Geekbrains

Frontend-разработчик

Разработка приложения для создания и хранения резюме

Frontend-программист

Цифровые профессии

Кутуева Ольга Сергеевна

Санкт-Петербург

2024

Содержание

[Введение 4](#_Toc160969703)

[Инструменты, использованные в работе 5](#_Toc160969704)

[Проектирование программной архитектуры 9](#_Toc160969705)

[Общая архитектура системы 9](#_Toc160969706)

[User Stories 12](#_Toc160969707)

[Er-диаграмма 15](#_Toc160969708)

[Технологии, планируемые к использованию в проекте 18](#_Toc160969709)

[MySQL 18](#_Toc160969710)

[Docker 20](#_Toc160969711)

[Выбор языка программирования 22](#_Toc160969712)

[Изучение выбранных технологий, подготовка, эксперименты 23](#_Toc160969713)

[Запуск MySQL в Docker 24](#_Toc160969714)

[Инициализация базы данных 30](#_Toc160969715)

[Выполнение SQL-инструкций в docker-контейнере MySQL 31](#_Toc160969716)

[Разработка запросов к базе данных 33](#_Toc160969717)

[Реализация каскадного удаления 35](#_Toc160969718)

[Настройка среды VSCode 36](#_Toc160969719)

[MySQL Connector - Python 39](#_Toc160969720)

[Хэширование паролей в web-приложениях. Библиотека passlib 44](#_Toc160969721)

[Markdown 45](#_Toc160969722)

[Микрофреймфорк Flask 47](#_Toc160969723)

[Механизм маршрутизации в web-приложениях 48](#_Toc160969724)

[Выводы 50](#_Toc160969725)

[Развитие проекта 51](#_Toc160969726)

[Список литературы 52](#_Toc160969727)

# Введение

Дипломный проект представляет собой онлайн-конструктор резюме — это инструмент, позволяющий пользователю создавать резюме в электронном виде, а затем отправить ссылку на резюме работодателю. В электронном резюме можно хранить информацию об образовании, опыте работы, навыках и достижениях. Онлайн-конструктор может быть полезны для людей, которые ищут новую работу или хотят обновить свое резюме, он позволяет сэкономить время и усилия на создание резюме, а также предоставляет возможность проверить и отредактировать уже готовое резюме перед отправкой.

**Цель проекта:** освоить полный цикл разработки программного обеспечения, пройти все стадии – от идеи до готового продукта. Разобраться, как реализовывается взаимодействие серверной части (backend), базы данных и клиентской части (frontend) приложения. Для этой цели разработать онлайн-конструктор резюме, который будет включать в себя все вышеуказанные компоненты.

План работы:

1. Определить цели и задачи проекта: какие функции должен выполнять конструктор резюме, какие возможности он должен предоставлять пользователям,
2. Разработать архитектуру системы — из каких частей будет состоять система, как они будут связаны между собой,
3. Описать инструменты и технологии для разработки,
4. Изучить описанные технологии и компоненты на уровне простых экспериментов,
5. Реализация готового продукта.

# Инструменты, использованные в работе

1. Хранение проекта: GitHub

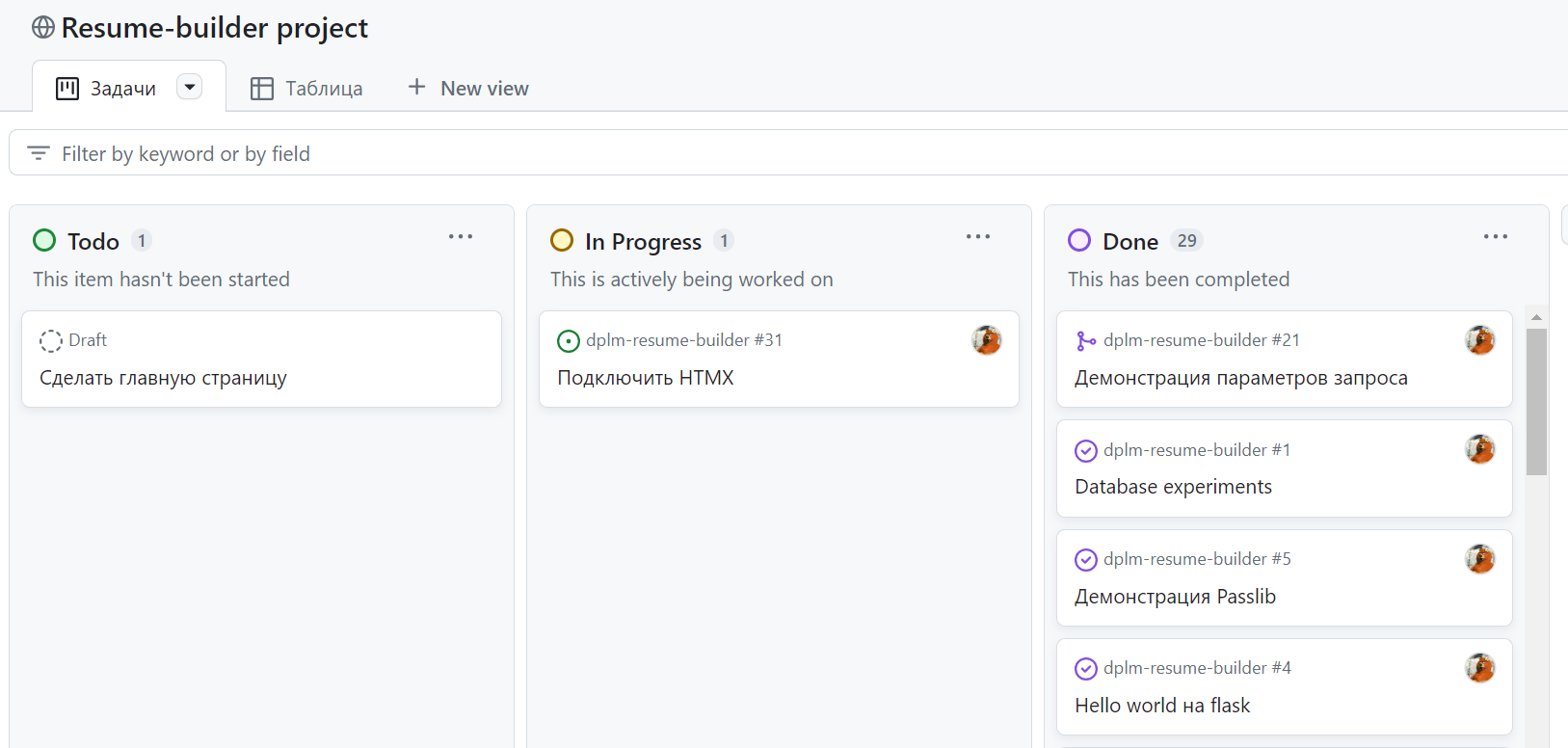
GitHub – это веб-платформа для хостинга и совместной работы над проектами с использованием системы контроля версий Git. Она предоставляет возможность разработчикам хранить, управлять и отслеживать свой код, а также обеспечивает средства для совместной разработки, управления задачами и обсуждения изменений.

Ссылка на репозиторий: <https://github.com/she-bear/dplm-resume-builder>

2) Управление проектом: GitHub Projects

GitHub Projects – это инструмент, предоставляемый самим GitHub для управления проектами и организации задач. Он позволяет создавать задачи, устанавливать приоритеты, управлять карточками задач, а также отслеживать прогресс разработки проекта. GitHub Projects интегрируется с репозиториями на GitHub, что позволяет удобно связывать код и задачи.

Ссылка на GitHub Projects проекта: https://github.com/users/she-bear/projects/1



3) Средство разработки (IDE): Visual Studio Code с WSL

VSCode (Visual Studio Code) – это бесплатный редактор с открытым исходным кодом. Он предоставляет широкие возможности для разработки программного обеспечения, включая поддержку многих языков программирования, инструменты для отладки, автоматизацию задач, интеграцию с системами контроля версий и многие другие функции.

VSCode отличается наличием обширной базы расширений, которые позволяют настраивать и расширять функционал редактора под нужды разработчика. Он также обладает удобным интерфейсом, быстрым запуском и отзывчивостью.

Преимущества использования в данном проекте:

1. интеграция с GitHub;
2. обширная поддержка для языка Python, включая интегрированный интерпретатор, подсветку синтаксиса, автодополнение кода и другие функции, упрощающие разработку;
3. возможность установки различных расширений, которые облегчают разработку на Python и Flask, такие как поддержка виртуальных окружений, отладка кода и работа с базами данных;
4. предоставление доступа к командной строке напрямую из интерфейса, что удобно для запуска сервера Flask, установки зависимостей и управления виртуальными окружениями;
5. поддержка markdown;
6. поддержка работы с Node.JS;
7. поддержка отладчика для Docker, что делает возможным отладку контейнеров Docker (для запуска mysql, например).

WSL (Windows Subsystem for Linux) – это среда подсистемы для запуска систем Linux внутри операционной системы Windows. WSL предоставляет возможность разработчикам выполнять команды и запускать утилиты Linux в среде Windows без необходимости установки фактической операционной системы Linux. Также, WSL позволяет использовать Docker-контейнеры для запуска приложений на Linux-серверах, что делает разработку более удобной и быстрой.

Visual Studio Code (VSCode) с Windows Subsystem for Linux (WSL) объединяет возможности редактора кода Visual Studio Code с возможностями Linux-среды, предоставляемой WSL.

4) Ведение документации: Notion

Notion – это платформа для организации, планирования и совместной работы над проектами, задачами и документами. Она предоставляет инструменты для создания различных типов контента – от заметок и списков до баз данных и досок задач. Notion позволяет пользователям создавать персонализированные рабочие пространства и организовывать информацию по своим потребностям.

В данном проекте Notion использовался как для ведения документации, так и для совместной работы (обмен информацией, заметками, уведомлениями и комментариями с руководителем проекта).

5) Совместное планирование и обсуждение: Excalidraw

Excalidraw – это открытое и свободное программное обеспечение для создания векторных диаграмм и чертежей. Он предоставляет интуитивный интерфейс и возможность быстро и легко создавать простые или сложные изображения, отображая структуру, процессы или концепции.

В данном проекте для обсуждения использовались разделяемые доски, и также экспорт и импорт диаграмм в различные форматы.

Разработка велась под руководством наставника: Яковлева Владимира Вячеславовича, что позволило дополнительно освоить технологию командной разработки (работа с Issues в GitHub Projects, CodeReview в Git).

# 

# Проектирование программной архитектуры

Для начала определим общую структуру и организацию системы, ее компоненты, их взаимосвязь и взаимодействие. Это позволит учесть основные аспекты проектирования до начала разработки и обеспечить возможность масштабирования и изменения в будущем.

Для этого рассмотрим укрупненную архитектуру системы, user stories (описание пользовательских историй) и er-диаграмму (диаграмму сущность-связь). Каждый из этих аспектов представляет собой способ представления или описания системы с разных точек зрения.

## Общая архитектура системы

Архитектура системы при разработке программного обеспечения представляет собой структурное описание программной системы, включающее в себя описание ее компонентов, их взаимодействие и отношения. Она определяет общий механизм работы системы, ее основные компоненты и их взаимодействие, а также описывает принципы, на основе которых происходит создание программного обеспечения.

Архитектура системы обычно включает в себя высокоуровневые диаграммы, описывающие структуру системы, ее модули и связи между ними, а также принципы, на которых основано разделение системы на компоненты и их взаимодействие.

Формирование архитектуры системы является одним из первых и самых важных этапов в разработке программного обеспечения, поскольку она определяет общее строение системы, которое в дальнейшем будет использоваться для разработки, тестирования и поддержания программного продукта.

Архитектурная структура web-приложения обычно состоит из четырех основных частей, каждая из которых будет входить в разрабатываемый проект (см. рис. 1):

**Backend (серверная часть)** – это часть приложения, которая отвечает за обработку данных и выполнение логики. Она обычно работает на стороне сервера и взаимодействует с frontend через API. Backend может включать в себя различные компоненты, такие как серверы приложений, базы данных и сервисы интеграции.

**Сервер** – это аппаратное или виртуальное устройство, которое обеспечивает вычислительные ресурсы для работы приложений. На сервере может работать один или несколько экземпляров backend-приложения. В зависимости от архитектуры приложения, сервер может также содержать базу данных или другие компоненты.

**База данных** – это система хранения данных, которая используется для хранения и управления информацией, необходимой для работы приложения. Backend-приложение обычно взаимодействует с базой данных через SQL или другие языки запросов. База данных может быть локальной или удаленной, и может находиться на том же сервере, что и backend, или на другом сервере.

**Frontend (клиентская часть)** – это интерфейс, который пользователь видит и с которым взаимодействует. Он обычно работает на стороне клиента и общается с backend через API для получения и отправки данных. Frontend может быть реализован в виде web-страницы, мобильного приложения или другого интерфейса, который позволяет пользователю взаимодействовать с приложением.

Выбор конкретных технологий для разработки отдельных частей системы не входит в рамки специальности и данного дипломного проекта, технологии были предложены дипломным руководителем.

|  |
| --- |
| Рис. 1 Общая архитектура системы |

Совместное использование сервера Nginx с базой данных MySQL и web-приложением, написанным на Python (Flask), представляет собой распространенную архитектурную модель для web-приложений:

**Nginx** используется в качестве web-сервера для обработки HTTP-запросов от клиентов и маршрутизации их к приложению Flask или статическим файлам. Nginx принимает запросы от клиентов и маршрутизирует их к приложению Flask на основе определенных правил маршрутизации.

База данных **MySQL** используется для хранения данных приложения и обеспечивает манипуляцию и управление данными, используемыми web-приложением Flask. Flask взаимодействует с базой данных MySQL для получения и обновления данных, необходимых для обработки запросов клиентов.

**Flask** – это микрофреймворк для Python, который используется для разработки web-приложений. Он обрабатывает HTTP-запросы от клиентов, взаимодействует с базой данных MySQL для получения и обновления данных, и возвращает ответы клиентам.

**Клиентские запросы** отправляются на сервер Nginx, Nginx получает **ответы** от приложения Flask и передает их обратно **клиентам.**

Как уже было указано выше, такая архитектурная модель является довольно распространенной и обеспечивает высокую производительность, при необходимости масштабируемость и безопасность для web-приложений, а также упрощает управление и конфигурацию различных компонентов. Кроме того, использование Nginx позволит легко добавить дополнительные сервисы и масштабировать приложение в случае его развития.

## User Stories

**User Stories** – это короткие описания функциональности, которую должно предоставить разрабатываемое программное обеспечение, рассказанные с точки зрения конечного пользователя.

В данном проекте User Stories помогут понять, какие функции действительно важны для пользователя и какие задачи пользователь будет выполнять, чтобы оценить функционал, который будет заложен в проект.

Диаграмма, используемая для описания User Stories, называется “Карта истории пользователя” (User Story Map). Такая диаграмма помогает визуализировать, как пользователь будет взаимодействовать с продуктом, и позволяет увидеть все User Stories в контексте реального использования продукта.

Каждая User Story обычно состоит из трех основных элементов:

* Кто (Who): определенный пользователь или роль, который будет пользоваться программным продуктом.
* Что (What): описывает конкретную функциональность или задачу, которую пользователь хочет выполнять.
* Зачем (Why): обоснование или цель того, почему пользователю нужна эта функциональность.

«Как [тип клиента], [хочу то-то], [чтобы делать что-то]».

В случае разрабатываемого конструктора резюме можно выделить две роли “who” – Пользователь и HR-специалист.

1. Я, как пользователь, хочу иметь профессионально оформленное резюме, ссылку на которое могу отправить для соискания работы.
   1. Я, как пользователь, хочу иметь возможность зарегистрироваться на сайте (US001).
   2. Я, как пользователь, хочу иметь возможность создать одно или несколько резюме (US003).
   3. Я, как пользователь, хочу войти на сайт, чтобы получить доступ к своим резюме (US002).
   4. Я, как пользователь, хочу иметь возможность отредактировать ранее созданное резюме, чтобы получить новый вариант резюме.
   5. Я, как пользователь, хочу удалить ранее созданное резюме.
   6. Я, как пользователь, хочу получить ссылку на резюме, чтобы отправить её для публичного просмотра.
2. Я, как HR-специалист, хочу открыть ссылку на резюме, чтобы просмотреть резюме соискателя.

Список действий пользователя:

* Регистрация (запрос логина и пароля)
* Авторизация (запрос логина и пароля)
* Получение списка резюме
* Создание резюме
* Редактирование резюме
* Удаление резюме
* Получение ссылки для публичного просмотра резюме

Список действий HR-специалиста:

* Получение резюме по прямой ссылке

На рисунке 2 приведена диаграмма User Stories для данного проекта.

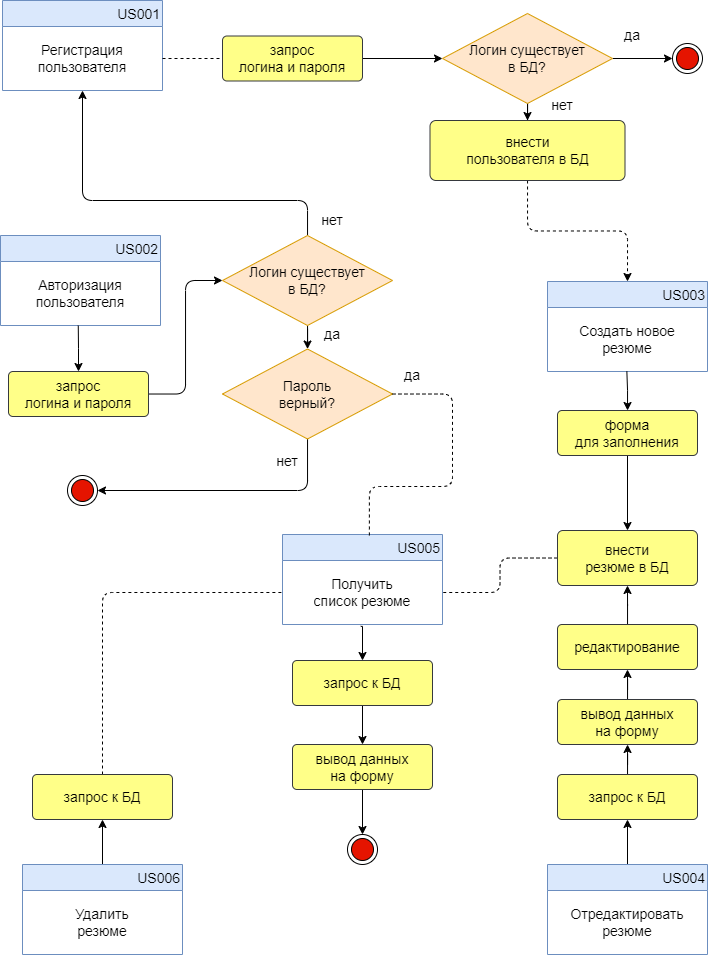


Рис. 2 Диаграмма User Stories

## Er-диаграмма

ER-диаграммы (Entity-Relationship diagrams) – это графическое представление структуры данных в информационной системе.

При разработке web-приложений ER-диаграммы играют важную роль. Они помогают определить структуру базы данных, которая будет использоваться приложением. ER-диаграммы позволяют описать сущности, их атрибуты и связи между ними, что помогает понять логику работы приложения и организовать данные эффективным образом.

Опираясь на user stories посмотрим, как может быть организовано хранение данных в контексте нашего приложения.

Итак, есть пользователь (User), для которого системе требуется хранить одно или несколько резюме. Пользователь будет иметь уникальный идентификатор (ID), логин (Login) и пароль (Password). Резюме пользователя также будет иметь уникальный идентификатор (ID), заголовок (Title) и текст резюме (Text).

Отношения сущностей представлены на рисунке 3.

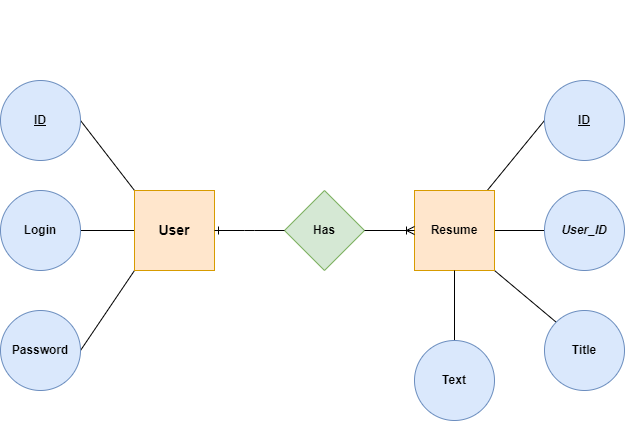


Рисунок 3. Отношение сущностей

На основе диаграммы отношения сущностей можем перейти к ERD-диаграмме базы данных, которая будет является визуальным представлением структуры базы данных, состоящей из таблиц (entities), связей между таблицами (relationships) и атрибутов (attributes), определенных для каждой таблицы (см. рисунок 4).

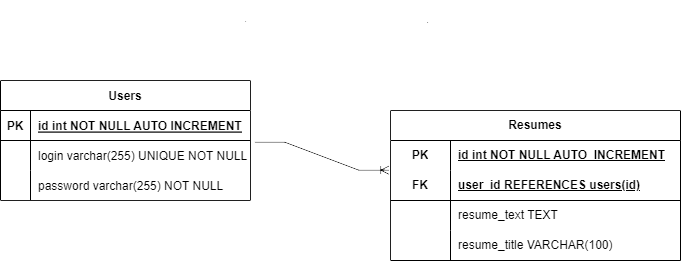


Рисунок 4. ERD-диаграмма базы данных

В дальнейшем, на основе этой структуры перейдем к непосредственному созданию таблиц базы данных на языке SQL, запросов и манипулирования данными.

# Технологии, планируемые к использованию в проекте

## MySQL

MySQL – это одна из самых популярных систем управления реляционными базами данных (СУБД), которая широко используется в web-разработке. MySQL предоставляет ряд функций и возможностей:

1. Открытый исходный код: MySQL имеет открытый исходный код, что означает, что его можно использовать бесплатно и без ограничений. Это делает его доступным для всех разработчиков и организаций.
2. Многоверсионность: MySQL поддерживает множество версий и редакций, что позволяет пользователям выбирать наиболее подходящую версию для своих нужд.
3. Масштабируемость: MySQL может масштабироваться от небольших проектов до крупных корпоративных систем с большим количеством пользователей.
4. Производительность: MySQL использует оптимизатор запросов, который автоматически оптимизирует запросы к базе данных для обеспечения максимальной производительности.
5. Безопасность: MySQL имеет встроенные функции безопасности, такие как шифрование данных, аутентификация пользователей и защита от атак на уровне приложений.

Причины выбора MySQL для данного проекта:

1. Открытый исходный код,
2. MySQL основан на реляционной модели данных (данные организованы в виде таблиц, связанных друг с другом), что как раз подходит для разработанной архитектуры системы (см. пп. Проектирование программной архитектуры),
3. Масштабируемость: в случае, если разрабатываемый проект в дальнейшем потребуется перевести на крупную базу данных, то MySQL справится с этой задачей,
4. Поддержка транзакций, обеспечение целостность данных и выполнение операций ACID (атомарность (A), согласованность (C), изоляция (I), долговечность (D)). При разработке не придется заботиться о контроле изменения данных, т.к. изменения данных, произведенные в результате выполнения одной транзакции, будут либо полностью применены, либо полностью отменены.
5. В данном проекте планируется использовать язык программирования Python c микрофреймворком Flask (см. далее). Для совместного использования MySQL и Flask, MySQL предоставляет модуль mysql-connector-python, который обеспечит прямое подключение к базе данных MySQL из приложения Flask без использования ORM-библиотеки.

## Docker

Docker – это платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеры Docker позволяют упаковать приложение в единообразный набор, включая код, среду выполнения и все зависимости, необходимые для его работы, такие как библиотеки, файлы конфигурации и т.д. Это обеспечивает изолированное и повторяемое окружение, которое можно легко перемещать между различными средами разработки и развертывания, а также масштабировать в зависимости от потребностей.

Рассмотрим основные преимущества:

1. Изолированность: контейнеры Docker обеспечивают изоляцию приложений и их зависимостей, что позволяет запускать несколько приложений на одном хосте без конфликтов и вмешательства друг в друга. Это особенно полезно в средах разработки и развертывания, где могут использоваться различные версии языков программирования или библиотек.
2. Портативность: Docker-контейнеры могут быть легко перемещены между различными средами, такими как разработка, тестирование и развертывание.
3. Легковесность и быстрота: контейнеры Docker используют виртуализацию на уровне операционной системы, что делает их легкими и быстрыми в запуске.
4. Масштабируемость: Docker позволяет масштабировать приложения горизонтально путем запуска нескольких контейнеров с одним и тем же образом приложения.
5. Управление версиями и ресурсами: Docker позволяет управлять версиями приложений и их зависимостями с помощью Dockerfile, что упрощает процесс развертывания и обновления приложений.

Использование Docker в данном проекте обусловлено:

1. Для работы с базой данной планируется использовать MySQL Server – Docker обеспечит изоляцию базы данных и её зависимостей в контейнере, а также портативность базы данных при её развертывании. Это позволит избежать конфликтов с другими приложениями или сервисами, работающими на хост-системе. В случае необходимости вертикального (изменение ресурсов контейнера) или горизонтального (добавление дополнительных экземпляров контейнера) масштабирования, Docker также обеспечит необходимое управление нагрузкой на базу данных.
2. В данном проекте планируется использовать язык программирования Python с web-фреймворком Flask (см. далее) – Docker позволит упаковать приложение Python с его зависимостями (включая Flask и любые другие библиотеки, необходимые для его работы) в контейнер. Это обеспечит изолированное окружение, в котором приложение может быть запущено без необходимости установки и настройки зависимостей на хост-системе.
3. Web-серверы: Docker поддерживает различные web-серверы, такие как Apache, Nginx, IIS и другие. В данном проекте будет использоваться сервер Nginx.

## Выбор языка программирования

Выбор языка программирования при разработке web-приложений имеет большое значение, поскольку от него зависит множество аспектов процесса разработки, а также производительность, эффективность и удобство использования конечного продукта. Нужно учитывать несколько ключевых аспектов: целевые потребности проекта, опыт разработчика, сообщество языка программирования (доступ к библиотекам, инструментам и ресурсам, которые могут упростить и ускорить процесс разработки), требования к производительности и безопасности, возможная масштабируемость проекта и интеграция с другими технологиями.

Учитывая полученный учебный опыт в GeekBrains на момент начала работы над проектом и все вышеописанные аспекты, дипломным наставником был предложен язык программирования Python и микрофреймворк Flask. На момент начала разработки были очевидны следующие преимущества Pyhon:

1. Простота и читаемость кода, а также уже имеющийся опыт.
2. Большое количество библиотек и фреймворков для web-разработки, которые имеют много готовых решений для разных задач. В частности, Python имеет библиотеки для работы с базами данных, включая MySQL.
3. Многоплатформенность: Python поддерживается на различных операционных системах, что делает его универсальным инструментом для разработки web-приложений, независимо от платформы.
4. Python имеет огромное и активное сообщество разработчиков, что обеспечивает доступ к множеству ресурсов, библиотек и решений проблем при разработке web-приложений, что очень важно для начинающих разработчиков.
5. Доступные в VSCode инструменты разработки и отладки.

# Изучение выбранных технологий, подготовка, эксперименты

Для последовательного изучения выбранных технологий, задача была разделена на небольшие части, каждой из которых соответствовала своя Issue в GitHub Projects (см. <https://github.com/users/she-bear/projects/1>). Порядок выполнения задач определялся наставником. По каждой задаче можно увидеть список подзадач и историю коммитов.

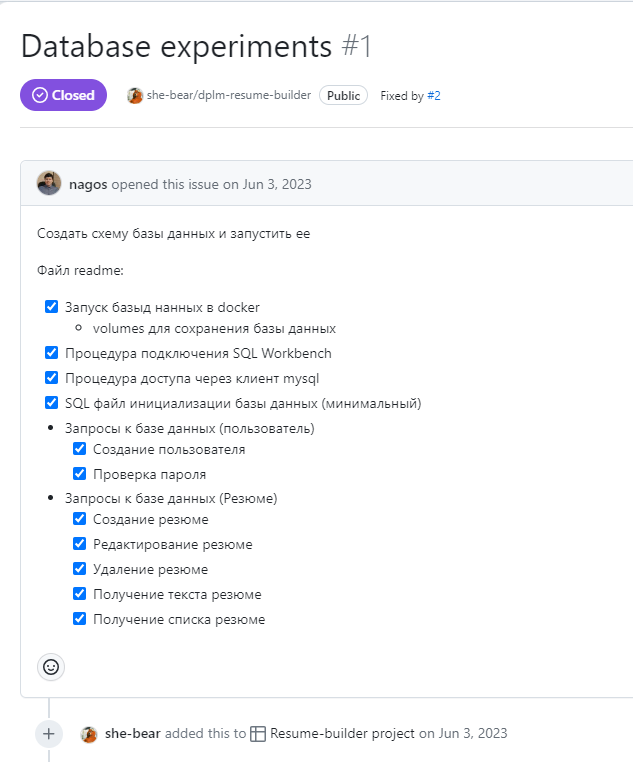


Рисунок N. Пример Issue из GitHub Projects

## Запуск MySQL в Docker

Для запуска MySQL в Docker необходимо:

Запустить приложение Docker Desktop на Windows

https://www.docker.com/products/docker-desktop/

Создать контейнер Docker

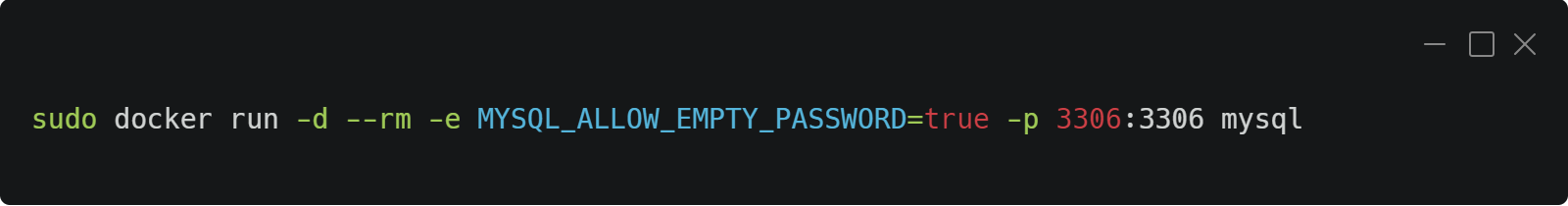
Для создания контейнера в Docker необходимо разработать команду запуска с нужными для выполнения задачи параметрами.

Базовая команда для создания docker-контейнера выглядит следующим образом:

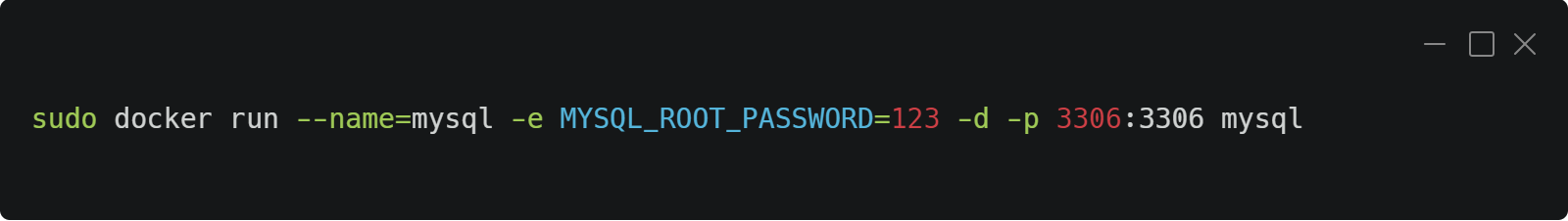


Согласно документации по запуску MySQL в Docker, формируем и проверяем команды:

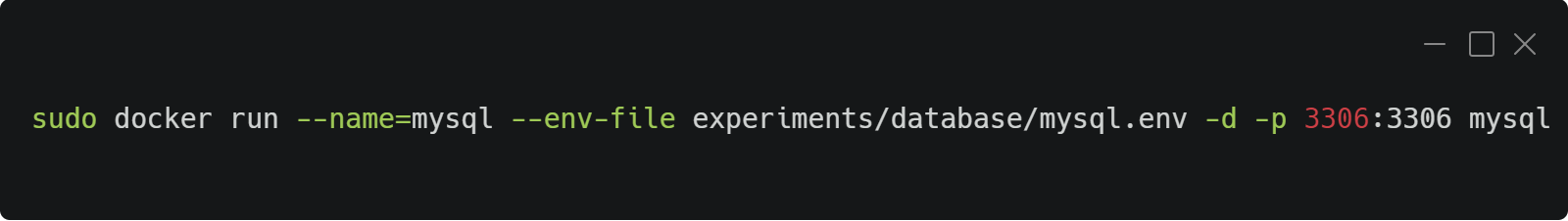
* Запустить контейнер MySQL с доступом без пароля, удалить контейнер после окончания работы):



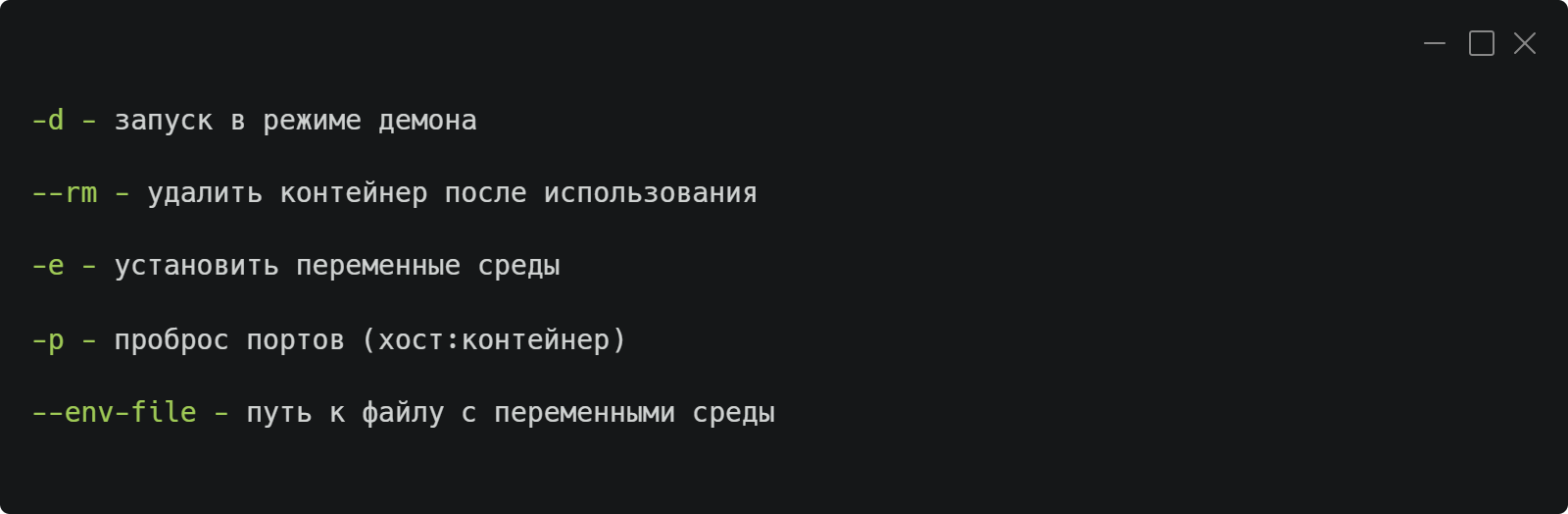
* Запустить с root-паролем (пароль задаем произвольный, для отладки):



* Запустить контейнер и считать переменные среды из файла (имя файла произвольное, путь – относительно места запуска):



Опции:



1. Env-файл, чтение переменных среды из env-файла

Файлы .env, или файлы переменных среды, широко используются в среде Docker для управления конфигурационными параметрами и переменными окружения при запуске контейнера. Они могут быть особенно полезны при работе с контейнером Docker MySQL для настройки параметров конфигурации базы данных.

Обычно файл .env при работе с контейнером Docker MySQL используется следующим образом:

* Определение переменных окружения: файл .env содержит набор переменных окружения в формате *KEY=VALUE*. В контексте MySQL, это могут быть переменные, такие как *MYSQL\_ROOT\_PASSWORD, MYSQL\_DATABASE, MYSQL\_USER, MYSQL\_PASSWORD* и т.д., которые используются для настройки доступа к базе данных и установки паролей.
* Использование переменных окружения в Docker Compose: файл .env может быть загружен в Docker Compose для автоматической подстановки переменных окружения при запуске контейнера (например, в файле docker-compose.yml, будет рассмотрен далее).
* Загрузка переменных окружения при запуске контейнера: при запуске контейнера Docker MySQL, переменные окружения из файла .env будут автоматически загружены в контейнер и использованы для настройки параметров конфигурации базы данных.

Составим env-файл согласно документации и требованиям нашей задачи:

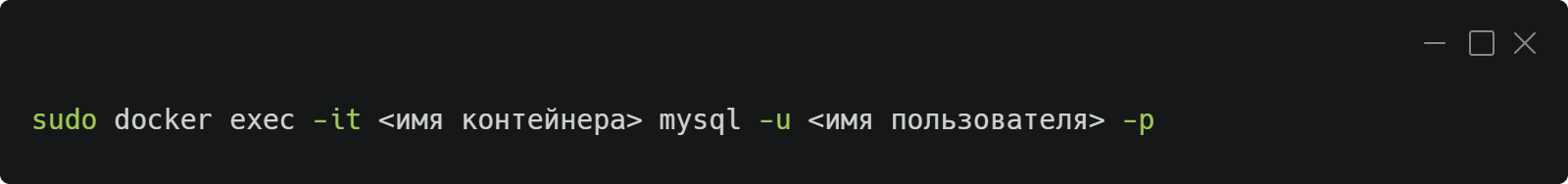


Пользователь с именем *MYSQL\_USER* и паролем *MYSQL\_PASSWORD* будет иметь доступ только к БД *MYSQL\_DATABASE*.

1. Доступ к данным внутри контейнера

В рамках эксперимента были изучены два способа доступа к данным внутри контейнера: доступ через клиент mysql из командной строки WSL и процедура подключения MySQL Workbench.

Согласно инструкции MySQL, команда запуска клиента mysql выглядит следующим образом:

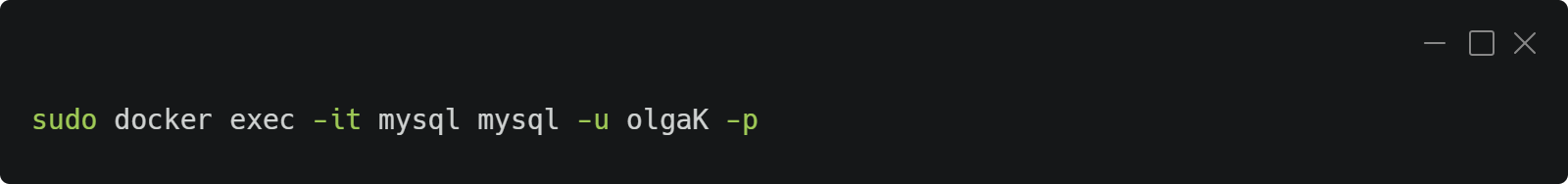


*<имя пользователя>* берётся из файла mysql.env (см. пп. 3), переменная *MYSQL\_USER*.

Войти и запросить пароль для root-пользователя (для случая с MYSQL\_ROOT\_PASSWORD=123):



В нашем случае команда примет вид:



Для подключения к контейнеру через MySQL Workbench (<https://www.mysql.com/products/workbench/>) после установки программы необходимо сделать следующее:

* Запустить БД в контейнере
* Запустить MySQL Workbench
* На главном экране выбрать MySQL Connections и задать параметры соединения с БД: Connection Method: Standard (TCP/IP), Host Name: 127.0.0.1, Port: 3306, User Name: root или заданный в параметре *MYSQL\_USER* в env-файле.

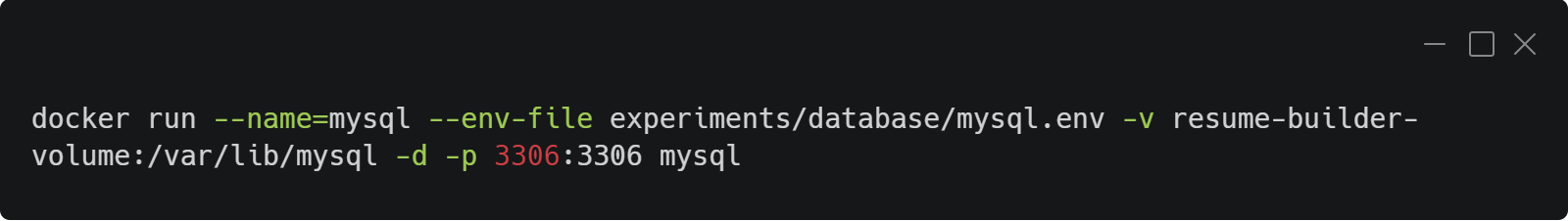
1. Хранение данных

При работе с MySQL, запущенным в Docker-контейнере, существует несколько способов хранения данных:

* Хранение данных внутри контейнера: по умолчанию, когда запускается контейнер MySQL, данные хранятся внутри контейнера. Это значит, что данные будут сохраняться только на время жизни контейнера, и при его удалении будут утеряны. Этот способ удобен для тестирования и разработки, но не может быть использован в production.
* Bind mount – монтирование каталога с хоста. Файл с данными хранится на хосте (в существующем каталоге) и открывается внутри контейнера.
* Docker volumes – данные сохраняются в именованный том, который располагается в определенном каталоге docker на хостовой машине и не удаляется при удалении контейнера. Том может быть подключен к нескольким контейнерам.
* Использование сетевых хранилищ (Network-attached storage, NAS).

Для решения нашей задачи будем использовать способ с docker volumes –этот способ не требует дополнительных забот с внешней папкой (проверки прав доступа, например).

Согласно документации по использованию volumes для docker-контейнера MySQL, команда создания нашего контейнера примет вид:



В рамках эксперимента выполним эту команду в WSL и оценим результат.

Проверим, что docker-volume создан:



Зайдем внутрь контейнера и создадим тестовую таблицу с данными:



Остановим контейнер:



Создадим новый контейнер вышеописанной командой, зайдем внутрь контейнера и убедимся, что данные сохранились:



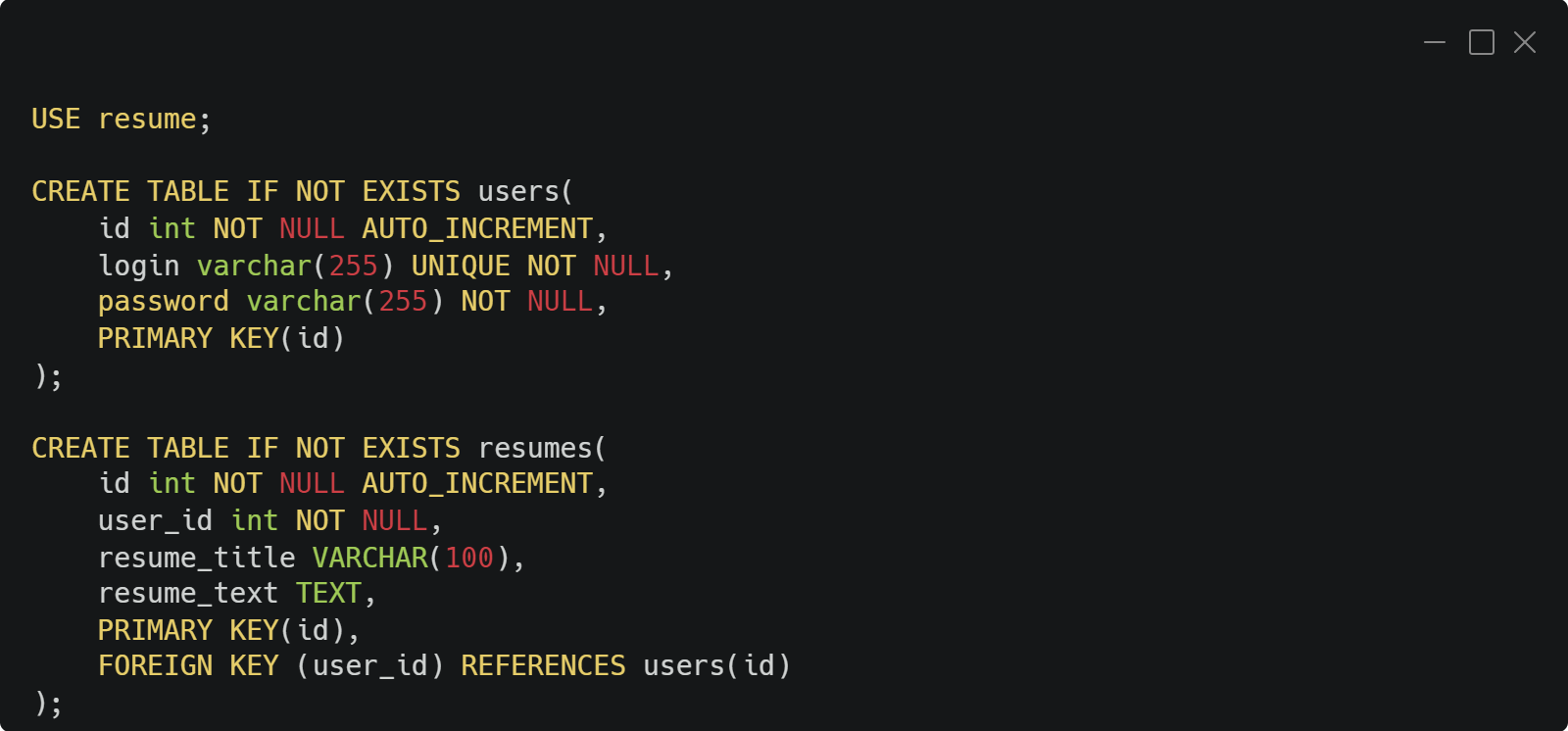
## Инициализация базы данных

Для инициализации базы данных при запуске контейнера используется файл ***db\_init.sql***. Этот файл содержит SQL-инструкции, которые будут выполнены при создании базы данных или при запуске контейнера MySQL впервые. Можно создать таблицы, индексы и другие объекты базы данных, осуществить начальную загрузку данных и настройку параметров базы данных.

Дальнейшее использование ***db\_init.sql*** обеспечивает воспроизводимость окружения базы данных при развертывании или масштабировании приложения. Также этот файл может быть интегрирован в процесс автоматического развертывания (CI/CD), что позволяет быстро настраивать базу данных в новых средах или при обновлении приложения.

Согласно разработанной ранее ER-диаграмме базы данных (см. пп. Проектирование программной архитектуры), наша БД состоит из двух таблиц users и resumes, связанных между собой внешним ключом.

Таким образом, на данном этапе для инициализации базы данных будем использовать файл ***db\_init.sql*** вида:



## Выполнение SQL-инструкций в docker-контейнере MySQL

Для ускорения процесса разработки, необходимо заранее отлаживать SQL-инструкции для работы с базой данных. Это позволит обнаружить и исправить потенциальные ошибки и проблемы в запросах, убедиться в корректности запросов и их соответствии требованиям к базе данных и приложению.

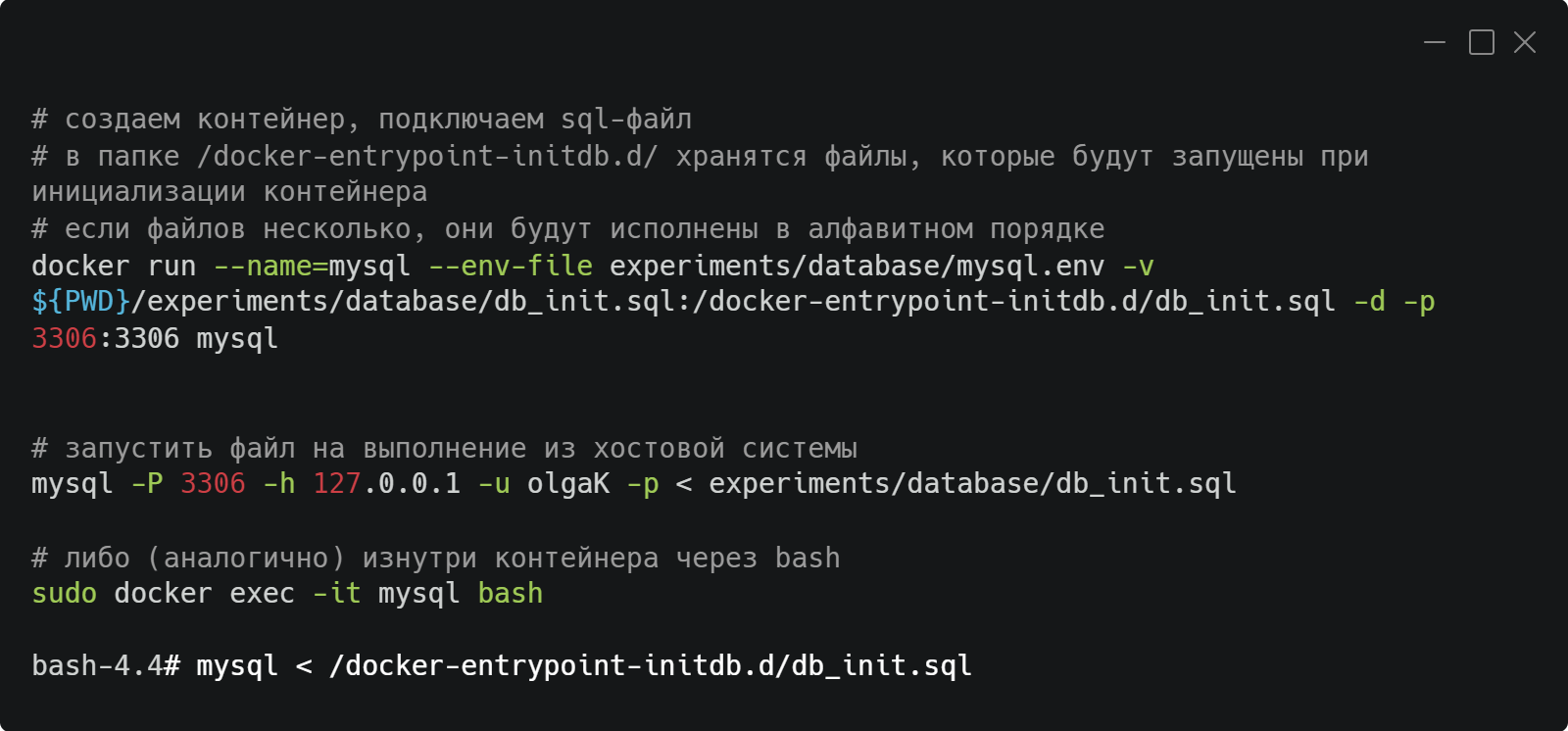
Разберёмся, как выполнять SQL-инструкции в docker-контейнере MySQL. Существует два способа:

1. Скопировать sql-файл внутрь контейнера:

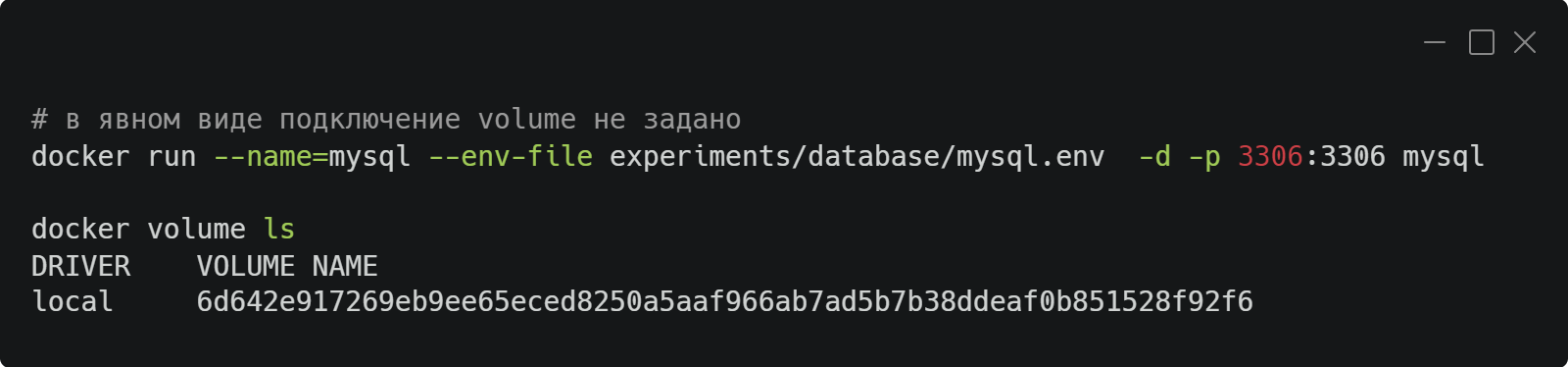


Недостатком этого способа является то, что SQL-файл будет скопирован внутрь контейнера один раз, и если потребуется изменить файл, то операцию нужно будет повторить (нет автоматизации).

1. Подключить файл через volume и запустить на выполнение из хостовой системы:

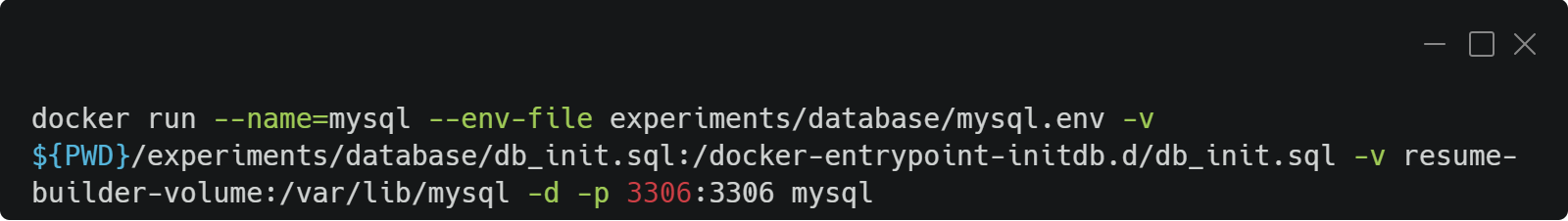


Нужно обратить внимание на то, что volume для хранения данных при использовании MySQL в контейнере создается всегда, даже если явно не указано, например:



В данном проекте будет использоваться способ, при котором SQL-файл запускается на выполнение автоматически при создании контейнера.

Таким образом, команда создания docker-контейнера примет вид:



## Разработка запросов к базе данных

Следующим шагом является предварительная разработка запросов к базе данных. Не будем подробно описывать заполнение базы данных тестовыми данными и результаты выборок, приведём только сами запросы.

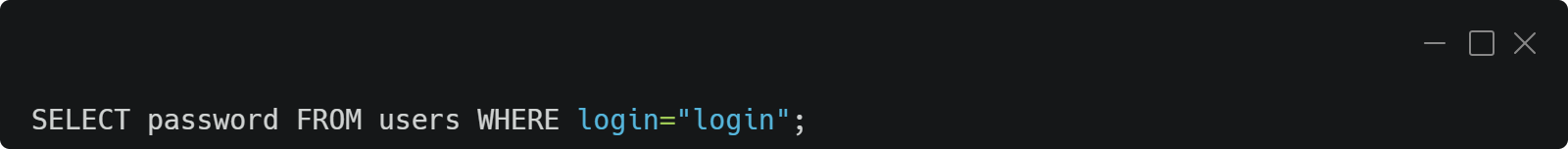
Создание пользователя:



Получение ID последнего добавленного пользователя:



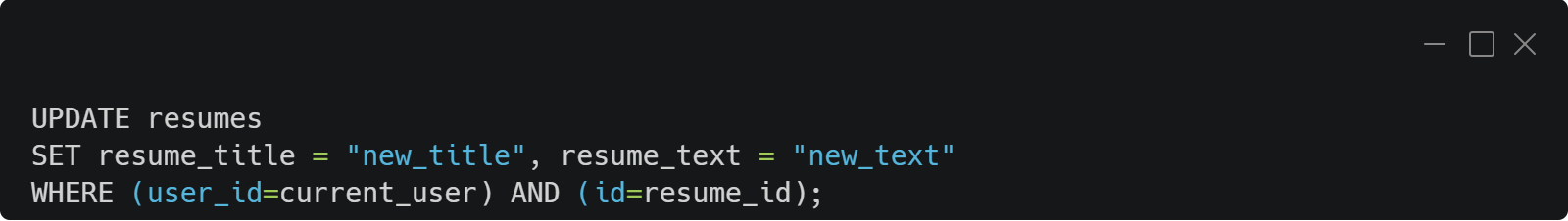
Для проверки пароля пользователя, будет необходимо получать пароль пользователя по его логину:



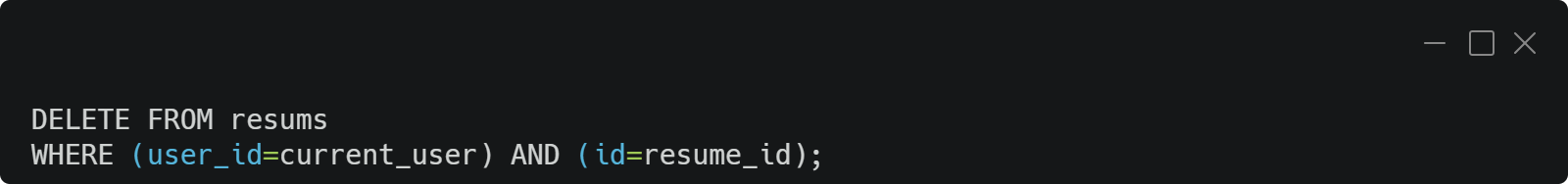
Создание резюме:



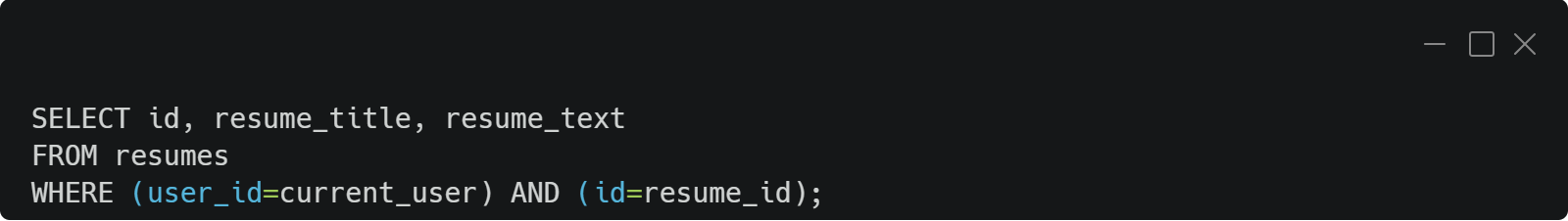
Редактирование резюме:



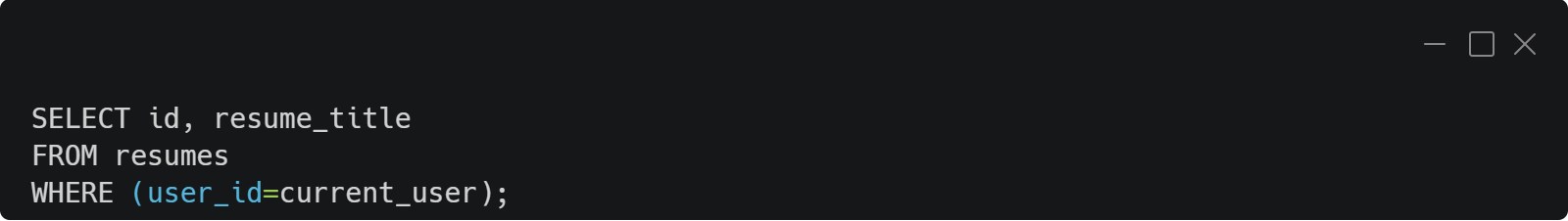
Удаление резюме:



Получение текста резюме:



Получение списка резюме:



Все приведенные выше запросы были отлажены на тестовых данных.

## Реализация каскадного удаления

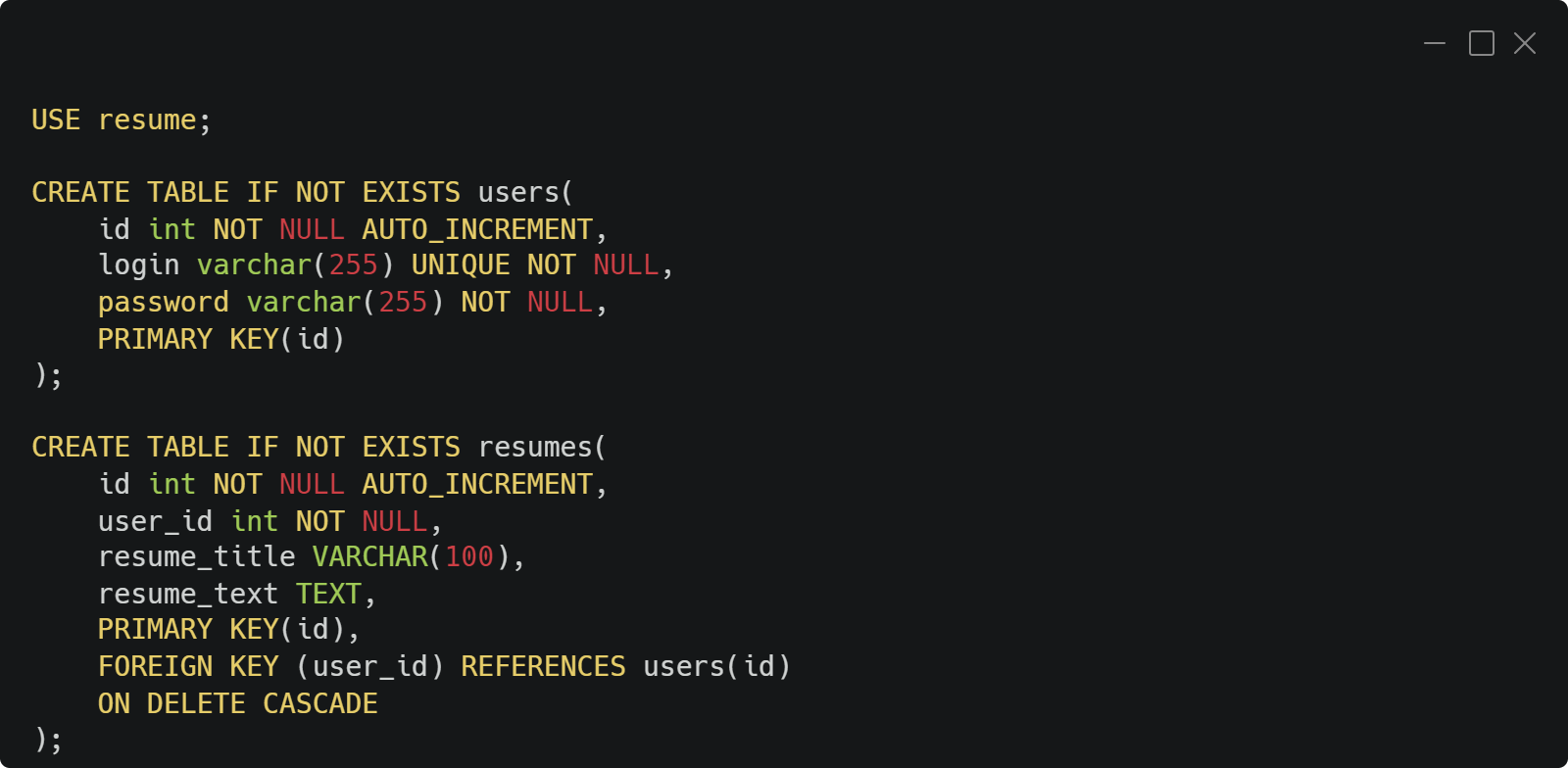
Каскадное удаление в базах данных – это процесс, при котором удаляются связанные записи из других таблиц при удалении основной записи. Реализуется с помощью механизма внешних ключей, который позволяет контролировать ссылочную целостность между таблицами.

В базе данных нашего проекта есть две таблицы – users (Пользователи) и resumes (Резюме), и одна запись пользователя может быть связана с несколькими резюме. При удалении пользователя из таблицы users автоматически удаляются все связанные записи этого пользователя из таблицы resumes.

Для реализации каскадного удаления в связанную таблицу необходимо добавить директиву:



Таким образом, на данном этапе работы файл инициализации базы данных ***db\_init.sql*** будет иметь вид:



## Настройка среды VSCode

Настройка Visual Studio Code (VS Code) перед началом работы с проектом важна, потому что это позволяет создать нужное рабочее окружение, соответствующее планируемым к использованию технологиям.

1. Установка необходимых расширений: Python Microsoft,
2. Интеграция с системой контроля версий: на GitHub был создан репозиторий, а затем подключен к удаленному remote с помощью стандартных команд,
3. Обновление пакетов для WSL



1. Установка PIP

PIP (Python Package Index) – это менеджер пакетов Python, который позволяет устанавливать, обновлять и удалять пакеты Python. PIP используется для установки пакетов из репозиториев пакетов.



1. Установка и настройка виртуального окружения .venv, запуск py-файлов

Venv (Virtual Environment) – это виртуальное окружение, которое создается для изоляции среды разработки от других проектов и зависимостей. Оно позволяет создавать отдельные среды для каждого проекта и устанавливать только нужные пакеты и зависимости.

В корневой папке проекта выполняем следующие команды:

Установка venv:



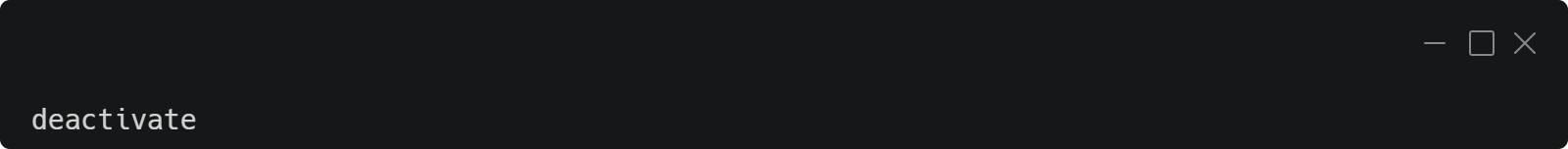
Cоздать виртуальную среду с именем .venv (в корневой папке появится папка .venv):



Активировать виртуальную среду. При срабатывании перед командной строкой появится (.venv):



Отключение виртуальной среды:



Также виртуальное окружение может быть настроено через Command Palette VSCode.

После активации виртуального окружения и перехода в каталог с py-файлом, можно запустить py-файл, указав путь к интерпретатору Python из виртуального окружения (в данном случае – Python 3.10.12).

Py-файл будет запущен и сможет использовать все установленные в этом окружении зависимости (см. далее) и библиотеки.

1. Файл зависимостей requirements.txt

Файл зависимостей requirements.txt в проектах на языке программирования Python используется для описания списка зависимостей, необходимых для работы приложения или проекта. Этот файл обычно используется вместе с инструментами управления зависимостями, такими как pip, чтобы установить все необходимые библиотеки и пакеты одной командой.

Каждая строка в файле requirements.txt обычно представляет собой одну зависимость и ее версию. Версии зависимостей указывать важно, чтобы обеспечить воспроизводимость окружения.

Файл requirements.txt можно сгенерировать автоматически на основе текущего окружения и установленных в нем пакетов с помощью команды (эта команда записывает список установленных пакетов в файл requirements.txt, включая их названия и версии):



После того как файл requirements.txt создан, его можно использовать для установки всех зависимостей проекта:



Файл requirements.txt будет меняться по мере разработки проекта и добавления новых зависимостей. Мы не будем каждый раз останавливаться на его изменении. Окончательную версию можно увидеть в репозитории проекта.

## MySQL Connector - Python

Следующим шагом будет освоение использования модуля **mysql-connector-python**. Как уже было описано выше, этот модуль позволяет взаимодействовать с базой данных MySQL из приложения, написанного на Python.

Для установки модуля в текущей виртуальной среде выполним команду:



Теперь необходимо установить связь с сервером базы данных, запущенным в контейнере Docker. Но для начала нужно экспортировать, а затем получить список переменных окружения (env) для работы с базой данных.

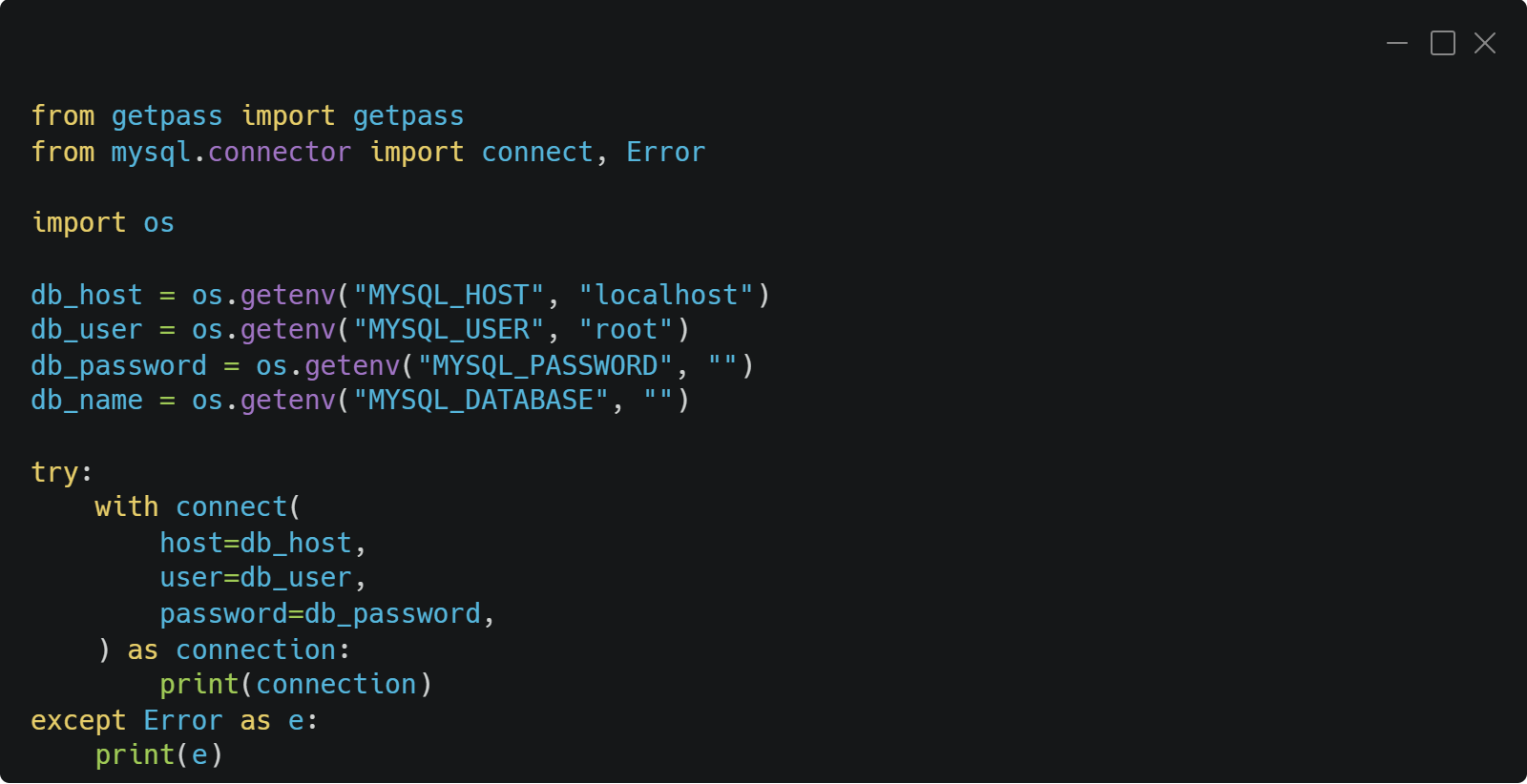
Для того, чтобы экспортировать переменные окружения нужно:

* в папке .venv/bin/ находим файл activate
* добавляем в этот файл экспорт переменных окружения:



Команда export экспортирует переменную в окружающую среду оболочки так, чтобы ее значение стало доступным для MySQL и других процессов.

В Python чтение переменных окружения осуществляется через os.environ object. В отличие от прямого доступа к объекту, функция os.getenv не вызовет ошибки если нет переменной, а возьмет значение по умолчанию (если значение по умолчанию не указано, то None). Таким образом, на данном этапе мы можем осуществить подключение к базе данных:



Теперь попробуем сформировать запрос к базе данных из нашей py-программы.

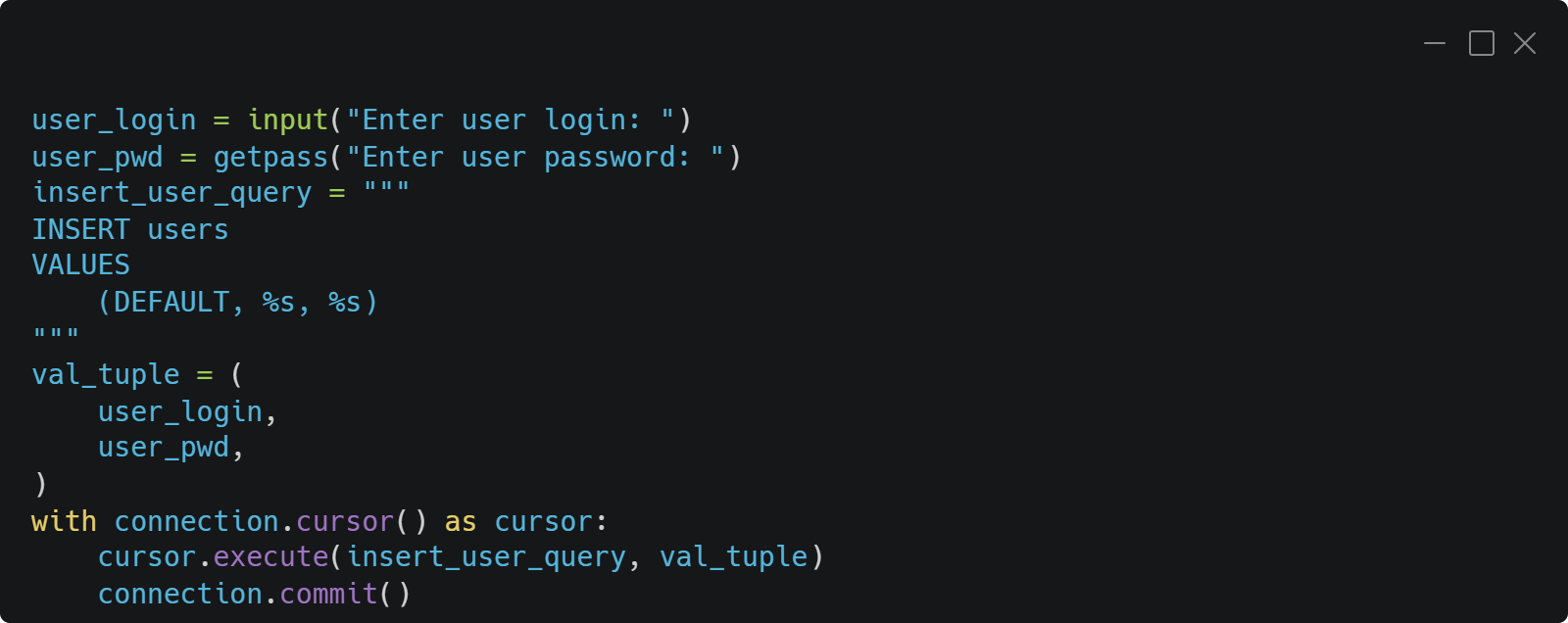
Особенностью формирования запросов к БД является то, что нельзя добавлять значения запроса, предоставленные пользователем, напрямую в строку запроса, это влечёт неустойчивость к SQL-инъекции. SQL-инъекция – это вид атаки на web-приложение, при которой злоумышленник внедряет SQL-код в пользовательские поля форм, чтобы получить доступ к базе данных и выполнить различные действия. Это может привести к утечке конфиденциальной информации, удалению или модификации данных, даже к полному контролю над базой данных.

Для защиты от SQL-инъекций в приложениях на Python с использованием mysql-connector-python рекомендуется использовать параметризованные запросы и подготавливать SQL-запросы, используя специальные методы для экранирования символов и предотвращения внедрения вредоносного кода.

Неверно (параметры добавлены напрямую в строку запроса):



Мы будем использовать способ, при котором значения запроса передаются в качестве аргументов в .execute().



Теперь, опираясь на уже разработанные SQL-запросы (см. пп. «Разработка запросов к базе данных») мы можем написать процедуры обращения к БД и оперирования данными. Для примера приведём процедуры добавления пользователя в БД и создания резюме:



Все процедуры были отлажены и проверены на реальных данных с доступом через клиент mysql из командной строки WSL. Полный текст py-файла после данного этапа можно посмотреть по ссылке в истории git: <https://github.com/she-bear/dplm-resume-builder/blob/647a30760596edcec15d450a36291512846fa64f/experiments/backend/test_connection.py>

## Хэширование паролей в web-приложениях. Библиотека passlib

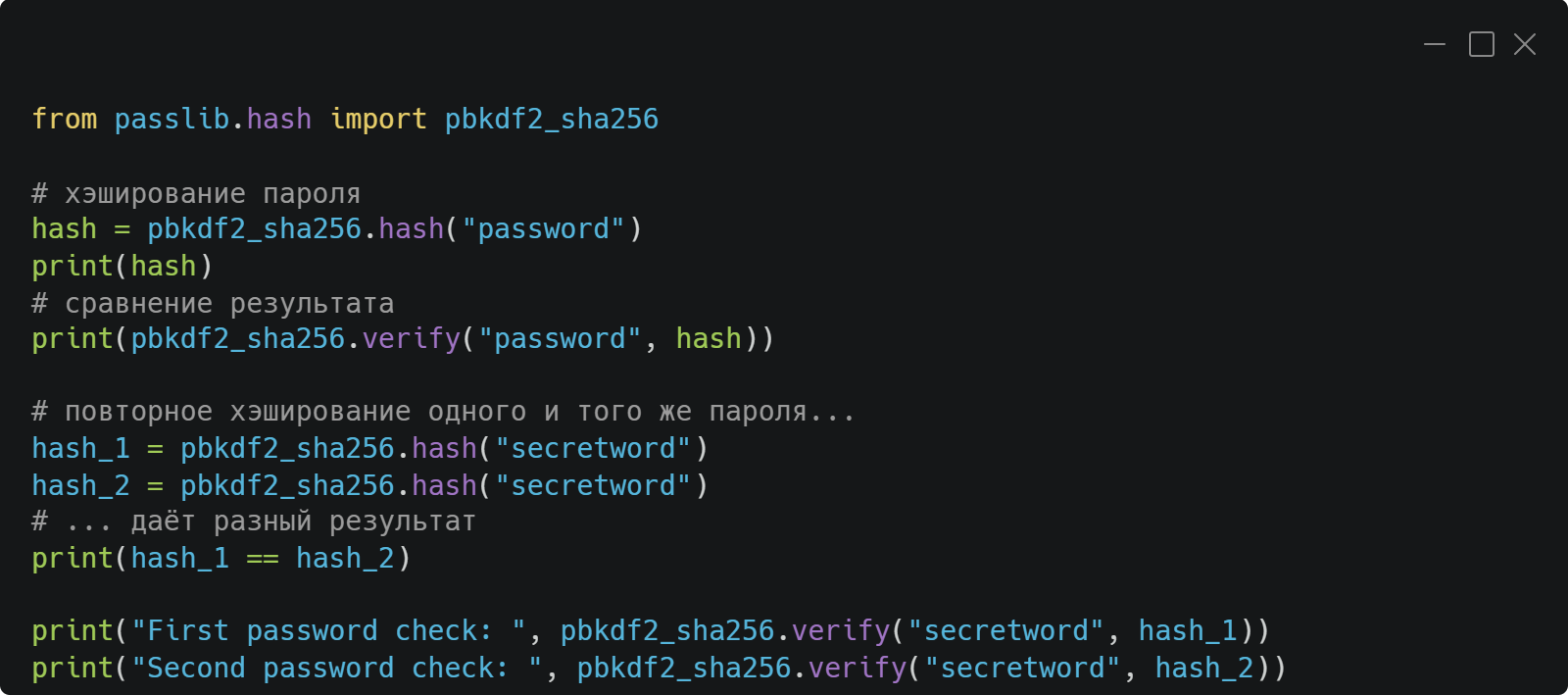
Для дальнейшей работы над проектом, необходимо решить задачу с хранением паролей пользователей. Для этого используется хэширование паролей.

Хэширование паролей в web-приложениях является процессом преобразования исходного пароля в уникальную строку фиксированной длины (хэш), которая не может быть обратно преобразована в исходный пароль. Это позволяет безопасно хранить пароли пользователей без возможности их прочтения в случае утечки данных.

Хэш сохраняется на сервере вместо самого пароля. Когда пользователь вводит свой пароль, он также преобразуется в хэш, который сравнивается с тем, что хранится на сервере. Если хэши совпадают, то пароль считается верным и пользователь получает доступ к системе.

Библиотека passlib в Python предоставляет удобные инструменты для хэширования и проверки паролей. С ее помощью можно создавать хеши паролей с использованием различных алгоритмов хэширования, таких как SHA256, bcrypt, PBKDF2 и другие.

Ниже приведен пример работы с библиотекой passlib. При запуске данного кода будет видно, что хэширование одного и того же пароля дает разный результат.



## Markdown

https://github.com/users/she-bear/projects/1/views/1?pane=issue&itemId=29855219

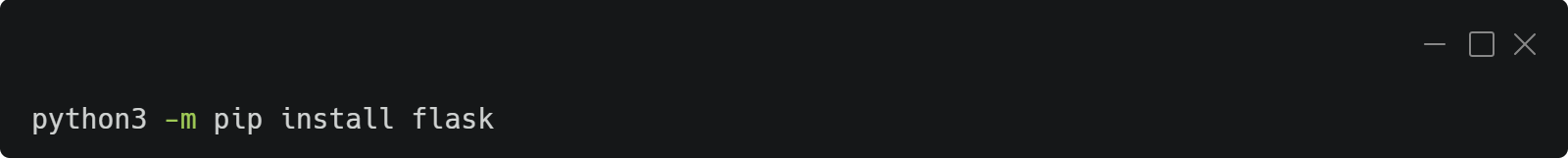
## Микрофреймфорк Flask

Следующим этапом работы будет изучение микрофреймворка Flask.

Flask – это микрофреймворк для создания веб-приложений на языке Python. Он обеспечивает базовые возможности для web-разработки, такие как маршрутизация, обработка запросов, ответ на них и другие.

Для начала установим микрофреймворк и научимся запускать py-приложения.

После активации виртуальной среды (про виртуальную среду, см. пп. Настройка среды VSCode), необходимо установить flask:



Запуск программы осуществляется командой:



Запуск в режиме отладки:



Простейшее приложение на flask выглядит следующим образом:

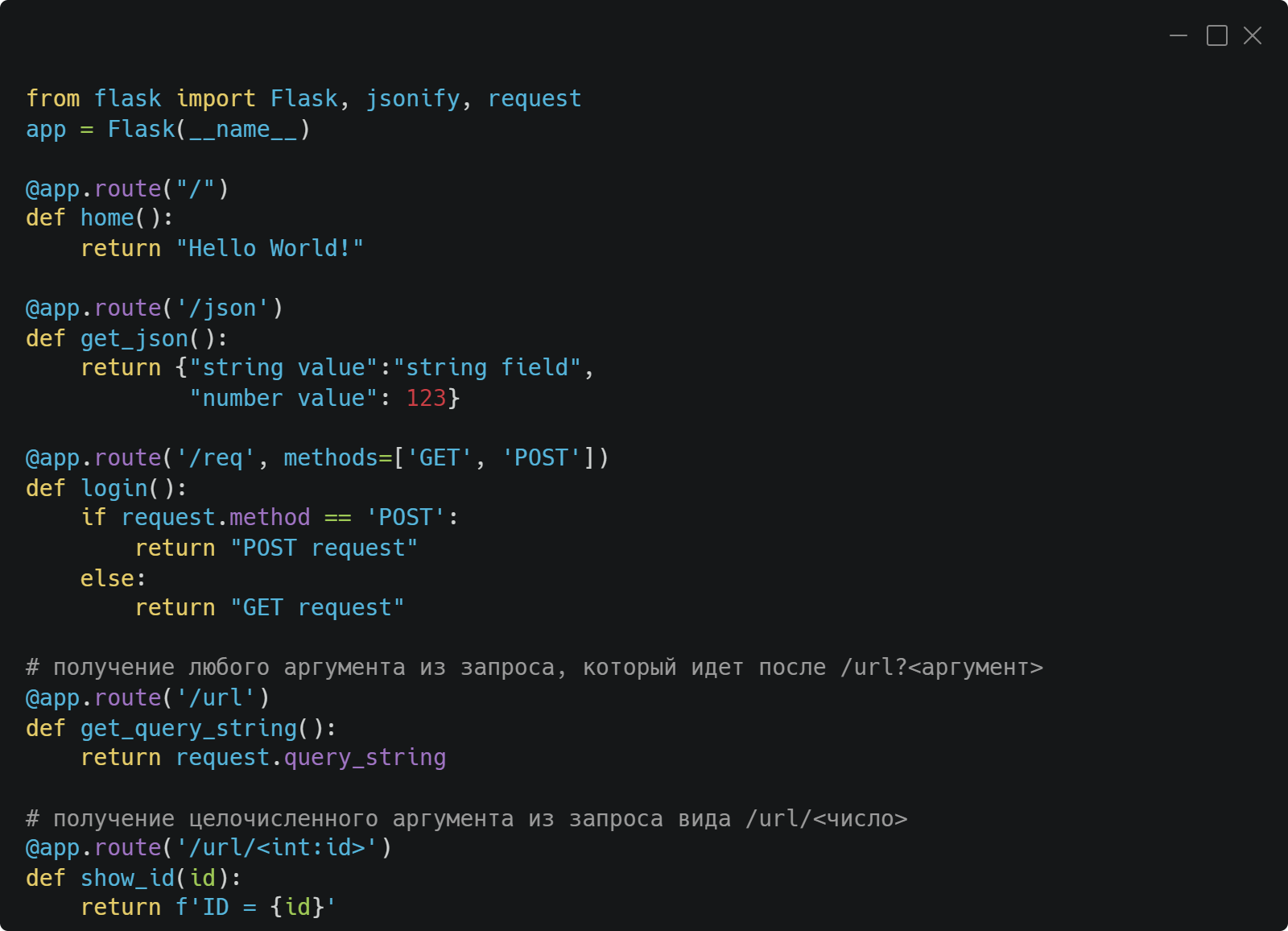


### Механизм маршрутизации в web-приложениях. Параметры HTTP-запроса

В Flask механизм маршрутизации (route) используется для определения того, как запросы клиентов должны быть обработаны web-приложением.

Механизм маршрутизации Flask позволяет указать функции, которые должны быть вызваны при получении определенного запроса HTTP (GET, POST, PUT, DELETE и т. д.) для определенного URL-адреса. Эти функции называются «обработчиками маршрутов» и обычно внутри себя возвращают результат для клиента.

Рассмотрим следующий код:



В данном примере у нас есть несколько маршрутов. При получении запроса на «/» будет вызвана функция home(), которая вернет в браузер строку «Hello, World!», маршрут «/json» показывает, как могут быть переданы параметры через нотацию json, в маршруте «/req» демонстрируется, как производится анализ типа запроса, а в маршрутах «/url» осуществляется передача параметров через строку запроса.

Этот файл можно посмотреть в git-истории: https://github.com/she-bear/dplm-resume-builder/blob/ac29158178735cd0e2c6e99df9c962021f1f6a69/experiments/backend/test\_flask.py

Рассмотрим HTTP-запросы более детально. Понимание HTTP-запросов, их устройства и назначения важно для web-разработчика – разработчик должен иметь возможность контролировать все аспекты работы web-страницы, включая взаимодействие с браузером, обмен данными (HTTP-запросы используются для отправки и получения данных между браузером и сервером) и безопасность (понимание XSS и внедрение SQL-кода – будут подробнее рассмотрены далее).

Не будем останавливаться на теоретических аспектах структуры HTTP-запроса. При разработке данного дипломного проекта важно, в первую очередь, разобраться, как именно осуществляется передача параметров в теле запроса.

Для тестирования различных методов передачи параметров было решено использовать программу HTTPie.

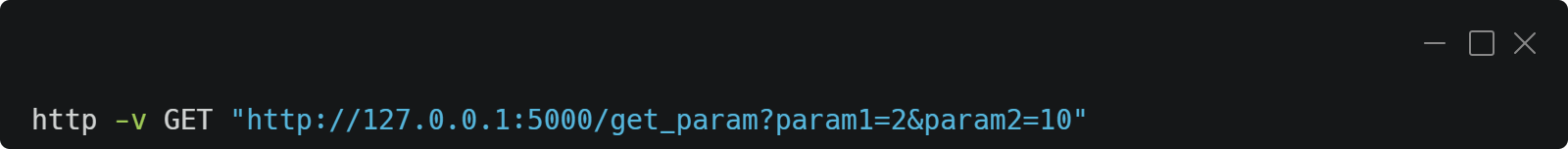
HTTPie – это удобная утилита командной строки, которая позволяет отправлять HTTP-запросы, используя простой синтаксис. Она облегчает выполнение различных HTTP-запросов и позволяет быстро проверять ответы сервера. С помощью HTTPie можно легко отправлять GET, POST, PUT, PATCH, DELETE и другие типы HTTP-запросов на любой URL, устанавливать заголовки запроса, отправлять данные формы, передавать JSON и многое другое. Также утилита поддерживает просмотр HTTP-заголовков ответа, форматирование JSON, авторизацию и другие функции.

Рассмотрим разные способы передачи на примере двух параметров.

Передача параметров через URL.

В HTTP-запросах параметры могут передаваться двумя способами: через путь (path) или через запрос (query). Значения параметров должны быть URL-encoded, чтобы избежать проблем с пробелами, специальными символами и т.д. URL-encoded заменяет специальные символы на соответствующие им escape-последовательности. Например, пробел заменяется на %20, а символ амперсанда (&) заменяется на %26.

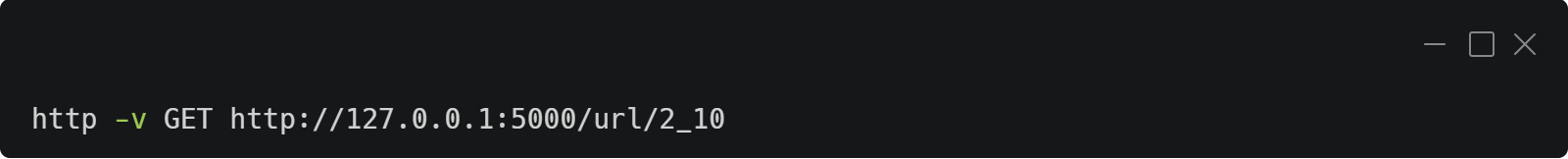
Тип передачи параметров: **query** используется для передачи данных в качестве строки запроса в URL-адресе. Параметры передаются в виде пар ключ=значение, разделенных амперсандом (&).



Из расшифровки запроса видно, что параметры ушли в URL (после GET):



Тип передачи параметров: **path** также используется для передачи данных в качестве строки запроса. Когда параметры передаются через путь, они являются частью URL после основной части домена. Например, в следующем примере, «2» и «10» являются параметрами, переданным через путь.



Из расшифровки запроса видно, что параметры также ушли в URL (после GET):



Передача параметров через post form.

Передача параметров через post-форму в HTML осуществляется с помощью полей формы. В теле формы указываются имена и значения параметров, которые нужно передать на сервер. При отправке формы браузер формирует запрос с параметрами в теле запроса.

Следует отличать поля *id* и *name* в HTML-элементе: они используются для идентификации и именования данного поля ввода или иного элемента:

* *id* – это уникальный идентификатор, который позволяет скриптам на стороне клиента или сервера обращаться к конкретному элементу ввода. Он используется для стилизации, скриптов на стороне клиента и других целей.
* *name* используется для передачи данных на сервер при отправке формы. Оно должно быть уникальным для каждого элемента ввода в форме и используется для получения данных на сервере.

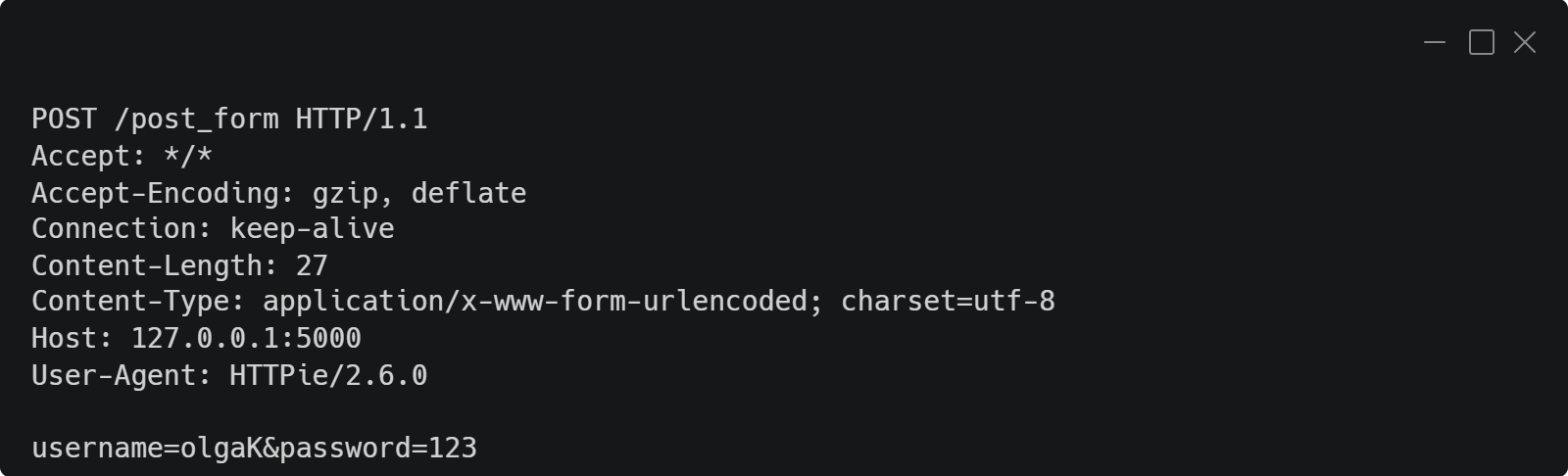
Например:



В HTTPie такие запросы отправляются с флагом -f:



Из расшифровки запроса видно, что параметры ушли в **ТЕЛЕ ЗАПРОСА** и **Content-Type = application/x-www-form-urlencoded**:



Передача параметров через post json

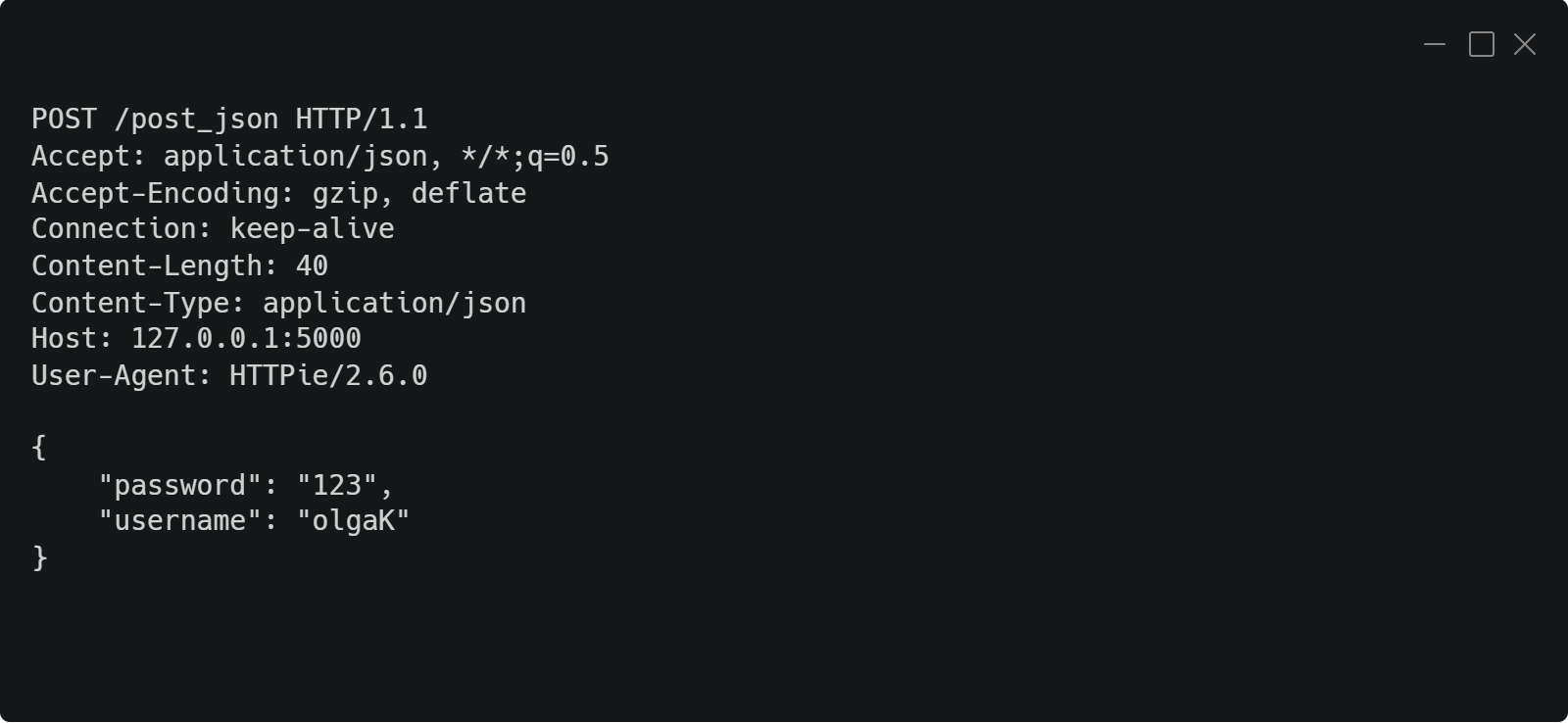
Передача JSON в HTML-запросе означает передачу данных в формате JavaScript Object Notation (JSON) через гипертекстовую передачу (HTTP) в качестве дополнительного параметра к запросу. Это может быть полезно для передачи структурированных данных, таких как объекты, списки или сложные структуры, между web-страницами или web-сервисами.

При использовании JSON в HTTP-запросах, данные передаются в виде строки, которая затем может быть проанализирована на стороне сервера или клиента с помощью соответствующего парсера JSON. Это позволяет обмениваться данными в формате, который легко читается и анализируется программами.

В HTTPie такие запросы отправляются с флагом -j:



Из расшифровки запроса видно, что параметры ушли в **ТЕЛЕ ЗАПРОСА** и **Content-Type = application/json**:



Важно иметь ввиду, что количество, тип и название параметров запроса ЗАРАНЕЕ фиксируется в документации на API. Например, если в последнем запросе одному из аргументов указать другое имя, то на стороне backend он не будет принят и проанализирован.

Демонстрация работы с запросами также была сделана на python, файл можно посмотреть в git-истории: https://github.com/she-bear/dplm-resume-builder/blob/71232472edd66a05edea5ba39808332ceaa0643e/experiments/backend/test\_req\_demo.py

### Аутентификация пользователя, модуль Flask-Login

Аутентификация пользователя в web-приложении играет решающую роль в обеспечении безопасности и конфиденциальности данных. Роль аутентификации заключается в проверке подлинности пользователей, их идентификации и установлении прав доступа к конкретным ресурсам web-приложения.

Когда пользователь пытается получить доступ к web-приложению, первым шагом является процесс аутентификации. Идентификационные данные, такие как логин и пароль, вводятся пользователем и передаются на сервер для проверки. В случае успешной аутентификации, пользователь получает доступ к своему аккаунту и разрешение на выполнение определенных операций.

Также важной частью аутентификации пользователя является управление сеансами пользователей, их вход и автоматический выход из системы после определенного периода неактивности.

В данном проекте для аутентификации пользователей будет использоваться модуль Flask-Login микрофреймворка Flask. Flask-Login помогает осуществить процесс аутентификации, управления сеансами пользователей (сохранение информации в cookie) и устанавливает различные права доступа для пользователей.

В нижеприведенном коде был разработан класс UserSession, осуществляющий аутентификацию и авторизацию пользователей.

Рассмотрим подробнее каждый route.

Обработка запроса на route **/login**:

* Выполняется проверка данных пользователя – например, соответствие логина и пароля.
* Если валидация прошла успешно, то создается объект UserSession(). Этот объект может хранить не только информацию об аутентификации, но и прочие данные пользователя (полное имя, фото, номер счета и т.п.).



* Вызов flask-login-функции login\_user – сохраняет информацию о пользователе в сессии и устанавливает переменную user\_id, которая может быть использована другими частями приложения для проверки, что пользователь вошел в систему.

Обработка запроса на route **/status**:

* flask-login после проверки cookies (есть такая сессия или нет) создаст переменную current\_user,
* затем будет создан и заполнен объект UserSession и выполнена проверка is\_authenticated – авторизован пользователь или нет.

Объект UserSession содержит не только те свойства и методы, которые реализованы в примере. Этот объект наследуется от UserMixin, таким образом, мы получаем доступ к различным методам и атрибутам, которые связаны с аутентификацией и авторизацией пользователя. Например, можно получить доступ к методам is\_authenticated() и is\_active(), которые проверяют, аутентифицирован ли пользователь и активен ли его аккаунт.

Обработка запроса на route **/logout**:

* Будет вызвана функция logout\_user. Эта функция используется для выхода текущего пользователя из системы. Она очищает информацию о пользователе из сессии.

Корректная работа модуля аутентификации проверялась с помощью программы HTTPie и подробно описана в файле readme.md папки experiments/backend (раздел Flask-Login):

<https://github.com/she-bear/dplm-resume-builder/blob/71232472edd66a05edea5ba39808332ceaa0643e/experiments/backend/readme.md>

В этом же файле приведена схема, поясняющая механизм работы данного модуля.

## Уязвимости: Cross-site scripting (XSS) и SQL-инъекция

## Выводы

Был проделан путь от простой команды создания docker-контейнера MySQL до команды, которая учитывает все особенности проектируемой состемы:

|  |
| --- |
|  |

# Развитие проекта

1. Добавление функциональности для роли “Администратор сайта”:
   1. реализация каскадного удаления
2. Масштабирование системы

# Список литературы

1. Документация по GitHub: <https://docs.github.com/ru>
2. Документация по GitHub Projects: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects
3. Работа с VSCode, возможности, основные понятия: <https://code.visualstudio.com/docs>
4. Документация по Notion: https://www.notion.so/help/guides/category/documentation
5. Архитектура системы: понятие, виды: <https://gb.ru/blog/arhitektuta-sistemy/>
6. User Stories: <https://www.visual-paradigm.com/learning/handbooks/agile-handbook/user-story.jsp>
7. Пользовательские истории с примерами и шаблоном: [https://www.atlassian.com/ru/agile/project-management/user-stories#](https://www.atlassian.com/ru/agile/project-management/user-stories)
8. Модель «сущность-связь»: <https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model>
9. Запуск mysql в docker: <https://hub.docker.com/_/mysql>
10. Установка переменных среды: <https://docs.docker.com/engine/reference/run/#env-environment-variables>
11. Список опций для docker run: <https://docs.docker.com/engine/reference/commandline/run/>
12. Хранение данных MySQL в docker-контейнере: <https://hub.docker.com/_/mysql>
13. Работа с MySQL Workbench: <https://www.mysql.com/products/workbench/>
14. Копирование SQL-файла внутрь контейнера: <https://stackoverflow.com/questions/14684063/mysql-source-error-2>
15. Вставка данных в таблицы с полями AUTO\_INCREMENT: <https://stackoverflow.com/questions/8753371/how-to-insert-data-to-mysql-with-auto-incremented-columnfield>
16. Использование LAST\_INSERT\_ID: <https://stackoverflow.com/questions/15821532/get-current-auto-increment-value-for-any-table/15821655#15821655>
17. Установка и настройка виртуального окружения для Python в VS Code: <https://learn.microsoft.com/ru-ru/windows/python/web-frameworks>
18. Работа с os.environ object в python: <https://www.geeksforgeeks.org/python-os-environ-object/>, https://docs.python.org/3/library/os.html#os.getenv
19. Про SQL-инъекции: <https://realpython.com/prevent-python-sql-injection/>
20. Библиотека passlib: <https://passlib.readthedocs.io/en/stable/#>
21. Flask: начало работы: <https://learn.microsoft.com/ru-ru/windows/python/web-frameworks#hello-world-tutorial-for-flask>
22. Документация по flask: <https://flask.palletsprojects.com/en/3.0.x/>
23. Минимальное приложение на flask: <https://flask.palletsprojects.com/en/2.3.x/quickstart/#a-minimal-application>
24. Получение данных формы во flask: <https://stackoverflow.com/questions/10434599/get-the-data-received-in-a-flask-request>
25. Параметры строки запроса во flask: <https://stackabuse.com/get-request-query-parameters-with-flask/>
26. Принцип работы форм: <https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data>
27. Flask, API с json: <https://flask.palletsprojects.com/en/2.3.x/quickstart/#apis-with-json>
28. Работа с HTTPie: <https://httpie.io/>
29. HTTPie, GET и POST-запросы: <https://httpie.io/docs/cli/optional-get-and-post>
30. HTTPie, отправка данных форм: <https://httpie.io/docs/cli/regular-forms>
31. HTTPie, работа с сессиями: <https://httpie.io/docs/cli/sessions>
32. Алгоритм процентного кодирования: <https://en.wikipedia.org/wiki/Percent-encoding>
33. Flask-Login: https://flask-login.readthedocs.io/en/latest/