Golang Developer. Professional

Разработка сетевых приложений и микросервисов на Go

Длительность курса: 150 академических часов

1 Начало работы с Go

1 Инструментарий и начало работы с Go

Цели занятия:

разобраться с базовым инструментарием Go.
после занятия вы сможете:
приступить к написанию программ на Go.

Краткое содержание:

GOROOT и GOPATH;
процесс установки модулей и сборки программ;
кросс-компиляция;
форматирование кода и линтеры;
воркшоп "как сдать ДЗ".

Домашние задания

1 Hello, OTUS!

Цель: В результате выполнения ДЗ вы напишете первую «hello-world» программу на языке Go.
В данном задании тренируются навыки:
- работы с git, GitHub и GitHub Actions;
- работы с модулями в Go.

Необходимо написать программу, печатающую в стандартный вывод перевернутую фразу
```
Hello, OTUS!
```

Для переворота строки следует воспользоваться возможностями
[golang.org/x/example/stringutil]
(https://github.com/golang/example/tree/master/stringutil).

Кроме этого необходимо исправить **go.mod** так, чтобы для данного модуля работала
команда `go get`, а полученный **go.sum** закоммитить.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw01_hello_otus

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

2 **Тестирование в Go. Часть 1**

Цели занятия:

после занятия вы сможете:
писать юнит-тесты к программам на Go.

Краткое содержание:

тестирование программ на Go;
понятие табличных тестов;
пакет testing и библиотека testify;

---

3 **Элементарные типы данных в Go**

Цели занятия:

узнать элементарные типа языка.
после занятия вы сможете:
использовать основные типы языка;
избегать ошибок при работе со строками и указателями.

Краткое содержание:

элементарные типы;
понятие "zero value";
константы и указатели;
строки, руны и массивы байт;
стандартные функции для работы со строками и Unicode;
преобразование и присвоение типов;
указатели;
передача аргументов по ссылке и по значению.

Домашние задания

1 Распаковка строки

Цель: В результате выполнения ДЗ вы напишете функцию, работающую со строками.
В данном задании тренируются навыки:

- работы со строками и рунами;
- работы с пакетами "strconv", "strings" и "unicode".

Необходимо написать Go функцию,
осуществляющую примитивную распаковку
строки,
содержащую повторяющиеся символы/руны,
например:
* "a4bc2d5e" => "aaaabccddddde"
* "abcd" => "abcd"
* "3abc" => "" (некорректная строка)
* "45" => "" (некорректная строка)
* "aaa10b" => "" (некорректная строка)
* "aaa0b" => "aab"
* "" => ""
* "d\n5abc" => "d\n\n\n\n\nabc"

Как видно из примеров, разрешено использование цифр, но не чисел.

В случае, если была передана некорректная строка, функция должна возвращать ошибку.
При необходимости можно выделять дополнительные функции / ошибки.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw02_unpack_string

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

4 **Массивы, слайсы и словари**

Цели занятия:

узнать про основные структуры данных (СД) языка.
после занятия вы сможете:
пользоваться основными СД языка.
не совершать ошибок при работе с ссылочными типами Go.

Краткое содержание:

массивы и слайсы;
внутренняя структура слайсов и словарей;
работа со слайсами и словарями;
отличия длины и ёмкости (length vs capacity);
различные способы итерации по слайсам и словарям;
сортировка структур данных;
распространённые ошибки и затруднения.

---

5 **Структуры**

Цели занятия:

после занятия вы сможете:
создавать пользовательские типы данных и комбинировать их между собой.

Краткое содержание:

процесс определения структур;
инкапсуляция полей структуры;
определение методов структуры;
вложенные и анонимные структуры;
структурные тэги и их использование в контексте JSON, XML и СУБД.

Домашние задания

1 Частотный анализ

Цель: В результате выполнения ДЗ вы напишете функцию, работающую с текстом.
В данном задании тренируются навыки:
- работы со строками;
- работы со слайсами и словарями.

Необходимо написать Go функцию, принимающую на вход строку с текстом и
возвращающую слайс с 10-ю наиболее часто встречаемыми в тексте словами.

Если слова имеют одинаковую частоту, то должны быть отсортированы **лексикографически**.

* Словом считается набор символов, разделенных пробельными символами.

* Если есть более 10 самых частотых слов (например 15 разных слов встречаются ровно 133 раза,
остальные < 100), то следует вернуть 10 лексикографически первых слов.

* Словоформы не учитываем: "нога", "ногу", "ноги" - это разные слова.

* Слово с большой и маленькой буквы считать за разные слова. "Нога" и "нога" - это разные слова.

* Знаки препинания считать "буквами" слова или отдельными словами.
"-" (тире) - это отдельное слово. "нога," и "нога" - это разные слова.

Пример
```
cat and dog, one dog,two cats and one man
```
Топ 7:
- `and` (2)
- `one` (2)
- `cat` (1)
- `cats` (1)
- `dog,` (1)
- `dog,two` (1)
- `man` (1)

При необходимости можно выделять дополнительные функции / ошибки.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw03_frequency_analysis

Процесс сдачи домашнего задания:

		https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ
6	**Функции и методы**	**Цели занятия:** после занятия вы сможете: объявлять и определять функции; избегать ошибок, связанных с областью видимости. **Краткое содержание:** области видимости; функции: именованные, анонимные, с переменным числом аргументов и пр.; понятие замыкания; методы.
7	**Интерфейсы. Часть 1**	**Цели занятия:** узнать, что такое интерфейсы, как они устроены внутри и как их использовать. **Краткое содержание:** понятие интерфейса, его определение и реализация; внутреннее устройство интерфейса; влияние использования интерфейсов на производительность программы; композиция интерфейсов; пустой интерфейс (interace{}).
8	**Интерфейсы. Часть 2**	**Цели занятия:** рассмотреть интерфейсы с более практической стороны; после занятия вы сможете: пользоваться type assertion и type switch. **Краткое содержание:** значение типа интерфейс и ошибки, связанные с nil; правила присваивания значений переменным типа

интерфейс;
опасное и безопасное приведение типов (type cast);
использование switch в контексте интерфейсов;
слайсы и словари с интерфейсами;
реализация подхода обобщенного программирования (generics) через интерфейсы.

Домашние задания

1 LRU-кэш

Цель: В результате выполнения ДЗ вы реализуете LRU-кэш на основе собственного двусвязного списка :)
В данном задании тренируются навыки:
- работы с интерфейсами;
- работы со стандартными структурами данных Go;
- реализации алгоритмов, полезных для разработчика.

Необходимо реализовать LRU-кэш на основе двусвязного списка.

Задание состоит из двух частей, которые необходимо выполнять последовательно.

1) Реализация двусвязного списка
Список имеет структуру вида
```text
nil <- (prev) front <-> ... <-> elem <-> ... <-> back (next) -> nil
```

Необходимо реализовать следующий интерфейс List:
- Len() int // длина списка
- Front() *ListItem // первый элемент списка
- Back() *ListItem // последний элемент списка
- PushFront(v interface{}) *ListItem // добавить значение в начало
- PushBack(v interface{}) *ListItem // добавить значение в конец
- Remove(i *ListItem) // удалить элемент
- MoveToFront(i *ListItem) // переместить элемент в начало

**Считаем, что методы Remove и MoveToFront

вызываются только от существующих в списке элементов.**

Элемент списка ListItem:
- Value interface{} // значение
- Next *ListItem // следующий элемент
- Prev *ListItem // предыдущий элемент

Сложность всех операций должна быть O(1),
т.е. не должно быть мест, где осуществляется полный обход списка.

2) Реализация кэша на основе ранее написанного списка
Необходимо реализовать следующий интерфейс Cache:
- Set(key Key, value interface{}) bool // Добавить значение в кэш по ключу.
- Get(key Key) (interface{}, bool) // Получить значение из кэша по ключу.
- Clear() // Очистить кэш.

Структура кэша:
- ёмкость (количество сохраняемых в кэше элементов)
- очередь \[последних используемых элементов\] на основе двусвязного списка
- словарь, отображающий ключ (строка) на элемент очереди

Элемент кэша хранит в себе ключ, по которому он лежит в словаре, и само значение.
Для чего это нужно понятно из алгоритма работы кэша (см. ниже).

Сложность операций `Set`/`Get` должна быть O(1), при желании `Clear` тоже можно сделать O(1).

Алгоритм работы кэша:
- при добавлении элемента:
- если элемент присутствует в словаре, то обновить его значение и переместить элемент в начало очереди;
- если элемента нет в словаре, то добавить в словарь и в начало очереди
(при этом, если размер очереди больше ёмкости кэша,
то необходимо удалить последний элемент из очереди и его значение из словаря);
- возвращаемое значение - флаг, присутствовал ли

элемент в кэше.
- при получении элемента:
- если элемент присутствует в словаре, то переместить элемент в начало очереди и вернуть его значение и true;
- если элемента нет в словаре, то вернуть nil и false
(работа с кешом похожа на работу с `map`)

Ожидаются следующие тесты:
- на логику выталкивания элементов из-за размера очереди
(например: n = 3, добавили 4 элемента - 1й из кэша вытолкнулся);
- на логику выталкивания давно используемых элементов
(например: n = 3, добавили 3 элемента, обратились несколько раз к разным элементам: изменили значение, получили значение и пр. - добавили 4й элемент,
из первой тройки вытолкнется тот элемент, что был затронут наиболее давно).

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw04_lru_cache

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

9 **Обработка ошибок. Понятие паники**

Цели занятия:

поговорить про ошибки.
узнать, что такое паника, почему не стоит паниковать и как пользоваться отложенным вызовом функции.
после занятия вы сможете:
правильно обрабатывать ошибки;
создавать собственные типы ошибок.

Краткое содержание:

интерфейс error;
лучшие и худшие практики обработки ошибок;
defer;
функции panic и recover.

---

10 **Тестирование в Go. Часть 2**

Цели занятия:

после занятия вы сможете:
писать юнит-тесты к программам на Go.

Краткое содержание:

моки (для интерфейсов, времени, фс);
faker (для генерации тестовых данных);
тестирование мутированием;
golden files.

2 Concurrency в Go

1 Горутины и каналы

Цели занятия:

начать работу с горутинами.
после занятия вы сможете:
реализовать передачу данных между горутинами с помощью канала.

Краткое содержание:

горутины и как их запускать;
канал и как он устроен внутри;
сравнение буферизированных и небуферизованных каналов;
использование каналов для передачи данных и синхронизации;
оператор select;
таймеры в Go.

---

2 Примитивы синхронизации. Часть 1

Цели занятия:

после занятия вы сможете:
пользоваться частью механизмов синхронизации в Go;
бороться с «гонками» в Go.

Краткое содержание:

группа ожидания (sync.WaitGroup);
гарантировано одноразовое выполнение (sync.Once);
"простой" мьютекс (sync.Mutex);
детектор гонок (race detector).

Домашние задания

1 Параллельное исполнение

Цель: В результате выполнения ДЗ вы реализуете функцию, конкурентно выполняющую заданный список задач.
В данном задании тренируются навыки:
- работы с горутинами и каналами;

- работы с оператором select;
- использования структур синхронизации данных.

Необходимо написать функцию для параллельного выполнения заданий в n параллельных горутинах:
* количество создаваемых горутин не должно зависеть от числа заданий, т.е. функция должна запусать n горутин для параллельно обработки заданий и, возможно, еще несколько вспомогательных горутин;
* функция должна останавливать свою работу, если произошло m ошибок;
* после завершения работы функции (успешного или из-за превышения m) не должно оставаться работающих горутин;
* если задачи работают без ошибок, то выполнятся `len(tasks)` задач (т.е. все задачи);
* если в первых m задачах происходят ошибки, то всего выполнится не более n+m задач.

Нужно учесть, что задания могут выполняться разное время, а длина списка задач `len(tasks)` может быть больше или меньше n.

Значение m <= 0 трактуется на усмотрение программиста:
- или это знак игнорировать ошибки в принципе;
- или считать это как "максимум 0 ошибок", значит функция всегда будет возвращать `ErrErrorsLimitExceeded`;
- на эту логику следует написать юнит-тест.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw05_parallel_execution

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

3 **Примитивы синхронизации. Часть 2**

Цели занятия:

после занятия вы сможете:
пользоваться полным набором механизмов синхронизации.

Краткое содержание:

пул объектов (sync.Pool);
read-write мьютекс (sync.RWMutex);
concurrent-safe словарь (sync.Map);
условные переменные (sync.Cond);
атомарные операции (пакет atomic);
модель памяти в Go.

---

4 **Concurrency patterns**

Цели занятия:

узнать больше о конкурентности в Go.
после занятия вы сможете:
применять concurrency-паттерны на практике.

Краткое содержание:

паттерны синхронизации данных;
функции-генераторы и пайплайн;
работа с многими каналами: or, fanin, fanout, etc.

Домашние задания

1 Пайплайн

Цель: В результате выполнения ДЗ вы реализуете функцию, конкуррентно выполняющую пайплайн заданных функций.
В данном задании тренируются навыки:
- работы с горутинами и каналами;
- работы с оператором select;
- работы с паттернами конкуррентного выполнения программы.

Необходимо реализовать функцию для запуска конкуррентного пайплайна, состоящего из стейджей.

Стейдж - функция, принимающая канал на чтение и отдающая канал на чтение,
а внутри в горутине выполняющая полезную работу:
```golang
func Stage(in <-chan interface{}) (out <-chan interface{}) {
out = make(<-chan interface{})
go func() { /* Some work */ }()
return out
}
```

Особенность пайплайна в том, что обработка последующего элемента входных данных должна происходить **без ожидания завершения всего пайплайна** для текущего элемента.

Т.е. пайплан из 4 функций по 100 мс каждая для 5 входных элементов **должен выполняться гораздо быстрее**, чем за 2 секунды (4 * 100 мс * 5).

Также **должна быть реализована возможность остановить пайплайн** через дополнительный сигнальный канал (`done`/`terminate`/etc.).

При необходимости можно выделять дополнительные функции.

Нельзя менять сигнатуры исходных функций.

Для большего понимания см. тесты.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw06_pipeline_execution

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

5 **Go внутри. Планировщик**

Цели занятия:

познакомиться с устройством планировщика Go;
узнать, когда именно происходит «переключение» горутин.

Краткое содержание:

основные структуры планировщика: P, M, G;
механизм переключения горутин;
обработка системных и сетевых вызовов планировщиком;
Net Poller.

---

6 **Go внутри. Память и сборка мусора**

Цели занятия:

получить базовую информацию об устройстве памяти в Go;
узнать о механизме GC в Go.

Краткое содержание:

структура памяти в Linux процессе;
особенности памяти программы на Go;
выделение и освобождение памяти на стеке и куче;
понятие "escape analysis";
механизм сборки мусора в Go.

---

7 **Разбор домашних заданий и ответы на вопросы. Ч.1**

Цели занятия:

посмотреть на рассуждения и процесс решения задач другим человеком (преподавателем), подметить для себя подходы к решению задач, чтобы лучше справляться с ними на курсе и в жизни.

Краткое содержание:

разберем решения домашних заданий;
посмотрим на типичные ошибки;
узнаем некоторые тонкости, о которых редко задумываются.

3 Стандартные библиотеки и практики

1 Работа с вводом/выводом в Go

Цели занятия:

узнать тип Buffer;
рассмотреть стандартные интерфейсы: Reader, Scanner, Writer, Closer;
узнать блочные устройства, Seeker;
узнать форматированный ввод и вывод: fmt.

Краткое содержание:

стандартные интерфейсы io.Reader, io.Writer и io.Closer;
работа с файлом;
последовательные и произвольные доступы и интерфейс io.Seeker;
буферизация ввода/вывода и оптимизация копирования;
форматированный ввод и вывод (пакет fmt).

Домашние задания

1 Утилита для копирования файлов

Цель: В результате выполнения ДЗ вы реализуете упрощенный аналог dd (man dd).
В данном задании тренируются навыки работы с файловой системой,
происходит первое знакомство с обработкой аргументов командной строки.

Необходимо реализовать утилиту копирования файлов (упрощенный аналог `dd`).

Тулза должна принимать следующие аргументы:
* путь к исходному файлу (`-from`);
* путь к копии (`-to`);
* отступ в источнике (`-offset`), по умолчанию - 0;
* количество копируемых байт (`-limit`), по умолчанию - 0 (весь файл из `-from`).

Особенности:
* offset больше, чем размер файла - невалидная ситуация;

		* limit больше, чем размер файла - валидная ситуация, копируется исходный файл до его EOF; * программа может НЕ обрабатывать файлы, у которых неизвестна длина (например, /dev/urandom); Также необходимо выводить в консоль прогресс копирования в процентах (%), допускается использовать для этого стороннюю библиотеку. Юнит-тесты могут использовать файлы из `testdata` (разрешено добавить свои, но запрещено удалять имеющиеся) и должны чистить за собой создаваемые файлы (или работать в `/tmp`). При необходимости можно выделять дополнительные функции / ошибки. Подробности: https://github.com/OtusGolang/home_work/tree/master/hw07_file_copying Процесс сдачи домашнего задания: https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ
2	**Форматирование данных**	**Цели занятия:** после занятия вы сможете: сериализовывать и десериализовывать данные различных форматов стандартными средствами языка и сторонними библиотеками. **Краткое содержание:** кодировки quoted-printable и base64; текстовые форматы JSON, XML и YAML; использование структур и интерфейсов для парсинга данных; сравнение бинарных сериализаторов: gob, msgpack и protobuf.
3	**Взаимодействие**	**Цели занятия:**

с OS

после занятия вы сможете:
работать с операционной системой из программы на Go.

Краткое содержание:

обработка аргументов командной строки: flags, pflag, cobra;
работа с переменными окружения;
запуск внешних программ;
создание временных файлов;
обработка сигналов.

Домашние задания

1 Утилита envdir

Цель: В результате выполнения ДЗ вы реализуете утилиту envdir (man envdir) на Go.
В данном задании тренируются навыки:
- работы с переменными окружения;
- запуска других процессов из программы на Go.

Необходимо реализовать утилиту `envdir` на Go.

Эта утилита позволяет запускать программы, получая переменные окружения из определенной директории:
- если директория содержит файл с именем `S`, первой строкой которого является `T`, то `envdir` удаляет переменную среды с именем `S`, если таковая существует, а затем добавляет переменную среды с именем `S` и значением `T`;
- имя `S` не должно содержать `=`; пробелы и табуляция в конце `T` удаляются; терминальные нули (`0x00`) заменяются на перевод строки (`\n`);
- если файл полностью пустой (длина - 0 байт), то `envdir` удаляет переменную окружения с именем `S`.

Пример использования:
```bash
$ go-envdir /path/to/env/dir command arg1 arg2
```
Если в директории `/path/to/env/dir` содержатся файлы:
* `FOO` с содержимым `123`;

* `BAR` с содержимым `value`,

то вызов выше эквивалентен вызову
```bash
$ FOO=123 BAR=value command arg1 arg2
```

Также необходимо, чтобы:
* стандартные потоки ввода/вывода/ошибок пробрасывались в вызываемую программу;
* код выхода утилиты совпадал с кодом выхода программы.

При необходимости можно выделять дополнительные функции / ошибки.

Юнит-тесты могут использовать файлы из `testdata` или создавать свои директории / файлы,
которые **обязаны** подчищать после своего выполнения.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw08_envdir_tool

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

4 **Рефлексия**

Цели занятия:

познакомиться с механизмом рефлексии в Go.

Краткое содержание:

использование пакета reflect;
плюсы и минусы рефлексии;
reflect.Type и reflect.Value;
примеры использования рефлексии;
пакет unsafe и тип unsafe.Pointer.

---

5 **Кодогенерация в Go**

Цели занятия:

познакомиться с механизмом кодогенерации в Go;
после занятия вы сможете:
не бояться библиотек, использующих кодогенерацию.

Краткое содержание:

понятие кодогенерации;
инструмент `go generate`;
полезные библиотеки, использующие кодогенерацию: impl, stringer, jsonenums, easyjson и пр.;
решение проблемы обобщенного программирования (generics) с помощью кодогенерации;
спецификация Protobuf.

Домашние задания

1 Валидатор структур

Цель: В результате выполнения ДЗ вы реализуете функцию-валидатор структур на языке Go.
В данном задании тренируются навыки:
- работы с рефлексией;
- написания тестов и работы с ошибками.

Необходимо реализовать функцию
```golang
func Validate(v interface{}) error
```
, валидирующую публичные поля входной структуры на основе структурного тэга `validate`.

Функция может возвращать
- или программную ошибку, произошедшую во время валидации;
- или `ValidationErrors` - ошибку, являющуюся слайсом структур, содержащих имя поля и ошибку его валидации.

Таким образом, нужно накопить все ошибки валидации, а не прерывать валидацию на первой ошибке.

Если у поля нет структурных тэгов или нет тэга `validate`, то функция игнорирует его.

Типы полей, которые обязательно должны поддерживаться:

- `int`, `[]int`;
- `string`, `[]string`.

При желании можно дополнительно поддержать любые другие типы (на ваше усмотрение).

Необходимо реализовать следующие валидаторы:
- Для строк:
* `len:32` - длина строки должна быть ровно 32 символа;
* `regexp:\\d+` - согласно регулярному выражению строка должна состоять из цифр (`\\` - экранирование слэша);
* `in:foo,bar` - строка должна входить в множество строк {"foo", "bar"}.
- Для чисел:
* `min:10` - число не может быть меньше 10;
* `max:20` - число не может быть больше 20;
* `in:256,1024` - число должно входить в множество чисел {256, 1024};
- Для слайсов валидируется каждый элемент слайса.

При желании можно дополнительно добавить парочку новых правил (на ваше усмотрение).

Допускается комбинация валидаторов по логическому "И" с помощью `|`, например:
* `min:0|max:10` - число должно находится в пределах [0, 10];
* `regexp:\\d+|len:20` - строка должна состоять из цифр и иметь длину 20.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw09_struct_validator

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

6 **Файлы конфигурации и логирование**

Цели занятия:

познакомиться с механизмами логирования и конфигурирования в Go.

Краткое содержание:

различные варианты конфигурации программы;
использование простых форматов конфигурации: .ini, .yaml, .json и т.п.;
чтение конфигурации из окружения;
библиотека для работы с конфигурацией: viper и confita;
стандартная библиотека для логирования;
расширенное логирование с помощью Zap.

---

7 **Профилирование и оптимизация Go программ**

Цели занятия:

познакомиться с инструментами профилирования в Go.
после занятия вы сможете:
писать бенчмарки в Go;
профилировать программы на Go;
использовать практики оптимизации кода на Go.

Краткое содержание:

бенчмарки в Go;
различные оптимизации в Go: преаллокации, переиспользование объектов, работа со строками, регулярными выражениями и пр.;
профилирование в Go;
`go tool proof` и `go tool trace`: профилирование CPU и памяти, flame-диаграммы, дизассемблирование и пр.

Домашние задания

1 Оптимизация программы

Цель: В результате выполнения ДЗ вы оптимизируете предоставленную вам программу. В данном задании тренируются навыки профилирования и оптимизации кода.

Вам дан исходный код функции `GetDomainStat(r io.Reader, domain string)`, которая:
* читает построчно из `r` пользовательские данные вида
```text
{"Id":1,"Name":"Howard Mendoza","Username":"0Oliver","Email":"aliquid_qui_ea@Browsedrive.gov","Phone":"6-866-899-36-79","Password":"InAQJvsq","Address":"Blackbird Place 25"}
{"Id":2,"Name":"Brian Olson","Username":"non_quia_id","Email":"FrancesEllis@Quinu.edu","Phone":"237-75-34","Password":"cmEPhX8","Address":"Butterfield Junction 74"}
{"Id":3,"Name":"Justin Oliver Jr. Sr.","Username":"oPerez","Email":"MelissaGutierrez@Twinte.gov","Phone":"106-05-18","Password":"f00GKr9i","Address":"Oak Valley Lane 19"}
```
(осторожно, в отличие от конкретной строки файл целиком не является валидным JSON);
* подсчитывает количество email-доменов пользователей на основе домена первого уровня `domain`.

Например, для данных, представленных выше:
```text
GetDomainStat(r, "com") // {}
GetDomainStat(r, "gov") // {"browsedrive": 1, "twinte": 1}
GetDomainStat(r, "edu") // {"quinu": 1}
```

Для большего понимания см. исходный код и тесты.

Необходимо оптимизировать программу таким образом, чтобы она проходила все тесты.

Нельзя:
- изменять сигнатуру функции `GetDomainStat`;
- удалять или изменять существующие юнит-тесты.

Можно:
- писать любой новый необходимый код;
- удалять имеющийся лишний код (кроме

функции `GetDomainStat`);
- использовать сторонние библиотеки по ускорению анмаршалинга JSON;
- добавлять юнит-тесты.

Обратите внимание на запуск TestGetDomainStat_Time_And_Memory
```bash
go test -v -count=1 -timeout=30s -tags bench .
```

Здесь используется билд-тэг bench, чтобы отделить обычные тесты от тестов производительности.

Оформление пул-риквеста
В идеале к подобным пул-риквестам пишут бенчмарки и прикладывают результаты работы benchstat, чтобы сразу было видно, что стало лучше и насколько.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw10_program_optimization

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

4 Работа с сетью и БД

1 Контекст и низкоуровневые сетевые протоколы

Цели занятия:

после занятия вы сможете:
работать с контекстом в Go;
использовать инструменты отладки сетевых проблем.

Краткое содержание:

контекст в Go (пакет context);
сравнение протоколов TCP и UDP;
стандартные типы net.Dialer и net.Conn;
возможные сетевые проблемы и их отладка;
обеспечение сетевых таймаутов.

Домашние задания

1 Клиент TELNET

Цель: В результате выполнения ДЗ вы реализуете крайне примитивный клиент TELNET.
В данном задании тренируются навыки:
- работы с сетью;
- работы с читателями (io.Reader) и писателями (io.Writer).

Необходимо реализовать крайне примитивный TELNET клиент
(без поддержки команд, опций и протокола в целом).

Примеры вызовов:
```bash
$ go-telnet --timeout=10s host port
$ go-telnet mysite.ru 8080
$ go-telnet --timeout=3s 1.1.1.1 123
```

* Программа должна подключаться к указанному хосту (IP или доменное имя) и порту по протоколу TCP.
* После подключения STDIN программы должен записываться в сокет,
а данные, полученные из сокета, должны выводиться в STDOUT - всё происходит

конкурентно.
* Опционально в программу можно передать таймаут на подключение к серверу
(через аргумент `--timeout`) - по умолчанию `10s`.
* При нажатии `Ctrl+D` программа должна закрывать сокет и завершаться с сообщением.
* При получении `SIGINT` программа должна завершать свою работу.
* Если сокет закрылся со стороны сервера, то при следующей попытке отправить сообщение программа
должна завершаться (допускается завершать программу после "неудачной" отправки нескольких сообщений).
* При подключении к несуществующему серверу, программа должна завершаться с ошибкой соединения/таймаута.

При необходимости можно выделять дополнительные функции / ошибки.

Подробности:
https://github.com/OtusGolang/home_work/tree/master/hw11_telnet_client

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

2 **Работа с SQL**

Цели занятия:

после занятия вы сможете:
работать с SQL в Go.

Краткое содержание:

подключение к СУБД и настройка пула соединений;
выполнение SQL-запросов и получение результатов;
обработка ошибок при работе с sql;
стандартные интерфейсы sql.DB, sql.Rows и sql.Tx;
использование транзакций;
SQL-инъекции и борьба с ними.

Домашние задания

1 Заготовка сервиса Календарь

Цель: В результате выполнения ДЗ вы получите каркас сервиса Календарь.
В данном задании тренируются навыки:
- декомпозиции предметной области;
- построения элементарной архитектуры проекта;
- работы с СУБД и SQL.

Необходимо реализовать скелет сервиса «Календарь», который будет дорабатываться в дальнейшем.

Описание того, к чему мы должны придти, представлено [в техническом задании] (https://github.com/OtusGolang/home_work/blob/master/hw12_13_14_15_calendar/docs/CALENDAR.MD).

В репозитории представлена заготовка сервиса, позволяющая понять, что вообще происходит и получить вектор для дальнейшей доработки.

Этот код можно менять/удалять/добавлять каким-угодно способом по усмотрению разработчика.

Подробности:
https://github.com/OtusGolang/home_work/blob/master/hw12_13_14_15_calendar/docs/12_README.md

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

3 Работа с HTTP

Цели занятия:

после занятия вы сможете:
работать с HTTP в Go.

Краткое содержание:

протокол HTTP;
сравнение HTTP/1.1 и HTTP/2.0;
использование HTTP-клиента;
создание простого HTTP-сервера;
декораторы и middleware;
REST, RPC и OpenAPI;
тестирование HTTP-хэндлеров.

---

4 Работа с gRPC

Цели занятия:

после занятия вы сможете:
писать обратно совместимые Protobuf схемы;
писать gRPC сервисы.

Краткое содержание:

HTTP и gRPC;
Protocol buffers;
прямая и обратная совместимость в Protobuf;
описание API с помощью Protobuf;
создание gRPC клиента и сервера.

Домашние задания

1 «API к Календарю»

Цель: В результате выполнения ДЗ вы разработаете API к созданному ранее Календарю. В данном задании тренируются навыки работы с GRPC и HTTP - построения современного API.

Необходимо реализовать HTTP и GRPC API для сервиса календаря.

Методы API в принципе идентичны методам хранилища и [описаны в ТЗ](./CALENDAR.MD).

Для GRPC API необходимо:
* создать отдельную директорию для Protobuf спецификаций;
* создать Protobuf файлы с описанием всех методов API, объектов запросов и ответов (
т.к. объект Event будет использоваться во многих ответах разумно выделить его в отдельный message);
* создать отдельный пакет для кода GRPC сервера;
* добавить в Makefile команду `generate`; `make generate` - вызывает `go generate`, которая в свою очередь
генерирует код GRPC сервера на основе Protobuf спецификаций;
* написать код, связывающий GRPC сервер с методами доменной области (бизнес логикой);
* логировать каждый запрос по аналогии с HTTP

API.

Для HTTP API необходимо:
* расширить "hello-world" сервер из [ДЗ №12](./12_README.md) до полноценного API;
* создать отдельный пакет для кода HTTP сервера;
* реализовать хэндлеры, при необходимости выделив структуры запросов и ответов;
* сохранить логирование запросов, реализованное в [ДЗ №12](./12_README.md).

Общие требования:
* должны быть реализованы все методы;
* календарь не должен зависеть от кода серверов;
* сервера должны запускаться на портах, указанных в конфиге сервиса.

Можно использовать https://grpc-ecosystem.github.io/grpc-gateway/.

Подробности:
https://github.com/OtusGolang/home_work/blob/master/hw12_13_14_15_calendar/docs/13_README.md

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

5 **Разбор домашних заданий и ответы на вопросы. Ч.2**

Цели занятия:

посмотреть на рассуждения и процесс решения задач другим человеком (преподавателем), подметить для себя подходы к решению задач, чтобы лучше справляться с ними на курсе и в жизни.

Краткое содержание:

разберем решения домашних заданий;
посмотрим на типичные ошибки;
узнаем некоторые тонкости, о которых редко задумываются

5 Микросервисы

1 Монолит и микросервисы

Цели занятия:

получить представление о популярных архитектурах сложных систем.

Краткое содержание:

сравнение монолитной и микросервисной архитектур;
плюсы и минусы микросервисов;
понятие 12-факторного приложения;
serverless.

---

2 Очереди сообщений

Цели занятия:

получить представление об очередях сообщений и событийно-ориентированной архитектуре.
после занятия вы сможете:
работать с RabbitMQ из Go.

Краткое содержание:

очереди сообщений;
событийно-ориентированные архитектуры;
Apache Kafka;
RabbitMQ;
примеры работы с RabbitMQ из Go;
возможные проблемы с очередями: перегрузка, падение обработчиков, сбойные сообщения и пр.

Домашние задания

1 Кроликизация Календаря

Цель: В результате выполнения ДЗ вы получите процессы Рассыльщик и Планировщик (в составе сервиса Календарь),
связанные между собой очередью сообщений.
В данном задании тренируются навыки работы с RabbitMQ и очередями в принципе.

Необходимо реализовать "напоминания" о событиях с помощью RabbitMQ (кролика).
Общая концепция описана в [техническом задании](./CALENDAR.MD).

Порядок выполнения ДЗ:
* установить локально очередь сообщений RabbitMQ (или сразу через Docker, если знаете как);
* создать процесс Планировщик (`scheduler`), который периодически сканирует основную базу данных,
выбирая события о которых нужно напомнить:
- при запуске процесс должен подключаться к RabbitMQ и создавать все необходимые структуры (топики и пр.) в ней;
- процесс должен выбирать сообытия для которых следует отправить уведомление (у события есть соотв. поле),
создавать для каждого Уведомление (описание сущности см. в [ТЗ](./CALENDAR.MD)),
сериализовать его (например, в JSON) и складывать в очередь;
- процесс должен очищать старые (произошедшие более 1 года назад) события.
* создать процесс Рассыльщик (`sender`), который читает сообщения из очереди и шлёт уведомления;
непосредственно отправку делать не нужно - достаточно логировать сообщения / выводить в STDOUT.
* настройки подключения к очереди, периодичность запуска и пр. настройки процессов вынести в конфиг проекта;
* работу с кроликом вынести в отдельный пакет, который будут использовать пакеты, реализующие процессы выше.

Процессы не должны зависеть от конкретной реализации RMQ-клиента.

В результате компиляции проекта (`make build`) должно получаться 3 отдельных исполняемых файла
(по одному на микросервис):
- API (`calendar`);
- Планировщик (`calendar_scheduler`);
- Рассыльщик (`calendar_sender`).

Каждый из сервисов должен принимать путь файлу конфигурации:

```bash
./calendar --config=/path/to/calendar_config.yaml
./calendar_scheduler --
config=/path/to/scheduler_config.yaml
./calendar_sender --
config=/path/to/sender_config.yaml
```

После запуска RabbitMQ и PostgreSQL процессы `calendar_scheduler` и `calendar_sender` должны запускаться без дополнительных действий.

Подробности:
https://github.com/OtusGolang/home_work/blob/master/hw12_13_14_15_calendar/docs/14_README.md

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

3	**Docker**	**Цели занятия:** получить представление о Docker контейнере и образе. после занятия вы сможете: докеризировать своё приложение. **Краткое содержание:** контейнеризация; компоненты Docker: engine, cli, registry; написание Dockerfile и сборка собственных контейнеров; запуск и управление контейнерами; передача конфигурации и получение логов из контейнеров; команды Docker: build, run, up, down, pull, push и пр.; сети в Docker; docker compose.

4	**Тестирование микросервисов**	**Цели занятия:** после занятия вы сможете: писать интеграционные тесты для своего приложения.

Краткое содержание:

сравнение интеграционного и юнит-тестирования;
TDD и BDD;
написание тестов на языке Gherkin;
использование godog для запуска тестов;
использование docker-compose.

Домашние задания

1 Докеризация и интеграционное тестирование Календаря

Цель: В результате выполнения ДЗ вы получите полноценный сервис Календарь, способный запускаться вместе со своим окружением и покрытый интеграционными тестами.

В данном задании тренируются навыки:
- работы с docker, написания Dockerfile'ов;
- работы с docker-compose, написания compose-файлов;
- написания интеграционных тестов к web-сервисам;
- работы с BDD, языком Gherkin, библиотекой github.com/DATA-DOG/godog.

Данное задание состоит из двух частей.

Не забываем про https://github.com/golang-standards/project-layout.

1) Докеризация сервиса
Необходимо:
* создать Dockerfile для каждого из процессов (Календарь, Рассыльщик, Планировщик);
* собрать образы и проверить их локальный запуск;
* создать docker-compose файл, который запускает PostgreSQL, RabbitMQ и все микросервисы вместе (для "неродных" сервисов использовать официальные образы из Docker Hub);
* при желании доработать конфигурацию так, чтобы она поддерживала переменные окружения (если вы используете библиотеку, то скорее всего она уже это умеет); в противном случае придется "подкладывать" конфиг сервису с помощью Dockerfile / docker-compose -

при этом можно "заполнять" конфигурационный файл из переменных окружения, например
```bash
$ envsubst < config_template.json > config.json
```
* если миграции выполняются руками, а не на старте сервиса, то также в docker-compose должен запускаться one-shot скрипт, который делает это (применяет SQL миграции, создавая структуру БД).
* порты серверов, предоставляющих API, пробросить на host.

У преподавателя должна быть возможность запустить весь проект с помощью команды `make up` (внутри `docker-compose up`) и погасить с помощью `make down`.

HTTP API, например, после запуска должно быть доступно по URL **http://localhost:8888/**.

2) Интеграционное тестирование
Необходимо:
* создать отдельный пакет для интеграционных тестов.
* реализовать интеграционные тесты на языке Go; при желании можно использовать [godog](https://github.com/cucumber/godog) / [ginkgo](https://github.com/onsi/ginkgo), но обязательным требованием это **не является**.
* создать docker-compose файл, поднимающий все сервисы проекта + контейнер с интеграционными тестами;
* расширить Makefile командой `integration-tests`, `make integration-tests` будет запускать интеграционные тесты;
не стоит смешивать это с `make test`, иначе CI-пайплайн не пройдёт.
* прикрепить в Merge Request вывод команды `make integration-tests`.

Преподаватель может запустить интеграционные тесты с помощью команды `make integration-tests`:
- команда должна поднять окружение (`docker-compose`), прогнать тесты и подчистить окружение за собой;
- в случае успешного выполнения команда должна возвращать 0, иначе 1.

Подробности:

https://github.com/OtusGolang/home_work/blob/master/hw12_13_14_15_calendar/docs/15_README.md

Процесс сдачи домашнего задания:
https://github.com/OtusGolang/home_work/wiki/[Студентам]-Процесс-сдачи-ДЗ

---

5 **Мониторинг**

Цели занятия:

после занятия вы сможете:
мониторить свои приложения.

Краткое содержание:

виды мониторинга;
мониторинг Linux-серверов;
системные метрики: LA, CPU, MEM, IO;
мониторинг web-серверов и баз данных;
«health check» в Docker;
количественный мониторинг, система Prometheus и работа с ней из Go.

6 Проект

1 **Проектная работа. Вводное**

Цели занятия:

выбрать тему проектной работы, обсудить и начать выполнять её;
спланировать работу над проектом;
получить ответы на возможные вопросы о проекте;
ознакомиться с регламентом работы над проектом.

Краткое содержание:

проект и требования к нему;
сроки выполнения и проверка задания;
вопросы.

Домашние задания

1 Проект

Цель: закрепить знания и навыки, полученные в течение курса;
пополнить своё портфолио качественным проектом.

выбрать тему;
подтвердить тему в чате с преподавателем;
предоставить промежуточные наработки на промежуточное ревью;
предоставить финальную версию проекта к дедлайну.

2 **Консультация по проектам**

Цели занятия:

получить ответы на вопросы, возникшие во время выполнения проектной работы;
получить ответы на вопросы по ДЗ и курсу в целом.

Краткое содержание:

возможные вопросы, связанные с проектом;
затруднения, возникшие при выполнении ДЗ;
вопросы по программе.

---

3 **Итоговое занятие**

Цели занятия:

узнать, как получить сертификат об окончании курса, как взаимодействовать после окончания курса с OTUS и преподавателями, какие вакансии и позиции есть для выпускников (опционально - в России и за рубежом) и на какие компании стоит обратить внимание.

Краткое содержание:

организационные вопросы;
рынок вакансий по направлению;
статистика курса и вопросы по курсу.