# Software Design Specification (SDS)

**Project Name**: El Downtown  
**Prepared By**: Riham Fouad– Ahmed Abdelsalam –Mohamed Taha– Ali Nazeer – Abdelaleem Baher

**Date**: 11/9/2024

## 1. Introduction

### 1.1 Purpose

### The purpose of this document is to describe the design, architecture, and technical specifications of the El Downtown project. It outlines the functionality, system components, and design decisions to be followed during the development process. This SDS serves as a detailed guide for developers and stakeholders, ensuring the system is built according to the defined requirements and meets user needs.

### 1.2 Scope

This SDS covers the design and implementation details of the **El Downtown** project. The software will perform the following major tasks:

* **User Registration and Authentication:** Enables users to create accounts, log in, and manage their secure data through email or third-party authentication.
* **Content Sharing and Interaction:** Allows users to create, edit, delete, and interact with posts, including text, images, and videos.
* **User Profile Management and Social Features:** Provides options for users to customize their profiles, connect with others, and engage through a social feed and commenting, upvoting/downvoting on the posts .

## 2. System Overview

The system consists of the following components:

* **Frontend**: React native
* **Backend**: Flask to handle server-side logic
* **Database**: MySQL used for storing application data.

## 3. System Architecture

### 3.1 Architectural Design

This project follows a **client-server architecture**, where:

* **Frontend** communicates with the backend using APIs to send and receive data.
* **Backend** interacts with the MySQL database to manage, store, and retrieve application data.

### 3.2 Data Flow

1. **User Interaction**: The user interacts with the UI to perform an action.
2. **Request Processing**: The frontend sends an API request to the backend server.
3. **Data Handling:** The backend processes the request, interacts with the MySQL database to fetch, update, or delete data as necessary, based on the user's action**.**
4. **Response**: The backend sends a response back to the frontend, and the UI updates accordingly to reflect the changes for the user.

## 4. Database Design

### 4.1 Database Schema

The system will store data in a **relational database (MySQL)** with the following entities and relationships:

**Table 1**: User

* + **user\_id**: Unique identifier for each user (Primary Key).
  + **username**: Display name for social interaction.
  + **email**: User’s email address for login and notifications.
  + **password\_hash**: Securely hashed password.
  + **role**: User’s role (Admin or User)

**Table 2**: Post

* + **post\_id:** Unique identifier for each post (Primary Key).
  + **user\_id:** Foreign key linking to the user who created the post.
  + **content:** Text content of the post.
  + **time:** Time and date when the post was created.

**Table 3:** Comment

* **comment\_id**: Unique identifier for each comment (Primary Key).
* **post\_id**: Foreign key linking to the post being commented on.
* **user\_id**: Foreign key linking to the user who made the comment.
* **content**: Text of the comment.
* **time**: Time and date when the comment was made.

**Table 4:** Friendship

* **user\_id**: Foreign key linking to the user initiating the connection.
* **friend\_id**: Foreign key linking to the connected user.
* **status**: Status of the friendship (e.g., pending, accepted).

**Table 5:** Notification

* **notification\_id**: Unique identifier for each notification (Primary Key).
* **user\_id**: Foreign key linking to the user receiving the notification.
* **type**: Type of notification (like, comment, friend request).
* **time**: Time and date of the notification.

**Table 6:** Messages

* **message\_id**: Unique identifier for each message (Primary Key).
* **sender\_id**: Foreign key linking to the user sending the message.
* **receiver\_id**: Foreign key linking to the user receiving the message.
* **content**: Text content of the message.
* **time**: Time and date when the message was sent.

**Table 7:** Vote

* **vote\_id**: Unique identifier for each vote action (Primary Key).
* **post\_id**: Foreign key linking to the post being voted on.
* **user\_id**: Foreign key linking to the user who voted.
* **vote\_type**: Indicates whether the vote is an upvote or downvote. This could be represented as a boolean.

***Relationships:***

* ***One-to-Many****: Each user can have multiple posts, comments, and votes, but each post, comment, and like belongs to one user.*
* ***Many-to-Many****: The Friendships table enables a many-to-many connection between users.*

## 5. Technology Stack

* **Frontend:** React Native, used for a responsive and cross-platform user interface on both web and mobile applications.
* **Backend:** Flask, used to handle server-side logic, manage API interactions, and process business logic.
* **Database:** MySQL, employed for managing and storing relational data, including user profiles, posts, comments, and connections.
* **Hosting:** Cloud service such as AWS or Google Cloud, providing a scalable and reliable environment for backend services.

## 6. Testing Plan

### 6.1 Unit Testing

### User Registration and Authentication: Tests for input validation, email verification, password encryption, and error handling (e.g., incorrect password).

### Content Creation and Management: Tests for post creation, editing, deletion, and error handling for invalid content or permissions.

### Profile Management: Tests for updating profile details, changing profile picture, and saving updates.

### 6.2 Integration Testing

### Frontend-Backend Integration: Tests for API communication between the client (React Native) and server (Flask) to handle data transfer.

**Backend-Database Interaction**: Testing the MySQL queries and data transactions, ensuring actions like user registration, posting, and commenting are accurately stored and retrieved.

**Third-Party Authentication**: Test integration with Google and Facebook for user authentication.

### 6.3 User Acceptance Testing (U AT)

End users will be involved in testing the system to verify that it meets their requirements and expectations.

### 6.4 Performance Testing

Stress and load testing will be conducted to ensure the system can handle the required number of users and operations without degradation in performance.

## 7. Conclusion

The **El Downtown** project is designed to fulfill the specified functional and non-functional requirements outlined in this Software Design Specification (SDS). By using the client-server architecture, using technologies like React Native, Flask, and MySQL, and implementing testing plans, this design will ensure the system is robust, scalable, and user-friendly.