### **System Architecture Document**

**Project Title**: **FriendsNest**

**Document Version**: 1.0  
**Date**: 09/08/2024  
**Author**: Damandeep

### **1. Introduction**

#### **1.1 Purpose**

This document provides a comprehensive overview of the system architecture for the "FriendsNest" social media application. It describes the components, their interactions, and how they work together to achieve the application's objectives.

#### **1.2 Scope**

The document covers the high-level architecture of the system, including the frontend, backend, database, and external services. It outlines how these components interact, data flow, and key design considerations.

#### **1.3 Definitions, Acronyms, and Abbreviations**

* **API**: Application Programming Interface
* **OAuth**: Open Authorization protocol
* **REST**: Representational State Transfer
* **WebSocket**: A protocol for real-time communication
* **WebRTC**: Web Real-Time Communication
* **CDN**: Content Delivery Network

#### **1.4 References**

* NestJS Documentation
* React.js Documentation
* PostgreSQL Documentation
* WebSocket and WebRTC Specifications

### **2. System Overview**

The "FriendsNest" application is a social media platform that allows users to register, create and interact with posts, send friend requests, and engage in real-time chat. The application also supports media streaming using WebRTC.

#### **2.1 High-Level Architecture Diagram**

*(Include a diagram showing the main components: Frontend, Backend, Database, External Services)*

### **3. Component Description**

#### **3.1 Frontend**

* **Technologies**: React.js, Redux
* **Components**:
  + **User Interface**: Handles the display and interaction of user-facing elements (e.g., registration forms, post feeds, chat interface).
  + **State Management**: Uses Redux for managing the application state across components.
  + **API Integration**: Communicates with backend APIs using Axios or Fetch.
  + **Real-Time Communication**: Uses WebSocket for real-time chat functionality.
  + **Media Streaming**: Integrates with WebRTC for streaming video and photos in posts.

#### **3.2 Backend**

* **Technologies**: NestJS
* **Components**:
  + **API Gateway**: Manages API requests, authentication, and routing.
  + **Authentication Service**: Handles user registration, login, and OAuth authentication.
  + **Post Management Service**: Manages creation, retrieval, updating, and deletion of posts.
  + **Friendship Management Service**: Manages friend requests and relationships.
  + **Chat Service**: Manages real-time messaging using WebSocket.
  + **Media Service**: Handles media uploads, streaming, and management.
  + **Notification Service**: Manages user notifications for interactions and updates.

#### **3.3 Database**

* **Technology**: PostgreSQL
* **Components**:
  + **Users Table**: Stores user information, including credentials and profile data.
  + **Posts Table**: Stores posts, including content and media URLs.
  + **Comments Table**: Stores comments on posts.
  + **FriendRequests Table**: Manages friend request statuses and relationships.
  + **Messages Table**: Stores chat messages between users.
  + **Indexes**: Optimizes query performance for frequently accessed data.

#### **3.4 External Services**

* **OAuth Providers**: Used for authentication (e.g., Google, Facebook).
* **CDN**: For delivering media content efficiently (optional, depending on scale and media handling).

### **4. Component Interaction**

#### **4.1 Frontend to Backend**

* **API Requests**: The frontend makes HTTP requests to backend APIs for user registration, login, post creation, and more.
* **Real-Time Communication**: WebSocket connections are established for real-time chat.

#### **4.2 Backend to Database**

* **Data Storage**: Backend services interact with PostgreSQL to store and retrieve data related to users, posts, comments, and messages.
* **Data Integrity**: Transactions and constraints ensure data consistency and integrity.

#### **4.3 Backend to External Services**

* **OAuth**: The backend integrates with third-party OAuth providers to handle user authentication.
* **Media Streaming**: If using a CDN or media service, the backend manages interactions for uploading and streaming media.

#### **4.4 Real-Time Communication**

* **WebSocket Server**: Handles real-time messaging and updates for chat and notifications.
* **WebRTC**: Manages peer-to-peer connections for media streaming.

### **5. Data Flow**

#### **5.1 User Registration**

1. User submits registration details through the frontend.
2. Frontend sends a request to the backend API.
3. Backend validates the data, creates a new user in PostgreSQL, and sends a confirmation email.
4. User confirms their email, and backend updates the user status.

#### **5.2 Post Creation**

1. User creates a post through the frontend interface.
2. Frontend sends the post data to the backend API.
3. Backend stores the post in PostgreSQL and updates the user's feed.
4. Other users' feeds are updated via API polling or real-time updates.

#### **5.3 Real-Time Chat**

1. User sends a message through the chat interface.
2. Frontend sends the message to the WebSocket server.
3. WebSocket server relays the message to the recipient in real time.
4. Both sender and recipient see the message instantly.

#### **5.4 Media Streaming**

1. User uploads a video/photo through the frontend.
2. Frontend sends the media to the backend for processing and storage.
3. Backend stores the media in a suitable storage solution (e.g., cloud storage, CDN).
4. Media is streamed to users using WebRTC.

### **6. Security Considerations**

* **Authentication**: OAuth 2.0 for secure user authentication.
* **Data Encryption**: Use TLS for data in transit and AES-256 for data at rest.
* **Authorization**: Implement role-based access control (RBAC) for administrative functions.
* **Input Validation**: Sanitise and validate all user inputs to prevent security vulnerabilities (e.g., SQL injection, XSS).

### **7. Scalability and Performance**

* **Load Balancing**: Use load balancers to distribute traffic across multiple servers.
* **Horizontal Scaling**: Scale backend services and database instances horizontally to handle increased load.
* **Caching**: Implement caching strategies for frequently accessed data to reduce load on the database.

### **8. Deployment and Maintenance**

* **Deployment Strategy**: Use continuous integration/continuous deployment (CI/CD) pipelines for automated deployment.
* **Monitoring**: Implement monitoring and logging to track system performance and detect issues.
* **Backup**: Regularly back up the database and application data to prevent data loss.

**Conclusion**This System Architecture Document outlines the high-level structure of the "FriendsNest" application, detailing how the frontend, backend, database, and external services interact. It provides a clear blueprint for the development and deployment of the system.