BRAUDE
College of Engineering, Karmiel

Factory Management Web App

Advanced Web Technologies - 61776
Group #29 - Project #A22

Presenting:

Tomer Meidan - 204750459

Roman Milman - 320942808

Or Biton - 308090539

Table of Contents

1. Introduction

The Factory Management Web Application represents a comprehensive solution designed to streamline operations and enhance the efficiency of managing a manufacturing facility. This project is aimed at developing a web-based platform that centralizes the control and oversight of key factory operations, including employee management, shift scheduling, department organization. By leveraging modern web technologies and best practices, the application offers a secure, scalable, and user-friendly interface for factory administrators, employees, and clients.

2. Assembly and Division of the Project

System Engineer - Tomer Meidan

Team Member	Assigned Tasks	Status
Roman Milman	Define Authentication Requirements	Completed
	Implement Authentication with API Endpoint	Completed
	Develop CRUD Functionality for Employees	Completed
	Build Router for Employee Management	Completed
	Integrate Endpoint for BLL Management of Users	Completed
	Implement Endpoint for DAL Management of Shifts	Completed
Or Biton	Outline Security JWT Authentication	Completed
	Share Action in Functionality (Back End)	Completed
	Create Model for Shift Leads (Back End)	Completed

	Implement Feature for Creating Shift (Front End)	Completed
	Develop Feature for Deleting Employee (Front End)	Completed
Tomer Meidan	Assign Subtasks for Management	Completed
	Integrate Front End with Back End for Employee Management	Completed
	Share Action in Registration Operations	Completed
	Build Server Structure	Completed
	Build Router for Shift Management	Completed
	Build Router for Department Management	Completed
	Addition of Tailwind to the Project	Completed
	Frontend Implementation on Vercel	Completed
	Backend Implementation on Render	Completed

All team members contributed to developing and integrating CRUD operations for managing departments and shifts across both the front end and back end of the application. Each team member played a critical role in the project's development, from planning and defining requirements to implementing detailed technical functionalities and ensuring the project's successful completion.

3. Requirements Specifications

3.1. Functional Requirements

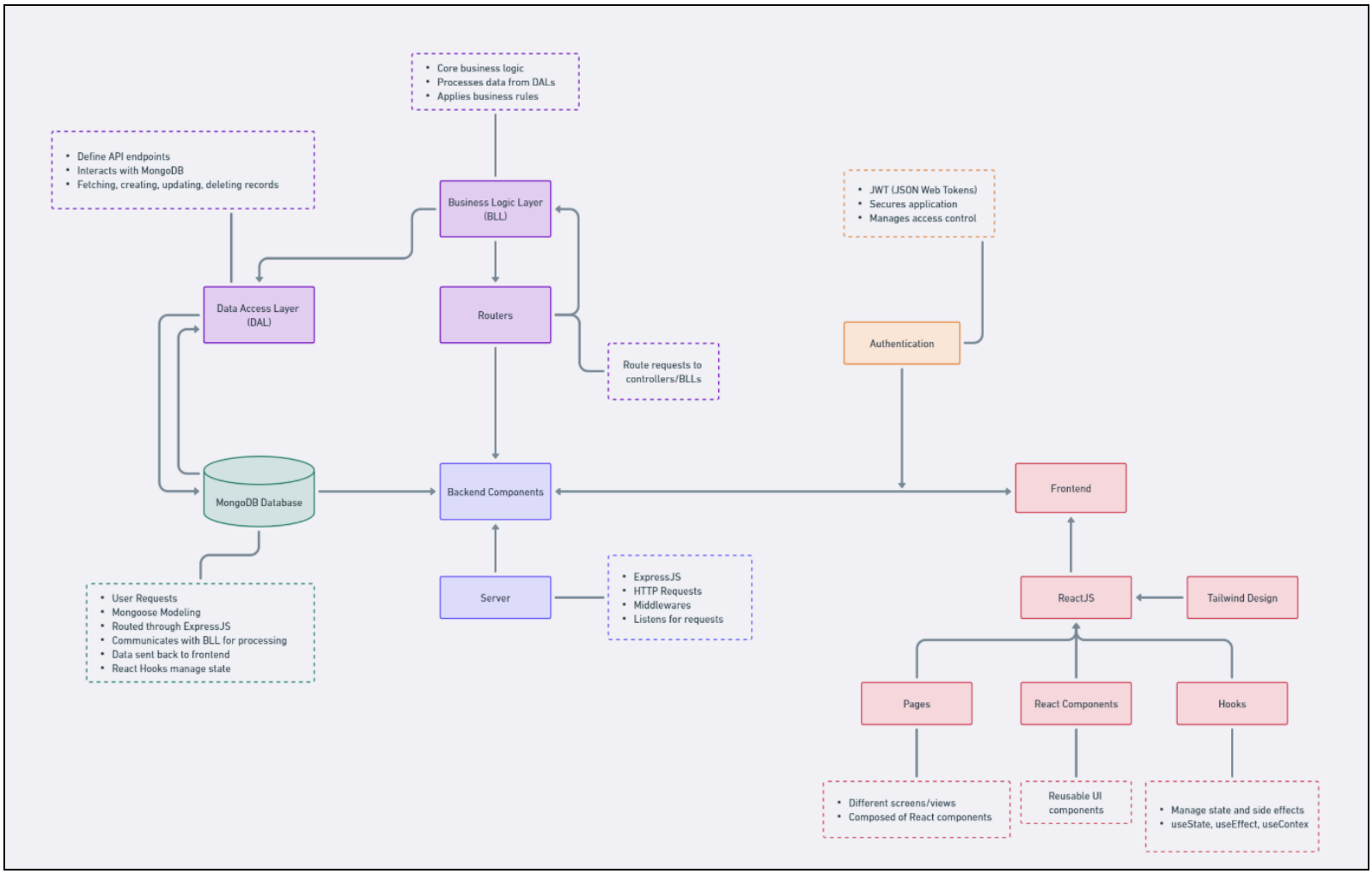
- User Authentication - Users must log in with a valid username and password.
- Authentication uses Session for secure access.
- Employee management - the system allows adding, editing and deleting employee details.
- Employees can be assigned to specific departments.
- Scheduling shifts - the application supports the creation and modification of work shifts.
- Employees can only be assigned to existing shifts.
- Department Management - Administrators can create, edit and delete department records.
- Registration system - the system records the user's actions for tracking and accountability.
- Logs are kept for auditing and debugging purposes.
- Users can view and manage employees and all their shifts, departments list.
- Search and filter - users have the ability to search for employees, departments and shifts.
- The system supports filtering options for quick data retrieval.

3.2. Non-Functional Requirements

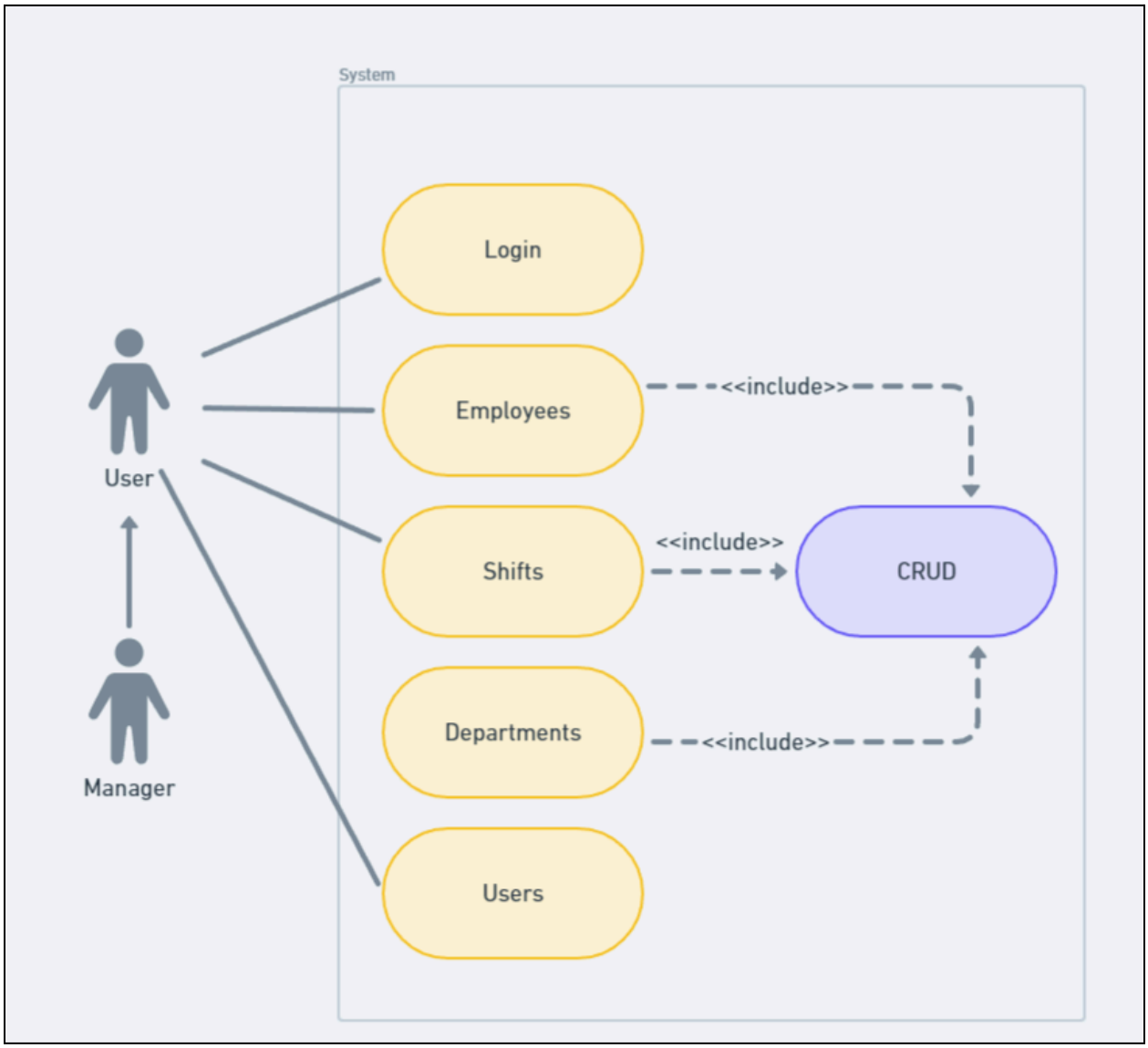
- Usability - the system provides a friendly interface for efficient management of employees and departments.
- Security - User data and system information are stored securely and protected against unauthorized access.
- Performance - the application responds immediately to user actions, ensuring a smooth and efficient user experience.
- Maintenance - the system is designed for maintenance and easy updates. Scalability - The application scales horizontally to handle increased user loads and data volume.
- Reliability - the system guarantees the confidentiality and integrity of sensitive information, such as employee data.
- Accessibility - the interface is accessible to users with disabilities, in accordance with accessibility standards for a diverse user audience.
- Auditability - the system maintains detailed logs for audit purposes, while ensuring a transparent record of user actions and system activities.

4. Website Diagrams

4.1. Architecture Diagram



4.2. Use Case Diagram



5. Website Design

5.1. Three Layer Architecture

Business Logic Layer (BLL)

The BLLs are responsible for executing the business rules and workflows specific to the factory management domain. This layer manages the logical operations of the system, such as validating user inputs, processing business transactions, and ensuring data integrity throughout the application.

Data Access Layer (DAL)

DALs provides a unified interface to the underlying database, whether it's MongoDB or any other storage system, allowing for data retrieval, insertion, update, and deletion operations. The DAL ensures that the BLL can manipulate the data without needing to know about the specifics of the database implementation.

Routers

Routers define the application's endpoints (URLs) and delegate the requests to the appropriate handlers in the BLL. In the context of the Factory Management Web App, routers are tailored for specific entities such as employees, shifts, departments, and authentication processes. Each router maps the client's actions to the corresponding business logic operations, ensuring a clean and organized structure for managing request flows.

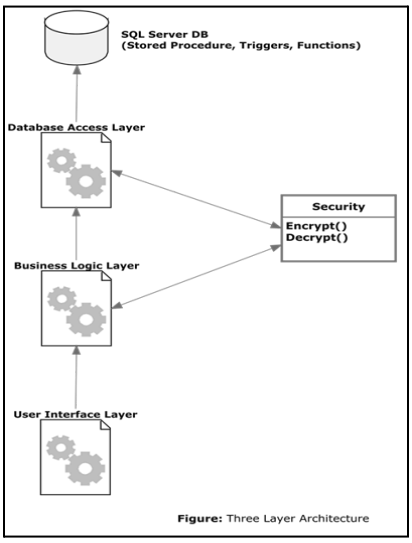
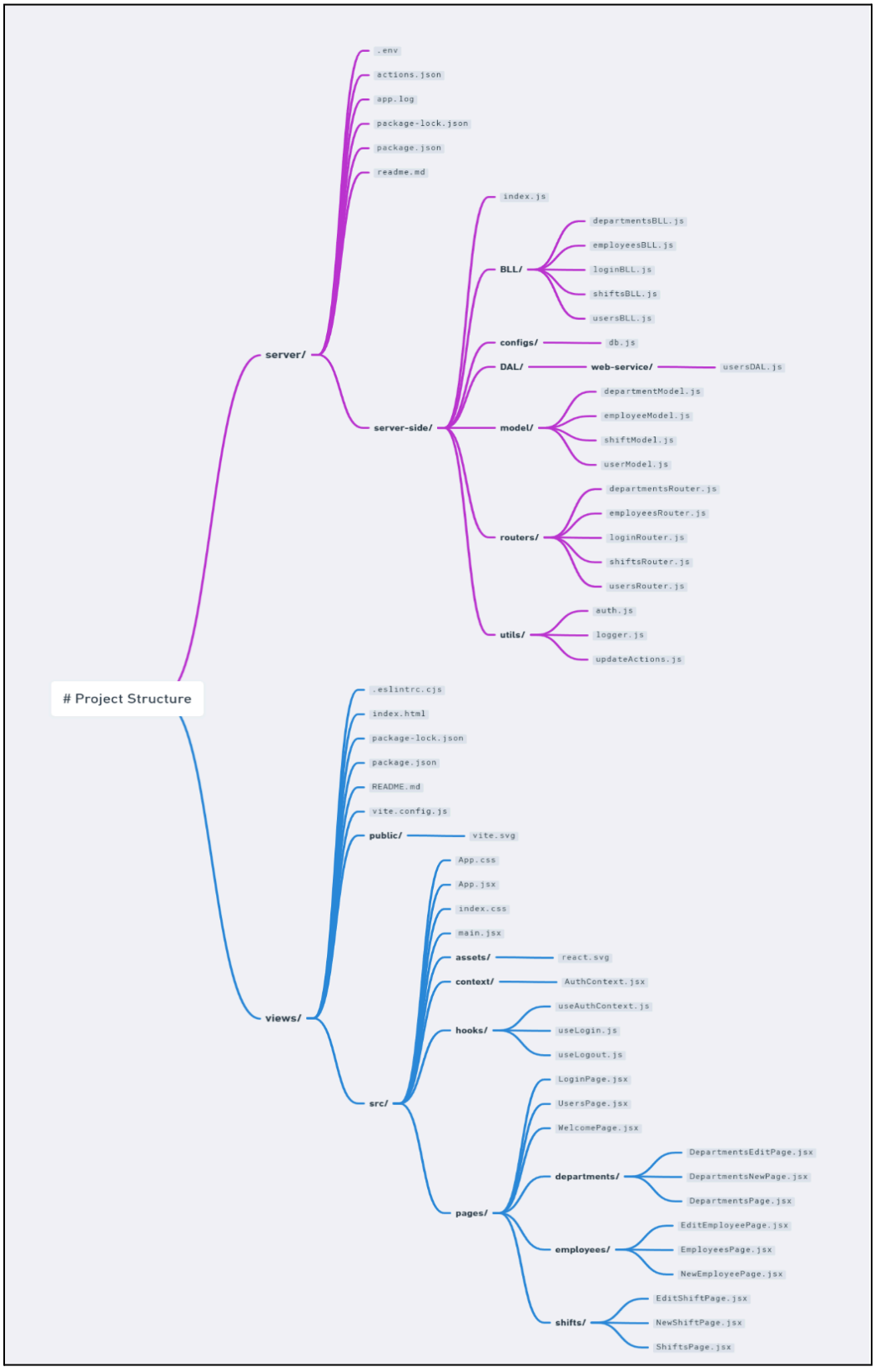


Figure: Three Layer Architecture

5.2. Folder and File Structure



Server Side Structure (`server/`)

The server side is structured into directories and subdirectories, each serving a specific purpose in the application's backend:

- `server/` Root Directory: Contains global configuration and utility files for the server-side application.
  - `.env:` Environment configuration file containing sensitive keys and settings.
  - `actions.json:` Logs of user actions for monitoring and auditing.
  - `app.log:` Logging file for application events and errors.
  - `package.json, package-lock.json:` NPM configuration and lock files for managing project dependencies.
- `server-side/` Directory: Encapsulates the core logic of the backend.
  - `BLL/:` Business Logic Layer, containing JavaScript files (`departmentsBLL.js, employeesBLL.js, loginBLL.js, shiftsBLL.js, usersBLL.js`) that implement the business rules and operations.
  - `configs/:` Holds configuration files like `db.js` for database connections.
  - `DAL/:` Data Access Layer, with a subdirectory `web-service/` specifically for accessing user data (`usersDAL.js`).
  - `model/:` Mongoose models (`departmentModel.js, employeeModel.js, shiftModel.js, userModel.js`) representing the structure of data in the database.
  - `routers/:` Express routers (`departmentsRouter.js, employeesRouter.js, loginRouter.js, shiftsRouter.js, usersRouter.js`) that define the endpoints and HTTP request handling.
  - `utils/:` Utility functions and middleware for authentication (`auth.js`), logging (`logger.js`), and action tracking (`updateActions.js`).

Client Side Structure (`views/`)

The client side, developed with React and Tailwind CSS, features a modular and component-based structure:

- `views/` Root Directory: Contains the main entry point and configuration files for the frontend application.
  - `index.html:` The main HTML file that serves as the entry point for the React app.
  - `package.json, package-lock.json:` Manages dependencies and project settings for the frontend.
  - `.eslintrc.cjs:` ESLint configuration file for maintaining code quality and consistency.
- `src/` Directory: Source code for the React application, structured into several key subdirectories and files.
  - `App.jsx, App.css:` Main React component and its styling.
  - `index.css, main.jsx:` Global styles and the entry JavaScript file for React.
  - `assets/:` Static assets like images and logos.
  - `context/:` Contains React context (`AuthContext.jsx`) for global state management.
  - `hooks/:` Custom React hooks (`useAuthContext.js, useLogin.js, useLogout.js`) for reusable logic.
  - `pages/:` Components representing individual pages, organized by functionality (e.g., `Departments, Employees, LoginPage`), with further subcomponents for specific actions like edit or new entry forms.

6. Database and API Integration

6.1. External Database/API Integration

User Authentication via External REST API

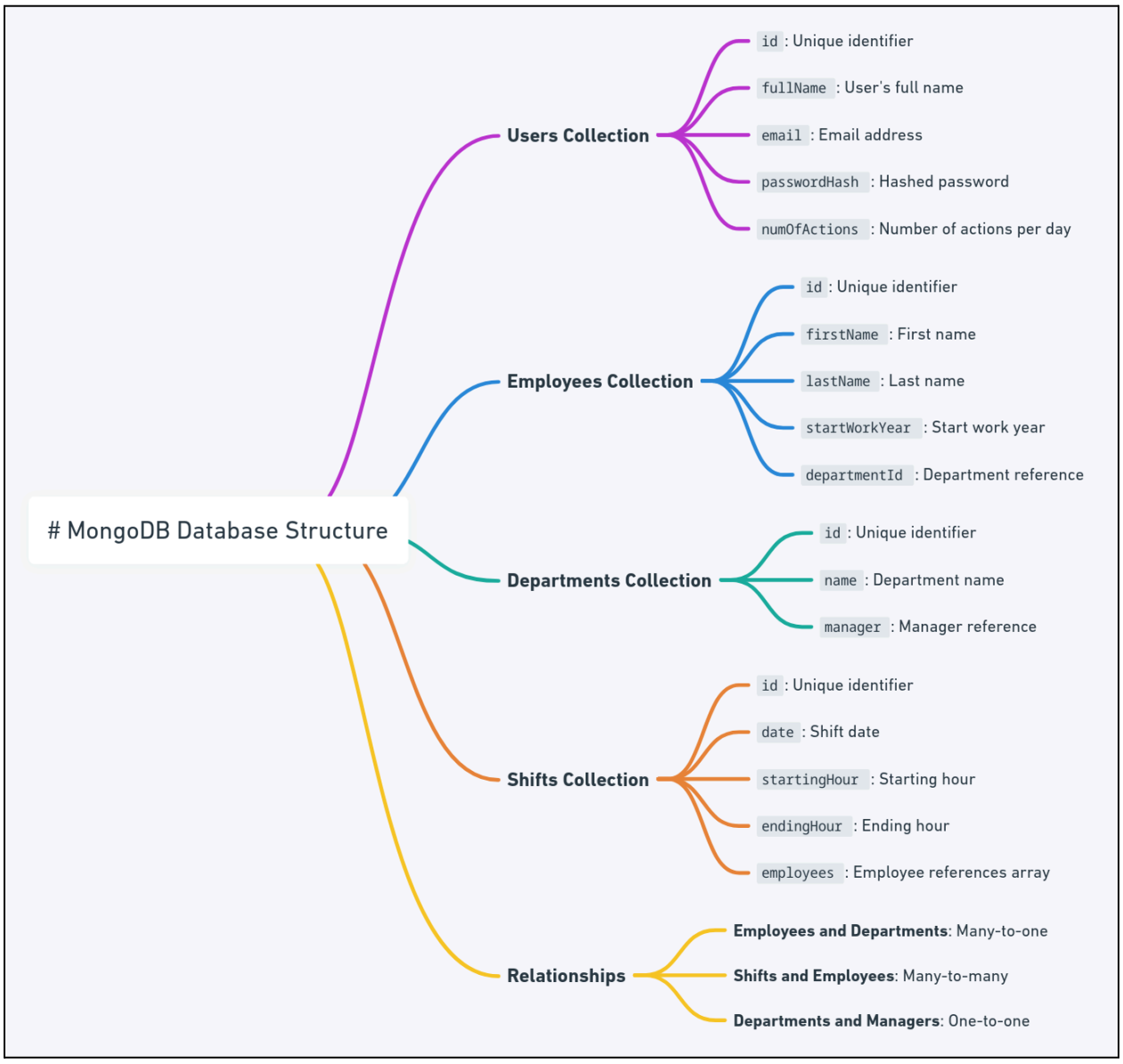
The project uses JWT (JSON Web Tokens) for user authentication, leveraging an external REST API, specifically **`https://jsonplaceholder.typicode.com/users`**, to validate user credentials. This process is outlined in the server-side architecture, particularly within the **`loginBLL.js`** and **`usersDAL.js`** files under the **`BLL`** (Business Logic Layer) and **`DAL`** (Data Access Layer) directories, respectively.

- Process Flow:
  i. User Login Request: When a user attempts to log in, the front end sends the credentials (username and password) to the server.
  ii. Credential Verification: The **`loginRouter.js`** receives the login request and forwards the credentials to **`loginBLL.js.`**
  iii. External API Call: The **`loginBLL.js`** utilizes **`usersDAL.js`** to make a call to the external REST API (**`https://jsonplaceholder.typicode.com/users`**) to verify if the user exists.
  iv. Token Generation: Upon successful verification, a JWT token is generated and sent back to the client, signifying successful authentication.

MongoDB Integration for Data Storage and Management

- The database configuration and connection details are managed in **`configs/db.js`**, which sets up the connection to the MongoDB instance.
- Data models defined in the `model/` directory (**`departmentModel.js`**, **`employeeModel.js`**, **`shiftModel.js`**, **`userModel.js`**) represent the structure of data stored in MongoDB. These models are used by the `DAL` components to interact with the database effectively, allowing for operations like create, read, update, and delete (CRUD) to be performed on the data.
- The `DAL` components, such as **`departmentsBLL.js`**, **`employeesBLL.js`**, and **`shiftsBLL.js`**, use the Mongoose models to access and manipulate the data in MongoDB according to the business logic requirements.

6.2. Database Structure Diagram



Collections and Schemas

1. Users Collection
    - This collection stores information about the users who have access to the system. Each document in the collection represents a user with fields for authentication and authorization purposes.
    - Fields:
        - `id:` Unique identifier for the user.
        - `fullName:` The user's full name.
        - `email:` Email address used for login.
        - `numOfActions:` Integer representing the number of actions a user can perform per day (for rate limiting).

2. Employees Collection
    - Contains details about factory employees. Each document corresponds to an individual employee.
    - Fields:
        - `id:` Unique identifier.
        - `firstName:` Employee's first name.
        - `lastName:` Employee's last name.
        - `startWorkYear:` The year the employee started working at the factory.
        - `departmentId:` Reference to the department the employee belongs to.
3. Departments Collection
    - This collection holds information about the different departments within the factory. Each document represents one department.
    - Fields:
        - `id:` Unique identifier.
        - `name:` The name of the department.
        - `manager:` Reference to an employee id who is the manager of the department.

4. Shifts Collection
    - Stores information about work shifts. Each document in this collection details a specific shift.
    - Fields:
        - `id:` Unique identifier.
        - `date:` The date of the shift.
        - `startingHour:` The starting hour of the shift.
        - `endingHour:` The ending hour of the shift.
        - `employees:` An array of references to employee ids who are assigned to the shift.

7. Development and Documentation

7.1. Key Functions

1. **User Authentication,** This function involves verifying user credentials against an external API and issuing JWT tokens for session management. It ensures that users can securely log in and interact with the app according to their permissions.

2. **Employee Management,** Enables CRUD operations on employee data, including adding new employees, updating their information, and removing them from the system. This function is critical for keeping the factory's workforce data up to date.

3. **Department Management,** Similar to employee management, this function allows for the creation, update, and deletion of department records. It's vital for organizing the workforce into structured units and managing them efficiently.

4. **Shift Scheduling,** Facilitates the creation and modification of work shifts, assigning employees to these shifts. This function is key to managing the operational aspects of the factory, ensuring that all shifts are adequately staffed.

7.2. Listing APIs

Employees API

- GET /employees
  - Retrieves a list of all employees in the system. This endpoint can support query parameters for filtering, sorting, and pagination to efficiently manage large datasets.
- GET /employees/{id}
  - Fetches detailed information about a specific employee, identified by their unique ID. This can include personal details, department affiliation, and assigned shifts.

Departments API

- GET /departments
  - Returns a list of all departments, with the option to include additional details such as the department manager and the employees belonging to each department.
- GET /departments/{id}
  - Provides detailed data about a single department, including its name, manager, and list of employees. This endpoint is essential for managing departmental structures and hierarchies.

Shifts API

- GET /shifts
  - Obtains a list of all work shifts, which can be filtered by date, department, or other criteria to aid in scheduling and workforce management.
- GET /shifts/{id}
  - Delivers detailed information about a particular shift, including the date, start and end times, and the employees assigned to the shift.

Authentication and User Management API

- POST /login
  - Handles user authentication, accepting username and password parameters and returning a JWT token upon successful authentication.
- GET /users
  - Lists all users with access to the system, primarily used for administrative purposes to manage user permissions and roles.

7.3. Special Environments

Development Environment

- Visual Studio Code: Chosen for its extensive ecosystem of extensions, integrated terminal, and source control features, making it ideal for both frontend and backend development.
- Vite for Frontend: Utilizes Vite as the build tool for the React-based frontend, enabling fast development with features like Hot Module Replacement (HMR). Vite's out-of-the-box support for React and its performance optimizations make it an excellent choice for this project.
- Express.js for Backend: The backend is built on Express.js, structured around Routers, Business Logic Layer (BLL), and Data Access Layer (DAL) to manage API requests, business rules, and database interactions cleanly and efficiently.

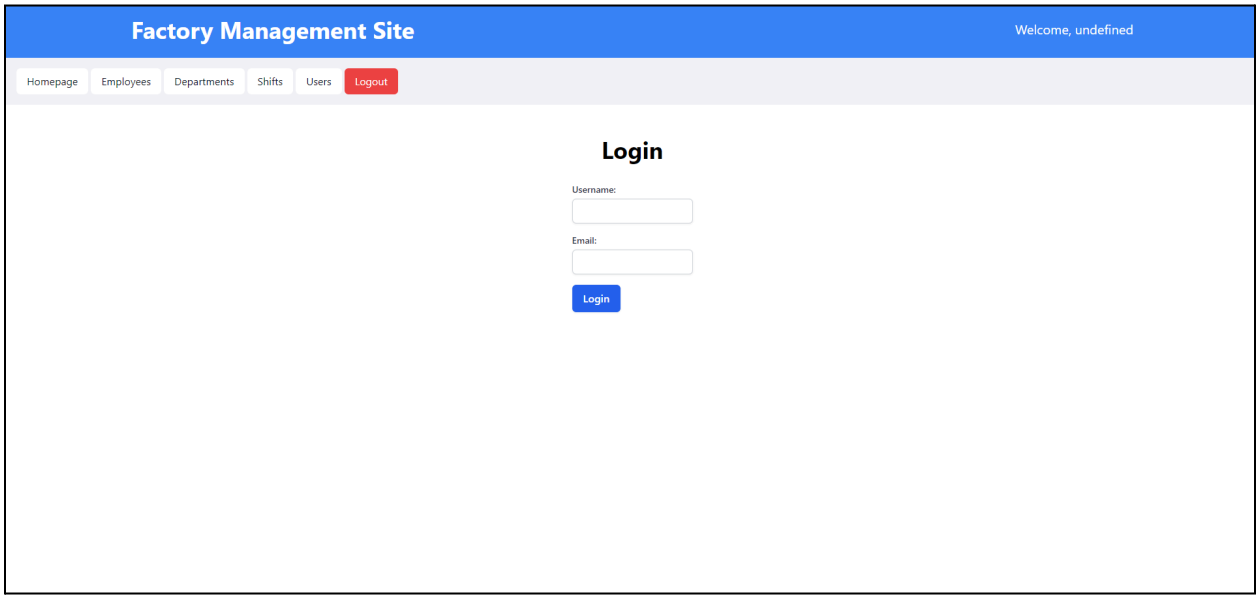
External APIs and Database

- External API: The project integrates with `https://jsonplaceholder.typicode.com/users` for user data, demonstrating the capability to interact with third-party services for authentication or other features.
- MongoDB Atlas: As the chosen database solution, MongoDB Atlas offers a fully managed cloud database that provides scalability, high availability, and global distribution, crucial for modern web applications.

7.4. User Documentation

Getting Started

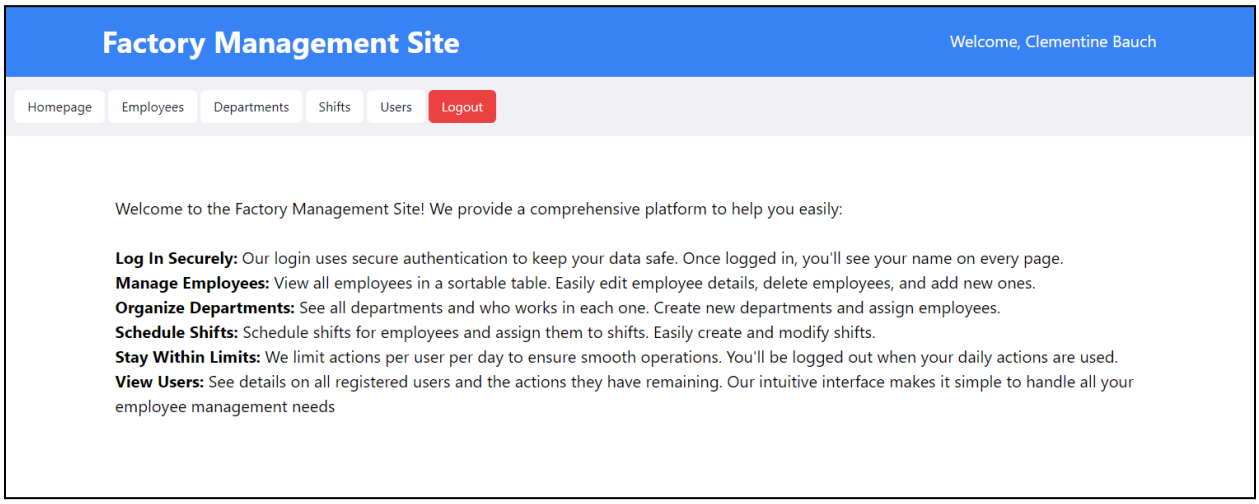
1. Accessing the Web App
    - Open a web browser of your choice.
    - Navigate to the URL where the Factory Management Web App is hosted (e.g., `https://a22-web-project.vercel.app/`).
2. **Logging In**
    - On the landing page, locate the login section.
    - Enter your username and email from the list of users on the API `https://jsonplaceholder.typicode.com/users`.
    - Click the "Login" button to access the main dashboard.



Navigating the Dashboard

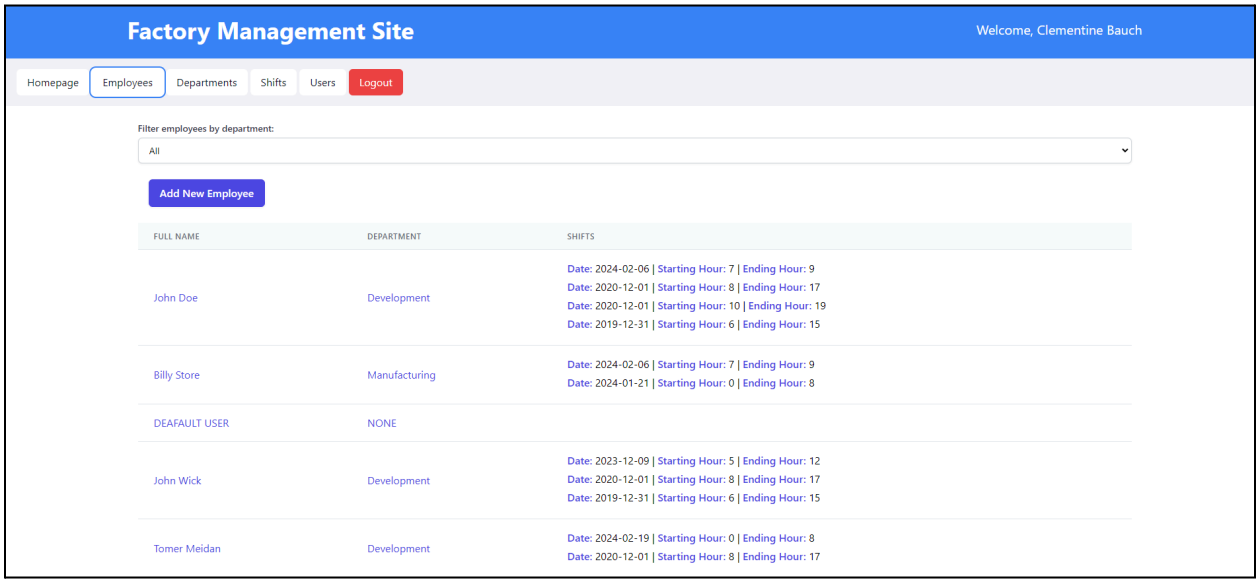
Upon successful login, you'll be directed to the dashboard, which serves as the central hub for accessing the various features of the web app.

- Dashboard Overview - The dashboard displays an overview of the factory's operations.

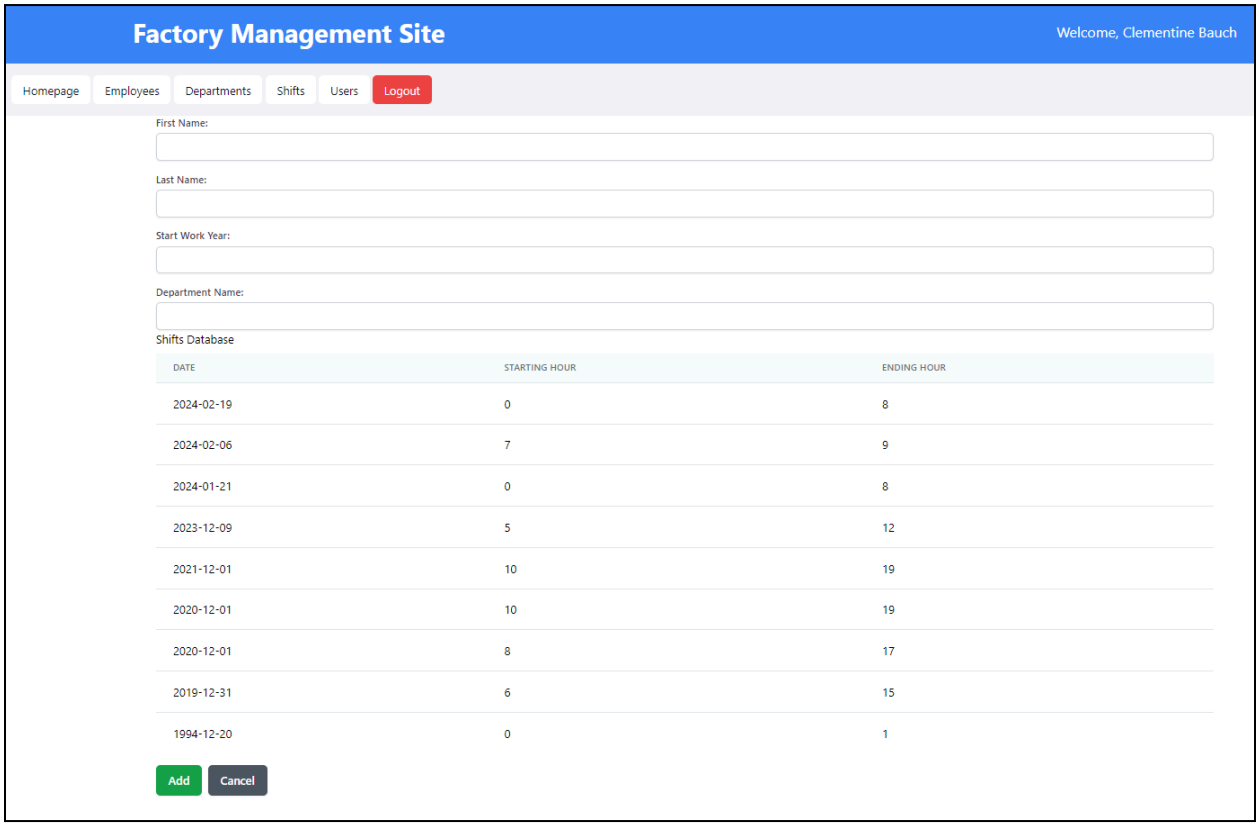


Managing Employees

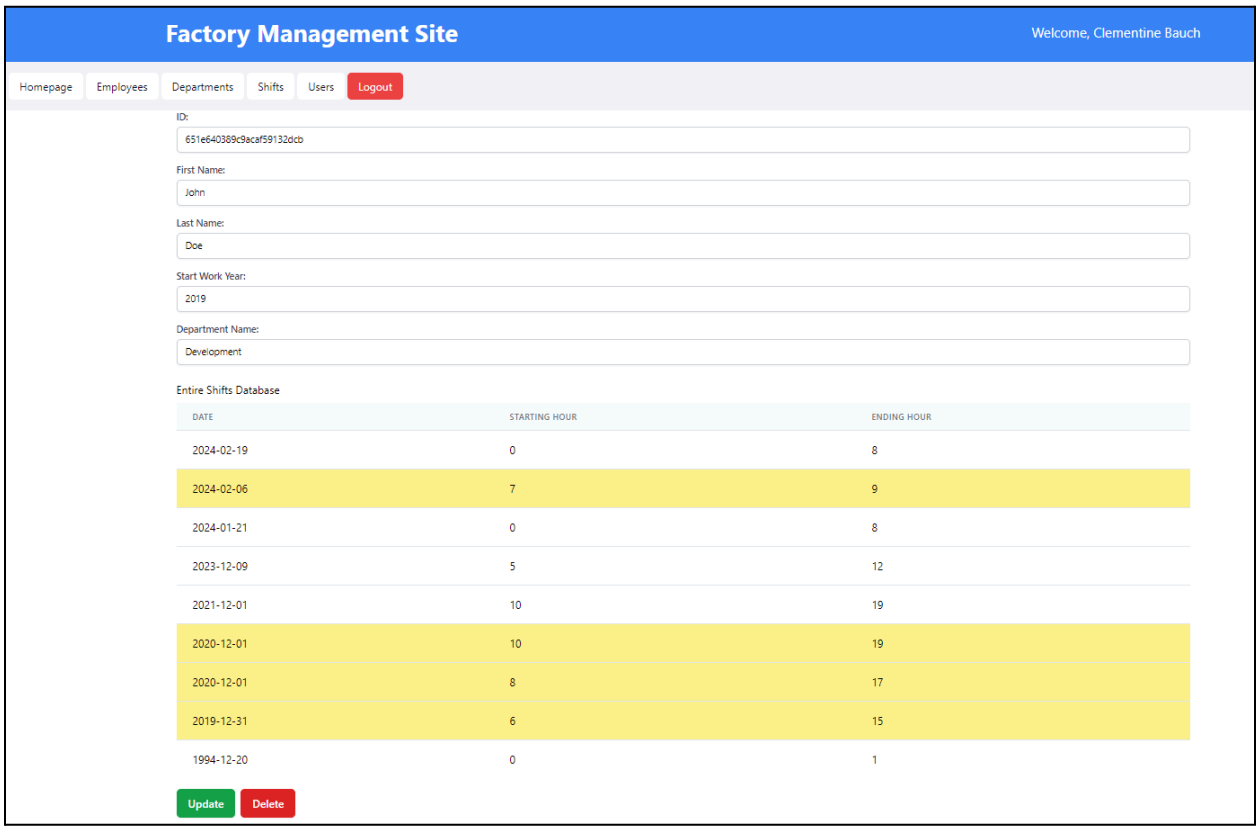
1. Viewing Employees
    - Navigate to the "Employees" section from the dashboard or menu.
    - Here, you'll see a list of all employees, their departments, and other relevant details.



2. Adding a New Employee
    - Click on the "New Employee" button.
    - Fill in the required information, such as first name, last name, department, and start work year.
    - Submit the form to add the employee to the database.

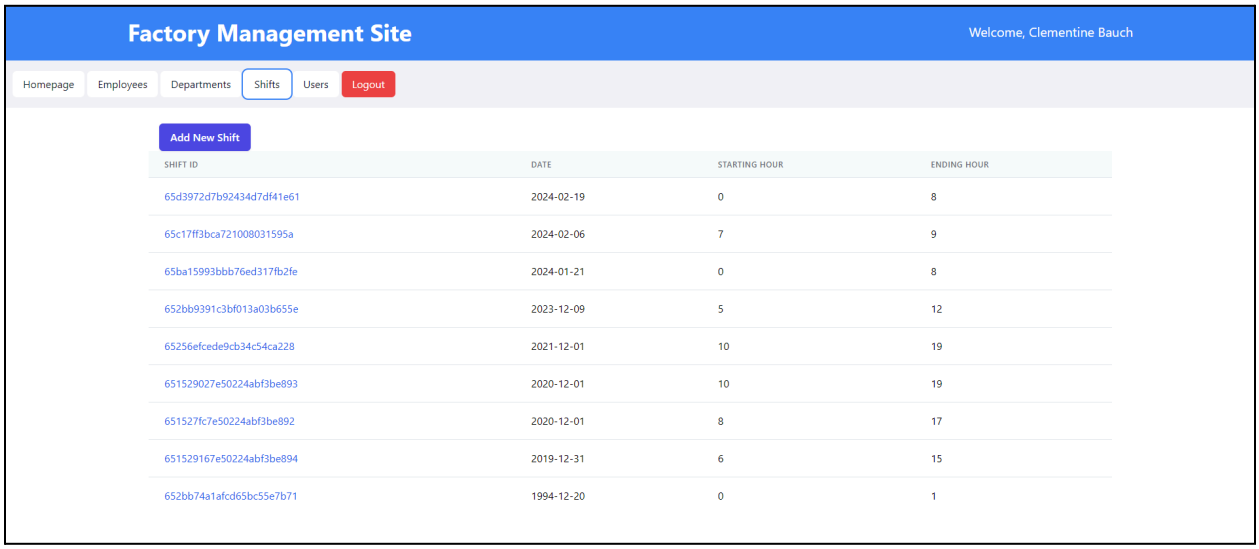


3. Editing Employee Details
    - Find the employee you wish to edit and click on their name or the edit icon next to their record.
    - Update the necessary fields and save the changes.

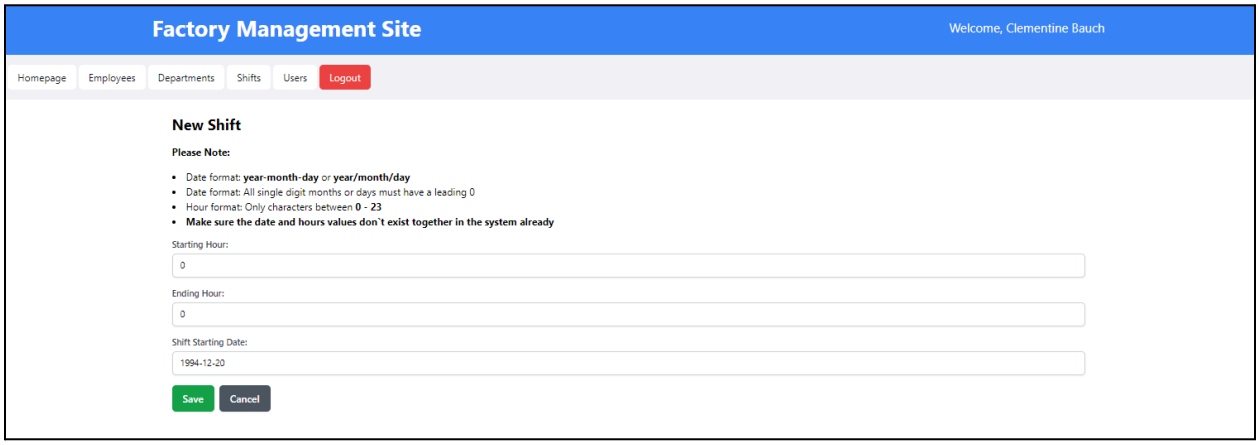


Scheduling Shifts

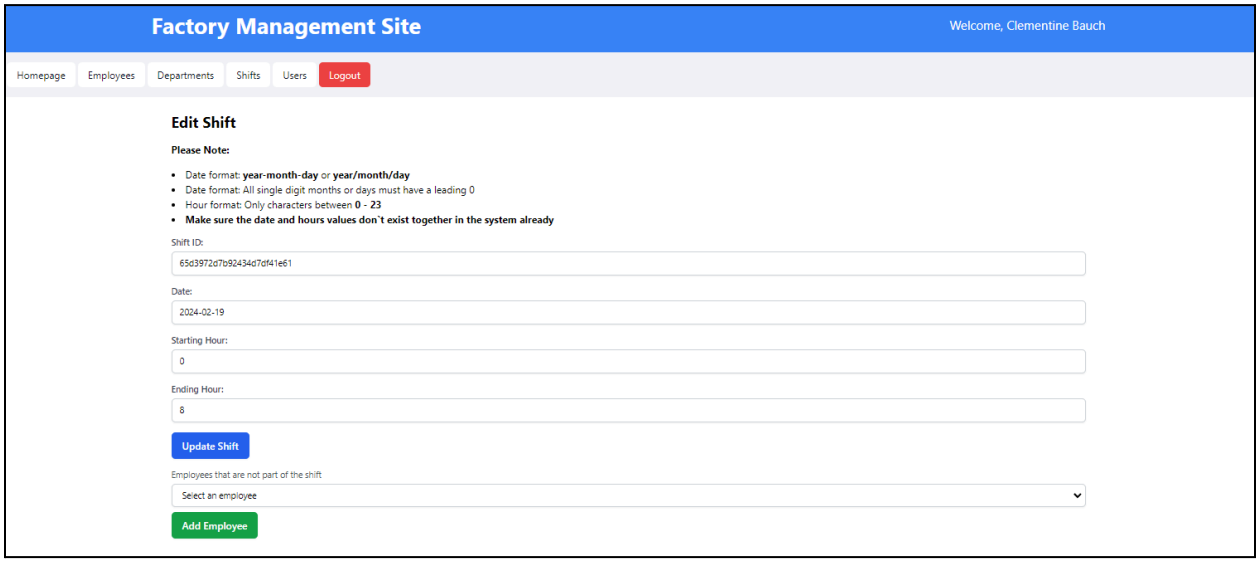
1. Viewing Shifts
    - Access the "Shifts" section to view the current shift schedule.



2. Creating a New Shift
    - Click on "New Shift" and specify the shift details, including the date, starting hour, and ending hour.
    - Assign department name to the shift and submit the form.

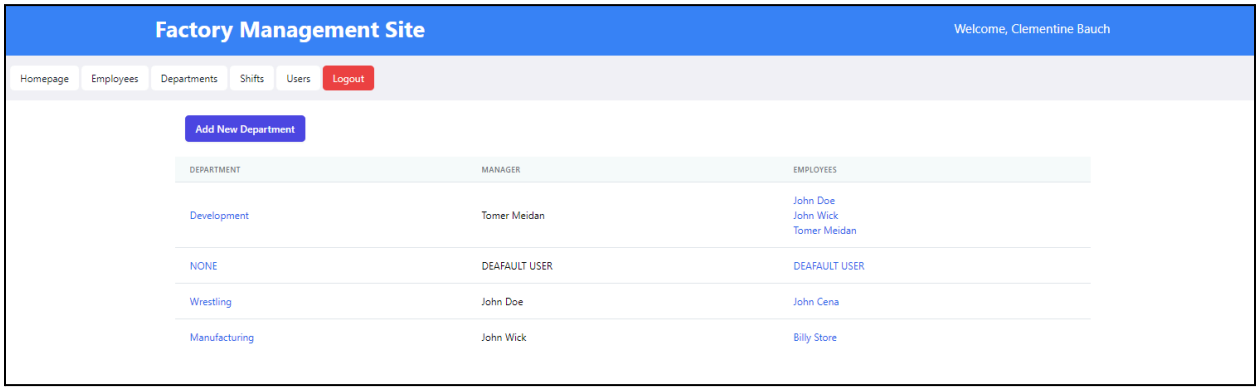


3. Edit Shift
    - Click on a shift and it will lead you to an edit page for said shift.

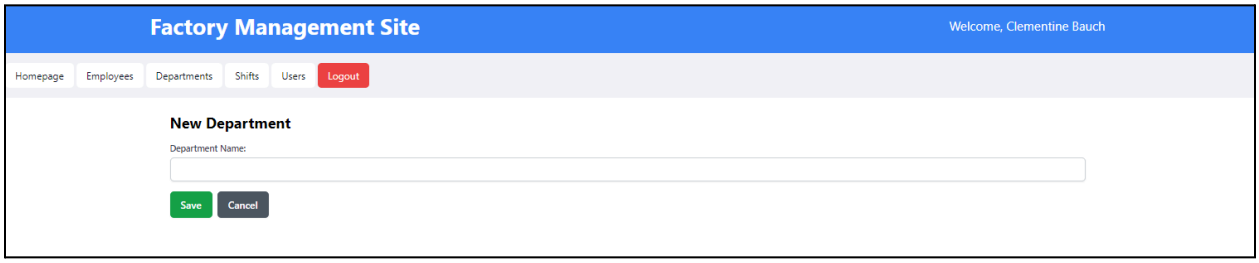


Department Management

1. Viewing Departments
    - Go to the "Departments" area to see a list of all departments and their managers.



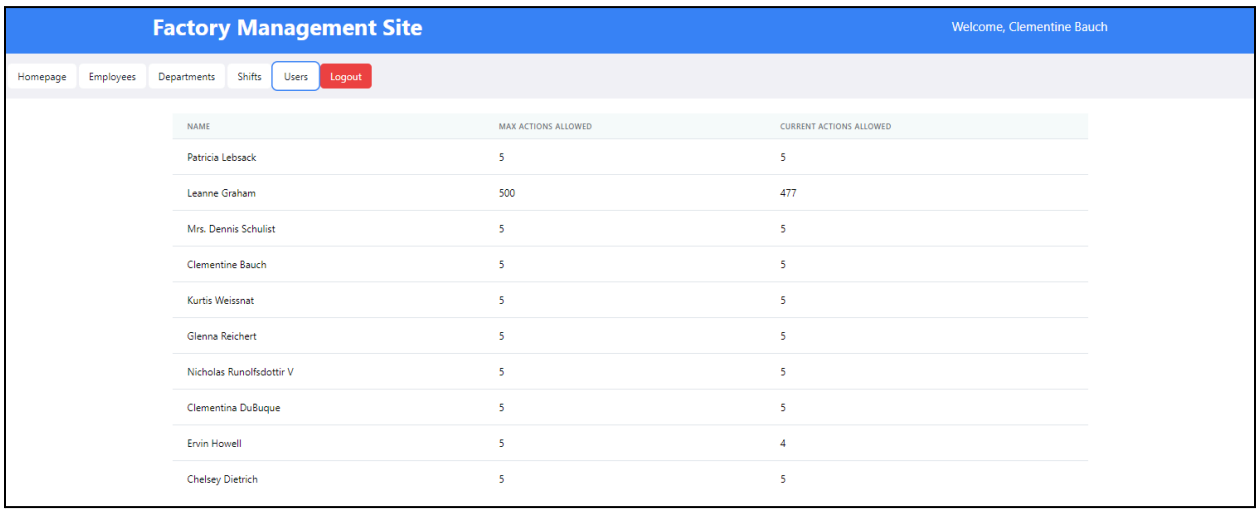
2. Adding/Editing Departments
    - To add a new department, click "New Department", fill in the name and submit.



    - Edit a department by selecting it from the list and updating its information as needed.

Users Management

3. Viewing Users
    - Go to the "Users" area to see a list of all Users and their actions allowed.



Logging Out

- To log out of the application, click the "Log Out" link or button located at the top left of the screen.

8. References

- Whimsical for UML Diagrams
  Whimsical provides a suite of online tools for creating flowcharts, wireframes, and UML diagrams, making it a useful resource for planning and visualizing software architectures and workflows.
  Visit Whimsical

- React Documentation
  The official React documentation offers comprehensive guides, tutorials, and API references for developers working with React, a popular JavaScript library for building user interfaces.
  Learn more about React

- Three-Tier Architecture - IBM
  IBM's explanation of the three-tier architecture provides insights into this design pattern, which separates applications into three logical and physical computing tiers: the presentation layer, the application layer, and the data layer. This architecture is fundamental for building scalable, maintainable, and secure applications.
  Understanding Three-Tier Architecture