



# Project: Critical innovation laboratories

Coimbra University – Program “Curso de Formação Acertar o Rumor — Programação em Java”

July 2024

Pedro Manuel Duarte Monteiro

Daniel Alves da Silva

## Introduction

In the fast-paced environment of modern business, efficient task management is crucial. This application has the objective of offering an intuitive, powerful, and centralized platform where employees can organize, prioritize, and track tasks effectively.

This application aims to solve disorganized task management, that when left unsupervised, can lead to missing deadlines, reduced productivity, which in turn results in bad company ratings. Teams must have constant communication to mitigate these issues, often resulting in time and productivity loss. Through this tool, task assignment, process tracking, and simplified collaboration, team members can be more aligned and objective focused.

The primary objectives of app are:

- Creation and management of projects in a straightforward way, showing project managers and members their respective project participation through an intuitive UI.
- Improve task organization by enabling users to create, assign and categorize tasks efficiently.
- Provide tools for task prioritization, set deadlines and monitoring progress to improve productivity.
- Offer features that help team communication with relevant information from the project page, a notification system that alerts users about project and tasks changes, as well as enabling private messaging between users and between project members inside the project.

The app offers a user-friendly interface with a clean and intuitive design, tools for creating tasks and changing them, assigning them to team managers, and setting priorities, deadlines, and real-time updates to facilitate communication.

The initial version may have limitations as the app can slow down with the increasing information and increasing database load.

# Requirements

## User Requirements

The user expectation using the app should be smooth, intuitive, with an interface that is easy to learn and use. The user, as an employee can focus on the tasks and projects that are their responsibility, understand which one is the priority and their deadlines. The employee also can change the status of the task to help the team know its evolution. The user, as a manager can divide tasks between team members and can divide depending on their skills, add deadlines to the tasks, and choose the priority of the task. Both can use in-project chat to communicate and be updated.

## **Use case**

As a manager, I want to be able to create a project I find relevant and select team members according to their skills and interests. I want to assign tasks to my team members, by their skills and interests, so that everyone knows their responsibilities. I can choose the deadline of the project, manage the project, and talk with my team members through a chat. As an employee, I want to know the real-time updates of the tasks, I want to know exactly what my tasks are, their respective deadlines and responsibilities, and change the status to inform my team, in order to avoid duplicate work. Everyone can generate productive reports to evaluate team performance and ways to improve.

## Functional Requirements

- **User registration and Authentication:**
  - Users should be able to create accounts with unique e-mails. When the account is created the user will receive a confirmation e-mail. Through the link in the email, the user account is confirmed, and the user can login in the application.
  - Can log in and log out their accounts, edit their respective profiles, add interests and skills relevant for their profile.
  - Users are also able to create projects they find relevant, select keywords for the respective project and submit them for further evaluation from page administrators, which in turn decide if the project moves on to the progress state or is cancelled.
  - In case the user decides, he can leave the project without any commitment issues, as the application manages task transference in case a user leaves.
- **Project creation:**
  - Projects can be created by any registered user.
  - Projects require a name, description, a set of keywords, a start date and end date and mandatorily a task named "Presentation task", with date due to the day before the project ending.
  - Projects also have an "estimated termination date", according to the cumulative time it takes for all the tasks in the project to be completed.
- **Task creation and management:**

- Users within a project can create, edit and delete tasks.
- Tasks can be assigned to individual users as well as allowing external individuals to also participate.
- Some tasks have dependencies, in which they can only progress if the previous task is completed. Tasks have owners, start dates and end dates.
- **Collaboration and communication features:**
  - Users can communicate in between other users in the application through a message system, as well as to other project members when within a project.
- **Progress tracking:**
  - Users can change the task status, and a notification system informs the whole team about it.
- **Customizable workflow:**
  - Users can create and manage tasks (manage tasks and delegate tasks depend on role of the user).
- **Report and analyze:**
  - The app can generate reports on project completion rate according to location and project progress.
- **Search and filter:**
  - Users are able to search for a specific project or user. Projects can be filtered by name or tags, as well as by project state. Materials, skills and interests can be filtered by name. Materials and skills can be filtered by type.
- **Material page:**
  - Users are capable of observing the materials available for projects, as well as creation of new materials according to the project requirements.

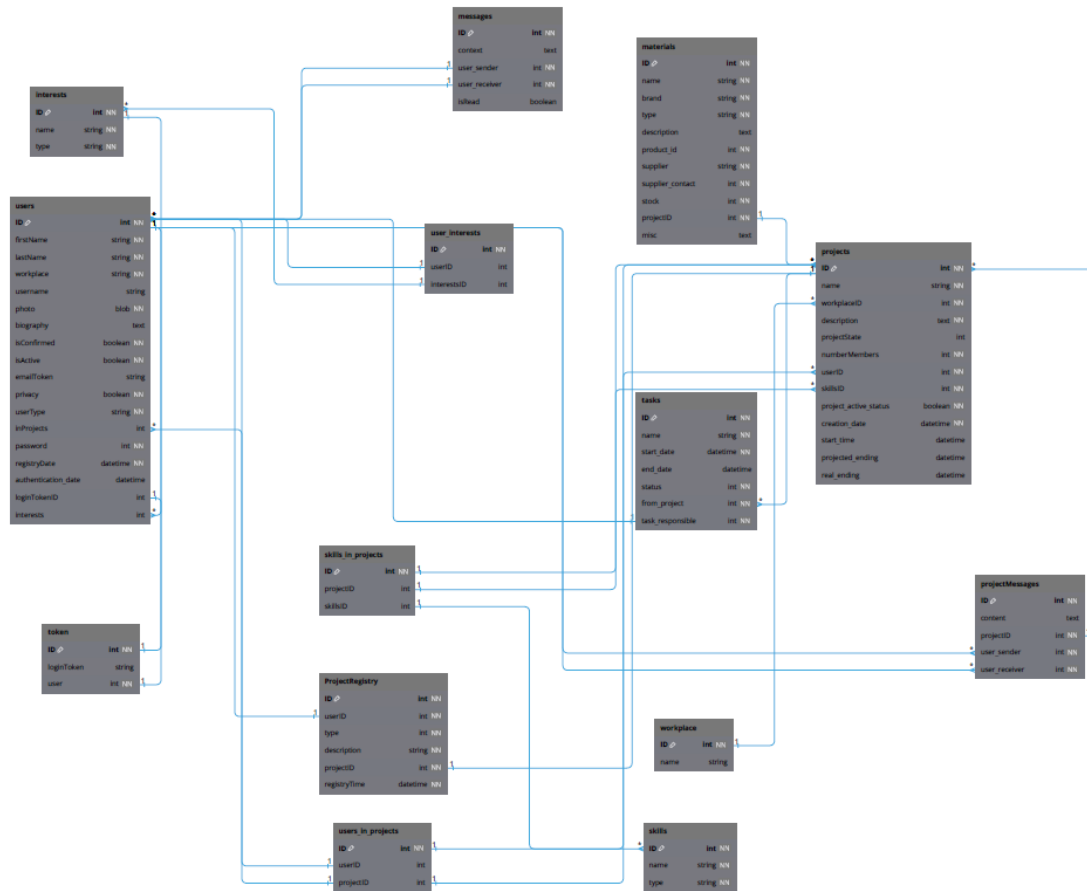
### Non-Functional Requirements

The app capacities and constrains are determined by his executability.

- The performance of the app, the application should take less than a second to load and update information.
- The security it is preserved by existing encrypted passwords, secure and logical API requests and through parameter validation.
- The stability of the app to accommodate the increase of users and tasks/projects and should be easy to add new features that could be helpful for users in their daily work.
- The usability of the app with the friendly user interface, that make it intuitive and easy to use.
- The reliability of the regular backups in the database to prevent data loss.
- The maintainability through modular architecture to allow easy updates and bug fixes.
- The compatibility of the app with major web browsers.

# ER Model

## Entity-Relationship Diagram



## Description of Entities

### interests:

- Represents: The interest of the users and projects.
- Key attributes:
  - **"ID"**: interest id.
  - **"name"**: name of the interest.
  - **"type"**: type/category of the interest.

### users:

- Represents: An individual using the application.
- Key attributes:
  - **"ID"**: users id.
  - **"firstName"**: First name of the user.

- **"lastName"**: Last name of the user.
- **"workplace"**: The workplace or company the user is associated with.
- **"username"**: The user's chosen display name
- **"photo"**: URL of the user's profile photo.
- **"biography"**: A short biography of the user.
- **"isConfirmed"**: Boolean indicating if the user's email is confirmed.
- **"isActive"**: Boolean indicating if the user's account is active.
- **"emailToken"**: Token used for email verification.
- **"privacy"**: User's privacy settings.
- **"userType"**: Type of user (e.g., admin, manager, employee).
- **"inProjects"**: List of projects the user is involved in.
- **"password"**: The user's encrypted password.
- **"registryDate"**: Date the user registered.
- **"authentication\_date"**: Date of the last authentication.
- **"loginTokenID"**: Token used for two-factor authentication.
- **"interests"**: List of user's interests (link to Interests entity).

#### **token:**

- Represents: Authentication tokens used for secure login.
- Key attributes:
  - **"ID"**: token id.
  - **"loginToken"**: The token used for login.
  - **"user"**: The user associated with the token.

#### **messages:**

- Represents: The messages between users.
- Key attributes:
  - **"ID"**: message id.
  - **"context"**: The content of the message.
  - **"user\_sender"**: The user who sent the message.
  - **"user\_receiver"**: The user who received the message.
  - **"isRead"**: Boolean indicating if the message was read.

#### **user\_interests:**

- Represents: The connection between the interests and the users.
- Key attributes:
  - **"ID"**: connection between interests and users id.
  - **"userID"**: The ID of the user.
  - **"interestsID"**: The ID of the interest.

#### **skills\_in\_project:**

- Represents: The connection between the skills and the task.
- Key attributes:

- **"ID"**: skills in project id.
- **"projectID"**: The ID of the project.
- **"skillsID"**: The ID of the skill.

#### **ProjectRegistry:**

- Represents: The registration of users in projects.
- Key attributes:
  - **"ID"**: project registry id.
  - **"userID"**: The ID of the user.
  - **"type"**: The type of registration (e.g., role in project).
  - **"description"**: A description of the user's role or involvement.
  - **"projectID"**: The ID of the project.
  - **"registryTime"**: The time of registration.

#### **user\_in\_projects:**

- Represents: The connection between the users and the projects they are assigned.
- Key attributes:
  - **"ID"**: user in projects id.
  - **"userID"**: The ID of the user.
  - **"projectID"**: The ID of the project.

#### **materials:**

- Represents: The materials used in projects.
- Key attributes:
  - **"ID"**: materials id.
  - **"name"**: The name of the material.
  - **"brand"**: The brand of the material.
  - **"type"**: The type or category of the material.
  - **"description"**: A description of the material.
  - **"product\_id"**: The ID of the product.
  - **"supplier"**: The supplier of the material.
  - **"supplier\_contact"**: Contact information for the supplier.
  - **"stock"**: The current stock level of the material.
  - **"project\_ID"**: The ID of the project using the material.
  - **"misc"**: Any additional information about the material.

#### **tasks:**

- Represents: A task or project that needs to be completed.
- Key attributes:
  - **"ID"**: tasks id.
  - **"name"**: The name of the task.
  - **"start\_date"**: The start date of the task.
  - **"end\_date"**: The end date of the task.

- **"status"**: The current status of the task.
- **"from\_project"**: The project ID the task is associated with.
- **"task\_responsible"**: The user ID of the person responsible for the task.

#### **workplace:**

- Represents: The workplaces or companies" users are associated with.
- Key attributes:
  - **"ID"**: workplace id.
  - **"name"**: The name of the workplace.

#### **skills:**

- Represents: The skills that are chosen by the user.
- Key attributes:
  - **"ID"**: skills id
  - **"name"**: The name of the skill.
  - **"type"**: The type or category of the skill.

#### **projects:**

- Represents: The projects managed in the application.
- Key attributes:
  - **"ID"**: projects id
  - **"name"**: The name of the project.
  - **"workplaceID"**: The ID of the workplace associated with the project.
  - **"description"**: A description of the project.
  - **"projectState"**: The current state of the project.
  - **"numberMembers"**: The number of members in the project.
  - **"userID"**: The ID of the user who created the project.
  - **"skillsID"**: The ID of the skills required for the project.
  - **"project\_active\_status"**: Boolean indicating if the project is active.
  - **"creation\_date"**: The date the project was created.
  - **"start\_time"**: The start time of the project.
  - **"projected\_ending"**: The projected end date of the project.
  - **"real\_ending"**: The actual end date of the project.

#### **projectsMessages:**

- Represents: The messages associated with projects.
- Key attributes:
  - **"ID"**: messages in projects id
  - **"content"**: The content of the message.
  - **"projectID"**: The ID of the project.
  - **"user\_sender"**: The user who sent the message.
  - **"user\_receiver"**: The user who received the message.



## Relationships between entities

### **User – Interests:**

- Type: Many-to-Many
- Description: A user can have multiple interests and an interest can have multiple users. This is represented by the “user\_interests” join table.
- Join table: “user\_interests” where the “user\_ID” and the “interest\_ID” are both foreign keys. “user\_id” linked in “users” and “interests\_ID” linked in “interests”.

### **User – Project:**

- Type: Many-to-Many
- Description: A user can be involved in multiple projects, and a project can have multiple users. This is represented by the “user\_in\_projects” join table.
- Join table: “user\_in\_projects” with the “user\_ID” and the “projectID” are both foreign keys. “user\_id” linked in “users” and “projectID” linked in “Projects”.

### **User – Tasks:**

- Type: One-to-Many
- Description: A user can be responsible for many tasks, but each task is assigned to only one user.
- Foreign key: “task\_responsible” in “Tasks” mention in “user\_ID” in “Users”.

### **User – Messages:**

- Type: One-to-Many
- Description: A user can receive and send multiple messages.
- Foreign key: “user\_send” in “Messages” mention in “user\_ID” in “Users” and “user\_receiver” in “Messages” mention in “user\_ID” in “Users”.

### **Projects – Tasks:**

- Type: One-to-Many
- Description: A project can have multiple tasks, but a task can be only linked to one project.
- Foreign key: “from\_project” in “Tasks” mention in “project\_id” in “Projects”.

### **Projects – Material:**

- Type: One-to-Many
- Description: A project can have multiple materials, but each material is linked to one project.
- Foreign key: “project\_ID” in “Materials” mention in “projects\_id” in “Projects”

### **Projects – ProjectRegistry:**

- Type: One-to-Many
- Description: A project can have multiple registry entries, but each registry belongs to a project.
- Foreign key: “projectID” in “ProjectRegistry” mention in “project\_id” in “Projects”.

**Projects – ProjectMessages:**

- Type: One-to-Many
- Description: A project can have multiple messages, but each message is linked with only one project.
- Foreign Key: “projectID” in “ProjectMessages” mention in “project\_id” in “Projects”.

**Workplace – Users:**

- Type: One-to-Many
- Description: A workplace can have multiple users, but each user can only have one workplace.
- Foreign Key: “workplace” in “Users” mention in “workplace\_id” in “Workplaces”.

**Workplace – Projects:**

- Type: One-to-Many
- Description: A workplace can have multiple projects, but each project can only have one workplace.
- Foreign Key: “workplaceID” in “Projects” mention in “workplace\_id” in “Workplaces”.

**User – Token:**

- Type: One-to-Many
- Description: A user can have multiple tokens, but each token belongs to only one user.
- Foreign Key: “user” in “Token” mention in “user\_id” in “Users”.

# Architecture

## System Architecture

A system architecture sets the foundation of the application, defining its structure by connecting the frontend services - backend services – database services. The efficiency of the architecture has the objective of making the code more modular, as well as stable and secure.

### Frontend

The app frontend was build using the React frameworks, by using JSX and CSS as well some libraries and tools to make the application more responsive and with a user-friendly interface.

- Global state management: Global state management was achieved through the usage of Zustand, which allows state variables such as user tokens, ID's and other valuable parameters to be stored in the user session, allowing the API request parameters to be easily accessible, reducing overall code construction and streamlining processes.
- Component-Specific State Management through the capability and modularity of the React framework that allows specific component creation and edition.
- Routing: Routing between pages through react navigation.
- UI components: Through open-source libraries available for UI design (such as MUI and Bootstrap), giving the application a more attractive look through curated designs.
- Language: using the react-i18next we can manage language resources.
- WebSocket: Through construction and implementation of web-sockets, users can have a streamlined communication and notification system, allowing real time updates on important information such as project and task updates.

### Backend

The app backend was built using Jakarta EE (Java language) and Wildfly/Maven for development in a local server. This structure allows the backend to be more responsive e communicative with the server through the deployments. This structure holds all the relevant logic and data for communication with the frontend.

- Application Programming Interface (API) services: using the java (Jakarta EE), the API handle the HTTP requests from the frontend, process de information and return the adequate response. This interacts with the Data transfer objects (DTO's), either sending them to the frontend or processing them through more layers of the backend logic, such as being converted to Data Access Objects (DAO's) by the service layers, allowing for persistence, merge or removal of entities from the database.
- Database: Using MySQL as the database framework, the app can store information regarding user, projects, and tasks, as well as other required data for the correct functioning of the application.
- WebSocket server: using the java (Jakarta EE), right now this feature is not available for the users, but the objective is to create connecting with the frontend to give updates, notification, permit communication.

- Authentication service: using Java (Jakarta EE), it validates tokens to secure access to the app and make sure that each information it's only available for each user.

### **Middleware**

The middleware components are crucial for authentication and state persistence.

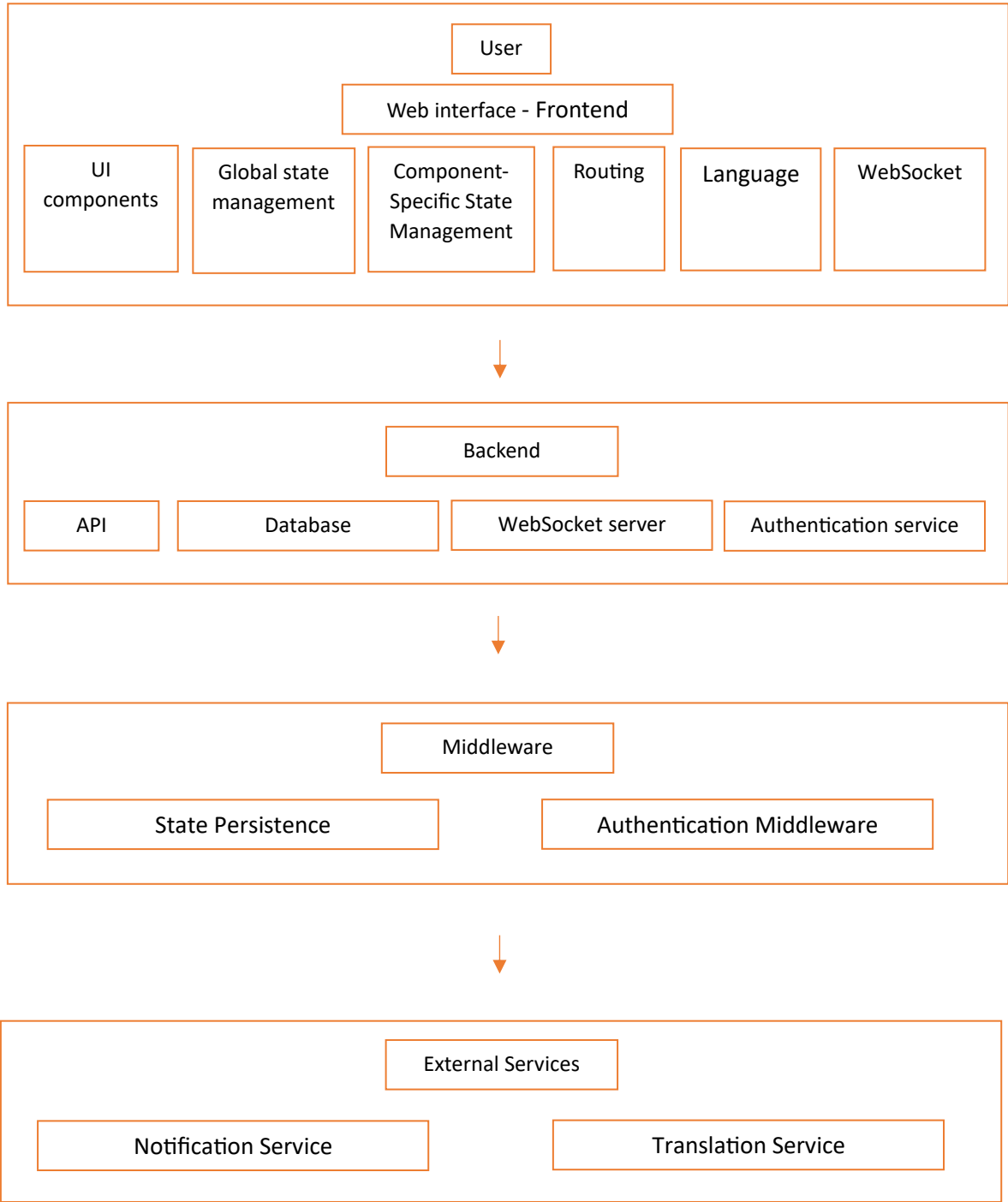
- State Persistence: using Zustand to ensures that variables that are required in several pages are stored, in order to avoid unnecessary requests to return crucial information. Examples are user ID and token, often required in API requests.
- Authentication Middleware: intercepts API requests and checks the email for validation through the user token.

### **External Services**

The external services allow to provide other functionalities as notifications and translation. This specific functionality is not working, the code of this part is incomplete.

- Notification Service: send notifications to users about updates, deadlines, and other occurrences.
- Translation Service: provide translation in different languages, allowing to switch language according to the user preference.

Component Diagram



## Data Flow

### API Requests:

- Flow: Frontend → Backend → Database → Backend → Frontend
- The frontend makes API requests to the backend, this one process the request interacting with the database and send back the response to the frontend.

### Authentication:

- Flow: Frontend → Backend → Tokens → Frontend
- When some interaction like log in/log out in the frontend, there are some validations in the backend linked to tokens, and these tokens are used for authentication and give information to the frontend.

### User Interaction:

- Flow: User → Frontend (React components)
- Users interact with the app through the web page, in this case created with react, this performs various actions such as logging in, registering, manage tasks, (...)

### WebSocket Communication:

- Flow: Frontend → WebSocket Server → Frontend
- For WebSocket communication exist instant communication for the real-time updates, as notifications and messages. This communication this does not exist in the app presented.

### Languages:

- Flow: Frontend → Translation Service → Frontend → User interface
- Translations can be fetched from an external service, which is connected to the frontend in both sides, and then display to the user in the language determined. This communication this does not exist in the app presented.

### Notification Delivery:

- Flow: Backend → Frontend → User interface (UI)
- For real time updates, the backend communicates to the frontend through API responses or through WebSocket connection. Consequently the frontend updates the UI to display these notifications. This communication this does not exist in the app presented.

### State Management:

- Flow: Frontend → Backend → Interact in frontend → State Management → Session storage
- Users logs in and requests the API for the user information. Backend response sends user information. Information from the user (such as token and ID) is stored in the Zustand store persistence (session storage) to later be used in API requests.

This detailed description of the architecture system ensures that all components are well-documented and the interaction are clear, that make easy to develop more features and maintain the existing ones.

## Technologies Used

### Frontend Technologies

- **React:**
  - React is a JavaScript library for building user interfaces. It allows developers to create large web app with efficiency.
  - Inside React we used JSX and CSS, as well as available UI libraries for the construction and deployment of the user interface.

### Backend Technologies

- **Jakarta EE (Java):**
  - Jakarta EE define the application programming interface and their interactions, some of these features can be distributed computing and web services. The Jakarta EE as other services as security and scalability.
  - It was used to build the backend services and APIs.
- **Maven:**
  - Maven is an automation tool that help us to manage the project” build, dependencies, and documentation.
- **WildFly:**
  - WildFly was used to deploy and run the backend services.

### Database

- **MySQL:**
  - MySQL is an open-source relational database management system, and we used it to store and manage the app, projects, tasks, and users, as well as required data for the correct functioning of these components, such as interests and skills for user and project construction, materials required for the project execution and all the relationships that exist within these components.

### Tools and Libraries

- In the frontend we used the following:
  - **Material UI:** for implementing responsive and accessible UI components.
  - **Bootstrap:** for creating a consistent design and ensure the app is responsive.
  - **Npm:** for managing dependencies and packages required for the frontend development.
- In the backend we used the following:
  - **Maven:** for managing the project’s build, dependencies, and documentation.

The previously cited tools were used to develop the application to provide a robust system and scalable solution that can support more tasks and more users. The frontend used the three tools described for creating a responsive and friendly interface, the other three tools for the backend helped to manage the build and dependencies, and the MySQL served as the database technology, ensuring management and data storage to be efficient.

# Presentation of the Final Product

Home page with projects for all users (logged in or not).

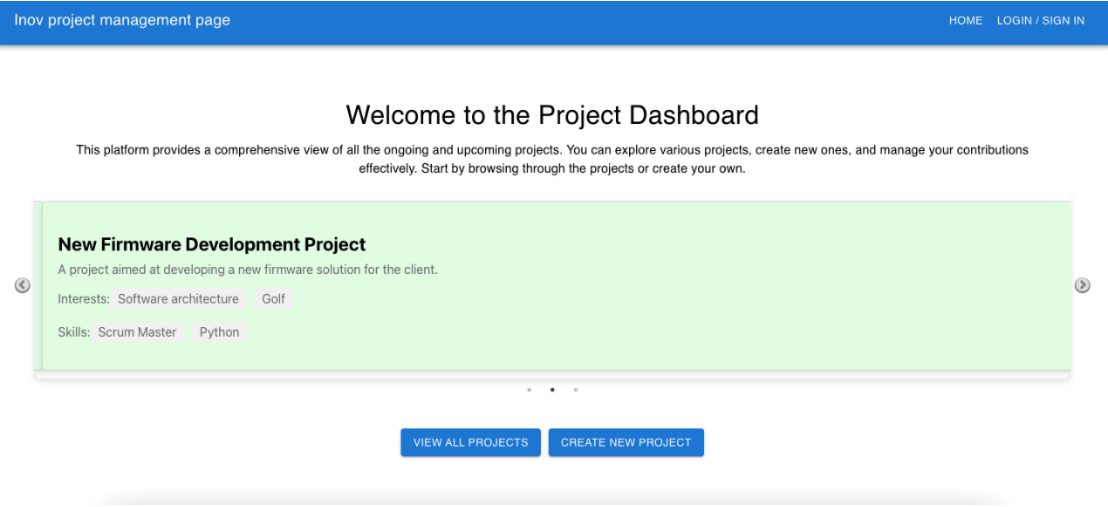


Figure 1 – Project Dashboard.

Flow to login/sign in.

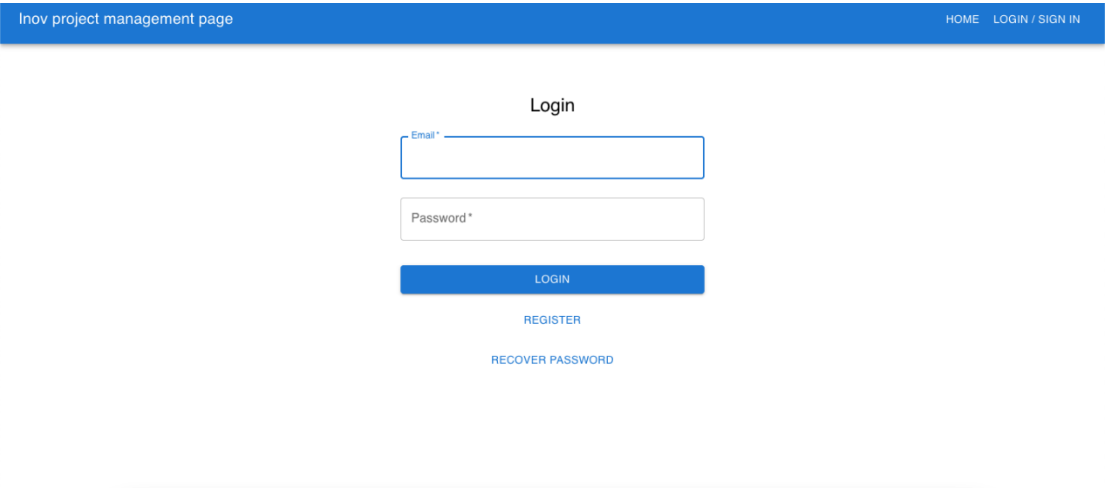


Figure 2 – Login/sign in page.



User can register a new account or recover password.

inov project management page

HOME   LOGIN / SIGN IN

Registry

First Name \*

Last Name \*

Nickname \*

Email Address \*

Location

Password \*

Confirm Password \*

REGISTER

RETURN TO LOGIN PAGE

Figure 3 – Register page.

inov project management page

HOME   LOGIN / SIGN IN

Recover Password


Enter your email \*

SUBMIT

Figure 4 – Recover password page.

Page of materials:

inov project management page

HOME   MATERIALS   ADMIN PAGE   alice   LOGOUT

Search

Filter: All

+ CREATE NEW MATERIAL

ID	Name	Brand	Type	Identifier	Supplier	Supplier Contact	Amount	Comments	
7	Breadboard	Elegoo	RESOURCE	1007	Elegoo	1234567896	5	A solderless prototyping board.	Essential for prototyping circuits.
4	Temperature Sensor	DHT11	COMPONENT	1004	DHT Electronics	1234567893	5	A basic temperature and humidity sensor.	Commonly used in weather stations.
6	LiPo Battery	Turnigy	RESOURCE	1006	Turnigy	1234567895	5	A rechargeable lithium polymer battery.	Used for powering portable devices.
9	Multimeter	Fuke	RESOURCE	1009	Fuke Corporation	1234567898	5	A digital multimeter for measuring voltage, current, and resistance.	Essential tool for troubleshooting circuits.
5	Step Motor	NEMA	COMPONENT	1005	NEMA Motors	1234567894	5	A high precision step motor.	Used in precise motion control applications.
2	Raspberry Pi 4	Raspberry Pi Foundation	COMPONENT	1002	Raspberry Pi Org	1234567891	5	A small single-board computer.	Ideal for small projects.
10	Soldering Kit	Weller	RESOURCE	1010	Weller Tools	1234567899	5	A complete soldering toolkit.	Includes a soldering iron, solder, and accessories.
1	Arduino Uno	Arduino	COMPONENT	1001	Arduino Inc.	1234567890	5	A microcontroller board for beginners.	Used for prototyping.
3	OLED Display	Adafruit	COMPONENT	1003	Adafruit Industries	1234567892	5	A 128x64 pixel OLED display.	Used for displaying text and graphics.
8	Jumper Wires	Dupont	RESOURCE	1008	Dupont	1234567897	5	Male to male jumper wires.	Used for connecting components on a breadboard.

Figure 5 – Materials page.

