**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина: Б**ек-энд разработка

Отчет

## Лабораторная работа №3+4: Микросервисы

Выполнила:

Пронина Александра

K33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Дисклеймер: Эта работа была сделана ещё до окончательной версии Лабораторной работы 2 из-за путаницы с дедлайнами. Лабораторная работа 2 изначально должна была быть по-другому сделана(в плане моделей), но, опять же, из-за путаницы с дедлайнами, итоговый вариант быть упрощен. P.S. прошу понять и простить за расхождение версий

Первым шагов я определила, какие папки должны быть в каталоге, то есть разделить функционал на: то, что относится к книгам, что относится к клиенту и что относится и покупкам.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Далее внутри каждой папки создадим свою src, в которой также будет разделим функционал. Далее будет всё описано на примере папки books( для папок customer и shopping аналогично):

**Api:** код представляет собой типичную структуру приложения Node.js, использующего Express.js для обработки маршрутов и выполнения логики.

**Config:**   
Этот модуль загружает переменные среды из файла .env, в зависимости от окружения (dev, test, prod). Затем он экспортирует набор ключей, используемых в приложении, такие как порт, URL базы данных, секрет приложения и другие, обеспечивая централизованное управление конфигурацией приложения.

**Database:**   
Эта папка отвечает за работу с базой данных MongoDB через Mongoose в Node.js. В ней определены модели данных, репозиторий для взаимодействия с продуктами, а также модуль для подключения к базе данных.

**Services:**   
Этот файл содержит класс **ProductService**, который служит слоем бизнес-логики для взаимодействия с продуктами. Он использует методы из **ProductRepository**, импортированного из директории **../database**, и включает функции, такие как создание продуктов, получение продуктов, получение описаний продуктов, получение продуктов по категориям, получение выбранных продуктов и генерацию данных для событий, связанных с продуктами. Он также использует утилитарную функцию **FormatData** из директории **../utils** для форматирования данных перед возвратом.

**Ulils:** Этот блок кода определяет несколько классов ошибок и обработчик ошибок для приложения Node.js.

Далее в папке идет пара айлов,представляющих собой точку входа для запуска приложения.

В файле index.js создается экземпляр приложения Express. Затем устанавливаются middleware для обработки JSON-запросов и статических файлов. После этого создается канал для работы с очередью сообщений с помощью функции CreateChannel из утилитарного модуля. Наконец, модуль products инициализируется, передавая ему созданный канал и экземпляр приложения Express.

В файле StartServer.js функция StartServer вызывает последовательность действий для запуска сервера. В первую очередь происходит подключение к базе данных. Затем инициализируется Express-приложение, используя модуль expressApp. После этого сервер запускается на указанном порту, который определен в файле конфигурации. Если происходит ошибка при запуске сервера, она выводится в консоль, и процесс завершается.

**Gateway** – входная точка общей программы.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Для лабораторной 4: Docker Compose определены следующие сервисы:

nosql-db: Сервис MongoDB, который используется как база данных для приложения. Он использует образ mvertes/alpine-mongo.

rabbitmq: Сервис RabbitMQ, который используется для обмена сообщениями между различными сервисами приложения. Он использует образ rabbitmq:alpine.

books: Сервис для приложения "Books". Он собирается из Dockerfile в директории books и использует файл .env.dev для переменных среды.

shopping: Сервис для вашего приложения "Shopping". Аналогично сервису "Books", он собирается из Dockerfile в директории shopping и использует файл .env.dev для переменных среды.

customer: Сервис для приложения "Customer". Аналогично предыдущим двум сервисам, он собирается из Dockerfile в директории customer и использует файл .env.dev для переменных среды.

nginx-proxy: Сервис Nginx, который используется в качестве обратного прокси для приложения. Он проксирует запросы на сервисы "Books", "Shopping" и "Customer".

Вывод:

Этот проект представляет собой модульную архитектуру микросервисов для онлайн-платформы, включающей сервисы для управления книгами, покупками и клиентами. Каждый сервис разделен на отдельные функциональные блоки, такие как API, конфигурация, база данных, службы и утилиты, что позволяет поддерживать чистоту кода и обеспечивать гибкость разработки.

Каждый сервис имеет свою собственную структуру, включая уникальные модели данных, логику бизнес-процессов и обработку ошибок. Использование Docker Compose обеспечивает легкость развертывания и масштабирования всего приложения, включая базу данных MongoDB и систему обмена сообщениями RabbitMQ.

В целом, я старалась хорошо структурировать и организовать проект, чтобы облегчалась его поддержка и дальнейшее развитие. Либо какие-то правки в случае надобности.