Включить запись

education

Разработка веб-приложения и его архитектура

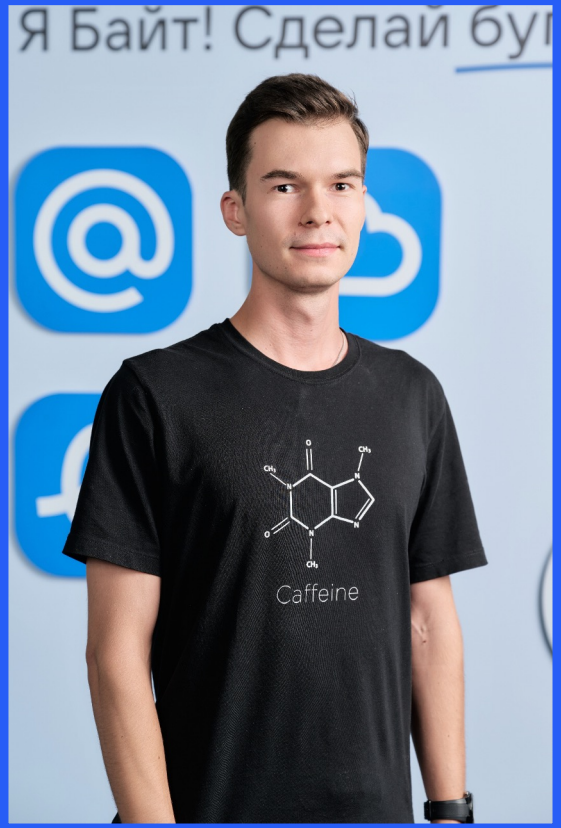
Разработка веб-сервисов на Golang

Григорий Гусев

- Окончил ВМК МГУ им. М.В. Ломоносова

- Успешно прошёл курс
«Разработка веб-сервисов на Golang»

- Backend-разработчик в команде API Почты Mail



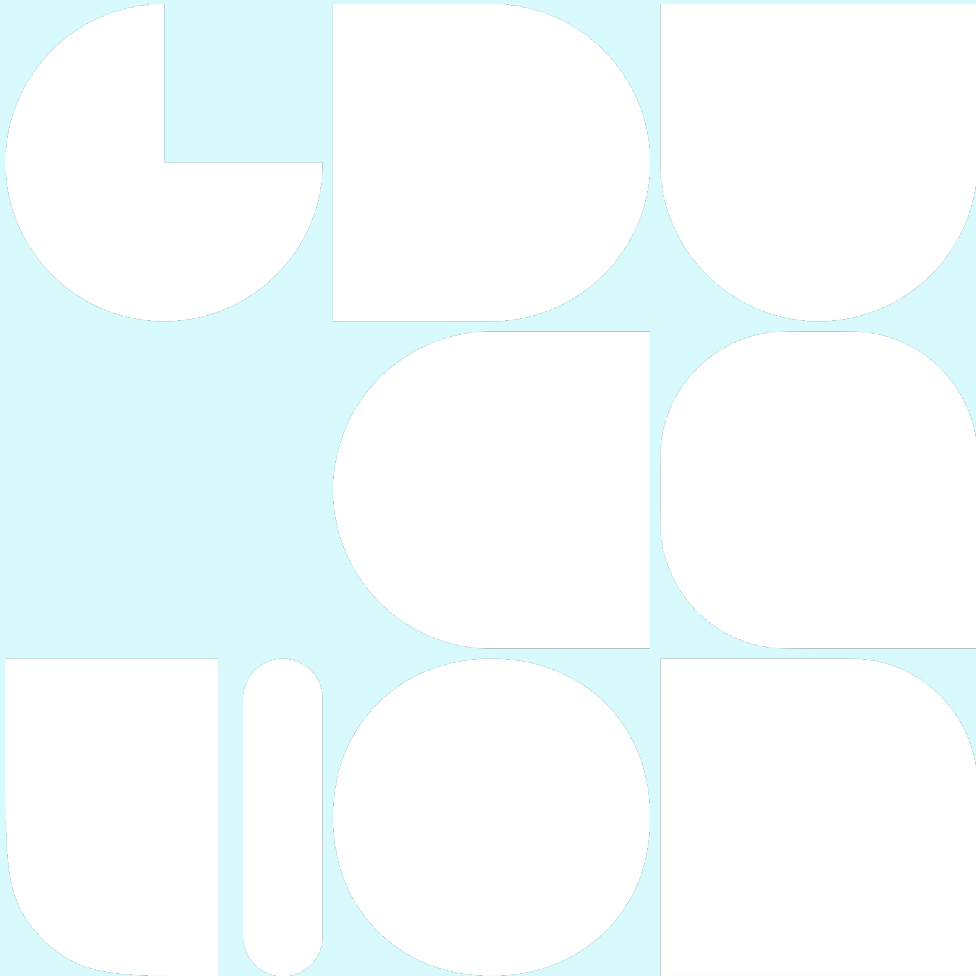
education

В прошлых сериях

- ➔ Intro / Знакомство с Golang
- ➔ Асинхронное программирование
- ➔ Основы работы с сетью, HTTP
- ➔ Проектирование API, авторизация

и хакатон!

О чём поговорим?



О чём поговорим?

- ➔ Middleware для хендлеров (через декораторы)
- ➔ Обработка ошибок
- ➔ Роутеры
- ➔ Валидация
- ➔ Логирование
- ➔ Модули и структура веб-приложения
- ➔ Пример CRUD-приложения
- ➔ Веб-сокеты
- ➔ Домашнее задание

1. Middleware

➔ **Проблема.**

Для нескольких эндпоинтов нужно делать одинаковые действия:

- логировать время запроса
- проверять авторизацию
- ловить панику и делать recover

➔ *Но не хочется дублировать код и разносить логику этих действий по всеми приложению*

➔ **Что делать?**

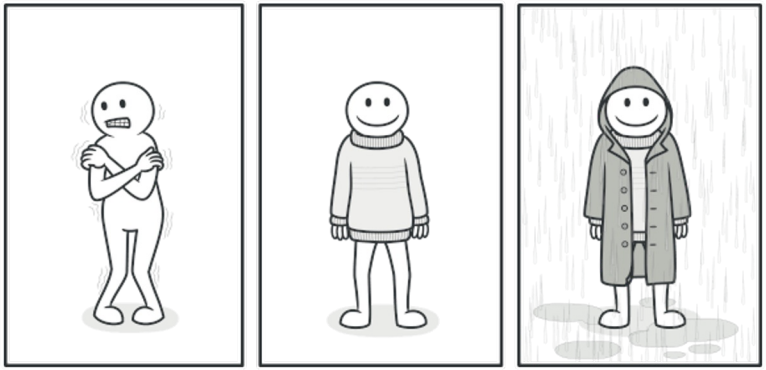
1. Middleware

➔ Решение.
Добавить middleware в хэндлер (роутер) этих эндпоинтов

➔ Middleware — код, «присоединяемый» к произвольному хендлеру (роутеру). Он будет выполняться до и/или после обработки запроса хендлером

1.1 Декораторы

➔ Декоратор — паттерн проектирования, предназначенный для динамического подключения дополнительного поведения к объекту.



```
 func decorator1(input myType) myType {
     // Добавили новую логику к input
и вернули его
}

var myVar myType

// Обернули myVar в первый декоратор
myVar = decorator1(myVar)
// Обернули myVar во второй декоратор
myVar = decorator2(myVar)
...
```

1.2 Middleware для хэндлеров

```
// Создали роутер:
siteMux := http.NewServeMux()

// Добавили в роутер хендлеры для эндпоинтов:
siteMux.HandleFunc("/login", loginPage)
...

// Добавили middlware:
siteHandler := accessLogMiddleware(siteMux)
siteHandler = authMiddleware(siteHandler)
siteHandler = panicMiddleware(siteHandler)
```

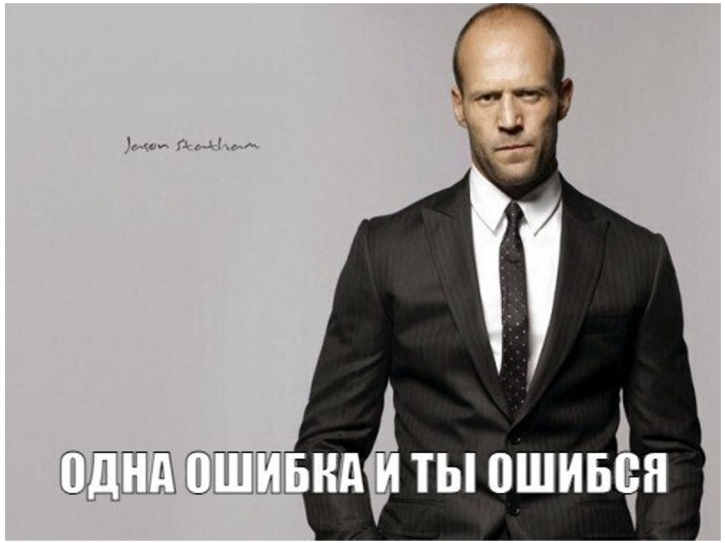
1.3 Ещё примеры middleware

- Проверка доступа (ACL, access control list)
  - server side-эндпоинты доступны только внутренним сервисам

- Ограничение количества запросов (anti bruteforce)
  - не более 10 отправок письма для одного юзера

- Отправка статистики
  - эндпоинт, время запроса, user id, код ответа, время обработки запроса

2. Обработка ошибок



➔ При ошибке нужно не паниковать, а обрабатывать её:

- Оборачивать с информацией об источнике ошибки
- Передавать наверх
- Логировать
- Не возвращать клиенту информацию из ошибки
- Для особых ошибок использовать именованные/типизированные ошибки

➔ Ошибка должна рассказывать историю.

2.1 Но иногда можно вызвать панику

➔ Когда что-то пошло не так при инициализации/старте приложения

```
// Паникует при ошибке:
tmpl :=
template.Must(template.New("gentimings").Parse(templateStr))

// Паникует при ошибке:
re := regexp.MustCompile(`abc`)

func MustInit() {
   if err := Init(); err != nil {
            panic(err)
   }
}
```

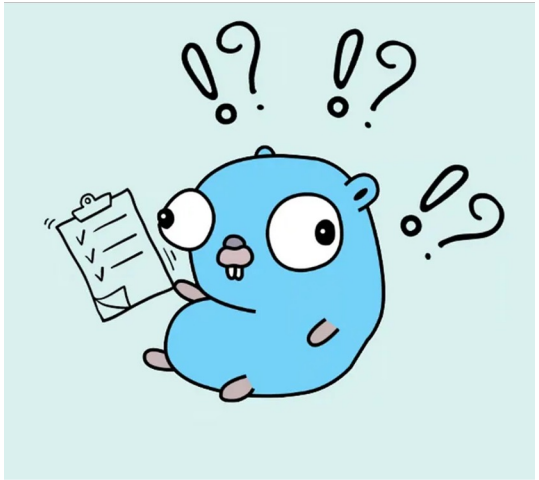
3. Роутеры

- Их много
- Отличаются функциональностью
  - Маршрутизация на основе методов (GET, POST, ...)
  - Переменные в URL-путях
  - Шаблоны маршрутов на основе регулярных выражений
  - Совместимость с http.Handler

- github.com/gorilla/mux
- github.com/julienschmidt/httprouter
- github.com/valyala/fasthttp

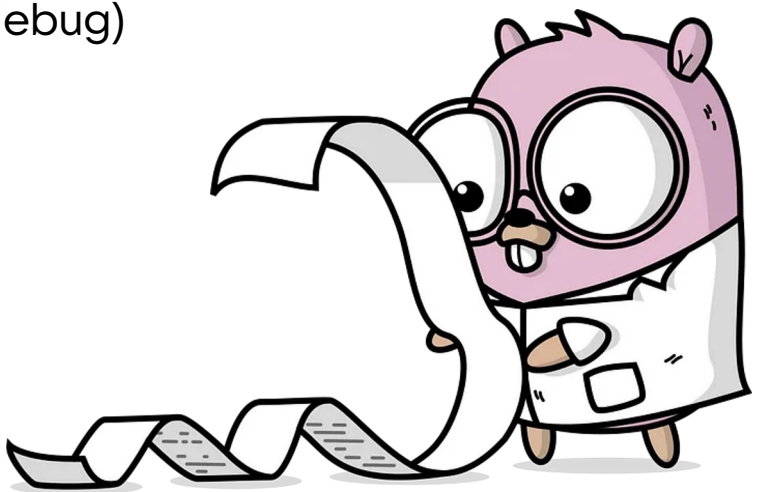
4. Валидация

- ➔ Данные в запросе клиента нужно проверять
- ➔ Лучше это делать унифицированным образом
- ➔ Данные в запросе могут быть кастомного типа
  - email
  - message id



5. Логирование

- ➔ Логи нужны
- ➔ Логи должны быть информативны
- ➔ Очень много логов — проблема
- ➔ В логах нельзя светить чувствительные данные (куки и токены)
- ➔ Как найти логи конкретного запроса? – Выводить в логе request id
- ➔ У логов могут быть уровни (error, warn, info, debug)
- ➔ Логи могут быть структурированными (json)

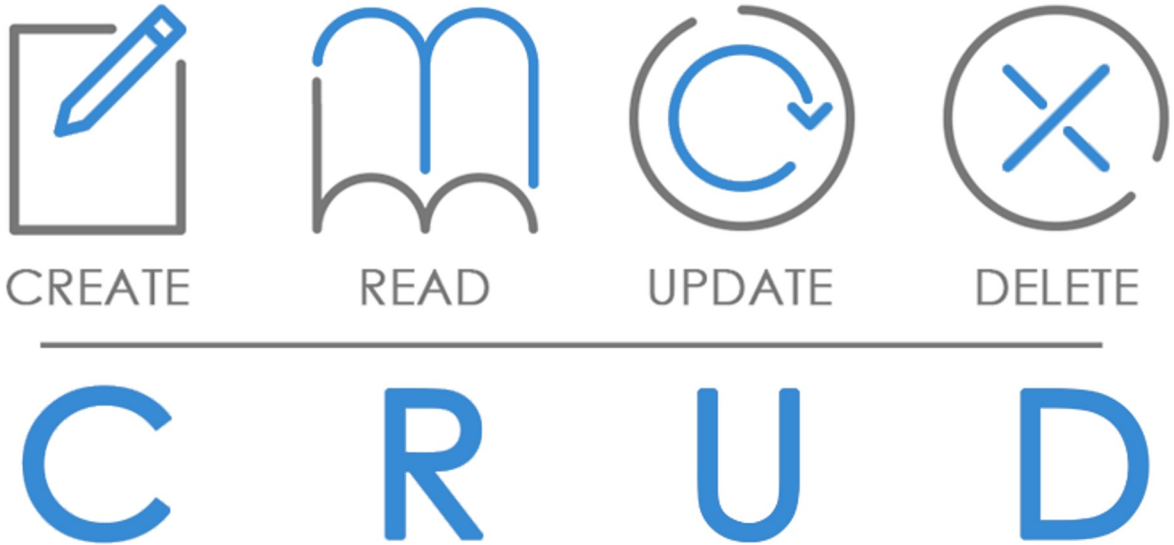


6. Модули и структура веб-приложения

➔ Модули — то, как Golang работает с зависимостями
- *go.mod* – файл с описанием module path и зависимостей модуля
- У зависимостей есть версия модуля: v0.1.2 – major, minor, patch
- Module path – путь, определяющий модуль и задающий префикс при импорте, например "golang.org/x/net"

➔ Структура приложения
- https://github.com/golang-standards/projectlayout/blob/master/README_ru.md

7. Пример CRUD-приложения



8. Веб-сокеты

➔ Проблема.

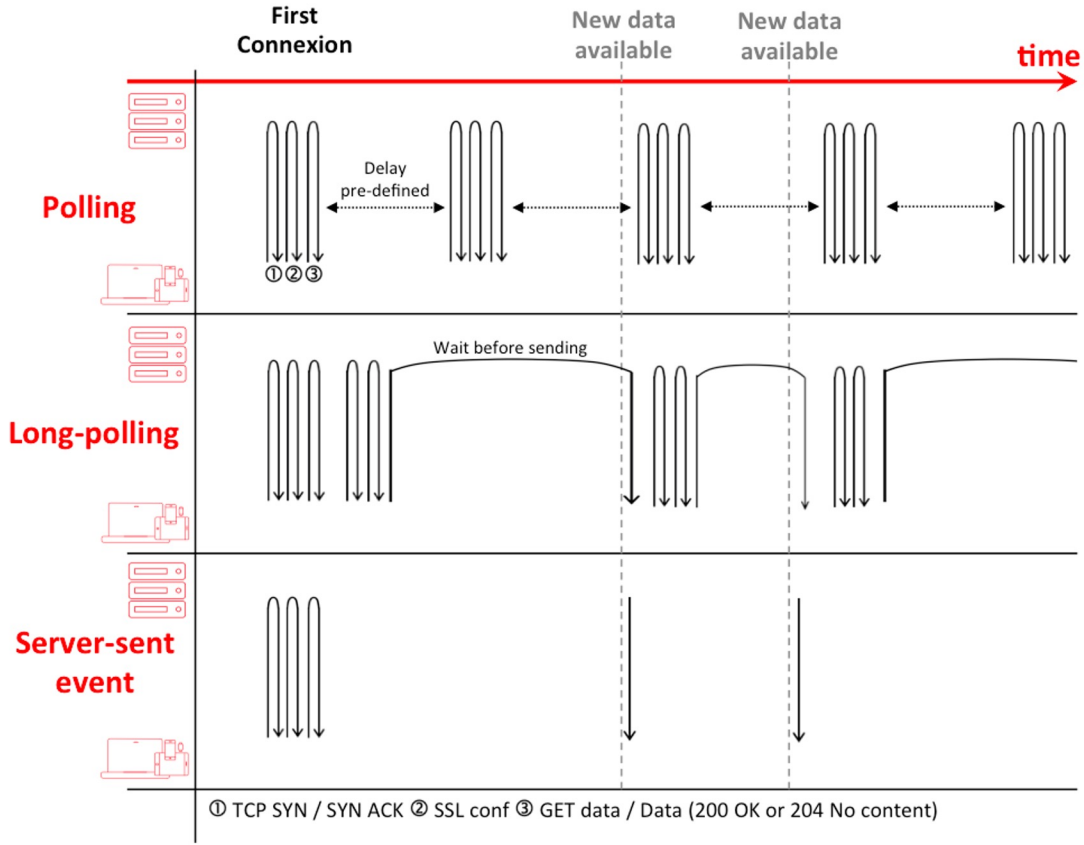
- В HTTP клиент не получит новых данных, пока не сделает запрос к серверу
- Как клиенту всегда иметь актуальные данные?

8. Веб-сокеты

➔ **Проблема.**
- В HTTP клиент не получит новых данных, пока не сделает запрос к серверу
- Как клиенту всегда иметь актуальные данные?

➔ **3 метода:**
- Polling
- Long-polling
- Server-sent event

8.1 Методы поддержки актуальности данных на клиенте



8.1 Методы поддержки актуальности данных на клиенте

- Polling
  - есть накладные расходы
  - есть задержка

- Long-polling
  - накладные расходы ниже, но всё равно есть
  - задержка ниже, но всё равно есть

- Server-sent event
  - накладных расходов (почти) нет
  - задержки нет
  - не поддерживается в HTTP

8.1 Методы поддержки актуальности данных на клиенте

➔ Polling
- есть накладные расходы
- есть задержка

➔ Long-polling
- накладные расходы ниже, но всё равно есть
- задержка ниже, но всё равно есть

➔ Server-sent event
- накладных расходов (почти) нет
- задержки нет
- не поддерживается в HTTP — **используем WebSocket!**

8.2 Протокол WebSocket

➔ WebSocket.
- Протокол поверх TCP-соединения
- Использует постоянное соединение
- Реализует двунаправленный канал связи
- Для установки соединения клиент формирует особый HTTP-запрос, на который сервер отвечает определенным образом

➔ github.com/gorilla/websocket

Итоги и анонсы

➔ О чём поговорили:
- Middleware
- Обработке ошибок
- Роутеры
- Валидация
- Логирование
- Модули и структура веб-приложения
- Пример CRUD-приложения
- Веб-сокеты

➔ В следующей серии: базы данных и работа с ними в Golang
- MySQL
- ORM, gorm
- MongoDB
- Memcache
- Redis
- и не только...

P.S. оставьте фидбек о лекции ^_^

education

Вопросы?