Software Requirement Specifications

Fundamentals of Software Engineering

Version: 1.9

Project Code	*CS-3009*
Supervisor	*Ms. Sobia Iftikhar*
Co Supervisor	*Mr. Zeeshan Mustafa*
Project Team	*1. Hafiz Farhad* *2. 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 Added
1.2	Sohaib Ali	01/05/2025	Revised Introduction
1.3	Sohaib Ali	01/05/2025	Overall System Description Added
1.4	Hafiz Farhad	02/05/2025	External Interface Added
1.5	Sohaib Ali	02/05/2025	Functional Requirements Added
1.6	Sohaib Ali	02/05/2025	Non-Functional Requirements Added
1.7	Hafiz Farhad	03/05/2025	Revised Functional Requirements
1.8	Hafiz farhad	03/05.2025	References
1.9	Sohaib Ali	03/05/2025	Appendices

Distribution List

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

Document Sign-Off

Version	Sign-off Authority	Sign-off Date
1.1	Ms. Sobia Iftikhar	
1.2	Ms. Sobia Iftikhar	
1.3	Ms. Sobia Iftikhar	
1.4	Ms. Sobia Iftikhar	
1.5	Ms. Sobia Iftikhar	
1.6	Ms. Sobia Iftikhar	
1.7	Ms. Sobia Iftikhar	
1.8	Ms. Sobia Iftikhar	
1.9	Ms. Sobia Iftikhar	

Table of Contents

1. Introduction

1.1. Purpose of Document

This document outlines the software requirements for the "Ixion" Identity and Access Management (IAM) system. It defines the system functionalities, interfaces, and constraints to guide development and ensure stakeholder alignment.

1.2. Intended Audience

This document is intended for various stakeholders involved in the development and use of the Ixion system. These include software developers who will implement the functionalities described herein, test engineers responsible for validating the system's features, system administrators who will manage user access and deployment environments, and security auditors who will review the system's compliance with access control and authentication standards.

1.3 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

1.4 Document Convention

- ***Headings**: Bold, Title Case, 12pt*
- *Text: Arial, 10pt*
- *Code: Monospace font*

2. Overall System Description

2.1. Project Background

Identity and Access Management (IAM) plays a fundamental role in ensuring the security and integrity of digital systems by regulating access to resources and safeguarding user information. As digital applications scale and user bases grow, the need for a centralized, secure, and easily manageable identity system becomes increasingly critical. The Ixion project was initiated to address this need by offering a modern, scalable, and secure IAM system. Its development is driven by the aim to provide a comprehensive platform that facilitates authentication, role-based access control, and user identity management. The project aligns with the current cybersecurity demands of web-based applications and services that rely heavily on robust identity verification mechanisms.

2.2. Project Scope

The scope of the Ixion project includes the design and implementation of a full-featured IAM system. It handles user authentication using JSON Web Tokens (JWT), manages user roles and permissions, and enforces role-based access control (RBAC). The system provides RESTful APIs that allow seamless integration with other services and platforms. A React-based frontend enables administrators and users to interact with the system through an intuitive graphical interface. Core functionalities include Admin login, secure session handling, password management, and administrative oversight of user roles. The backend is powered by Flask and integrates with a PostgreSQL database for storing user and permission data

2.3. Not In Scope

Despite its robust set of features, Ixion does not encompass all IAM capabilities. Multi-factor authentication (MFA) is not currently supported, which may limit security in certain high-risk environments. Biometric authentication mechanisms such as fingerprint or facial recognition are also out of scope. Additionally, the system does not integrate with external LDAP directories, which are often used in enterprise environments for centralized identity management.

2.4. Project Objectives

The primary objectives of the Ixion project are to develop a secure, reliable, and user-friendly IAM system that simplifies user management and access control. Specific goals include enabling secure user login functionality, implementing RBAC to control access based on defined roles, exposing a well-documented RESTful API for easy integration, and delivering an efficient and aesthetically pleasing React-based frontend for both users and administrators. Through these objectives, the project seeks to improve security posture and administrative efficiency in managing application access.

2.5. Stakeholders

Stakeholders in the Ixion project include multiple user groups and contributors. The primary developers and maintainers of the system are responsible for designing, building, and updating the software. Organizations that require a dedicated IAM solution form another critical stakeholder group, as they will utilize the system to manage internal or external user access. End-users, who interact with the system for login purposes, are also considered stakeholders. Lastly, security professionals may be involved in auditing and ensuring the compliance of the system with relevant security standards.

2.6. Operating Environment

The Ixion system is intended to operate in environments that support containerized deployment using Docker and Docker Compose. The backend is developed using the Python Flask framework and relies on a PostgreSQL database for persistence. The frontend is built with React and communicates with the backend over HTTP/HTTPS. For email verification and user notifications, the system uses SMTP. These components are designed to work together in a networked environment and assume a reasonably modern infrastructure with internet access and administrative control over the hosting environment.

2.7. System Constraints

The development and operation of the Ixion system are subject to a number of constraints imposed by the external environment. These include a range of software, hardware, cultural, legal, environmental, user, and third-party component constraints:

- ***Software constraints**: The system depends on specific technologies including Docker, Docker Compose, Python, Flask, PostgreSQL, and React. These dependencies restrict its operation to environments where these technologies are supported and properly configured.*
- ***Hardware constraints**: While no specialized hardware is required, the system assumes access to hardware resources sufficient to run containerized applications, including a modern processor and adequate RAM.*
- ***Cultural constraints**: The interface language is English only. The system lacks localization or multilingual support, which could hinder adoption in non-English speaking regions.*
- ***Legal constraints**: Since the system handles user data and authentication, it must comply with data protection laws such as the GDPR depending on the deployment region.*
- ***Environmental constraints**: A stable and reliable network connection is required to support system functionalities such as email verification using SMTP. The system is not optimized for extreme or offline environments.*
- ***User constraints**: Ixion is intended for use by technically skilled users such as system administrators and developers. It is not tailored for use by general or non-technical users.*
- ***Off-the-shelf component constraints**: The system incorporates third-party libraries and Docker images that carry their own licensing and functional limitations. Any restrictions or security concerns associated with these components are consequently inherited by the Ixion system.*

2.8. Assumptions & Dependencies

Several assumptions underlie the development and deployment of Ixion. It is assumed that all JWT secret keys will be generated and stored securely to prevent token forgery. The application assumes the presence and proper configuration of a PostgreSQL database and that it remains accessible to the backend service at runtime. Furthermore, the system depends on the availability of a functioning SMTP server to handle email-based user verification. Developers and users are also assumed to have access to Docker-compatible environments to ensure seamless deployment and operation.

3. External Interface Requirements

3.1. Hardware Interfaces

There are no specific hardware interface requirements for the Ixion system. It is intended to run on standard virtual machines or cloud environments that support containerized applications and do not require interaction with specialized hardware.

3.2. Software Interfaces

Ixion is built on a modular software stack with defined interfaces between its components. The backend service is developed using Flask, a Python-based microframework. It communicates with a PostgreSQL database to manage user data and session information. The frontend interface is built using the React JavaScript library and interacts with the backend via HTTP RESTful APIs. The application is deployed using Docker and Docker Compose, which provide consistent environments for testing and deployment.

3.3. Communications Interfaces

The system communicates primarily over HTTP/HTTPS protocols to serve API endpoints to the frontend and other clients. It uses SMTP to send verification and notification emails to users. All communications involving sensitive information are encrypted using SSL/TLS to ensure confidentiality and data integrity. API endpoints are secured using JWT tokens to authenticate and authorize users for various operations. Synchronization between the frontend and backend is maintained via RESTful API conventions and JSON data exchange formats.

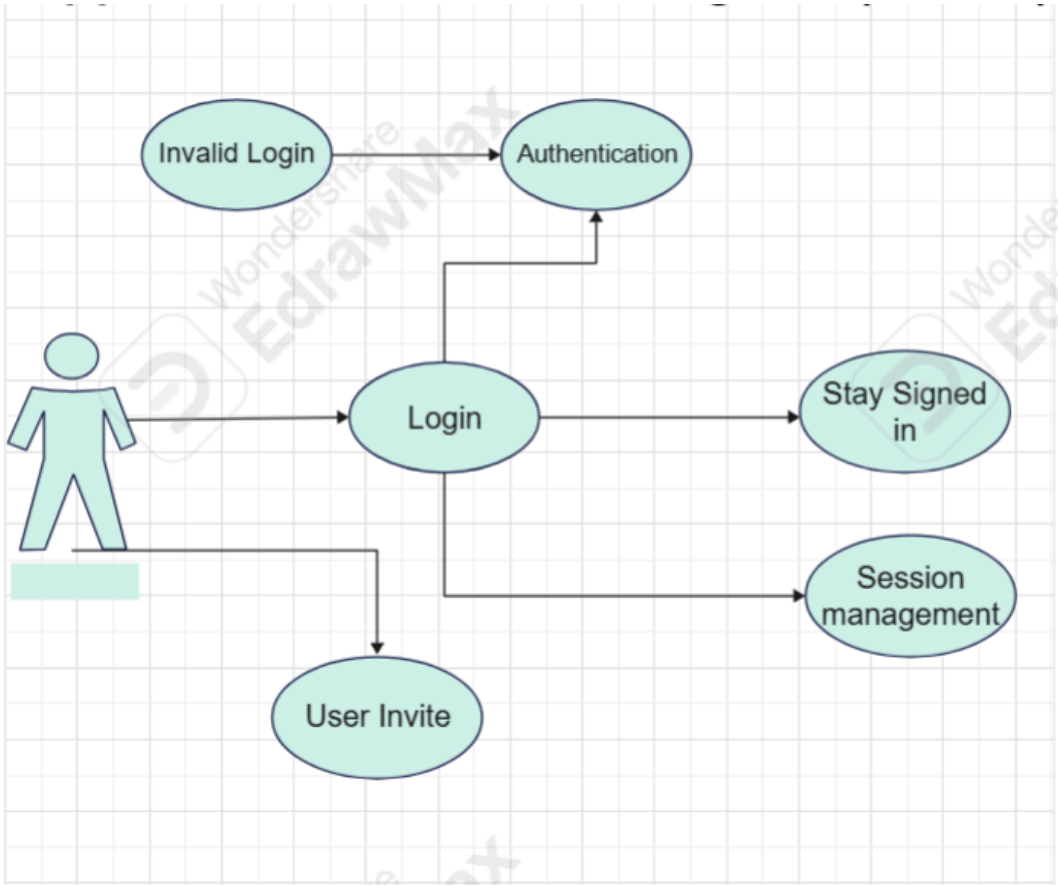
4. Functional Requirements

4.1. Functional Hierarchy

The Ixion system is divided into several main functional modules. The authentication module includes features such as login, which rely on JSON Web Tokens for stateless session management. The authorization module enables administrators to define roles and assign permissions to control access to system resources. The user management module allows profile updates and password reset functionalities, ensuring a complete identity lifecycle management system. These functions are integrated into both the frontend and backend systems to provide a unified user experience.

4.2. Use Cases

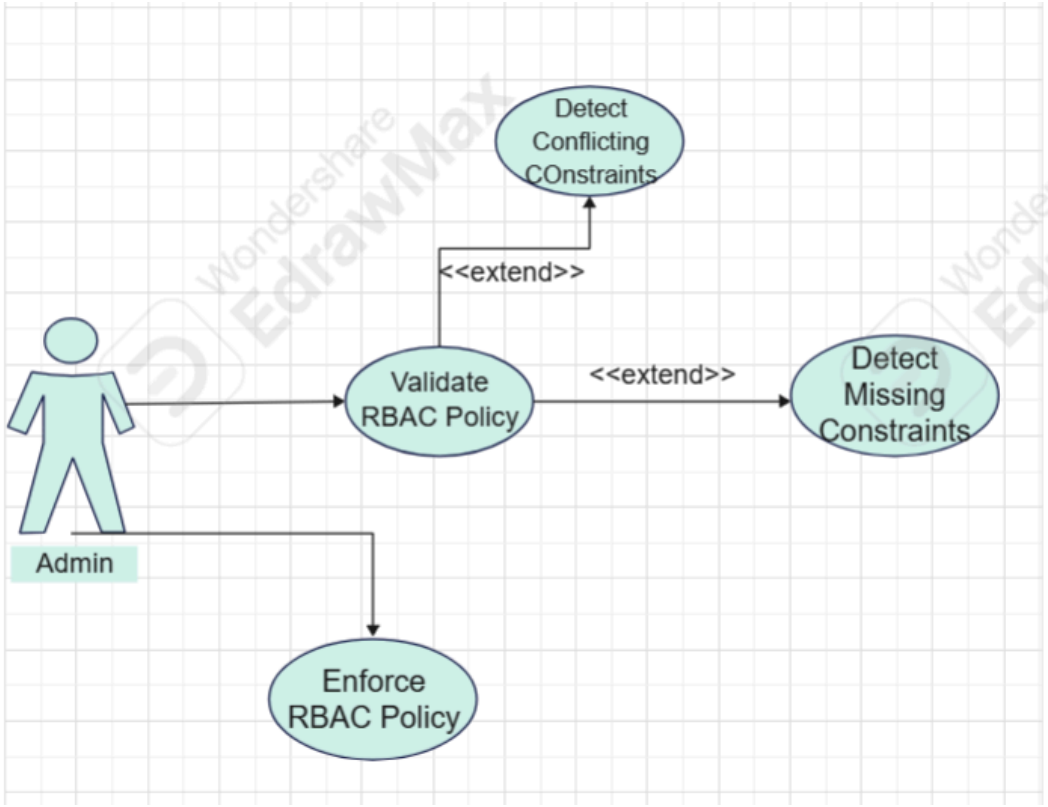
4.2.1. User Login



This use case describes the process through which a registered user logs into the system. The user provides their login credentials (email and password), which are validated by the backend service. If the credentials are correct, the system generates a signed JWT token and returns it to the user, allowing authenticated access to protected resources. This token is stored by the client (frontend) and attached to subsequent API requests. If the credentials are incorrect, an error message is displayed, and access is denied. This use case is part of the broader authentication feature and ensures secure and seamless login functionality for users.

User Login		
Use case Id:	UC-01	
Actors: *Admin(System User)*		
Feature: *Authentication*		
Pre-condition:	Admin is registered in the system, Admin has access to the login page.	
Scenarios		
Step#	***Action***	***Software Reaction***
1.	Admin enters username and password	System validates credentials
2.	*Admin clicks the login button*	*If valid, JWT is generated and admin is redirected to the dashboard*
Alternate Scenarios:		
1a: *Admin enters incorrect credentials – System displays an error message "Invalid login"* ***2a:*** *Admin's account is deactivated – System blocks login and shows "Account disabled*		
Post Conditions		
Step#	***Description***	
1	Admin is authenticated and logged into the system	
2	*JWT token is issued and stored in the client's browser.*	

4.2.2.



This use case explains how an administrator assigns roles to users to control access to features. The admin logs into the system, accesses the user management interface, and selects a user account. Through the interface, the admin assigns one or more predefined roles (e.g., admin, user) to the user. The system validates the request and updates the user's permissions in the database accordingly. This functionality supports the enforcement of Role-Based Access Control (RBAC), ensuring that users only access features appropriate to their roles. This use case describes the process through which a registered user logs into the system. The user provides their login credentials (email and password), which are validated by the backend service. If the credentials are correct, the system generates a signed JWT token and returns it to the user, allowing authenticated access to protected resources. This token is stored by the client (frontend) and attached to subsequent API requests. If the credentials are incorrect, an error message is displayed, and access is denied. This use case is part of the broader authentication feature and ensures secure and seamless login functionality for users.

Role Assignment		
Use case Id:	UC-02	
Actors: *Admin(System User)*		
Feature: *Role Management*		
Pre-condition:	Admin is logged into the system and has appropriate privileges.	
Scenarios		
Step#	***Action***	***Software Reaction***
1.	Admin navigates to the User Management page	System displays a list of users with search and filter options
2.	*Admin selects a user*	*Admin assigns a role (e.g.,* `admin`*,* `user`*)*
Alternate Scenarios:		
1a: *Admin selects a non-existent user – System displays an error message* ***2a:*** *Admin assigns an invalid or undefined role – System rejects the action and displays validation error*		
Post Conditions		
Step#	***Description***	
1	Selected user's role is updated in the system	
2	*System logs the action for auditing*	
3	*User gains new access permissions based on access role*	

5. Non-functional Requirements

5.1. Performance Requirements

The system is expected to maintain high responsiveness, with API endpoints returning responses in under 500 milliseconds for 95% of the requests under normal load. The React frontend is optimized for quick load times, and user actions should result in real-time feedback where applicable.

5.2. Safety Requirements

Ixion is designed with safety and reliability in mind. All passwords are securely hashed using modern cryptographic algorithms before being stored in the database. The system prevents common vulnerabilities such as SQL injection and cross-site scripting (XSS) through input validation and proper coding practices. Error handling mechanisms are in place to ensure graceful failure and user-friendly error messages.

5.3. Security Requirements

Security is a core requirement of the Ixion system. The system employs several best practices and security controls to ensure data confidentiality, integrity, and access control. These include the following:

- ***Use strong secrets**: Replace the default JWT secret with a strong, randomly generated value to prevent unauthorized token creation.*
- ***Secure communications**: Ensure all data exchanges occur over HTTPS in production environments to protect against man-in-the-middle attacks.*
- ***Regular updates**: Keep all project dependencies and libraries up to date to avoid known vulnerabilities.*
- ***Password policies**: Enforce password complexity rules such as a mix of uppercase, lowercase, numbers, and special characters.*
- ***Monitoring**: Enable logging and routinely review audit logs to detect and investigate suspicious activities.*
- ***Access review**: Conduct periodic reviews of user roles and access rights to maintain the principle of least privilege.*

All authentication is handled using JWT tokens, and access to system features is governed through role-based permissions. Additionally, sensitive data such as user credentials and tokens are stored using secure encryption and hashing mechanisms. Input sanitization is applied consistently to protect against injection attacks, and security checks are embedded in both frontend and backend layers of the system. All user authentication is handled using JWT tokens, and access to resources is strictly controlled through role-based permissions. The system enforces HTTPS for secure communication and relies on secure storage practices for sensitive data such as tokens and email credentials. User inputs are sanitized to avoid injection attacks, and authorization checks are implemented on both frontend and backend components.

5.4. User Documentation

The following user documentation provides clear, step-by-step guidance to assist developers and administrators in setting up, configuring, and using the Ixion system:

Installation

- *Ensure the following prerequisites are installed: Python 3.9+, Node.js 14+, PostgreSQL 12+, Docker and Docker Compose (optional).*

Backend Setup

- *Clone the repository using `git clone https://github.com/hafizfarhad/Ixion.git` and navigate to the backend directory.*
- *Create and activate a virtual environment using `python -m venv Ixion-venv` and activate it accordingly.*
- *Install backend dependencies using `pip install -r requirements.txt`.*
- *Create a `.env` file and define environment variables such as `JWT_SECRET`, `DATABASE_URL`, `ADMIN_EMAIL`, and `ADMIN_PASSWORD`.*
- *Initialize the PostgreSQL database and run `python app.py` to apply schema and start the backend server.*

Frontend Setup

- *Navigate to the `frontend` directory.*
- *Install frontend dependencies using `npm install`.*
- *Create a `.env` file with `REACT_APP_API_URL=http://localhost:5000/api`.*
- *Start the frontend server using `npm start` to access the UI at `http://localhost:3000`.*

Docker Deployment

- *For containerized deployment, run `docker-compose up -d` from the project root.*
- *This will launch the backend, frontend, and PostgreSQL services.*

Configuration

- *Key environment variables include:*
  - *`JWT_SECRET`: Secret key for JWT token signing.*
  - *`DATABASE_URL`: PostgreSQL connection URI.*
  - *`ADMIN_EMAIL`: Email for the initial admin account.*
  - *`ADMIN_PASSWORD`: Password for the initial admin account.*

Usage

- *Log in via the frontend at `http://localhost:3000` using the admin credentials defined in the environment.*
- *Change the default admin password after first login.*
- *Create additional users and manage accounts from the admin interface.*

User Management

- *Admins can create, edit, and delete users.*
- *Assign roles and manage account statuses.*
- *Send invitations to new users.*

Role Management

- *Create and assign roles.*
- *Configure permissions associated with each role.*

Permission Management

- *View existing permissions.*
- *Add custom permissions and assign them to roles accordingly.*

Additional documentation is available in the README and Swagger UI for API interaction and testing. to facilitate installation, configuration, and use of the Ixion system. This includes a README file with step-by-step setup instructions, environment configuration details, and troubleshooting tips. The API is documented using Swagger, offering a user-friendly interface for exploring available endpoints and testing API calls.

6. References

- *Hafiz Farhad, "Ixion - Identity and Access Management System", GitHub Repository, https://github.com/hafizfarhad/Ixion*
- *Flask Documentation, Pallets Projects, https://flask.palletsprojects.com*
- *React Documentation, Meta, https://reactjs.org/docs/getting-started.html*
- *JWT Introduction and Specification, JWT.io, https://jwt.io/introduction*
- *Docker Documentation, Docker Inc., https://docs.docker.com*
- *Docker Compose Documentation, Docker Inc., https://docs.docker.com/compose*

7. Appendices

The appendices include configuration files such as the Docker Compose YAML file used for deployment and a list of required environment variables with sample values. These resources support easy setup and consistent deployment across development and production environments.