CLYNC App Architecture Overview

This document provides an overview of the architecture, technologies, and structure of the CLYNC app developed with NodeJS (Express), PostgreSQL databases and Azure Blob Storage.

Technologies and Infrastructure

1. Backend Framework: Node.js (Express)
2. ORM: Prisma ORM
3. Database: PostgreSQL
   - Two databases are used for the microservices:
     - Database 1: For Social and Admin services
     - Database 2: For Financial service
4. Storage: Azure Blob Storage for media files
5. API Gateway: Node.js API Gateway for routing requests
6. Container Orchestration: Kubernetes for managing microservices
7. Architecture: Microservices

Microservices Overview

The application consists of four independent microservices.

1. **Social Service:** Handles social module functionalities.
2. **Financial Service:** Manages financial-related features (Planning Pockets).
3. **Flagship Service:** Integrates third-party (Flagship) financial APIs .
4. **Admin Service:** Manages the APIs for the admin portal.

API Gateway and Request Flow

A Node.js API Gateway is utilized to route incoming requests to the appropriate microservices running in a Kubernetes (k8s) cluster. The gateway handles the incoming routing to ensure efficient and secure communication between the client and services.

Directory Structure

Each microservice follows a consistent directory structure to promote modularity and maintainability. The directory layout is as follows:

- index.js: Entry file for each service
- routes: API route definitions
- controllers: Controllers for handling request logic
- services: Business logic implementations
- utils: Utility functions

- helpers: Helper functions and reusable code
- middleware: Middleware functions
- validation: Request validation logic

Cross-Database Communication

Since separate databases are used for different modules, cross-database communication is facilitated via API calls. Data from different databases is retrieved through API endpoints, and results are joined and processed at the code level.

Prerequisites

- **Node.js**: Version 20.x
- **PostgreSQL**: Ensure the database is set up and accessible
- **Docker and Kubernetes** (if running in a containerized environment)
- **Azure Blob Storage**: For media file handling
- **Environment Variables**: Set up environment variables in .env files (refer to environments/.env.development for development)

Installation

1. Clone the repository:
   git clone <repository_url>
2. Install dependencies:
   npm install
3. Generate Prisma client:
   npm run prisma:generate

Running the Project

Run the following command to start the application in development mode (with nodemon):

npm run dev

To start the application in production mode, use the following command:

npm run start

Example Workflow

For example, when a user makes a request to view their social profile:

1. The client sends a request to the API Gateway.
2. The API Gateway verifies the request (e.g., checking for authentication tokens).
3. The gateway routes the request to the Social Service within the k8s cluster.
4. The Social Service accesses the PostgreSQL database to retrieve the required user information.

5. The Social Service responds to the gateway with the data.
6. The API Gateway forwards the response to the client.