Лабораторная работа №5. Разработка микросервисов

1. Введение

В рамках работы выполнена разработка микросервисного приложения на примере проекта **BusinessThing**. Система разделена на набор независимых сервисов, каждый из которых отвечает за отдельную часть бизнес-функциональности и разворачивается как отдельный контейнер.

Цели:

- выделить сервисы по ответственности (bounded contexts) и обеспечить их автономность;
- обеспечить единый способ локального запуска и окружения для всех сервисов;
- зафиксировать состав сервисов и инфраструктуры, необходимых для работы системы.

2. Состав микросервисов

Проект разделён на несколько сервисов (каждый — отдельный модуль с собственным `Dockerfile` и `compose.yaml`):

- **core-service** — основной бэкенд (CRUD доменных сущностей, управление организациями/пользователями, документы, статусы обработки).
- **docs-processor** — воркер обработки документов (извлечение текста, чанкинг, подготовка данных для индексирования).
- **llm-service** — сервис, отвечающий за взаимодействие с LLM и оркестрацию сценариев (чат/агенты/RAG).
- **telegram-bot** — бот для взаимодействия в Telegram.
- **webapp** — Telegram WebApp (клиент).

Инфраструктурные компоненты, поднимаемые вместе с сервисами:

- **RabbitMQ** — брокер сообщений (используется для асинхронных задач).
- **PostgreSQL** — базы данных сервисов.
- **OpenSearch** — векторное/текстовое хранилище.
- **Jaeger** — трассировка запросов.
- **Traefik** — reverse proxy / edge-компонент.

3. Единый способ развёртывания (Docker Compose)

Для запуска системы используется корневой `compose.yaml`, который подключает compose-файлы отдельных сервисов и инфраструктуры через директиву `include`.

Пример структуры подключения сервисов и инфраструктуры (фрагмент `BusinessThing/compose.yaml`):

```
include:
    - llm-service/compose.yaml
```

```
      - llm-service-db/compose.yaml
      - core-service/compose.yaml
      - core-service-db/compose.yaml
      - docs-processor/compose.yaml
      - webapp/compose.yaml
      - telegram-bot/compose.yaml
      - jaeger/compose.yaml
      - rabbitmq/compose.yaml
      - opensearch/compose.yaml
      - traefik/compose.yaml
```

Таким образом достигается:

- единая команда запуска окружения;
- согласованное сетевое окружение и зависимости;
- возможность включать/выключать части инфраструктуры через профили.

4. Структура сервисов и код-стайл

Для сервисов используется единый подход к организации кода.

Пример модульной структуры (на примере `core-service/internal/`):

- `domain/` — доменные модели и правила;
- `service/` — прикладные сценарии (use cases);
- `repository/`, `db/`, `storage/` — инфраструктурные адаптеры;
- `config/`, `logger/`, `tracer/` — общие компоненты окружения.

Такое разделение упрощает расширение системы: добавление нового сервиса или доменной подсистемы минимально затрагивает соседние компоненты.

5. Вывод

В результате сформирован микросервисный контур приложения: сервисы выделены по ответственности, каждый сервис контейнеризован и подключается к общему окружению через Docker Compose. Это создаёт основу для последующей настройки взаимодействия между сервисами и масштабирования системы.