**Практическая работа № 8.**

**Введение**

Вы уже познакомились с такими языками, как Java и PHP. Оба этих языка появились достаточно давно, и сегодня развились на столько, что компиляция или сборка проектов занимает очень много времени, а сложность синтаксических конструкций очень сильно повышает сложность разработки и вероятность ошибки. 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#. Сейчас его применяют ведущие компании по всему миру: известные всем [MAANGMULA](https://jargonism.com/words/1253), российские Яндекс, Авито, VK, Ozon и многие-многие другие. Так чем же так хорош этот язык на сегодняшний день?

Можно выделить несколько самых важных особенностей Go:

1. Простой и понятный синтаксис

У языка лаконичный и простой синтаксис, основанный на С, но существенно доработанный, с большим количеством [синтаксического сахара](https://ru.wikipedia.org/wiki/Синтаксический_сахар). Читать код, написаный на Go, достаточно приятно, что является одним из существенных плюсов данного языка.  [Официальное руководство Go](https://golang.org/ref/spec" \t "_blank) занимает всего 50 страниц, просто читается и содержит примеры. Даже в сам компилятор Go встроено некоторое количество стилистических проверок, которые сильно упрощают работу всей команде разработки.

1. Статическая типизация

Позволяет избежать ошибок, допущенных по невнимательности разработчика, упрощает чтение и понимание кода, делает код однозначным.

1. Быстродействие и производительность

Go разрабатывался с учетом широкого распространения многоядерных процессоров. Из коробки поддерживает конкурентность и параллелизм за счет встроенных потоков (горутин) и разных дополнительных инструментов для работы с ними.

Также, Go — компилируемый язык, что делает Go достаточно быстрым языком. Это значит что программы транслируются сразу в машинный код. Это убирает прослойку виртуальной машины, как в случае с интерпретируемыми языками (Python, JavaScript, Ruby и т.д), или семейством JVM (Java, Kotlin, Scala). Официальный компилятор Go поддерживает все Unix-подобные операционные системы, а также Windows. Код на Go не нужно будет менять для работы в разные операционных системах.

1. Наличие сборщика мусора

Работая на Go вам практически не нужно будет думать о том, что происходит с неиспользуемыми объектами (кроме случаев, когда сборщик мусора будет тормозить работу программы, но вы с этим столкнетесь при 1000-х RPM). За вас это сделает Garbage Collector.

1. Go – не ООП язык

Но это не компромисс или проблема, которую не решили разработчики языка, а намеренное упрощение. Убрав из языка классы наследование и классы, разработчики еще больше его ускорили. Зато остались интерфейсы и структуры.

1. Достаточно большое комьюнити, которое стремительно развивается

В сети уже существует огромное множество видео, статьей и книг по данному языку, что облегчает процесс обучения и поиска информации.

Также на GitHub можно найти большое количество Open-Source проектов, библиотек и фреймворков, что дает возможность использовать готовые, качественные инструменты в своем коде, а также изучать кодовые базы на других проектах.

Ну и конечно все больше компаний и стартапов выбирают Go, а это значит, что будет проще найти работу, выбрав данный язык.

**Теория**

1. **Как начать писать на Go?**

Как и любой программист, начинающий изучать новый язык, начнем с «Hello World!». Мы не будем рассматривать все тонкости создания первой программы и тонкости работы с языком, просто рассмотрим то, что может пригодиться в практике. 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. После установки необходимо проверить, установился ли язык. Для этого вводим в консоли go version консоли. Если версия не показалась, то скорее всего путь до Go нужно прописать в PATH.

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).

1. **Cookies**

Думаю, многие слышали, что такое Cookies и для чего они в общем нужны. Но на всякий случай повторим.

Cookies – информация о пользователе, которую сервер передаёт браузеру при посещении сайта: время посещения сайта, источник перехода на сайт, геолокацию, идентификаторы товаров в корзине интернет-магазина и другую информацию.

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. Обычно удаляются после того, как пользователь закрывает страницу браузера.
2. Постоянные cookies-файлы сохраняют информацию о пользователе и его действиях, даже если посетитель ушёл с сайта. Этот вид cookies удаляется через определённое время или когда пользователь решит очистить куки. То есть браузер будет передавать информацию на сервер при каждом заходе посетителя на сайт, пока данные не будут удалены.
3. Сторонние cookies — сторонние файлы принадлежат другому домену. Обычно этот тип cookies встраивается на страницы сайтов других компаний, например, в виде рекламного блока. Создатель cookies-файла может собирать и получать данные о посетителях.
4. Защищённые cookies — передаются только через шифрованное https-соединение.
5. HttpOnly cookies — не видны коду браузера, отправляются только на сервер компании, что позволяет предотвратить воровство cookies-файлов.

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. Если бы браузер не мог сохранять информацию, посетителям пришлось бы авторизовываться при каждом входе на сайт.

Файлы cookie помогают собирать и анализировать данные о посетителях сайта. Это позволяет определить и сегментировать целевую аудиторию, узнать о её предпочтениях, интересах, запросах и потребностях. Также куки позволяют отследить самые посещаемые страницы на сайте, ссылки, пользователи попадают на страницы 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.

1. **Немного про шифрование Cookies**

Cookie легко [перехватить](https://ru.wikipedia.org/wiki/Сниффер" \t "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" \t "Wi-Fi) и не использующие при этом таких механизмов, как [SSL](https://ru.wikipedia.org/wiki/SSL" \t "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.

1. **Логирование в Go**

Логирование помогает найти места с ошибками в работающей программе. С его помощью можно записывать сообщения в выводы: в поток вывода операционной системы, файл или сетевое соединение. В Go вывод логов по умолчанию осуществляется в стандартный вывод операционной системы stdout, однако с помощью сторонних пакетов, таких как, например, Zap или Logrus. Эти пакеты позволяют выводить логи структурированно и так, чтобы другие сервисы, такие как Kibana или Prometeus, могли их читать и позволять разработчику их анализировать.

**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.

*Задание со звездочкой, которое ни на что не виляет: покрыть код unit-тестами.*

**Полезные ссылки**

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)

**Вопросы**

1. Приведите пример, когда будут полезны конкурентные вычисления на Go.
2. Как в Go обычно происходит работа с JSON и что такое omitempty?
3. Вышел новый супер современный и быстрый логгер, я хочу заменить в своем сервсие старый логгер. У нового логгера немного отличаются сигнатуры методов. Как было бы удобно организовать архитектуру так, чтобы менять пришлось как можно меньше кода?
4. Что горутина?
5. Что такое HTTP Middleware, и как стоит с ним работать в Go?