**ДИПЛОМНАЯ РАБОТА**

**Анализ и сравнение написания web-приложений с использованием разных фреймворков**

[Введение 3](#_abv9yxhv8n3r)

[Обоснование выбора темы 3](#_9ltacpcfznhw)

[Определение целей и методов исследования 4](#_gqob9vsuts69)

[Основные понятия и определения 5](#_lhyszr18txj6)

[Проектирование веб-приложения 6](#_he6opr1tam7c)

[UI/UX 6](#_qh63op3zu4or)

[Шапка сайта 6](#_3g9mf5na1xwi)

[Главная страница 7](#_3j4zpi36r9ml)

[Страница авторизации 7](#_so3b3fr1hzmf)

[Страница регистрации нового пользователя 8](#_s8or9z2a3hgj)

[Страница основного функционала сайта 9](#_4tvvmvhvvdco)

[Хранение данных (база данных) 10](#_ilc4hseb0p7d)

[Таблицы для хранения данных 10](#_qx49y4cppj1q)

[Данные пользователя, обязательные поля (столбцы): 10](#_45ei5oo1wlh9)

[Данные об изображениях: 11](#_lo8zjgaqrh4v)

[Данные о распознанных объектах: 11](#_3l231dm37z7y)

[Маршрутизация 11](#_q4jt7iw5xlha)

[Бизнес-логика 12](#_8qfa3o8rige4)

[Порядок и особенности реализации 13](#_pmbdqjfhlsqj)

[Общая информация 13](#_vybxa5mv25h0)

[Django 13](#_plh7oqiwiess)

[FastAPI 13](#_rzexlmj3rxaj)

[Flask 13](#_s5opjotfdky5)

[Особенности фреймворков 14](#_yz8b469dabbu)

[Django 14](#_8gm1btdphcog)

[FastAPI 14](#_h479fpgkjb58)

[Flask 15](#_nr9a36cffykw)

[Анализ и интерпретация результатов 16](#_x3g1xe42eu6q)

[Flask 16](#_aa5k41nwpel8)

[Django 17](#_bqjlsswldxk)

[FastAPI 17](#_sziwwr64syma)

[Заключение 18](#_tqkquypj390y)

# Введение

## Обоснование выбора темы

Если начинать с начала, разработка любого продукта производится, чтобы им могли пользоваться люди. Веб-сайты или веб-приложения отличаются от других цифровых продуктов (будь то мобильные или десктопные приложения), в первую очередь доступностью (разумеется, при наличии соединения с сетью интернет) и кроссплатформенностью. В то же самое время они ограничены по ресурсоемкости (хотя бы потому, что подразумевают работу с большим количеством пользователей) и не могут обеспечивать достаточной оптимизации работы тяжелых приложений. Таким образом, веб-приложения (имеющие пользовательский интерфейс) в основном используются для целей информационного, коммуникативного характера, а также легковесных сервисных задач по обработке медиа-контента, например: новостные сайты, социальные сети, интернет-магазины, хранилища контента, конвертеры файлов из одного формата в другой и т.д. Учитывая предполагаемое направление развития современного общества, можно предположить, что в обозримой перспективе спрос на веб-приложения будет только увеличиваться, что определяет актуальность изучения и развития методов и способов их разработки.

Разработка веб-приложения включает, по меньшей мере, организацию пользовательского интерфейса (фронтэнд) и серверного функционала (бэкэнд). Фронтенд разработка в рамках данного проекта рассматриваться не будет, что же касается бэкенд разработки, то здесь мы сталкиваемся с рядом задач, которые необходимо решать при разработке практически любого веб-приложения, к примеру:

* Регистрация, авторизация и администрирование пользователей;
* Хранение пользовательских данных;
* Получение и передача данных пользователю;
* Маршрутизация и структурирование сайта;
* Обеспечение безопасности как самого ресурса, так и хранящихся пользовательских данных.

Таким образом, для бэкэнд разработки возникает спрос на инструментарий, позволяющий быстро и эффективно решать вышеперечисленные и иные задачи, определяемые структурой и назначением веб-приложения. На сегодняшний день существует немалое количество инструментов, созданных для решения этих задач. В рамках данного проекта будут рассмотрены только три из них, реализованных на базе языка Python, а именно:

* Django;
* Flask;
* FastAPI.

## Определение целей и методов исследования

Коль скоро темой проекта обозначен “анализ и сравнение фреймворков”, основной целью исследования является получение и систематизация информации об особенностях и ограничениях каждого из рассматриваемых фреймворков.

Для достижения целей исследования необходимо:

* Создать три python проекта, реализующих веб-сайт, по одному на каждый из рассматриваемых фреймворков;
* Обеспечить работу каждого проекта с заданным функционалом, а именно:
* Регистрация и авторизация пользователя, включающая получение данных от пользователя (обработку POST запроса), сохранение и обработку информации о пользователе в реляционной базе данных;
* Навигация по разделам сайта;
* Биснес-логика ограниченного функционала (например, распознавание объектов на картинке), включающая получение и обработку файлов от пользователя;
* В процессе разработки выявить особенности применения каждого из фреймворков;
* Провести анализ и сравнение использованных в процессе реализации соответствующего проекта модулей, приемов, методов.

Выбор метода разметки страниц сайта для целей настоящего исследования принципиального значения не имеет. Для удобства будет использован Bootstrap.

Производительность работы сайта в трех реализациях оцениваться не будет в связи с ограниченностью ресурсов (временных и технических) для исследования.

# 

# Основные понятия и определения

Фреймворк (Framework) - программная платформа, определяющая структуру программной системы;

Веб-приложение (веб-сайт, сайт) - клиент-серверное приложение, в котором клиент (пользователь) взаимодействует с веб-сервером при помощи браузера;

Браузер - прикладное программное обеспечение для просмотра веб-страниц;

Реляционная база данных (БД) - база данных, основанная на реляционной модели данных (в целях данного проекта, в упрощенной форме: данные хранятся во взаимосвязанных таблицах);

UI - user interface, пользовательский интерфейс, та часть веб-приложения, которая видна и доступна для взаимодействия конечному пользователю;

User Experience, UX - взаимодействие человека с веб-приложением, в целях проекта: описание действий пользователя и получаемого/ожидаемого результата;

Кликабельность - обеспечение возможности взаимодействия пользователя с объектом путем нажатия кнопки мыши и прикосновения к экрану для сенсорных устройств;

Предобученная модель - модель машинного обучения, которая была заранее обучена на большом наборе данных и может быть использована для решения задач распознавания объектов;

Валидация - проверка значения на соответствие необходимым условиям (например длина пароля, возрастные ограничения и т.п.);

ORM - Object-Relational Mapping или объектно-реляционное отображение, технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»;

CRUD - акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete);

# Проектирование веб-приложения

Раздел включает общее описание и концепцию работы разрабатываемого в рамках проекта веб-приложения.

В качестве функционала сайта будет реализовано:

*распознавание объектов на загруженной пользователем картинке* с применением предобученной модели MobileNet SSD, которая определяет следующие классы: "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor".

## UI/UX

Язык интерфейса сайта: английский. Локализация отсутствует.

### Шапка сайта

В верхней части каждой страницы сайта располагается меню, в котором отображаются:

* Название сайта;
* Меню:
* Для неавторизованного пользователя: Home | Login | Register;
* Для авторизованного пользователя: Home | Dashboard | Logout;

Каждый пункт меню кликабелен.

Переход по пунктам меню “Home” и \**Название сайта*\* приводит пользователя на главную страницу.

Переход по пункту меню “Login” приводит пользователя на страницу авторизации.

Переход по пункту меню “Register” приводит пользователя на страницу регистрации.

Переход по пункту меню “Dashboard” приводит авторизованного пользователя на страницу, реализующую основной функционал сайта.

Переход по пункту меню “Logout” приводит пользователя на главную страницу, меню меняется на вариант для неавторизованного пользователя.





### Главная страница

На главной странице сайта отображается приветственное сообщение:

*“Welcome to the Object Detection Platform*

*This is the home page of your application.”*

и ссылка:

*“Go to Dashboard”.*

При нажатии на ссылку осуществляется переход на страницу основного функционала сайта для авторизованных пользователей или на страницу авторизации - для неавторизованных.



### Страница авторизации

На странице авторизации представлены:

Заголовок: Login;

Поле ввода: Username;

Поле ввода: Password;

Кнопка: Login;

Ссылка: Register;

При нажатии на кнопку Login происходит валидация введенных данных в полях ввода. В случае введения валидных значений происходит проверка логина/пароля на наличие в списке зарегистрированных пользователей. В случае, если данные введены неверно, повторно открывается форма авторизации с дополнительным сообщением о неверно введенных данных. Если данные пользователя введены верно, осуществляется переход на страницу основного функционала сайта. Меню меняется на вариант для авторизованных пользователей.

При нажатии на ссылку Register происходит переход на страницу регистрации нового пользователя.



### Страница регистрации нового пользователя

На странице регистрации нового пользователя представлены:

Заголовок страницы: Register;

Поле ввода: Username;

Поле ввода: Password;

Информация о требованиях к паролю (длина, допустимые символы и т.д.);

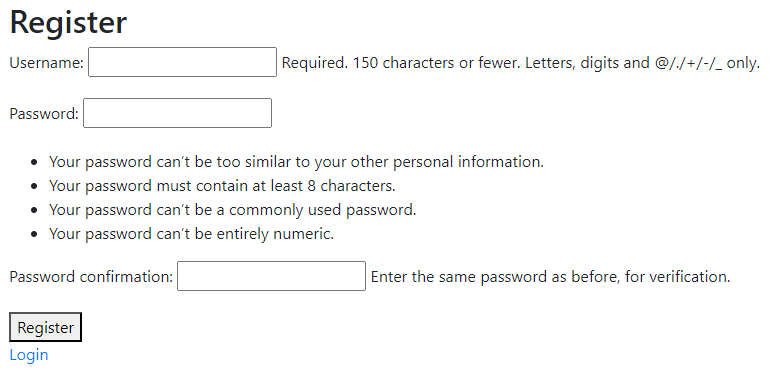
Поле ввода: Password confirmation;

Кнопка: Register;

Ссылка: Login.

При нажатии кнопки Register происходит первичная валидация введенных в поля данных. При успешной проверке происходит отправка формы на сервер, где данные проходят вторичную проверку (соответствие введенных паролей, наличие недопустимых символов и т.д.). Если проверка пользовательских данных на сервере не пройдена, осуществляется возврат к форме регистрации и появляется сообщение о том, какие необходимо внести изменения в данные нового пользователя. Если проверка данных прошла успешно, пользователь регистрируется в системе, происходит его автоматическая авторизация и переход на страницу основного функционала сайта.

При нажатии на ссылку “Login” осуществляется переход на страницу авторизации.

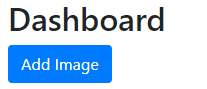


### Страница основного функционала сайта

На странице отображаются:

Заголовок: “Dashboard”;

Кнопка “Add image”;



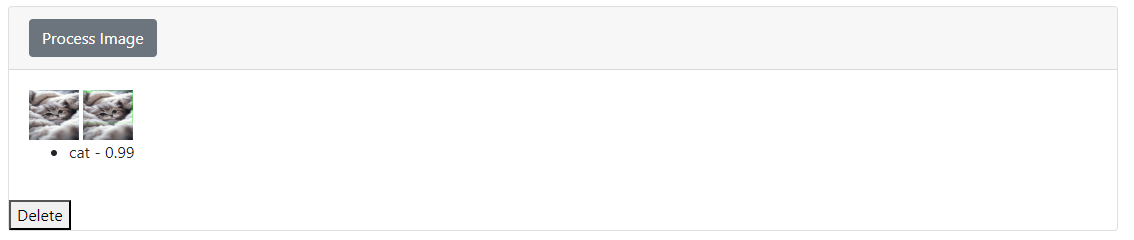
При наличии загруженных изображений также отображается отдельный блок для каждого изображения, в котором доступны:

Кнопка “Process Image”;

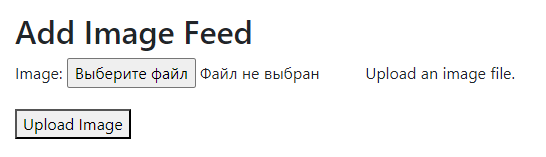
Уменьшенное оригинальное изображение;

Уменьшенное обработанное изображение (если обработка уже произведена);

Кнопка “Delete”.



При нажатии на кнопку “Add Image” осуществляется переход на страницу загрузки изображения (“Add Image Feed”), на которой в наличии кнопки выбора и загрузки изображения и информационные поля (название файла, заголовок страницы). При выборе неподдерживаемого формата файла, попытка его загрузить приводит к появлению информационного сообщения о несоответствии формата.



При успешной загрузке изображения осуществляется обратный переход на страницу основного функционала и появляется блок изображения.

При нажатии на кнопку “Process Image” в блоке изображения появляется второе изображение, на котором отображаются распознанные объекты, и информационное поле об уверенности модели в распознанном результате (например, “cat - 1.00).

При нажатии кнопки “Delete” блок изображения удаляется вместе с содержимым.

При нажатии на уменьшенное изображение, открывается новая страница с данным изображением в оригинальном размере.

## Хранение данных (база данных)

Для реализации цели хранения пользовательских данных будет использован SQLite3. Выбор обусловлен тем, что данная БД не требует наличия сервера, позволяет хранить всю базу локально на одном устройстве.

Для работы SQLite не нужны сторонние библиотеки или службы. Тем не менее, взаимодействие с БД будет осуществляться посредством ORM, поддерживаемой или встроенной в соответствующий фреймворк.

### Таблицы для хранения данных

#### Данные пользователя, обязательные поля (столбцы):

* id INTEGER PRIMARY KEY AUTOINCREMENT
* name VARCHAR(150) NOT NULL UNIQUE
* password VARCHAR(128) NOT NULL

Пароль может храниться как в зашифрованном, так и в натуральном виде.

Допускается применять автоматически создаваемые фреймворком таблицы для хранения данных о пользователе или стандартные модели, включенные во фреймворк или иную стороннюю библиотеку.

#### Данные об изображениях:

* id INTEGER PRIMARY KEY AUTOINCREMENT
* image VARCHAR(100) NOT NULL
* user\_id INTEGER NOT NULL REFERENCES “auth\_user”[[1]](#footnote-0) (“id”) DEFERRABLE INITIALLY DEFERRED
* processed\_image VARCHAR(100) NULL

Поля image и processed\_image будут хранить информацию о расположении соответствующих файлов изображений (первоначального и обработанного). Записи в таблице связаны с соответствующим пользователем (“один ко многим”).

#### Данные о распознанных объектах:

* id INTEGER PRIMARY KEY AUTOINCREMENT
* object\_type VARCHAR(100) NOT NULL
* confidence REAL NOT NULL
* location VARCHAR(255) NOT NULL
* image\_feed\_id BIGINT NOT NULL REFERENCES “imagefeed”[[2]](#footnote-1) (“id”) DEFERRABLE INITIALLY DEFERRED

Таблица связывается с таблицей информации об изображениях (“один ко многим”), здесь хранится информации о распознанных объектах и их расположении на соответствующей картинке.

## Маршрутизация

Главная страница: “/”;

Страница авторизации: “login/”;

Страница регистрации: “register/”;

Страница основного функционала: “dashboard/”;

Страница загрузки изображения: “add-image-feed/”;

Отмена авторизации пользователя (logout): “logout/”;

Обработка изображения c id = image\_id: “process/<int:feed\_id>/”;

Удаление изображения c id = image\_id: “image/delete/<int:image\_id>/”.

## Бизнес-логика

Для реализации основного функционала приложения используется предобученная модель MobileNet SSD (<https://github.com/chuanqi305/MobileNet-SSD>). Для работы с ней требуются сторонние библиотеки:

* numpy;
* opencv-python;

Логика работы модели описывается в функции process\_image(image\_id). Упрощенно, модель получает на вход изображение, на выходе предоставляет и обрабатывает данные:

* Наименование распознанных объектов и уверенность модели в результате;
* Расположение распознанных объектов на изображении;

Кроме того, распознанные объекты отмечаются рамкой на изображении (модифицированное изображение можно сохранить отдельно).

# Порядок и особенности реализации

## Общая информация

Все рассматриваемые фреймворки имеют широкую известность и поддержку в среде разработчиков веб-приложений. Сами авторы фреймворков позиционируют свои творения примерно следующим образом:

### Django

The Web framework for perfectionists with deadlines.

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

Иначе говоря, для тех, кто любит, чтобы все было идеально, т.е. в данном случае, имело жесткую и понятную структуру. Разумеется, понятную для тех, кому не лень детально изучить фреймворк, т.к. он накладывает ограничения на процесс разработки (расположение, именование файлов, порядок конфигурирования, использования и т.д.).

### FastAPI

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints.

The key features are:

Fast: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic). One of the fastest Python frameworks available.

Из тезисов выше можно сделать вывод, что в данном случае сделан акцент на высокую производительность готового приложения.

### Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.

А в данном случае мы имеем дело с “легковесным” фреймворком, позволяющим быстро реализовывать простые веб-приложения, но предусматривающим масштабирование и развитие в большой и сложный проект,

## Особенности фреймворков

### Django

Весь, необходимый для реализации нашего тестового приложения функционал, мы получаем “из коробки” (т.е. они встроены во фреймворк).

При запуске сервера, для обработки исходного кода, django использует встроенный python интерпретатор, настроенный на взаимодействие со структурой проекта, что в некоторых случаях (например, при импорте модулей) воспринимается стандартным инструментом анализа кода python, встроенным в IDE (например, PyCharm), как ошибки.

Для передачи данных в шаблоны html django использует собственный язык, схожий с jinja2, но не идентичный ему.

В базовый функционал веб-приложения встроена панель администрирования, реализующая визуальный интерфейс взаимодействия суперпользователя с базой данных и позволяющая указанный интерфейс настраивать.

### FastAPI

Структура проекта жестко не регламентируется, но гибко настраивается.

В наличии встроенный модуль APIRouter, позволяющий удобно реализовывать и систематизировать маршрутизацию.

Для запуска сервера и работы приложения (реализации тестового функционала) требуются сторонние библиотеки:

* uvicorn (запуск сервера);
* sqlalchemy (работа с БД);
* pydantic (валидация данных);
* alembic (миграция БД);
* jinja2 (для использования html шаблонов и DTL);
* python-multipart (аутентификация, получение данных из форм html);

Данные пользовательской сессии можно хранить в объекте встроенного класса Request, однако для организации безопасной аутентификации пользователя и для передачи чувствительных пользовательских данных используются иные методы хранения и передачи информации (в настоящем проекте не рассматриваются, т.к. выходят далеко за пределы его темы).

Для передачи данных с сервера в шаблон html используется модуль jinja2.

Все функции, реализующие маршрутизацию - асинхронные.

Фреймворк имеет встроенный интерактивный API (site\_address/docs, Swagger UI).

### Flask

Структура проекта никак не регламентируется.

Фреймворк имеет встроенный функционал для запуска сервера.

Для работы с flask существуют (в частности) адаптированные библиотеки flask\_login и flask\_sqlalchemy (для работы, соответственно, с авторизацией пользователя и базой данных).

Экземпляр приложения Flask() используется при инициализации базы данных и LoginManager-а.

Для передачи данные с сервера в шаблон html используется модуль jinja2.

# Анализ и интерпретация результатов

На начальном этапе реализации приложения наиболее быстрые решения нам предлагает Flask. Django требует предварительной настройки проекта, изучения его структуры и правил работы с ним. FastAPI более схож с Flask по методам решения поставленных задач. Причем данная схожесть проявляется не только в применении схожих подходов к решению задач (таких как декораторы-handler-ы запросов), но также и в использовании аналогичных внешних библиотек (например, sqlalchemy, jinja2).

Все фреймворки работают с шаблонизаторами, позволяющими удобно настраивать и формировать html страницы приложения, выстраивать в них структуру наследования и внедрять данные, полученные с сервера.

Все рассмотренные фреймворки имеют широкую известность среди разработчиков и хорошо задокументированы.

Отдельно про рассмотренные фреймворки хотелось бы отметить нижеследующее:

## Flask

Данный фреймворк позиционируется его же разработчиками как микро-(или легковесный) фреймворк, обеспечивающий быструю реализацию веб-приложения.

Одной из проблем Flask-а кажется жесткая привязка ключевых компонентов структуры приложения (например, базы данных) к экземпляру приложения (SQLAlchemy(Flask(\_\_name\_\_)), что, в свою очередь, вызывает ошибку циклического импорта при попытке выделить модели таблиц БД в отдельный модуль. Но это еще полбеды, - также в отдельный модуль (по той же причине) отказываются без танцев с бубнами выделяться и любые другие функции, задействующие экземпляр БД или приложения (например, обработчики маршрутизации, которые взаимодействуют с БД). Сам фреймворк предлагает для решения этой задачи использовать модуль Blueprint, но такое решение кажется не оптимальным.

Таким образом, разработка большого структурированного приложения (тем более командная, когда требуется раздавать задачи разным разработчикам, выстраивать спринты, сливать все это в систему контроля версий и т.д.) на Flask кажется неоправданной.

Тем не менее, для задачи реализации небольшого проекта, демонстрирующего потенциал стартапа, или иной веб-сервис, который не планируется развивать в большой высоконагруженный сервис, фреймворк вполне подходит.

## Django

Django напротив, требует соответствия проекта жесткой структуре расположения файлов проекта в строго определенных местах и соблюдения заранее установленных правил конфигурирования и взаимодействия между элементами структуры проекта. Данная особенность является частным случаем общей концепции, как будто лежащей в основе фреймворка: “изучи => подумай => еще подумай => сделай”. То есть фреймворк позволяет реализовывать проекты любой сложности, активно и независимо друг от друга задействовать разных разработчиков или даже команды разработчиков для решения задач реализации проекта в рамках отдельных модулей (одни пишут бизнес-логику, вторые - маршрутизацию, третьи - шаблоны, четвертые - базу данных и т.д.), но как будто ограничивает гибкость проекта. Особенно с учетом использования на сервере django собственного python интерпретатора, проект на django вроде как подразумевает реализацию проекта средствами django с минимальным взаимодействием с другими пакетами и модулями. Впрочем, в рамках реализации тестового функционала, django предоставляет всё необходимое (даже с избытком), а время, затраченное на его изучение, в перспективе с лихвой окупается временем, которое фреймворк позволяет сэкономить при развитии проекта (или реализации большого проекта).

Кроме того, отдельно хочется отметить панель администрирования, предоставляемую фреймворком. Данный функционал фреймворка позволяет настроить UI суперпользователя (администратора) сайта для взаимодействия с базой данных, включая CRUD записей, их фильтрацию, сортировку, анализ и т.д. Реализация одного только этого функционала средствами других фреймворков выглядит задачей, сопоставимой по сложности (если не более сложной), чем реализация основного функционала веб-приложения. Здесь же нам эта панель доступна “из коробки”.

## FastAPI

FastAPI производит впечатление фреймворка, разработанного с учетом накопленного опыта разработчиков, решавших задачи реализации проектов веб-приложений. С одной стороны, FastAPI предоставляет возможность быстрого старта проекта, а с другой стороны - возможность конфигурирования проекта и гибкого (без костылей) разделения на компоненты и модули. В дополнение прилагается интерфейс Swagger UI и асинхронность обработки запросов. Сами разработчики FastAPI позиционируют его, как самый быстрый фреймворк среди решений для веб-разработки и есть подозрение, что они не врут.

# 

# Заключение

По итогам реализации данного исследования можно заключить:

1. Каждый из рассмотренных фреймворков позволяет в полной мере реализовать тестовый функционал;
2. По субъективной оценке автора: наиболее быстро и легко реализовать тестовый функционал оказалось на базе фреймворка Flask;
3. Выбор фреймворка зависит от конкретных требований проекта: Django больше подходит для создания сложных приложений с многофункциональной архитектурой и жесткими требованиями к безопасности. Flask хорош для небольших проектов, предоставляя гибкость в организации кода, тогда как FastAPI справляется с созданием современных, высокопроизводительных API, позволяя, в числе прочего, эффективно работать с асинхронными задачами.

1. Название таблицы, в которой будут храниться данные о пользователях [↑](#footnote-ref-0)
2. Название таблицы, в которой будут храниться данные об изображениях [↑](#footnote-ref-1)