Software Design Final Project

Michael Smith

Professor Nalubandhu

SWENG 837

4/28/24

# 

[Problem Statement and Requirements](#_wn912ltl8ybo)

[1. Business Requirements](#_99egqdn6vmbk)

[2. Non-Functional Requirements](#_l7yvqkwsik09)

[System Design using Domain Modeling](#_go2a6v4u11t3)

[UML Use Case Diagram](#_etuy3xmmz76r)

[UML Domain Model](#_9o1lung94ws1)

[UML Class Diagram](#_t9hemwqo5r3i)

[UML Sequence Diagrams](#_fz9oa8zftx60)

[System Sequence Diagram](#_sq21viqutn7c)

[Sequence Diagram: Password Reset](#_5rjs151ofic2)

[System Sequence Diagram: MFA Setup](#_c17wg3swsgbc)

[UML State Diagram](#_ltislcikhl1)

[UML Activity Diagram](#_89bnjoyo11gl)

[UML Component Diagram](#_lvt89asq9ivj)

[UML Deployment Diagram](#_h6hrawhu1cgg)

[1. Design Patterns](#_vhfd54uvghqx)

# 

# Problem Statement and Requirements

## Business Requirements

### Problem Statement

As fraud and cybercrime become a staple of the modern digital landscape, ensuring the safety of user data during authentication has become critical.

### Functional Requirements

* User Registration:
  + User registration with personal details such as email and password
  + Implement strong password policies to enhance security.
* Password Hashing:
  + Employ modern cryptographic methods to hash passwords securely.
  + Ensure that passwords are never stored in plaintext.
* Login Process:
  + Provide a secure login mechanism that verifies hashed passwords.
  + Include measures to prevent brute force and dictionary attacks, such as rate limiting.
* Multi-Factor Authentication (MFA):
  + Support multi-factor authentication options, including SMS-based OTP, email verification, or authenticator apps.
  + Make MFA mandatory for accessing sensitive features.
* Session Management:
  + Manage user sessions securely with timeout policies and secure cookie attributes.
  + Provide a logout functionality that fully clears session data.
* Account Recovery:
  + Offer a secure account recovery process using email or SMS.
  + Include identity verification steps to prevent unauthorized account access.

### Actors List

| **Type** | **Actor** | **Goal Description** |
| --- | --- | --- |
| Primary | User | Access various aspects of user data to qualify user identification and security |
|  | System Administrator | Monitor and control system’s accounts, permissions, settings, and security. |
| Supporting | Cloud Storage | Cloud provider for data storage |
|  | Credential Management System | Authority to authenticate user credentials |
|  | MFA Solution | Providing Multi-factor Authentication Management |
| Offstage | External entities | Access user data to inform a variety of federal and state agencies decisions. |

### Business Goals

* Decrease company data breach incidents
* Simplify user workflow to secure company data
* Compatible with all mainstream devices

## Non-Functional Requirements

### Performance Requirements

* **Scalability**: The system must, without degrading performance, be able to handle users in excess of 1 million users concurrently.
* **Response Time**: Under normal system circumstances all authentication responses should not exceed 2 seconds.
* **Throughput**: System should be able to process authentications at a rate of 1000 transactions per second with no system performance degradation.

### Security Requirements

* **Authentication**: Employ methods for robust user verification, including such as password hashing and multi-factor authentication.
* **Authorization**: Employ system control policies to prevent users accessing restricted areas of the system.
* **Data Encryption**: Implement AES for data storage and TLS for data transmission in order to prevent data from being accessed without authorization.
* **Regular Security Audits**: Schedule regular security audits to identify potential data security risk vectors.

### Maintainability Requirements

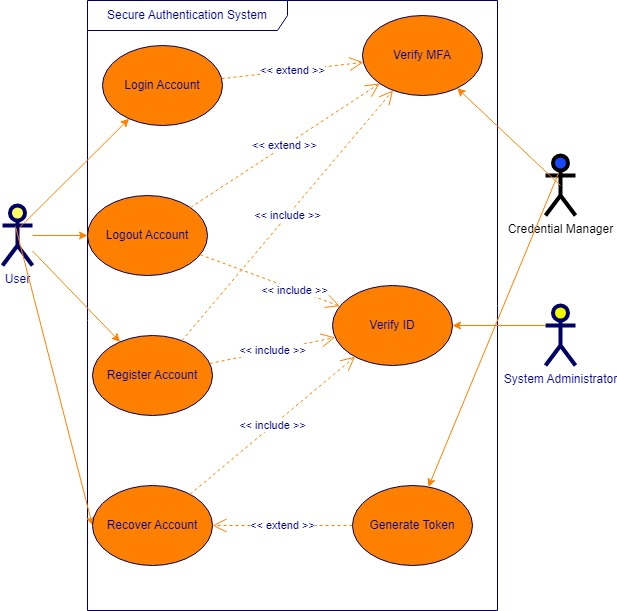
* **Code Modularity**: System architecture should be modular for simpler maintainability.
* **Documentation**: Maintain up-to-date documentation of all pertinent aspects of the system, including codebase, user guidelines and API.
* **Testing Strategies**: Employ rigorous, automated testing protocols such as unit, integration and system tests.

### Additional Non-Functional Requirements

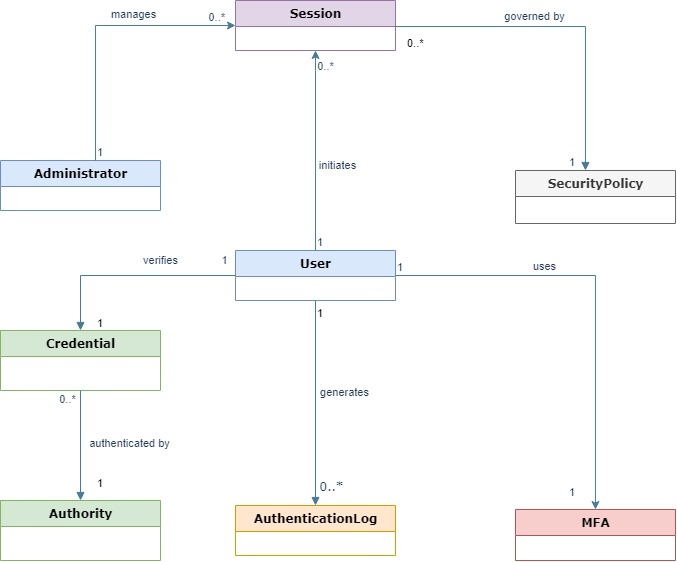
* **Usability**: Design a visually intuitive interface to minimize user dissatisfaction.
* **Accessibility**: Provide modern accessibility features for disabled users.
* **Reliability**: System target uptime: 99.9%.
* **Interoperability**: System should be able to communicate without translation to higher languages.
* **Internationalization**: Include regional and multilingual settings to support global users.
* **Compliance**: Adhere to all relevant legal and government regulatory requirements.

# System Design using Domain Modeling

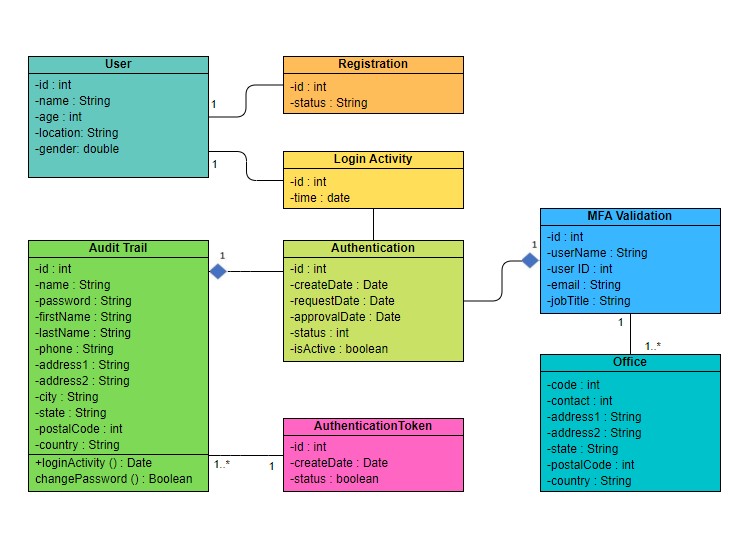
## UML Use Case Diagram



## UML Domain Model

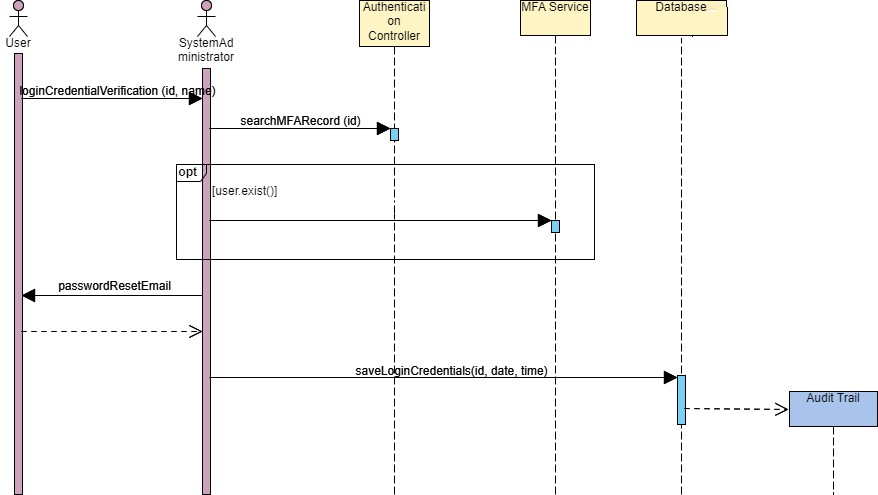


## UML Class Diagram



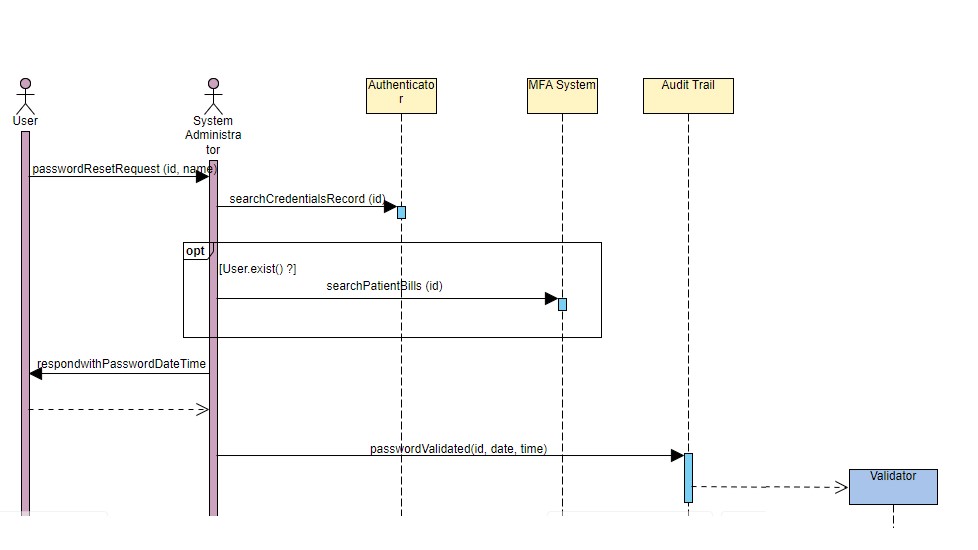
## UML Sequence Diagrams

### System Sequence Diagram



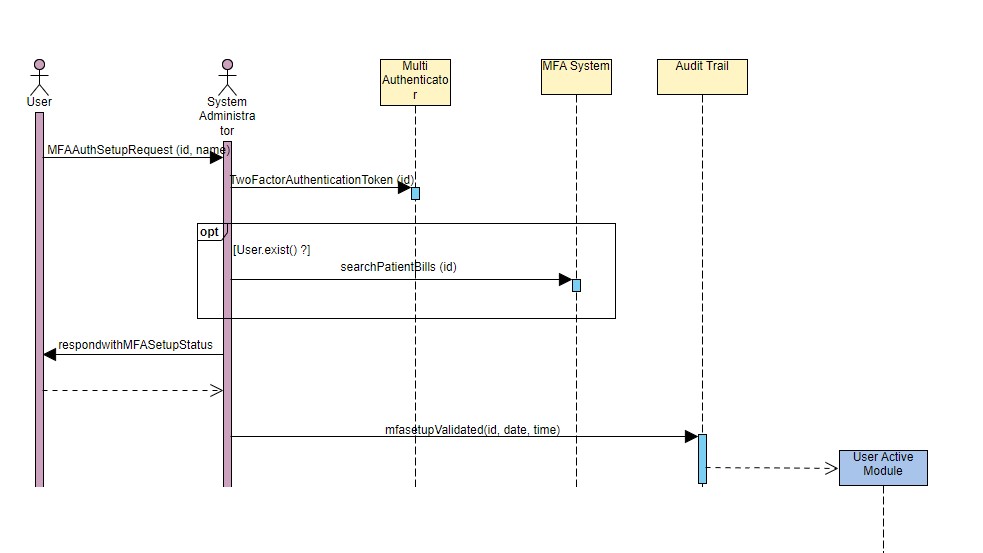
### 

### Sequence Diagram: Password Reset

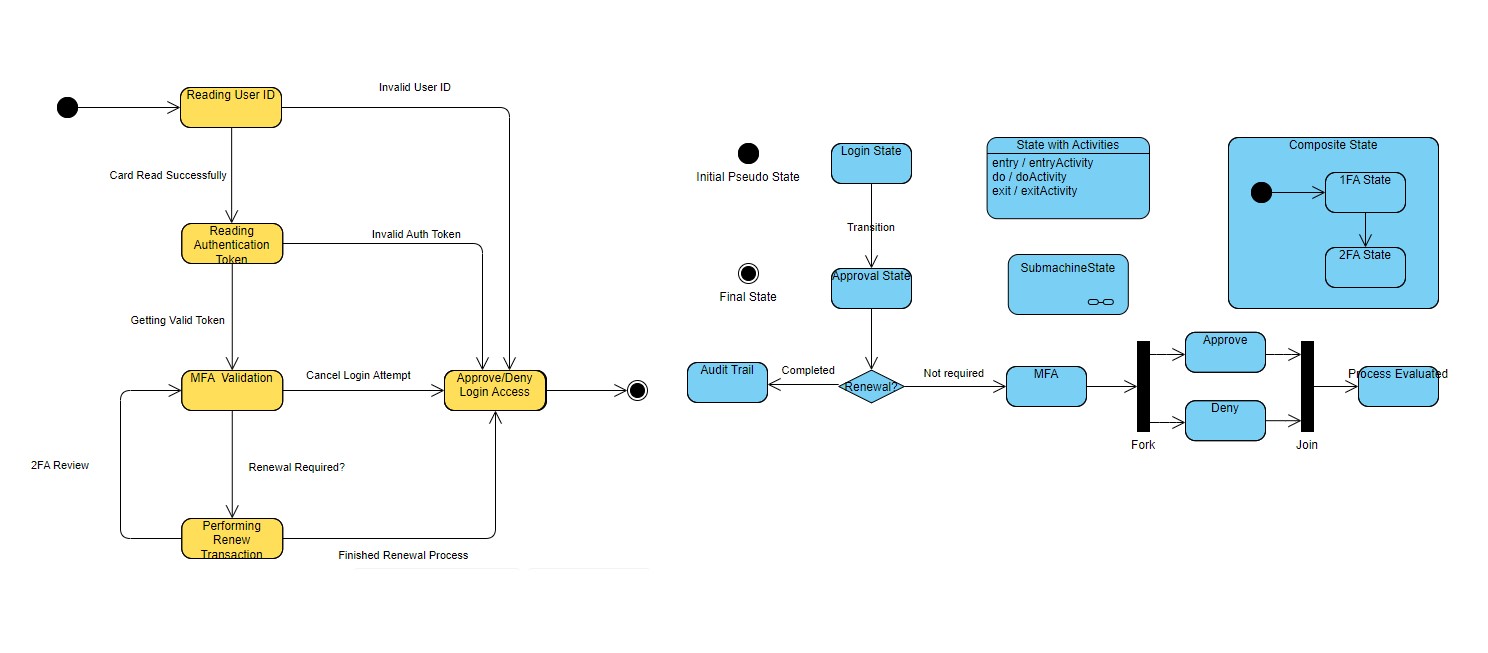


### 

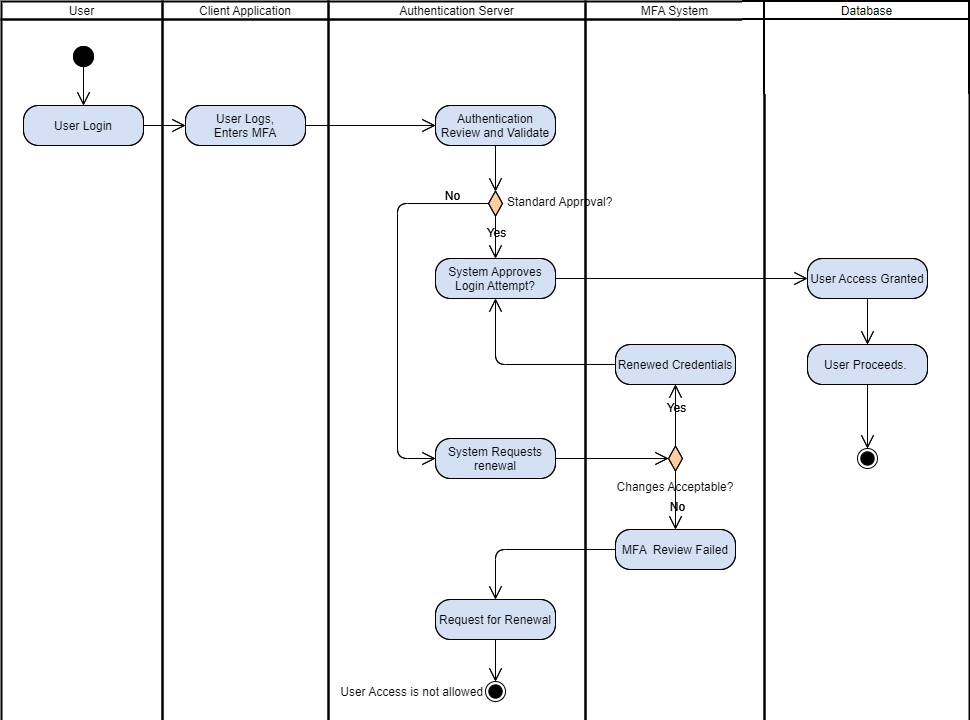
### Sequence Diagram: MFA Setup



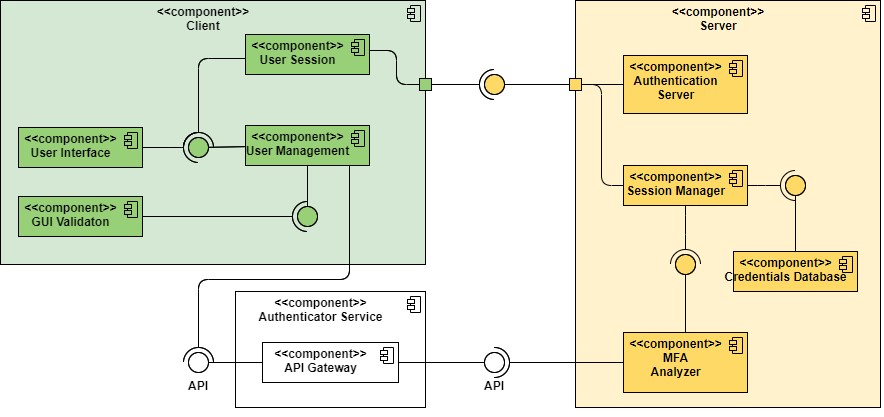
## UML State Diagram



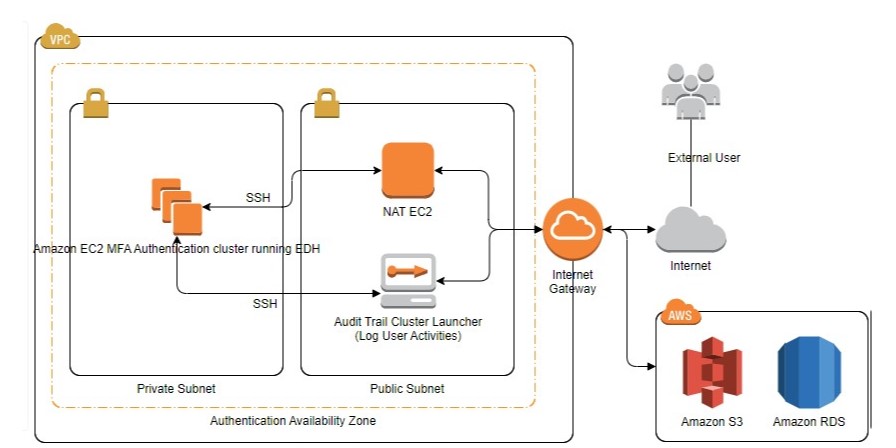
## UML Activity Diagram



## UML Component Diagram



## UML Deployment Diagram



## Design Patterns

### Grasp

#### Controller

AuthenticationController is the intermediary between the UI and backend services, this was done in order to keep the system easy to maintain and scale.

#### Low Coupling

AuthenticationController also achieves lower coupling for essentially the same reason as above. This allows components to be altered or replaced with little impact on the system.

#### High Cohesion

Classes in the system are kept in narrow, well defined roles;this modularity, in addition to achieving the aforementioned results, also helps keep complexity to a minimum, reducing likelihood of errors.

### Solid

#### Single Responsibility Principle

Ensuring classes have only one reason to change helps prevent changes in one class causing unnecessary changes to another class.

### GoF

#### Factory Method

For creating objects without specifying their class, useful for MFA challenges.

### Microservices

#### API Gateway

The API Gateway acts as a single entry point for the client, efficiently streamlining authentication and authorization.

#### Database per Service

The systems microservices each have their own separate database. This assists with service decoupling and data security.

#### Circuit Breaker

By employing a proxy to monitor service responses, cascading failures across multiple services can be cut off like a “circuit breaker.”