

PROJECT REPORT

Project: SocialSphere – Social Media Platform

Report Date: December 25, 2025

Name:

Hadia Masood 3350

Zoya Zaheer 3378

Shozab Raza 3388

Taha Khan 3417

Class: Bscs 4a

ABSTRACT

SocialSphere is a database-driven social media platform designed to enable user interaction, content sharing, and secure communication. The system is built using a three-tier architecture consisting of a Windows Forms frontend, a Business Logic Layer (BLL), and a Data Access Layer (DAL), with SQL Server as the backend database. The project focuses on normalized database design, security, scalability, and performance optimization.

INTRODUCTION

2.1 Problem Statement

Existing social media systems face challenges such as inefficient database design, poor scalability, security vulnerabilities, and lack of proper access control. Managing large volumes of user data while ensuring privacy and performance remains a critical issue.

2.2 Objectives

1. Design a secure and normalized database following Third Normal Form (3NF).
2. Implement reliable user authentication and role-based access control.

3. Develop core social networking features such as posts, messaging, and friendships.
4. Ensure privacy of user data and secure password storage.
5. Provide a scalable and well-documented system.

2.3 Project Scope

In Scope:

- Database design
- Authentication
- Post management
- Friendships
- Messaging
- Notifications
- Admin panel
- Reporting

Out of Scope:

- Mobile applications
- Real-time notifications
- Payment systems
- Third-party integrations
- Advanced analytics

2.4 Implementation Timeline

1. **Phase 1 (Week 1):** Requirement analysis and database design
 2. **Phase 2 (Week 2):** Backend development (DAL & BLL)
 3. **Phase 3 (Week 3):** Frontend development (Windows Forms)
 4. **Phase 4 (Week 4):** Testing, documentation, and deployment
-

LITERATURE REVIEW

The database design follows normalization principles up to Third Normal Form (3NF) to eliminate data redundancy and ensure data integrity. Security best practices such as SHA256 password hashing and role-based access control are implemented to protect user data.

3.1 Database Normalization

Normalization is applied to minimize redundancy and dependency by organizing fields and table relationships. The database achieves 3NF by ensuring:

- Each table has a primary key
- Non-key attributes depend on the primary key
- No transitive dependencies exist

3.2 Security Considerations

- **Password Hashing:** SHA256 algorithm is used for secure password storage
- **Role-Based Access Control:** Admin and User roles with different permissions
- **SQL Injection Prevention:** Parameterized queries in DAL
- **Data Validation:** Input validation at both frontend and backend layers

3.3 Three-Tier Architecture

The system follows a layered architecture:

- **Presentation Layer:** Windows Forms UI
 - **Business Logic Layer (BLL):** Business rules and validation
 - **Data Access Layer (DAL):** Database operations and queries
-

DATABASE DESIGN

4.1 Database Schema

The SocialSphere database consists of five core tables: Users, Posts, Friendships, Messages, and Notifications. Proper primary and foreign key constraints enforce referential integrity, while indexes improve query performance.

4.2 Table Descriptions

4.2.1 USERS Table

Stores user account information including credentials, profile details, and account status.

Attributes:

- **UserId** (int, PK) - Primary identifier
- **Username** (string, UK) - Unique username
- **Email** (string, UK) - Unique email address
- **Password** (string) - Hashed password (SHA256)
- **FullName** (string) - User's full name
- **DateOfBirth** (date) - Birth date

- **Bio** (string) - Profile biography
- **ProfilePicture** (string) - Profile image path
- **CreatedAt** (datetime) - Account creation timestamp
- **IsActive** (bool) - Account active status
- **IsAdmin** (bool) - Admin role flag

4.2.2 POSTS Table

Contains user-generated content including text and images.

Attributes:

- **PostId** (int, PK) - Primary identifier
- **UserId** (int, FK) - References USERS(UserId)
- **Content** (string) - Post content
- **CreatedAt** (datetime) - Creation timestamp
- **UpdatedAt** (datetime) - Last modification timestamp
- **IsDeleted** (bool) - Soft delete flag

4.2.3 FRIENDSHIPS Table

Manages friend connections between users.

Attributes:

- **FriendshipId** (int, PK) - Primary identifier
- **UserId** (int, FK) - References USERS(UserId) - Requester
- **FriendId** (int, FK) - References USERS(UserId) - Receiver
- **CreatedAt** (datetime) - Request timestamp
- **Status** (string) - Status: Pending, Accepted, Rejected

4.2.4 MESSAGES Table

Stores private messages between users.

Attributes:

- **MessageId** (int, PK) - Primary identifier
- **SenderId** (int, FK) - References USERS(UserId)
- **ReceiverId** (int, FK) - References USERS(UserId)
- **Content** (string) - Message content
- **CreatedAt** (datetime) - Send timestamp
- **IsRead** (bool) - Read status flag

4.2.5 NOTIFICATIONS Table

Tracks user notifications for various activities.

Attributes:

- **NotificationId** (int, PK) - Primary identifier
- **UserId** (int, FK) - References USERS(UserId) - Recipient
- **TriggeredByUserId** (int, FK) - References USERS(UserId) - Initiator
- **NotificationType** (string) - Type: Friend Request, Friend Accepted, Post, Message
- **Message** (string) - Notification message
- **RelatedPostId** (int, FK) - References POSTS(PostId) - Optional
- **CreatedAt** (datetime) - Creation timestamp
- **IsRead** (bool) - Read status

4.3 Relationships

1. **Users** → **Posts**: One-to-Many (User creates multiple posts)
 2. **Users** → **Friendships**: Many-to-Many (Users can have multiple friends)
 3. **Users** → **Messages**: One-to-Many (User sends/receives messages)
 4. **Users** → **Notifications**: One-to-Many (User receives notifications)
 5. **Posts** → **Notifications**: One-to-Many (Post triggers notifications)
-

CHAPTER 5: SYSTEM DIAGRAMS

5.1 Use Case Diagram

The Use Case Diagram illustrates interactions between users and the system, including registration, login, posting, messaging, managing friendships, and administrative tasks.

Actors:

1. **User** - Regular platform user
2. **Admin** - System administrator

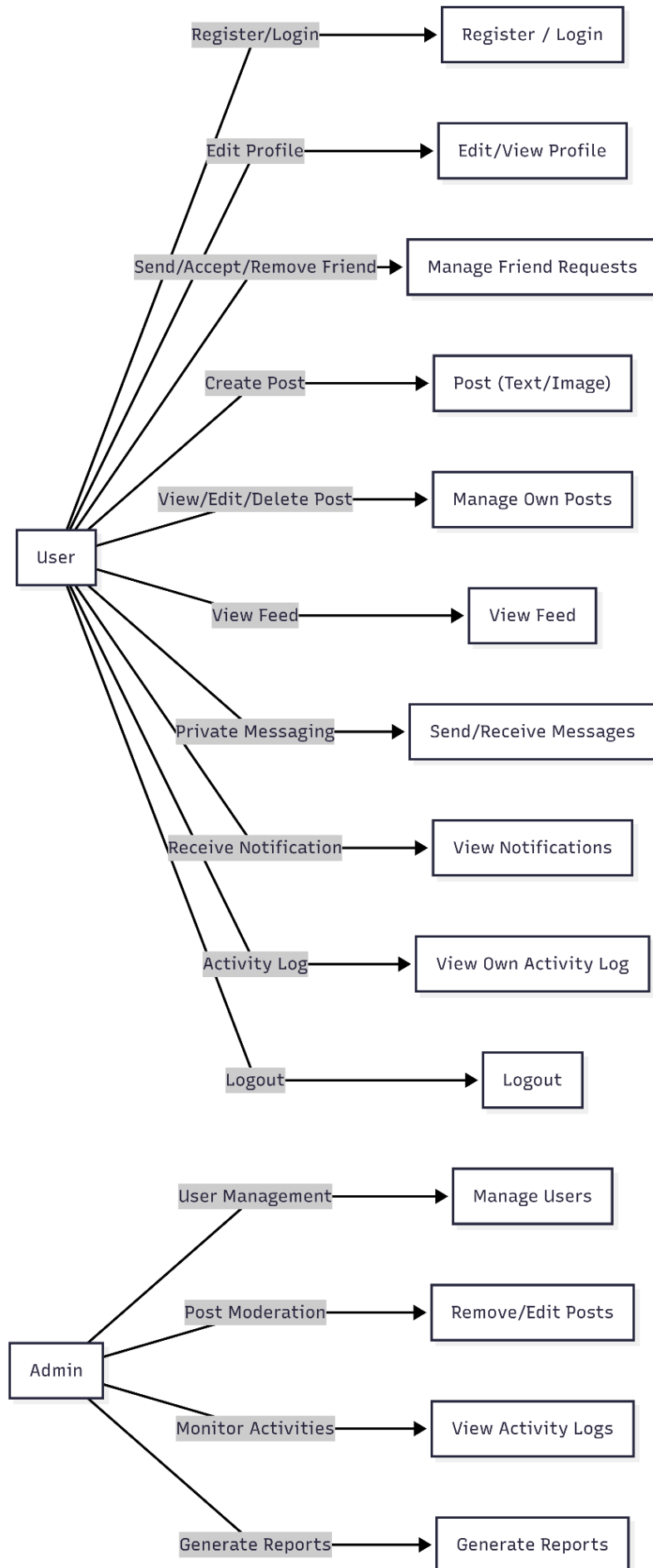
User Use Cases:

- Register/Login
- Edit Profile
- Send/Accept/Remove Friend
- Create Post
- View/Edit/Delete Post

- View Feed
- Private Messaging
- Receive Notification
- Activity Log
- Logout

Admin Use Cases:

- User Management
- Post Moderation
- Monitor Activities
- Generate Reports



5.2 Sequence Diagram

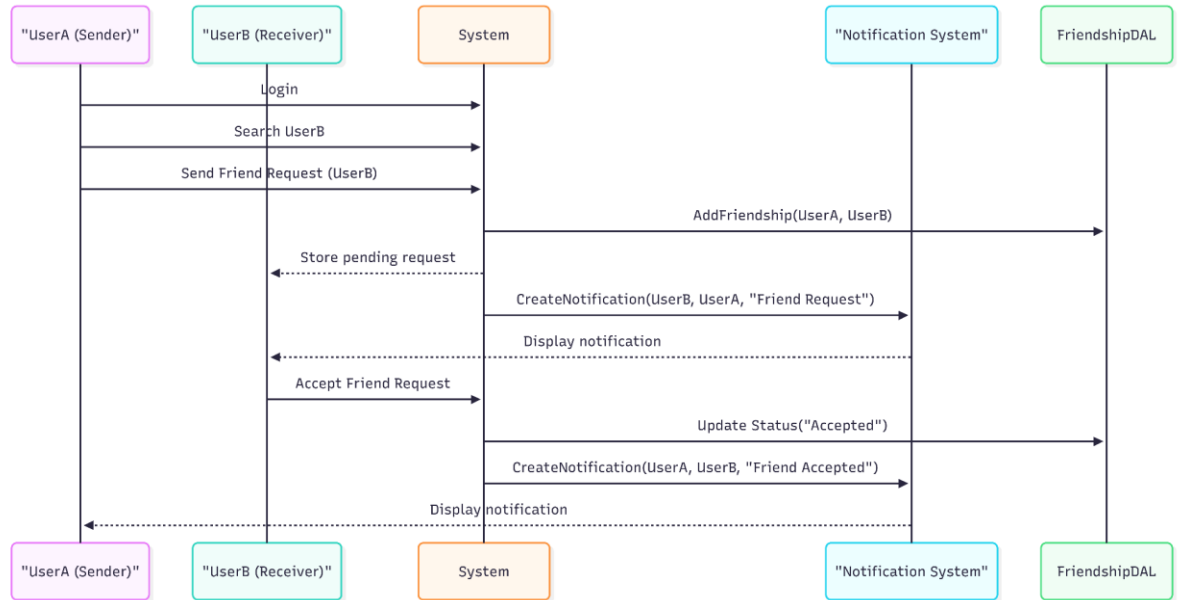
The Sequence Diagram demonstrates the flow of actions involved in sending and accepting friend requests, including notification generation and database updates.

Friend Request Flow:

1. **UserA (Sender)** logs in to the system
2. **UserA** searches for UserB
3. **UserA** sends friend request to UserB
4. **System** calls `AddFriendship(UserA, UserB)` in FriendshipDAL
5. **System** stores pending request in database
6. **System** creates notification for UserB: "Friend Request"
7. **Notification System** displays notification to UserB
8. **UserB (Receiver)** views notification
9. **UserB** accepts friend request
10. **System** calls `Update Status("Accepted")` in FriendshipDAL
11. **System** creates notification for UserA: "Friend Accepted"
12. **Notification System** displays notification to UserA

Components Involved:

- UserA (Sender)
- UserB (Receiver)
- System (Main Application)
- Notification System
- FriendshipDAL (Data Access Layer)



5.3 Entity Relationship Diagram (ERD)

The ERD defines the structural relationships among Users, Posts, Friendships, Messages, and Notifications, ensuring consistency with the implemented database schema.

Entities and Relationships:

USERS (Pink/Purple)

- Primary Key: UserId
- Unique Keys: Username, Email
- Relationships:
 - Makes FRIENDSHIPS
 - Receives FRIENDSHIPS
 - Sends MESSAGES
 - Receives MESSAGES
 - Creates POSTS
 - Receives NOTIFICATIONS

FRIENDSHIPS (Orange)

- Primary Key: FriendshipId
- Foreign Keys: UserId, FriendId
- Attributes: CreatedAt, Status

MESSAGES (Cyan/Blue)

- Primary Key: MessageId
- Foreign Keys: SenderId, ReceiverId
- Attributes: Content, CreatedAt, IsRead

POSTS (Green)

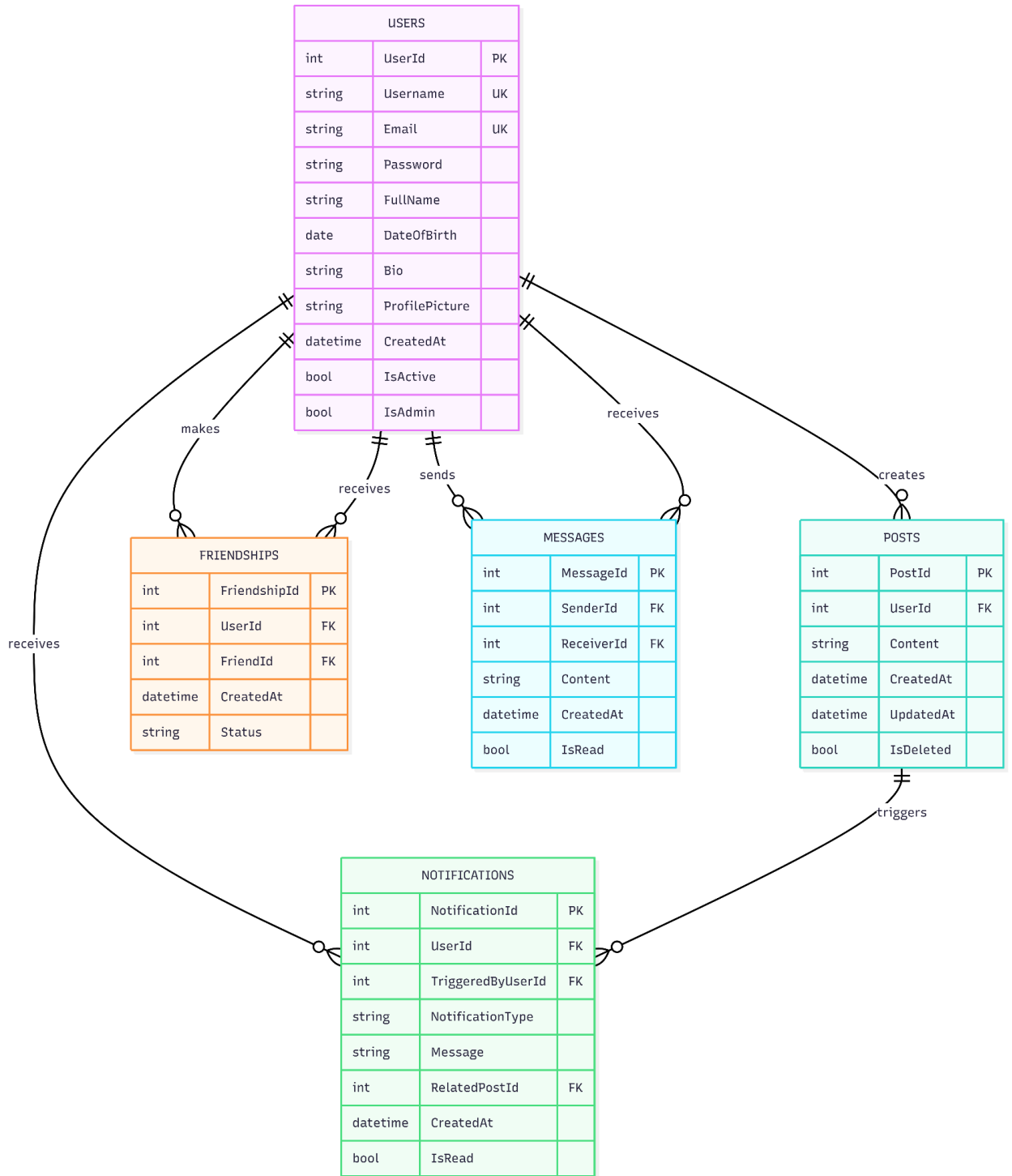
- Primary Key: PostId
- Foreign Key: UserId
- Attributes: Content, CreatedAt, UpdatedAt, IsDeleted
- Triggers NOTIFICATIONS

NOTIFICATIONS (Light Green)

- Primary Key: NotificationId
- Foreign Keys: UserId, TriggeredByUserId, RelatedPostId
- Attributes: NotificationType, Message, CreatedAt, IsRead

Cardinality:

- Users to Friendships: One-to-Many (both sides)
- Users to Messages: One-to-Many (sender and receiver)
- Users to Posts: One-to-Many
- Posts to Notifications: One-to-Many
- Users to Notifications: One-to-Many (receiver)



IMPLEMENTATION DETAILS

6.1 Technology Stack

- **Frontend:** Windows Forms (C#)
- **Backend:** C# .NET Framework
- **Database:** Microsoft SQL Server
- **Architecture:** Three-Tier (Presentation, BLL, DAL)
- **Security:** SHA256 Password Hashing, Parameterized Queries

6.2 Key Features Implemented

1. User Authentication & Authorization

- Secure registration and login
- Role-based access control (User/Admin)
- Session management

2. Social Networking Features

- User profiles with bio and profile pictures
- Post creation, editing, and deletion
- News feed display
- Friend request system (send, accept, reject, remove)

3. Communication

- Private messaging between users
- Real-time notification system
- Activity logging

4. Administration

- User management (activate/deactivate accounts)
- Post moderation (remove inappropriate content)
- Activity monitoring
- Report generation

6.3 Database Operations

Common DAL Methods:

- **Insert()** - Add new records
 - **Update()** - Modify existing records
 - **Delete()** - Remove records (soft/hard delete)
 - **GetById()** - Retrieve single record
 - **GetAll()** - Retrieve all records
 - **Search()** - Find records by criteria
-

TESTING & VALIDATION

7.1 Test Cases

1. **User Registration:** Valid/invalid inputs, duplicate username/email
2. **Login:** Correct/incorrect credentials, inactive accounts
3. **Post Creation:** Text/image posts, empty content validation
4. **Friendship:** Send requests, accept/reject, duplicate requests
5. **Messaging:** Send messages, read status updates
6. **Notifications:** Trigger conditions, display accuracy
7. **Admin Functions:** User management, post moderation, reporting

7.2 Performance Testing

- Database query optimization using indexes
 - Load testing with multiple concurrent users
 - Response time measurements for common operations
-

CONCLUSION

SocialSphere successfully integrates database design principles with practical system implementation. The project delivers a secure, scalable, and well-structured social media platform supported by comprehensive documentation and diagrams.

8.1 Achievements

- ✓ Normalized database design (3NF)
- ✓ Secure authentication and authorization
- ✓ Complete social networking functionality
- ✓ Comprehensive system documentation
- ✓ Clear visual diagrams (Use Case, Sequence, ERD)

8.2 Future Enhancements

- Mobile application development
- Real-time notifications using WebSockets
- Advanced search and filtering
- Post reactions and comments
- Group functionality
- Media file storage optimization
- Analytics dashboard

8.3 Lessons Learned

- Importance of proper database normalization
 - Security considerations in social platforms
 - Effective use of three-tier architecture
 - Value of comprehensive documentation and diagrams
-

A. Database Schema SQL Script

-- Step 1: Create database

```
CREATE DATABASE AdvanceProject;  
GO
```

-- Step 2: Use the database

```
USE AdvanceProject;  
GO
```

-- Step 3: Create Users table (with bracket notation for safety)

```
CREATE TABLE [Users] (  
    UserId INT PRIMARY KEY IDENTITY(1,1),  
    Username NVARCHAR(50) NOT NULL UNIQUE,  
    [Password] NVARCHAR(255) NOT NULL,  
    Email NVARCHAR(100) NOT NULL,  
    FullName NVARCHAR(100),  
    DateOfBirth DATE,  
    Bio NVARCHAR(500),  
    ProfilePicture NVARCHAR(MAX),  
    CreatedAt DATETIME DEFAULT GETDATE(),  
    IsActive BIT DEFAULT 1  
);  
GO
```

-- Step 4: Create Posts table

```
CREATE TABLE Posts (  
    PostId INT PRIMARY KEY IDENTITY(1,1),  
    UserId INT NOT NULL,  
    Content NVARCHAR(MAX) NOT NULL,  
    CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),  
    FOREIGN KEY (UserId) REFERENCES [Users](UserId) ON DELETE CASCADE  
);  
GO
```

-- Step 5: Create Friendships table

```
CREATE TABLE Friendships (  
    FriendshipId INT PRIMARY KEY IDENTITY(1,1),  
    UserId INT NOT NULL,  
    FriendId INT NOT NULL,  
    CreatedAt DATETIME NOT NULL DEFAULT GETDATE(),  
    FOREIGN KEY (UserId) REFERENCES [Users](UserId),  
    FOREIGN KEY (FriendId) REFERENCES [Users](UserId),  
    CONSTRAINT UQ_Friendship UNIQUE (UserId, FriendId)  
);  
GO
```

-- Step 6: Create Messages table

```
CREATE TABLE Messages (  
    MessageId INT PRIMARY KEY IDENTITY(1,1),  
    SenderId INT NOT NULL,  
    ReceiverId INT NOT NULL,  
    Content NVARCHAR(MAX) NOT NULL,  
    CreatedAt DATETIME DEFAULT GETDATE(),  
    IsRead BIT DEFAULT 0,  
    FOREIGN KEY (SenderId) REFERENCES [Users](UserId),  
    FOREIGN KEY (ReceiverId) REFERENCES [Users](UserId)  
);  
GO
```

-- Step 7: Create Notifications table

```
CREATE TABLE Notifications (  
    NotificationId INT PRIMARY KEY IDENTITY(1,1),  
    UserId INT NOT NULL,  
    TriggeredByUserId INT NOT NULL,  
    NotificationType NVARCHAR(50) NOT NULL,  
    Message NVARCHAR(MAX) NOT NULL,  
    RelatedPostId INT,  
    CreatedAt DATETIME DEFAULT GETDATE(),  
    IsRead BIT DEFAULT 0,  
    FOREIGN KEY (UserId) REFERENCES [Users](UserId),  
    FOREIGN KEY (TriggeredByUserId) REFERENCES [Users](UserId),  
    FOREIGN KEY (RelatedPostId) REFERENCES Posts(PostId)  
);  
GO
```

-- Step 8: Create Indexes for better performance

```
CREATE INDEX idx_messages_sender ON Messages(SenderId);  
CREATE INDEX idx_messages_receiver ON Messages(ReceiverId);  
CREATE INDEX idx_notifications_user ON Notifications(UserId);
```

```

CREATE INDEX idx_notifications_read ON Notifications(IsRead);
CREATE INDEX idx_posts_user ON Posts(UserId);
CREATE INDEX idx_friendships_user ON Friendships(UserId);
GO

```

```
-- Step 9: Verify tables created
```

```

SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA
= 'dbo';
GO
USE AdvanceProject;
GO

```

```
-- Insert sample users
```

```

INSERT INTO [Users] (Username, [Password], Email, FullName, DateOfBirth, Bio, CreatedAt,
IsActive)
VALUES
('alex_brown', 'hashed_password_1', 'alex.brown@email.com', 'Alex Brown', '1995-05-15', 'Tech
enthusiast and developer', GETDATE(), 1),
('emily_davis', 'hashed_password_2', 'emily.davis@email.com', 'Emily Davis', '1998-08-22',
'Designer and creative thinker', GETDATE(), 1),
('hadia_masood', 'hashed_password_3', 'hadia.masood@email.com', 'Hadia Masood', '2000-12-
10', 'Student and aspiring developer', GETDATE(), 1),
('john_doe', 'hashed_password_4', 'john.doe@email.com', 'John Doe', '1992-03-18', 'Software
engineer', GETDATE(), 1),
('jane_smith', 'hashed_password_5', 'jane.smith@email.com', 'Jane Smith', '1997-07-25',
'Product manager', GETDATE(), 1),
('mike_jones', 'hashed_password_6', 'mike.jones@email.com', 'Mike Jones', '1994-11-30', 'Data
scientist', GETDATE(), 1),
('noor_khan', 'hashed_password_7', 'noor.khan@email.com', 'Noor Khan', '1999-02-14',
'Graphic designer', GETDATE(), 1),
('sara_ahmed', 'hashed_password_8', 'sara.ahmed@email.com', 'Sara Ahmed', '2001-06-08',
'Content creator', GETDATE(), 1);
GO

```

```
-- Insert sample posts
```

```

INSERT INTO Posts (UserId, Content, CreatedAt)
VALUES
(1, 'Just learned about the new C# 12 features! The collection expressions are game-changing.
☐', DATEADD(HOUR, -2, GETDATE())),
(2, 'Coffee + Code = Productivity ☕☐ What"s your favorite programming setup?',
DATEADD(HOUR, -5, GETDATE())),
(3, 'Just finished designing a new UI for our social platform. Clean, modern, and user-friendly!
☐', DATEADD(HOUR, -8, GETDATE())),

```

```

(4, 'Debugging is like being a detective in a crime movie where you"re also the murderer. ☐',
DATEADD(DAY, -1, GETDATE())),
(5, 'Started my journey with ASP.NET Core. Any tips for beginners? ☐', DATEADD(DAY, -2,
GETDATE())),
(1, 'Web development is evolving so fast! Keeping up with new frameworks is challenging but
exciting.', DATEADD(DAY, -3, GETDATE())),
(6, 'Machine learning models are becoming more accessible. Excited to explore this field!',
DATEADD(DAY, -4, GETDATE())),
(7, 'Design thinking workshop today was amazing! Learned so much about user-centered
design.', DATEADD(DAY, -5, GETDATE()));
GO

```

-- Insert sample friendships

```

INSERT INTO Friendships (UserId, FriendId, CreatedAt)
VALUES
(1, 2, GETDATE()),
(1, 3, GETDATE()),
(1, 4, GETDATE()),
(2, 1, GETDATE()),
(2, 5, GETDATE()),
(3, 1, GETDATE()),
(3, 6, GETDATE()),
(4, 1, GETDATE()),
(4, 7, GETDATE()),
(5, 2, GETDATE()),
(5, 8, GETDATE()),
(6, 3, GETDATE()),
(7, 4, GETDATE()),
(8, 5, GETDATE());
GO

```

-- Insert sample messages

```

INSERT INTO Messages (SenderId, ReceiverId, Content, CreatedAt, IsRead)
VALUES
(1, 2, 'Hey! How are you doing?', DATEADD(MINUTE, -30, GETDATE()), 1),
(2, 1, 'I"m doing great! Just finished a project.', DATEADD(MINUTE, -25, GETDATE()), 1),
(1, 2, 'That"s awesome! Let"s catch up soon.', DATEADD(MINUTE, -20, GETDATE()), 0),
(3, 1, 'Hi Alex! Did you see my new design?', DATEADD(MINUTE, -15, GETDATE()), 1),
(1, 3, 'Yes! It looks amazing. Great work!', DATEADD(MINUTE, -10, GETDATE()), 0),
(4, 1, 'Want to collaborate on a project?', DATEADD(MINUTE, -5, GETDATE()), 0);
GO

```

-- Insert sample notifications

```
INSERT INTO Notifications (UserId, TriggeredByUserId, NotificationType, Message,
RelatedPostId, CreatedAt, IsRead)
VALUES
(1, 2, 'Like', 'liked your post', 1, DATEADD(MINUTE, -45, GETDATE()), 0),
(1, 3, 'Like', 'liked your post', 1, DATEADD(MINUTE, -40, GETDATE()), 0),
(2, 1, 'Comment', 'commented on your post', 2, DATEADD(MINUTE, -35, GETDATE()), 0),
(3, 1, 'FriendRequest', 'sent you a friend request', NULL, DATEADD(MINUTE, -30,
GETDATE()), 1),
(1, 4, 'Like', 'liked your post', 5, DATEADD(MINUTE, -25, GETDATE()), 0),
(4, 1, 'Message', 'sent you a message', NULL, DATEADD(MINUTE, -20, GETDATE()), 0),
(5, 2, 'Like', 'liked your post', 2, DATEADD(MINUTE, -15, GETDATE()), 1),
(2, 5, 'Comment', 'commented on your post', 2, DATEADD(MINUTE, -10, GETDATE()), 0);
GO
```

-- Verify data inserted

```
SELECT 'Users' as TableName, COUNT(*) as RecordCount FROM [Users]
UNION ALL
SELECT 'Posts', COUNT(*) FROM Posts
UNION ALL
SELECT 'Friendships', COUNT(*) FROM Friendships
UNION ALL
SELECT 'Messages', COUNT(*) FROM Messages
UNION ALL
SELECT 'Notifications', COUNT(*) FROM Notifications;
GO
```