# Software Design Specifications

## *Identity and Access Management System*

**Version: 1.7**

| | |
|---|---|
| *Project Code* | *CS3009* |
| *Supervisor* | *Ms. Sobia Iftikhar* |
| *Co Supervisor* | *Mr. Zeeshan Mustafa* |
| *Project Team* | *Hafiz farhad*<br>*Sohaib Ali* |
| *Submission Date* | *08/05/2025* |

# Document History

| Version | Name of Person | Date | Description of change |
|---|---|---|---|
| 1.1 | Hafiz farhad | 30/04/2025 | Introduction |
| 1.2 | Sohaib Ali | 01/05/2025 | Design Consideration |
| 1.3 | Hafiz Farhad | 02/05/2025 | System Architecture |
| 1.4 | Sohaib Ali | 02/05/2025 | Design Strategy |
| 1.5 | Sohaib Ali | 03/05/2025 | Detailed System Design |
| 1.6 | Hafiz Farhad | 04/05/2025 | References |
| 1.7 | Sohaib Ali | 05/05/2025 | Appendices |

# Distribution List

| Name | Role |
|---|---|
| Ms. Sobia Iftikhar | Supervisor |
| Mr. Zeeshan Mustafa | Co Supervisor |

# Document Sign-Off

| Version | Sign-off Authority | Project Role | Signature | Sign-off Date |
|---|---|---|---|---|
| 1.1 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.2 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.3 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.4 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.5 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.6 | Ms. Sobia Iftikhar | Supervisor | | |
| 1.7 | Ms. Sobia Iftikhar | Supervisor | | |

# Document Information

| Category | Information |
|---|---|
| Customer | FAST-NU |
| Project | Identity and Access management System |
| Document | Software Design Specification |
| Document Version | 1.0 |
| Status | Final Document |
| Author(s) | Hafiz farhad, Sohaib Ali |
| Approver(s) | Ms. Sobia Iftikhar |
| Issue Date | 29/04/2025 |
| Document Location | - |
| Distribution | Supervisor, Teacher’s Assistant |

# Definition of Terms, Acronyms and Abbreviations

| Term | Description |
|---|---|
| IAM | Identity and Access Management |
| JWT | JSON Web Token |
| API | Application Programming Interface |
| RBAC | Role-based Access Control |
| UI | User Interface |
| DB | Database |
| LDAP | Lightweight Directory access Protocol |

# Table of Contents

# 1 Introduction

## *1.1 Purpose of Document*

*This Software Design Specification (SDS) document outlines the design approach for Ixion, an Identity and Access Management (IAM) system developed using Laravel and Vue.js. The document serves as a comprehensive reference for software engineers, project managers, and stakeholders involved in the development, testing, and maintenance of the system. The design follows an object-oriented methodology to ensure modularity, maintainability, and extensibility in the system's architecture.*

## *1.2 Intended Audience*

*This document is intended for project supervisors, co-supervisors, developers, testers, and future maintainers of the Ixion IAM system. It provides sufficient technical detail for implementation and serves as a guide to understanding the design decisions and structural layout of the system.*

## *1.3 Document Convention*

*The document is written using Arial, font size 10, with section headings in bold and of 12pts. Diagrams are included using standardized UML notations.*

## *1.4 Project Overview*

*Ixion is a robust Identity and Access Management solution built to manage authentication, authorization, organization hierarchy, and secure API interactions across client applications. The system employs Laravel Passport to implement OAuth2 token-based authentication and features a modern Vue.js front-end for managing users, roles, permissions, and organizations. Ixion acts as a central identity service capable of integration with other platforms requiring secure and scalable access control.*

## *1.5 Scope*

*The Ixion IAM system allows users to log in, manage multiple organizations, define roles and permissions, and secure API access through OAuth2 tokens. It supports administrators in assigning roles and managing access control within organizational boundaries. However, it does not provide third-party login options (e.g., Google OAuth) out of the box, and is limited to web-based access through REST APIs.*

# 2 Design Considerations

## 2.1 Assumptions and Dependencies

*The design assumes that the underlying Laravel framework and Laravel Passport are properly configured and operational. Vue.js is assumed to be used as the front-end framework, and MySQL or PostgreSQL as the database system. Internet connectivity is assumed for API communication.*

## 2.2 Risks and Volatile Areas

*Potential risks include changes in authentication protocols, such as updates to OAuth2 specifications, or shifts in frontend/backend frameworks that may necessitate redesign. Another volatile area is the dynamic nature of organizational requirements, which may demand flexible role and permission models. The system is designed to be modular, allowing for easy adaptation to such changes.*

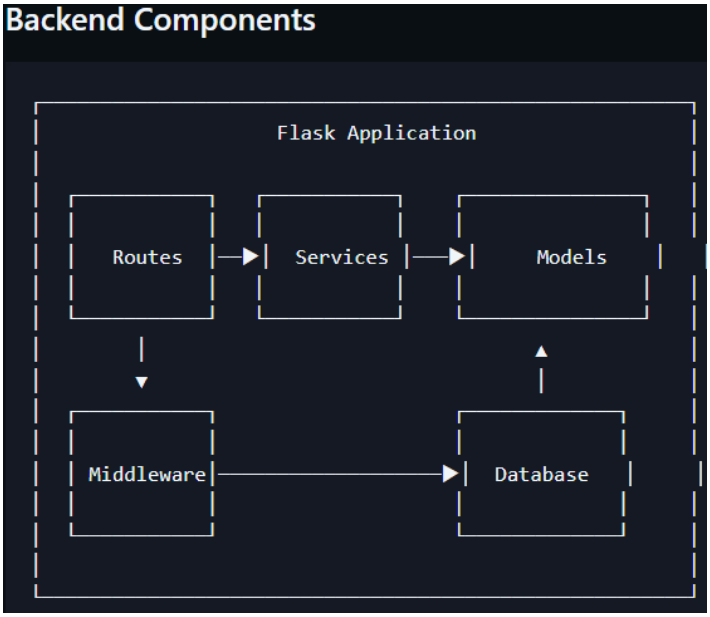
# 3 System Architecture

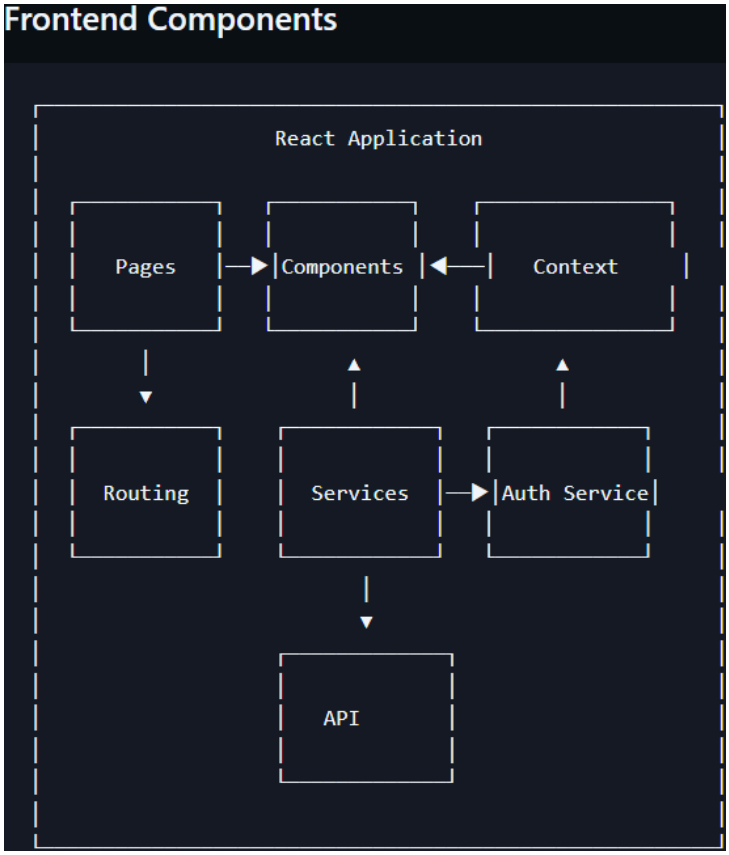
## 3.1 System Level Architecture

*The Ixion system is composed of several major components. The front-end Vue.js application interacts with the back-end Laravel API through HTTP requests. The Laravel API serves as the middle tier, handling business logic, authorization policies, and user sessions. The backend communicates with the database through an ORM layer to perform data persistence. The Laravel Passport module acts as an embedded OAuth2 server, issuing and managing access tokens for clients.*

## 3.2 Software Architecture

*The software architecture is divided into three main layers. The **User Interface Layer** consists of Vue.js components responsible for interacting with users. The **Middle Tier** includes Laravel controllers and service classes that handle application logic. The **Data Access Layer** involves Laravel's Eloquent ORM, managing communication with the relational database. The system uses Laravel policies and middleware to enforce access control between these layers.*


Backend Components
Flask Application
Routes
Services
Models
Middleware
Database


Frontend Components
React Application
Pages
Components
Context
Routing
Services
Auth Service
API

# 4 Design Strategy

*The design of Ixion emphasizes modularity and reusability. Components such as controllers, services, and policies are designed to be extensible to accommodate future features such as SSO or external identity provider integrations. The role-based access control model allows fine-grained authorization that can scale with new roles or resource types. The user interface adheres to a single-page application model for responsive and seamless user experiences. Persistent data storage is handled using MySQL/PostgreSQL, with the Laravel ORM facilitating database interaction. Concurrency is managed at the API level through Laravel's queue and job system, while token management ensures secure multi-session access.*

# 5 Detailed System Design

## *5.1 Database Design*

### 5.1.1 ER Diagram

*The **ER diagram** represents a structured view of how different entities in the Ixion system are related to one another. Below is a breakdown of the core entities, their attributes, and the relationships between them:*

#### *1. Users*

*The `Users` table stores core user information. Each user has a unique `id`, along with attributes like `email`, `password`, `first_name`, `last_name`, `is_active`, `is_admin`, `created_at`, `updated_at`, and `last_login`.*

*This entity is **central to the system** as it links to multiple other entities, such as roles (via a mapping table), invitations, and audit logs.*

#### *2. Roles and User Roles (Many-to-Many)*

*Users can have multiple roles, and roles can be assigned to multiple users. This **many-to-many relationship** is modeled using the `User Roles` table, which contains `user_id` and `role_id`. This approach ensures scalability in assigning flexible roles across users.*

*The `Roles` entity defines each role with an `id`, `name`, `description`, and a `system_role` flag (indicating if it's a default/system-level role). These roles define what actions users are allowed to perform in the system.*

#### *3. Role Permissions (Many-to-Many)*

*Roles do not have functionality by themselves—they are connected to actual system actions through permissions. The **Role Permissions** table forms a many-to-many relationship between `Roles` and `Permissions`, mapping `role_id` to `permission_id`.This structure allows fine-grained control over access and enforces **Role-Based Access Control (RBAC)**.*

#### *4. Permissions*

*The `Permissions` table defines what actions can be performed and on what resources. It includes fields like `id`, `name`, `description`, `resource`, and `action`. For example, a permission could define access*

*like `edit:user_profile` or `delete:organization`.Permissions represent **atomic access rights**, and roles are essentially collections of these rights*

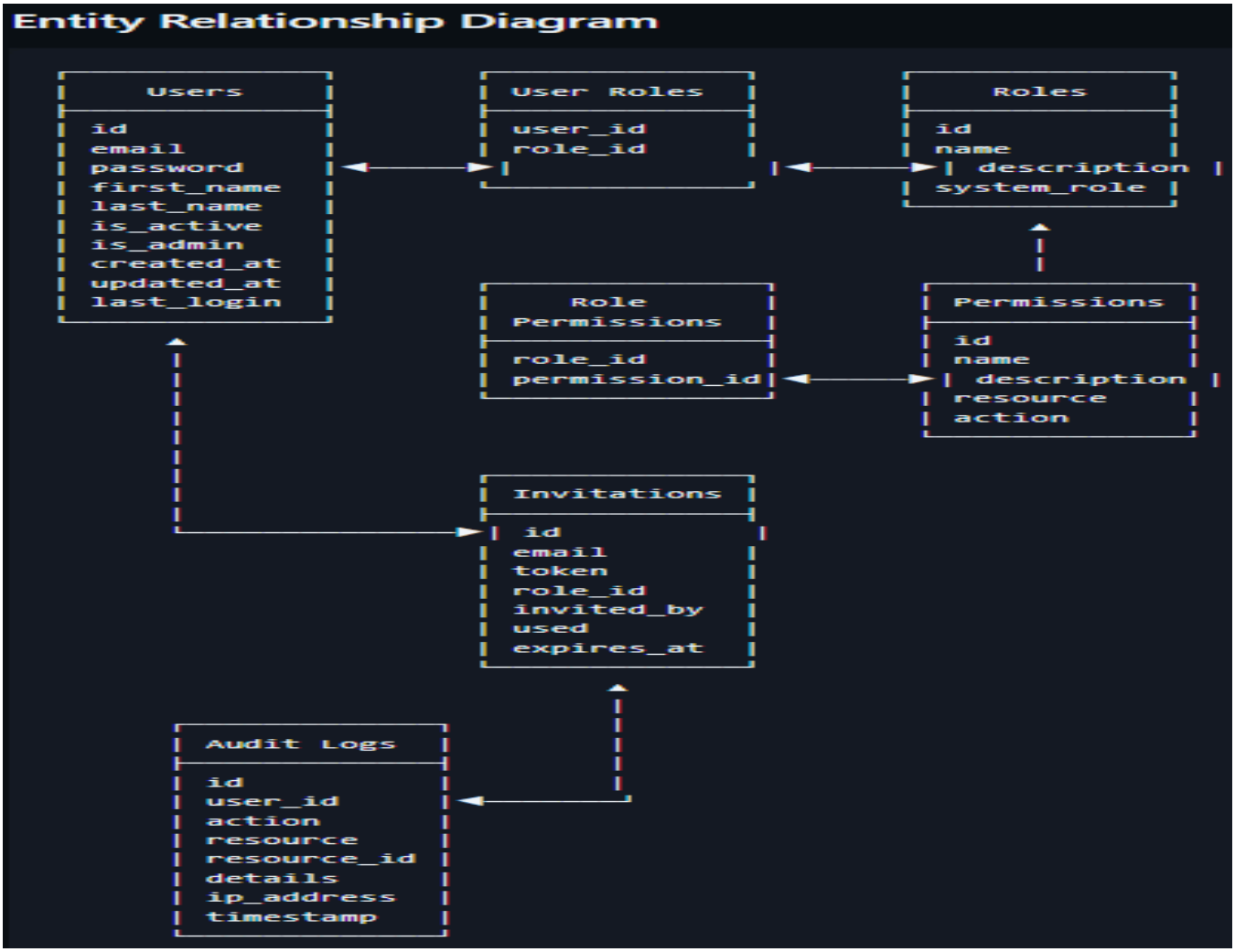
## *5. Invitations*

*The `Invitations` entity allows admins or users with adequate rights to invite new users to the system. Each invitation has fields such as `id`, `email`, `token`, `role_id`, `invited_by` (user who sent the invite), `used` (flag to indicate usage), and `expires_at`.It links back to the `Users` entity through `invited_by` and references a `role_id` that will be assigned upon registration.*

## *6. Audit Logs*

*The `Audit Logs` entity tracks user activity within the system. Each record includes fields such as `id`, `user_id`, `action`, `resource`, `resource_id`, `details`, `ip_address`, and `timestamp`.*

*This entity is used for **security auditing and accountability**, linking directly to the `Users` table to track who did what, where, and when.*



## 5.1.2 Data Dictionary

### 5.1.2.1 Data 1

### *Users Table*

*The **Users** table stores core information about each registered user in the system. Each user is uniquely identified by the **id (PK)**, which serves as the primary key. The **email** field holds the user's email address and is used as the unique login identifier. To secure user authentication, passwords are stored in the **password** field after hashing. Users' personal identity is captured through the **first_name** and **last_name** fields. The **is_active** boolean field indicates whether a user account is currently active, while the **is_admin** field specifies if the user has administrative rights within the system. The **created_at** and **updated_at** fields record the timestamps of when the user account was created and last modified, respectively. The **last_login** field logs the date and time of the user's most recent successful login.*

### 5.1.2.2 Data 2

*The **User Roles** table defines a many-to-many relationship between users and roles. It includes **user_id (FK)**, which links to the **id** field in the Users table, and **role_id (FK)**, which references the **id** in the Roles table. This table allows a single user to be assigned multiple roles, and a single role to be assigned to multiple users.*

### 5.1.2.3 Data 3

*The **Roles** table defines the different roles that can be assigned to users within the system. Each role has a unique **id (PK)** that serves as the primary key. The **name** field specifies the role's name, such as "admin" or "editor". A textual explanation of what the role entails is stored in the **description** field. The **system_role** field is a boolean value used to identify if a role is predefined by the system and should not be modified or deleted by regular users.*

### 5.1.2.4 Data 4

The **Role Permissions** table manages the many-to-many relationship between roles and permissions. It includes **role_id (FK)**, which references the **id** in the Roles table, and **permission_id (FK)**, which references the **id** in the Permissions table. This structure allows multiple permissions to be assigned to a single role and a single permission to be associated with multiple roles, supporting granular access control.

| Data 1 | |
|---|---|
| **Name** | Users Roles |
| **Alias** | User Accounts, System Users, IAM Users |
| **Where-used/how-used** | This table is used across all authentication and authorization processes. It serves as the primary data store for user identities. It is used as input for login validation, output for profile displays, and as a reference store for invitations, roles, and audit logging processes. |
| **Content description** | = id + email + password + first_name + last_name + is_active + is_admin |

| Column Name | Description | Type | Length | Null able | Default Value | Key Type |
|---|---|---|---|---|---|---|
| *ID* | *Unique Identifier* | *Integer* | *11* | *No* | *Auto Increment* | *PK* |
| *Email* | *User's email* | *String* | *255* | *No* | *NULL* | *-* |
| *Password* | *Hashed for Authentication* | *string* | *255* | *No* | *NULL* | |
| *First Name* | *First name of the user* | *string* | *100* | *Yes* | *NULL* | |
| *Last Name* | *Last name of the user* | *string* | *100* | *yes* | *NULL* | |
| *Is Active* | *Shows whether the user is active* | *Boolean* | *1* | *No* | *true* | |
| *Is admin* | *Checks whether the user is an admin* | *boolean* | *1* | *No* | *false* | |

| Data 2 | |
|---|---|
| **Name** | Users |
| **Alias** | User Role Mapping |
| **Where-used/how-used** | Implements many-to-many relationship between users and roles. Used in role-checking during access control. |
| **Content description** | user_id + role_id |

| Column Name | Description | Type | Length | Null able | Default Value | Key Type |
|---|---|---|---|---|---|---|
| *ID* | *Unique Identifier* | *Integer* | *11* | *No* | *NULL* | *FK* |
| *Role ID* | *Id assigned to the role* | *Integer* | *11* | *No* | *NULL* | *FK* |

| Data 3 | |
|---|---|
| **Name** | Roles |
| **Alias** | System Roles |
| **Where-used/how-used** | Defines named roles within the system. Referenced in user-role mapping, permissions, and invitations. |
| **Content description** | = id + name + description + system_role |

| Column Name | Description | Type | Length | Null able | Default Value | Key Type |
|---|---|---|---|---|---|---|
| *Role ID* | *Unique Identifier for Role* | *Integer* | *11* | *No* | *NULL* | *PK* |
| *Role Name* | *name of the Role* | *String* | *100* | *No* | *NULL* | |
| *Description* | *Role Description* | *String* | *255* | *Yes* | *Null* | |
| *System Role* | *System Defined Role* | *Boolean* | *1* | *No* | *False* | |

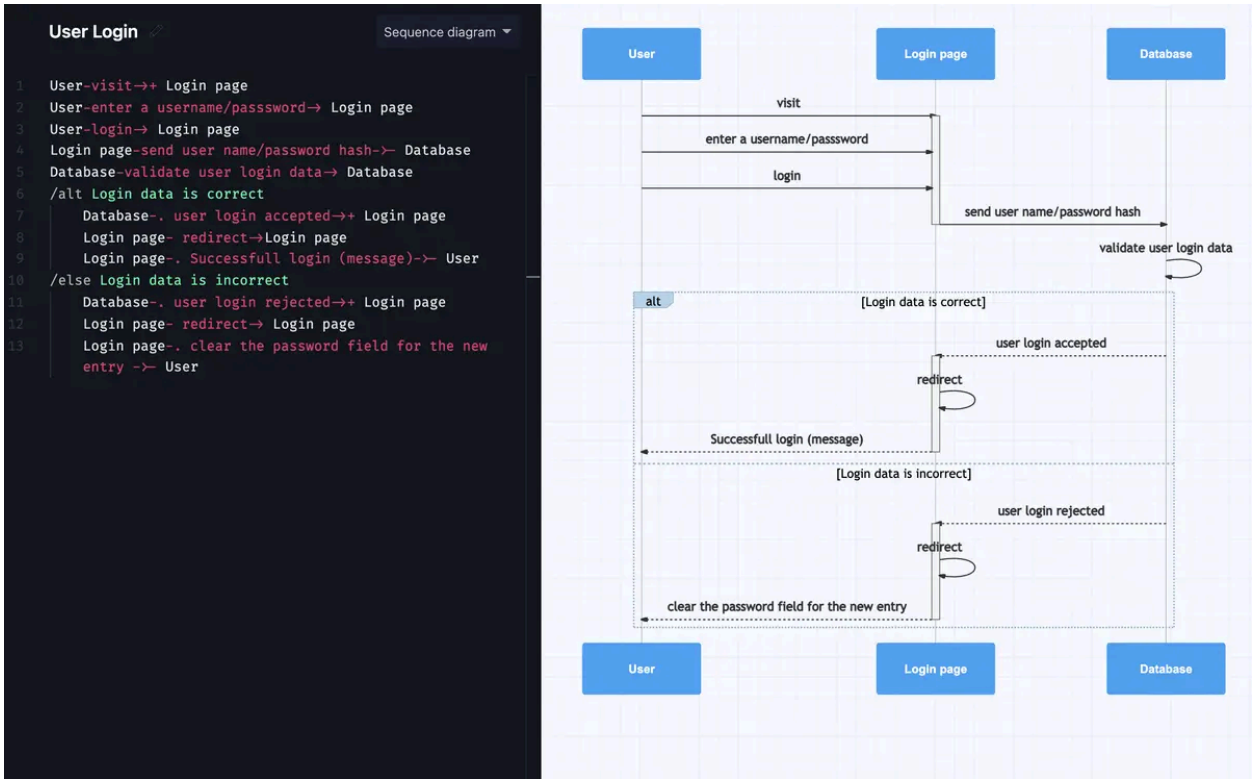
| Data 4 | |
|---|---|
| **Name** | Role permissions |
| **Alias** | Role permissions Mapping |
| **Where-used/how-used** | Defines the many-to-many relationship between roles and permissions. Used in authorization checks. |
| **Content description** | role_id+permission_id |

| **Column Name** | **Description** | **Type** | **Length** | **Null able** | **Default Value** | **Key Type** |
|---|---|---|---|---|---|---|
| *Role ID* | *Unique Identifier* | *Integer* | *11* | *No* | *NULL* | *FK* |
| *Permission ID* | *Id of the Permission* | *Integer* | *11* | *No* | *NULL* | *FK* |

## *5.2 Application Design*

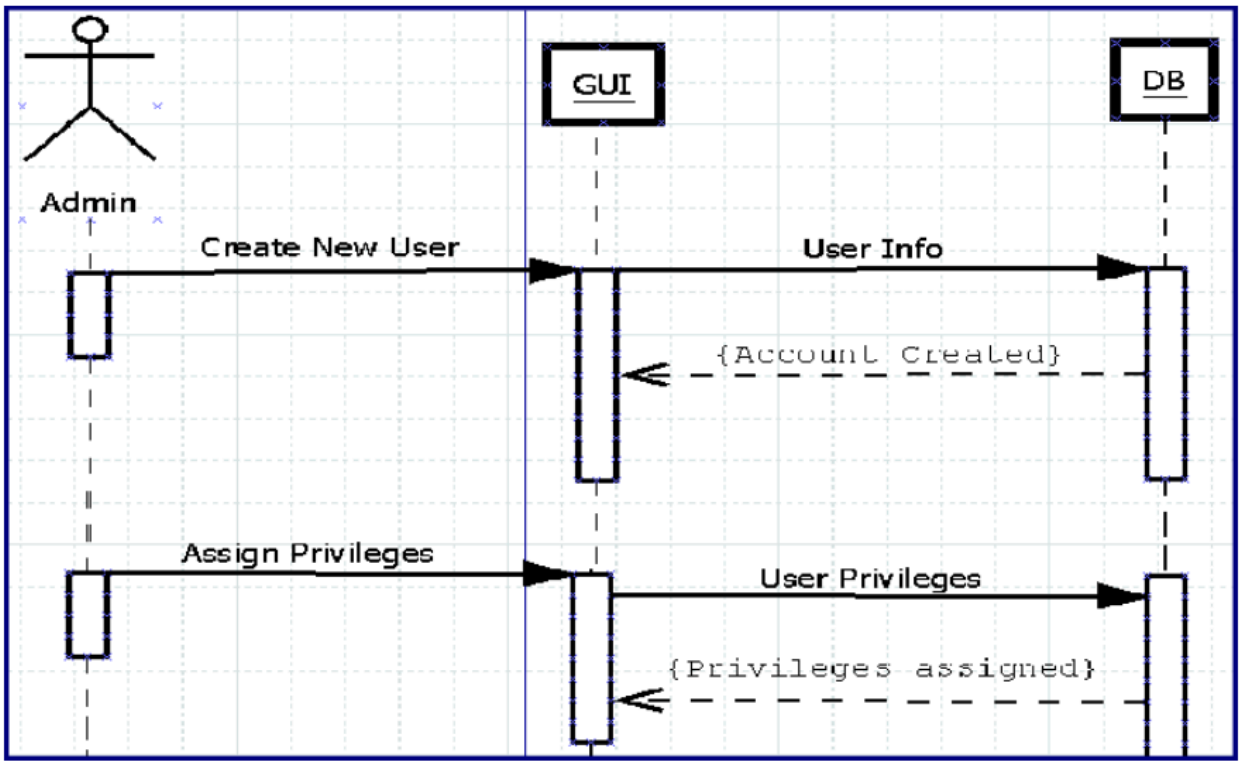
### 5.2.1 Sequence Diagram

#### 5.2.1.1 User Login




*The login sequence begins when a user submits credentials through the front end. The request is passed to the Laravel controller, which validates the input and uses Passport to generate an access token. The token is returned to the front end, which stores it for subsequent API requests. Similar sequences exist for role creation, organization management, and permission assignments.*

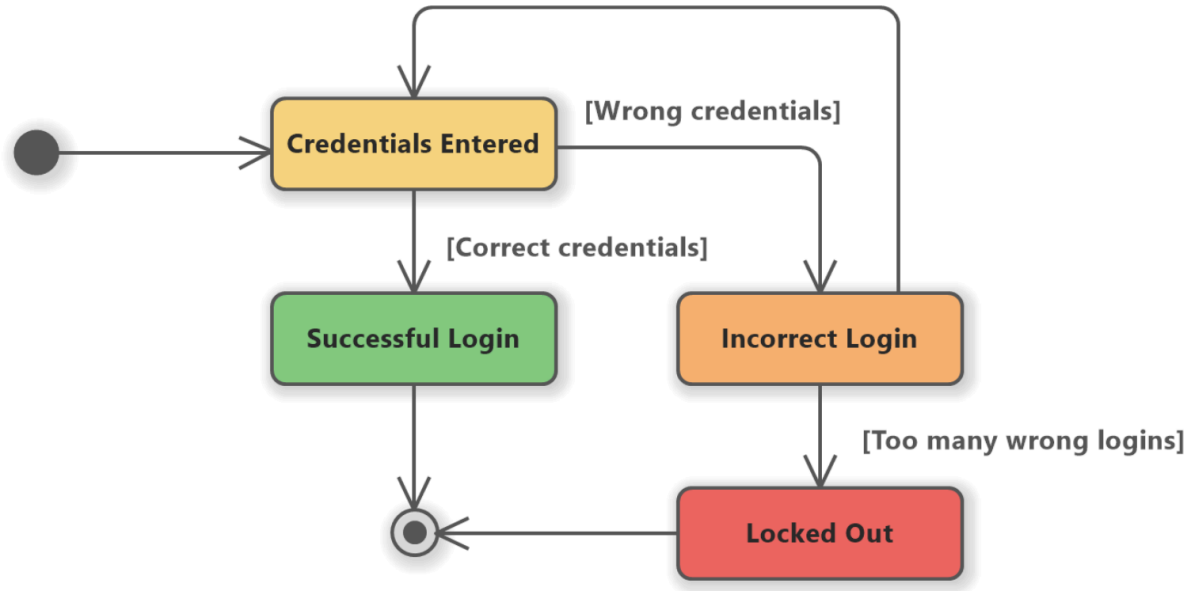
#### 5.2.1.2 Assign Role to User Sequence




*An admin selects a user and a role from the frontend UI. The system checks permissions, then maps the user to the role in the `user_roles` table.*

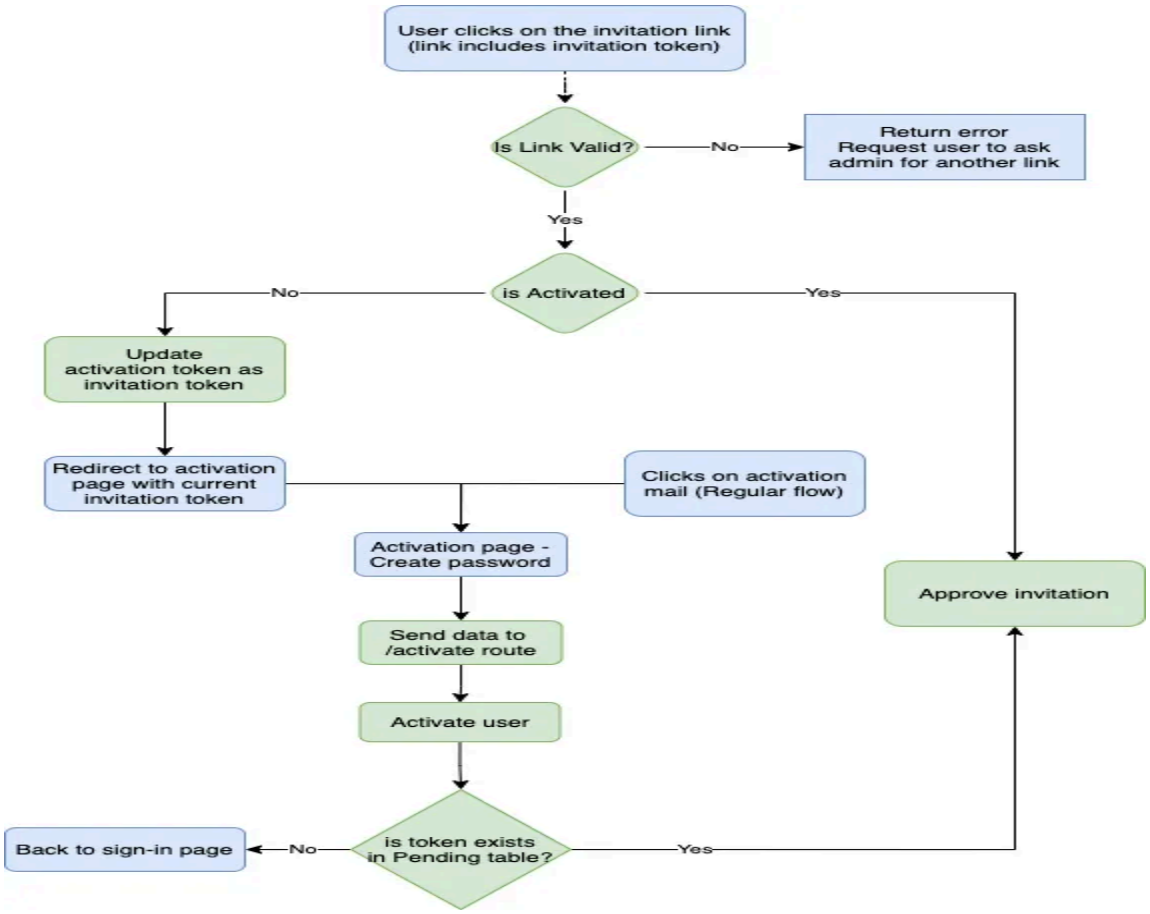
## 5.2.2 State Diagram

### 5.2.2.1 Login System State Diagram




*The initial state in a login process is when the user has not yet entered their login credentials. From this state, the user can transition to the **"credential entered"** state by entering their credentials and submitting the form.If the login credentials are correct, the user will transition to the **"successful login"** state. In this state, the user has successfully logged in to the system and can access the various features and functions that are available to them.However, if the login credentials are incorrect, the user will transition to the **"incorrect login"** state. In this state, the user will be prompted to try again, either by re-entering their login credentials or by resetting their password.If the user continues to enter incorrect login credentials, they may eventually reach the **"locked out"** state.*

### 5.2.2.2 Invitation System State Diagram




*The flow chart starts when the invited user clicks on the confirmation email trying to approve his invitation request. The link is redirected to an invitation page in our frontend site.*

# 6 References


- ***Ixion GitHub Repository – Identity and Access Management System***
  *https://github.com/hafizfarhad/Ixion*
- ***Laravel Documentation (v10.x)***
  *https://laravel.com/docs*
- ***Laravel Passport – OAuth2 Server Documentation***
  *https://laravel.com/docs/10.x/passport*
- ***Vue.js Documentation (v3.x)***
  *https://vuejs.org/guide/introduction.html*
- ***OAuth 2.0 Authorization Framework – RFC 6749***
  *https://datatracker.ietf.org/doc/html/rfc6749*
- ***MySQL 8.0 Reference Manual***
  *https://dev.mysql.com/doc/refman/8.0/en/*
- ***UML 2.5 Specification by OMG***
  *https://www.omg.org/spec/UML/2.5*


# 7 Appendices

*nginx*
*CopyEdit*

```
curl -X POST https://ixion/api/token \
-d "client_id=CLIENT_ID" \
-d "client_secret=SECRET" \
-d "grant_type=password" \
-d "username=email@example.com" \
-d "password=yourpassword"
```

*Default development admin credentials (for testing purposes):*
*Email*: `admin@ixion.local`
*Password*: `password`