Московский государственный технический университет им. Н.Э. Баумана

Факультет Радиотехнический
Кафедра РТ5

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашнему заданию «Разработка API To-Do List сервиса на языке Go»

Выполнил:

студент группы РТ5-31Б:
Пичурин В. Е.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023

Описание задания

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).

2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.

3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.

4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).

5. В случае создания проекта необходимо детально комментировать код.

6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.

7. Приветствуется написание черновика статьи по результатам выполнения ДЗ.

Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Основная часть

Go, также известный как Golang, представляет собой язык программирования, созданный инженерами из Google. Этот язык обладает простым и читаемым синтаксисом, который напоминает C, но предоставляет более высокоуровневые возможности. Одной из ключевых черт Go является эффективная компиляция в машинный код, что обеспечивает высокую производительность.

В контексте веб-разработки Go становится все более популярным выбором благодаря своей простоте, эффективности и поддержке параллелизма. Многие разработчики отмечают его высокую производительность при написании веб-серверов и микросервисов.

Для демонстрации возможностей языка в веб-разработке я решил создать небольшой Web-Api сервис, представляющий собой backend часть To-Do List приложения.

Приложение использует веб-фреймворк Gin и ORM-библиотеку GORM для создания простого веб-сервера с базой данных SQLite. Оно предоставляет API для управления задачами (tasks), включая создание, чтение, обновление и удаление задач.

Основные возможности приложения:

- Инициализация базы данных: При запуске приложения происходит инициализация базы данных SQLite и автоматическое создание таблицы для хранения задач.
- Создание задачи: При отправке HTTP POST-запроса на /tasks, можно создать новую задачу, передав JSON-объект с полями title и description.
- Получение списка задач: При отправке HTTP GET-запроса на /tasks, возвращается список всех задач.
- Получение конкретной задачи: При отправке HTTP GET-запроса на /tasks/:id, где :id - идентификатор задачи, возвращается информация о конкретной задаче.
- Обновление задачи: При отправке HTTP PUT-запроса на /tasks/:id, можно обновить существующую задачу, передав JSON-объект с обновленными полями.
- Удаление задачи: При отправке HTTP DELETE-запроса на /tasks/:id, можно удалить существующую задачу по идентификатору.

Код программы

```
package main

import (
    "fmt"
    "github.com/gin-gonic/gin"
    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
    "net/http"
)

type Task struct {
    gorm.Model
    Title       string `json:"title"`
    Description string `json:"description"`
}

var db *gorm.DB

func main() {
    initDB()

    router := gin.Default()

    router.POST("/tasks", createTask)

    router.GET("/tasks", getTasks)

    router.GET("/tasks/:id", getTask)

    router.PUT("/tasks/:id", updateTask)

    router.DELETE("/tasks/:id", deleteTask)
```

```
    router.Run(":8000")
}

func initDB() {
    var err error
    db, err = gorm.Open(sqlite.Open("tasks.db"), &gorm.Config{})
    if err != nil {
        panic("Не удалось подключиться к базе данных")
    }

    db.AutoMigrate(&Task{})
}

func createTask(c *gin.Context) {
    var task Task
    if err := c.ShouldBindJSON(&task); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    db.Create(&task)
    c.JSON(http.StatusOK, task)
}

func getTasks(c *gin.Context) {
    var tasks []Task
    db.Find(&tasks)
    c.JSON(http.StatusOK, tasks)
}

func getTask(c *gin.Context) {
    id := c.Params.ByName("id")
    var task Task
    if err := db.Where("id = ?", id).First(&task).Error; err != nil {
        c.AbortWithStatus(http.StatusNotFound)
        return
    }
    c.JSON(http.StatusOK, task)
}

func updateTask(c *gin.Context) {
    id := c.Params.ByName("id")
    var task Task
    if err := db.Where("id = ?", id).First(&task).Error; err != nil {
        c.AbortWithStatus(http.StatusNotFound)
        return
    }

    var updatedTask Task
    if err := c.ShouldBindJSON(&updatedTask); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    db.Model(&task).Updates(updatedTask)
    c.JSON(http.StatusOK, task)
}
```

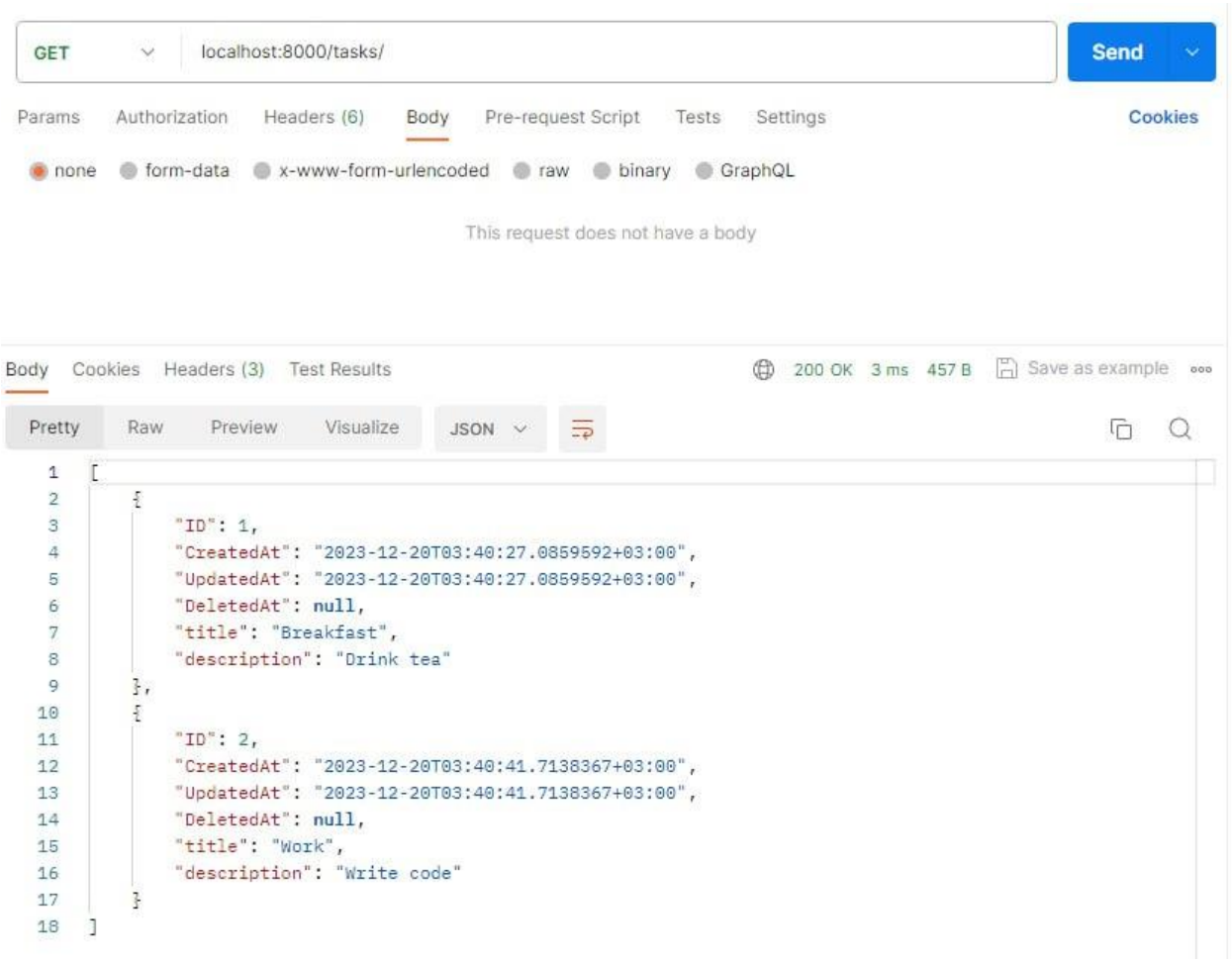
```
func deleteTask(c *gin.Context) {
    id := c.Params.ByName("id")
    var task Task
    if err := db.Where("id = ?", id).First(&task).Error; err != nil {
        c.AbortWithStatus(http.StatusNotFound)
        return
    }

    db.Delete(&task)
    c.JSON(http.StatusOK, gin.H{"message": fmt.Sprintf("Задача с ID %s успешно удалена", id)})
}
```

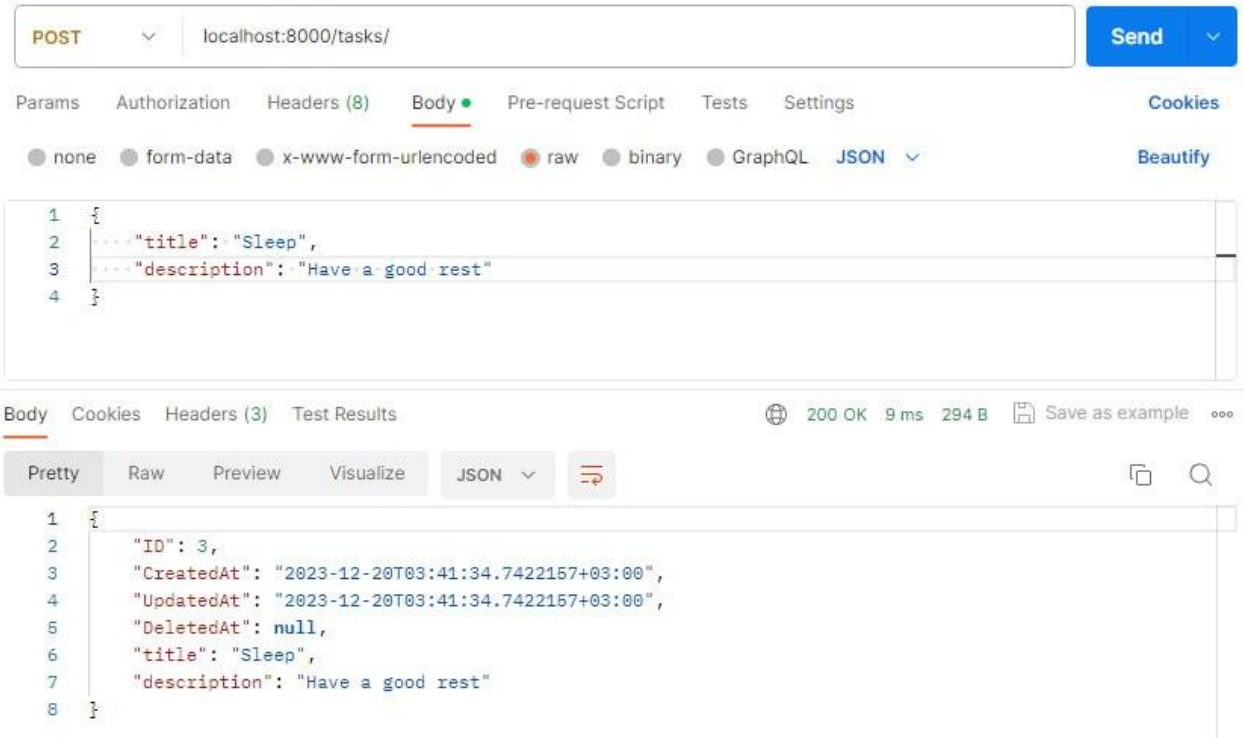
Результат выполнения

Далее представлены результаты запросов к API из приложения Postman, демонстрирующие весь функционал веб-сервиса.

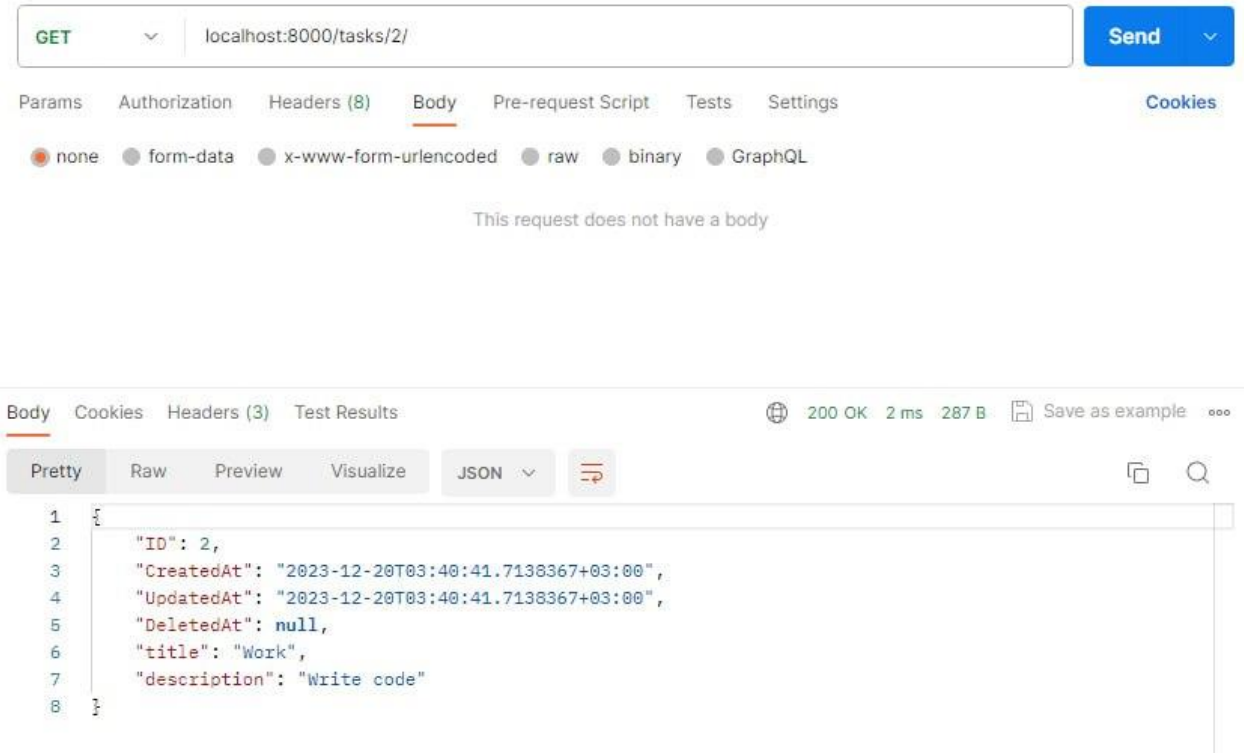
- List



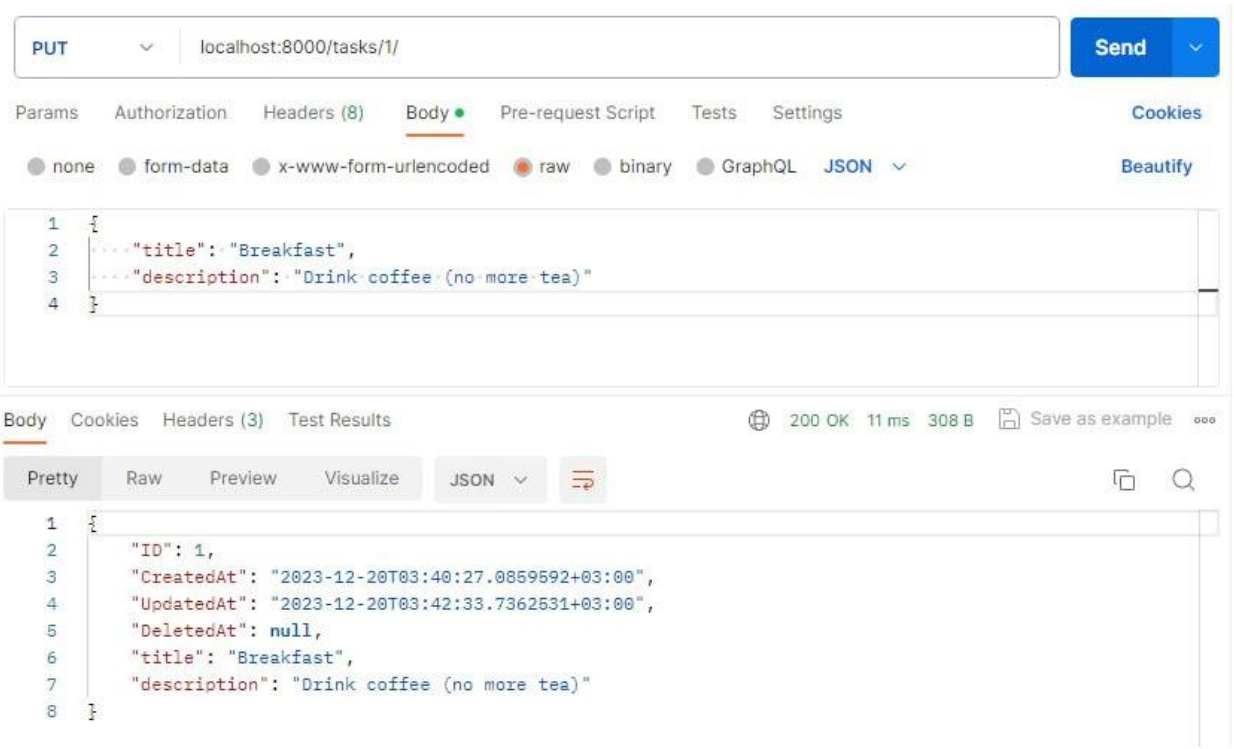
- Create



- Retrieve



- Update



- Delete