|  |  |
| --- | --- |
| ДИСЦИПЛИНА | Технологии индустриального программирования |
| ИНСТИТУТ | Институт перспективных технологий и индустриального программирования |
| КАФЕДРА | Кафедра индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Практическое занятие |
| ПРЕПОДАВАТЕЛЬ | Адышкин Сергей Сергеевич |
| СЕМЕСТР | 1 семестр, 2025-2026 гг. |

**Практическое занятие №1**

Установка и настройка окружения Go.

**Цель**

Развернуть рабочее окружение Go на Windows, создать минимальный HTTP-сервис на net/http, подключить и использовать внешнюю зависимость, собрать и проверить приложение.

**Задание**

* Установить Go и Git, проверить версии.
* Инициализировать модуль Go в новом проекте.
* Реализовать HTTP-сервер с маршрутами /hello (текст) и /user (JSON).
* Подключить внешнюю библиотеку (генерация UUID) и использовать её в /user.
* Запустить и проверить ответы curl/браузером.
* Собрать бинарник .exe и подготовить README и отчёт.

**Выполнение работы**

**1. Подготовка окружения и проверка версий**

Откройте **PowerShell**.

go version

git --version

**Ожидаемо:** команды выводят версии (например, go version go1.22.x и git version ...).

Если go не найден — установите Go и перезапустите PowerShell.

**Совет (кириллица в консоли):**

chcp 65001 | Out-Null # UTF-8 в текущем сеансе

**2. Структура проекта и модуль Go**

Выберите рабочую папку (пример: D:\Edu\Go). Создайте проект **helloapi** и инициализируйте модуль.

mkdir helloapi

cd helloapi

go mod init example.com/helloapi

Появится go.mod — «паспорт» проекта с именем модуля и версией Go.

**Рекомендуемая структура на это ПЗ:**

helloapi/

cmd/

server/

main.go

go.mod

go.sum (появится позже)

**macOS/Linux:** команды те же. Для создания вложенных папок можно mkdir -p cmd/server.

**3. Минимальный HTTP-сервер (первая версия)**

Создайте файл cmd/server/main.go и вставьте код:

package main

import (

"encoding/json"

"fmt"

"log"

"net/http"

)

type user struct {

ID string `json:"id"`

Name string `json:"name"`

}

func main() {

mux := http.NewServeMux()

// Текстовый ответ

mux.HandleFunc("/hello", func(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintln(w, "Hello, world!")

})

// Пока временный JSON (без UUID — добавим на шаге 4)

mux.HandleFunc("/user", func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

\_ = json.NewEncoder(w).Encode(user{

ID: "temp",

Name: "Gopher",

})

})

addr := ":8080"

log.Printf("Starting on %s ...", addr)

log.Fatal(http.ListenAndServe(addr, mux))

}

**Пояснения:**

http.NewServeMux() — простой роутер из стандартной библиотеки.

Для JSON всегда выставляем заголовок Content-Type: application/json.

**4. Подключение внешней зависимости и доработка /user**

Подтянем пакет для генерации UUID и заменим «temp» на реальный идентификатор.

go get github.com/google/uuid@latest

go mod tidy

Измените cmd/server/main.go:

import "github.com/google/uuid"

// ...

mux.HandleFunc("/user", func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

\_ = json.NewEncoder(w).Encode(user{

ID: uuid.NewString(), // теперь реальный UUID

Name: "Gopher",

})

})

**Проверка:**  
В корне проекта появился файл go.sum — фиксирует точные версии зависимостей.

**5. Запуск сервера и быстрая проверка**

Запустите сервис **из корня проекта**:

go run ./cmd/server

**Ожидаемо в логе:** Starting on :8080 ...

В другом окне PowerShell проверьте эндпоинты:

curl http://localhost:8080/hello

curl http://localhost:8080/user

**Ожидаемые ответы:**

/hello → текст Hello, world!

/user → JSON вида:

{

"id": "3f1c1c5c-9a1b-4a6a-8a26-7b3f2f7d8b0b",

"name": "Gopher"

}

**Если видите кракозябры** — выполните chcp 65001 в окне, где делаете curl.

**6. Сборка бинарника и повторная проверка**

Остановите запущенный сервер (Ctrl+C) и соберите .exe:

go build -o helloapi.exe ./cmd/server

.\helloapi.exe

Повторите проверку curl — ответы должны совпасть с предыдущим шагом.

**7. Код-стайл и базовые проверки**

Отформатируйте и проверьте код стандартными инструментами Go:

go fmt ./...

go vet ./...

**Ожидаемо:** команды завершаются без ошибок.  
Если vet ругается — исправьте замечания и запустите снова.

**8. (Опционально, «\*звёздочка») Конфигурация через переменные окружения**

Сделайте порт настраиваемым через переменную APP\_PORT.

1. Обновите код (версия main.go с чтением переменной):

package main

import (

"encoding/json"

"log"

"net/http"

"os"

"github.com/google/uuid"

)

type user struct {

ID string `json:"id"`

Name string `json:"name"`

}

func main() {

port := os.Getenv("APP\_PORT")

if port == "" {

port = "8080"

}

mux := http.NewServeMux()

mux.HandleFunc("/hello", func(w http.ResponseWriter, r \*http.Request) {

w.Write([]byte("Hello, world!"))

})

mux.HandleFunc("/user", func(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

\_ = json.NewEncoder(w).Encode(user{

ID: uuid.NewString(),

Name: "Gopher",

})

})

addr := ":" + port

log.Printf("Starting on %s ...", addr)

log.Fatal(http.ListenAndServe(addr, mux))

}

1. Запуск на другом порту:

$env:APP\_PORT="8081"

go run ./cmd/server

# теперь сервер слушает :8081

Проверьте: curl http://localhost:8081/hello.

**Доп. звёздочка:** добавить /health c JSON { "status":"ok", "time":"<RFC3339>" }.

**9. Типичные проблемы и моментальные решения (во время выполнения)**

* **go: command not found** — переоткройте PowerShell после установки Go, проверьте PATH, переустановите Go-installer.
* **no Go files in ... при go run** — запускайте из корня helloapi командой go run ./cmd/server.
* **Порт уже занят (listen tcp :8080: bind: ...)** — запустите на другом порту ($env:APP\_PORT="8081"), либо завершите процесс, который держит порт.
* **Неверный JSON/заголовки** — проверяйте Content-Type: application/json и используйте json.NewEncoder(w).Encode(...).
* **go: module github.com/google/uuid found (vX.Y.Z), but does not contain package ...** — проверьте правильность import и повторите go get/go mod tidy.

**10. Контрольная точка (перед сдачей)**

* Сервер запускается через go run ./cmd/server и через собранный .exe.
* /hello возвращает 200 и текст.
* /user возвращает 200 и валидный JSON с UUID.
* В проекте присутствуют go.mod и go.sum, в коде реально используется внешняя зависимость.
* go fmt/go vet — без ошибок/варнингов.

**Подготовка отчёта**

1. **Размещение кода в Git**

git init

git add .

git commit -m "PZ1: minimal HTTP server + uuid"

git branch -M main

git remote add origin <URL\_вашего\_репозитория>

git push -u origin main

1. **README.md (минимум)**

* Описание проекта и требования.
* Команды запуска/сборки, примеры запросов.
* Структура проекта (короткое дерево).
* Примечания по конфигурации (порт, переменные окружения).

1. **Отчётные материалы**

* Скриншоты go version и ответов /hello и /user.
* Ссылка на репозиторий.

**Итоговая проверка (чек-лист)**

* Репозиторий клонируется, go build проходит без ошибок.
* go run ./cmd/server запускается, /hello и /user отвечают 200.
* В go.mod и go.sum зафиксированы зависимости; UUID реально генерируется.
* README содержит шаги запуска и примеры запросов.
* Код отформатирован (go fmt), базовая проверка go vet — без критики.

**Критерии оценивания (10 баллов)**

* Установка и инициализация модуля — **2 б.**
* Маршрут /hello (200 + корректный ответ) — **2 б.**
* Маршрут /user (валидный JSON + заголовок Content-Type) — **3 б.**
* Подключение и использование внешней зависимости — **2 б.**
* README и аккуратность репозитория — **1 б.**  
  **Бонус (+1…2 б.)**: /health с временем в RFC3339, чтение порта из переменной окружения, базовый лог запросов.

**Вопросы на защиту**

1. Что такое модуль Go? Роль go.mod и go.sum.
2. Отличия go run и go build. Когда что использовать?
3. Как сервер формирует JSON-ответ и зачем выставлять Content-Type: application/json?
4. Как подключить и зафиксировать стороннюю библиотеку? Роль go get и go mod tidy.
5. Где и как читать переменные окружения для конфигурации порта?

**Частые ошибки и решения**

* **go: command not found** — перезапустите PowerShell/установите Go, проверьте PATH.
* **Порт занят** — смените порт (например, 8081) и/или остановите процесс, занимающий порт.
* **Неправильный JSON/кодировка** — используйте json.NewEncoder(w).Encode(...), выставляйте Content-Type.
* **Зависимость не подтягивается** — проверьте импорт и выполните go get + go mod tidy.
* **Нет README** — приёмка затруднена: добавьте команды запуска и примеры curl (на манер образца).