|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКИЕ РАБОТЫ №8-10**

по дисциплине «Разработка серверных частей интернет-ресурсов»

**Студент группы** ИКБО-20-21 Сидоров С.Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** ассистент Благирев М.М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена « » 2023 г.

Допущен к работе « » 2023 г.

Москва 2023

**Содержание**

[Практическая работа №8 3](#_Toc153387124)

[Теоретическое введение. 3](#_Toc153387125)

[Задание 4](#_Toc153387126)

[Выполнение задания 5](#_Toc153387127)

[Вывод 9](#_Toc153387128)

[Практическая работа №9 10](#_Toc153387129)

[Теоретическое введение. 10](#_Toc153387130)

[Задание 12](#_Toc153387131)

[Выполнение задания 13](#_Toc153387132)

[Вывод 18](#_Toc153387133)

[Практическая работа №10 19](#_Toc153387134)

[Теоретическое введение. 19](#_Toc153387135)

[Задание 20](#_Toc153387136)

[Выполнение задания 21](#_Toc153387137)

[Вывод 26](#_Toc153387138)

# Практическая работа №8

## Теоретическое введение.

Проекты на Java и, особенно, PHP сложно расширять и оптимизировать. Примерно такие мысли пришли в голову разработчикам компании Google во второй половине нулевых годов (они тогда разрабатывали в основном на С++ и C). Поэтому в стенах компании Google был разработан язык Go. Как говорили сами разработчики – «Go был разработан для решения реальных проблем, возникающих при разработке программного обеспечения в Google». Итак, Golang должен был решить следующие проблемы:

* медленная сборка программ;
* неконтролируемые зависимости;
* использование программистами различных подмножеств языка;
* трудности с пониманием программ — из-за сложного синтаксиса, плохого документирования;
* дублирование разработок;
* высокая стоимость обновлений;
* сложности разработки инструментария;
* плохое межъязыковое взаимодействие.

Для широкой общественности язык стал доступен в 2009 году, а в 2021 году, по данным GitHub, Go вошел в топ 5 востребованных языков, опередив PHP, Typescript и C#

## Задание

Вам необходимо реализовать простой сервис на Go, который будет принимать JSON запросы и хранить данные, введенные пользователем с помощью cookie. Достаточно одной REST ручки с основной логикой и ручки получения данных из cookie.

Cookie должны быть зашифрованы так, чтобы пользователь не мог их изменить без потери информации, а также прочитать содержимое. В Cookie проще всего будет хранить запрос пользователя и ответ сервиса.

Важно также показать, как Go работает с конкурентными вычислениями. Это можно сделать даже используя time.Sleep(). Для наглядности нужно сделать два вариации ручки сервиса - линейную и конкурентную.

Сервис необходимо прикрыть Nginx (будет запущен в отдельном docker контейнере), который будет проксировать на него все запросы с префиксом /api.

Обязательно соблюдать [Go Project Layout](https://github.com/golang-standards/project-layout).

Конфигурация сервиса должна задавать с помощью ENV файла (порт, имя cookie и т.д.).

Не стоит забывать и про логирование ошибок, они должны писаться в поток стандартного вывода и в файл, который можно будет прочитать извне контейнера.

Приветствуется использование различных библиотек.

## Выполнение задания

Для создания образа, был использован Dockerfile, изображенный на рисунке 1.

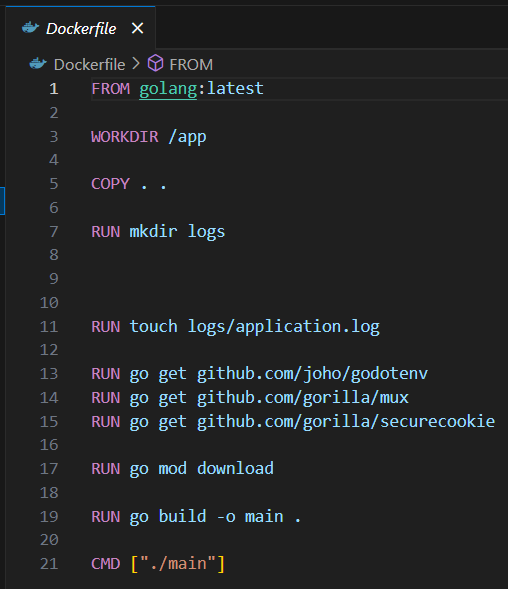


Рисунок 1 – Dockerfile

Для связи приложения и базы данных мы будем использовать docker-compose. Его содержимое показано на рисунке 2.

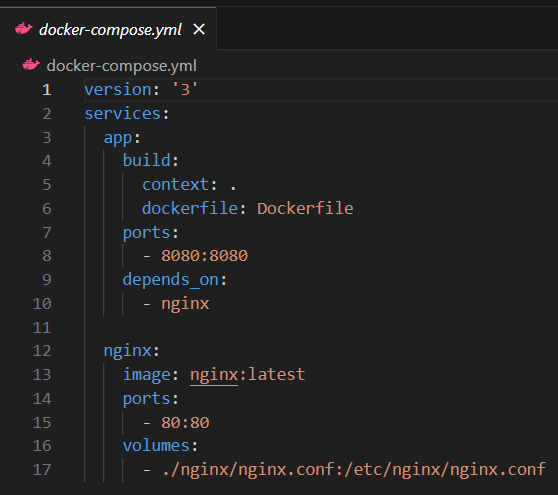


Рисунок 2 – docker-compose.yml

Для выполнения задания был создан файл main.go, его содержимое представлено на рисунках 3 – 6.

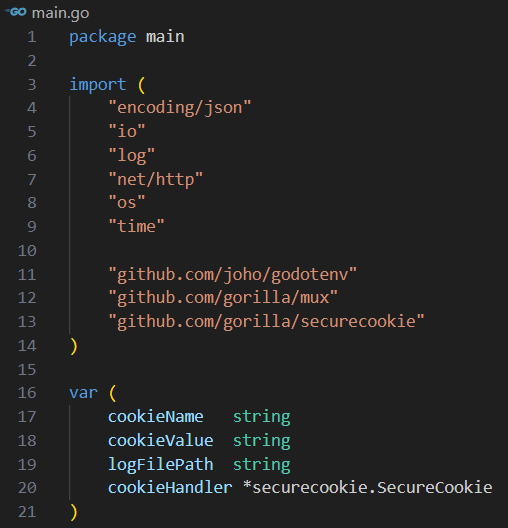


Рисунок 3 – Фрагмент кода

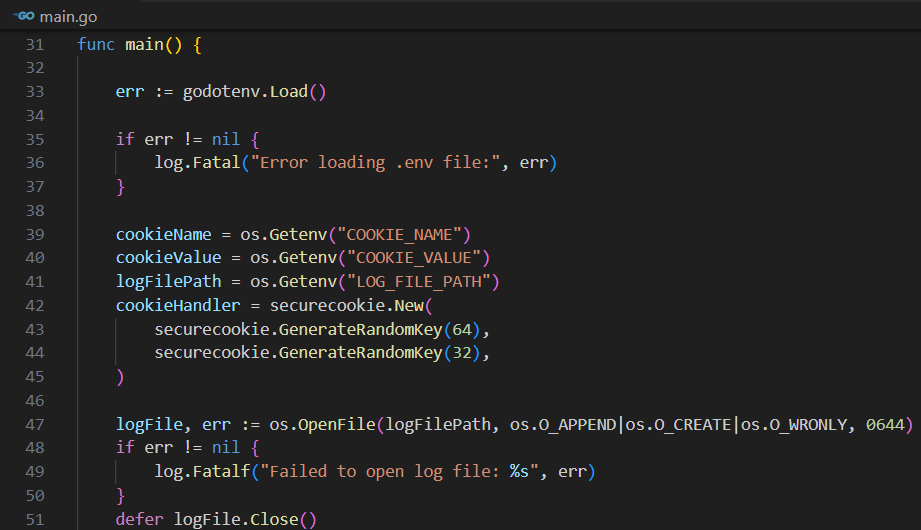


Рисунок 4 – Фрагмент кода функции main



Рисунок 5 – Фрагмент кода функции linearHandler



Рисунок 6 – Фрагмент кода функции concurrentHandler

Результат работы программы представлен на рисунке 7.

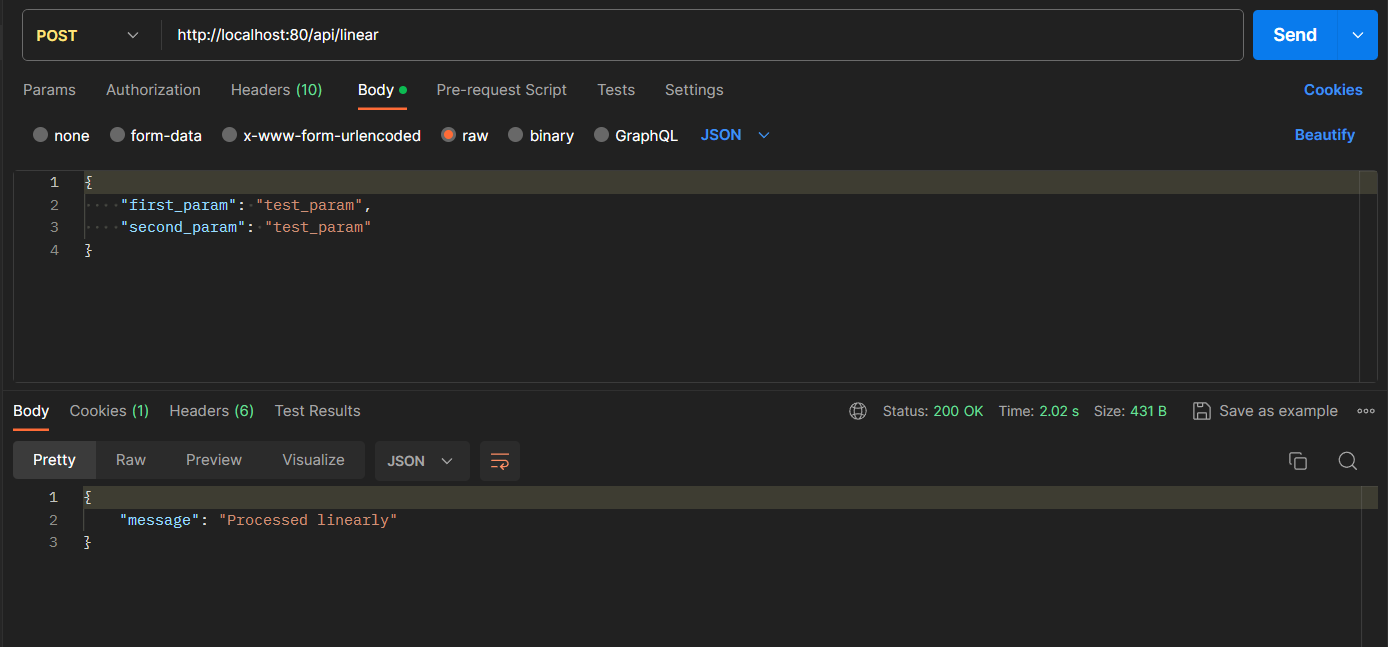


Рисунок 7 – Результат работы программы

## Вывод

В ходе выполнения практической работы был реализован сервис на Go, который может хранить данные, введенные пользователем в cookie. Сервис был прикрыт Ngix.

# Практическая работа №9

## Теоретическое введение.

MongoDB — система управления базами данных, которая работает с документоориентированной моделью данных. В отличие от реляционных СУБД, MongoDB не требуются таблицы, схемы или отдельный язык запросов. Информация хранится в виде документов либо коллекций.

Разработчики позиционируют продукт как промежуточное звено между классическими СУБД и NoSQL. MongoDB не использует схемы, как это делают реляционные базы данных, что повышает производительность всей системы.

У MongoDB есть ряд свойств, которые выделяют ее на фоне других продуктов:

У MongoDB есть ряд свойств, которые выделяют ее на фоне других продуктов:

1. Кроссплатформенность. СУБД разработана на языке программирования С++, поэтому с легкостью интегрируется под любую операционную систему (Windows, Linux, MacOS и др.).
2. Формат данных. MongoDB использует собственный формат хранения информации — Binary JavaScript Object Notation (BSON), который построен на основе языка JavaScript.
3. Документ. Если реляционные БД используют строки, то MongoDB — документы, которые хранят значения и ключи.
4. Вместо таблиц MongoDB использует коллекции. Они содержат разные типы наборов данных
5. Репликация. Система хранения информации в СУБД представлена узлами. Существует один главный и множество вторичных. Данные реплицируются между точками. Если один первичный узел выходит из строя, то вторичный становится главным.
6. Индексация. Технология применяется к любому полю в документе на усмотрение пользователя. Проиндексированная информация обрабатывается быстрее.
7. Для сохранения данных большого размера MongoDB использует собственную технологию GridFS, состоящую из двух коллекций. В первой (files) содержатся имена файлов и метаданные по ним. Вторая (chunks) сохраняет сегменты информации, размер которых не превышает 256 Кб.
8. СУБД осуществляет поиск по специальным запросам. Например, пользователь может создать диапазонный запрос и мгновенно получить ответ.
9. Балансировщик нагрузки используется в СУБД не только для распределения нагрузки между разными базами данных, но и для горизонтального масштабирования. Сегменты БД распределяются по разным узлам, что повышает производительность. При этом базы данных, расположенные на разных узлах, синхронизированы между собой и обеспечивают целостность информации для клиента.
10. MongoDB может поставляться для конечного клиента как облачное решение.

СУБД используют для хранения событий в системе (логирование), записи информации с датчиков мониторинга на предприятии, а также в сфере электронной коммерции и мобильных приложений. Часто MongoDB применяют как хранилище в сфере машинного обучения и искусственного интеллекта.

MongoDB относится к классу NoSQL СУБД и работает с документами, а не с записями. Это кроссплатформенный продукт, который легко внедряется в любую операционную систему. Ряд уникальных особенностей позволяет использовать СУБД под определённые задачи, в которых она обеспечивает максимальную производительность и надежность.

## Задание

Вам необходимо реализовать небольшой сервис хранения файлов, который будет хранить файлы в GridFS хранилище MongoDB.

У сервиса должно быть несколько основных REST ручек:

|  |
| --- |
| GET /files - получение списка файлов  GET /files/{id} - файла по id  GET /files/{id}/info - получение информации о файле по id  POST /files - загрузка файла  UPDATE /files/{id} - обновление файла по id  DELETE /files/{id} - удаление файла по id |

Информация о файле должна записываться в MongoDB, а сам файл должен попадать в GridFS хранилище.

Получается 3 контейнера:

1. Сервис хранилища
2. MongoDB к нему
3. И сверху Nginx, куда же без него

Обязательно соблюдать [Go Project Layout](https://github.com/golang-standards/project-layout).

Конфигурации микросервисов должны задаваться с помощью ENV файлов (порт, адрес БД и т.д.).

Не забывайте про правильное логирование.

## Выполнение задания

Для создания образа, был использован Dockerfile, изображенный на рисунке 8.

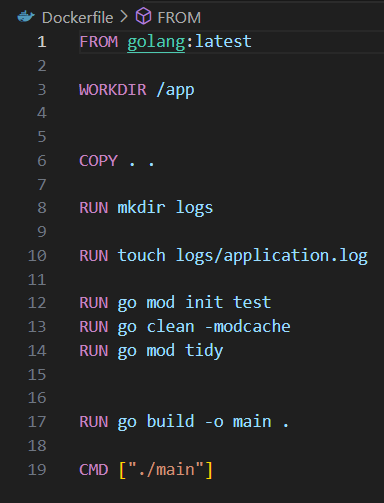


Рисунок 8 – Dockerfile

Для связи приложения и базы данных мы будем использовать docker-compose. Его содержимое показано на рисунке 9.

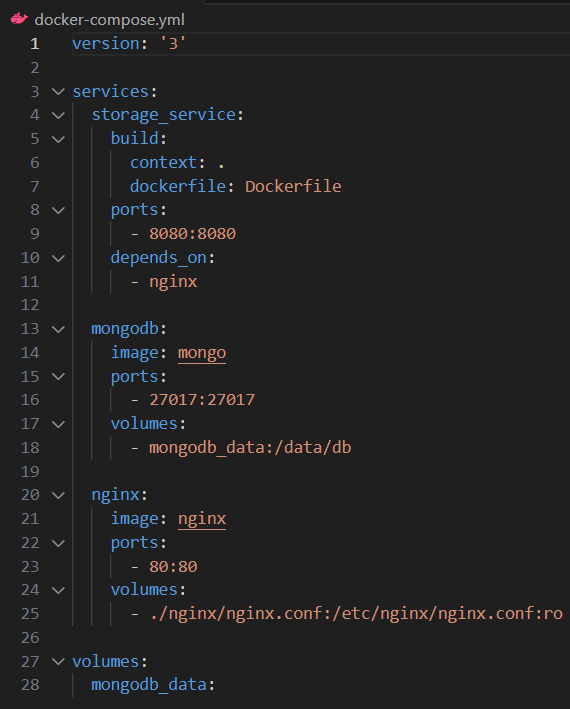


Рисунок 9 – docker-compose.yml

Для выполнения задания был создан файл main.go, его содержимое представлено на рисунках 10 – 13.



Рисунок 10 – Фрагмент кода

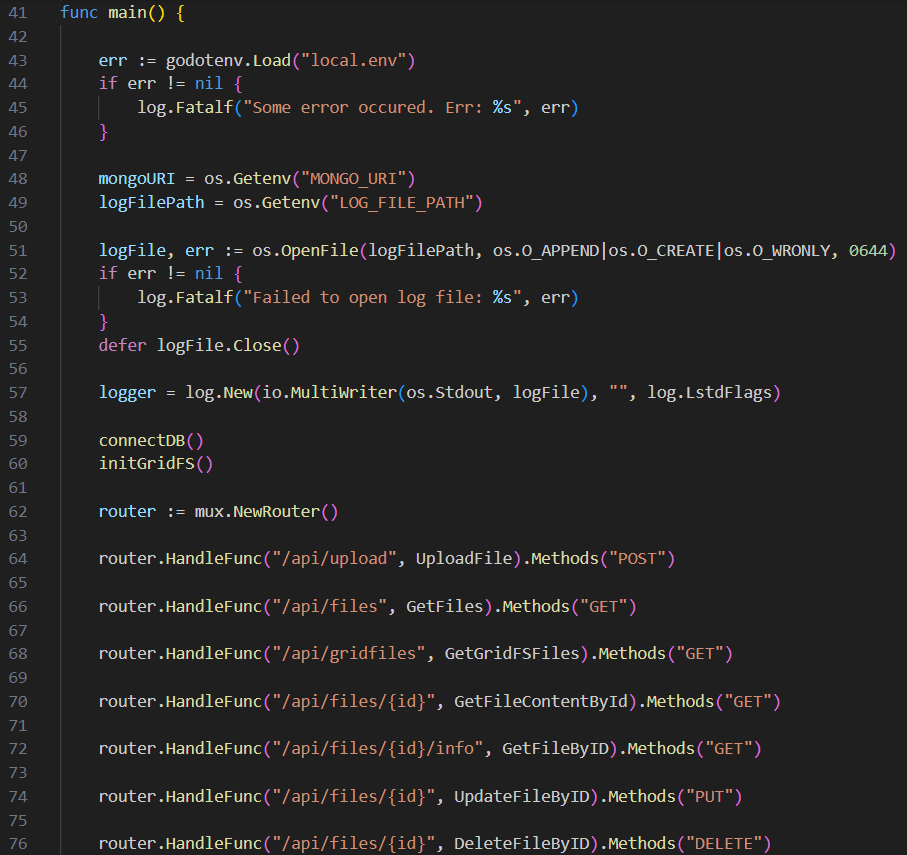


Рисунок 11 – Фрагмент кода функции main

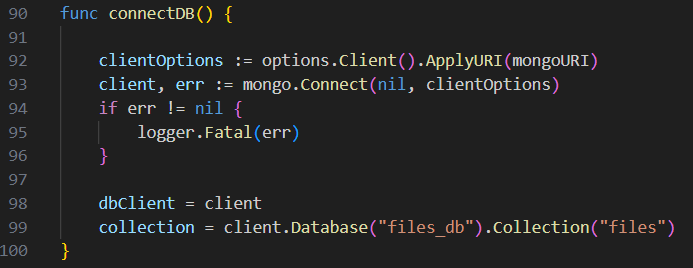


Рисунок 12 – Фрагмент кода функции connectDB

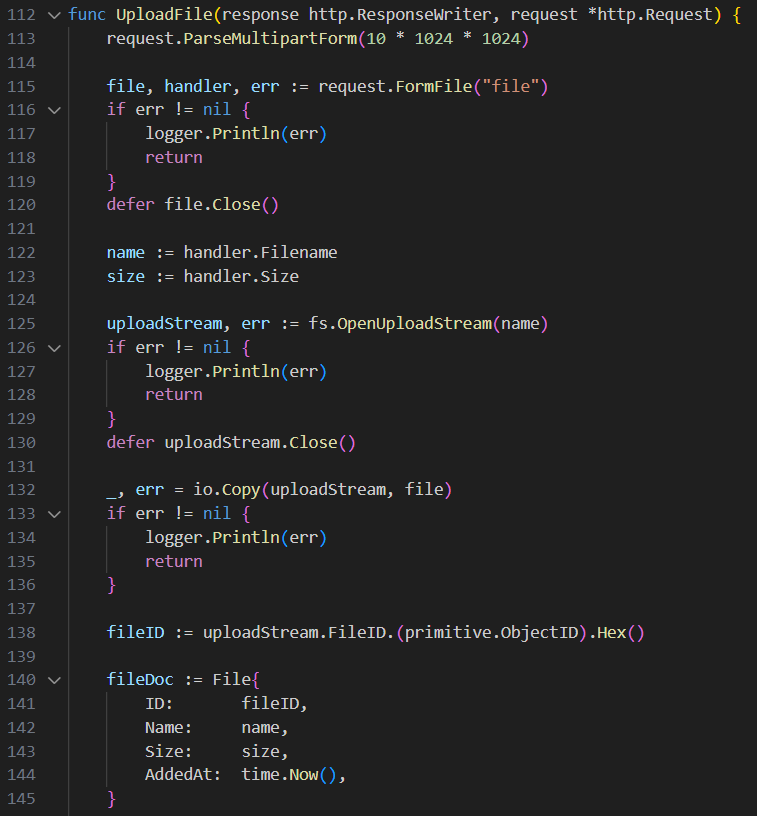


Рисунок 13 – Фрагмент кода функции UploadFile

Результат работы программы представлен на рисунках 14 - 16.

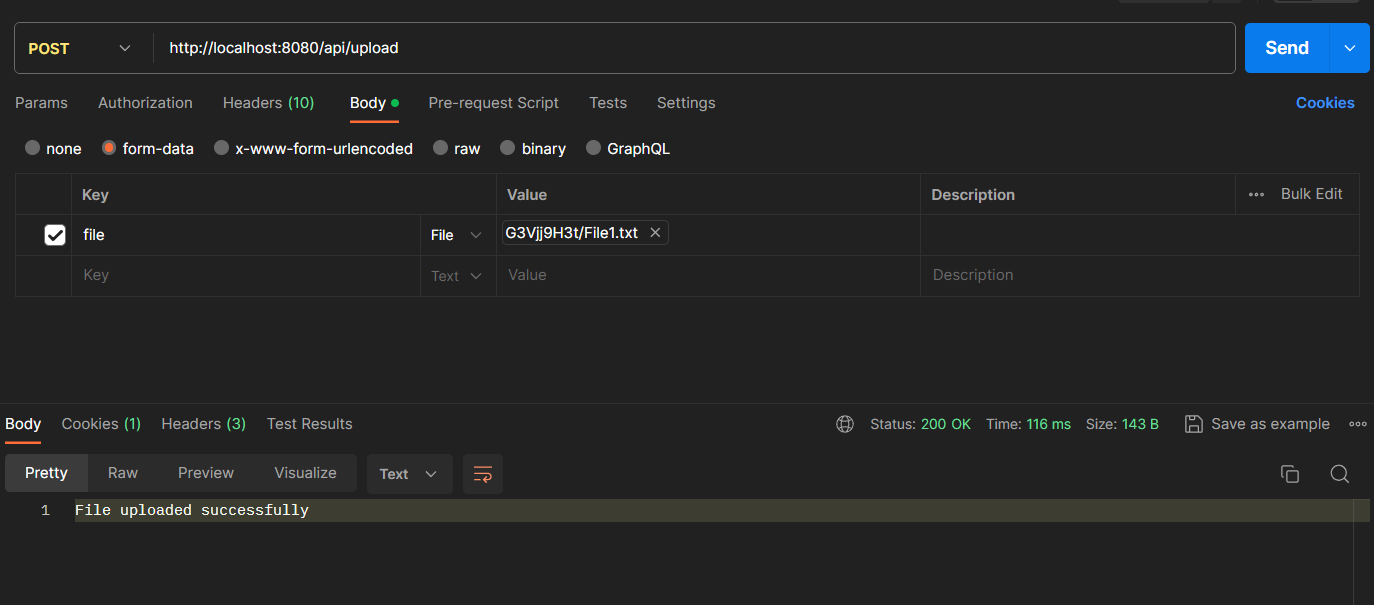


Рисунок 14 – Загрузка файла

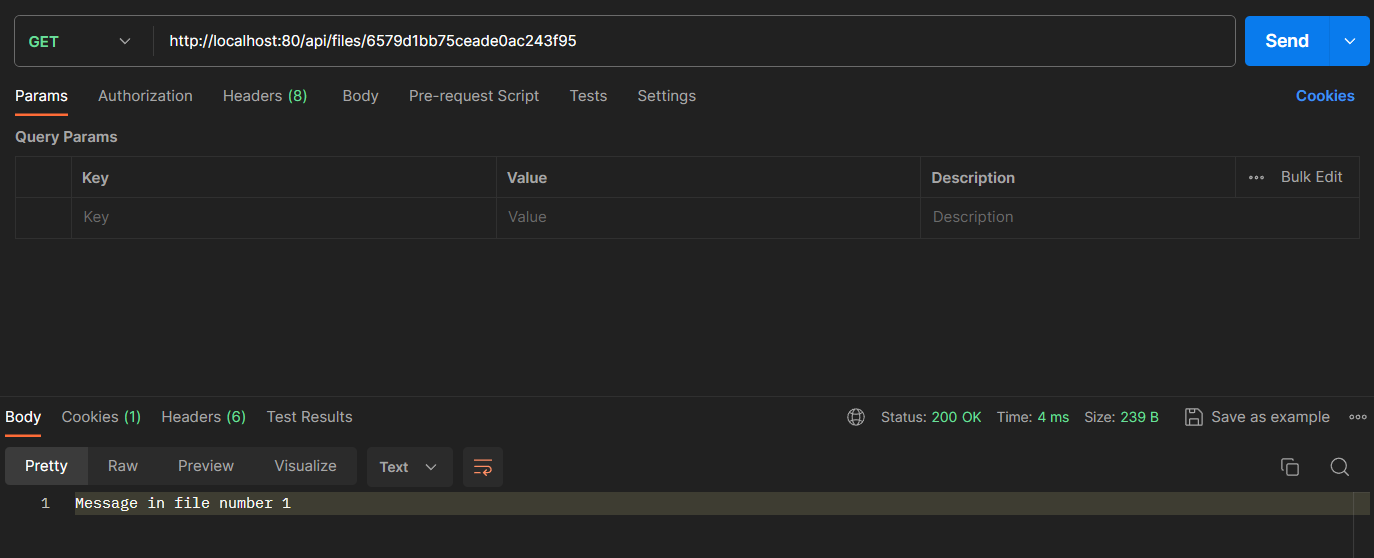


Рисунок 15 – Чтение файла

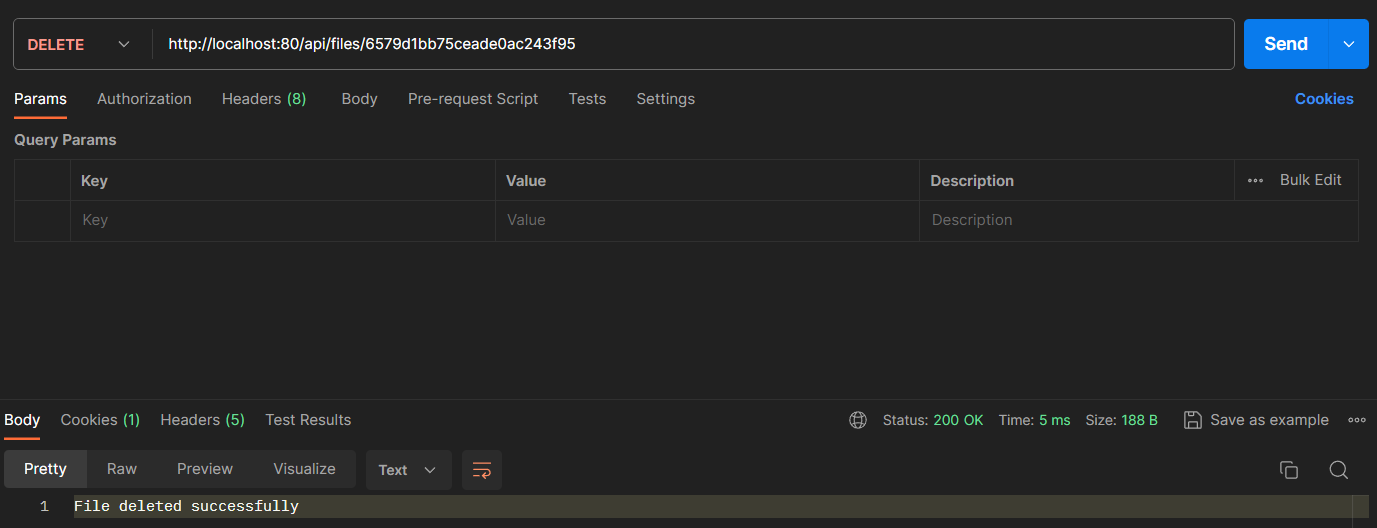


Рисунок 16 – Удаление файла

## Вывод

В ходе выполнения практической работы был реализован сервис хранения файлов в GridFS хранилища MongoDB.

# Практическая работа №10

## Теоретическое введение.

Как предшественник REST, RPC (удаленный вызов процедур) представляет собой программную архитектуру, восходящую к 1970-м годам. RPC позволяет вызывать функцию на удаленном сервере в определенном формате и получать ответ в том же формате. Не имеет значения, какой формат использует сервер, выполняющий запрос, и не имеет значения, локальный это сервер или удаленный. RPC позволяет вызывать функцию на сервере и получать результат в том же формате.

Основная концепция RPC API аналогична концепции REST API. RPC API определяет правила взаимодействия и методы, которые клиент может использовать для взаимодействия с ним. Клиенты отправляют вызовы, которые используют «аргументы» для вызова этих методов. Однако в случае RPC API метод находится в URL-адресе. Аргументы, вызывающие методы, находятся в строке запроса. Чтобы проиллюстрировать это, вот как запрос RPC API сравнивается с запросом REST API:

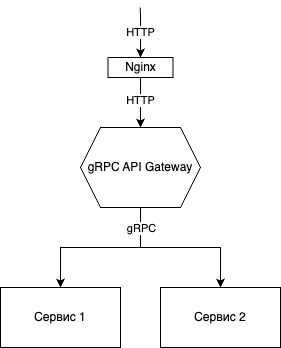
* RPC: запрос RPC API может использовать POST / deleteSmth и иметь строку запроса, которая говорит {"id": 777}.
* REST: запрос REST API записывает этот запрос как DELETE / smth / 777.

Однако, сегодня еще нельзя создать общедоступный сервис, который будет общаться с пользователями, используя HTTP/2 и gRPC. Чтобы решить эту проблему мы создадим API Gateway, который будет принимать запросы от пользователей в виде стандартных HTTP/1 запросов и дальше роутить их на нужные внутренние микросервисы с использованием gRPC и Protobuf.

## Задание

Студенту предлагается реализовать небольшую микросервисную архитектуру с использованием gRPC и API Gateway. Чтобы немного упростить задачу и не реализовывать сами сервисы, предлагается использовать сервисы из прошлых практик (как на Go, так и на Java), правда их необходимо будет перевести на работу с gRPC. Помимо API Gateway должно быть два и более микросервисов.

У вас получится следующая архитектура:



Обязательно соблюдать [Go Project Layout](https://github.com/golang-standards/project-layout).

Конфигурация сервисов должна задаваться с помощью ENV файла (порт, имя cookie и т.д.).

Не стоит забывать и про логирование ошибок, они должны писаться в поток стандартного вывода и в файл, который можно будет прочитать извне контейнера. Для каждого сервиса будет свой отдельный файл логов.

## Выполнение задания

Для создания образа, были использованы dockerfile, изображенные на рисунках 17 - 19.

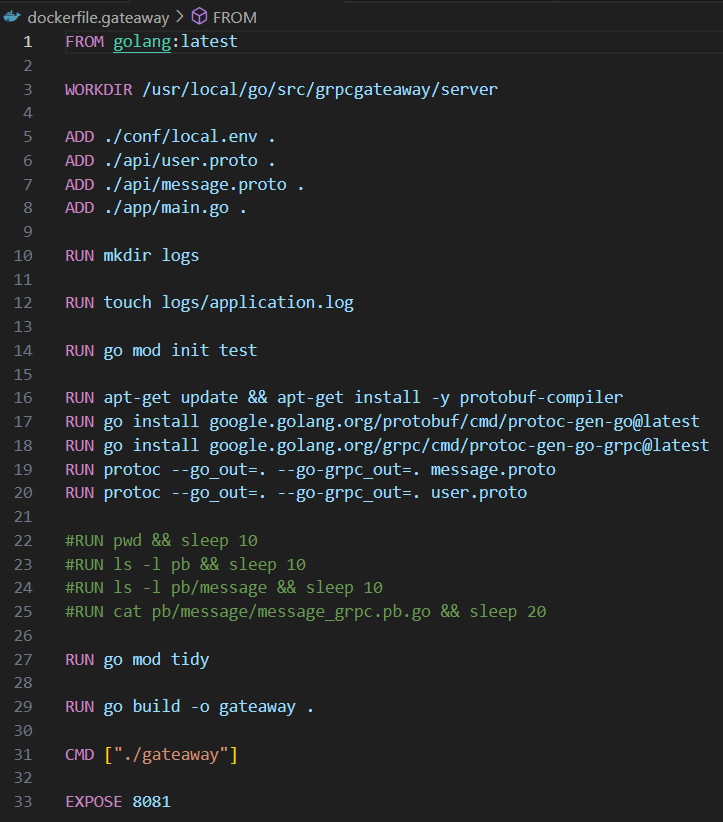


Рисунок 17 – dockerfile.gateaway

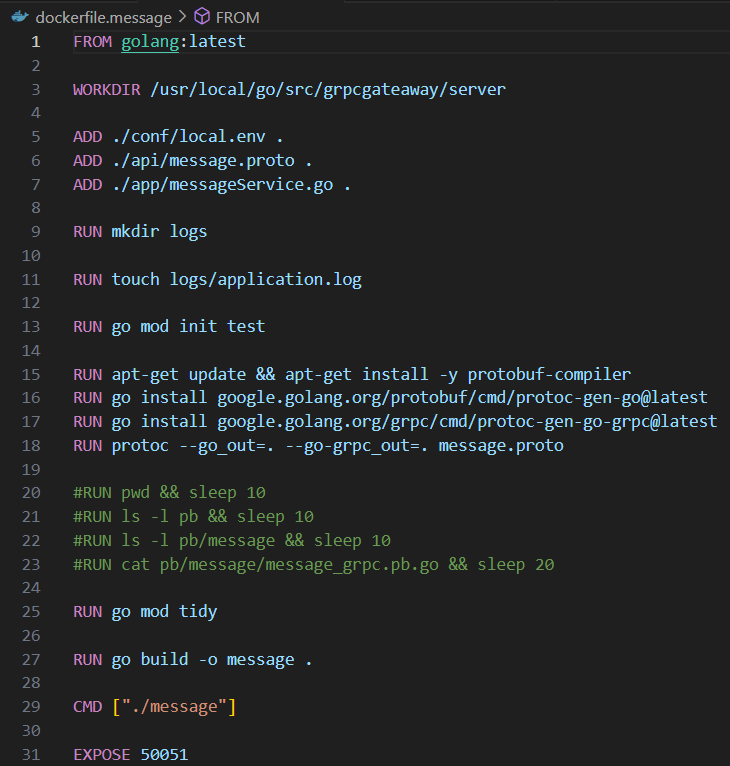


Рисунок 18 – dockerfile.message

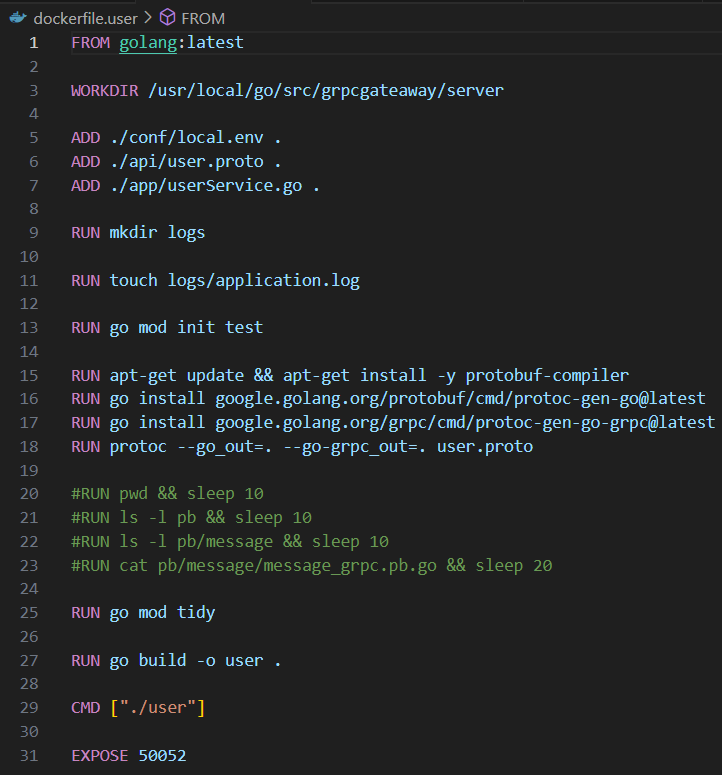


Рисунок 19 – dockerfile.user

Для связи приложения и базы данных мы будем использовать docker-compose. Его содержимое показано на рисунке 20.



Рисунок 20 – docker-compose.yml

Для решения поставленной задачи было создано два сервиса – userService и messageService, их содержимое представлено на рисунках 21, 22.

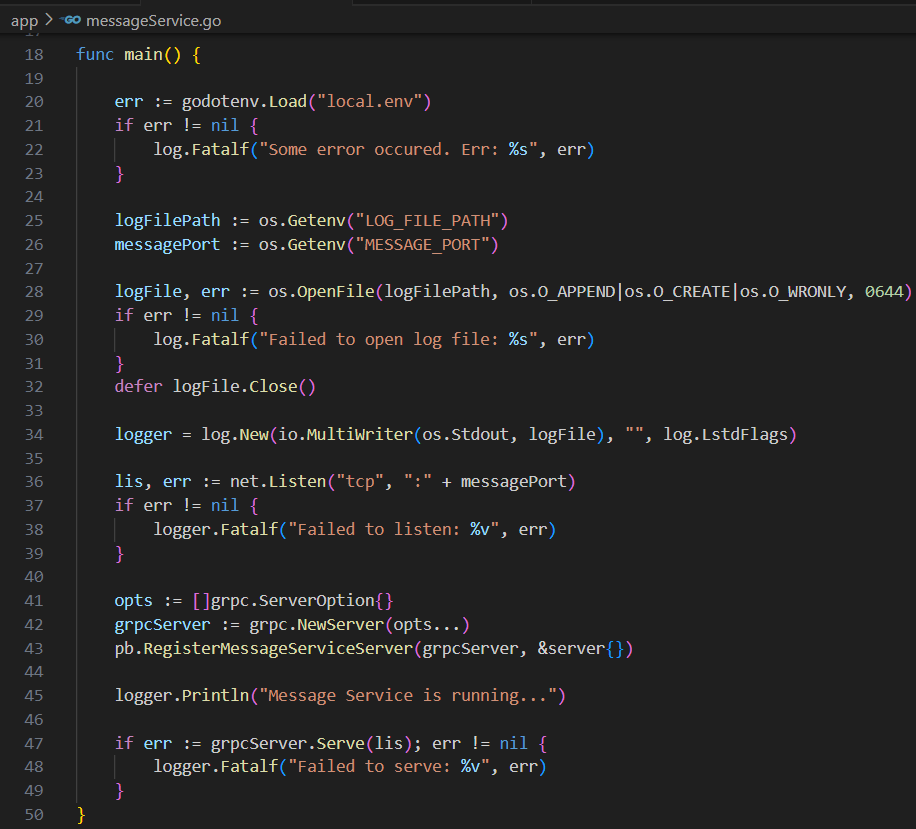


Рисунок 21 – Фрагмент кода messageService



Рисунок 22 – Фрагмент кода userService

Результат работы программы представлен на рисунках 23, 24.

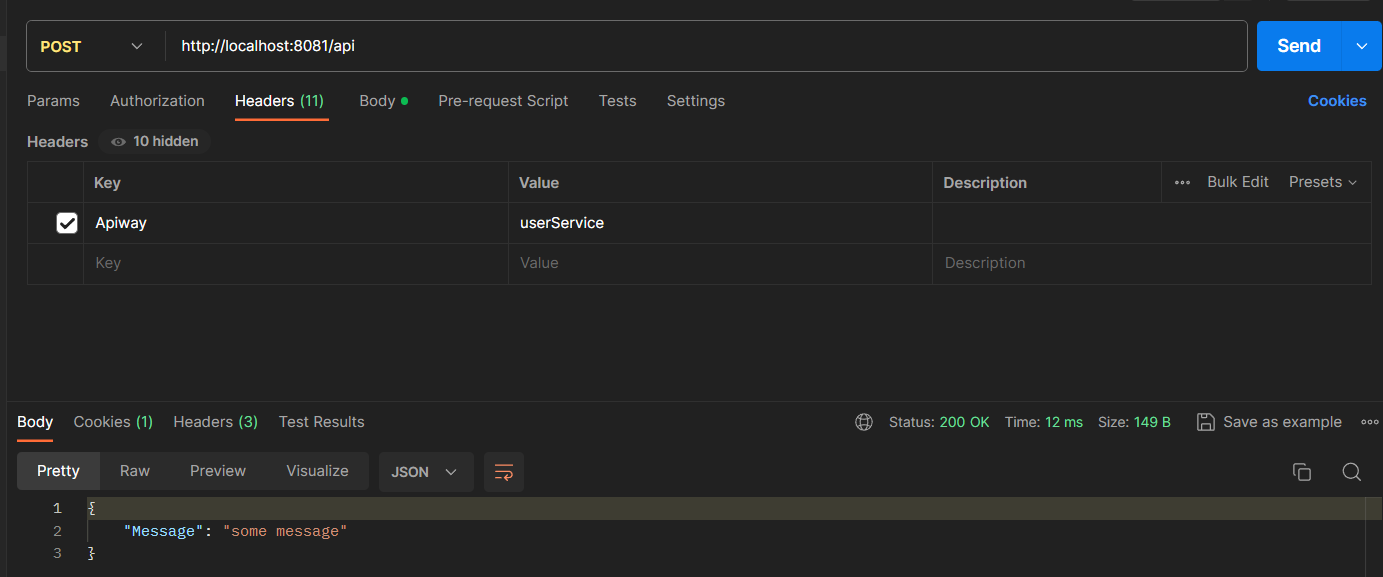


Рисунок 23 – Результат работы userService

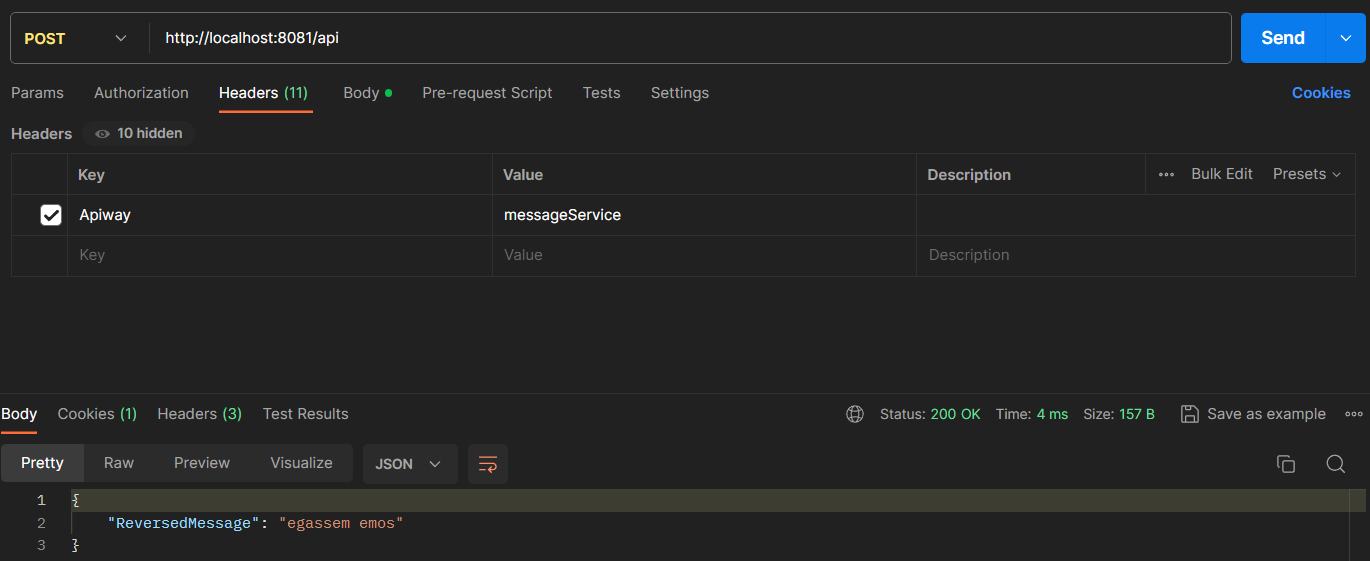


Рисунок 24 – Результат работы messageService

## Вывод

В ходе выполнения практической работы была реализована микросервисная архитектура с использование gRPC и API Gateway.