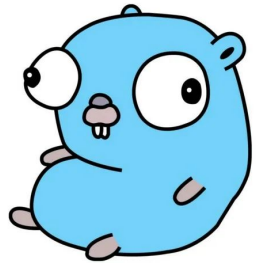
API & GO

API

Application Programming Interface - способ коммуникации между программами.

Контракт, который предоставляет программа.



API описывает формат

API описывает формат как одна программа должна обращаться к другой

“Ко мне можно обращаться так и так, я принимаю это на вход и выдаю это”

API состоит из:

- Метод
- Входные параметры
- Выходные параметры

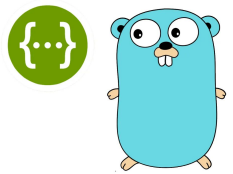
API описывает формат

API – “набор функций”

REST API - вариант реализации API

Основные компоненты

- URL - адрес API метода
- HTTP-метод - чаще всего GET для чтения и/или POST для записи
- Headers - заголовок, предоставляют информацию об авторизации и пр.
- Body - Входные параметры в формате JSON



Примеры REST API - чтение

```
GET /order/1
```

Response:

```
{
  "name": "Vegetables",
  "price": "1000$",
  "count": 4
}
```

Примеры REST API - чтение

```
GET /orders/list

Response:
{
  orders: [
    {
      "id": 1,
      "name": "Vegetables",
      "price": 1000,
      "count": 4,
    },
    {
      "id": 2,
      "name": "Clothes",
      "price": 400
      ...
  ]
}
```

Примеры REST API - создание

```
POST /order

Request:
{
  "name": "Car",
  "price": 100,
  "count": 1
}
```

Обработка в GO

- Создать обработчик URL метода API
- Пропарсить тело запроса
- Выполнить полезную нагрузку - сохранить в базу новый пост

Обработчик запроса

```
// Создать обработчик урла как обычно.
// Через Methods указать, что данный метод работает только через POST

mux.HandleFunc("/order", createOrder(dbx)).Methods(http.MethodPost)
```

Пропарсить тело запроса - определить структуру

```
// Описываем структуру запроса
type createOrderRequest struct {
   Title   string `json:"title"`
   Content string `json:"content"`
}
```

Аннотация `json` напротив поля структуры указывает какое поле из JSON при парсинге будет записано в поле.

Работает по аналогии с аннотацией db при чтении из базы данных.

Пропарсить тело запроса

```
import (
   ...
   "encoding/json" // Импортируем библиотеку для работы с JSON
   ...
)
```

Пропарсить тело запроса

```
// … тело ф-ии createOrder

reqData, err := io.ReadAll(r.Body) // Прочитали тело запроса с reqData в виде массива байт
if err != nil {
    // ... обработка ошибки
}

var req createOrderRequest  // Заранее объявили переменную  createOrderRequest

err =  json.Unmarshal(reqData, &req) // Отдали reqData и req на парсинг библиотеке json
if err != nil {
    // ... обработка ошибки
}
// ...
```

Выполнить полезную нагрузку - сохранить в базу. Возможно что-то ещё...

```
func saveOrder(db *sqlx.DB, req createOrderRequest) error {
   const query = `
       INSERT INTO
           order
       (
           title,
           content
       )
       VALUES
       (
           ?, // Вместо конкретных значений - ? плейсхолдеры для значений
           ?
       )
   `

   _, err := db.Exec(query, req.Title, req.Content) // Сами данные передаются через аргументы к ф-ии Exec
   return err
}
```

Выводы

- Узнали про API - почитать про API
- Узнали про REST API
- Пощупали JSON парсинг со стороны бекенда

Финальный мем