Comprehensive Technical Documentation

Project Overview

This project implements a scalable and modular microservices-based architecture for a Ride-Sharing platform. The application includes distinct services: Authentication (`auth-service`), User Management (`user-service`), Driver Management (`driver-service`), and Trip Management (`trip-service`). An API Gateway manages and routes requests to each microservice, facilitating seamless inter-service communication.

Microservice Architecture

1. Auth Service

Purpose: Manages authentication and authorization using JWT and OTP via Vonage SDK.

Key Functionalities:

- User Signup
- Login with password or OTP
- OTP verification

Technologies Used:

- Node.js, Express.js
- PostgreSQL
- bcrypt, JWT
- Vonage SDK

2. Driver Service

Purpose: Handles operations related to driver profiles and availability.

Key Functionalities:

- Create driver profiles
- Update driver status and service type
- Retrieve available drivers based on criteria

Technologies Used:

- Node.js, Express.js
- PostgreSQL

3. Trip Service

Purpose: Manages trip lifecycle including creation, assignment to drivers, and completion.

Key Functionalities:

- Trip creation by users
- Trip assignment to drivers by admin
- Trip completion marking
- Interacts with Driver Service to assign available drivers

Technologies Used:

- Node.js, Express.js
- PostgreSQL
- Axios/Fetch for external HTTP requests

4. User Service

Purpose: Manages user profile information and roles.

Key Functionalities:

- User profile creation and updates
- Role management

Technologies Used:

- Node.js, Express.js
- PostgreSQL

Directory Structure

Each service follows a structured pattern:

```
<service-name>/
├── controllers/   (business logic)
├── routes/        (API endpoints)
├── middleware/    (authentication & authorization)
├── utils/         (utility functions)
├── db.js          (database connection)
├── server.js      (entry point)
├── .env           (environment variables)
└── package.json   (dependencies & scripts)
```

API Gateway

Purpose: Routes API requests from clients to appropriate microservices.

Routing Example:

- `/api/auth` → Auth Service
- `/api/drivers` → Driver Service
- `/api/trip` → Trip Service
- `/api/users` → User Service

Technologies Used:

- Node.js, Express.js
- `http-proxy-middleware` for proxy routing

Environment Variables & Configuration

Each service uses a `.env` file to store configurations:

Example (`auth-service/.env`):

```
PORT=4001
DB_HOST=localhost
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=your_password
DB_NAME=userdb
JWT_SECRET=your_jwt_secret
JWT_EXPIRATION=1h
VONAGE_API_KEY=your_vonage_key
VONAGE_API_SECRET=your_vonage_secret
DRIVER_SERVICE_URL=http://localhost:8000/api/drivers
```

Setup & Running Instructions

Initial Setup:

```
npm install
```

Run Individual Services:

```
cd <service-name>
node server.js
```

Run API Gateway:

```
cd api-gateway
node server.js
```

API Usage Examples

Authentication (Login):

```
POST http://localhost:8000/api/auth/login
{
  "phone": "999",
  "password": "111"
}
```

Trip Assignment:

```
PUT http://localhost:8000/api/trip/<tripId>/assign
Authorization: Bearer <token>
```

Driver Availability:

```
GET http://localhost:8000/api/drivers/available?type=taxi
Authorization: Bearer <token>
```

Inter-Service Communication

- Trip Service queries Driver Service via API Gateway.
- Authentication tokens (JWT) used across services for security.

Security & Middleware

- JWT tokens verify users and roles.
- Middleware ensures endpoint security:
- `authenticateToken`: Validates JWT.
- `authorizeRole`: Restricts endpoints based on roles (`user`, `admin`).

Technologies & Tools Overview

- **Node.js & Express.js:** Server-side runtime and framework.
- **PostgreSQL:** Database for structured data storage.
- **JWT:** Secure token-based authentication.
- **Axios/Fetch:** HTTP client for inter-service requests.
- **Vonage SDK:** OTP management via SMS.
- **http-proxy-middleware:** Efficient request routing in API Gateway.

Best Practices

- Separation of concerns
- Secure handling of sensitive information (JWT, environment variables)
- Error handling and logging for easy debugging
- Clear inter-service communication strategies

This detailed technical documentation provides comprehensive coverage of your microservices project, including setup, configuration, and usage guidelines.