МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
Московский технический университет связи и информатики

Учебное пособие
«Введение в информационные технологии»

для студентов направлений
09.03.04 «Программная инженерия»

Москва 2025

Оглавление

Лабораторная работа №1: Работа с структурами данных, циклами и условными операторами в Python

Цель работы: Освоить работу со структурами данных, циклами и условными операторами в языке программирования Python.

1. Работа со структурами данных

Для создания списка чисел от 1 до 10 следует использовать следующий код:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Для вывода на экран первого, третьего и последнего элементов списка можно использовать следующий код:

```
print(my_list[0])    # Вывод первого элемента
print(my_list[2])    # Вывод третьего элемента
print(my_list[-1])   # Вывод последнего элемента
```

Для замены второго элемента списка на значение 100 используйте следующий код:

```
my_list[1] = 100     # Замена второго элемента
```

2. Работа с циклами

Цикл for: Для вывода на экран всех чисел от 1 до 10 следует использовать цикл for:

```
for i in range(1, 11):
    print(i)
```

Цикл while: Для вывода на экран всех чисел от 10 до 1 с помощью цикла while используйте следующий код:

```
i = 10
while i >= 1:
    print(i)
    i -= 1
```

3. Работа с условными операторами

Условный оператор if: Для проверки четности введенного пользователем числа и вывода соответствующего сообщения используйте следующий код:

```
num = int(input("Введите число: "))
if num % 2 == 0:
    print("Число четное")
else:
    print("Число нечетное")
```

Условный оператор if-else: Для проверки положительного или отрицательного числа используйте следующий код:

```
num = int(input("Введите число: "))
if num > 0:
    print("Число положительное")
elif num < 0:
    print("Число отрицательное")
else:
    print("Число равно нулю")
```

4. Домашнее задание

Задача 1:

Напишите программу, которая запрашивает у пользователя ввод числа и выводит на экран все числа от 1 до введенного числа включительно.

Пример:

```
Введите число: 5
1
2
3
4
5
```

Задача 2:

Напишите программу, которая запрашивает у пользователя ввод 2 чисел и выводит на экран большее из них.

Пример:

```
Введите первое число: 10
Введите второе число: 5
Большее число: 10
```

Лабораторная работа №2: Функции в Python и базовые алгоритмы

Цель работы: Освоить принципы определения и использования функций в языке программирования Python, понять механизмы передачи аргументов в функции, научиться применять функции для решения практических задач, а также изучить базовые алгоритмические конструкции.

Задание 1: Написание простых функций

Напишите функцию **greet**, которая принимает имя пользователя в качестве аргумента и выводит приветствие с этим именем.

```
def greet(name):
    #Напишите тело функции
```

Создайте функцию **square**, которая возвращает квадрат переданного ей числа.

```
def square(number):
    #Напишите тело функции
```

Реализуйте функцию **max_of_two**, которая принимает два числа в качестве аргументов и возвращает большее из них.

```
def max_of_two(x, y):
    #Напишите тело функции
```

Цель работы: Освоить принципы определения и использования функций в языке программирования Python, понять механизмы передачи аргументов в функции, научиться применять функции для решения практических задач, а также изучить базовые алгоритмические конструкции.

Задание 2: Работа с аргументами функций

Напишите функцию **describe_person**, принимающую имя и возраст человека, и печатающую эту информацию в читаемом виде. Сделайте возраст опциональным аргументом со значением по умолчанию 30.

```
def describe_person(name, age=30):
    #Напишите тело функции
```

Задание 3: Использование функций для решения алгоритмических задач

Напишите функцию **is_prime**, которая определяет, является ли число простым, и возвращает **True** или **False** соответственно.

```
def is_prime(number):
   #Напишите тело функции
```

Лабораторная работа №3: Работа с файлами в Python: открытие, чтение, запись, работа с исключениями

Цель работы: Освоить принципы определения и использования функций в языке программирования Python, понять механизмы передачи аргументов в функции, научиться применять функции для решения практических задач, а также изучить базовые алгоритмические конструкции.

Задание 1: Открытие и чтение файла

1. Создайте текстовый файл example.txt и заполните его несколькими строками текста.
2. Напишите функцию на Python, которая открывает файл example.txt в режиме чтения и выводит его содержимое на экран.
3. Используйте разные методы чтения файла: чтение всего файла сразу, построчное чтение, реализуйте выбор типа чтения в принимаемых аргументах функции.

```
#Чтение всего файла
with open('example.txt', 'r') as file:
    content = file.read()
#Построчное чтение
with open('example.txt', 'r') as file:
    for line in file:
```

Задание 2: Запись в файл

1. Напишите программу, которая запрашивает у пользователя текст и записывает его в новый файл user_input.txt.
2. Реализуйте функционал добавления текста в существующий файл, не удаляя его предыдущее содержимое.

Задание 3: Запись в файл

1. Модифицируйте программу из Задания 1 так, чтобы она корректно обрабатывала исключение, возникающее при попытке открыть несуществующий файл. Вместо вывода ошибки программа должна выводить пользователю понятное сообщение.

Используйте в блоке try except следующий класс исключений: **FileNotFoundError.**

Лабораторная работа №4: Модули и пакеты: импорт, создание, использование

Цель работы: Понять, как импортировать модули и пакеты в Python, научиться создавать собственные модули и пакеты, изучить способы использования модулей и пакетов для структурирования программы.

Задание 1: Импорт стандартных модулей

1. Импортируйте модуль **math** и используйте функцию **sqrt()** для вычисления квадратного корня.
2. Используйте модуль datetime для отображения текущей даты и времени.

Задание 2: Создание и использование собственного модуля

1. Создайте модуль **my_module.py**, который содержит минимум одну функцию. Например, функция может принимать два аргумента и возвращать их сумму.
2. Импортируйте **my_module** в другой файл Python и вызовите функцию, определённую в модуле.

Задание 3: Создание и использование пакетов

1. Создайте пакет, содержащий несколько модулей. Каждый модуль должен выполнять определённую задачу (например, операции с числами, работа со строками и т.д.).
2. Продемонстрируйте, как импортировать различные модули из вашего пакета в другой файл Python.

Лабораторная работа № 5 Работа с классами

Цель работы: Получить практический опыт работы с ООП в Python.

Задание 1: Базовый класс и методы

1. Определите класс Book, который имеет три атрибута: **title** (название), **author** (автор), и **year** (год издания).
2. Добавьте метод get_info(), который возвращает информацию о книге в формате: "Название книги: [**title**], Автор: [**author**], Год издания: [**year**]".

Задание 2: Работа с конструктором

1. Определите класс **Circle** для представления круга.
2. Используйте конструктор **__init__** для инициализации радиуса круга (**radius**).
3. Добавьте метод **get_radius()**, который возвращает значение радиуса.
4. Добавьте метод **set_radius(new_radius)**, который позволяет изменить радиус круга.
5. Создайте объект класса **Circle**, измените его радиус и выведите новый радиус на экран

Лабораторная работа № 6 Работа с классами ч.2

Цель работы: Получить практический опыт работы с ООП в Python. использование инкапсуляции, наследования.

Задание 1: Защита данных пользователя

1. Создайте класс **UserAccount**, который представляет аккаунт пользователя с атрибутами: имя пользователя (**username**), электронная почта (**email**) и приватный атрибут пароль (**password**).
2. Используйте конструктор **__init__** для инициализации этих атрибутов.
3. Реализуйте метод **set_password**(**new_password**), который позволяет безопасно изменить пароль аккаунта.
4. Реализуйте метод **check_password**(**password**), который проверяет, соответствует ли введённый пароль текущему паролю аккаунта и возвращает **True** или **False**.
5. Создайте объект класса **UserAccount**, попробуйте изменить пароль и проверить его с помощью методов **set_password** и **check_password**.

Задание 2: Полиморфизм и наследование

1. Определите базовый класс **Vehicle** с атрибутами: **make** (марка) и **model** (модель), а также методом **get_info**(), который возвращает информацию о транспортном средстве.
2. Создайте класс **Car**, наследующий от **Vehicle**, и добавьте в него атрибут **fuel_type** (тип топлива). Переопределите метод **get_info**() таким образом, чтобы он включал информацию о типе топлива.

Лабораторная работа №7 Работа с классами ч.3

Цель работы: Разработать систему управления сотрудниками, демонстрирующую множественное наследование, инкапсуляцию и полиморфизм в Python. Система должна уметь обрабатывать различные типы сотрудников, включая менеджеров и технических специалистов, а также предоставлять возможность для расширения и добавления новых ролей.

Задание

1. Создайте класс Employee с общими атрибутами, такими как **name** (имя), **id** (идентификационный номер) и методами, например, **get_info**(), который возвращает базовую информацию о сотруднике.
2. Создайте класс **Manager** с дополнительными атрибутами, такими как **department** (отдел) и методами, например, **manage_project**(), символизирующим управление проектами.
3. Создайте класс **Technician** с уникальными атрибутами, такими как **specialization** (специализация), и методами, например, **perform_maintenance**(), означающим выполнение технического обслуживания.
4. Создайте класс **TechManager**, который наследует как Manager, так и Technician. Этот класс должен комбинировать управленческие способности и технические навыки, например, иметь методы для управления проектами и выполнения технического обслуживания.
5. Добавьте метод **add_employee**(), который позволяет **TechManager** добавлять сотрудников в список подчинённых.
6. Реализуйте метод **get_team_info**(), который выводит информацию о всех подчинённых сотрудниках.

7. Создайте объекты каждого класса и демонстрируйте их функциональность.

Лабораторная работа №8: Установка WSL и выполнение базовых команд

Цель работы: Понять, как импортировать модули и пакеты в Python, научиться создавать собственные модули и пак0еты, изучить способы использования модулей и пакетов для структурирования программы.

Задание 1: Установка WSL

1. Открытие PowerShell с правами администратора: Нажмите **Win+X** и выберите **'Windows PowerShell (Администратор)'.**
2. Включение возможности WSL: В PowerShell введите и выполните следующую команду:

```
dism.exe /online /enable-feature
/featurename:Microsoft-Windows-Subsystem-Linux /all
/norestart
```

3. Включение 'Виртуальной машины' (требуется для WSL 2): Введите и выполните следующую команду:

```
dism.exe /online /enable-feature
/featurename:VirtualMachinePlatform /all /norestart
```

4. Перезагрузка компьютера.

5. Скачивание и установка пакета обновления ядра Linux для WSL 2 (только для WSL 2). Ссылку на скачивание можно найти на официальном сайте Microsoft.
6. Установка дистрибутива Linux из Microsoft Store: Откройте Microsoft Store, найдите предпочитаемый дистрибутив Linux (например, Ubuntu) и нажмите 'Установить'.
7. Запуск установленного дистрибутива Linux после установки для завершения настройки, включая создание пользователя и пароля.

Задание 2: Создание и использование собственного модуля

1. Откройте терминал WSL. Создайте новую директорию в вашем домашнем каталоге с именем **LabWork**.

```
mkdir ~/LabWork
```

2. Внутри созданной директории LabWork создайте текстовый файл с именем **example.txt**.

```
cd ~/LabWork
```

3. Используйте команду **echo** для добавления текста в файл: **Hello, World!**

```
echo "Hello, World!" > example.txt
```

4. Скопируйте файл **example.txt** в ту же директорию, но с новым именем **copy_example.txt.**

```
cp example.txt copy_example.txt
```

5. Переименуйте файл copy_example.txt в renamed_example.txt.

```
mv copy_example.txt renamed_example.txt
```

6. Удалите файл renamed_example.txt.

```
rm renamed_example.txt
```

Контрольные вопросы для самопроверки:

1. Какая команда используется для создания новой директории?
2. Как можно создать файл и сразу записать в него текст?
3. Какая команда позволяет скопировать файл?
4. Как переименовать файл?
5. Как удалить файл?

Лабораторная работа №9: Практическая работа с Git

Цель работы: познакомить студентов с основными возможностями системы управления версиями Git и научить их выполнять основные операции с помощью этой системы.

Задание 1: Установить Git на свой компьютер:

1. Скачайте установочный пакет Git введя в командной строке команду: apt-get install git
2. Запустите установочный пакет и следуйте инструкциям по установке Git на свой компьютер.

Задание 2: Создать новый репозиторий на GitHub или GitLab:

1. Зайти на свой аккаунт на GitHub или GitLab.
2. Нажать кнопку "Create repository" (Создать репозиторий).
3. Заполнить имя репозитория, описание и выбрать опции создания (например, добавить README файл).

4. Нажать "Create repository" (Создать репозиторий).

Задание 3: Клонировать репозиторий на локальный компьютер:

1. Скопировать URL репозитория с GitHub или GitLab.
2. Открыть командную строку (или Git Bash) на локальном компьютере.
3. Ввести команду git clone <URL репозитория> чтобы склонировать репозиторий на локальный компьютер.

Задание 4: Создать новый файл в репозитории:

1. Перейти в каталог, в который был склонирован репозиторий.
2. Создать новый файл (например, touch new_file.txt для Unix-подобных систем или echo.>new_file.txt для Windows)

Задание 5: Добавить новый файл в индекс и произвести коммит изменения:

1. В командной строке выполнить команду git add new_file.txt чтобы добавить файл в индекс.
2. Выполнить команду git commit -m "Добавлен новый файл" чтобы закоммитить добавленный файл с комментарием "Добавлен новый файл".

Лабораторная работа №10: Практическая работа с Git

Цель работы: Познакомить студентов с основными операциями работы с удаленными репозиториями в Git, включая клонирование, добавление удаленных репозиториев, отправку изменений и работу с ветками на удаленном репозитории.

Задание 1: Клонировать удаленный репозиторий::

1. Найти удаленный репозиторий на GitHub, GitLab или другой платформе.
2. На локальном компьютере выполнить команду git clone <URL удаленного репозитория>.
3. Убедиться, что репозиторий успешно склонирован на локальный компьютер.

Задание 2: Добавить удаленный репозиторий:

1. В командной строке выполнить команду git remote add origin <URL удаленного репозитория> для добавления удаленного репозитория в локальный.

Задание 3: Отправить изменения на удаленный репозиторий:

1. Выполнить команду git push origin <branch_name> для отправки изменений из локального репозитория на удаленный.
2. Проверить, что изменения успешно отправлены на удаленный репозиторий.

Задание 4: Работа с ветками на удаленном репозитории:

1. Создать новую ветку локально с помощью команды git checkout -b <branch_name>.
2. Запушить новую ветку на удаленный репозиторий с помощью команды git push origin <branch_name>.
3. Удалить ветку из удаленного репозитория с помощью команды git push origin --delete <branch_name>.

Задание 5: Получение изменений из удаленного репозитория::

1. Выполнить команду git pull origin <branch_name> для получения изменений из удаленного репозитория на локальный.
2. Убедиться, что изменения успешно применены к локальному репозиторию.

Вопросы для самопроверки

1. Что такое система контроля версий и для чего она используется?
2. Какие основные преимущества использования Git перед другими системами контроля версий?
3. Что такое команда "git clone" и как она используется?

4. Какая команда используется для добавления файлов в индекс в Git?

5. Как создать новую ветку в Git и что означает "ветвление" в контексте системы контроля версий?

6. Что означает команда "git push" и как она используется для отправки изменений на удаленный репозиторий?

7. Как можно получить изменения из удаленного репозитория на локальный с помощью Git?

8. Что такое merge (слияние) в Git и какое его предназначение?

9. Какое предназначение у команды "git pull"?

10. Какая команда используется для удаления ветки из удаленного репозитория в Git?

Лабораторная работа 11: Основы SQL

1. Создание таблиц

Для выполнения работы воспользуемся сайтом https://sunnygoyal.com/jade/ .

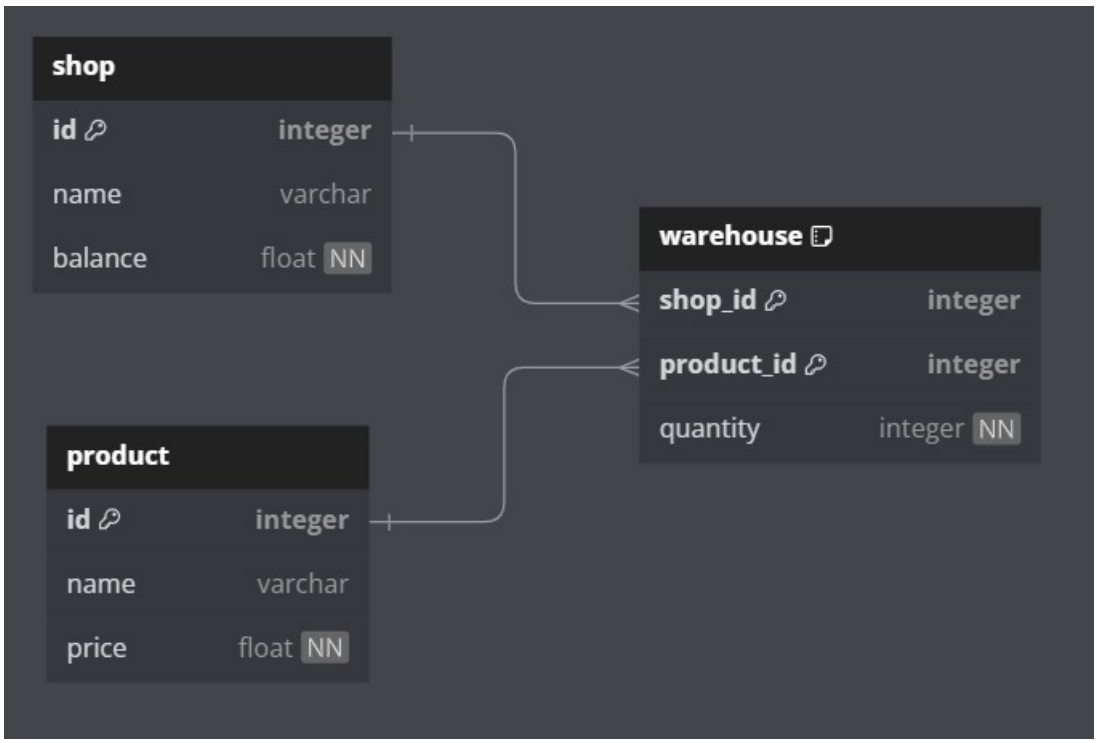


Рисунок 1 – пример схемы базы данных

В качестве примера будем рассматривать создание базы данных для сети магазинов. Создадим таблицы на основе рисунка 1. Для создания требуемых таблиц воспользуемся скриптом ниже.

Листинг 1 – создание таблиц БД

```
CREATE TABLE shop (
id INTEGER PRIMARY KEY,
name VARCHAR(255) UNIQUE,
balance FLOAT NOT NULL);

CREATE TABLE product (
id INTEGER PRIMARY KEY,
name VARCHAR(255) UNIQUE,
price FLOAT NOT NULL);

CREATE TABLE warehouse (
shop_id INTEGER REFERENCES shop (id),
product_id INTEGER REFERENCES product (id),
quantity INTEGER NOT NULL,
PRIMARY KEY (shop_id, product_id));
```

Рассмотрим подробнее листинг 1. Для создания таблицы используется команда CREATE TABLE после чего указывается название таблицы. В скобках производится перечисление полей таблицы. В листинге использовались следующие типы данных INTEGER – целочисленный тип данных, FLOAT – числа с плавающей точкой и VARCHAR – строка переменной длины с указанием предела по количеству символов. Для создания ограничения на незаполненность используется ключевое слово NOT NULL при объявлении переменной. Обозначение первичного ключа осуществляется ключевыми словами PRIMARY KEY в поле, которое должно обладать данным свойством, или в конце отдельной строкой с перечислением названия полей в скобках. Создание внешнего ключа осуществляется при помощи ключевого слова REFERENCES, дальше указывается таблица и поле, на которое идет ссылка. Ключевое слово UNIQUE отвечает за уникальность значения в таблице

2. Внесение данных

Внесем данные в ранее созданные таблицы.

Листинг 2 – внесение данных в таблицу shop

```
INSERT INTO shop (id, name, balance) VALUES (1,
'пятерочка',31);
INSERT INTO shop (id, name, balance) VALUES (2,
'перекресток',133);
```

Листинг 3 – внесение данных в таблицу product

```
INSERT INTO product VALUES (1, 'молоко', 100);
INSERT INTO product VALUES (2, 'хлеб', 25);
INSERT INTO product VALUES (3, 'хлеб', 30);
```

Листинг 4 – внесение данных в таблицу warehouse

```
INSERT INTO warehouse VALUES (1, 1, 20);
INSERT INTO warehouse VALUES (1, 2, 10);
INSERT INTO warehouse VALUES (2, 1, 30);
```

Для внесения данных используются ключевые слова INSERT INTO, далее указывается название таблицы и опционально поля, которые будут заполняться, после ключевого слова VALUES в скобках через запятую идет перечисление значений, которые мы хотим занести в таблицу.

3. Создание запросов

Листинг 5 – Запрос на получение данных таблицы shop

```
SELECT * FROM shop;
```

Для проверки работы выше выполним запрос с листинга 5. Для запросов на получение информации используется слово SELECT, далее указываются требуемые для вывода поля или * в случае, когда необходимо выбрать все

поля. После ключевого слова FROM указывается таблица, к которой мы обращаемся.

Листинг 6 – Запрос на получение данных таблицы product.

```
SELECT * FROM product;
```

Пример запроса с указанием полей показан на листинге 6. Отметим, что в результате мы видим всего 2 строки, хотя и вводили 3. Здесь показывается работы ограничения UNIQUE на поле product.name

Листинг 7 – Запрос на получение данных таблицы warehouse с фильтрацией.

```
SELECT * FROM warehouse
WHERE shop_id = 1;
```

В листинге 7 показывается выполнение запроса с фильтрацией. Для этого используется ключевое слово WHERE, после чего указывается поле, по которому будут фильтровать данные, и значение, которое будет искать в строках таблицы.

4. Практическое задание

- Создать таблицы на основе рисунка 2. Создание должно производится через скрипты.
- Внесите в каждую таблицу данные.
- К каждой таблице напишите запросы с фильтрацией. Результаты фильтрации отобразить в виде скриншотов и скачать файл итоговой работы.

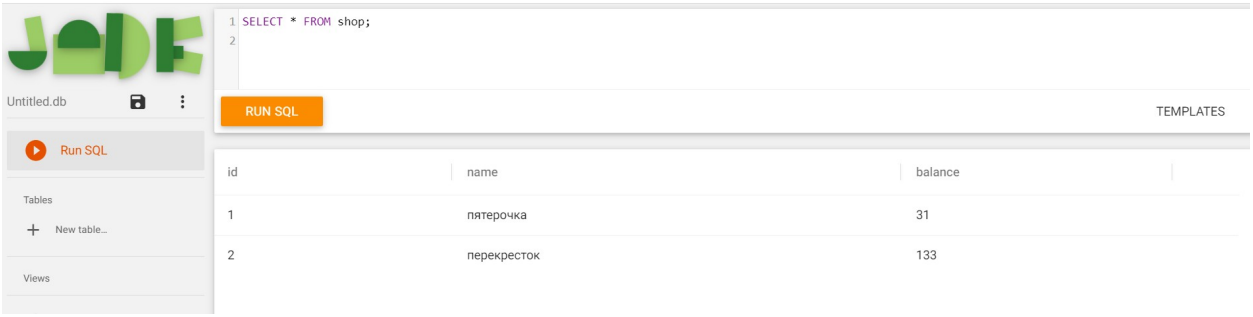


Рисунок 2 – пример скриншота

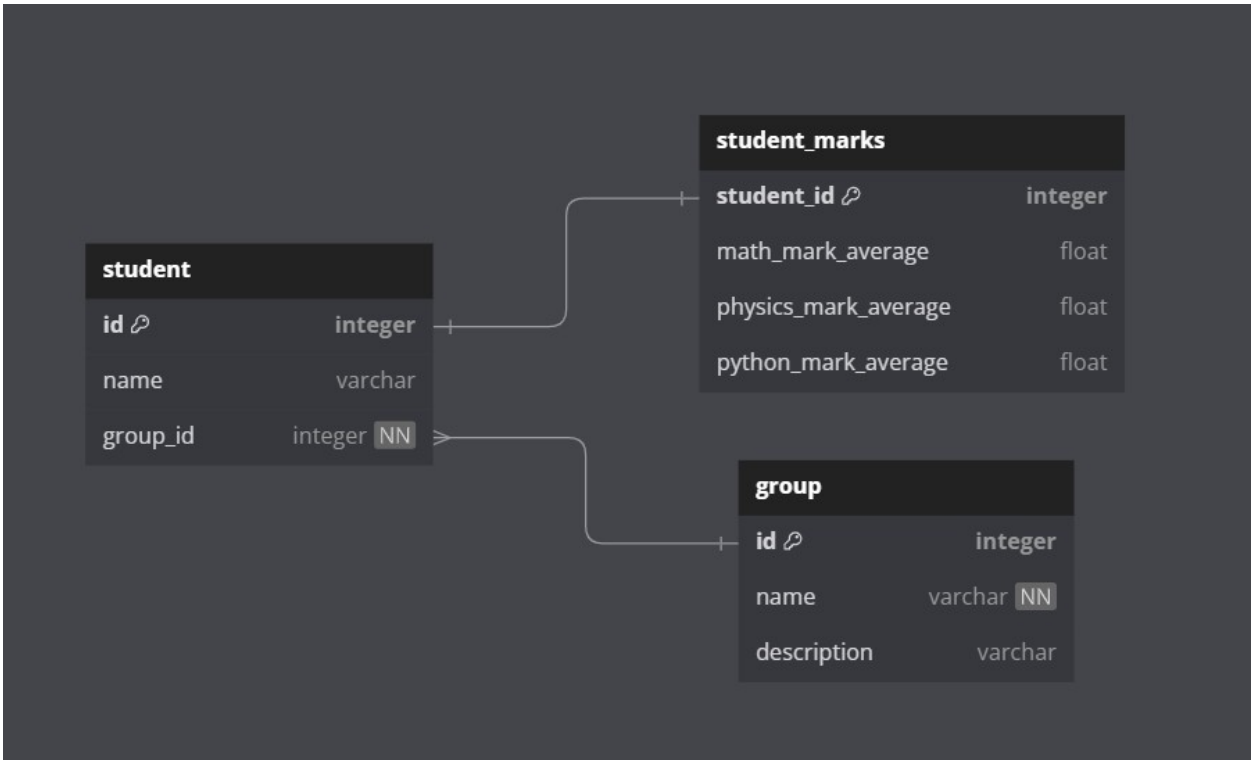


Рисунок 3 – схема БД

Лабораторная работа 12: Основы выборки SQL

1. Для выполнения работы воспользуйтесь схемой БД из лабораторной работы №5

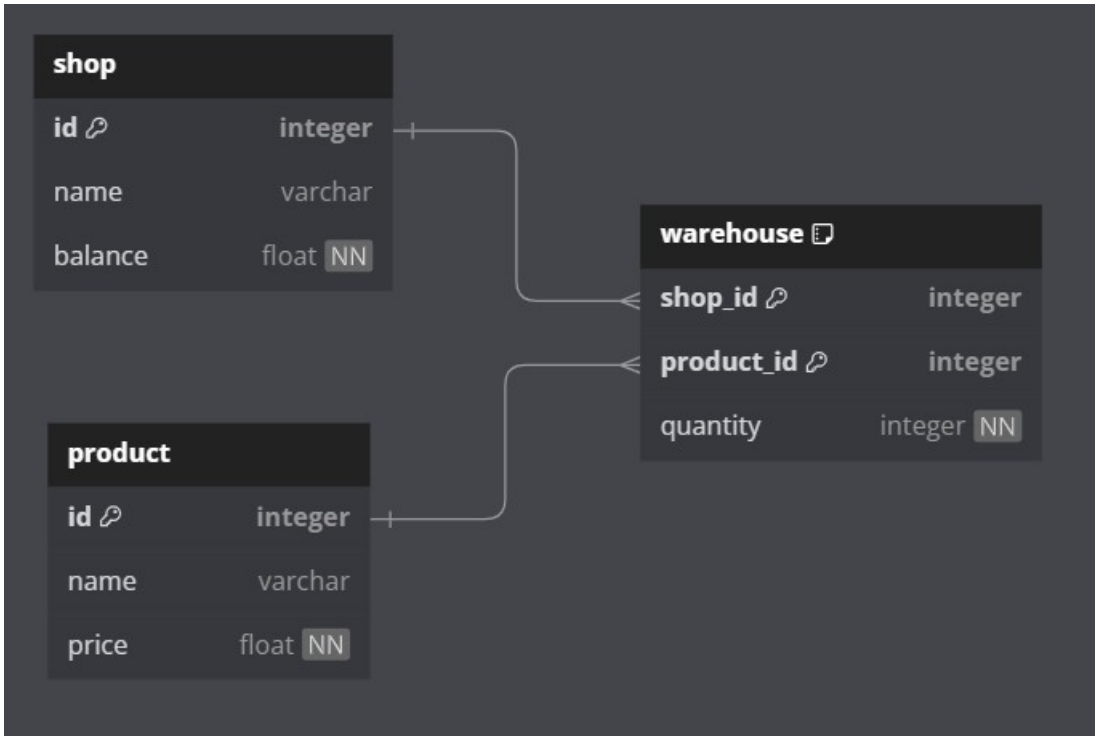


Рисунок 1 – Схемы базы данных

Сортировка, оператор ORDER BY

При выполнении SELECT запроса, строки по умолчанию возвращаются в неопределённом порядке. Фактический порядок строк в этом случае зависит от плана соединения и сканирования, а также от порядка расположения данных на диске, поэтому полагаться на него нельзя. Для упорядочивания записей используется конструкция ORDER BY

```
SELECT поля_таблиц FROM наименование_таблицы
WHERE ...
ORDER BY столбец_1 [ASC | DESC][, столбец_n [ASC |
DESC]]
```

Где ASC и DESC - направление сортировки:

- ASC - сортировка по возрастанию (по умолчанию)
- DESC - сортировка по убыванию

Для сортировки результатов по двум или более столбцам их следует указывать через запятую.

```
...ORDER BY столбец_1 [ASC | DESC], столбец_2 [ASC | DESC];
```

Группировка, оператор GROUP BY

Оператор GROUP BY в SQL используется для группировки строк из таблицы в более крупные наборы результатов на основании одного или нескольких столбцов. Эти группы включают разные записи в таблице и, соответственно, обладают разными характеристиками, которые нам могут быть весьма полезны.

Общая структура запроса с GROUP BY выглядит следующим образом

```
SELECT [литералы, агрегатные_функции, поля_группировки]
FROM имя_таблицы
GROUP BY поля_группировки;
```

При использовании оператора GROUP BY мы перешли от работы с отдельными записями на работу с образовавшимися группами. В связи с этим мы не можем просто вывести любое поле из записи , как мы это могли делать

раньше. Так как в каждой группе может быть несколько записей и в каждой из них в этом поле может быть разное значение.

При использовании GROUP BY мы можем выводить только:

- литералы, т.е. указанное явным образом фиксированные значения
- результаты агрегатных функций, т.е. вычисленные значения на основании набора значений.

Агрегатные функции

Агрегатная функция – это функция, которая выполняет вычисление на наборе значений и возвращает одиночное значение.

Общая структура запроса с агрегатной функцией выглядит следующим образом:

```
SELECT [литералы, агрегатные_функции,
поля_группировки]
FROM имя_таблицы
GROUP BY поля_группировки;
```

Виды агрегатных функций

SUM(поле_таблицы)	Возвращает сумму значений
AVG(поле_таблицы)	Возвращает среднее значение
COUNT(поле_таблицы)	Возвращает количество записей
MIN(поле_таблицы)	Возвращает минимальное значение
MAX(поле_таблицы)	Возвращает максимальное значение

Ниже представлены несколько примеров использования агрегатных функций:

- Найдём количество видов магазинов и отсортируем полученный список по убыванию:

```
SELECT name, COUNT(*) as amount FROM Shop
GROUP BY name
ORDER BY amount DESC
```

- Найдем средний остаток на балансе магазинов отсортированный по возрастанию

```
SELECT name, avg(balance) as avg_balance from shop
group by name
order by avg_balance
```

Практическое задание

1. Создайте дополнительную таблицу “Сотрудник” выполнив следующую команду

```
CREATE TABLE worker (
worker_id INTEGER PRIMARY KEY,
shop_id INTEGER REFERENCES product (id),
name VARCHAR(255),
salary INTEGER NOT NULL,
position VARCHAR(255));
```

2. Внесите в таблицу данные.
3. К таблицам напишите запросы операциями группировки и сортировки. Также реализуйте запросы с использованием агрегатных функций для таблицы “Сотрудник”

Лабораторная работа № 13 Основы работы с протоколом HTTP

Цель работы: Получить практические навыки в использовании Postman для тестирования и разработки API, изучить основные методы HTTP, научиться создавать и отправлять запросы, а также анализировать ответы сервера. По завершении каждого задания студенты должны сделать скриншоты своих запросов и ответов для включения в отчет. Отчет должен также содержать краткий анализ полученных данных и выводы о выполненной работе.

Задание №1: Ознакомление с Postman

1. Установка Postman. Перейти на официальный сайт Postman и скачать последнюю версию приложения для вашей операционной системы.
2. Установить и открыть Postman.
3. Создание первого запроса. В интерфейсе Postman нажать на кнопку "**New**" и выбрать "**Request**". Дать запросу название и сохранить его в новую или существующую коллекцию. В поле ввода URL ввести адрес публичного API (например, **https://api.publicapis.org/entries**) и выбрать метод **GET**. Нажать на кнопку "**Send**" для отправки запроса и просмотреть ответ сервера внизу экрана.

Задание 2: Работа с публичным API

1. Выбрать **REST Countries API (URL: https://restcountries.com/v3.1/all)** для получения информации о всех странах.

2. Создать новый GET запрос к выбранному API.
3. Нажать "Send" и изучить структуру ответа в формате JSON.

Задание 3: Отправка данных с использованием POST

1. Использовать JSONPlaceholder (URL: https://jsonplaceholder.typicode.com/posts) для отправки тестовых POST запросов.
2. Переключиться на метод POST и ввести URL для создания поста.
3. В "Body" выбрать тип "raw" и формат JSON, ввести тело запроса.
4. Отправить запрос и проанализировать ответ сервера.

Задание 4: Модификация данных с использованием PUT

1. Использовать JSONPlaceholder, выбрать существующий пост для обновления.
2. Изменить метод на PUT и указать полный URL для обновления поста.
3. В "Body" изменить содержание поста, отправить запрос и проанализировать ответ.
4.

Задание 5: Удаление данных с использованием DELETE

1. Использовать JSONPlaceholder, выбрать пост для удаления.
2. Изменить метод на DELETE и указать URL поста для удаления.
3. Отправить запрос и проверить статус-код ответа сервера.

Лабораторная работа 14: Разработка RESTful API: спецификация и архитектура

Для выполнения лабораторный работы, вам необходимо установить пакеты fastapi и uvicorn, для этого используйте команды:

```
pip install fastapi
pip install uvicorn
```

Создаем файл main.py. Импортируем fastapi и pyjokes в main.py

```
from fastapi import FastAPI
import pyjokes
```

Создадим объект фастапи, куда далее будут подключаться роуты. Будем называть его далее приложение фастапи.

```
app = FastAPI()
```

Создадим простой роут. Для этого напишем простую функцию, которую обернем декоратором. Декоратор использует приложение созданное ранее, http метод и путь по которому будет работать данный роут.

```
@app.get("/")
def joke():
```

```
    return pyjokes.get_joke()
```

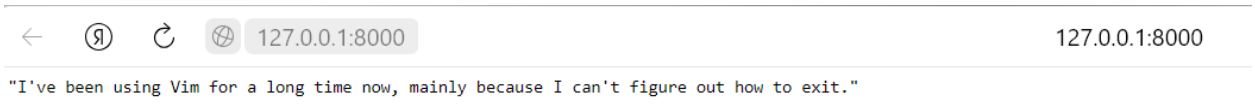
Для начала работы необходимо запустить uvicorn – наш веб-сервер. Воспользуемся командой в консоли, запущенной в месте в одной директории с main.py.

```
uvicorn main:app --reload
```

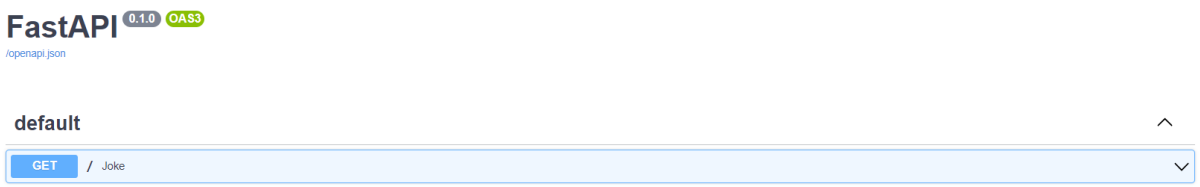
Результат запуска uvicorn.

```
K:\MTUCI\Students\FastAPI_01>uvicorn main:app --reload
←[32mINFO←[0m:     Will watch for changes in these directories: ['K:\\MTUCI\\Students\\FastAPI_01']
←[32mINFO←[0m:     Uvicorn running on ←[1mhttp://127.0.0.1:8000←[0m (Press CTRL+C to quit)
←[32mINFO←[0m:     Started reloader process [←[36m←[1m14448←[0m] using ←[36m←[1mWatchGod←[0m
←[32mINFO←[0m:     Started server process [←[36m14744←[0m]
←[32mINFO←[0m:     Waiting for application startup.
←[32mINFO←[0m:     Application startup complete.
```

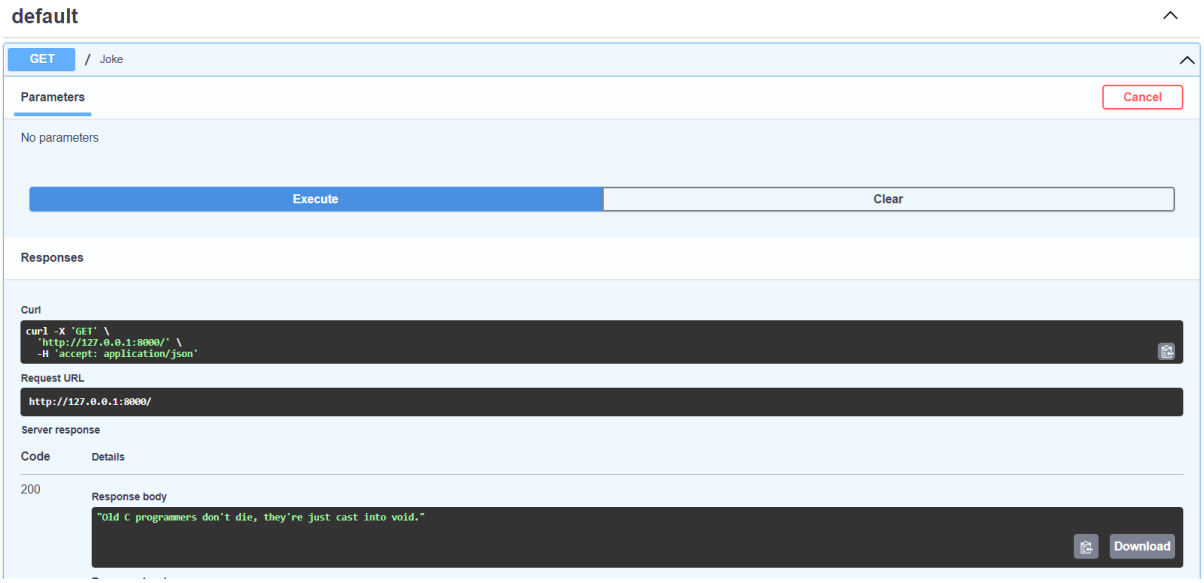
Перейдем по базовому адресу, который указывается при запуске uvicorn - http://127.0.0.1:8000. Перейдя по ссылке, увидим результат, как на рисунке ниже.



Для удобной работы с нашим приложением будем использовать swagger. Он открывается по следующей ссылке http://127.0.0.1:8000/docs. По данной ссылке мы должны увидеть изображение ниже.



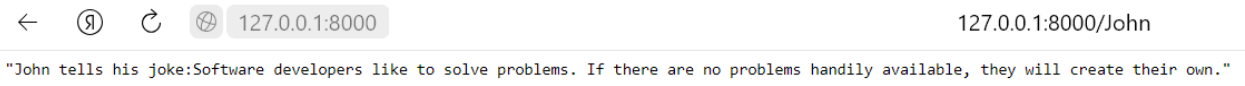
На странице сваггера отображаются все добавленные роуты. Развернем наш единственный роут и попробуем выполнить его. Для этого сначала нажмем на кнопку “Try it out” и далее execute.



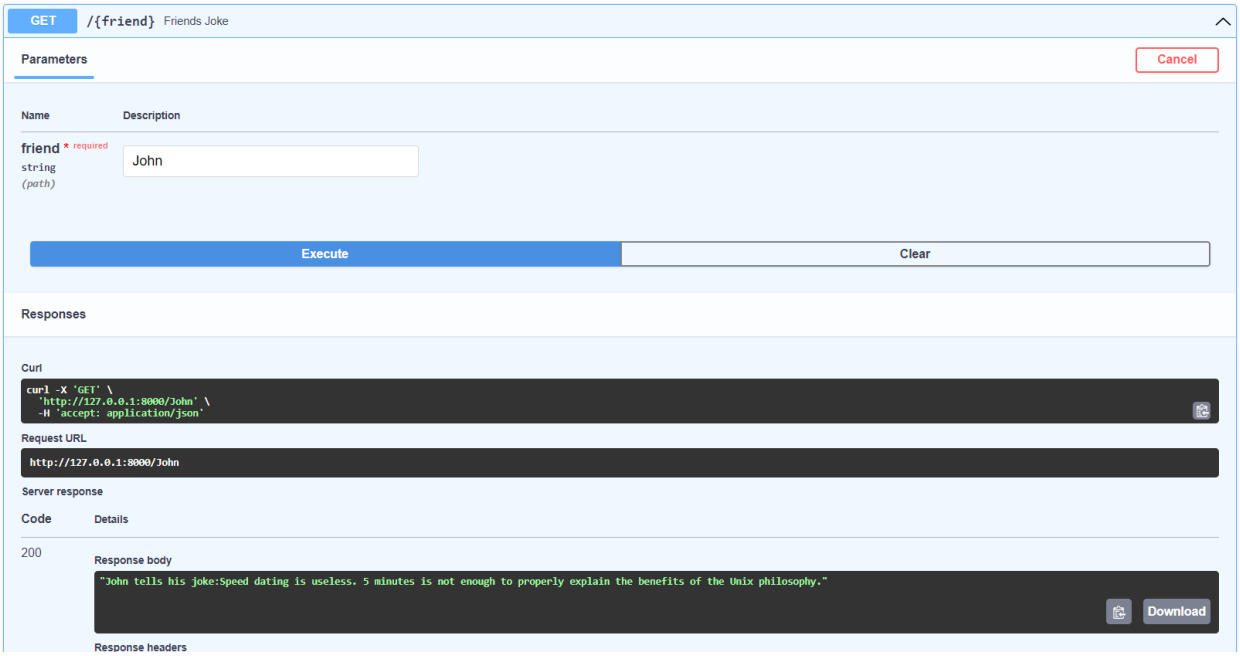
Добавим еще один роут, где будет параметр в пути, чтобы мы могли представить шутку от какого-то конкретного человека. Для этого в фигурных скобках добавим название желаемого параметра и добавим его же в параметрах функции. Итоговый вид у роута будет, как на рисунке ниже.

```
@app.get("/{friend}")
def friends_joke(friend: str):
    return friend + " tells his joke:" +
pyjokes.get_joke()
```

Добавим к базовому пути http://127.0.0.1:8000/John через слеш желаемое значение параметра name, чтобы результат был, как на рисунке ниже.



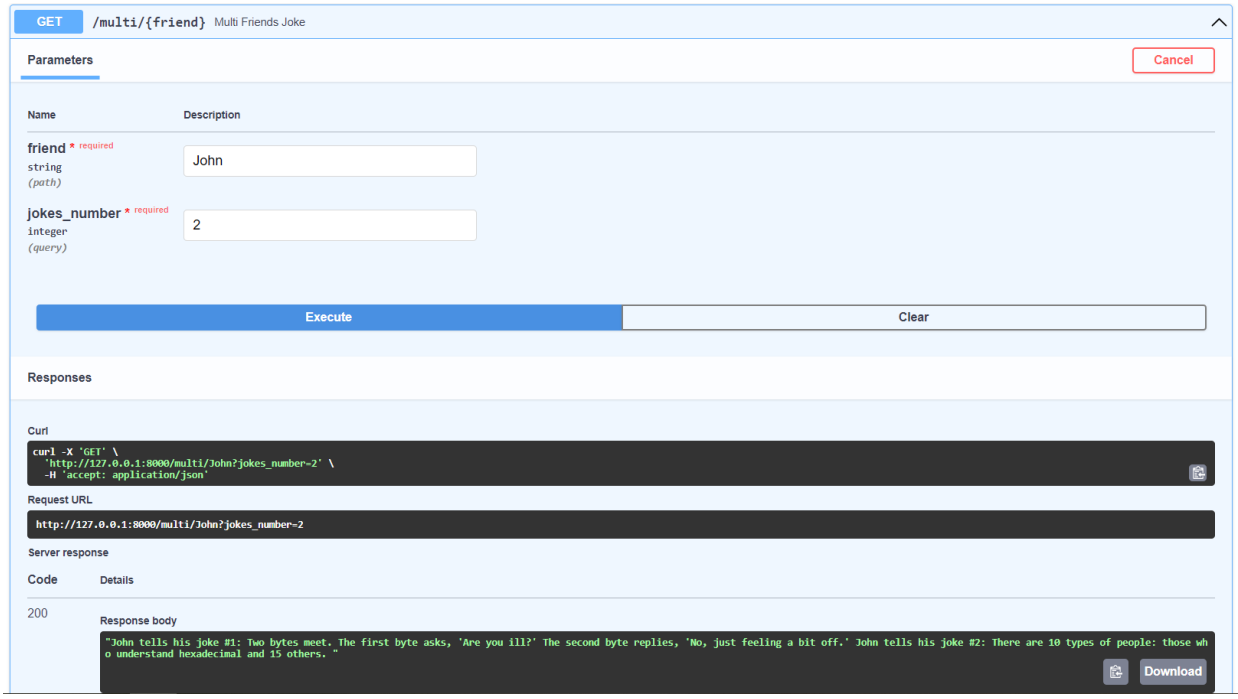
В сваггере новый роут будет выглядеть следующем образом. Попробуем снова его запустить.



Добавим еще один роут, где будет возможность выбрать количество шуток. Для этого добавим еще один параметр, который не будем указывать в пути роута. Это будет query параметр jokes_number. Он не будет указан в пути, но также необходим для корректной работы роута. Итоговый вид роута показан на изображении ниже.

```
@app.get("/multi/{friend}")
def multi_friends_joke(friend: str, jokes_number:
int):
    result = ""
    for i in range(jokes_number):
        result += friend + f" tells his joke #{i + 1}:
" + pyjokes.get_joke() + " "
    return result
```

Откроем новый роут в сваггере и попробуем запустить его.



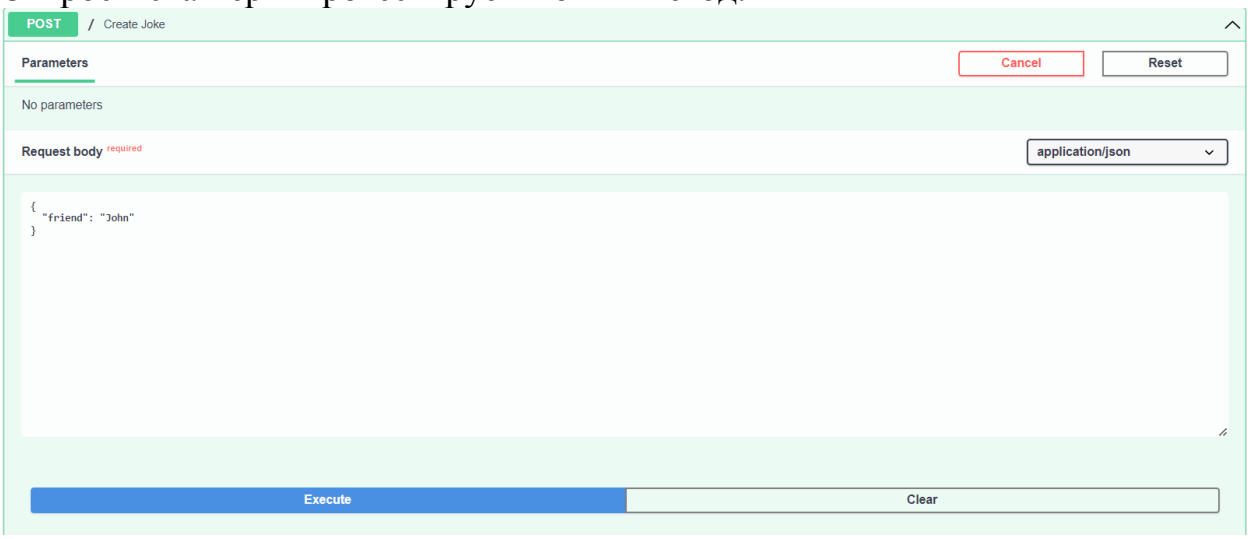
Рассмотрим и другой способ передачи информации в роут. Например, некоторые http запросы поддерживают передачу данных в теле запроса (Body). Создадим новый роут, используя метод POST, и создадим схему тела запроса, которую будет принимать роут для корректной работы. Импортируем из библиотеки pydantic класс BaseModel, как показано на рисунке ниже.

```
from pydantic import BaseModel
```

И создадим на его основе схему получения шутки, которую и передадим функции на вход, как показано ниже.

```
class Joke(BaseModel):
    friend: str
    joke: str

class JokeInput(BaseModel):
    friend: str

@app.post("/")
def create_joke(joke_input: JokeInput):
    return joke_input.friend + " tells his joke:" +
pyjokes.get_joke()
```

Откроем сваггер и протестируем новый метод.

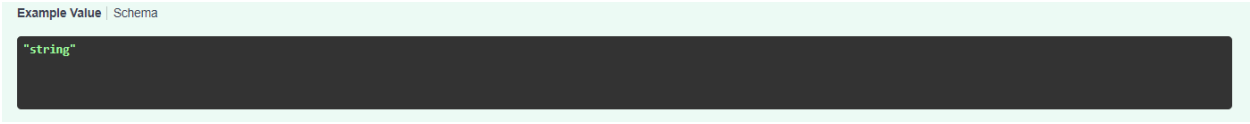


Результат успешного запроса показан ниже.

Схемы позволяют и задавать образец ожидаемого ответа. Обновим ранее созданный POST метод, согласно примеру ниже, чтобы ответ выдавался в формате ранее описанным в схеме Joke.

```
@app.post("/")
def create_joke(joke_input: JokeInput):
    return Joke(friend=joke_input.friend,
joke=pyjokes.get_joke())
```

Посмотрим на изменения в сваггере и протестируем обновленный роут. Перед запуском обратим внимание, что схема не показывается в ожидаемом ответе.



Результат успешного выполнения показан ниже.

Добавим комментарий с описанием роута и поставим в параметрах response_model, чтобы добавить валидацию ответа функции. Обновленный эндпоинт будет выглядеть следующим образом.

```
@app.post("/", response_model=Joke)
def create_joke(joke_input: JokeInput):
    """Создание шутки"""
    return Joke(friend=joke_input.friend,
joke=pyjokes.get_joke())
```

Обновим страницу со сваггером и увидим добавленное описание эндпоинта и схему ожидаемого ответа.

Самостоятельно задание:

Создать простой REST сервис, использующий библиотеку wikipedia. Создайте 1 роут с параметром path, 1 роут с параметром query, 1 роут с передачей параметров в теле запроса. Все запросы должны возвращаться и валидироваться по схемам.

Ссылка на документацию по Wikipedia в python

https://rukovodstvo.net/posts/id_1061/

Лабораторная работа №15 - Изучение основ docker и деплой проекта

Аннотация

Для выполнения данной работы потребуется Unix система (Linux, нативный или при помощи виртуализации или MacOS)

Установка Docker

Если у вас не установлен Docker, воспользуйтесь официальным гайдом для Ubuntu или другого дистрибутива Linux. Для MacOS можно воспользоваться Docker Desktop (гайд).

Из предлагаемых трёх вариантов установки рекомендуется воспользоваться вариантом “Install using the repository”.

Также необходимо установить Docker compose, для этого воспользуйтесь инструкцией.

Установка Gitlab

Для установки Gitlab и Gitlab-Runner создайте папку *devops*, а в ней папку *gitlab*.

В этой папке создайте файл **docker-compose.yml (содержимое файла приведено ниже)** и четыре папки: *config, logs, data, runner-config*.

```
nikolay@Nikolays-iMac ~> mkdir devops
nikolay@Nikolays-iMac ~> cd devops/
nikolay@Nikolays-iMac ~/devops> mkdir gitlab
nikolay@Nikolays-iMac ~/devops> cd gitlab
nikolay@Nikolays-iMac ~/d/gitlab> mkdir data
nikolay@Nikolays-iMac ~/d/gitlab> mkdir logs
nikolay@Nikolays-iMac ~/d/gitlab> mkdir config
nikolay@Nikolays-iMac ~/d/gitlab> mkdir runner-config
nikolay@Nikolays-iMac ~/d/gitlab> vim docker-compose.yaml
```

Файл **docker-compose.yml**:

```
version: '3.6'
services:
  gitlab:
    image: 'gitlab/gitlab-ee:latest'
    restart: always
    hostname: 'gitlab.example.com'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://gitlab.example.com'
    ports:
      - '443:443'
      - '22:22'
      - '80:80'
    volumes:
      - '/Users/nikolay/devops/gitlab/config:/etc/gitlab'
      - '/Users/nikolay/devops/gitlab/logs:/var/log/gitlab'
      - '/Users/nikolay/devops/gitlab/data:/var/opt/gitlab'
    shm_size: '256m'
  gitlab-runner:
    image: 'gitlab/gitlab-runner:latest'
    restart: always
    depends_on:
      - gitlab
    volumes:
      - '/var/run/docker.sock:/var/run/docker.sock'
      - '/Users/nikolay/devops/gitlab/runner-config:/etc/gitlab-runner'
```

*в местах, выделенных жирным шрифтом, необходимо указать путь **к папкам** на **НАШЕМ** компьютере. (порты меняются в случае, если у вас уже запущены программы на них).

Далее выполняем команду “*docker-compose up -d*”. Эта команда загрузит образы контейнеров gitlab и запустит их.

```
nikolay@Nikolays-iMac ~/d/gitlab> vim docker-compose.yaml
nikolay@Nikolays-iMac ~/d/gitlab> docker-compose up -d
[+] Running 3/3
 ⠿ Network gitlab_default                Created
 ⠿ Container gitlab-gitlab-1             Started
 ⠿ Container gitlab-gitlab-runner-1      Started
nikolay@Nikolays-iMac ~/d/gitlab>
```

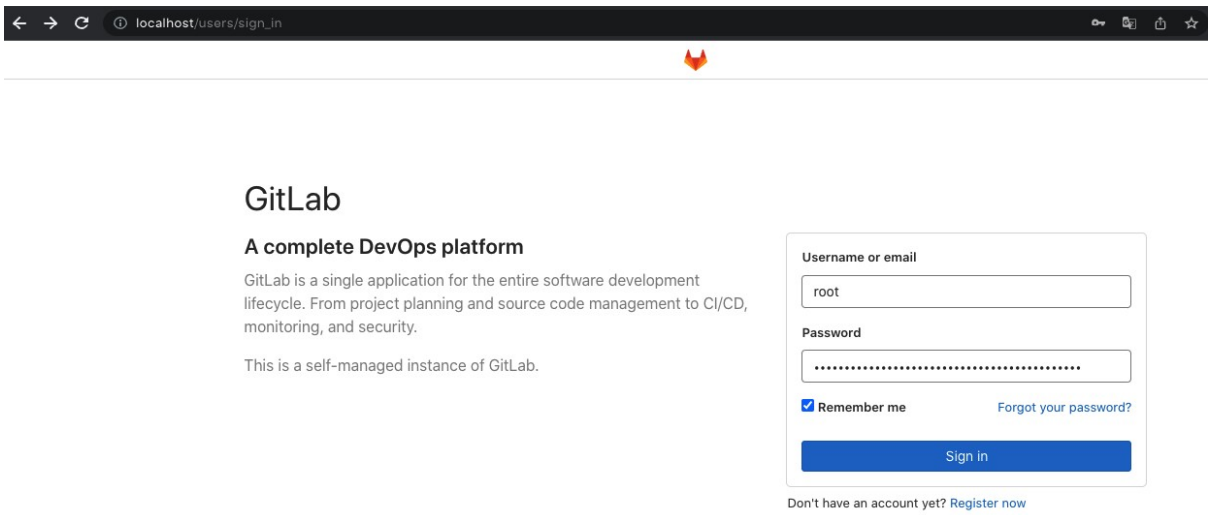
Теперь дождитесь момента, когда в папке *config* появиться файл **initial_root_password**. В этом файле храниться изначальный пароль для аккаунта root, необходимый, чтобы зайти в панель администрирования gitlab. **ВАЖНО, при перезапуске gitlab данный файл будет удалён.**

```
nikolay@Nikolays-iMac ~/d/gitlab> ls
config              data                docker-compose.yaml logs                runner-config
nikolay@Nikolays-iMac ~/d/gitlab> cd config/
nikolay@Nikolays-iMac ~/d/g/config> ls
gitlab-secrets.json      initial_root_password    ssh_host_ecdsa_key.pub   ssh_host_ed25519_key.pub ssh_host_rsa_key.pub
gitlab.rb                ssh_host_ecdsa_key       ssh_host_ed25519_key     ssh_host_rsa_key         trusted-certs
nikolay@Nikolays-iMac ~/d/g/config> cat initial_root_password
# WARNING: This value is valid only in the following conditions
#          1. If provided manually (either via `GITLAB_ROOT_PASSWORD` environment variable or via `gitlab_rails['initial_root_password']` setting in `gitlab
.rb`, it was provided before database was seeded for the first time (usually, the first reconfigure run).
#          2. Password hasn't been changed manually, either via UI or via command line.
#
#          If the password shown here doesn't work, you must reset the admin password following https://docs.gitlab.com/ee/security/reset_user_password.html
#reset-your-root-password.

Password: x36vY0qmzt/tVJGt1FjpTbC8yUQmNLSDLdUefU8ENW8=

# NOTE: This file will be automatically deleted in the first reconfigure run after 24 hours.
```

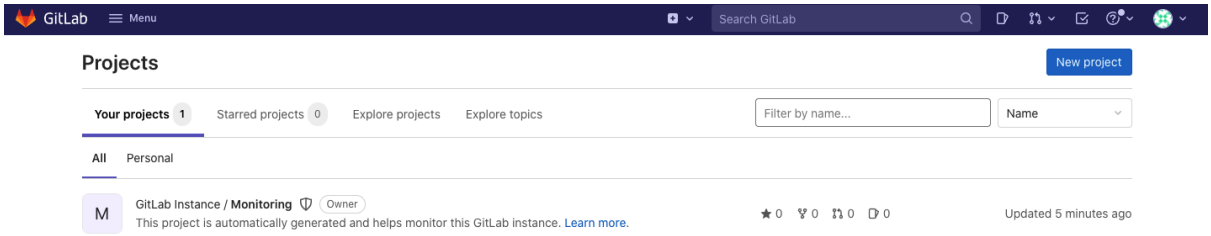
Вводим пароль, который лежит в файле “initial_root_password” и логин “root” по адресу “http://localhost/”. (Gitlab сконфигурирован для работы с адресом 'gitlab.example.com’, однако наш DNS не знает об этом, поэтомы мы заходим через локальный адрес и docker выполняет перенаправление).

ВАЖНЫЙ МОМЕНТ!

Веб-интерфейс появится далеко не сразу, т.к. для него контейнеру необходимо достаточно продолжительное время на запуск всех зависимостей (более 30 минут).

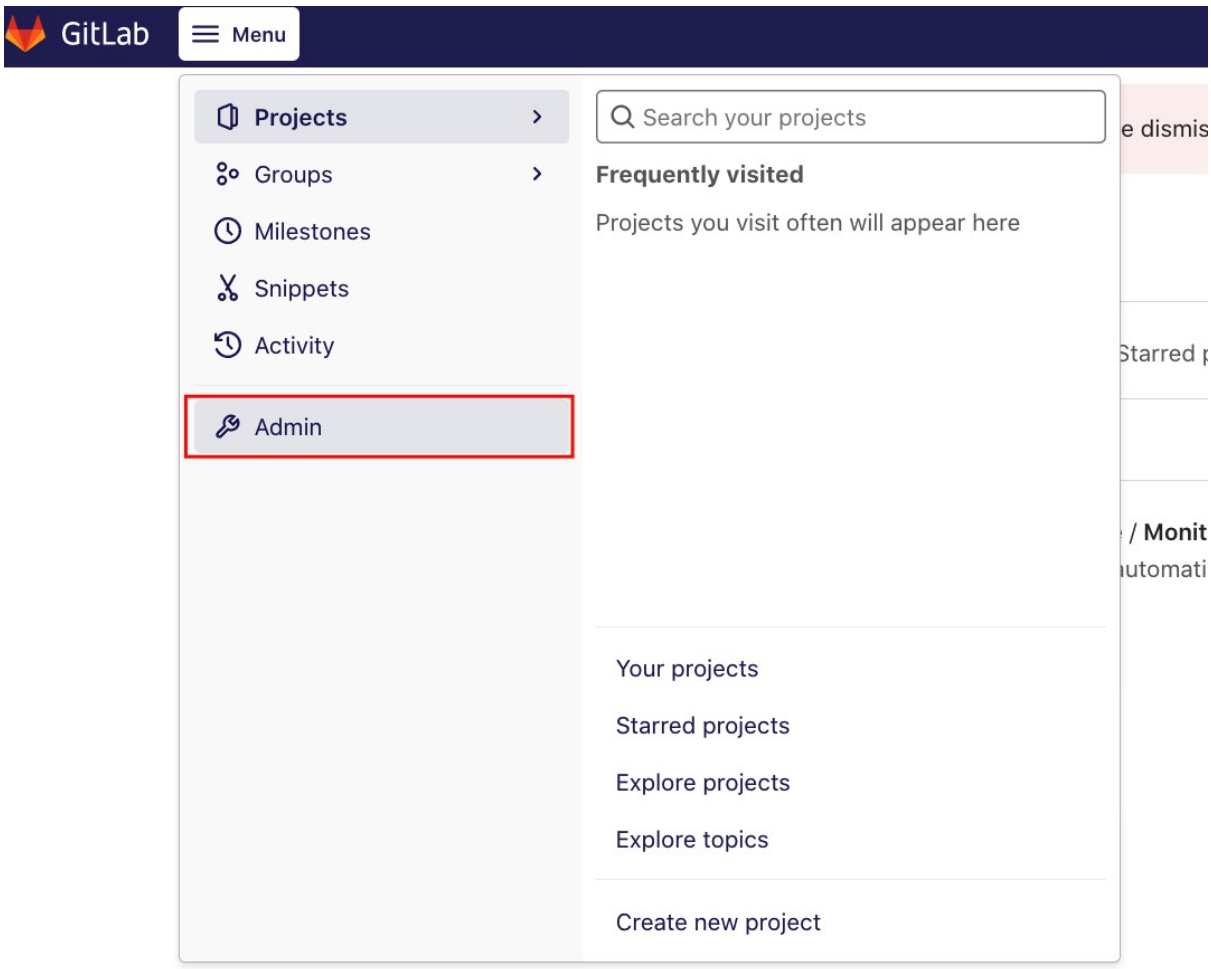


Авторизировавшись, мы увидим следующий интерфейс:

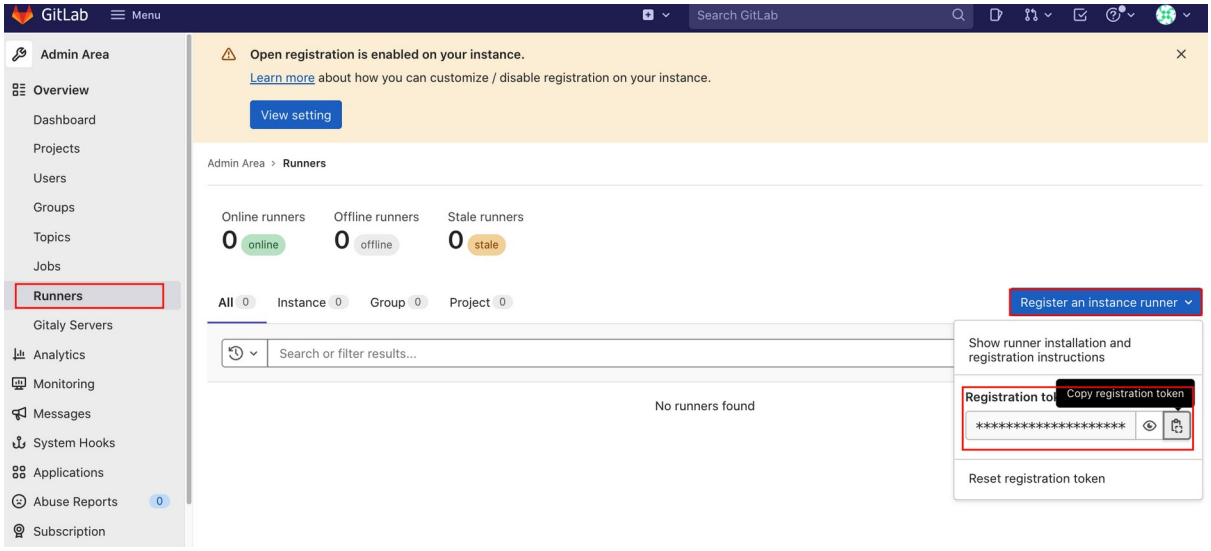


Теперь необходимо выполнить регистрацию Gitlab-Runner.

Для начала перейдем в панель админа.



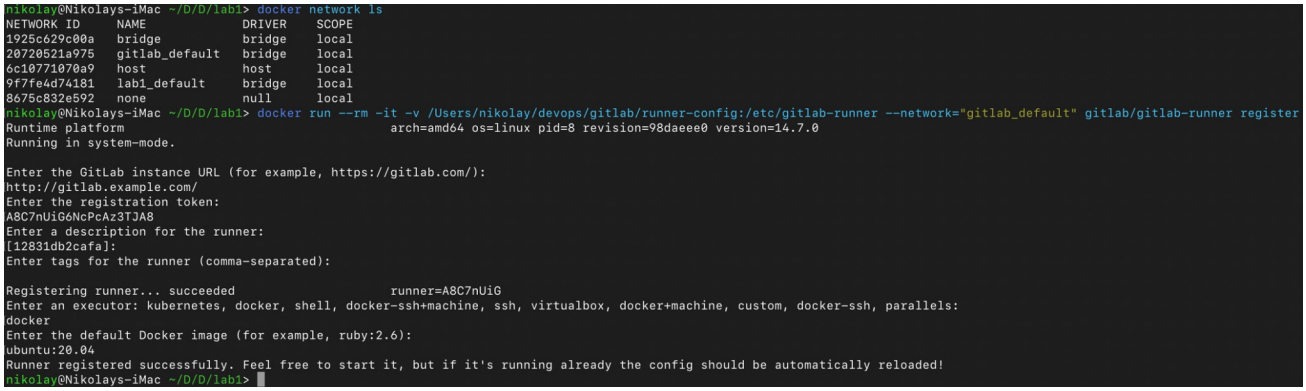
Перейдем в пункт “Runners” и нажмем на кнопку “Register an instance runner”, после чего можно будет скопировать токен, который нам потребуется дальше.



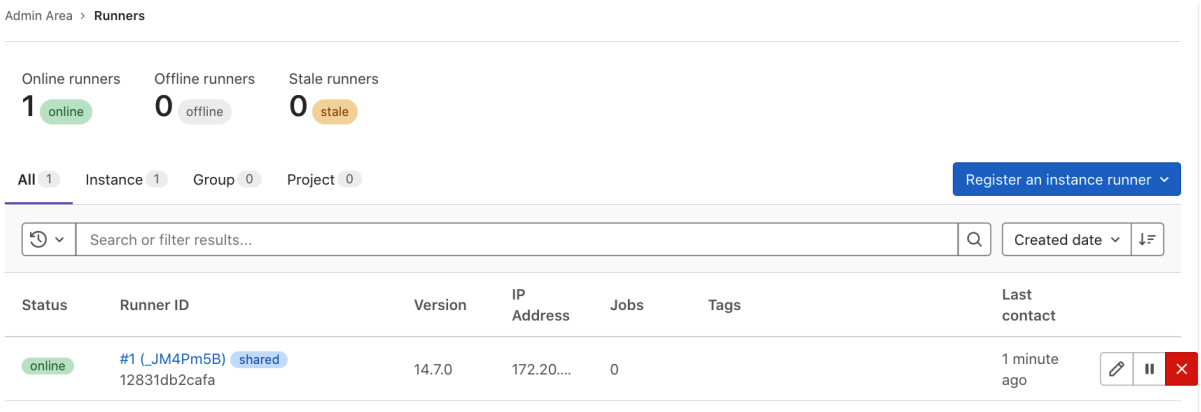
1. Возвращаемся к терминалу и выполняем команду:

"docker run --rm -it -v /Users/nikolay/devops/gitlab/runner-config:/etc/gitlab-runner --network="gitlab_default" gitlab/gitlab-runner register"

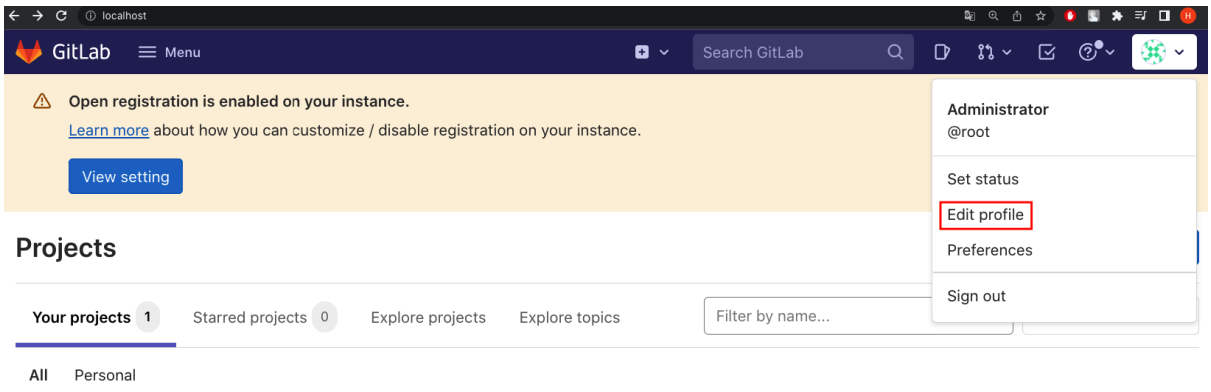
2. В строке ввода адреса объекта сети вводим: *http://gitlab.example.com/*.
3. Теперь необходимо будет ввести токен, который был скопирован в админской панеле.
4. Последним этапом вводим в качестве стандартного образа Docker - *ubuntu:20.04*



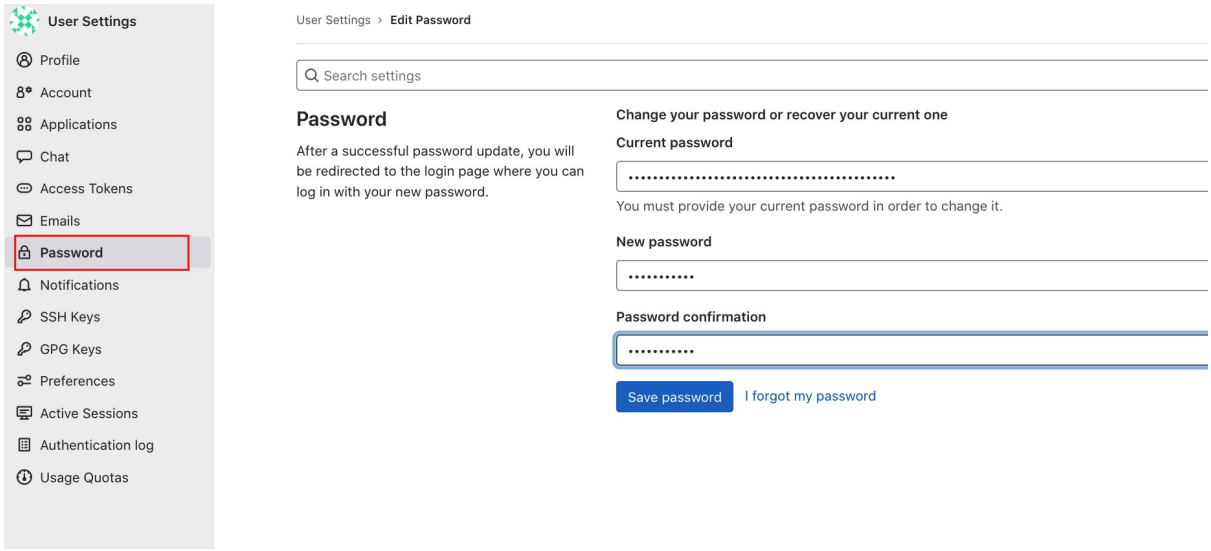
Теперь, если обновить страницу админской панели, мы увидим добавленный нами раннер.



Gitlab установлен и готов к работе, но необходимо сделать еще одно действие - изменить пароль админа. Для этого перейдем в раздел настроек профиля.



Далее раздел “password”, вводим сгенерированный пароль и придумываем новый (должен быть больше 8 символов). После сохранения пароля, произойдет logout, повторно вводим root и новый пароль.



Теперь gitlab полностью готов.

Важные команды:

- Чтобы приостановить его работу, выполните команду “***docker-compose stop***” из папки *gitlab*.
- Чтобы возобновить его работу, выполните команду “***docker-compose start***” из папки *gitlab*.
- Чтобы удалить контейнеры gitlab (но не его данные), выполните команду “***docker-compose down***” из папки *gitlab*.

Установка Jenkins

Вернемся в каталог “devops” и создадим рядом с папкой “gitlab” папку “jenkins”, а в ней директории “data” и “certs”.

Далее нам необходимо прописать docker-compose файл для jenkins.

docker-compose.yaml

```
version: '3.6'
services:
 jenkins:
   image: 'jenkins/jenkins:lts-jdk11'
   restart: always
   ports:
     - '8080:8080'
     - '50000:50000'
   volumes:
     - '/Users/nikolay/devops/jenkins/data:/var/jenkins_home'
   environment:
     - DOCKER_HOST='tcp://docker:2376'
 docker:
   image: 'docker:dind'
   restart: always
   privileged: true
   volumes:
     - '/Users/nikolay/devops/jenkins/data:/var/jenkins_home'
     - '/Users/nikolay/devops/jenkins/certs:/certs/client'
   environment:
     - DOCKER_TLS_CERT_DIR=/certs
```

```
networks:
  default:
    external: true
    name: gitlab_default
```

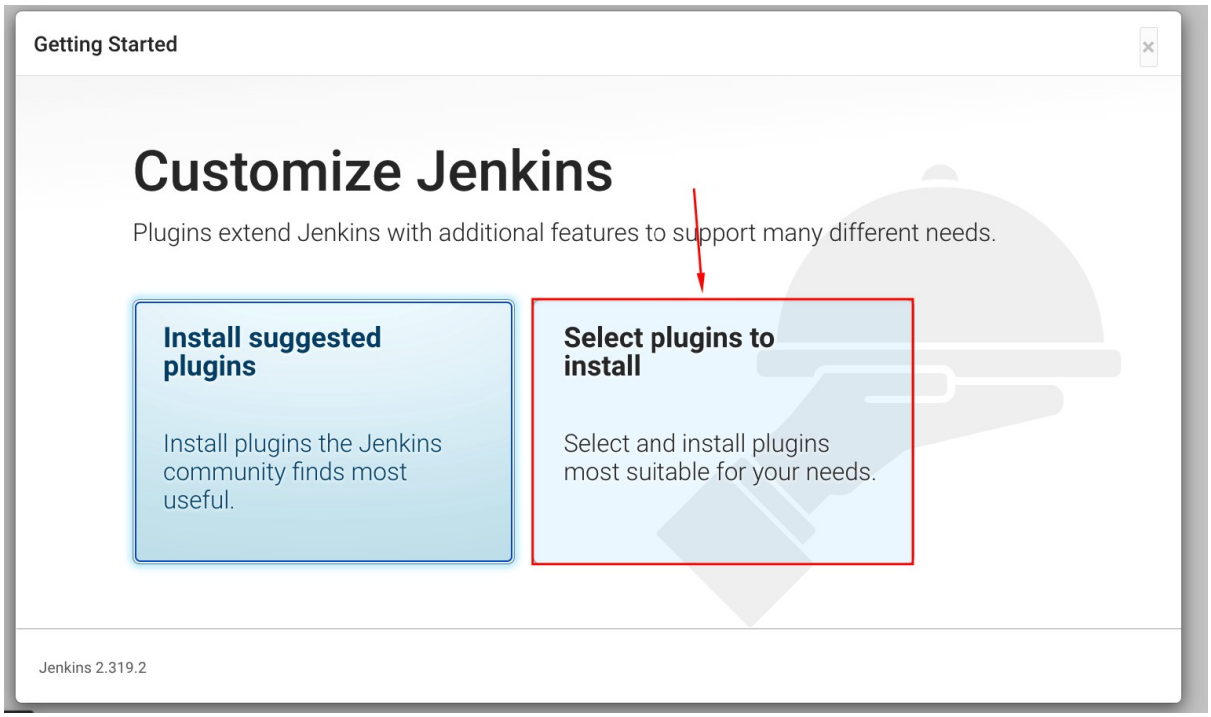
Поднимем docker-compose:

```
nikolay@Nikolays-iMac ~/d/jenkins> docker-compose up -d
[+] Running 3/3
 ⠿ Network jenkins_default      Created                                  0.0s
 ⠿ Container jenkins-docker-1   Started                                  0.4s
 ⠿ Container jenkins-jenkins-1  Starte...                                0.4s
```

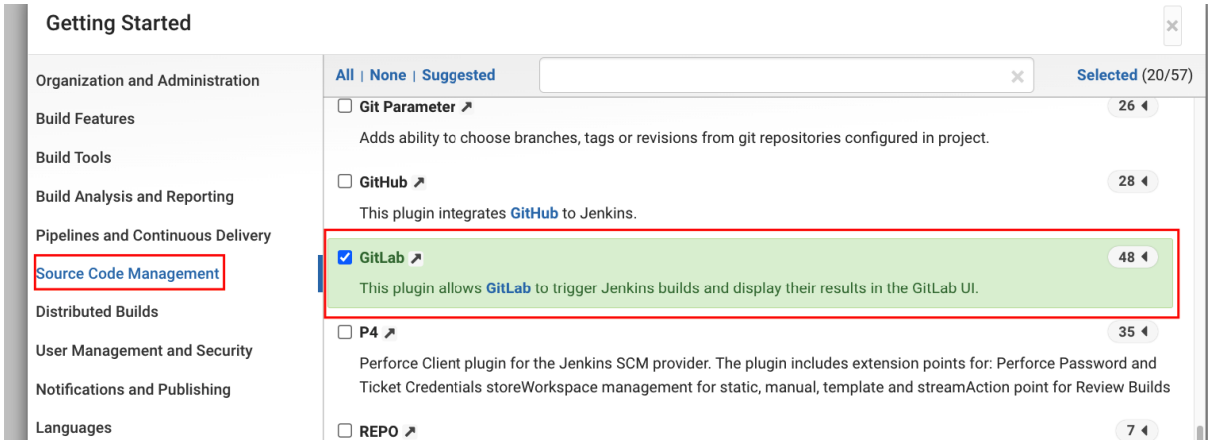
Следующим этапом нам необходимо узнать инициализационный пароль админа. Для этого нам необходимо перейти в папку “data/secrets” и в ней просмотреть файл initialAdminPassword, в нем и лежит пароль.

```
nikolay@Nikolays-iMac ~/d/jenkins> cd data/
nikolay@Nikolays-iMac ~/d/j/data> ls
config.xml                     queue.xml.bak
copy_reference_file.log        secret.key
hudson.model.UpdateCenter.xml  secret.key.not-so-secret
identity.key.enc               secrets
jenkins.telemetry.Correlator.xml updates
jobs                           userContent
nodeMonitors.xml               users
nodes                          war
plugins
nikolay@Nikolays-iMac ~/d/j/data> cd secrets/
nikolay@Nikolays-iMac ~/d/j/d/secrets> ls
filepath-filters.d
initialAdminPassword
jenkins.model.Jenkins.crumbSalt
master.key
org.jenkinsci.main.modules.instance_identity.InstanceIdentity.KEY
slave-to-master-security-kill-switch
whitelisted-callables.d
nikolay@Nikolays-iMac ~/d/j/d/secrets> cat initialAdminPassword
f9e4dce8267544959f8988f09c3548fb
nikolay@Nikolays-iMac ~/d/j/d/secrets>
```

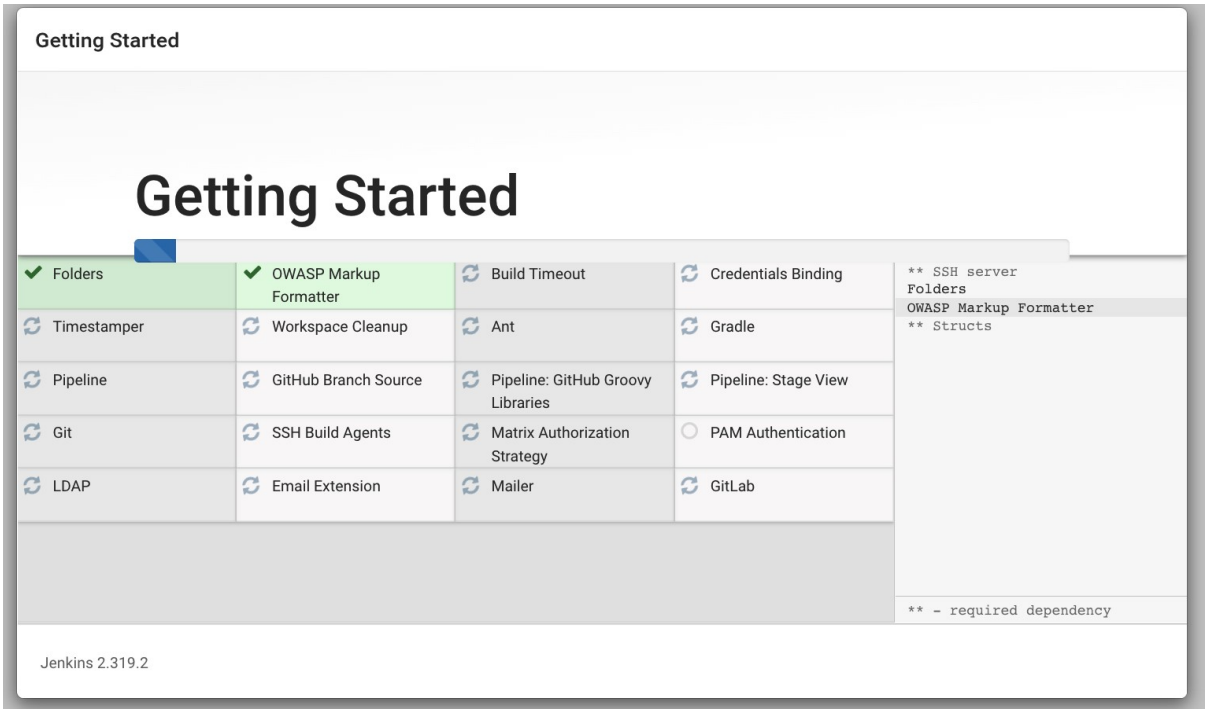
Теперь откроем браузер и перейдем по ссылке: *http://localhost:8080/*, перед нами откроется веб-интерфейс jenkins, вводим скопированный пароль. Далее необходимо произвести первичную настройку. Выбираем ручную установку плагинов.



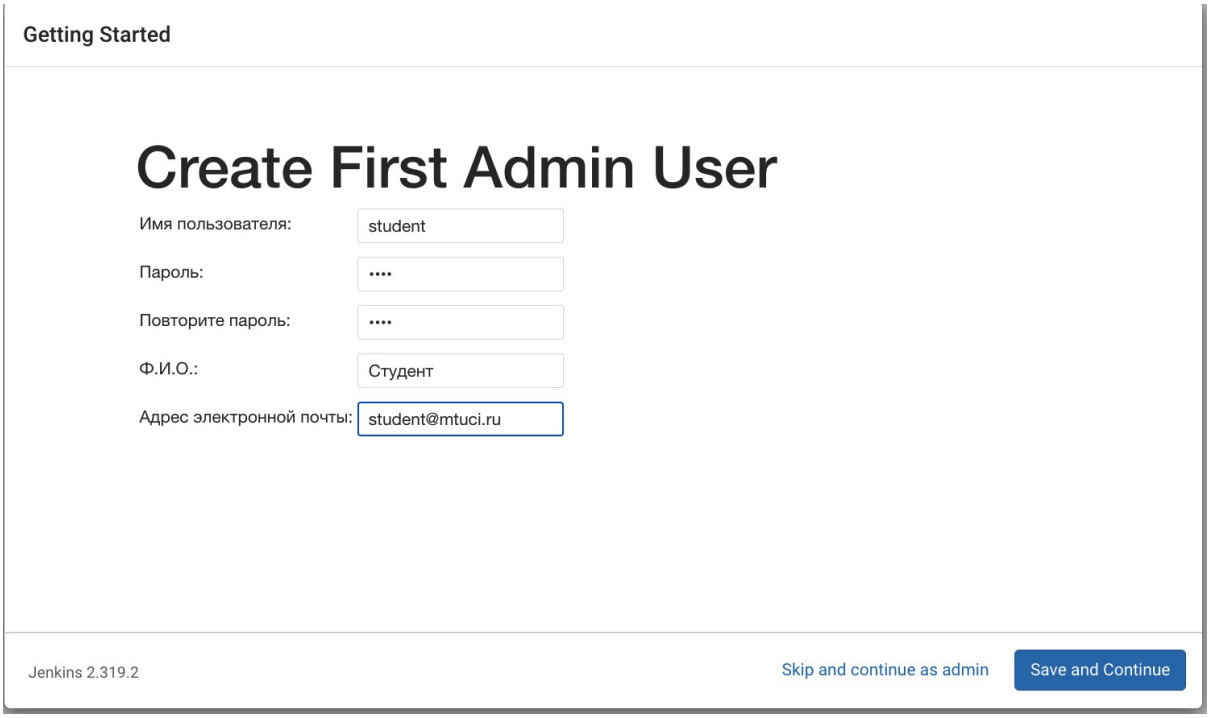
Находим в их перечне “GitLab”, ставим галочку и нажимаем установить.



Перед вами должно оказаться следующее окно

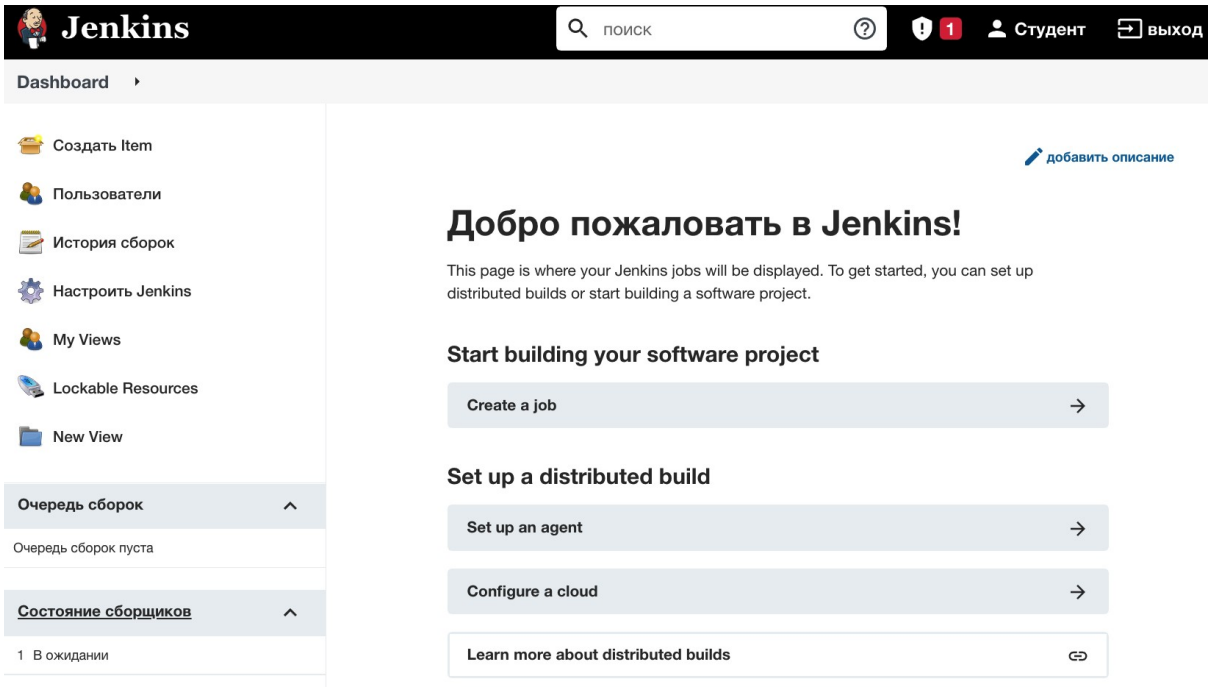


Следующим этапом jenkins попросит создать пользователя, заполняем все.



В последнем этапе, jenkins уточнит, оставляем ли мы его инстанс по стандартному пути, нажимаем далее.

Теперь мы попали на главную страницу jenkins и он готов к работе.



Для остановки и повторного запуска воспользуйтесь командами, который приведены в конце части про gitlab, но их уже необходимо выполнять находясь в папке jenkins.

Полезные ссылки

1. Установка GitLab
2. Установка GitLab Runner
3. Установка Jenkins

Лабораторная работа № 16 Разработка игрового приложения на PyQT

Цель работы:

Овладеть основами создания графического интерфейса пользователя (GUI) с использованием библиотеки PyQt5 на примере разработки игры "Крестики-нолики".

Исходные данные для выполнения работы:

Вам предоставлены куски кода, которые служат основой для вашего приложения. Ваша задача — использовать эти фрагменты для создания полноценной игры "Крестики-нолики".

1.Импорт необходимых модулей:

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow,
QPushButton, QLabel
from PyQt5.QtCore import Qt
```

2.Определение класса главного окна игры:

```
class TicTacToeGame(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Крестики-нолики на PyQt5")
        self.setGeometry(100, 100, 300, 320)  # X, Y,
ширина, высота
        self.buttons = []
        self.current_player = "X"
        self.move_count = 0  # Счетчик ходов
        self.create_buttons()
        self.status_label = QLabel(self)
        self.status_label.setGeometry(0, 300, 300, 20)
```

```
# Расположение под игровым полем
        self.status_label.setAlignment(Qt.AlignCenter)
```

3.Метод создания кнопок игрового поля

```
def create_buttons(self):
    for row in range(3):
        row_buttons = []
        for col in range(3):
            button = QPushButton(self)
            button.setText("")
            button.setGeometry(col * 100, row * 100,
100, 100)
            button.clicked.connect(self.on_click)
            row_buttons.append(button)
        self.buttons.append(row_buttons)
```

4.Метод обработки нажатия на кнопку

```
def on_click(self):
    button = self.sender()
    if button.text() == "":
        button.setText(self.current_player)
        self.move_count += 1
        self.check_winner()
        self.switch_player()
```

5.Метод проверки условий победы:

```
def check_winner(self):
    # Описаны выигрышные комбинации
```

6. Метод завершения игры и вывода результата:

```
def end_game(self, message):
    self.status_label.setText(message)
    for row in self.buttons:
        for button in row:
            button.setEnabled(False)
```

7. Метод смены игрока:

```
def switch_player(self):
    self.current_player = "O" if self.current_player ==
"X" else "X"
```

8. Запуск приложения:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = TicTacToeGame()
    window.show()
    sys.exit(app.exec_())
```

Задания для самостоятельной работы:

1.Используйте предоставленные фрагменты кода для создания игры "Крестики-нолики".

2.Дополните метод check_winner логикой определения победителя или ничьей.

3.Реализуйте механизм, позволяющий начать новую игру после завершения текущей без перезапуска приложения.

Лабораторная работа № 17 СОЗДАНИЕ ЧАТА С ИСПОЛЬЗОВАНИЕМ WEBSOCKETS (FASTAPI + AIOHTTP)

Цель работы:

Научиться реализовывать простой веб-чат с использованием WebSockets на базе FastAPI и AIOHTTP. Изучить основы асинхронного взаимодействия между клиентом и сервером.

```
pip install fastapi aiohttp uvicorn jinja2
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from pathlib import Path
import uvicorn

app = FastAPI()

html_path = Path(__file__).parent / "templates"
app.mount("/static", StaticFiles(directory=html_path), name="static")

clients = []

@app.get("/")
async def get():
    html = (html_path / "chat.html").read_text()
    return HTMLResponse(content=html, status_code=200)

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    clients.append(websocket)
    try:
        while True:
```

```
            data = await websocket.receive_text()
            for client in clients:
                if client != websocket:
                    await client.send_text(data)
    except WebSocketDisconnect:
        clients.remove(websocket)

if __name__ == "__main__":
    uvicorn.run("main:app", reload=True)
```

Используйте предоставленный код для загрузки основного серевера и установки библиотек

Напишите HTML-шаблон по коду ниже

```
<!DOCTYPE html>
<html>
<head>
    <title>WebSocket Chat</title>
</head>
<body>
    <h1>Чат на WebSockets</h1>
    <input type="text" id="messageInput" autocomplete="off"/>
    <button onclick="sendMessage()">Отправить</button>
    <ul id="messages"></ul>

    <script>
        const ws = new WebSocket("ws://localhost:8000/ws");

        ws.onmessage = function(event) {
            const messages = document.getElementById("messages");
            const message = document.createElement("li");
            message.textContent = event.data;
            messages.appendChild(message);
        };

        function sendMessage() {
            const input = document.getElementById("messageInput");
```

```
            ws.send(input.value);
            input.value = '';
        }
    </script>
</body>
</html>
```

Анализ результатов:

Запустите сервер FastAPI с помощью команды:

uvicorn main:app –reload

Перейдите по адресу http://localhost:8000 и откройте страницу в двух вкладках.

Отправьте сообщение из одной вкладки — оно должно появиться в другой.

Лабораторная работа № 18 АСИНХРОННАЯ ЗАГРУЗКА ДАННЫХ С САЙТА С ИСПОЛЬЗОВАНИЕМ aiohttp

Цель работы:

Научиться использовать библиотеку aiohttp для асинхронной загрузки веб-страниц. Изучить базовые принципы работы с asyncio и параллельного выполнения задач.

```
pip install aiohttp
import aiohttp
import asyncio

urls = [
    "https://example.com",
    "https://httpbin.org/get",
    "https://www.python.org"
]
```

```
async def fetch(session, url):
    async with session.get(url) as response:
        text = await response.text()
        print(f"{url} — статус: {response.status}, длина: {len(text)} символов")

async def main():
    async with aiohttp.ClientSession() as session:
        tasks = [fetch(session, url) for url in urls]
        await asyncio.gather(*tasks)

if __name__ == "__main__":
    asyncio.run(main())
```

Анализ результатов:

Запустите скрипт:
python main.py

1. Убедитесь, что загрузка страниц выполняется параллельно.
2. Обратите внимание на статус-коды и длину загруженных страниц.
3. Попробуйте добавить заведомо неработающий URL (например, https://badurl.test) и обработать ошибку через try/except.