Logistic Application Backend

A microservice-based backend system for the Logistic Application built with .NET 8, PostgreSQL, Kafka, and Hangfire.

Architecture

Microservices

- **API Gateway** (Port 5000) - Ocelot-based routing and load balancing
- **User Service** (Port 5001) - Authentication, user management
- **Item Service** (Port 5002) - Item tracking, batch management, analytics
- **Delivery Service** (Port 5003) - Delivery management, rider operations, hubs
- **Notification Service** (Port 5004) - SMS, Email, Push notifications

Infrastructure

- **PostgreSQL** - Primary database (separate DB per service)
- **Apache Kafka** - Message broker for inter-service communication
- **Hangfire** - Background job processing
- **Docker** - Containerization

Nigerian-Focused Features

SMS Service

- **Termii API** - Popular SMS gateway in Nigeria
- Supports all Nigerian networks (MTN, Airtel, Glo, 9mobile)

Email Service

- **MailKit/SMTP** - Reliable email delivery
- Configured for Gmail SMTP (easily configurable)

Push Notifications

- **Firebase Cloud Messaging** - Cross-platform push notifications

API Endpoints

Authentication

- `POST /api/auth/login` - User login
- `POST /api/auth/register` - User registration

Users

- `GET /api/users` - Get all users
- `GET /api/users/{id}` - Get user by ID

Items

- `GET /api/items` - Get items (with filtering)
- `POST /api/items` - Create item
- `PUT /api/items/{id}/status` - Update item status

Batches

- `GET /api/batches` - Get all batches
- `POST /api/batches` - Create batch with items

Deliveries

- `GET /api/deliveries` - Get deliveries
- `POST /api/deliveries/assign` - Assign delivery to rider
- `PUT /api/deliveries/{id}/pickup` - Mark as picked up
- `PUT /api/deliveries/{id}/deliver` - Mark as delivered with POD

Riders

- `GET /api/riders` - Get all riders
- `GET /api/riders/available` - Get available riders
- `POST /api/riders` - Create rider

Hubs

- `GET /api/hubs` - Get all hubs
- `POST /api/hubs` - Create hub

Analytics

- `GET /api/analytics/dashboard` - Get dashboard statistics

Notifications

- `POST /api/notifications/sms` - Send SMS
- `POST /api/notifications/email` - Send email
- `POST /api/notifications/push` - Send push notification

Setup Instructions

Prerequisites

- Docker Desktop
- .NET 8 SDK (for local development)

Quick Start

1. Clone the repository
2. Navigate to the backend directory
3. Run the setup script:

   ```
   scripts/start-services.bat
   ```

Manual Setup

1. Start infrastructure:

   ```
   docker-compose up -d postgres zookeeper kafka
   ```

2. Build and start services:

   ```
   docker-compose up -d
   ```

Configuration

Database

- Default PostgreSQL credentials: `postgres/postgres`
- Databases are auto-created: `logistic_users`, `logistic_items`, `logistic_deliveries`

Kafka Topics

Auto-created topics:

- `item-created`
- `item-status-changed`
- `delivery-assigned`
- `batch-uploaded`

Notification Services

Update `appsettings.json` in NotificationService:

- **Termii**: Add your API key from termii.com
- **Email**: Configure SMTP settings
- **Firebase**: Add service account JSON file path

Development

Local Development

Each service can be run independently:

```
cd src/UserService
dotnet run
```

Database Migrations

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Monitoring

- **Hangfire Dashboard**: http://localhost:5002/hangfire
- **Swagger UI**: Available on each service port
- **Docker Logs**: `docker-compose logs -f [service-name]`

Frontend Integration

The API Gateway at `http://localhost:5000` provides all endpoints needed by the React frontend. CORS is configured to allow requests from:

- `http://localhost:5173` (Vite dev server)
- `http://localhost:3000` (Create React App)

Production Deployment

1. Update connection strings for production databases
2. Configure proper Kafka cluster
3. Set up SSL/TLS certificates
4. Configure proper authentication secrets
5. Set up monitoring and logging

6. Configure auto-scaling for services

Security Features

- JWT-based authentication
- Password hashing with BCrypt
- CORS protection
- Input validation
- Audit logging
- Rate limiting (can be added via API Gateway)

Scalability

- Horizontal scaling via Docker containers
- Database per service pattern
- Event-driven architecture with Kafka
- Background job processing with Hangfire
- Load balancing via API Gateway