



DWSIM Semester Long Internship Autumn 2025-2026

On

Development of a Web-based App for Chemical PFD and P&ID

Submitted by

Thu Kha Pyae Sone

Final Year Student
Bachelor of Engineering (Honours) in
Computer Science and Engineering
Myanmar Institute of Information Technology
Mandalay, Myanmar

Under the guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

Self-Declaration by the Student

I hereby declare that this Capstone Project Final Report, submitted in partial fulfilment of the DWSIM Semester Long Internship Programme, represents work carried out by me at FOSSEE, IIT Bombay, under the guidance of Mr. Priyam Nayak, Technical Mentor, and Prof. Prabhu Ramachandran, Principal Investigator. I affirm that the contents of this report have been authored entirely by me, and that all text, figures, tables, and other material not originating from my own work have been duly acknowledged with appropriate citations and references. I further declare that this report is free from plagiarism and any form of academic misconduct. I acknowledge that full responsibility for any breach of academic integrity that may be identified in this report rests solely with me. Thu Kha Pyae Sone,

2017-MIIT-CSE-052 Self-Declaration by the Student I hereby declare that this Capstone Project Final Report, submitted in partial fulfilment of the DWSIM Semester Long Internship Programme, represents work carried out by me at FOSSEE, IIT Bombay, under the guidance of Mr. Priyam Nayak, Technical Mentor, and Prof. Prabhu Ramachandran, Principal Investigator. I affirm that the contents of this report have been authored entirely by me, and that all text, figures, tables, and other material not originating from my own work have been duly acknowledged with appropriate citations and references. I further declare that this report is free from plagiarism and any form of academic misconduct. I acknowledge that full responsibility for any breach of academic integrity that may be identified in this report rests solely with me.

Thu Kha Pyae Sone

2017-MIIT-CSE-052

Myanmar Institute of Information Technology

Date: _____

Certificate

This is to certify that the Capstone Project Final Report entitled Development of a web-based app for Chemical PFD and P&ID, submitted by Thu Kha Pyae Sone (2017-MIIT-CSE-052), represents work carried out by the student under the DWSIM Semester Long Internship Programme at FOSSEE, IIT Bombay, in partial fulfilment of the requirements for the award of the B.E. (Hons.) Degree in Computer Science and Engineering.

Mr. Priyam Nayak
Technical Mentor
FOSSEE, IIT Bombay

Date: _____

Dr. May Mya Aung
Professor
Myanmar Institute of Information
Technology

Date: _____

Dr. Myat Thuzar Tun
Pro-Rector
Myanmar Institute of Information
Technology

Date: _____

Dr. Win Aye
Rector
Myanmar Institute of Information
Technology

Date: _____

Acknowledgment

I am grateful to the FOSSEE team for giving me this opportunity to contribute to the Development of a web-based app for Chemical PFD and P&ID and for providing all the support and resources I needed throughout my internship. It has been a truly valuable experience.

I would like to thank **Prof. Prabhu Ramachandran** for his oversight and support of this internship, and Prof. **Kannan M. Moudgalya**, former head professor, whose efforts in arranging this opportunity made my internship at IIT Bombay possible.

I am deeply thankful to my instructor, Mr. **Priyam Nayak**, DWSIM Project Lead at FOSSEE, IIT Bombay, for his technical guidance, helpful feedback during weekly reviews, and patience in helping me understand the DWSIM codebase. His vision for making DWSIM accessible to a wider range of users.

I would also like to thank my internal mentor, Mr. **Chirag Koyande**, and my eSim members, Mr. **Varad Patil**, and Ms. **Shanthi Priya**, for their support, technical inputs, and feedback throughout the development process. Their guidance helped me stay on track and work through the technical challenges involved in building the web application. My thanks also go to the broader FOSSEE team for their help and guidance along the way.

I am grateful to Mr. **Piyush Kumar** from the Dean of International Relations (Dean IR) office for helping me and my fellow international students with our documentation and administrative needs, making our stay much smoother.

I would also like to thank my professor in Myanmar, Mr. **Joe Tun Sein**, for his role in helping us get this internship opportunity at IIT Bombay. His support and dedication to his students is something I truly appreciate.

I acknowledge my parent institute, Myanmar Institute of Information Technology, my supervisor, Professor Dr. **May Mya Aung**, Pro-Rector **Dr. Myat Thuzar Tun** and Rector **Dr. Win Aye**, for their encouragement and support.

Finally, I thank my family and friends for their patience, belief, and motivation throughout this period. This accomplishment means a lot to me, and I could not have done it without them.

This internship lasted four months, one month working remotely from Myanmar and three months on-site at IIT Bombay. Working on the Development of a Web-based App for Chemical PFD and P&ID, has been a meaningful and challenging experience that has grown my technical skills and broadened my perspective. I will carry what I have learned here with me going forward.

Content

Chapter 1: Introduction.....	7
1.1 Project Overview.....	7
1.2 Objectives.....	7
1.3 Scope of the Project.....	8
Chapter 2: System Design.....	9
2.1 System Architecture.....	9
2.2 User Roles.....	9
2.2.1 Admin.....	9
2.2.2 Authenticated Users.....	9
2.3 Technology Used.....	9
2.3.1 Backend Framework.....	10
2.3.2 Database.....	10
2.3.3 Authentication Method.....	10
2.3.4 Rate Limiting.....	10
Chapter 3: Database Design.....	11
3.1 Database Overview.....	11
3.2 Database Migration to PostgreSQL.....	11
3.3 Database Structure.....	11
3.3.1 User Table.....	11
3.3.2 Project Table.....	11
3.3.3 Component Table.....	12
3.3.4 Canvas State Table.....	12
3.3.5 Connection Table.....	12
3.4 Relationships Between Tables.....	12
3.4.1 User–Project Relationship.....	12
3.4.2 Project–Component Relationship.....	12
Chapter 4: Authentication Module.....	14
4.1 User Registration.....	14
4.1.1 Registration Process.....	14
4.2 User Login.....	14
4.2.1 Login Process.....	14
4.3 Token Management.....	14
4.3.1 Access Token.....	15
4.3.2 Refresh Token.....	15
Chapter 5: Rate Limiting with django-axes.....	16
5.1 Overview.....	16
5.2 Login Attempt Protection.....	16
5.3 API Rate Limiting.....	16

5.4 Integration with the Authentication Flow.....	17
Chapter 6: Admin Data Import.....	18
6.1 Import Feature Overview.....	18
6.1.1 Purpose of ZIP Import.....	18
6.2 Import Process.....	18
6.2.1 Upload ZIP File.....	18
6.2.2 Extract and Store Data.....	18
Chapter 7: API Implementation.....	20
7.1 Authentication APIs.....	20
7.1.1 Register API.....	20
7.1.2 Login API.....	20
7.1.3 Refresh Token API.....	20
7.2 Project APIs.....	20
7.2.1 Create Project API.....	20
7.2.2 Update Project API.....	21
7.2.3 Delete Project API.....	21
7.2.4 Fetch Projects API.....	21
7.2.5 Fetch Project Details API.....	21
7.3 Component APIs.....	21
7.3.1 Create Component API.....	21
7.3.2 Fetch Components API.....	21
Chapter 8: Current Progress.....	22
8.1 Completed Features.....	22
8.1.1 Database Setup and PostgreSQL Migration.....	22
8.1.2 Authentication System.....	22
8.1.3 Rate Limiting with django-axes.....	22
8.1.4 Admin Import Function.....	22
8.1.5 Project and Component APIs.....	23
8.2 Challenges Faced.....	23
8.2.1 Technical Issues Encountered.....	23
Chapter 9: Future Work.....	24
9.1 Planned Improvements.....	24
9.1.1 Deployment to a Cloud Server.....	24
9.1.2 CI/CD Pipeline with GitHub Actions.....	24
Chapter 10: Conclusion.....	25
10.1 Summary.....	25

Chapter 1: Introduction

1.1 Project Overview

This project involves the development of a web-based application for creating Process Flow Diagrams (PFD) and Process & Instrumentation Diagrams (P&ID) for chemical process plants. PFD and P&ID diagrams are widely used in chemical and process industries to represent process equipment, material flow, instrumentation, and control systems in a standardized and structured manner. Developing a web-based solution for such diagrams improves accessibility, collaboration, and flexibility compared to traditional desktop-based tools.

The overall project aims to provide an interactive platform that allows users to design PFD and P&ID diagrams using standardized engineering symbols, validation rules, and export functionalities. The complete system is designed as a client–server application, where the frontend handles user interaction and diagram visualization, while the backend manages data storage, authentication, and system logic.

In this team-based project, the scope of this work is focused specifically on the backend system development. The backend serves as the core infrastructure that supports the frontend diagramming interface developed by other team members. This includes the design of APIs, database structures, authentication mechanisms, rate limiting, and administrative features required for the system to function reliably and securely.

The increasing demand for digital transformation in the chemical and process industries has highlighted the need for flexible and collaborative engineering tools. Traditional desktop-based software for creating PFD and P&ID diagrams often lacks real-time collaboration capabilities and platform independence. A web-based solution addresses these limitations by enabling users to access and modify diagrams from any location without requiring specialized installations.

Additionally, such a system can integrate with modern cloud infrastructure, allowing for better scalability, data backup, and collaborative workflows. By centralizing diagram data and providing standardized interfaces, the application can also support future enhancements such as simulation integration, version control, and multi-user editing. This makes the platform not only a diagramming tool but also a foundation for a more comprehensive chemical process design ecosystem.

1.2 Objectives

The primary objective of the overall project is to develop a web-based application that enables the creation and management of Process Flow Diagrams (PFD) and Process & Instrumentation Diagrams (P&ID) for chemical process plants. The system is intended to provide an intuitive diagramming environment supported by standardized symbols and validation mechanisms.

The objectives of the backend development, which is the focus of this work, are as follows:

1. To design and implement a secure and scalable backend architecture to support a web-based PFD and P&ID diagramming application.
2. To develop RESTful APIs that allow the frontend to create, store, retrieve, update, and manage diagram-related data.
3. To implement authentication and authorization mechanisms using token-based security.
4. To design a database structure capable of storing projects, diagrams, components, and their relationships.
5. To migrate the database from SQLite to PostgreSQL for improved scalability and performance.
6. To implement application-level rate limiting using django-axes to protect against brute-force attacks and API abuse.
7. To support future system expansion by ensuring the backend architecture is modular and extensible.

While frontend features such as drag-and-drop diagram creation, symbol libraries, and diagram export are part of the overall project objectives, their implementation is handled by other members of the development team.

1.3 Scope of the Project

The scope of this project is limited to the backend development of a web-based PFD and P&ID diagramming system. The backend is responsible for managing application data, handling authentication, providing APIs, and supporting administrative operations.

The implemented backend functionalities include user registration and login, token-based authentication, rate limiting and brute-force protection, PostgreSQL database integration, project and diagram data management, component handling, and administrative data import. These features provide the necessary infrastructure for the frontend application to perform diagram creation and visualization.

Frontend implementation, including user interface design, drag-and-drop functionality, symbol rendering, and diagram export, is outside the scope of this work and is undertaken by other team members.

Chapter 2: System Design

2.1 System Architecture

The system is designed as a web-based client–server application to support the creation and management of Process Flow Diagrams (PFD) and Process & Instrumentation Diagrams (P&ID). The overall system architecture separates responsibilities between the frontend and backend to ensure modularity, scalability, and ease of maintenance. While the frontend handles diagram visualization and user interaction, the backend manages data storage, authentication, and application logic.

The backend system is implemented as an API-based service using the Django web framework. Django REST Framework (DRF) is used to expose RESTful APIs that enable communication between the frontend and backend through HTTP requests. All interactions with the system, including user authentication, project management, and diagram-related data operations, are performed through these APIs.

This API-based design ensures loose coupling between system components, allowing frontend and backend development to proceed independently. It also supports future system expansion, such as the integration of simulation tools, industrial tag databases, and collaborative features, without requiring significant architectural changes.

Furthermore, the architecture follows a layered design approach, separating concerns into presentation, application, and data layers. This separation improves maintainability and allows individual components to evolve independently. The use of RESTful APIs also enables interoperability with external systems, making it possible to integrate third-party tools such as simulation engines, reporting systems, or industrial databases in future iterations of the project.

2.2 User Roles

The system defines user roles to control access to backend functionalities and ensure secure operation. Role-based access is enforced at the backend level through authentication and authorization mechanisms.

2.2.1 Admin

Admin users have elevated privileges within the system. They are responsible for system-level operations such as managing core application data, performing bulk data imports, and maintaining standardized diagram-related information. Admin access is restricted to authorized users and is protected through backend authentication and authorization rules.

2.2.2 Authenticated Users

Authenticated users are standard users who can access the system after successful login. These users are allowed to create, update, and manage projects and diagram-related data through the backend APIs. Authentication ensures that only verified users can access

protected resources, and all actions performed by authenticated users are validated by the backend system.

The user role design allows for future expansion, such as introducing additional roles or fine-grained permission levels, without major changes to the backend architecture.

2.3 Technology Used

This section describes the technologies used for implementing the backend system and supporting its core functionalities.

2.3.1 Backend Framework

The backend is developed using the Django web framework, which provides a robust and scalable foundation for web application development. Django REST Framework (DRF) is used to implement RESTful APIs, enabling structured communication between the frontend and backend. The use of Django and DRF allows rapid development while maintaining code organization and security.

2.3.2 Database

A PostgreSQL relational database management system is used to store application data, including user information, project details, and diagram-related components. PostgreSQL was selected for its superior performance, support for high concurrency, advanced indexing capabilities, and better transaction management compared to SQLite. Django's Object-Relational Mapping (ORM) is utilized to manage database interactions, allowing database records to be handled as Python objects. This approach simplifies database operations and ensures data consistency through enforced relationships.

2.3.3 Authentication Method

Authentication is implemented using JSON Web Tokens (JWT). Upon successful login, the backend issues an access token and a refresh token to the user. The access token is required for accessing protected API endpoints, while the refresh token is used to obtain a new access token when the current one expires. This stateless authentication mechanism enhances security and allows the backend to efficiently handle multiple concurrent users.

2.3.4 Rate Limiting

Application-level rate limiting is implemented using `django-axes`, a Django library that provides two complementary layers of protection. First, it monitors and controls failed login attempts, enforcing configurable lockout policies to protect authentication endpoints against brute-force and credential stuffing attacks. Second, it is configured to apply request rate limiting across API endpoints, restricting the number of requests a given client can make within a defined time window. Both layers operate at the application level, providing consistent protection regardless of the deployment infrastructure.

Chapter 3: Database Design

3.1 Database Overview

The database is a core component of the backend system, responsible for storing and managing all data related to projects, diagram components, and their spatial and logical relationships. In the context of a web-based Process Flow Diagram (PFD) and Process & Instrumentation Diagram (P&ID) application, the database ensures that diagram data can be persistently stored, retrieved, and modified efficiently.

The backend system uses PostgreSQL as its relational database, managed through Django's Object-Relational Mapping (ORM). Django ORM provides an abstraction layer that allows database tables to be represented as Python classes, ensuring data integrity and simplifying database operations. PostgreSQL was chosen over SQLite to provide the performance, scalability, and concurrency support required for a production-grade web application.

Another important consideration in the database design is data consistency and integrity. Constraints such as foreign keys, unique fields, and validation rules ensure that invalid or inconsistent data cannot be stored. Transactions are used where necessary to guarantee atomic operations, especially in cases where multiple related records must be created or updated together. This is particularly important in diagram management, where components, connections, and canvas states must remain synchronized.

3.2 Database Migration to PostgreSQL

The database has been migrated from the default SQLite configuration to PostgreSQL. This migration was carried out to address the limitations of SQLite in a multi-user, production environment. The key steps in the migration process included:

8. Configuring the Django project settings to use the psycopg2 database adapter for PostgreSQL.
9. Creating a dedicated PostgreSQL database and user with appropriate permissions.
10. Applying all existing Django migrations to the new PostgreSQL database to recreate the schema.
11. Exporting existing data from SQLite and importing it into PostgreSQL to ensure data continuity.
12. Verifying data integrity and testing all API endpoints against the new database.

The migration ensures that the application is now backed by a production-ready database engine capable of supporting higher user loads, concurrent requests, and advanced query optimizations.

3.3 Database Structure

The database structure is designed to reflect the logical entities required for PFD and P&ID diagram creation and management. The key database tables include the User table, Project table, Component table, as well as supporting tables for canvas state and connections.

3.3.1 User Table

The User table is provided by Django's built-in authentication system (auth.User). It stores essential user-related information such as usernames, passwords, and authentication details. This table is used to manage system access and to associate projects with specific users. Using Django's built-in User model ensures secure handling of authentication data and reduces the need for custom user management logic.

3.3.2 Project Table

The Project table stores information related to individual PFD or P&ID projects created by users. Each project includes attributes such as a project name, description, thumbnail image, and timestamps indicating when the project was created and last updated. Each project is associated with a user through a foreign key relationship, allowing multiple projects to be created and managed by a single authenticated user. This structure supports organized project management and enables user-specific data access control.

3.3.3 Component Table

The Component table represents the standardized symbol library used for PFD and P&ID diagram creation. Each component stores metadata such as a unique serial number, component name, legend, suffix, and object type. Additionally, each component includes scalable vector graphics (SVG) and image (PNG) files used for diagram rendering. The table also stores grip information using a JSON field, allowing flexible definition of connection points for each component. This design supports reusable, standardized components that comply with engineering diagram conventions.

3.3.4 Canvas State Table

The CanvasState table stores the placement and transformation details of components within a specific project canvas. Each record represents an instance of a component placed on the drawing canvas, including its position, dimensions, rotation, scaling, and sequence order. CanvasState links a project with a component, enabling the system to reconstruct the complete diagram layout for a given project. Connection information is partially stored using JSON fields, allowing flexible representation of component relationships.

3.3.5 Connection Table

The Connection table stores explicit connectivity information between components placed on the canvas. It defines the source and target canvas items, grip indices, and intermediate waypoints for drawing connecting lines. This table supports validation of diagram connectivity and ensures that logical and visual connections between process elements are preserved when diagrams are saved and reloaded.

3.4 Relationships Between Tables

The database design establishes clear relationships between tables to ensure data consistency and support diagram functionality.

3.4.1 User–Project Relationship

A one-to-many relationship exists between users and projects. Each user can create and manage multiple projects, while each project is associated with a single user. This relationship is implemented using a foreign key in the Project table that references the User

table. This design allows user-specific access control and ensures that project data is securely separated between users.

3.4.2 Project–Component Relationship

The relationship between projects and components is managed through the CanvasState table. A project can contain multiple components placed on its canvas, and each component can be reused across multiple projects. This many-to-many relationship is resolved by associating both Project and Component entities within the CanvasState table. Additionally, connections between components within a project are managed through the Connection table, which references canvas state entries. This layered relationship design enables flexible diagram creation, modification, and validation.

Chapter 4: Authentication Module

The Authentication Module is responsible for verifying users and securing access to the system. It ensures that only authorized users can access protected resources by using a combination of user credentials and token-based authentication. This chapter describes the implementation of user registration, user login, and token management.

4.1 User Registration

The user registration feature allows new users to create an account in the system. This process collects necessary user information and securely stores it in the database.

4.1.1 Registration Process

13. The user provides required details such as username and password through the registration interface.
14. The system validates the input data to ensure correctness and completeness.
15. The password is encrypted before being stored in the database to maintain security.
16. Once validation is successful, a new user record is created in the database.
17. The system returns a success response indicating that the registration process has been completed successfully.

This process ensures that user data is securely handled and prevents invalid or duplicate registrations.

4.2 User Login

The user login feature allows registered users to authenticate themselves and gain access to the system.

4.2.1 Login Process

18. The user enters their registered email/username and password.
19. The system verifies the credentials by comparing them with stored user data.
20. django-axes checks whether the requesting IP or username has exceeded the allowed number of failed attempts before proceeding.
21. If the credentials are valid, the system authenticates the user and resets the failure counter for that account.
22. Upon successful authentication, the system generates authentication tokens for the user.
23. The user is then granted access to protected features of the application.

If the credentials are invalid, the system increments the failure counter tracked by django-axes and returns an error message indicating authentication failure. If the configured threshold is exceeded, the account or IP is temporarily locked to prevent further attempts.

4.3 Token Management

Token management is used to maintain secure user sessions without requiring the user to log in repeatedly. The system uses access tokens and refresh tokens to manage authentication efficiently.

In addition to improving user experience, token-based authentication also enhances scalability by eliminating the need for server-side session storage. Since the server does not need to maintain session state, it can handle a larger number of concurrent users more efficiently. Proper token expiration and refresh strategies are implemented to balance security and usability, ensuring that unauthorized access is minimized while maintaining seamless interaction for legitimate users.

4.3.1 Access Token

- The access token is a short-lived token used to access protected APIs.
- It is sent with each authenticated request to verify the user's identity.
- Due to its limited lifespan, it reduces security risks if compromised.

4.3.2 Refresh Token

- The refresh token is used to obtain a new access token when the current one expires.
- It has a longer lifespan compared to the access token.
- This mechanism improves user experience by preventing frequent logins while maintaining security.

Chapter 5: Rate Limiting with django-axes

5.1 Overview

To protect the application against brute-force attacks, unauthorized access attempts, and API abuse, rate limiting has been implemented at the application level using django-axes. django-axes provides two complementary layers of protection: login attempt monitoring with automatic lockout, and request rate limiting across API endpoints. Both layers are configured within the Django application and operate independently of the underlying web server or deployment infrastructure.

Implementing rate limiting at the application level ensures that protection is consistent across all environments and allows the behavior to be configured, tested, and maintained as part of the application codebase.

Implementing rate limiting is a critical aspect of modern web application security. Without such mechanisms, systems are vulnerable to automated attacks that can compromise user accounts or degrade system performance. By integrating django-axes directly into the application layer, the system ensures that security policies are enforced consistently across all environments, including development, testing, and production.

5.2 Login Attempt Protection

django-axes integrates with Django's authentication system by listening to login failure signals. When a user submits incorrect credentials, django-axes records the attempt in the database, tracking the username, IP address, and timestamp of the failure. When the number of failures from a given source exceeds a configurable threshold, django-axes blocks further login attempts from that source for a defined cooldown period.

The library hooks into Django's authentication backend, making it transparent to the rest of the application. No changes to individual API views are required; protection is applied consistently across all login endpoints. The following policies are enforced for login protection:

- Failed login attempts are tracked per IP address and per username.
- After a configurable number of consecutive failures, the account or IP is temporarily locked.
- Lockouts are stored in the database and automatically released after the cooldown period expires.
- Administrators can view and manage lockout records through the Django admin interface.
- Successful logins reset the failure counter for the corresponding account.

5.3 API Rate Limiting

In addition to login protection, django-axes is configured to enforce rate limits on API endpoints. This prevents clients from making an excessive number of requests within a short

time window, protecting the application from automated scraping, denial-of-service attempts, and API abuse.

The API rate limiting is applied at the application middleware level, meaning it is evaluated for every incoming request before it reaches the view logic. Clients that exceed the configured request threshold receive an HTTP 429 Too Many Requests response, and their access is temporarily restricted until the rate limit window resets. Key aspects of the configuration include:

- Rate limits are defined per IP address to prevent abuse from individual clients.
- Different thresholds can be configured for different endpoint categories, such as authentication endpoints versus data retrieval endpoints.
- Rate limit violations are logged, providing visibility into potential abuse patterns.
- The cooldown window is configurable, allowing the policy to be tuned based on observed traffic patterns.

5.4 Integration with the Authentication Flow

django-axes is registered as a Django authentication backend in the project settings alongside the default authentication backend. This ensures that every authentication attempt passes through the axes check before credentials are verified. The middleware component handles API-level rate limiting for all other endpoints. The integration requires no additional code in individual views or API endpoints, keeping the implementation clean and maintainable.

The django-axes database tables are managed through standard Django migrations and are included in the PostgreSQL database used by the application.

Chapter 6: Admin Data Import

The Admin Data Import module is designed to allow administrators to efficiently upload and manage component data in bulk. Instead of manually adding individual records, the system supports importing structured data using a ZIP file, ensuring consistency, accuracy, and reduced administrative effort.

6.1 Import Feature Overview

The import feature enables administrators to upload a ZIP file containing component data and related resources. The system processes the uploaded file, validates its structure, and stores the extracted data in the database.

This feature is particularly useful in scenarios where large sets of standardized components need to be imported into the system, such as when updating symbol libraries or integrating external datasets. By automating the import process, the system reduces the risk of human error and ensures that all components adhere to a consistent structure and format.

6.1.1 Purpose of ZIP Import

The ZIP import mechanism is implemented to achieve the following objectives:

- To simplify bulk data entry for components.
- To ensure standardized data structure for components and related assets.
- To reduce manual errors by automating the import process.
- To allow easy updates of existing components along with their associated files.

By using a ZIP file, the system can handle multiple data files and resources in a single upload operation.

6.2 Import Process

The import process follows a structured sequence of steps to ensure data integrity and proper validation.

6.2.1 Upload ZIP File

24. The administrator uploads a ZIP file through the admin interface.
25. The system validates whether the uploaded file is a valid ZIP file.
26. If the file is invalid or missing, an error message is displayed to the administrator.
27. Upon successful validation, the ZIP file is temporarily stored and prepared for extraction.

This step ensures that only properly formatted ZIP files are processed by the system.

6.2.2 Extract and Store Data

28. The system extracts the ZIP file into a temporary directory.
29. It verifies the presence of a required components folder inside the ZIP file.

30. Inside the components folder, the system checks for a CSV file containing component metadata and required subfolders such as svg and png for component assets.
31. The CSV file is read row by row, and each row is validated for mandatory fields such as component name and serial number.
32. For each component, if a component with the same serial number already exists, the existing record is updated; otherwise, a new component is created.
33. Associated SVG and PNG files are stored and linked to the corresponding component.
34. Any existing files are replaced to ensure updated assets are reflected correctly.
35. The system tracks the number of successfully processed, created, and updated components.
36. Upon completion, a success message is displayed to the administrator summarizing the import results.

Chapter 7: API Implementation

This chapter describes the implementation of the RESTful APIs developed for the system. The APIs are built using Django REST Framework (DRF) and follow standard REST principles. They provide secure access to system resources, including authentication, project management, and component management.

7.1 API Design Principles

The APIs developed in this system follow standard RESTful design principles to ensure consistency, scalability, and ease of use. Each API endpoint is designed around clear resource-based URLs and uses appropriate HTTP methods such as GET, POST, PUT, and DELETE to perform operations.

Error handling is standardized across all endpoints, with meaningful HTTP status codes and descriptive error messages returned to the client. Authentication is enforced for all protected routes using JWT tokens, and input validation is performed to prevent invalid or malicious data from being processed.

This structured approach ensures that the APIs are easy to integrate, maintain, and extend as the system evolves.

7.2 Authentication APIs

Authentication APIs are responsible for user identity verification and secure access to protected endpoints. The system uses JWT (JSON Web Token)–based authentication to manage user sessions.

7.2.1 Register API

The registration API allows new users to create an account in the system.

- Accepts user details such as username, email, and password.
- Validates input data before processing.
- Stores user credentials securely in the database.
- Returns a confirmation response upon successful registration.

This API ensures that only valid users are added to the system.

7.2.2 Login API

The login API authenticates registered users.

- Verifies user credentials against stored data.
- Passes the authentication attempt through the django-axes rate limiting check.
- Generates an access token and a refresh token upon successful authentication.
- Returns tokens to the client for subsequent authorized requests.

The access token is required to access protected APIs within the system.

7.2.3 Refresh Token API

The refresh token API is used to maintain user sessions securely.

- Accepts a valid refresh token from the client.
- Generates a new access token when the previous one expires.
- Prevents frequent re-login while maintaining security.

This approach improves usability without compromising system security.

7.3 Project APIs

Project APIs manage the lifecycle of projects within the system. These APIs allow users to create, view, update, and delete projects.

7.3.1 Create Project API

- Allows authenticated users to create new projects.
- Accepts project details such as name and description.
- Associates the project with the authenticated user.
- Returns the created project details as a response.

7.3.2 Update Project API

- Enables users to modify existing project information.
- Supports updating project metadata and canvas state.
- Ensures that only authorized users can update their own projects.

7.3.3 Delete Project API

- Allows users to delete a project permanently.
- Validates project ownership before deletion.
- Returns a success or error message based on the operation result.

7.3.4 Fetch Projects API

- Retrieves a list of all projects associated with the authenticated user.
- Returns project metadata in a structured JSON format.
- Helps users manage and navigate multiple projects efficiently.

7.3.5 Fetch Project Details API

- Fetches detailed information about a specific project.
- Includes project metadata, components, and canvas state.
- Returns related items and connections used in the diagram editor.

This API plays a key role in loading and restoring project workspaces.

7.4 Component APIs

Component APIs are used to manage reusable components required for building chemical process flow diagrams.

7.4.1 Create Component API

- Allows administrators or authorized users to create new components.
- Accepts component metadata along with SVG and PNG assets.
- Stores component data and associated files in the system.

7.4.2 Fetch Components API

- Retrieves a list of available components.
- Returns component details such as name, type, and media references.
- Supports rendering components dynamically in the client application.

This API ensures that all diagram components are centrally managed and consistently available.

Chapter 8: Current Progress

This chapter summarizes the current development progress of the system and highlights the major features that have been successfully implemented. It also discusses the technical challenges encountered during development and how they were addressed.

8.1 Completed Features

At the current stage of development, several core features of the system have been completed and are functioning as intended.

8.1.1 Database Setup and PostgreSQL Migration

- The database schema has been designed and implemented using Django's ORM.
- Models have been created to support users, projects, components, and related entities.
- The database has been successfully migrated from SQLite to PostgreSQL.
- All Django migrations have been applied to the PostgreSQL instance, and data integrity has been verified.
- The application is now configured to use psycopg2 as the PostgreSQL database adapter.

This setup provides a stable and production-ready foundation for all other system functionalities.

8.1.2 Authentication System

- A complete authentication system has been implemented using JWT.
- Features include user registration, login, and token refresh mechanisms.
- Secure access control has been enforced for protected API endpoints.

This ensures that only authenticated users can access system resources.

8.1.3 Rate Limiting with django-axes

- django-axes has been integrated at the application level to provide both login brute-force protection and API request rate limiting.
- Failed login attempts are tracked by IP address and username in the PostgreSQL database, with automatic logout after the configured failure threshold.
- API rate limiting middleware restricts excessive requests from individual clients, returning HTTP 429 responses when thresholds are exceeded.
- Administrators can view and manage logout records via the Django admin interface.

This feature significantly improves the security posture of the application.

8.1.4 Admin Import Function

- An admin-only ZIP import feature has been developed for bulk component uploads.
- The system validates ZIP structure, extracts CSV data, and processes SVG and PNG files.
- Existing components are updated, and new components are created automatically.

This feature significantly reduces manual administrative work.

8.1.5 Project and Component APIs

- RESTful APIs have been implemented for project creation, updating, deletion, and retrieval.
- Component APIs support creating and fetching reusable diagram components.
- APIs are secured and integrated with authentication middleware.

These APIs enable seamless communication between the frontend and backend systems.

8.2 Challenges Faced

During development, several technical challenges were encountered. Addressing these challenges contributed significantly to the learning outcomes of this internship. It provided hands-on experience in debugging complex issues, understanding production-level system design, and applying best practices in backend development. These experiences helped strengthen both technical skills and problem-solving abilities.

8.2.1 Technical Issues Encountered

- Migrating from SQLite to PostgreSQL required careful handling of data types and constraints, particularly for JSON fields and file path references stored in the database.
- Configuring django-axes alongside JWT authentication required ensuring that the axes authentication backend was correctly ordered in Django settings to intercept login attempts before token issuance.
- Handling large ZIP files and validating their internal structure required careful error handling.
- Managing JWT token expiration and refresh logic required precise implementation to avoid authentication failures.
- Ensuring consistency between project canvas data and database records was challenging due to nested data structures.

These challenges were addressed through testing, code refactoring, and the use of Django REST Framework and django-axes best practices.

Chapter 9: Future Work

9.1 Planned Improvements

9.1.1 Deployment to a Cloud Server

The current system has been developed and tested in a local development environment. As the next phase of the project, the backend will be deployed to a live cloud server to make the application accessible over the internet. The planned deployment approach involves provisioning a virtual machine or managed compute instance on a platform such as AWS, Google Cloud Platform, or DigitalOcean, configured as follows:

- A production-grade web server setup using Nginx as a reverse proxy, with Gunicorn serving the Django application.
- Environment variable management to securely store sensitive configuration values such as database credentials, secret keys, and JWT settings, keeping them out of source control.
- HTTPS configuration using SSL/TLS certificates via Let's Encrypt to ensure encrypted communication between clients and the server.
- Connection to a production PostgreSQL instance with automated backups and point-in-time recovery enabled.
- Static and media file serving configured appropriately for the production environment.

Deploying to a live server will allow the frontend team to integrate with the backend APIs in a shared environment and enable end-to-end testing of the complete application.

9.1.2 CI/CD Pipeline with GitHub Actions

To streamline the development workflow and ensure consistent, reliable deployments, a Continuous Integration and Continuous Deployment (CI/CD) pipeline will be established using GitHub Actions. The planned pipeline will include the following stages:

- Continuous Integration: Automated test runs on every push or pull request to the main branch to verify that new changes do not break existing functionality.
- Code Quality Checks: Linting with flake8 and static analysis as part of the CI pipeline to maintain code standards across the project.
- Automated Deployment: Deployment jobs that trigger on successful merges to the main branch, pushing updates to the live cloud server and running database migrations without manual intervention.
- Environment Separation: Separate workflow configurations for staging and production environments to validate changes before promoting them to production.

Establishing a CI/CD pipeline will reduce deployment overhead, minimize the risk of human error during releases, and support faster iteration on new features and bug fixes.

Chapter 10: Conclusion

10.1 Summary

This project successfully developed a complete backend system for a Chemical Process Flow Diagram web application using Django and Django REST Framework. The system includes secure authentication using JWT tokens, robust project and component management APIs, an efficient admin data import mechanism, rate limiting and brute-force protection via django-axes, and a production-grade PostgreSQL database.

The migration from SQLite to PostgreSQL represents a significant improvement in the system's scalability and readiness for production deployment. The integration of django-axes at the application level ensures that both the authentication endpoints and the broader API surface are protected against automated attacks, contributing to the overall security posture of the application.

The implementation demonstrates effective use of RESTful architecture, JWT-based authentication, modular design principles, and production-oriented engineering practices. The current system provides a strong foundation for the next phases of development, including live server deployment, CI/CD automation, and full integration with the frontend diagramming interface.

In addition to technical achievements, this project also provided valuable exposure to collaborative development practices, including working in a team environment, participating in code reviews, and following structured development workflows. The experience gained through this internship will be highly beneficial for future work in software engineering and system design.

Overall, the project meets its intended objectives and establishes a scalable, secure, and maintainable platform for managing chemical process flow diagrams.