

**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, транслятор) 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

The screenshot shows the Postman interface. At the top, a GET request is made to `localhost:8000/tasks/`. The 'Body' tab is selected, showing 'none' as the content type. Below this, the response is displayed in the 'Body' tab, showing a JSON array of two task objects. The status is 200 OK, with a response time of 3 ms and a body size of 457 B. The response is formatted as JSON.

```
[
  {
    "ID": 1,
    "CreatedAt": "2023-12-20T03:40:27.0859592+03:00",
    "UpdatedAt": "2023-12-20T03:40:27.0859592+03:00",
    "DeletedAt": null,
    "title": "Breakfast",
    "description": "Drink tea"
  },
  {
    "ID": 2,
    "CreatedAt": "2023-12-20T03:40:41.7138367+03:00",
    "UpdatedAt": "2023-12-20T03:40:41.7138367+03:00",
    "DeletedAt": null,
    "title": "Work",
    "description": "Write code"
  }
]
```

- Create

POST localhost:8000/tasks/ Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   "title": "Sleep",
3   "description": "Have a good rest"
4 }
```

Body Cookies Headers (3) Test Results 200 OK 9 ms 294 B Save as example

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "ID": 3,
3   "CreatedAt": "2023-12-20T03:41:34.7422157+03:00",
4   "UpdatedAt": "2023-12-20T03:41:34.7422157+03:00",
5   "DeletedAt": null,
6   "title": "Sleep",
7   "description": "Have a good rest"
8 }
```

- Retrieve

GET localhost:8000/tasks/2/ Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (3) Test Results 200 OK 2 ms 287 B Save as example

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "ID": 2,
3   "CreatedAt": "2023-12-20T03:40:41.7138367+03:00",
4   "UpdatedAt": "2023-12-20T03:40:41.7138367+03:00",
5   "DeletedAt": null,
6   "title": "Work",
7   "description": "Write code"
8 }
```

- Update

PUTlocalhost:8000/tasks/1/

Send

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1 {
2   "title": "Breakfast",
3   "description": "Drink coffee (no more tea)"
4 }
```

BodyCookiesHeaders (3)Test Results200 OK11 ms308 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "ID": 1,
3   "CreatedAt": "2023-12-20T03:40:27.0859592+03:00",
4   "UpdatedAt": "2023-12-20T03:42:33.7362531+03:00",
5   "DeletedAt": null,
6   "title": "Breakfast",
7   "description": "Drink coffee (no more tea)"
8 }
```

- Delete

DELETElocalhost:8000/tasks/2/

Send

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQL

This request does not have a body

BodyCookiesHeaders (3)Test Results200 OK10 ms187 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "message": "Задача с ID 2 успешно удалена"
3 }
```