eWallet Monorepo Project

This repository contains the complete source code for the eWallet platform, structured as a monorepo containing both **Spring Boot** microservices and **Angular** frontend applications.

Architecture Overview

The system is built on a microservices architecture using Spring Cloud:

Backend (Spring Boot)

Service	Folder	Port (Local)	Port (Dev)	Port (Prod)	Description
Service Registry	`serviceRegistry`	8761	9761	8761	Eureka Server for service discovery.
Config Server	`cloudConfigServer`	8296	9296	8296	Centralized configuration management backed by git.
Java Server	`ewallet`	8082	9082	8082	Core business logic and API server.
Java Admin	`EwalletAdmin`	8081	9081	8081	Administration backend API.
Gateway	`gateway`	8080	9080	8080	API Gateway handling routing and filtering.

Frontend (Angular)

Application	Folder	Port (Local)	Port (Dev)	Port (Prod)	Description
Angular Server	`ewallet-angular-server`	4200	9200	4200	User-facing web application.
Angular Admin	`ewallet-angular-admin`	4300	9300	4300	Back-office administration panel.

Infrastructure & Monitoring

Service	Technology	Port (Local/Std)	Description
Loki	Grafana Loki	3100	Log aggregation system.
Zipkin	Zipkin	9411	Distributed tracing.
Prometheus	Prometheus	9090	Metrics collection.
Grafana	Grafana	3000	Visualization dashboard.

Workflows & Scenarios

We have established three distinct workflows using dedicated Docker Compose files.

1. Local Development (Full Stack)

Goal: Run the entire ecosystem locally on your machine for development or testing. **File:** `docker-compose-local.yml`

This setup builds all images from source (your local code) and exposes standard default ports.

```
# Start all services
docker-compose -f docker-compose-local.yml up -d --build

# View logs
docker-compose -f docker-compose-local.yml logs -f
```

- **Configuration:** Uses `local` profiles (`application-local.properties`, `environment.local.ts`).
- **Infrastructure:** Connects to local Zipkin/Loki.
- **Database:** Connects to the Dev RDS instance (configured in `application-local.properties`).

2. Build & Release (Push to Server)

Goal: Build Docker images locally that correspond to specific server environments (Dev or Prod) and push them to ECR. **File:** `docker-compose-build.yml`

This file is used **ONLY** for building and pushing. It ensures that the images built match the environment variables defined in `env.dev` or `env.prod`.

For Development Environment:

```
# 1. Build Dev Images (Tags: dev-x.x.x)
docker-compose -f docker-compose-build.yml --env-file env.dev build

# 2. Push to ECR
docker-compose -f docker-compose-build.yml --env-file env.dev push
```

For Production Environment:

```
# 1. Build Prod Images (Tags: prod-x.x.x)
docker-compose -f docker-compose-build.yml --env-file env.prod build

# 2. Push to ECR
docker-compose -f docker-compose-build.yml --env-file env.prod push
```

3. Server Deployment

Goal: Run the application on the target server (Dev or Prod) using pre-built images from ECR. **File:** `docker-compose.yml`

This file is environment-agnostic. It relies completely on the keys provided by the `.env` file passed at runtime.

On Dev Server:

```
docker-compose --env-file env.dev -p ewallet-dev up -d
```

On Prod Server:

```
docker-compose --env-file env.prod -p ewallet-prod up -d
```

Project Configuration In-Depth

Spring Boot Profile Strategy

Each Spring Boot service follows a strict profile hierarchy: 1. **`dev` / `prod`**: Activated on servers via `SPRING_PROFILES_ACTIVE` in `docker-compose.yml`. Configured to use Docker service names (e.g., `http://loki:3100`) for infrastructure. 2. **`local`**: Activated for local development. Configured to use `localhost` (e.g., `http://localhost:3100`) so you can run services in your IDE.

Key Files per project: * `src/main/resources/application.properties`: Shared base config. * `src/main/resources/application-{profile}.properties`: Profile-specific overrides. * `src/main/resources/logback-spring.xml`: **Standardized Logging**. * **Features**: * JSON formatting for Loki. * 7-day rolling file retention (`logs/app.yyyy-MM-dd.log`). * Consolidated console output. * **Config**: Sends logs to `${LOGGING_LOKI_URL}`, which is dynamically set based on profile.

Angular Environment Strategy

Angular builds are parameterized to handle environments correctly without manual code commenting.

1. **Environments**:
   - `environment.ts`: Defaults to Dev.
   - `environment.prod.ts`: Production configuration (`api.bizionictech.com`).
   - `environment.local.ts`: Local configuration (`localhost:8080`).
2. **Build Configurations (`angular.json`)**:
   - `production`: Swaps `environment.ts` with `environment.prod.ts`.
   - `local`: Swaps `environment.ts` with `environment.local.ts`.
   - **Docker**: The `Dockerfile` accepts an `APP_ENV` argument.
     - `APP_ENV=production` -> runs `ng build -c production`
     - `APP_ENV=local` -> runs `ng build -c local`

Edge Cases & Troubleshooting

1. Config Server Dependency

- **Scenario**: Services (Gateway, Server) fail to start because they cannot fetch configuration.
- **Mitigation**:
  - Docker Compose allows `depends_on`, but Spring Boot acts faster than the Config Server startup.
  - **Solution**: We added `entrypoint` scripts in `docker-compose.yml` (for infra) and standard Spring `retry` mechanisms are enabled.
  - **Manual Fix**: If running locally, ensure `cloudConfigServer` is fully up (Health: typically 20-30s) before starting other services.

2. Loki & Logging

- **Scenario**: Loki is down or unreachable.
- **Behavior**: The `Loki4jAppender` is configured with timeouts (`connectionTimeoutMs=30000`). If it fails, the application **will not crash**; it will fallback to Console/File logging.
- **Logs**: Check the local `logs/` directory in the container or project folder if logs are missing from Grafana.

3. **Port Conflicts (Local)**

- **Scenario**: Port `8080` or `3306` is already in use.
- **Fix**: Check `docker-compose-local.yml` and modify the **host** port mapping (left side).
  - Example: Change `"8080:8080"` to `"8089:8080"`.
  - *Note: Do not change the container port (right side).*

4. **Database Access**

- **Note**: The `local` profile (`application-local.properties`) is currently hardcoded to point to the **AWS RDS Dev Database**.
- **Requirement**: You must have a stable internet connection and your IP must be whitelisted in the RDS Security Group on AWS. If the app hangs on startup, it is likely a DB connection timeout.

5. **Docker Networking**

- **Dev/Prod**: Uses a custom bridge network (`ewallet` for dev, `ewallet-prod` for prod). Service discovery relies on Eureka, but container-to-container talk relies on Docker DNS (service names).
- **Local**: Uses `driver: bridge` with network name `ewallet-local`.

Quick Reference: Environment Variables

Defined in `env.dev` and `env.prod`.

Variable	Description	Example (Dev)
`PROFILE`	Active Spring/App Profile	`dev`
`NETWORK_NAME`	Docker Network Name	`ewallet`
`TAG_SERVER`	Tag for Ewallet Server Image	`dev-1.2.27`
`PORT_SERVER`	Host Port for Server	`9082`
`NODE_ENV_ANGULAR`	Angular Build Mode	`development`

Maintained by: Antigravity