Отчет по большому домашнему заданию №2 по курсу «Конструирование программного обеспечения»

Каспари Давид Эрвинович.

В этом отчете я отражу результаты своей работы, опишу архитектуру написанной программы, приведу инструкции.

1. **Требования к функционалу.**

   3-й пункт необязателен, а второй в процессе разбирательств тоже стал необязательным (С. А. Виденин подтвердил в беседе). Поэтому мной был реализован только первый пункт по подсчёту статистики.

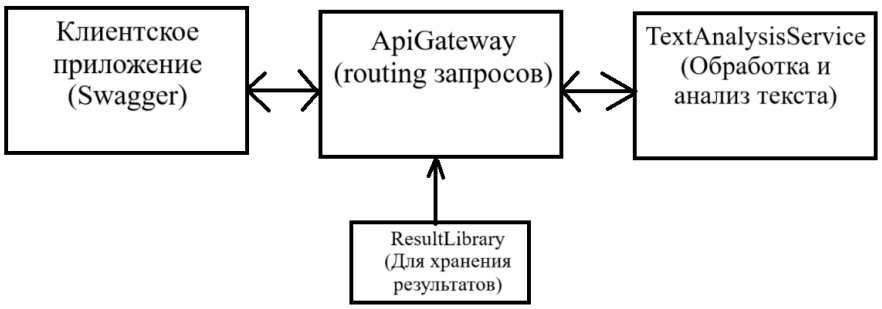
2. **Микросервисная архитектура.**

   Было реализовано 2 микросервиса из предложенных, так как FileStorage не нужен без проверки файлов на схожесть. Реализованные сервисы:
   1) API Gateway – отвечает только за routing запросов
   2) Text Analisys Service – отвечает только за проведение анализа, хранение результатов и их выдачу

3. **Архитектура программы.**

   В Visual Studio созданы Веб-API Asp.Net.Core с названиями ApiGateway и TextAnalysisService. Также создана библиотека классов ResultLibrary.

   На следующем рисунке показана общая схема работы:



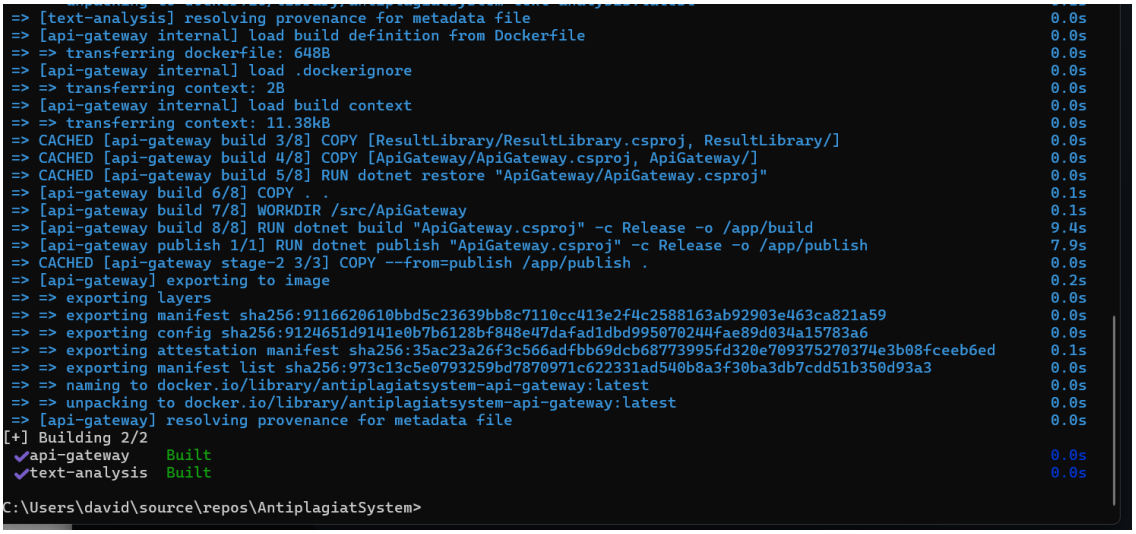
Библиотека классов ResultLibrary содержит класс AnalysisResult, который отвечает за представление результатов анализа данных.
ApiGateway содержит Program.cs, который занимается маршрутизацией.
TextAnalysisService содержит Program.cs, занимающийся обработкой запроса, в Services он содержит ITextAnalysisService.cs – интерфейс для класса, который занимается анализом, и TextAnalysisService.cs, который содержит методы для подсчета абзацев, слов, символов.

Более подробно с кодом можно ознакомиться и с помощью комментариев в программах.

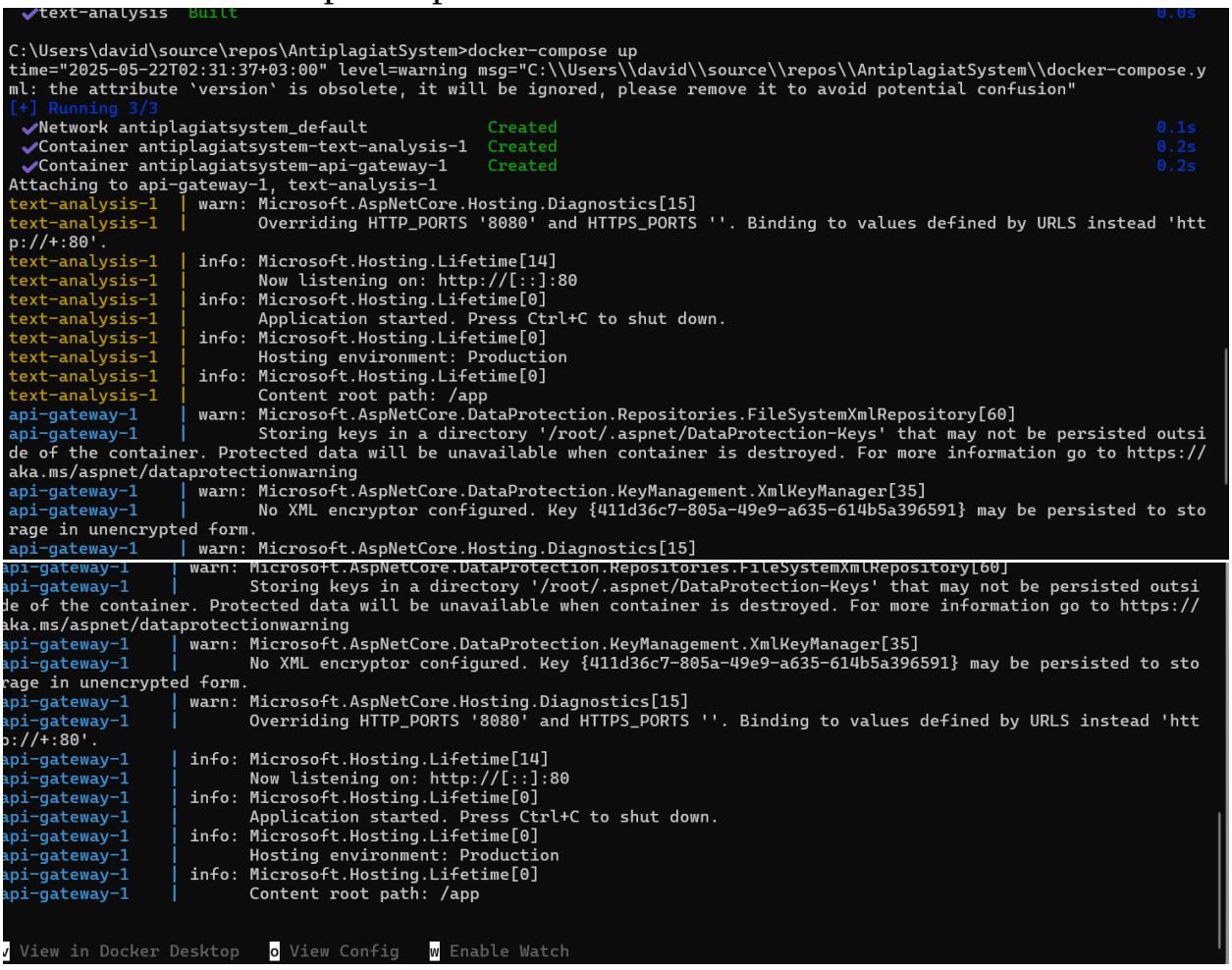
4. Сборка и запуск программы. Docker-файлы.

Программа собиралась через Docker. Было написано два Dockerfile для ApiGateway и TextAnalysisService, а также docker-compose.yml, собирающий программу.
Они были написаны с помощью инструкции, прикрепленной к материалам к этому домашнему заданию.

Приведу скриншоты процесса сборки.
После docker-compose build:



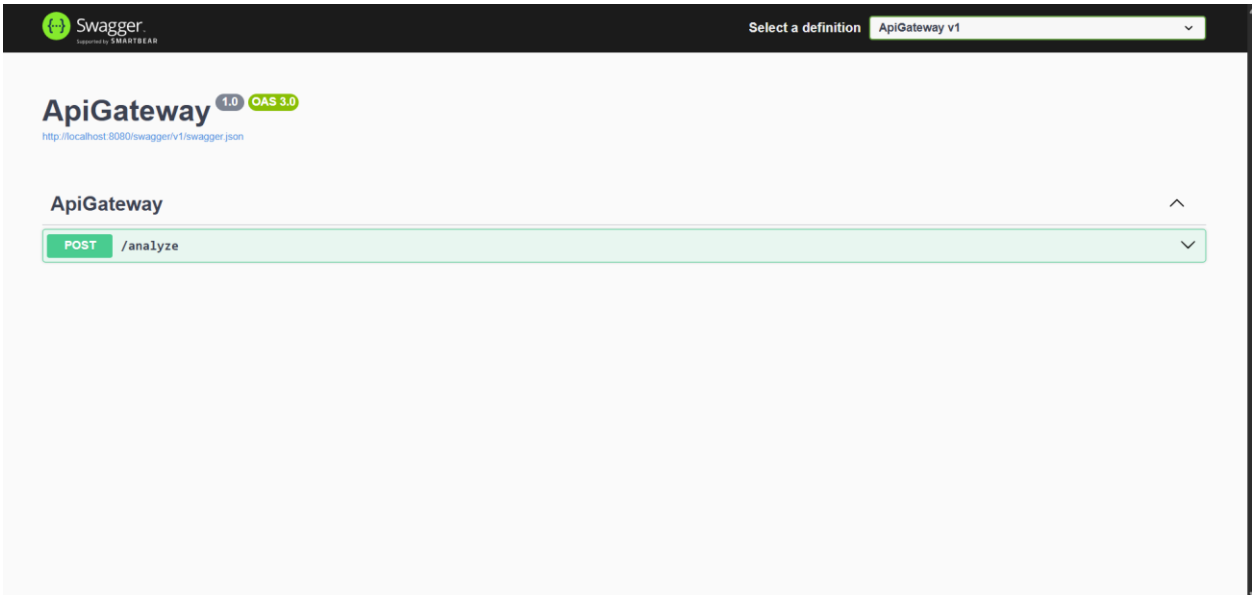
Программа собралась.
После docker-compose up:



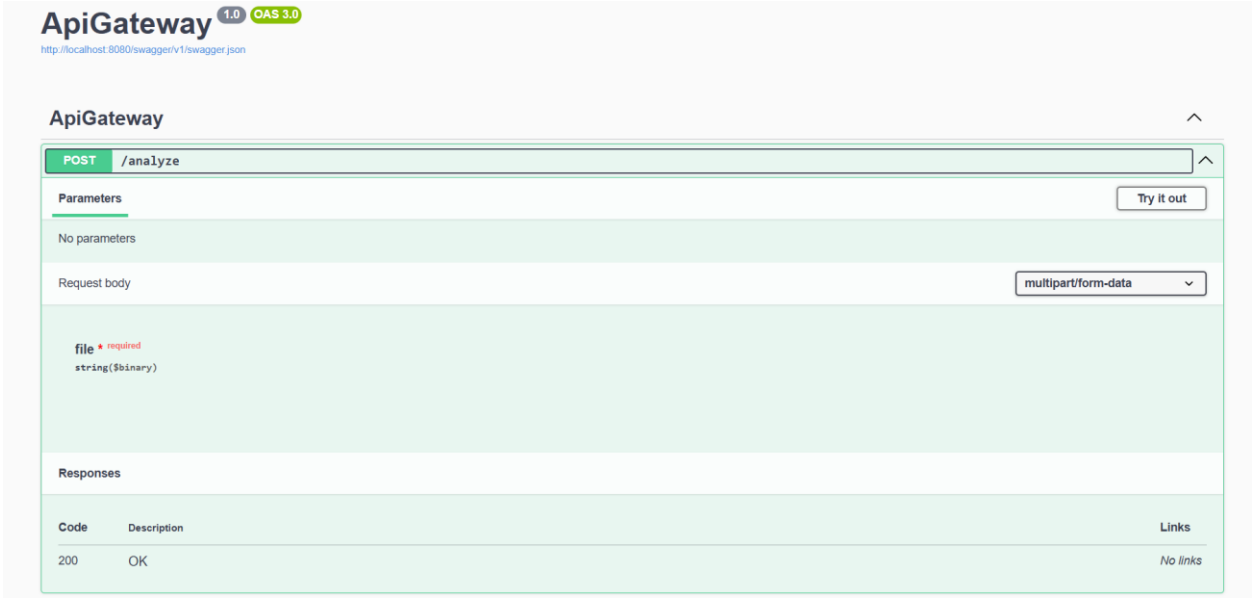
Приложение запустилось. Можно протестировать по ссылке:
http://localhost:8080/swagger

Теперь загрузим файлы разного типа и проверим работоспособность программы.

После открытия ссылки видим следующее:



Нажимаем try it out.



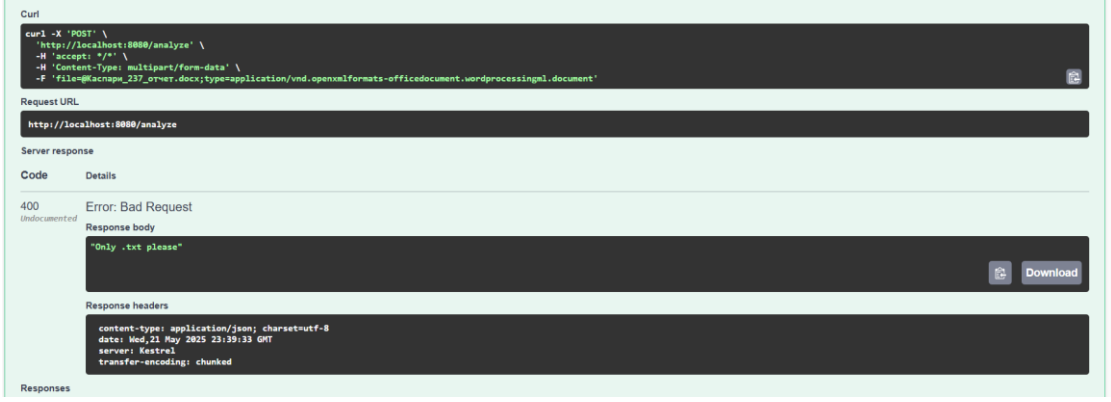
Загрузим такой txt-файл:

```
Hello, World!

KPO


Software Design
BHW-2

FCS

Kaspari David
```

Нажимаем Execute. Получаем такой ответ.

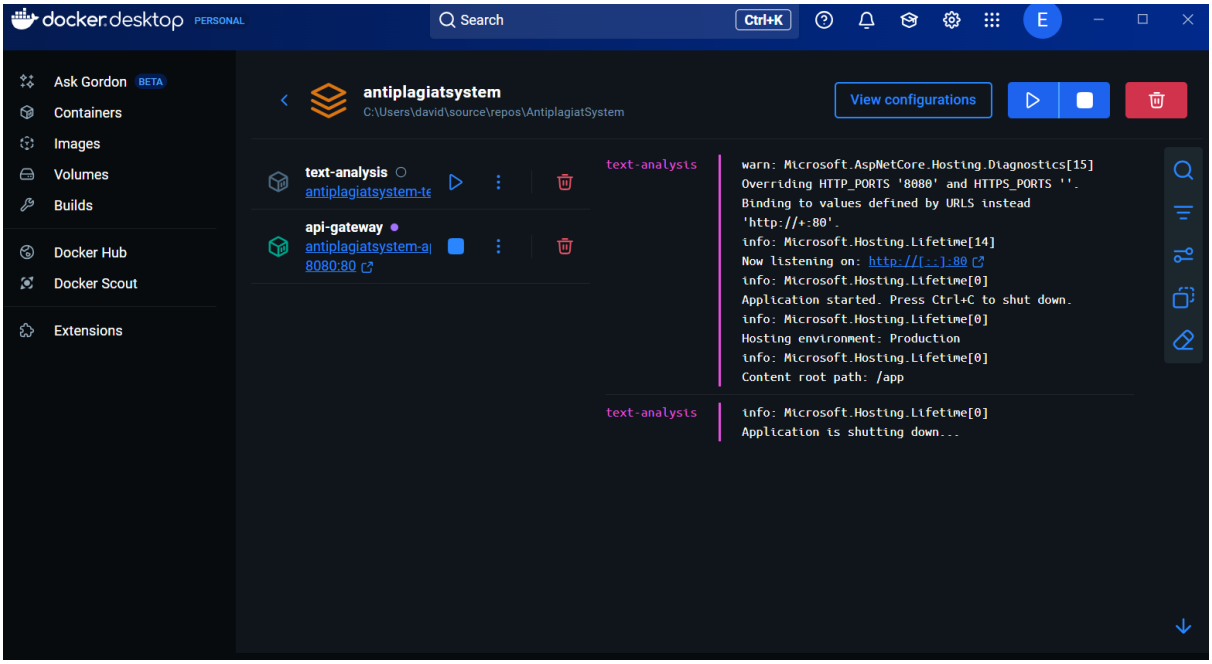
Как видим, получили верный ответ с кол-вом символов, абзацев, слов, а также датой обращения.

Загрузим файл другого формата (наш word-отчет, например).



Получили ответ, что принимаются только txt-файлы.

Если отключить один из серверов (TextAnalysis, например), то получим ошибку:



В целом, это вся работа. Пройдемся по критериям оценки, чтоб проверить соответствие данной работы заданию:

Основные требования к функциональности реализованы (так получилось, что это только первый пункт).
Серверная часть приложения состоит из двух микросервисов (ApiGateway и TextAnalysisService).
Если один из микросервисов не работает, то показывается ошибка.
Swagger демонстрирует функциональность реализованных микросервисов.

Код разделен на модули и имеет структуру, описанную выше. Документация (то есть этот отчет) содержит краткое описание архитектуры системы и спецификацию API.

Как видно, основные требования к программе выполнены.

В процессе работы я познакомился с тем, как работает синхронное межсервисное взаимодействие, поработал со Swagger’ом, а также с Docker.