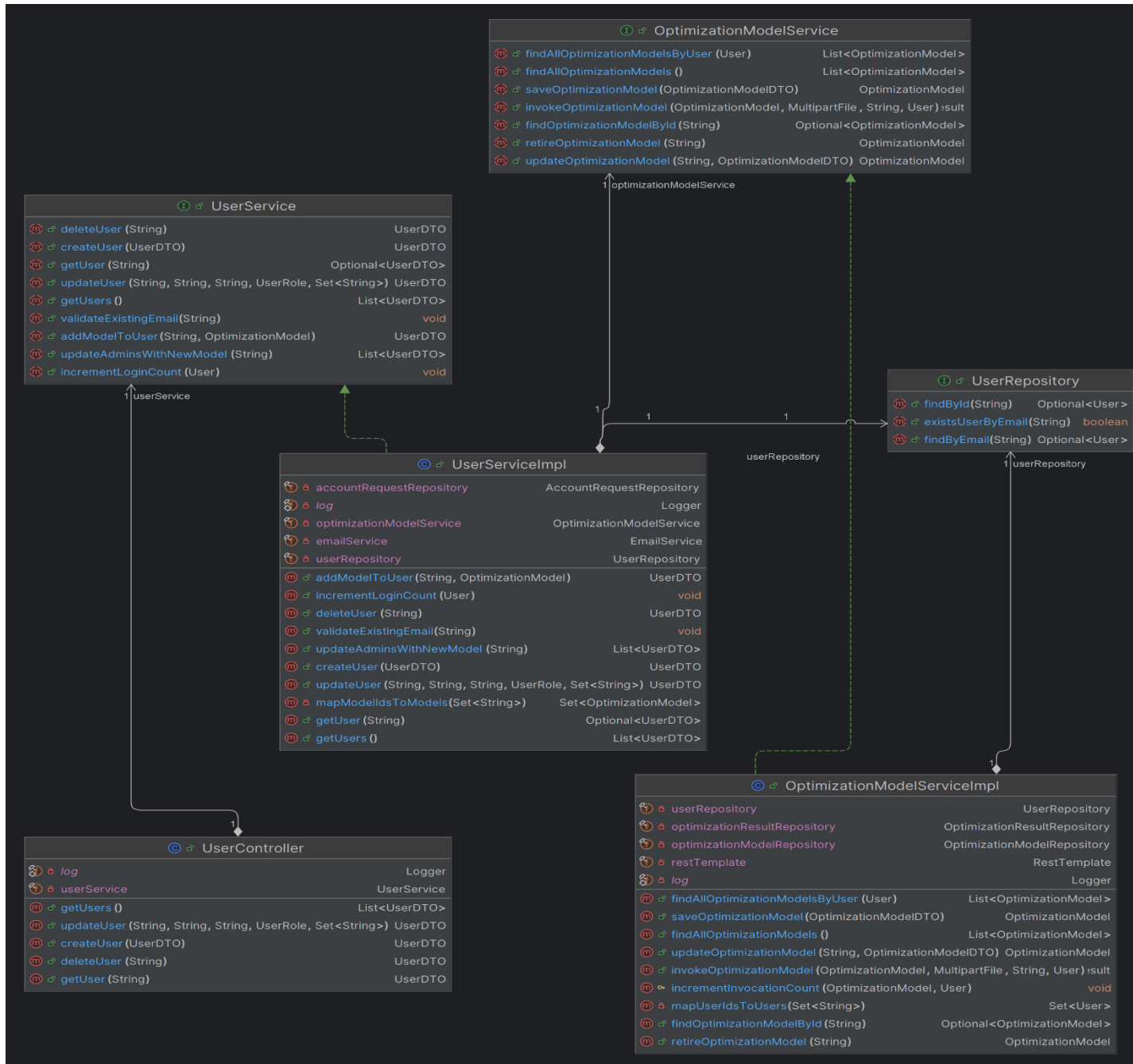




Optimization result logic classes



Statistics logic classes



User logic classes