ctx := context.Background()

Эта функция возвращает пустой контекст. Она должна использоваться только на высоком уровне (в main или обработчике запросов высшего уровня). Он может быть использован для получения других контекстов

ctx := context.TODO()

Эта функция также создает пустой контекст. И она тоже должна использоваться только на высоком уровне или когда вы не уверены, какой контекст использовать, или если в функции еще нет получения нужного контекста. Это значит, что вы (или тот, кто поддерживает код) планируете позже добавить контекст в функцию.

context.WithValue(parent Context, key, val interface{}) Context

Эта функция принимает контекст и возвращает производный от него контекст, в котором значение val связано с key и проходит через всё контекстное дерево. То есть, как только вы создадите контекст WithValue, любой производный контекст получит это значение

ctx, cancel := context.WithCancel(context.Background())

сtx, cancel := context.WithDeadline(context.Background(), time.Now().Add(2 \* time.Second))

**Отменять контекст можно разными способами**

**явно - context.WithCancel**

**по таймауту - context.WithTimeout**

**при наступлении какого-либо дедлайна context.WithDeadline**

1. **прямая отмена**

**ctx, cancel := context.WithCancel(context.Background())**

**defer cancel()**

1. **отмена по тайм ауту**

**ctx, cancel := context.WithTimeout(context.Background(), time.Second\*5)**

**defer cancel()**

1. **с передачей значений (желательно не использовать!)**

**ctx = context.WithValue(ctx, key, value)**

1. **по времени но можно мызвать cancel() досрочно**

**ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(time.Second\*5))**

**примеры использования времени WithDeadline**

**ow := time.Now()**

**deadline := now.Add(5 \* time.Minute) // через 5 минут от текущего момента**

**ctx, cancel := context.WithDeadline(context.Background(), deadline)**

**defer cancel()**

**deadlineStr := "2025-07-15T14:30:00Z"**

**deadline, \_ := time.Parse(time.RFC3339, deadlineStr)**

**ctx, cancel := context.WithDeadline(context.Background(), deadline)**

**defer cancel()**

**nixTime := time.Unix(1700000000, 0) // Конкретная временная точка**

**ctx, cancel := context.WithDeadline(context.Background(), unixTime)**

**defer cancel()**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

fmt.Println(time.Now()) // 2024-04-11 17:39:17.756388243 +0300 MSK m=+0.000053694

time.Now() – текущее время

fmt.Println(time.Now().UTC()) // 2024-04-11 14:52:40.385906179 +0000 UTC

fmt.Println(time.Now().Unix()) // 1712847160 (возвращает количество секунд, прошедших с начала эпохи Unix с 1 января 1970 года, 00:00:00 UTC)

HTTP

**Стартовая строка**:

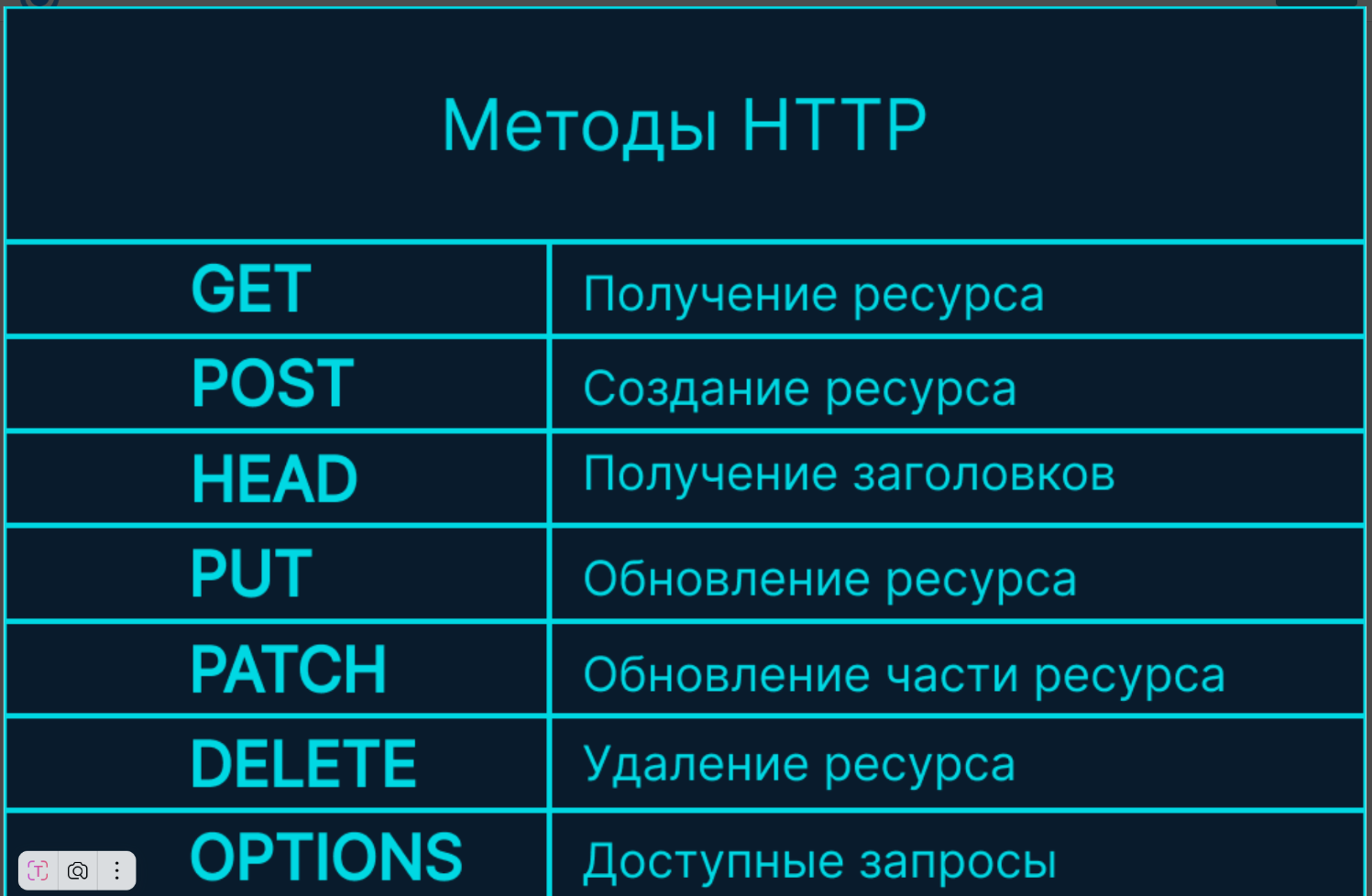
* Метод (GET, POST, PUT, DELETE и т.д.).
* URL (путь к ресурсу).
* Версия протокола (например, HTTP/1.1).

**2. Заголовки**:

* Дополнительная информация о запросе (например, тип контента, авторизация, язык клиента).

3. Тело:

* Данные, которые отправляются в запросе (например, данные формы или JSON).



func homeHandler(w http.ResponseWriter, r \*http.Request) {

    fmt.Fprintf(w, "Welcome to the home page!")

}

func aboutHandler(w http.ResponseWriter, r \*http.Request) {

    fmt.Fprintf(w, "This is the about page.")

}

// создаем сервер на порту 8080 который при любом запросе будет отdечать Hello World

func main() {

    // Регистрируем обработчик для всех запросов

    http.HandleFunc("/", homeHandler)

    http.HandleFunc("/about", aboutHandler)

    // Запускаем сервер на порту 8080

    fmt.Println("Starting server at port 8080")

    err := http.ListenAndServe(":8080", nil)

    if err != nil {

        fmt.Println("Error starting the server:", err)

    }

}

http.HandleFunc() – отслеживание по URL адресу

“/” – отслеживается переход на главную стреницу

“/abaut”