|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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»

**Тема практической работы: Go**

**Студент группы** ИКБО-03-21 Хречко Сергей Викторович

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

(подпись студента)

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

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

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Допущен к работе «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Москва 2023

**СОДЕРЖАНИЕ**

[1. Практическая работа №8 3](#_Toc26093)

[1.1. Теоретическая часть 3](#_Toc25902)

[1.2. Задание 5](#_Toc13907)

[1.3. Ход работы 6](#_Toc20891)

[1.4. Тестирование 11](#_Toc8523)

[1.5. Вывод 12](#_Toc9361)

[1.6. Ответы на контрольные вопросы 13](#_Toc32495)

[1.7. Использованная литература 15](#_Toc20914)

[2. Практическая работа №9 16](#_Toc18561)

[2.1. Теоретическая часть 16](#_Toc6334)

[2.2. Задание 18](#_Toc17469)

[2.3. Ход работы 19](#_Toc32400)

[2.4. Тестирование 23](#_Toc9433)

[2.5. Вывод 26](#_Toc20262)

[2.6. Ответы на контрольные вопросы 27](#_Toc25077)

[2.7. Использованная литература 28](#_Toc30931)

[3. Практическая работа №10 29](#_Toc30979)

[3.1. Теоретическая часть 29](#_Toc21000)

[3.2. Задание 31](#_Toc621)

[3.3. Ход работы 32](#_Toc9812)

[3.4. Тестирование 35](#_Toc9482)

[3.5. Вывод 37](#_Toc3220)

[3.6. Ответы на контрольные вопросы 38](#_Toc546)

[3.7. Использованная литература 40](#_Toc22373)

# Практическая работа №8

## Теоретическая часть

### Как начать писать на Го

Для работы предлагается использовать Goland от JetBrains (альтернатива – VS Code с различными расширениями для работы с Go).

Для начала установим Go. Качаем последнюю стабильную версию для своей операционной системы с официального сайта (<https://go.dev/dl/>). Затем просто следуем инструкциям установщика.

Рассказать о том, как начать писать на Go вам лучше всего расскажет небольшой обзорный курс от разработчиков – [Go Tour](https://go.dev/tour/welcome/1).

### Cookies

Cookies – информация о пользователе, которую сервер передаёт браузеру при посещении сайта.

Эти данные позволяют идентифицировать посетителя сайта без повторной авторизации. Сервер «вспоминает» пользователя по данным в файле и даёт ему доступ к контенту.

Куки бывают нескольких видов:

1. Сессионные (или временные) cookies — существуют, пока пользователь находится на сайте.
2. Постоянные cookies-файлы сохраняют информацию о пользователе и его действиях, даже если посетитель ушёл с сайта.
3. Сторонние cookies — сторонние файлы принадлежат другому домену.
4. Защищённые cookies — передаются только через шифрованное https-соединение.
5. HttpOnly cookies — не видны коду браузера, отправляются только на сервер компании.

Куки нужны, чтобы сделать использование использование интернета удобным. Например, пользователь может случайно закрыть страницу интернет-магазина с заполненной корзиной, а потом вернуться на сайт и продолжить покупки — заново складывать товары в корзину не придётся.

Также cookies упрощают работу с сервисами, для входа в которые нужно вводить логин и пароль.Файлы cookie помогают собирать и анализировать данные о посетителях сайта.

Маркетологи также используют cookies для рекламы: например, на одном сайте пользователь посмотрел пижаму — теперь на другом сайте он может встретить релевантную рекламу домашних костюмов.

### Шифрование Cookie

Cookie легко [перехватить](https://ru.wikipedia.org/wiki/Сниффер) и подменить (например, для получения доступа к учётной записи), если пользователь использует нешифрованное соединение с сервером. В группе риска — пользователи, выходящие в интернет при помощи публичных точек доступа [Wi-Fi](https://ru.wikipedia.org/wiki/Wi-Fi) и не использующие при этом таких механизмов, как [SSL](https://ru.wikipedia.org/wiki/SSL) и [TLS](https://ru.wikipedia.org/wiki/TLS). Шифрование позволяет также решить и другие проблемы, связанные с безопасностью передаваемых данных.

Как известно, Cookies приходят в хеддере Set-Cookie в формате ключ-значение. Как мы можем их зашифровать?

Первый этап – base64. Кодирование значения Cookie в base64 позволит нам хранить в нем не только текстовые данные с символами из таблицы ASCII, но класть в них любые бинарные данные, в первую очередь Unicode символы.

Однако base64 сможет раскодировать любой пользователь, который знает, как посмотреть cookies, он также может поменять данные, которые был записаны в cookies. Поэтому, чтобы гарантировать целостность cookie, мы можем подписывать cookie с помощью HMAC сигнатуры, которае высчитывается на основе значения. Чаще всего используют SHA256 кодирование. И по итогу значение cookie склеивается со значением этой сигнатуры и затем кодируется в base64.

Но при использовании HMAC сигнатур пользователь все также может прочитать значение. Поэтому лучше использовать AES шифрование с закрытым ключом, а точнее AES-GSM.

## 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.

time.Sleep() предлагается использовать, так как показать реальную временную нагрузку на сервис будет сложно без походов в базу или другие микросервисы.

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.).

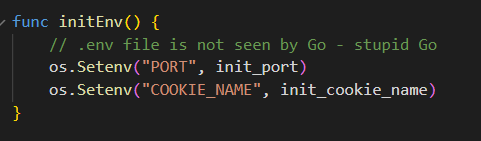
Не стоит забывать и про логирование ошибок, они должны писаться в поток стандартного вывода и в файл, который можно будет прочитать извне контейнера.

Приветствуется использование различных библиотек.

## Ход работы

### Получение конфигурации

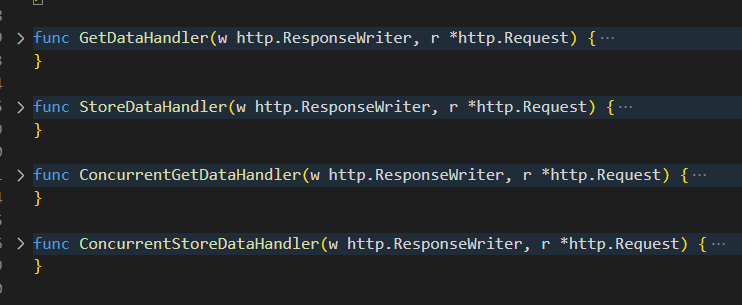
На рисунке 1 представлен код, который загружает конфиги из файла с конфигами.



1. Код для работы с параметрами приложения

### Запросы

На рисунке 2 представлены функции обрабатывающие запросы, связанные с куки. Как и требуется по заданию есть два типа – линейные и конкурентные.

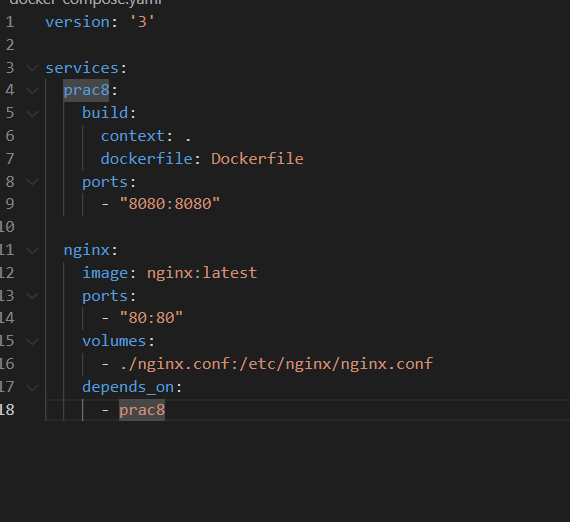


1. Функции обработки запросов

Для работы с cookie будем использовалась библиотека SecureCookie.

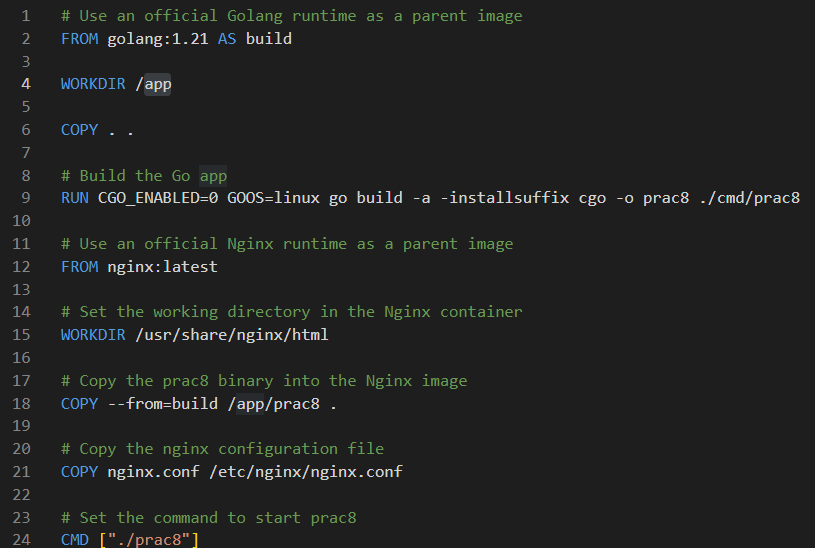
### Развёртывание

На рисунке 3 представлен docker-compose файл.



1. docker-compose файл

На рисунке 4 представлен dockerfile. В остальных работах будут использоваться похожие файлы, с небольшими изменениями связанными со спецификами работы.

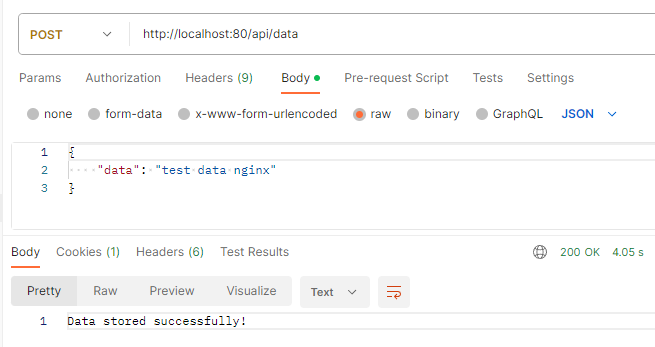


1. dockerfile

## Тестирование

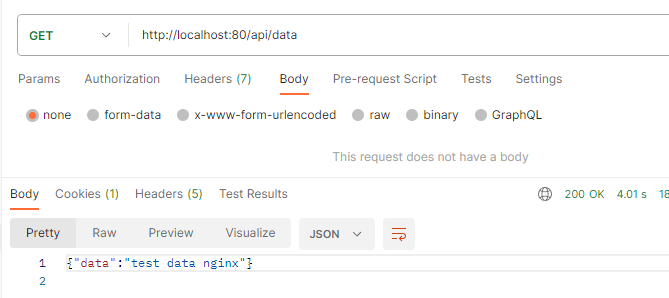
Для тестирования воспользуемся программой postman.

На рисунке 5 представлено тестирования сохранения куки. Обратите внимание на время – 4 секунды, данная функция работает линейно.



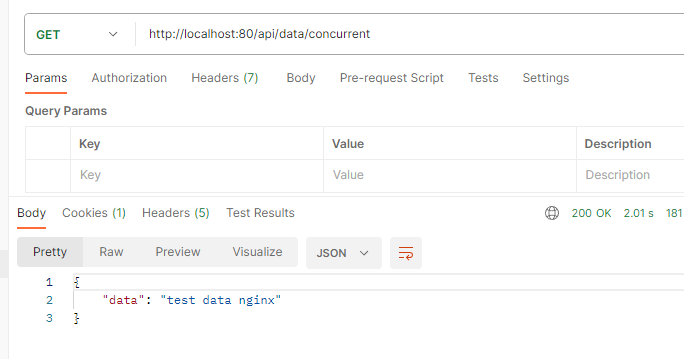
1. Линейное сохранение куки

На рисунке 6 показано получение линейное, мы получили сохраненные данные. Обратите внимание на время – 4 секунды.



1. Получение линейное

На рисунке 7 показано получение куки с использованием конкурентных вычислений в го. Обратите внимание на время – 2 секунды.



1. Получение конкурентное

## Вывод

В ходе данной работы был написан простейший веб-сервер на языке Go, реализующий работу с cookie-файлами. Также была продемонстрирована возможность Go по работе с конкурентными вычислениями. Полученное приложение было развёрнуто в docker-контейнерах.

## Ответы на контрольные вопросы

### Приведите пример, когда будут полезны конкурентные вычисления на Go.

Пример полезного применения конкурентных вычислений на Go: обработка параллельных запросов к внешним API, распараллеливание вычислений в многозадачных приложениях.

### Как в Go обычно происходит работа с JSON и что такоеomitempty?

В Go работа с JSON осуществляется с использованием пакета encoding/json. Для сериализации (преобразования объекта в JSON) и десериализации (преобразования JSON в объект) используются функции json.Marshal и json.Unmarshal соответственно, либо объекты Decoder и Encoder.

omitempty - это тег поля структуры в Go, которая указывает, что поле не должно быть включено в JSON-представление, если его значение равно нулю или нулевому значению по умолчанию.

### Вышел новый супер современный и быстрый логгер, я хочу заменить в своём сервисе старый логгер. У нового логгера немного отличаются сигнатуры методов. Как было бы удобно организовать архитектуру так, чтобы менять пришлось как можно меньше кода?

Свои функции для логгирования использовать. Тогда менять придется только их. В этих функциях выполняется логгирование с помощью логгера. По сути имитируем ооп без ооп.

### Что такое горутина?

Горутина в Go - это легковесная нить выполнения, управляемая Go runtime. Она предоставляет способ выполнять функции асинхронно, параллельно с другими горутинами, что повышает эффективность и масштабируемость программы.

### Что такое HTTPMiddleware, и как стоит с ним работать в Go?

HTTP Middleware в Go - это промежуточное программное обеспечение, которое выполняет операции на уровне HTTP-запроса и ответа перед тем, как они достигнут обработчика запроса. Это может включать в себя логирование, аутентификацию, обработку ошибок и многое другое. Для работы с Middleware в Go, обычно, используют функции, которые принимают и возвращают http.Handler или http.HandlerFunc, позволяя создавать цепочки Middleware для обработки запросов.

## Использованная литература

1. [Go Tour](https://go.dev/tour/welcome/1)
2. [Go Project Layout](https://github.com/golang-standards/project-layout)
3. [Курс на Stepik по Go](https://stepik.org/course/54403/info)
4. [Руководство для начинающих по Nginx](https://nginx.org/ru/docs/beginners_guide.html)
5. [Популярные логгеры в Go](https://blog.logrocket.com/5-structured-logging-packages-for-go/)
6. [Визуализация горутин](https://habr.com/ru/articles/276255/)
7. [Про cookie](https://developer.mozilla.org/ru/docs/Web/HTTP/Cookies)
8. [Стандартное шифрование в Go](https://pkg.go.dev/crypto/cipher)
9. [Библиотека для работы с unit-тестами](https://github.com/stretchr/testify)
10. [Стандартная библиотека net/http](https://pkg.go.dev/net/http)
11. [Статья про стандартные методы работы с HTTP запросами](https://www.alexedwards.net/blog/an-introduction-to-handlers-and-servemuxes-in-go)
12. [Статья про HTTP Middleware](https://www.alexedwards.net/blog/making-and-using-middleware)
13. [Простой пример работы с cookie](https://golangbyexample.com/cookies-golang/)
14. [Еще пример работы с cookie](https://www.alexedwards.net/blog/working-with-cookies-in-go)

# Практическая работа №9

## Теоретическая часть

### MongoDB

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.

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

Формат данных. MongoDB использует собственный формат хранения информации — Binary JavaScript Object Notation (BSON), который построен на основе языка JavaScript.

Документ. Если реляционные БД используют строки, то MongoDB — документы, которые хранят значения и ключи.

Вместо таблиц MongoDB использует коллекции. Они содержат разные типы наборов данных

Репликация. Система хранения информации в СУБД представлена узлами. Существует один главный и множество вторичных.

Индексация. Технология применяется к любому полю в документе на усмотрение пользователя. Проиндексированная информация обрабатывается быстрее.

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.

СУБД осуществляет поиск по специальным запросам. Например, пользователь может создать диапазонный запрос и мгновенно получить ответ.

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

### GridFS

GridFS — это спецификация для хранения и извлечения файлов, размер которых превышает предельный размер документа BSON, равный 16 МБ.

Вместо того, чтобы хранить файл в одном документе, GridFS делит файл на части или фрагменты и сохраняет каждый фрагмент в виде отдельного документа. По умолчанию GridFS использует размер фрагмента по умолчанию 255 КБ; то есть GridFS делит файл на фрагменты по 255 КБ, за исключением последнего фрагмента. Последний кусок настолько велик, насколько это необходимо. Точно так же файлы, размер которых не превышает размер фрагмента, имеют только последний фрагмент, используя столько места, сколько необходимо, плюс некоторые дополнительные метаданные.

GridFS использует две коллекции для хранения файлов. В одной коллекции хранятся фрагменты файлов, а в другой — метаданные файлов. В разделе Коллекции GridFS подробно описывается каждая коллекция.

Когда вы запрашиваете GridFS для файла, драйвер повторно собирает фрагменты по мере необходимости. Вы можете выполнять запросы диапазона к файлам, хранящимся в GridFS. Вы также можете получить доступ к информации из произвольных разделов файлов, например, «перескочить» к середине видео- или аудиофайла.

GridFS удобна не только для хранения файлов, размер которых превышает 16 МБ, но и для хранения любых файлов, к которым вам нужен доступ, без необходимости загрузки всего файла в память.

## 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.

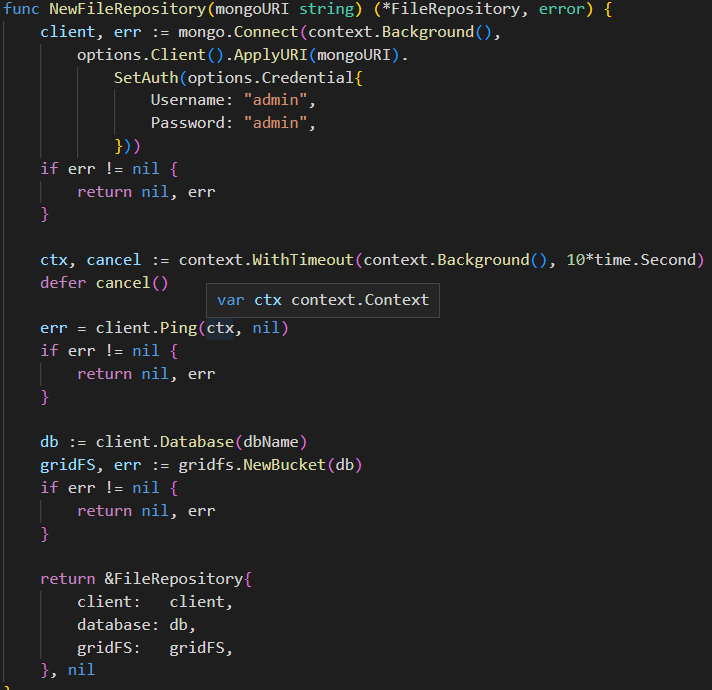
## Ход работы

### Предисловие

Так как данная работа делалась на основе предыдущей, то общие элементы реализованы одинакого и их можно посмотреть в предыдущей работе.

### Подключение к MongoDB

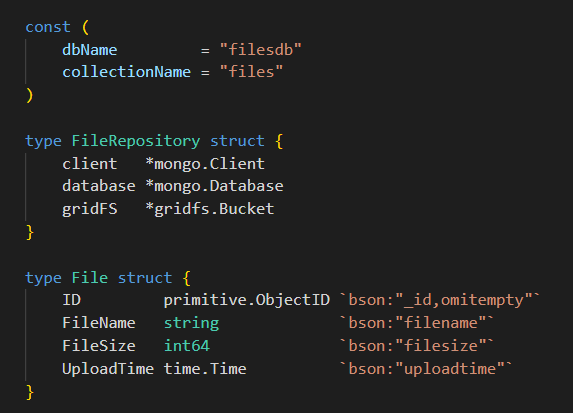
На рисунке 8 представлено подключение базе данных MongoDB.



1. Подключение к MongoDB

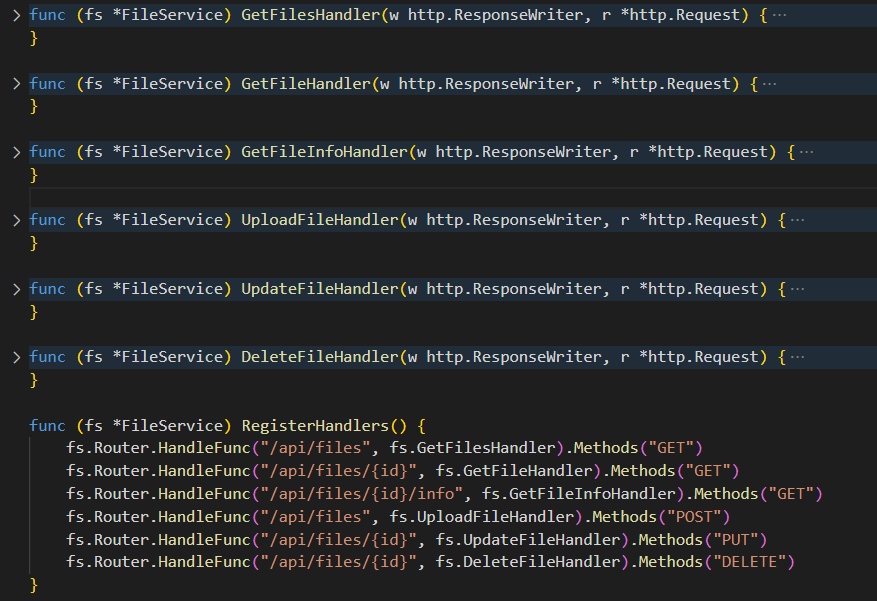
### Работа с файлами

На рисунке 9 представлены константы и структуры, которые были использованы.

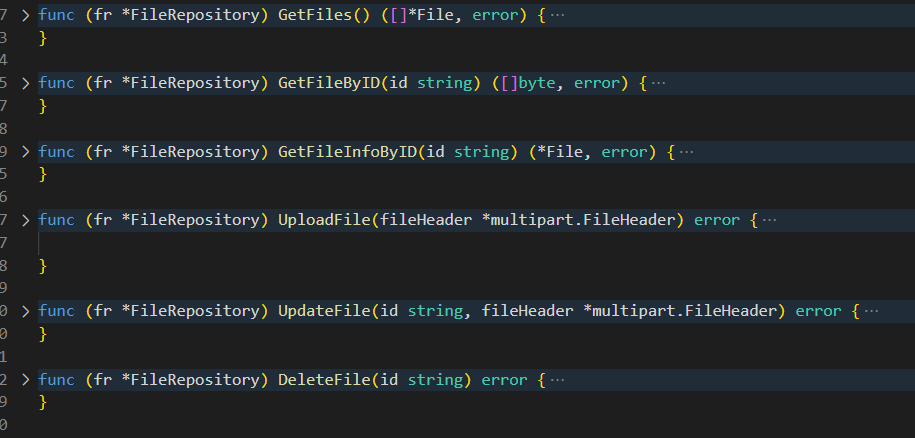


1. Константы и структуры

На рисунке 10 представлены функции, отвечающие за обработку запросов. Также, ниже, представлена функция которая настраивает все пути для api и соответствующие функции для вызова. На рисунке 11 представлены функции, которые вызываются из функций-обработчиков, как раз-таки они ответственны за работу с файлами.



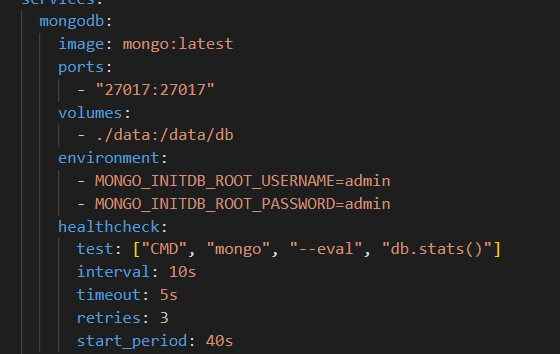
1. Функции обработки запросов и настройка



1. Функции обработки файлов

### Развёртывание

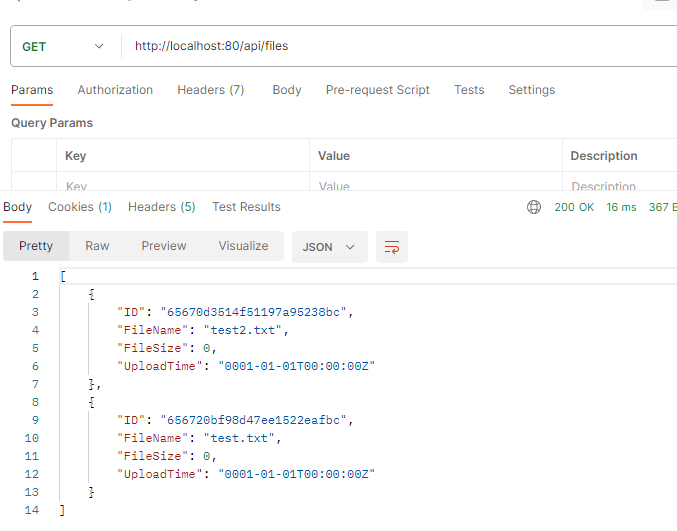
На рисунке 12 представлены добавления, внесенные в docker-compose. Они нужны, чтобы подключить MongoDB.



1. Изменения в docker-compose

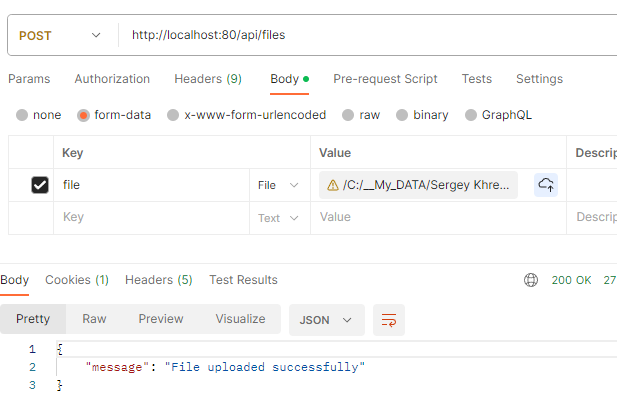
## Тестирование

На рисунке 13 представлено получение файлов.



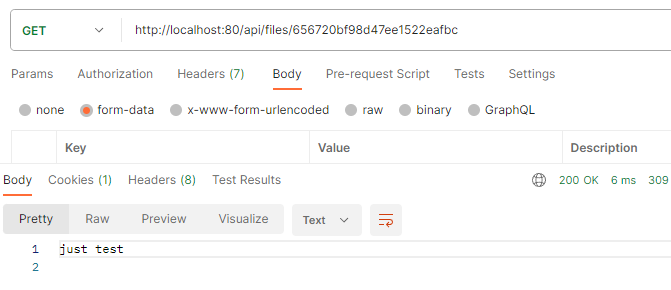
1. Получение файлов

На рисунке 14 представлена загрузка файла в базу данных.



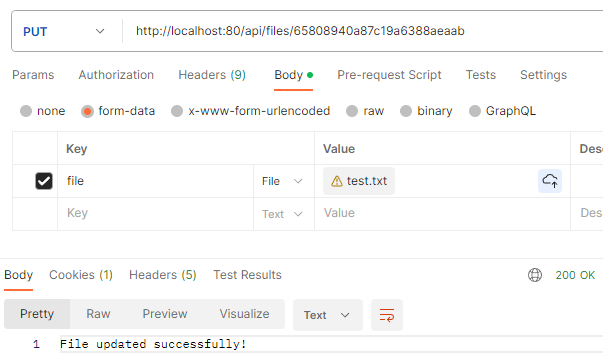
1. Загрузка файла в базу

На рисунке 15 представлено скачивание файла.



1. Скачивание файла

На рисунке 16 представлено обновление файла.



1. Обновление файла

## Вывод

В ходе данной работы был написан простейший веб-сервер на языке Go, реализующий хранение файлов с использованием GridFS. Тем самым была продемонстрирована возможность Go по работе с MongoDB. Приложение было развёрнуто в docker-контейнерах.

## Ответы на контрольные вопросы

### Зачем могут пригодиться NoSQL базы данных?

NoSQL базы данных могут быть полезны, когда необходимо хранить и обрабатывать большие объемы неструктурированных или полуструктурированных данных. Они обеспечивают гибкость в моделировании данных, легко масштабируются горизонтально и часто применяются в случаях, когда требуется высокая производительность при работе с большими объемами данных.

### Что такое контекст в Go?

В Go, контекст (context) представляет собой объект, используемый для управления сроками выполнения операций и передачи значений и ошибок между горутинами. Контексты могут использоваться для отслеживания отмены операций, передачи значений и управления сроками выполнения.

### Какие еще NoSQL СУБД вы знаете?

Кроме MongoDB, существует множество других NoSQL баз данных например Redis.

### Когда могут пригодиться мьютексы?

Мьютексы (mutexes) используются для обеспечения безопасности при доступе к общим данным из нескольких горутин. Они могут быть полезны в случаях, когда несколько горутин могут одновременно пытаться изменить общие данные, и без синхронизации могут возникнуть проблемы согласованности и целостности данных.

### Как лучше всего отправлять файлы между микросервисами?

Для передачи файлов между микросервисами можно использовать различные методы. Один из распространенных способов - это использование системы хранения файлов (например GridFS) и передача ссылок на файлы между микросервисами. Другой вариант - использование протоколов передачи файлов, таких как FTP или HTTP, для прямой передачи файлов между сервисами.

## Использованная литература

1. [Go Tour](https://go.dev/tour/welcome/1)
2. [Go Project Layout](https://github.com/golang-standards/project-layout)
3. [Курс на Stepik по Go](https://stepik.org/course/54403/info)
4. [Популярные логгеры в Go](https://blog.logrocket.com/5-structured-logging-packages-for-go/)
5. [Библиотека для работы с unit-тестами](https://github.com/stretchr/testify)
6. [Стандартная библиотека net/http](https://pkg.go.dev/net/http)
7. [Статья про стандартные методы работы с HTTP запросами](https://www.alexedwards.net/blog/an-introduction-to-handlers-and-servemuxes-in-go)
8. [Статья про HTTP Middleware](https://www.alexedwards.net/blog/making-and-using-middleware)
9. [Статья про работу с MongoDB в Go](https://blog.logrocket.com/integrating-mongodb-go-applications/)
10. [Еще статья про работу с MongoDB в Go](https://www.digitalocean.com/community/tutorials/how-to-use-go-with-mongodb-using-the-mongodb-go-driver)
11. [Официальная статья про работу с GridFS](https://www.mongodb.com/docs/drivers/go/current/fundamentals/gridfs/)
12. [Пример отправки файла в GridFS](https://stackoverflow.com/a/33976261)

# Практическая работа №10

## Теоретическая часть

### API RPC

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.

### gRPC

Как вариант архитектуры RPC, gRPC был создан Google для ускорения передачи данных между микросервисами и другими системами, которым необходимо взаимодействовать друг с другом. По сравнению с REST API, gRPC API уникальны в следующих отношениях:

* Protobuf вместо JSON
* Построен на HTTP/2 вместо HTTP/1.1
* Создание собственного кода вместо использования сторонних инструментов, таких как Swagger
* Передача сообщений в 7-10 раз быстрее

Более долгая имплементация и реализация, чем REST

### Protobuf

В отличие от REST и RPC, gRPC преодолевает проблемы, связанные со скоростью и весом, и предлагает большую эффективность при передаче сообщений, используя формат обмена сообщениями Protobuf. Вот несколько подробностей о Protobuf:

* Независимость от платформы и языка
* Сериализует и десериализует структурированные данные для передачи в двоичном формате.
* Поскольку он является сильно сжатым форматом, он не обеспечивает удобочитаемости JSON.
* Ускоряет передачу данных, удаляя множество обязанностей, которыми управляет JSON, поэтому он может сосредоточиться исключительно на сериализации и десериализации данных.
* Передача данных происходит быстрее, потому что Protobuf уменьшает размер сообщений и служит легким форматом обмена сообщениями.

Формат сообщений жестко задан proto файлами как со стороны сервера, так и со стороны клиента.

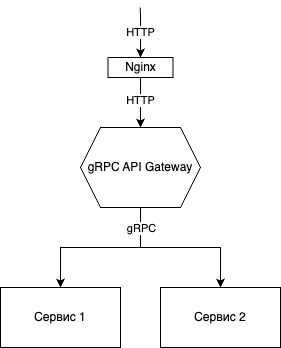
### gRPC API Gateway

Однако, сегодня еще нельзя создать общедоступный сервис, который будет общаться с пользователями, используя HTTP/2 и gRPC. Чтобы решить эту проблему мы создадим APIGateway, 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

Вам 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:



1. 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.

Приветствуется использование различных библиотек.

## Ход работы

Процесс переноса микросервисов на gRPC показан на примере одного из сервисов: cookieService.

### Определение сообщений gRPC

Определим возможные сообщения в файле cookieService.proto, рисунок 18.



1. Содержимое файла cookieService.proto

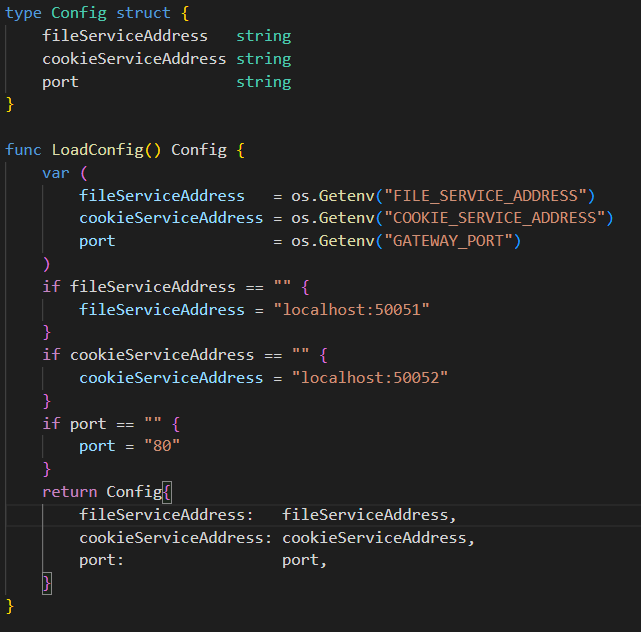
Для генерации go кода, соответствующего определению будем использовать утилиту buf, которую можно установить посредством npm командой “npm i -g buf”. Команда генерации кода показана на рисунке 19.



1. Генерация go кода

### Реализация Gateway

Определим структуру конфигурации и функцию получения её из переменных среды, рисунок 20.



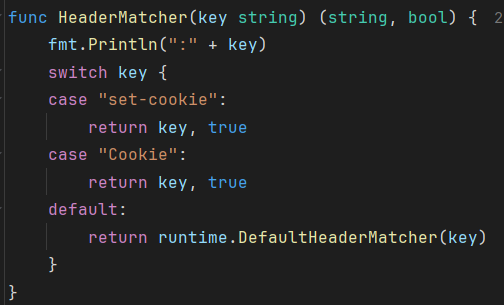
1. Получение конфигурации

Определим функцию запуска шлюза, рисунок 32.



1. Функция запуска шлюза

Здесь используется кастомный Matcher для трансляции имён http заголовков в имена gRPC метаданных, рисунок 22.



1. Функция сопоставления http заголовков gRPC метаданным

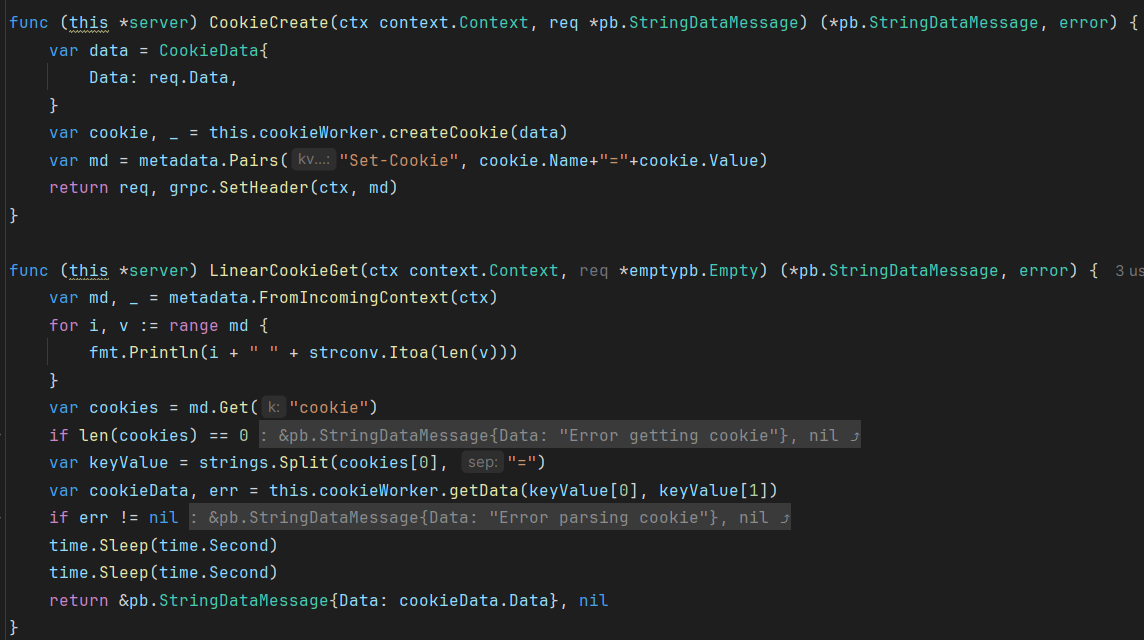
### Реализация сервиса на gRPC

Определим обработчики gRPC запросов.Для этого обновим метод SetupHandlers для запуска gRPC сервера, рисунок 23.



1. Создание gRPC сервера

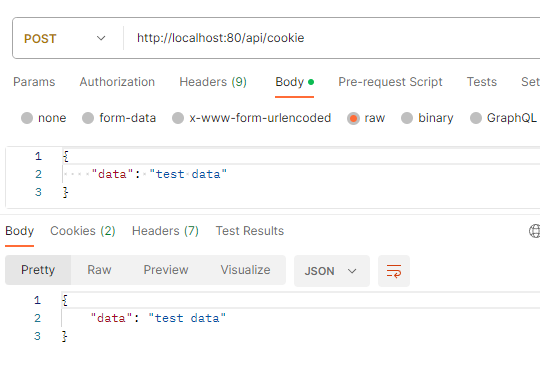
Также обновим обработчики запросов, чтобы они обрабатывали gRPC запросы вместо http, рисунок 24.



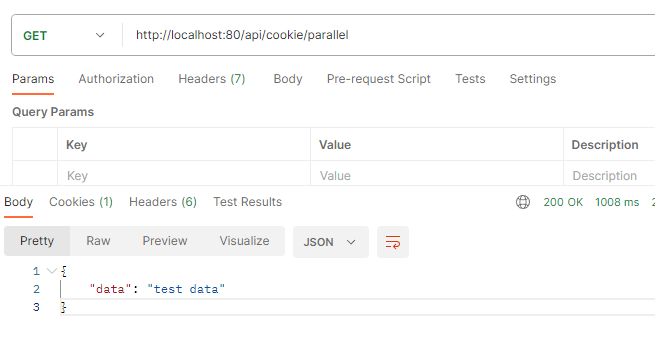
1. ОбработчикиgRPC запросов

## Тестирование

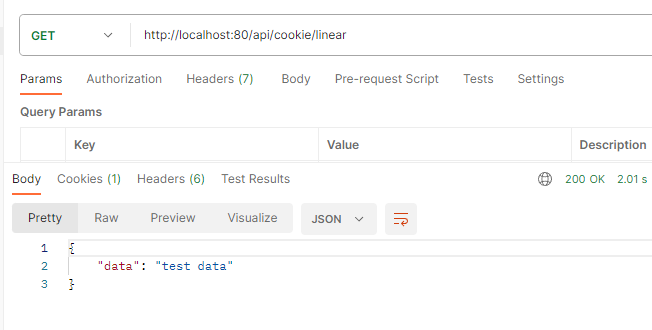
Протестируем, что шлюз корректно транслирует запросы. Для этого повторим запросы из 8 работы, рисунки 25–27.



1. Запрос на создание cookie



1. Запрос на параллельное получение cookie



1. Запрос на последовательное получение cookie

## Вывод

В ходе данной работы был написан простейший шлюз для преобразования http запросов в gRPC. Тем самым была продемонстрирована возможность Go по работе с gRPC. Приложение было развёрнуто в docker-контейнерах.

## Ответы на контрольные вопросы

### Какое главное отличие HTTP/2?

Одним из основных отличий HTTP/2 от предыдущей версии, HTTP/1.1, является использование мультиплексирования. В HTTP/2 несколько запросов и ответов могут передаваться одновременно по одному соединению, что повышает эффективность передачи данных. Кроме того, HTTP/2 поддерживает сжатие заголовков, что уменьшает объем передаваемых данных, а также вводит другие оптимизации, такие как предварительная загрузка ресурсов.

### Расскажите, как использовать gRPC в Go?

В Go, использование gRPC включает в себя следующие шаги:

Определение сервиса и сообщений в файле Protocol Buffers (.proto).

Генерация Go-кода из файла .proto с использованием компилятора Protocol Buffers.

Реализация сервера, который удовлетворяет сгенерированному интерфейсу.

Создание клиента для взаимодействия с сервером.

### Как передавать массивы с помощью gRPC?

Для передачи массивов используется ключевое слово repeated, например: message MyMessage { repeated int32 my\_array = 1; }

### Зачем нужен API Gateway в микросервисной архитектуре?

API Gateway служит как централизованная точка входа для микросервисов. Его задачи включают маршрутизацию запросов, агрегацию данных от различных микросервисов, обеспечение безопасности, управление версиями API, кеширование и другие функции. API Gateway упрощает управление и мониторинг API, а также обеспечивает единый интерфейс для клиентов.

### Как передавать опциональные поля по gRPC?

В Protocol Buffers можно использовать ключевое слово optional для указания опциональных полей, но в gRPC оно не является обязательным. Вместо этого, все поля в Protocol Buffers по умолчанию являются опциональными. Поэтому, если вы не укажете значение для поля в сообщении, оно будет считаться отсутствующим. В коде на Go, проверка наличия значения в опциональном поле осуществляется с использованием сгенерированных методов проверки на nil.

## Использованная литература

1. [Go Tour](https://go.dev/tour/welcome/1)
2. [Go Project Layout](https://github.com/golang-standards/project-layout)
3. [Курс на Stepik по Go](https://stepik.org/course/54403/info)
4. [Популярные логгеры в Go](https://blog.logrocket.com/5-structured-logging-packages-for-go/)
5. [Библиотека для работы с unit-тестами](https://github.com/stretchr/testify)
6. [Стандартная библиотека net/http](https://pkg.go.dev/net/http)
7. [Статья про стандартные методы работы с HTTP запросами](https://www.alexedwards.net/blog/an-introduction-to-handlers-and-servemuxes-in-go)
8. [Статья про HTTP Middleware](https://www.alexedwards.net/blog/making-and-using-middleware)
9. [Официальный сайт gRPC](https://grpc.io/)
10. [Пример работы с gRPC от Google](https://github.com/GoogleCloudPlatform/golang-samples/tree/main/endpoints/getting-started-grpc)
11. [Простой пример создания gRPCAPIGateway](https://blog.logrocket.com/guide-to-grpc-gateway/)
12. [Более сложный пример создания APIGateway](https://habr.com/ru/articles/496574/)