# Design Document

**Vu Binh Minh - V202100421**
**Nguyen Tung Lam - V202100571**
**Bui Huy Linh Phuc - V202100398**

## 1 Conceptual & Logical Design

### 1.1 Functional Requirements

- User registration and login functionality with JWT-based authentication
- Secure password handling via hashing algorithms
- CRUD operations for user-generated posts and comments
- Like and dislike features for posts
- Follow/unfollow functionality between users
- Media/file upload support (e.g., images in posts or profiles)
- Profile management and editable user information
- RESTful API to handle all frontend-backend communication

### 1.2 Non-functional Requirements

- Responsive and intuitive user interface across devices
- Modular and maintainable codebase with clear separation of concerns
- Scalability of backend components for increasing data volume
- Use of secure authentication practices and HTTPS-ready communication
- Efficient querying and indexing of relational data in MySQL
- Comprehensive error handling and input validation mechanisms

### 1.3 Entity–Relationship Diagram

Since the figure is too complex, here is a simpler view:

### 1.4 Normalization proof

- 1NF: All attributes are atomic, with with no repeating groups.
- 2NF: For tables with candidate keys, non-key attributes fully depend on the entire key.
- 3NF: No transitive dependencies exist; all non-key attributes depend directly on the primary or candidate keys.

## 2 Physical Schema Definition

### 2.1 Data Definition Language script

```
CREATE TABLE ‘User‘ (
  id VARCHAR(191) PRIMARY KEY,
  email VARCHAR(191) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  username VARCHAR(191) UNIQUE NOT NULL,
  avatar VARCHAR(191),
  cover VARCHAR(191),
  name VARCHAR(191),
  surname VARCHAR(191),
  description VARCHAR(191),
  city VARCHAR(191),
  school VARCHAR(191),
  work VARCHAR(191),
  website VARCHAR(191),
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Post (
  id INT AUTO_INCREMENT PRIMARY KEY,
  `desc` TEXT NOT NULL,
  img VARCHAR(191),
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  userId VARCHAR(191),
  FOREIGN KEY (userId) REFERENCES `User`(id) ON DELETE CASCADE
);

CREATE TABLE Comment (
  id INT AUTO_INCREMENT PRIMARY KEY,
  `desc` TEXT NOT NULL,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  userId VARCHAR(191),
  postId INT,
  FOREIGN KEY (userId) REFERENCES `User`(id) ON DELETE CASCADE,
  FOREIGN KEY (postId) REFERENCES Post(id) ON DELETE CASCADE
);

CREATE TABLE `Like` (
  id INT AUTO_INCREMENT PRIMARY KEY,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  userId VARCHAR(191),
  postId INT,
  commentId INT,
  FOREIGN KEY (userId) REFERENCES `User`(id) ON DELETE CASCADE,
  FOREIGN KEY (postId) REFERENCES Post(id) ON DELETE CASCADE,
  FOREIGN KEY (commentId) REFERENCES Comment(id) ON DELETE CASCADE
);

CREATE TABLE Follower (
  id INT AUTO_INCREMENT PRIMARY KEY,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  followerId VARCHAR(191),
  followingId VARCHAR(191),
  FOREIGN KEY (followerId) REFERENCES `User`(id) ON DELETE CASCADE,
  FOREIGN KEY (followingId) REFERENCES `User`(id) ON DELETE CASCADE
);

CREATE TABLE FollowRequest (
  id INT AUTO_INCREMENT PRIMARY KEY,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  senderId VARCHAR(191),
  receiverId VARCHAR(191),
  FOREIGN KEY (senderId) REFERENCES `User`(id) ON DELETE CASCADE,
  FOREIGN KEY (receiverId) REFERENCES `User`(id) ON DELETE CASCADE,
  UNIQUE (senderId, receiverId)
);
```

```
CREATE TABLE Block (
  id INT AUTO_INCREMENT PRIMARY KEY,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  blockerId VARCHAR(191),
  blockedId VARCHAR(191),
  FOREIGN KEY (blockerId) REFERENCES `User`(id) ON DELETE CASCADE,
  FOREIGN KEY (blockedId) REFERENCES `User`(id) ON DELETE CASCADE,
  UNIQUE (blockerId, blockedId)
);

CREATE TABLE Story (
  id INT AUTO_INCREMENT PRIMARY KEY,
  createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
  expiresAt DATETIME NOT NULL,
  img VARCHAR(191) NOT NULL,
  userId VARCHAR(191) UNIQUE,
  FOREIGN KEY (userId) REFERENCES `User`(id) ON DELETE CASCADE

-- UserProfileView
CREATE VIEW UserProfileView AS
SELECT id, username, name, surname, avatar, description, city, createdAt
FROM User;

-- PostWithUserView
CREATE VIEW PostWithUserView AS
SELECT p.id, p.desc, p.img, p.createdAt, p.updatedAt, u.username, u.avatar
FROM Post p
JOIN User u ON p.userId = u.id;

-- UserActivityView
CREATE VIEW UserActivityView AS
SELECT u.id, u.username,
       p.id AS postId, p.desc AS postDesc, p.createdAt AS postCreatedAt,
       c.id AS commentId, c.desc AS commentDesc, c.createdAt AS
           commentCreatedAt,
       l.postId AS likedPostId, l.createdAt AS likeCreatedAt
FROM User u
LEFT JOIN Post p ON u.id = p.userId
LEFT JOIN Comment c ON u.id = c.userId
LEFT JOIN `Like` l ON u.id = l.userId;

-- FollowerCountView
CREATE VIEW FollowerCountView AS
SELECT u.id, u.username,
       (SELECT COUNT(*) FROM Follower f WHERE f.followingId = u.id) AS
           followerCount,
       (SELECT COUNT(*) FROM Follower f WHERE f.followerId = u.id) AS
           followingCount
FROM User u;

-- Indexing
CREATE INDEX idx_post_userId ON Post(userId);
CREATE INDEX idx_post_createdAt ON Post(createdAt);
CREATE INDEX idx_comment_userId ON Comment(userId);
CREATE INDEX idx_comment_postId ON Comment(postId);
CREATE INDEX idx_comment_createdAt ON Comment(createdAt);
CREATE INDEX idx_like_userId_postId ON `Like`(userId, postId);
CREATE INDEX idx_like_userId_commentId ON `Like`(userId, commentId);
);
```

## 2.2 Database Optimization Strategies

### 2.2.1 Views

- **UserProfileView**: Displays user profile data, excluding sensitive fields like password.
- **PostWithUserView**: Joins posts with user details for feed displays.
- **UserActivityView**: Aggregates a user's posts, comments, and likes for activity feeds.
- **FollowerCountView**: Summarizes follower and following counts per user.

### 2.2.2 Indexes

- **Post**: Indexes on `userId` and `createdAt` for user posts and timelines.
- **Comment**: Indexes on `userId`, `postId`, and `createdAt` for comment retrieval.
- **Like**: Composite indexes on `(userId, postId)` and `(userId, commentId)` for like checks.
- **Story**: Index on `expiresAt` for active story queries.

# 3 Task Division & Project Plan

**Note: You can access our Github via this** Link.

## 3.1 Tasks Distribution

| **Name** | **Role** | **Responsibilities** |
|---|---|---|
| Nguyen Tung Lam | Frontend Developer & Project Manager | Responsible for UI/UX development, React components, and client-side state management. |
| Vu Binh Minh | Backend Developer | Implements Express.js API routes, middleware, and integrates security and RESTful practices. |
| Bui Huy Linh Phuc | Database & Authentication Engineer | Designs and manages the MySQL schema, handles data relationships, and implements secure authentication mechanisms. |

Table 1: Tasks Distribution

## 3.2 Project Plan

| **Milestone** | **Timeframe** | **Description** |
|---|---|---|
| Requirements Analysis & Planning | May 1–2 | Define system scope, entity models, and tech stack |
| Database Schema & Setup | May 3–5 | Design MySQL tables, relationships, and indexes |
| Backend API Development | May 6–10 | Develop Express.js endpoints for users, posts, auth |
| Frontend UI Implementation | May 11–15 | Build React components and page layouts |
| Authentication & File Upload | May 16–18 | JWT integration, cookie auth, media handling |
| Feature Integration & Testing | May 19–25 | Connect frontend to backend, test core features |
| Final Testing & Bug Fixing | May 26–28 | Final QA, performance tuning, and fixes |
| Documentation & Submission | May 29–30 | Complete README, code comments, and handover |

Table 2: Project Milestones and Timeline

# 4 Supporting Documentation

## 4.1 Rationale for design decisions

### 4.1.1 Database Design

The core design goals of this database is to support a full-featured social media platform; ensure data normalization, scalability, and referential integrity; enable key features: user management, posting, interactions (likes/comments), social graphs (follows/blocks), and stories.

### 4.1.2 Database Optimization Strategies

The selected views and indexes optimize query performance, simplify application logic, and ensure scalability, aligning with functional and non-functional requirements.

- **Views**:
  - **UserProfileView**: Chosen to simplify profile queries and exclude sensitive data (e.g., password), enhancing security and usability.
  - **PostWithUserView**: Joins `Post` and `User` for feed displays, reducing query complexity for frequent operations.
  - **UserActivityView**: Aggregates user interactions (posts, comments, likes) for activity feeds, minimizing complex joins in the application.
  - **FollowerCountView**: Precomputes follower/following counts, improving performance for profile displays.
- **Indexes**:
  - **Post**: Indexes on `userId` and `createdAt` support fast retrieval of user-specific posts and timeline sorting, critical for feeds.
  - **Comment**: Indexes on `userId`, `postId`, and `createdAt` optimize comment retrieval by user or post and date-based sorting.
  - **Like**: Composite indexes on `(userId, postId)` and `(userId, commentId)` speed up like checks, essential for real-time interactions.
  - **Story**: Index on `expiresAt` accelerates queries for active stories, supporting ephemeral content delivery.
- **Key Choices**: Indexes target non-key columns used in WHERE, JOIN, or ORDER BY clauses, complementing primary and foreign key indexes.

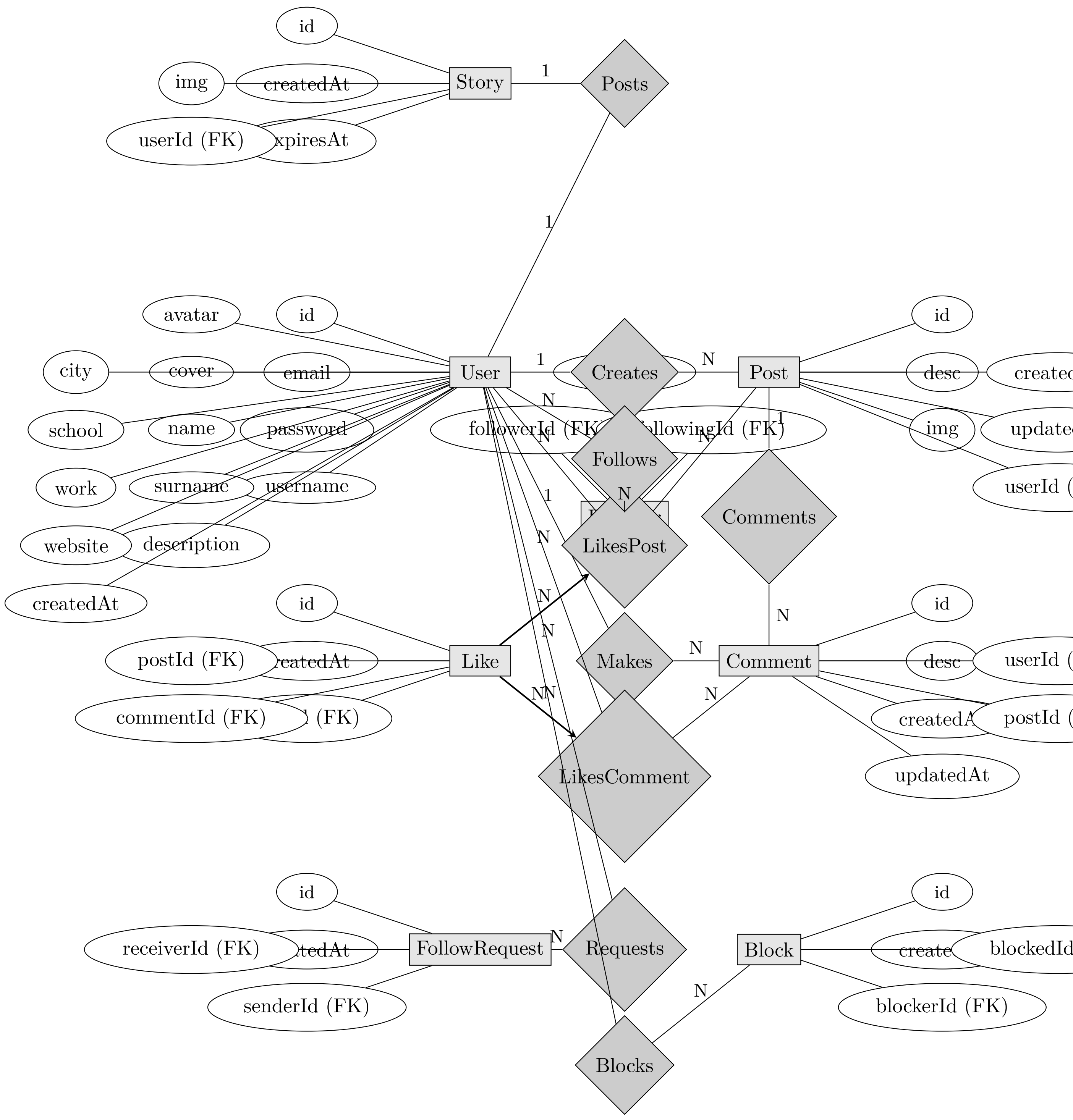

Figure 1: ERD in Chen Notation for Social Media Database

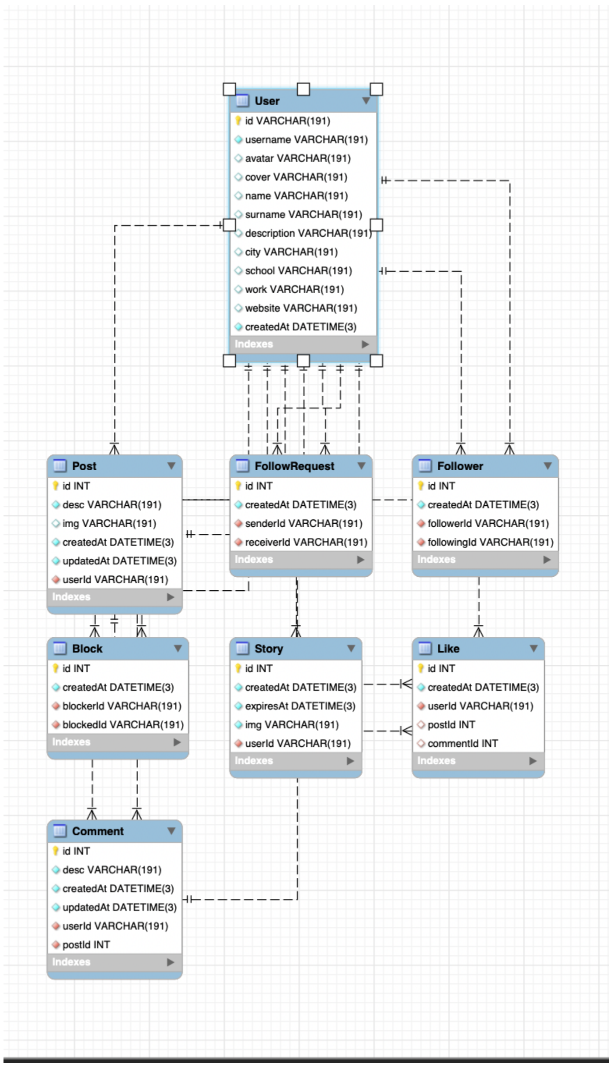
User
id VARCHAR(191)
username VARCHAR(191)
avatar VARCHAR(191)
cover VARCHAR(191)
name VARCHAR(191)
surname VARCHAR(191)
description VARCHAR(191)
city VARCHAR(191)
school VARCHAR(191)
work VARCHAR(191)
website VARCHAR(191)
createdAt DATETIME(3)
Indexes
Post
id INT
desc VARCHAR(191)
img VARCHAR(191)
createdAt DATETIME(3)
updatedAt DATETIME(3)
userId VARCHAR(191)
Indexes
FollowRequest
id INT
createdAt DATETIME(3)
senderId VARCHAR(191)
receiverId VARCHAR(191)
Indexes
Follower
id INT
createdAt DATETIME(3)
followerId VARCHAR(191)
followingId VARCHAR(191)
Indexes
Block
id INT
createdAt DATETIME(3)
blockerId VARCHAR(191)
blockedId VARCHAR(191)
Indexes
Story
id INT
createdAt DATETIME(3)
expiresAt DATETIME(3)
img VARCHAR(191)
userId VARCHAR(191)
Indexes
Like
id INT
createdAt DATETIME(3)
userId VARCHAR(191)
postId INT
commentId INT
Indexes
Comment
id INT
desc VARCHAR(191)
createdAt DATETIME(3)
updatedAt DATETIME(3)
userId VARCHAR(191)
postId INT
Indexes

Figure 2: ERD