# Social media Dashboard Project Documentation

Momen Hesham

Marwan Samy

# 1. Project Planning & Management

## 1.1 Project Proposal

Overview

The Social Media Dashboard is a web-based application designed to centralize and streamline social media management. It enables users to manage multiple social media accounts from a single interface, offering functionalities such as post scheduling, real-time analytics, and engagement tracking. The dashboard is particularly beneficial for businesses, social media managers, and marketing professionals, reducing the time and effort required to manage various platforms separately.

Objectives

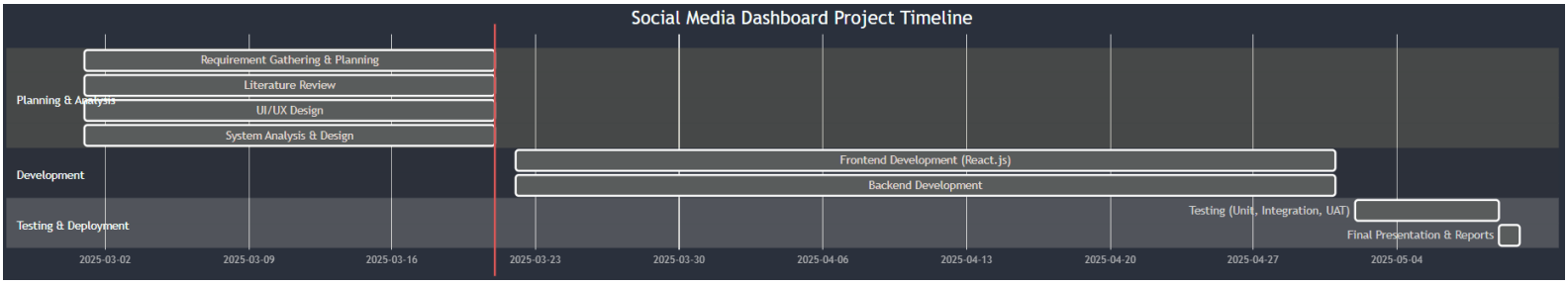
- Develop a user-friendly platform to manage multiple social media accounts.
- Enable users to schedule and automate posts across different platforms.
- Provide real-time analytics, including engagement metrics, impressions, and reach.
- Optimize system performance to ensure quick response times and scalability.
- Ensure compliance with API limitations of various social media platforms.
- Deliver a responsive and accessible UI that works across multiple devices.

Scope

- Multi-account management (Facebook, Twitter, Instagram, LinkedIn integration)
- Post scheduling and automation with a content calendar
- Real-time analytics and reporting
- Dashboard customization with widgets and filters

## 1.2 Project Plan

Timeline & Milestones




Deliverables

By 03/21/2025:

Project Proposal & Requirements:

- A document outlining project objectives, scope, literature review, and system analysis (including key diagrams).

By 05/09/2025

Functional Prototype:

- A working React.js frontend integrated with backend services

- Final Deployment Package and User Guide

Implementation Assets:

- Clean, modular, and well-commented React.js source code with secure practices

- Git repository

## 1.3 Task Assignment & Roles

| Role | Responsibilities |
| --- | --- |
| Momen Hesham | Designs innovative UI components, manages application state, and establishes API connections to deliver a smooth and engaging user interface. |
| Marwan Samy | Develops and refines the UI, handles state coordination, and implements API integrations to maintain effective data flow and consistent user experience. |

## 1.4 Risk Assessment & Mitigation Plan

| | | |
| --- | --- | --- |
| Performance | Slow dashboard load times and poor scalability for 500 users. | Optimize React rendering, use lazy loading, and conduct load tests. |
| Security | Data exposure due to unsecured | Enforce HTTPS, implement JWT, and use strong password encryption. |

| | | |
|---|---|---|
| | communication or weak authentication. | |
| Usability | Poor responsiveness or accessibility issues affecting user experience. | Follow responsive design practices and meet WCAG 2.1 AA standards. |
| Reliability | Downtime or inadequate logging hinders maintenance and debugging. | Use redundant hosting, continuous monitoring, and implement comprehensive logging. |
| Maintainability | Difficulty integrating third-party APIs or supporting future i18n requirements. | Design a modular architecture with clear API documentation and built-in internationalization support. |

## 1.5 Key Performance Indicators (KPIs)

To measure the success of the Social Media Dashboard, we define the following KPIs:

| Metric | Target Value |
|---|---|
| System Uptime | 99.9%+ |
| Frontend Performance | Maintain a React.js Lighthouse performance score of 90+ |

| | |
|---|---|
| Response Time | <2 seconds per request |
| User Engagement | At least 80% of users actively use the dashboard weekly |
| Post Scheduling Accuracy | 100% of scheduled posts are published correctly |
| Security | Zero reported security vulnerabilities post-deployment |

# 3. Requirements Gathering

## 3.1 Stakeholder Analysis

| Stakeholder | Role/Responsibility | Needs & Expectations |
|---|---|---|
| Admin Users | Primary users of the dashboard (e.g., managers, team leads) | - Clear data visualization (charts, tables, KPIs) - Easy navigation - Real-time data updates |
| Developers | Build, maintain, and improve the dashboard | - Clean, modular codebase - Scalable component structure - Documentation for setup and customization |

| Stakeholder | Role/Responsibility | Needs & Expectations |
| --- | --- | --- |
| Project Managers/Executives | Oversee project progress and metrics | - High-level overview of KPIs - Performance reports - Exportable data (PDF, CSV) |
| Security Officers | Ensure data security and compliance | - Secure login/authentication - Role-based access control - Compliance with data protection standards |
| End Clients/Customers | May view certain parts of the dashboard or receive reports | - Accurate reporting - User-friendly interface for limited access |
| UI/UX Designers | Ensure the dashboard is intuitive and user-friendly | - Consistent design system - Responsive layout - Accessibility features |
| System Administrators (IT) | Deploy and manage the infrastructure | - Easy deployment - Minimal downtime - Configurable settings |

## 3.2 User Stories & Use Cases

User Stories:

1. As an Admin User, I want to log in securely, so that only authorized users can access the dashboard.

2. As an Admin User, I want to view key performance metrics in charts and tables, so that I can quickly analyze project status.

3. As an Admin User, I want to filter and sort data, so that I can find specific information easily.

4. As a Project Manager, I want to generate downloadable reports, so that I can share progress updates with stakeholders.

5. As a Developer, I want to easily customize the dashboard components, so that I can add new features when required.

6. As a Security Officer, I want to enforce role-based access control, so that only specific users can access sensitive data.

7. As a System Administrator, I want to deploy the dashboard efficiently, so that it remains stable and reliable.

8. As a Client, I want to view a simplified version of the dashboard, so that I can see relevant information without complexity.

Use Cases:

| Use Case | Description | Actors |
|---|---|---|
| User Login | User logs in using secure authentication credentials. | Admin User, Client |
| View Dashboard Metrics | User views charts, graphs, tables showing project KPIs and performance. | Admin User, Manager |
| Filter & Sort Data | User applies filters and sorting options to datasets for better analysis. | Admin User |
| Export Reports | User generates and downloads reports in formats like PDF or CSV. | Project Manager |
| Manage User Roles | Admin assigns roles and permissions to control access levels. | Admin User, Security Officer |
| Customize Dashboard Components | Developer adds new components (e.g., new chart types, data sources) to meet changing needs. | Developer |

| Use Case | Description | Actors |
| --- | --- | --- |
| Deploy Dashboard | System Administrator deploys and configures the dashboard application. | System Administrator |

## 3.3 Functional Requirements

Requirement Description

1. The system shall provide a secure login page with email/password authentication.

2. The system should implement Role-Based Access Control (RBAC) to define different user permissions.

3. The dashboard shall display key performance indicators (KPIs) using charts, graphs, and tables.

4. Users shall be able to filter, sort, and search data dynamically.

5. The system should allow users to export reports in formats like PDF or CSV.

6. The dashboard shall be responsive, adapting to various screen sizes (desktop, tablet, mobile).

7. The system shall include a sidebar navigation with links to different dashboard sections/modules.

8. The system shall allow themed customization (dark/light modes) for better user experience.

| Requirement Description |
| --- |
| 9. Developers shall be able to easily integrate additional charts, components, or APIs. |
| 10. The system should provide a notifications panel for system alerts or important updates. |

## 3.4 Non-Functional Requirements

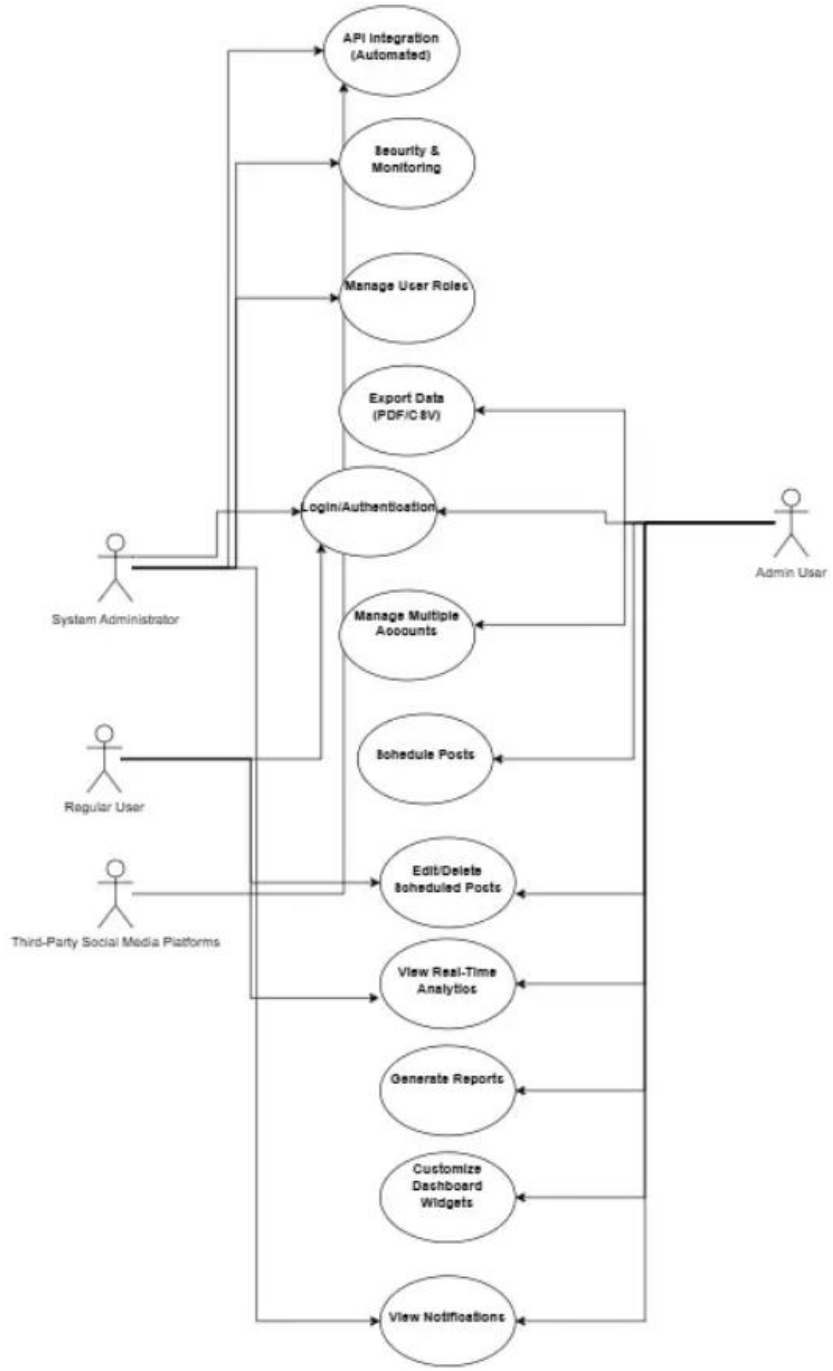
| Requirement Description | Category |
| --- | --- |
| 1. The system shall load the dashboard page in under 3 seconds on a standard broadband connection. | Performance |
| 2. The system shall support up to 500 concurrent users without significant degradation in performance. | Scalability |
| 3. The system shall enforce HTTPS for all communication to ensure secure data transmission. | Security |
| 4. The system shall implement password encryption and secure authentication protocols (e.g., JWT). | Security |
| 5. The dashboard shall be responsive and accessible, meeting WCAG 2.1 AA accessibility standards. | Usability |

| Requirement Description | Category |
| --- | --- |
| 6. The system shall provide 99.9% uptime availability. | Reliability |
| 7. The system shall allow easy integration with third-party APIs or backend services via modular architecture. | Maintainability |
| 8. The system’s UI components shall follow a consistent design system for a clean and intuitive experience. | Usability |
| 9. System logs and errors shall be captured for monitoring and debugging purposes. | Reliability |
| 10. The dashboard shall support multi-language (i18n) support if needed in future iterations. | Extensibility |

# 4. System Analysis & Design

## 4.1 Problem Statement & Objectives

Use case Diagram:



The diagram illustrates a social media management system where two main actors (System Administrator and Admin User) perform key tasks such as user role management, multi-account handling, post scheduling, analytics viewing, data export, and security monitoring. Third-party social media platforms are also integrated to facilitate automated post publishing and data retrieval.

Functional Requirements

These define specific features and system behavior:

| Requirement | Description |
| --- | --- |
| Login/Authentication | Users (Admin, Regular) and System Admin must authenticate securely to access the system. |
| Manage Multiple Accounts | Admin Users can link, view, and manage multiple social media accounts (Facebook, Twitter, Instagram, LinkedIn). |
| Schedule Posts | Admin Users can create and schedule posts across different platforms using a content calendar. |
| Edit/Delete Scheduled Posts | Admin Users can modify or delete already scheduled posts. |
| View Real-Time Analytics | Both Admin and Regular Users can view engagement metrics, impressions, and reach in real time. |

| Requirement | Description |
| --- | --- |
| Generate Reports | Admin Users can generate analytical reports based on social media performance. |
| Customize Dashboard Widgets | Admin Users can rearrange or customize dashboard widgets and filters. |
| View Notifications | Admin Users and System Admins can view system notifications (e.g., API failures, post status). |
| Export Data (PDF/CSV) | Admin Users can export reports and analytics in PDF or CSV format. |
| Manage User Roles | System Administrators can create, edit, and delete user roles and permissions. |
| Security & Monitoring | System Admins monitor system health, detect unauthorized access, and audit logs. |
| API Integration (Automated) | The system integrates with Third-Party social media APIs for data retrieval, post publishing, and account management. |

Non-Functional Requirements

These define system performance, reliability, security, and usability constraints:

| Requirement Category | Description |
| --- | --- |
| Performance | - System must respond within <2 seconds per request.<br>- Must handle concurrent requests without delay. |
| Security | - Apply OAuth2 and multi-factor authentication.<br>- Ensure Zero vulnerabilities post-deployment.<br>- Regular system audits and penetration testing. |
| Reliability/Uptime | - Maintain 99.9%+ system uptime.<br>- Failover mechanisms and database backups in place. |
| Scalability | - System designed to easily add more social media platforms and user accounts without performance degradation. |
| Usability | - UI/UX optimized for ease of use (React-based responsive design).<br>- Accessible across mobile, tablet, and desktop. |

| Requirement Category | Description |
| --- | --- |
| Compatibility | - Support all major browsers and platforms.<br>- Fully responsive design ensuring functionality on multiple screen sizes. |
| Maintainability | - Modular architecture (React components & clean API structure) to ease future updates and debugging. |
| Data Integrity | - Ensure accuracy of post-scheduling and analytics data.<br>- Prevent data loss during API syncs and system updates. |

High-Level Design Overview

1. Frontend (View Layer)

- Technology:
  React.js
- Components:
  - Authentication: Login, Registration
  - Dashboard: Post Scheduler, Analytics Charts, Notifications Panel, Customizable Widgets
  - User Management: Role assignment, account management
- Features:
  - Responsive UI/UX design
  - API consumption via Axios/Fetch for real-time data
  - State management (Redux or Context API)

2. Backend (Controller & Model Layer)

- Technology:
  Node.js with Express.js
- Components:
  - Authentication Module: Secure login, session management
  - API Controllers: Handle requests for post scheduling, analytics retrieval, account management

    - Database Models: User data, scheduled posts, analytics logs, notifications
    - Third-Party API Handlers: Interfaces for Facebook, Twitter, Instagram, LinkedIn APIs
- Features:
    - RESTful API services
    - JWT-based authentication
    - API rate-limit handling & error logging

3. Database (Model Layer)

- Technology:
  MongoDB (NoSQL) or a SQL alternative
- Entities:
    - Users
    - Social Media Accounts
    - Scheduled Posts
    - Analytics Data
    - System Notifications
    - User Roles & Permissions

4. Third-Party Integrations

- APIs:
    - Facebook Graph API
    - Twitter API
    - Instagram API
    - LinkedIn API
- Functions:
    - Fetch real-time analytics
    - Publish scheduled posts
    - Sync user accounts and data
    - Handle API limits and errors gracefully

5. System Interaction Flow

1. User Interaction
    - The user logs in via the frontend.
    - The frontend sends an authentication request.
    - The backend validates and returns a JWT token.
2. Dashboard Usage
    - The Admin User manages accounts and schedules posts.
    - The frontend sends data via API.

    - The backend stores data in the database and triggers the post scheduler.
3. Analytics & Reporting
    - The user requests analytics.
    - The backend fetches data from the database and third-party APIs.
    - Data is returned to the frontend and displayed in charts/widgets.
4. System Admin Actions
    - The System Admin manages user roles.
    - The backend updates roles in the database.
    - The change affects frontend permissions accordingly.

6. Scalability Consideration

- Backend: Stateless design allows horizontal scaling.
- Frontend: Component-based architecture is reusable and easily expandable.
- APIs: Modular design enables smooth integration of additional platforms (e.g., TikTok, YouTube).

## 4.2 Database Design & Data Modeling

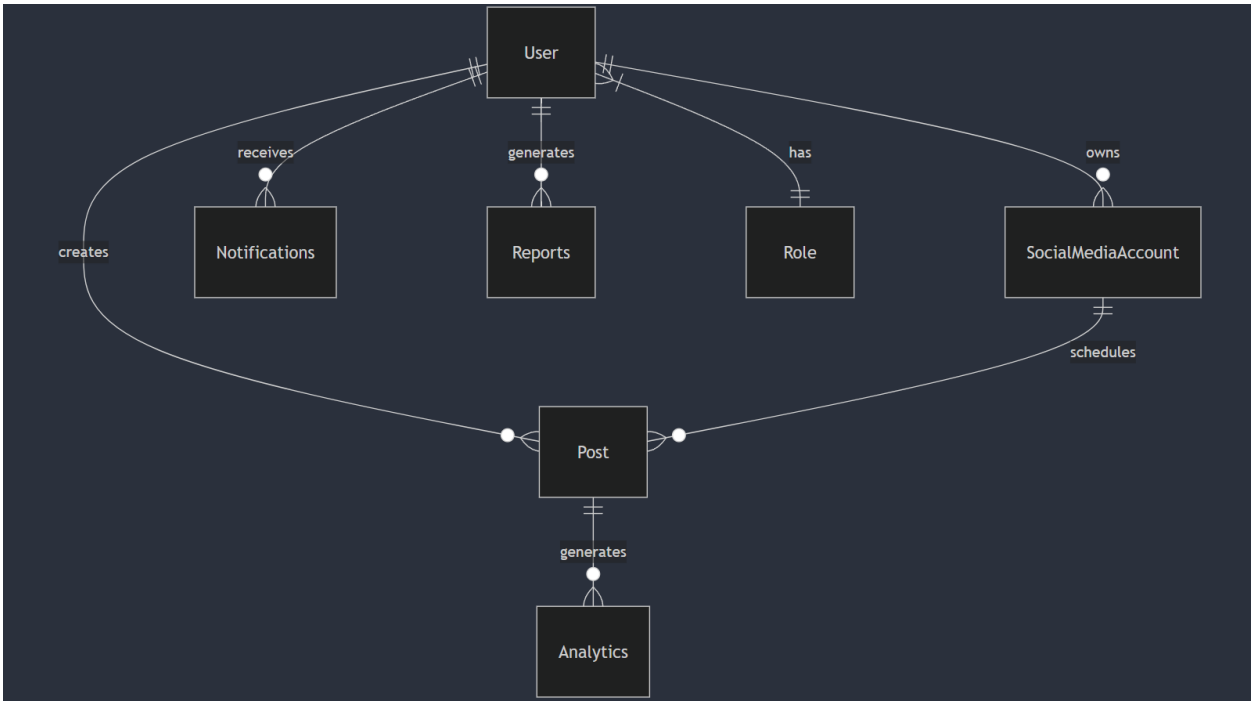
1. Entity-Relationship Diagram (ERD)

The database design for the Social Media Dashboard is structured to efficiently handle user authentication, post scheduling, analytics, and role-based access control. Below is a high-level overview of the key entities and their relationships:

Entities & Relationships:

- User (UserID, Name, Email, PasswordHash, RoleID)
    - One-to-Many: A user can have multiple social media accounts.
    - One-to-Many: A user can schedule multiple posts.
- SocialMediaAccount (AccountID, Platform, Username, AccessToken, UserID)
    - Many-to-One: Each account is linked to a single user.
- Post (PostID, Content, ScheduledTime, Status, AccountID, UserID)
    - Many-to-One: Each post is linked to a specific social media account.
    - Many-to-One: Each post is created by a user.
- Analytics (AnalyticsID, AccountID, PostID, Impressions, Engagement, Clicks, Timestamp)
    - Many-to-One: Analytics are associated with a specific post.

    - Many-to-One: Analytics are linked to a social media account.
- Role (RoleID, RoleName, Permissions)
    - One-to-Many: Each role is assigned to multiple users.
- Notifications (NotificationID, Message, UserID, Timestamp, Status)
    - Many-to-One: Each notification is assigned to a user.
- Reports (ReportID, UserID, GeneratedAt, FilePath)
    - Many-to-One: Reports are generated by users.




2. Logical & Physical Schema

Tables & Attributes

User Table

| Field | Type | Constraints |
| --- | --- | --- |
| UserID | INT | PRIMARY KEY, AUTO_INCREMENT |
| Name | VARCHAR(100) | NOT NULL |
| Email | VARCHAR(255) | UNIQUE, NOT NULL |
| PasswordHash | VARCHAR(255) | NOT NULL |
| RoleID | INT | FOREIGN KEY REFERENCES Role(RoleID) |

SocialMediaAccount Table

| Field | Type | Constraints |
| --- | --- | --- |
| AccountID | INT | PRIMARY KEY, AUTO_INCREMENT |
| Platform | ENUM('Facebook', 'Twitter', 'Instagram', 'LinkedIn') | NOT NULL |
| Username | VARCHAR(255) | NOT NULL |
| AccessToken | TEXT | NOT NULL |
| UserID | INT | FOREIGN KEY REFERENCES User(UserID) |

Post Table

| Field | Type | Constraints |
| --- | --- | --- |
| PostID | INT | PRIMARY KEY, AUTO_INCREMENT |
| Content | TEXT | NOT NULL |
| ScheduledTime | DATETIME | NOT NULL |
| Status | ENUM('Scheduled', 'Published', 'Failed') | DEFAULT 'Scheduled' |
| AccountID | INT | FOREIGN KEY REFERENCES SocialMediaAccount(AccountID) |
| UserID | INT | FOREIGN KEY REFERENCES User(UserID) |

Analytics Table

| Field | Type | Constraints |
| --- | --- | --- |
| AnalyticsID | INT | PRIMARY KEY, AUTO_INCREMENT |
| AccountID | INT | FOREIGN KEY REFERENCES SocialMediaAccount(AccountID) |
| PostID | INT | FOREIGN KEY REFERENCES Post(PostID) |
| Impressions | INT | DEFAULT 0 |

| Field | Type | Constraints |
| --- | --- | --- |
| Engagement | INT | DEFAULT 0 |
| Clicks | INT | DEFAULT 0 |
| Timestamp | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP |

Role Table

| Field | Type | Constraints |
| --- | --- | --- |
| RoleID | INT | PRIMARY KEY, AUTO_INCREMENT |
| RoleName | VARCHAR(50) | UNIQUE, NOT NULL |
| Permissions | TEXT | NOT NULL |

Notifications Table

| Field | Type | Constraints |
| --- | --- | --- |
| NotificationID | INT | PRIMARY KEY, AUTO_INCREMENT |
| Message | TEXT | NOT NULL |
| UserID | INT | FOREIGN KEY REFERENCES User(UserID) |
| Timestamp | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP |
| Status | ENUM('Unread', 'Read') | DEFAULT 'Unread' |

Reports Table

| Field | Type | Constraints |
|---|---|---|
| ReportID | INT | PRIMARY KEY, AUTO_INCREMENT |
| UserID | INT | FOREIGN KEY REFERENCES User(UserID) |
| GeneratedAt | TIMESTAMP | DEFAULT CURRENT_TIMESTAMP |
| FilePath | VARCHAR(255) | NOT NULL |

3. Normalization Considerations

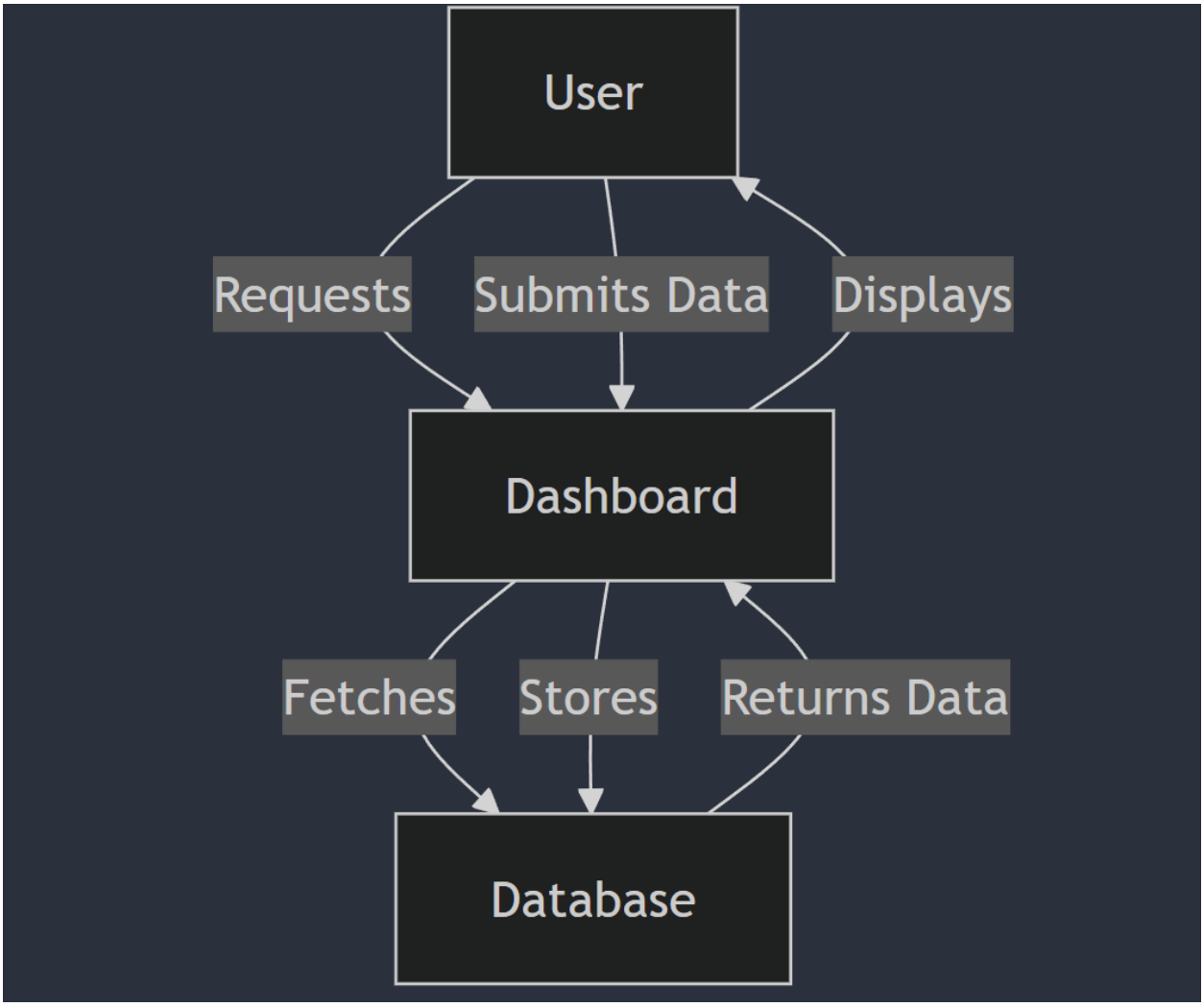
The database follows Third Normal Form (3NF) to ensure data integrity and avoid redundancy:

1. 1NF (Eliminate Repeating Groups)
    - Each table has a unique identifier (primary key), and all attributes contain atomic values.
2. 2NF (Eliminate Partial Dependencies)
    - Non-key attributes fully depend on the primary key.
    - Example: RoleID in the User table is linked to a separate Role table to avoid redundancy.
3. 3NF (Eliminate Transitive Dependencies)
    - All attributes are dependent only on the primary key.
    - Example: User roles and permissions are stored separately to ensure scalability.

## 4.3 Data Flow & System Behavior

Data Flow Diagram (DFD)

Overview:
The DFD offers a high-level view of how data flows between the main components of the system. It is structured at a context level (Level 0) to illustrate the interactions between the user, the dashboard, and the database.



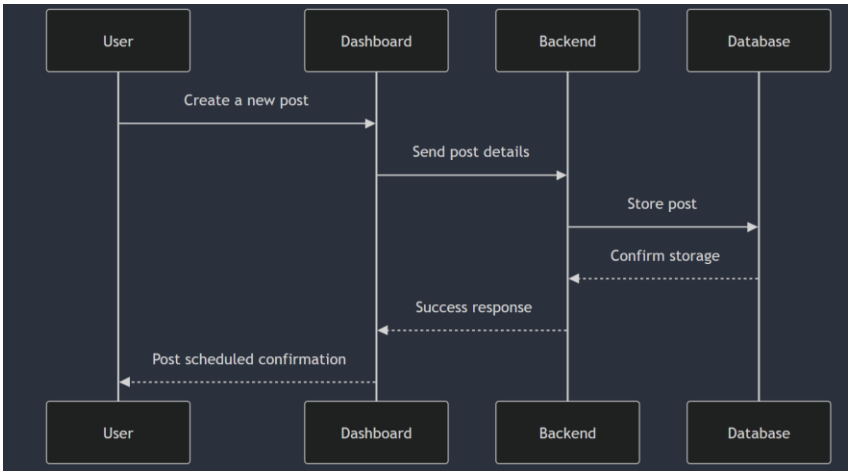
Key Points:

- User Interaction: The user initiates the process by sending requests to the dashboard.
- Dashboard Operations: The dashboard fetches and displays data by communicating with the database.
- Data Movement: The flow includes both requests for data (fetching) and submissions of new or updated data (storing).

Purpose:
To provide a simplified view of the entire system, showing how data is exchanged between the user interface and the backend database, ensuring clarity on where data originates and where it is consumed.

2. Sequence Diagram

Overview:
This diagram details the process flow for a key interaction: scheduling a new post. It shows the sequence of operations and the communication between different system components.



Key Points:

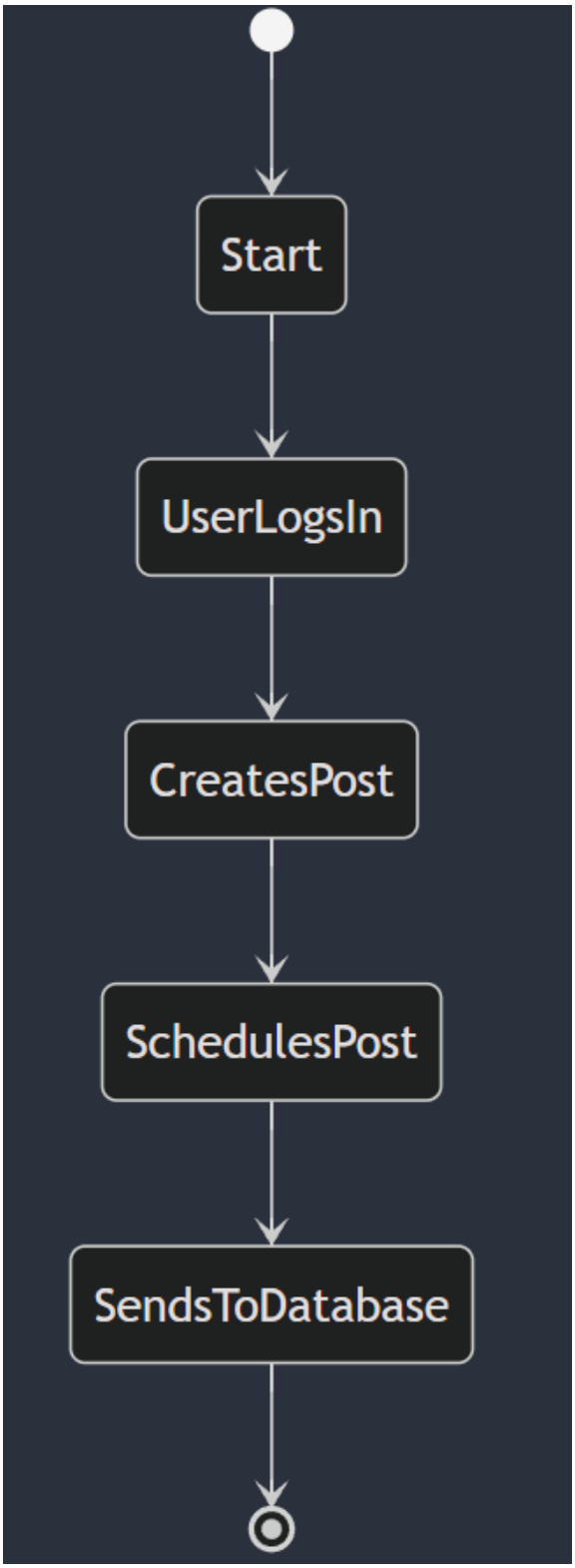
- Participants Involved: The primary actors are the User, Dashboard, Backend, and Database.
- Process Flow:
    1. User Action: The user creates a new post on the dashboard.
    2. Dashboard Request: The dashboard sends the post details to the backend.
    3. Backend Processing: The backend stores the post in the database.
    4. Confirmation: The database confirms the storage, and the backend then relays a success response back to the dashboard.
    5. User Notification: Finally, the dashboard informs the user that the post has been scheduled.

Purpose:
To illustrate the real-time interactions between system components during a post scheduling event, ensuring each component's responsibilities and the order of operations are clear.

3. Activity Diagram

Overview:
This diagram visualizes the workflow for scheduling a post,

capturing the different stages of the process and the transitions between them.



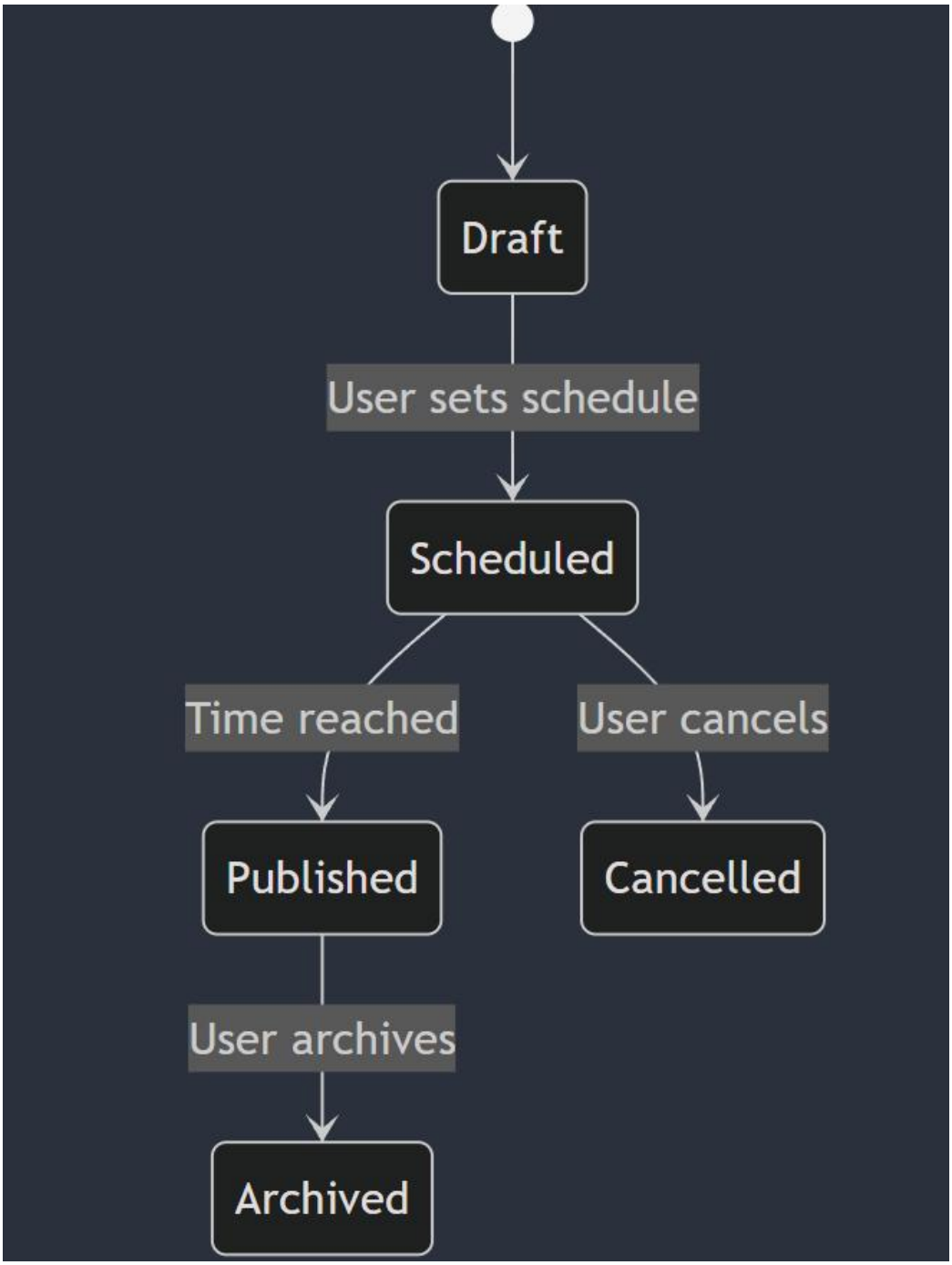
Key Points:

- Workflow Stages:
    - Start: The process begins.
    - User Logs In: The user authenticates to access the dashboard.
    - Creates Post: After logging in, the user drafts a post.
    - Schedules Post: The user then schedules the post.
    - Database Interaction: The scheduled post details are sent and stored in the database.
    - End: The process ends once the data has been stored.

Purpose:
To provide a step-by-step breakdown of the actions and transitions in the post scheduling workflow, helping to understand the user journey and system response at each stage.

4. State Diagram

Overview:
This diagram represents the lifecycle of a post, showing its various states and transitions based on user actions or system events.



Key Points:

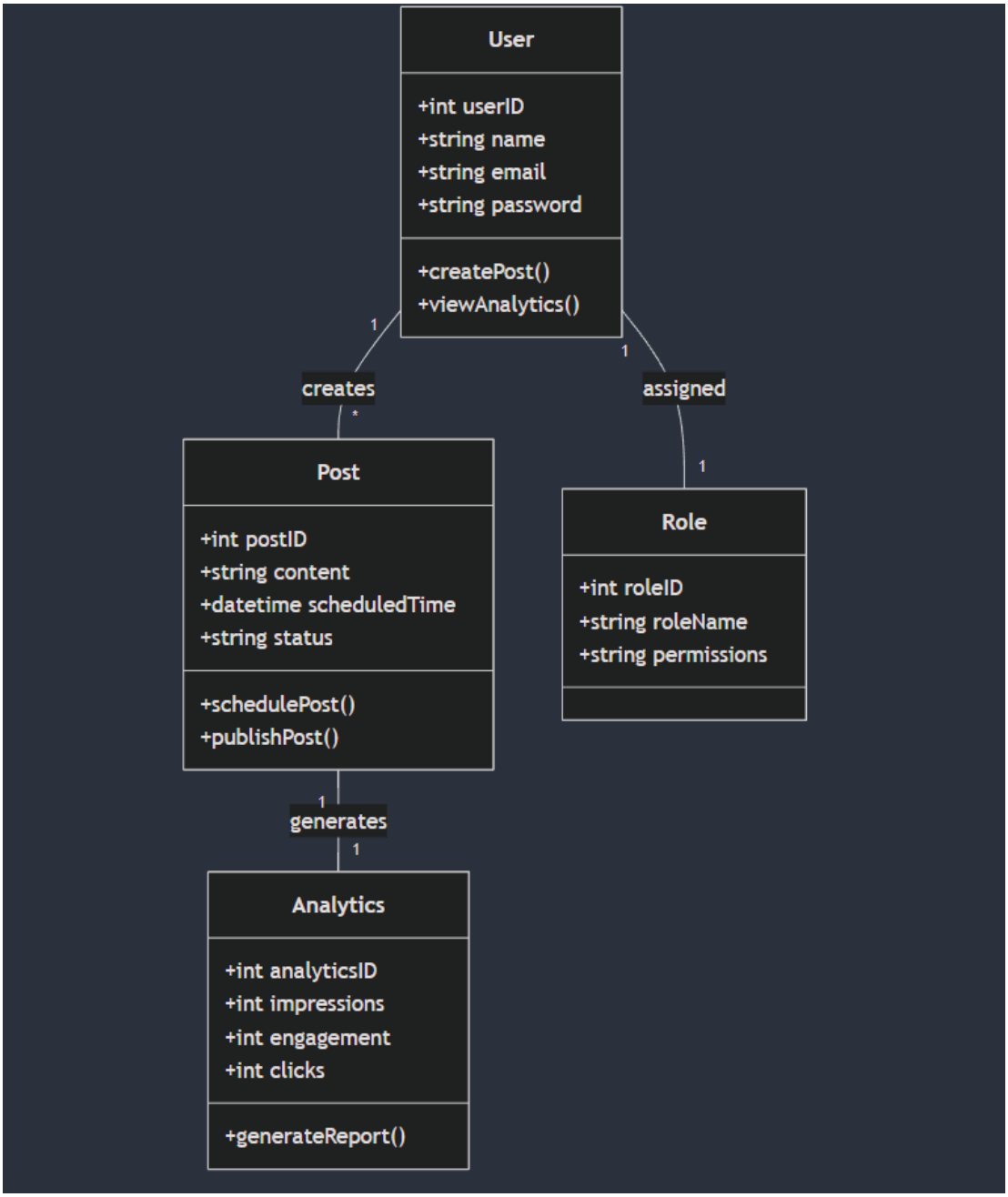
- States of a Post:
    - Draft: The initial state when a post is created.
    - Scheduled: The state after the user sets a schedule.
    - Published: When the scheduled time is reached, the post is published.
    - Cancelled: If the user cancels the scheduled post.
    - Archived: The post may be archived after it is published.
- Transitions:
    - A post moves from Draft to Scheduled when a schedule is set.
    - It then transitions from Scheduled to Published at the scheduled time or can be cancelled.
    - Once published, it may later be archived based on user action.

Purpose:
To track the various states a post goes through in its lifecycle, providing clarity on the conditions under which state transitions occur, ensuring a comprehensive understanding of post management.

5. Class Diagram

Overview:

The class diagram defines the structure of the system by outlining the main classes (or entities), their attributes, methods, and the relationships between them.



Key Points:

- Main Classes:
    - User: Contains attributes like userID, name, email, password; includes methods for creating posts and viewing analytics.
    - Post: Contains post details (postID, content, scheduledTime, status) and methods for scheduling and publishing posts.
    - Analytics: Holds data such as impressions, engagement, and clicks, along with methods to generate reports.
    - Role: Defines user roles with attributes like roleID, roleName, and permissions.
- Relationships:
    - A User can create multiple Posts.
    - Each Post is associated with one Analytics record.
    - A User is assigned one Role.

Purpose:
To outline the system's structural framework, showing how classes interact and the responsibilities of each class, facilitating clear system design and guiding developers during implementation.

# 4.4 UI/UX Design & Prototyping

## 1. Wireframes & Mockups


- Layout:
    - Header: Displays user info (photo, name) on the left and a “Download Reports” button on the right.
    - Sidebar: Vertical menu with icons and text links; user avatar at the top.
    - Main Content:
        - Top row of KPI cards (e.g., Total Revenue, Sales).
        - Central area with line chart (revenue trends), bar chart (sales quantity), pie chart (campaign performance).
        - Side widget for recent transactions and a world map for traffic insights.

- Dark Theme:
    - Dark navy or charcoal background, bright text/accents for contrast.

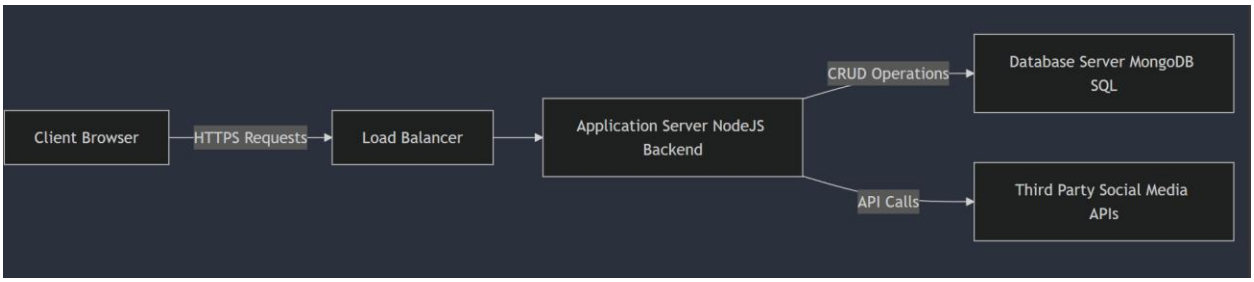
2. UI/UX Guidelines

- Design Principles:
    - Consistency in card/charts styling and spacing.
    - Clear visual hierarchy with large KPI numbers and concise labels.
- Color Scheme:
    - Dark background (#1E1F26 or similar).
    - Light text (#FFFFFF or #D3D3D3).
    - Accent colors for charts/buttons (teal/blue).
- Typography:
    - Sans-serif font (e.g., Roboto).
    - Bold for headers/KPI values, regular for body text.
- Accessibility:
    - High contrast for readability.
    - Clear focus states on interactive elements.

## 4.5 System Deployment & Integration

1. Technology Stack
    1. Backend:
        - Node.js (Express.js) for server-side logic and RESTful APIs.
        - JWT for secure authentication.
        - Integration with social media APIs (Facebook, Twitter, Instagram, LinkedIn) for post scheduling and analytics.
    2. Frontend:
        - React.js for building responsive, component-based user interfaces.
        - Axios/Fetch for API calls and real-time data fetching.
    3. Database:
        - MongoDB (NoSQL) or SQL alternative (e.g., MySQL or PostgreSQL) for storing user data, posts, analytics, and logs.
        - Ensures fast read/write operations and easy scalability.
    4. Additional Tools:
        - Docker for containerization (optional).
        - Git for version control and collaborative development.

2. Deployment Diagram (High-Level)



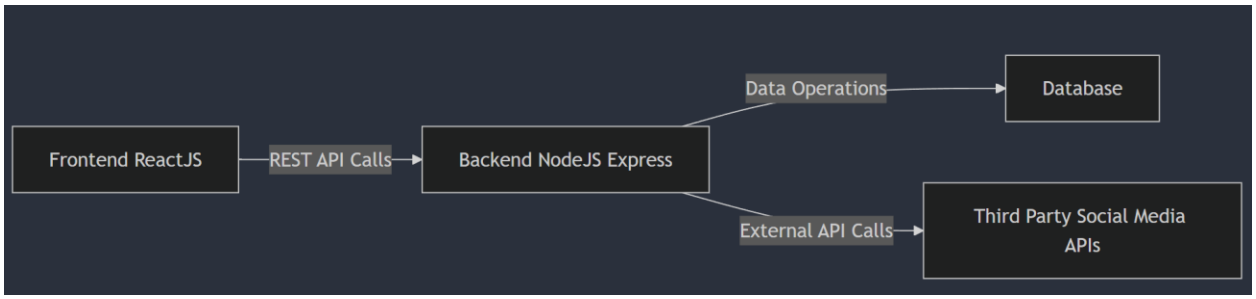

A typical deployment involves:

1. Client (Browser)
    - Runs the React.js application.
    - Communicates with the backend via HTTPS.
2. Application Server
    - Hosts Node.js (Express.js) backend.
    - Manages API endpoints for user authentication, post scheduling, analytics retrieval, etc.
    - Integrates with third-party social media APIs.
3. Database Server
    - Hosts the database (MongoDB/SQL).
    - Stores user accounts, scheduled posts, analytics logs, and notifications.
4. Optional Load Balancer
    - Distributes incoming requests across multiple application servers.

- Enhance performance and reliability (useful for high-traffic scenarios).

Summary: The client accesses the Node.js backend, which in turn reads/writes data to the database server. The backend also communicates with social media APIs for posting and data retrieval.

4. Component Diagram (High-Level)




1. Frontend (React.js)
    - UI Components (Dashboard, Charts, Post Scheduler, User Management).
    - State Management (Redux/Context) for handling global state.
    - API Layer for sending requests to the backend.
2. Backend (Node.js + Express.js)
    - Controllers (e.g., AuthController, PostController, AnalyticsController).
    - Services (e.g., SocialMediaAPI Service for Facebook/Twitter integrations).
    - Models (e.g., User, Post, Analytics).

3. Database
    - Collections/Tables for users, posts, analytics, notifications, roles.
    - ORM/ODM (Mongoose for MongoDB or Sequelize for SQL) handling data operations.
4. Third-Party APIs
    - Social media APIs (Facebook Graph, Twitter API, etc.) for posting and retrieving engagement data.

Relationships:

- The Frontend calls Controllers on the Backend via REST endpoints.
- Controllers use Services to handle logic and communicate with the Database or Third-Party APIs.
- Database stores all essential data (users, posts, analytics logs, etc.).