GO-09 04: Роутер Chi

Описание:

Сейчас речь пойдет о другом не менее популярном представителе роутеров на go - Chi. Chi, как он позиционируется, - это легкий, идиоматичный и составной маршрутизатор для создания HTTP сервисов на Go.
Особенности:

- Легковесный (всего +/- 1000 строк кода на весь проект).
- Имеет хорошие показатели в скорости и потреблении памяти - см. benchmarks.
- Модульный - сам по себе имеет минимальный функционал, но зато посредством различных подключаемых модулей может наращивать новый функционал. Есть как стандартные модули, так и возможность писать свои.
- Не имеет внешних зависимостей (только go stdlib + net/http).
- Полностью совместим с net/http.
- Умеет работать с context.

Установка

```
go get -u github.com/go-chi/chi
```

Простейший пример

```
package main

import (
	"net/http"

	"github.com/go-chi/chi"
	"github.com/go-chi/chi/middleware"
)

func main() {
	r := chi.NewRouter()
	r.Get("/", func(w http.ResponseWriter, r *http.Request) {
		w.Write([]byte("welcome"))
    })

	http.ListenAndServe(":3000", r)
}
```

Тут все достаточно легко и очень похоже на стандартный роутер net/http.

Группировка маршрутов
Организация группировки запросов выглядит следующим образом:

```
r.Route("/articles", func(r chi.Router) {
    r.Post("/", createArticle)
    r.Get("/search", searchArticles)
})
```

Также никто не запрещает внутри группы сделать еще одну группу маршрутов, например, по идентификатору:

```
r.Route("/articles", func(r chi.Router) {
    r.Post("/", createArticle)
    r.Get("/search", searchArticles)

    r.Route("/{id}", func(r chi.Router) {
        r.Get("/", getOneArticle)
        r.Delete("/", deleteArticle)
    })
})
```

Работа с query params
Также как и gorilla/mux, chi поддерживает парсинг query параметров и валидацию через регулярные выражения, здесь описания путей с переменными и шаблонами мало чем отличаются от gorilla/mux.

```
r.Get("/user/{mounts:[A-z]+}-{day:[0-9]+}-{year:[0-9]+}", func(w
http.ResponseWriter, r *http.Request) {
    mounth := chi.URLParam(r, "mounts")
    day := chi.URLParam(r, "day")
    year := chi.URLParam(r, "year")

    fmt.Println(mounth, day, year)
})
```

Метод chi.URLParam возвращает строку и не проверяет наличие параметра, но этим занимается парсер пути и, если параметр не будет передан или будет передан неверно (не в соответствии с регулярным выражением), тогда сервер будет отдавать статус 400 BadRequest.
Middlewares
Middleware - это функция, которая выполняется перед основным обработчиком запроса (авторизация, логирование запроса и т.п.) и не относится напрямую к логике самого обработчика.
Более подробно о механизме Middlewares мы поговорим в следующем задании!

Мы легко можем наращивать функционал chi при помощи стандартных или самописных Middleware. Например:

```
package main

import (
      "github.com/go-chi/chi"
      "github.com/go-chi/chi/middleware"
      "net/http"
)

func main() {
    r := chi.NewRouter()
    r.Use(middleware.Logger)
      r.Get("/", func(w http.ResponseWriter, r *http.Request) {
            w.Write([]byte("welcome"))
    })

    http.ListenAndServe(":3000", r)
}
```

При помощи метода r.Use мы можем передать в chi нужный нам middleware. В нашем примере мы передали в chi стандартный модуль middleware.Logger, и теперь каждый наш запрос будет логироваться дополнительной информацией.
Пример вывода:

```
2020/07/21 23:12:17 "GET http://localhost:3000/ HTTP/1.1" from
[::1]:52165 - 000 0B in 30.54µs
```

Полезные ссылки:

- Handling HTTP requests with go-chi
- More examples for use chi
- Github repo chi

Задание:

В этом задании мы предлагаем вам использовать роутер chi вместо gorilla/mux.
Условия:

- Функционал не должен пострадать, все должно работать, как и работало.

- Должен быть использован один из стандартных модулей, например, Logger или RequestID список стандартных chi модулей.

Порядок действий:

1. В вашем проекте module09 сделайте новую ветку 04_task.
2. В пакете module09/internal/routers/chi создайте роутер, зарегистрируйте маршруты и модули.
3. Отредактируйте логику обработчиков в пакете module09/internal/handlers/user (если это нужно).
4. В пакете module09/cmd/app подключите наш новый роутер и запустите сервер.
5. Проверьте работоспособность.
6. В ответе укажите:
    - ссылку на merge request в ветку master своего проекта ветки 04_task;