



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.01 ИУ6-32Б

О Т Ч Е Т

по лабораторной работе № 10

Название: Архитектура микросервисов на Golang

Дисциплина: Языки интернет-программирования

Студент

ИУ6-32Б

Кондратов С.Ю.

(Группа)

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Шульман В.Д.

(Подпись, дата)

(И.О. Фамилия)

Цель работы: получение первичных навыков организации кодовой базы проекта на Golang

Задание 1

Переделать сервисы с чистой архитектурой

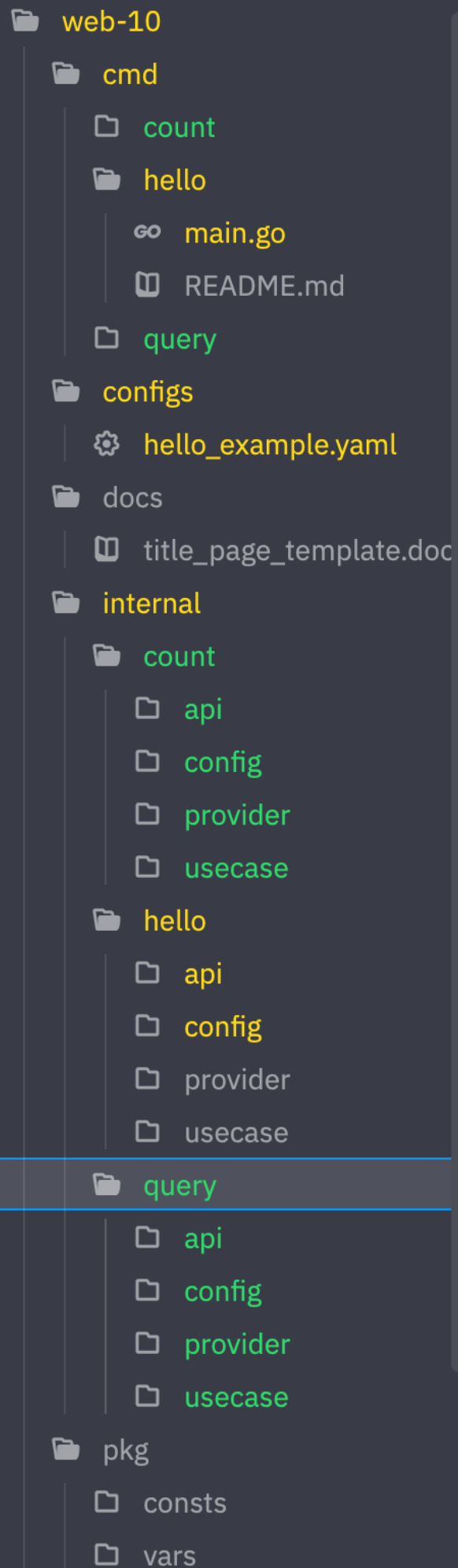


Рисунок 1

Директории Go /cmd

Основные приложения для текущего проекта.

Имя директории для каждого приложения должно совпадать с именем исполняемого файла, который вы хотите собрать (например, /cmd/myapp).

Не стоит располагать в этой директории большие объёмы кода. Если вы предполагаете дальнейшее использование кода в других проектах, вам стоит хранить его в директории /pkg в корне проекта. Если же код не должен быть переиспользован где-то еще - ему самое место в директории /internal в корне проекта. Вы будете удивлены тем, что могут сделать другие люди, по тому выражайте свои намерения явно!

Самой распространённой практикой является использование маленькой main функции, которая импортирует и вызывает весь необходимый код из директорий /internal и /pkg, но не из других.

Примеры смотрите в директории /cmd.

/internal

Внутренний код приложения и библиотек. Это код, который не должен использоваться в других приложениях и библиотеках. Стоит отметить, что этот шаблон применяется самим компилятором Golang. Ознакомьтесь с release notes Go 1.4 для более детальной информации. Также, вы вольны использовать более одной директории internal на разных уровнях структуры своего проекта.

Вы можете добавить дополнительное структурирование, чтобы разделить открытую и закрытую части вашего внутреннего кода. Такой подход не является необходимым, особенно для маленьких проектов, но позволяет сразу визуально оценить применение кода. Код самого приложения может находиться в директории /internal/app (например, /internal/app/myapp), а код, который это приложение использует - в директории /internal/pkg (например, /internal/pkg/myprivlib).

/pkg

Код библиотек, пригодных для использования в сторонних приложениях (например, /pkg/mypubliclib). Другие проекты будут импортировать эти библиотеки, ожидая их автономной работы, поэтому стоит подумать дважды, прежде чем класть сюда какой-нибудь код. Обратите внимание, что использование директории internal - более оптимальный способ гарантировать что ваши внутренние пакеты, не будут

импортированы, потому что это обеспечивает сам Go. Директория `/pkg` - всё еще хороший путь дать понять, что код в этой директории могут безопасно использовать другие. Статья [I'll take pkg over internal](#) в блоге Трэвиса Джеффери (Travis Jeffery) предоставляет хороший обзор директорий `pkg` и `internal` и когда есть смысл их использовать.

Это так же способ группировать код на Go в одном месте, когда ваша корневая директория содержит множество не относящихся к Go компонентов и директорий, что позволит облегчить работу с разными инструментами Go (как упомянуто в этих выступлениях: [Best Practices for Industrial Programming](#) с GopherCon EU 2018, [GopherCon 2018: Kat Zien - How Do You Structure Your Go Apps](#) и [GoLab 2018 - Massimiliano Pippi - Project layout patterns in Go](#)).

Ознакомьтесь с директорией `/pkg`, если хотите увидеть, какие популярные репозитории используют этот макет для организации проекта. Это часто используемый макет, но он не общепринятый, а некоторые в сообществе Go и вовсе не рекомендуют его использовать.

Вы можете не использовать эту директорию, если проект совсем небольшой и добавление нового уровня вложенности не несет практического смысла (разве что вы сами этого не хотите). Подумайте об этом, когда ваша корневая директория начинает слишком сильно разрастаться, особенно, если у вас много компонентов, написанных не на Go.

`/vendor`

Зависимости приложений, управляемые вручную или с использованием вашей любимой системы управления зависимостями, вроде доступного из коробки Go Modules. Команда `go mod vendor` создаст для вас директорию `/vendor`. Обратите внимание, что вам возможно придется добавить флаг `-mod=vendor` к команде `go build`, если вы используете версию, отличную от Go 1.14, где такой флаг выставлен по умолчанию.

Не стоит отправлять зависимости вашего приложения в репозиторий, если собираетесь создавать библиотеку.

Стоит отметить, что начиная с версии 1.13 Go добавил возможность проксирования модулей (с использованием <https://proxy.golang.org> как прокси-сервера по умолчанию). Здесь можно побольше узнать про эту возможность, чтобы убедиться, что она удовлетворяет вашим требованиям и ограничениям. Если это так - использование директории `vendor` не требуется вовсе.

Директории приложений-сервисов

`/api`

Спецификации OpenAPI/Swagger, JSON schema файлы, файлы определения протоколов.

Примеры смотрите в директории `/api`.

Директории Веб-приложений

/web

Специальные компоненты для веб-приложений: статические веб-ресурсы, серверные шаблоны и одностраничные приложения.

Распространенные директории

/configs

Шаблоны файлов конфигураций и файлы настроек по-умолчанию.

Поместите файлы конфигураций `confd` или `consul-template` сюда.

/init

Файлы конфигураций для инициализации системы (`systemd`, `upstart`, `sysv`) и менеджеров процессов (`runit`, `supervisord`).

/scripts

Скрипты для выполнения различных операций сборки, установки, анализа и т.п. операций.

Эти скрипты позволяют оставить основной `Makefile` небольшим и простым (например, <https://github.com/hashicorp/terraform/blob/main/Makefile>).

Примеры смотрите в директории `/scripts`.

/build

Сборка и непрерывная интеграция (Continuous Integration, CI).

Поместите файлы конфигурации и скрипты облака (AMI), контейнера (Docker), пакетов (`deb`, `rpm`, `pkg`) в директорию `/build/package`.

Поместите ваши файлы конфигурации CI (`travis`, `circle`, `drone`) и скрипты в директорию `/build/ci`. Обратите внимание, что некоторые инструменты CI (например, Travis CI) очень требовательны к расположению их конфигурационных файлов. Попробуйте поместить конфигурационные файлы в директорию `/build/ci` создав ссылку на них в месте, где их ожидают найти CI инструменты (если возможно).

/deployments

Шаблоны и файлы конфигураций разворачивания IaaS, PaaS, системной и контейнерной оркестрации (`docker-compose`, `kubernetes/helm`, `mesos`, `terraform`, `bosh`). Стоит заметить, что в некоторых репозиториях (особенно в приложениях, развернутых с использованием Kubernetes) эта директория называется `/deploy`.

/test

Дополнительные внешние тестовые приложения и данные для тестирования. Вы вольны организовывать структуру директории `/test` так, как вам угодно. Для больших проектов имеет смысл создавать вложенную директорию с данными для тестов. Например, `/test/data` или `/test/testdata`, если вы хотите, чтобы Go игнорировал находящиеся там файлы. Стоит заметить, что Go будет также игнорировать файлы,

путь к которому начинается с "." или "_", что предоставляет вам гибкость в наименовании вашей директории с тестовыми данными.

Примеры смотрите в директории /test.

Другие Директории

/docs

Проектная и пользовательская документация (в дополнение к документации сгенерированной godoc).

Примеры смотрите в директории /docs.

/tools

Инструменты поддержки проекта. Обратите внимание, что эти инструменты могут импортировать код из директорий /pkg и /internal.

Примеры смотрите в директории /tools.

/examples

Примеры ваших приложений и/или библиотек.

Примеры смотрите в директории /examples.

/third_party

Внешние вспомогательные инструменты, ответвления кода и другие сторонние утилиты (например, Swagger UI).

/githooks

Git hooks.

/assets

Другие ресурсы, необходимые для использования вашего репозитория (изображения, логотипы и т.д.)

/website

Здесь можно разделить файлы для сайта вашего проекта, если вы не используете GitHub pages.

Примеры смотрите в директории /website.

Директории, которые не стоит размещать у себя в проекте

/src

Некоторые проекты на Go имеют директорию src, но это обычно происходит, когда разработкой занялся человек, пришедший из мира Java, где такой подход весьма распространен. Постарайтесь не использовать этот Java паттерн. Вы же не хотите, чтобы ваш код на Go или Go проект выглядел, будто написан на Java.

Не путайте директорию уровня проекта /src с директорией /src, которую Go использует для своих рабочих пространств, как это описано в [How to Write Go Code](#). Переменная окружения \$GOPATH указывает на ваше (текущее) рабочее пространство (по-умолчанию она указывает на \$HOME/go на системах под управлением ОС, отличной от Windows). Это рабочее пространство включает высокоуровневые директории /pkg, /bin и /src. Ваш проект в свою очередь находится в директории вложенной в директорию /src, поэтому если у вас есть директорию /src внутри вашего проекта, путь к проекту будет выглядеть примерно так: /some/path/to/workspace/src/your_project/src/your_code.go. Стоит заметить, что в версиях Go начиная с 1.11 ваш проект может хранить за пределами вашего GOPATH, но это всё еще не значит, что применять этот шаблон компоновки - это хорошая идея

Рисунок 2

На рисунке 2 показан пример get запроса для сервиса count.
Для дальнейшей проверки отправим POST запрос с телом JSON

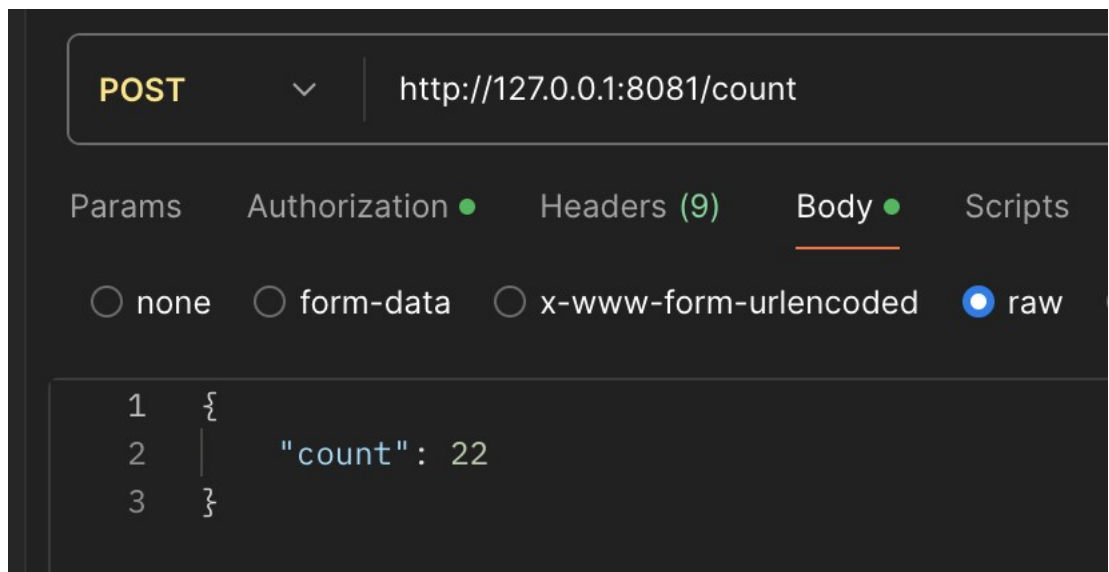


Рисунок 3


	number integer 
1	222

Рисунок 4

На рисунке 4 показано то, как это хранится в базе данных

Сервис Hello

Данный сервис должен возвращать Hello, something! На get request.

	message character varying (255) 🔒
1	Hello Stepan
2	Hello Emin
3	Hello Mike

Рисунок 4

Здесь показано как возможные сообщения хранятся в бд.

Также я добавил возможность добавления собственных сообщений через POST запрос с передачей JSON объекта.

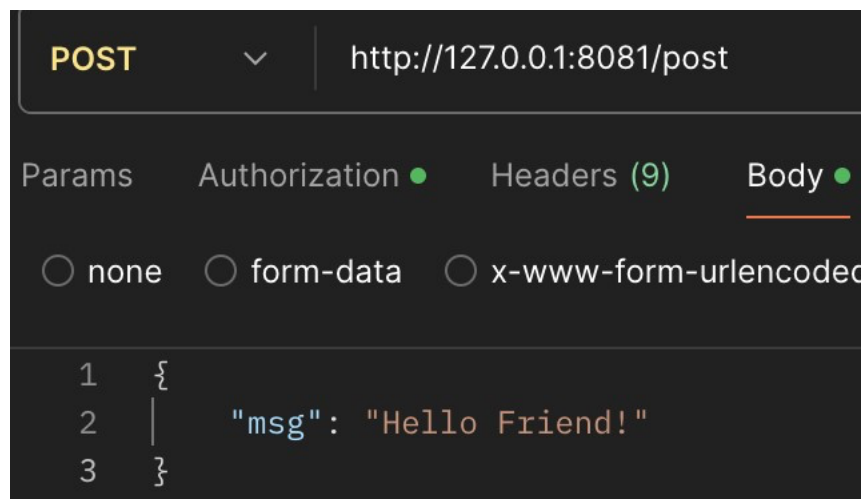


Рисунок 5(пример POST запроса)

На get запрос сервер отправляет случайное приветствующее сообщение

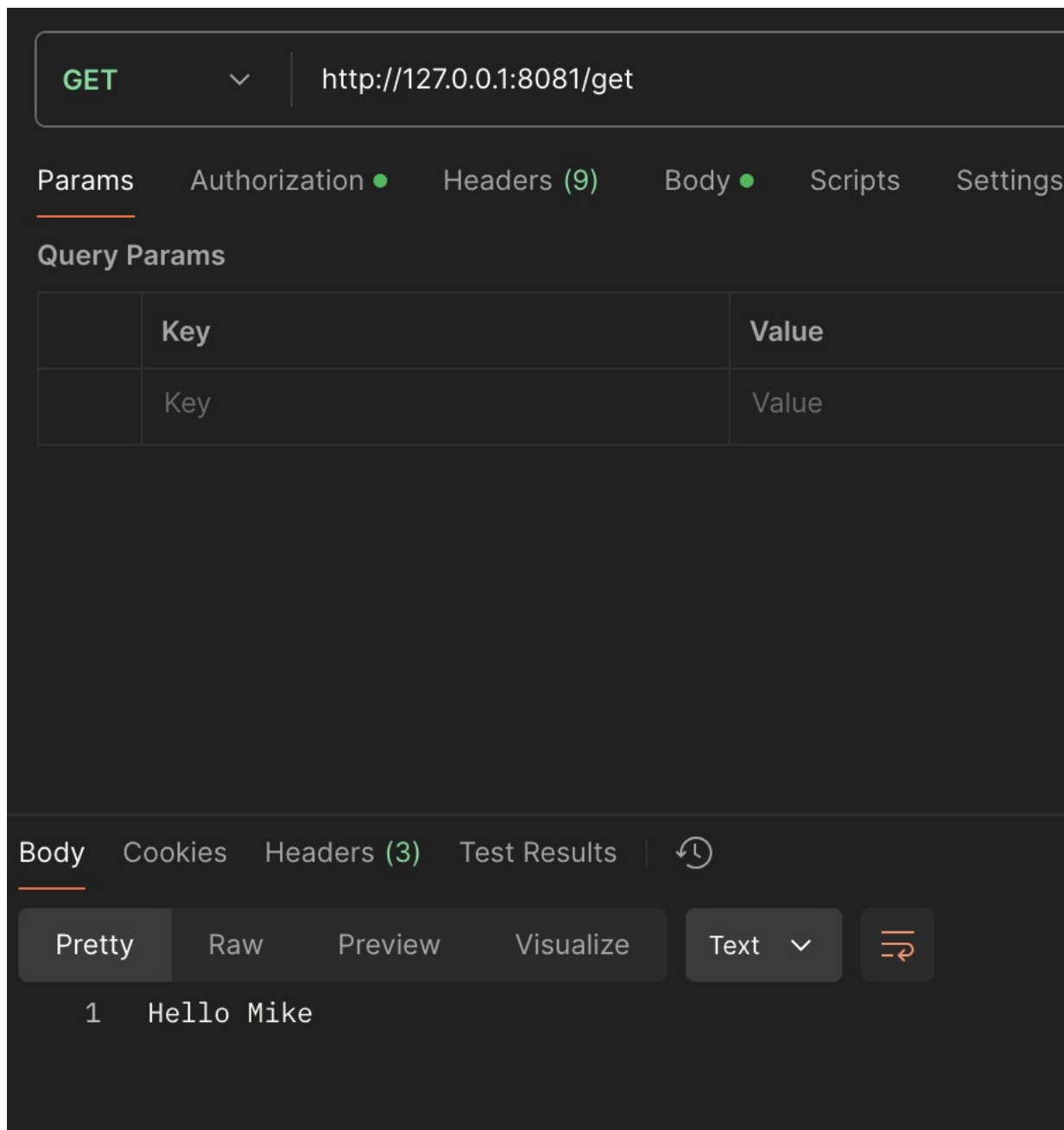


Рисунок 6

Сервис Query

В данном сервисе мы должны возвращать Hello <username> на get запросы пользователя если такой уже существует в бд.

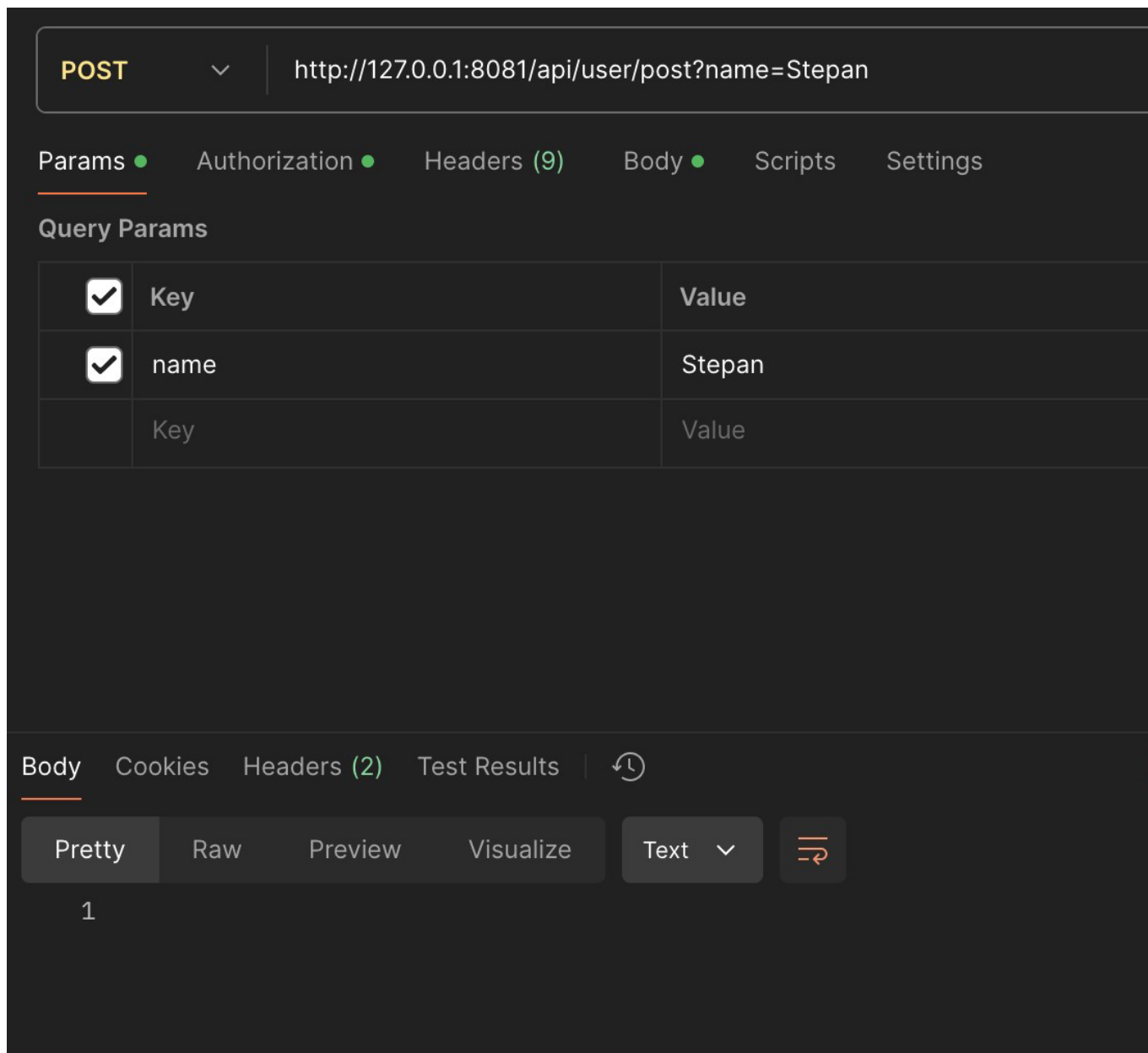


Рисунок 7 (Пример добавления имени в бд)

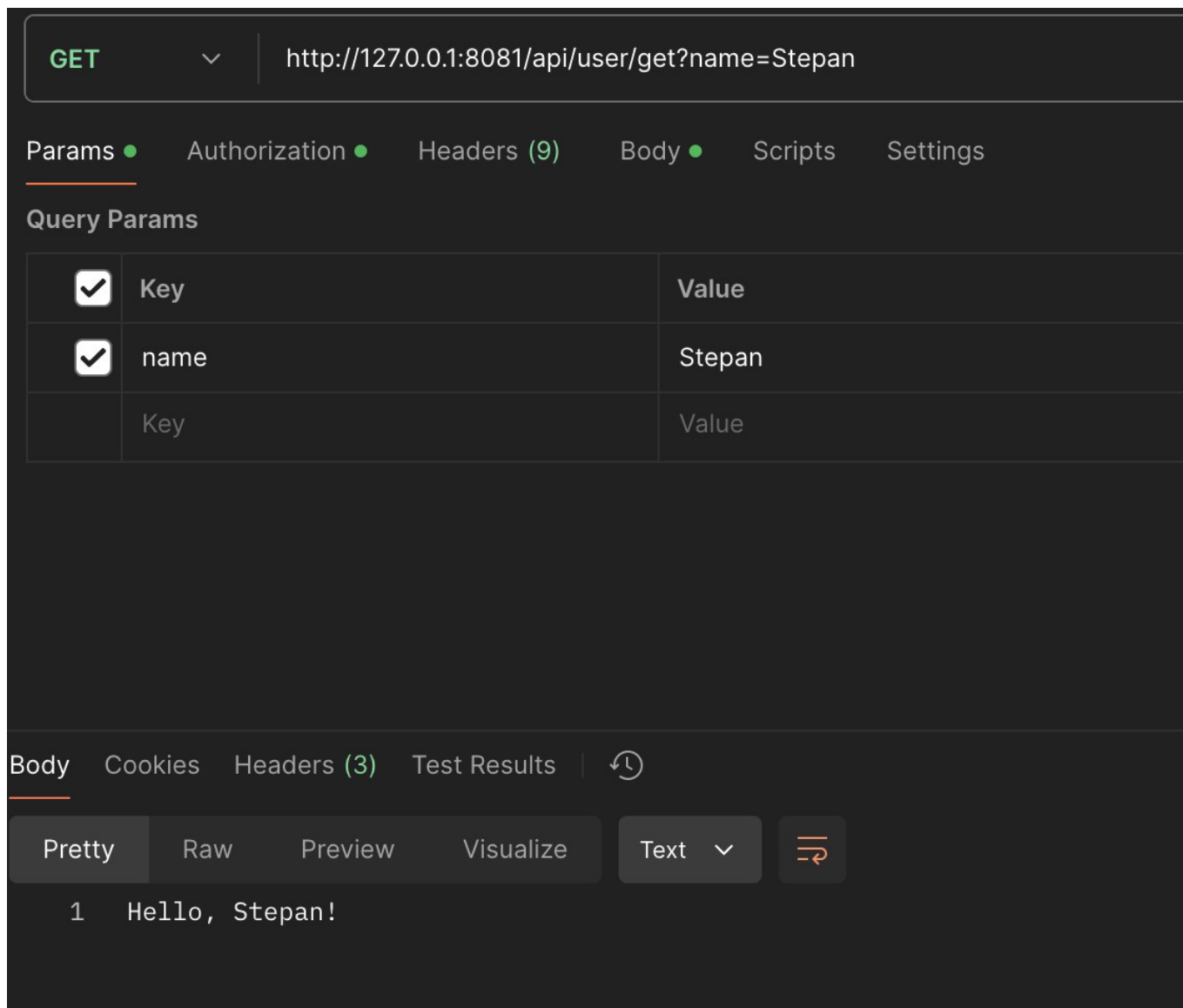


Рисунок 8 (пример запроса GET с тем же параметром имени)

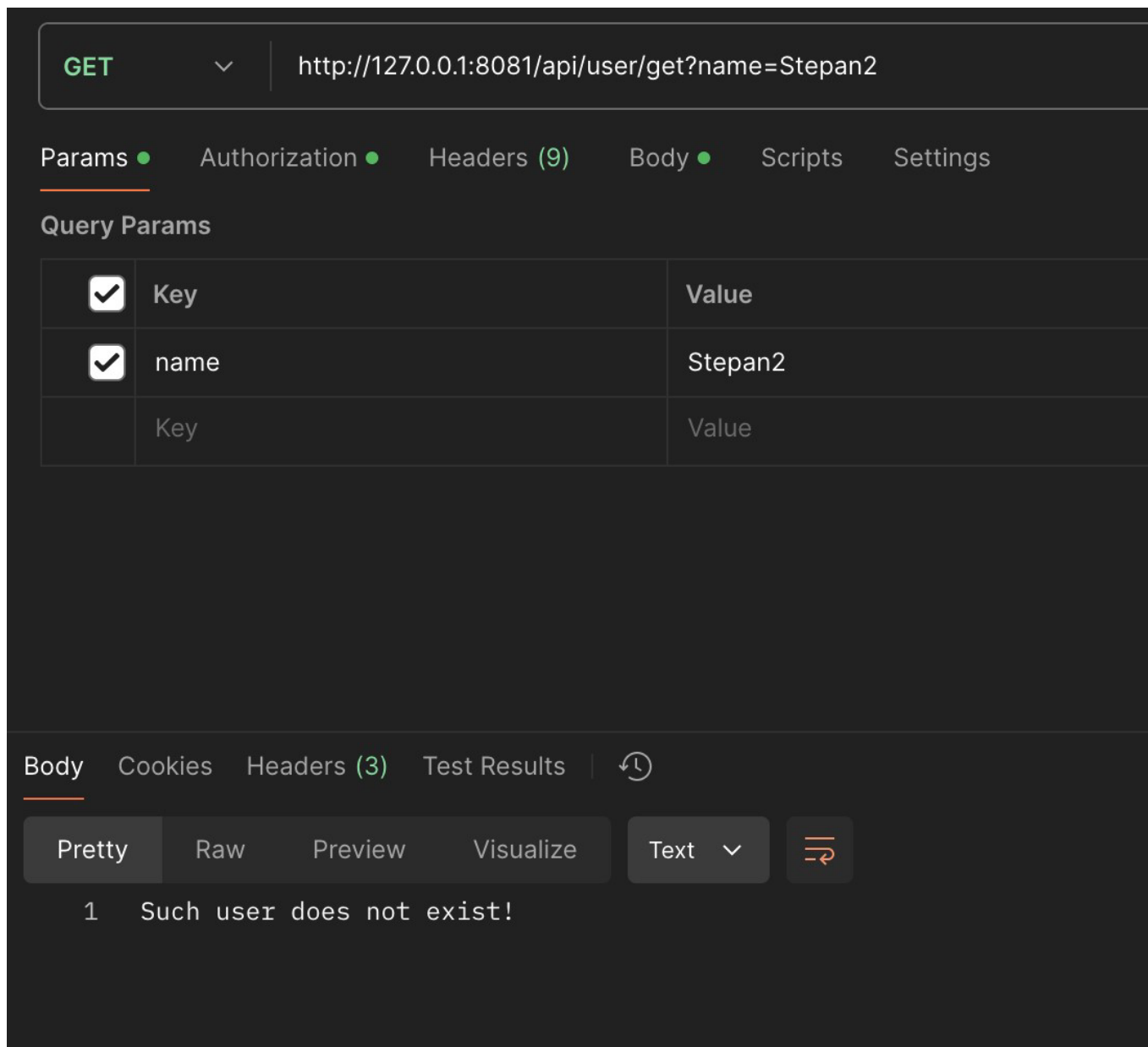


Рисунок 9(В случае отсутствия такого имени в бд, будет выдана ошибка)

Заключение: Я научился создавать веб-сервер с чистой архитектурой

Москва, 2024