**Дипломная работа по теме: Анализ и сравнение написания web-приложений с использованием разных фреймворков.**

**Автор: Дидковский Николай Алексеевич**

**Оглавление:**

1. **Введение (2)**
2. **Основные понятия и определения (3)**
3. **Общее представление фреймворков**

**4. Веб-приложения на разных фреймворках**

**4.1 Фреймворк Flask**

**4.2 Фреймворк Django**

**4.3 Фреймворк FastAPI**

**5. Сравнение фреймворков**

**6. Заключение**

1. **Введение**

Трудно представить современного человека без домашнего компьютера или смартфона в руках. Он проводит там свой досуг, занимается работой, совершает покупки - перечисление может продолжаться до бесконечности, потому что развитие технологий позволяет, по средствам интернета, сделать практически что угодно, буквально не вставая с места.

И во всём этом ему помогают безграничное количество сайтов и веб-приложений, которые, в свою очередь, стараются быть всё более полезными и понятными. И как, если не при помощи кода и фреймворков мы, программисты будущие и настоящие, можем модифицировать и упрощать жизнь простым пользователям.

Из вышесказанного следует, что программисту в текущем времени всегда есть чем себя занять и применить свои навыки в самых различных направлениях. Начиная от разработки приложений "для себя" (ежедневников, блогов, счётчиков всего подряд) и заканчивая серьёзными, если не сказать, глобальными задачами, такие как сбор данных или обработка тысяч и миллионов строк продукции.

Современные компании всегда ищут более быстрые и эффективные пути взаимодействия с потребителем, ищут новые способы автоматизации и увеличения выгоды с чем, безусловно, им помогает интернет.

Для меня же, как для начинающего программиста, работа с фреймворками является основополагающим навыком, с развитием которого, я и сам смогу придумывать, разрабатывать и реализовывать эффективные и, вполне возможно, никем раньше невиданные приложения, которыми смогут пользоваться люди по всему миру. Рост и прогрессирование в этом направлении очень привлекает меня и я с надеждой смотрю в наше компьютеризированное будущее, от которого не только не сбежать, но и, на самом деле, очень полезно дружить и идти в ногу.

Исходя из этих тезисов, можно понять, что я крайне заинтересован в анализе и сравнении написания веб-приложений с использованием разных фреймворков.

Цель исследования: разработка простых приложений с использованием фреймворков Django, Flask и FastAPI

Задачи исследования: Обзор фреймворков для написания веб-приложений.

Сравнение эффективности фреймворков: сложность реализации, эффективность работы самого фреймворка, взаимодействие со сторонними пакетами и базами данных.

Реализация веб-приложений на которых можно будет проверить функционал, эффективность, плюсы и минусы каждого фреймворка

Написание дипломной работы: составление дипломной работы, включающей в себя введение, обзор фреймворков, обзор приложений, сравнение фреймворков на базе этих приложений, анализ результатов и выводы

Результаты исследования позволят оценить слабые и сильные стороны фреймворков, область их применения, а также относительную взаимозаменяемость в тех или иных случаях

1. **Основные понятия и определения**

**Фреймворк (Framework):**

набор готовых инструментов, библиотек и функций, которые упрощают разработку программного обеспечения. Он предоставляет разработчикам готовые решения для типичных задач, таких как работа с базами данных, веб-серверами, аутентификацией пользователей и т. д.

Среди самых популярных фреймворков находятся обозреваемые мной Django, Flask и FastAPI.

**Веб-приложение (Web Application):**

программа, которая работает через интернет-браузер и предоставляет пользователю определённые функции и услуги. Оно не требует установки на компьютер или мобильное устройство, поскольку все вычисления происходят на сервере, а пользователь получает только результат работы программы.

При помощи вебприложений написаных на основе различных фреймворков и будет проводиться аннализ последних в данной работе

**База данных:**

совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных/

**SQLite:**

это **компактная встроенная система управления базами данных (СУБД)**, для работы которой не нужен отдельный сервер. Все данные она хранит в одном файле и позволяет получать доступ к ним локально.

**PostgreSQL:** это **объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом**. Она позволяет гибко управлять базами данных: создавать, модифицировать или удалять записи, отправлять транзакцию — набор из нескольких последовательных запросов на особом языке запросов SQL.

**ORM (Object-Relational Mapping):**

технология программирования, которая позволяет работать с реляционными базами данных, используя объектно-ориентированный подход.

Основные функции ORM:

Отображение объектов на таблицы базы данных.

Преобразование запросов к базе данных в вызовы методов объектов.

Управление транзакциями и изменениями данных.

**Миграции (Migrations):**

процесс изменения структуры базы данных без потери данных. Они используются для отслеживания и применения изменений в схеме базы данных, таких как добавление, удаление или изменение таблиц и столбцов.

**Маршрутизация:**

процесс направления запросов к определённым функциям или методам в зависимости от заданных правил. Маршрутизация позволяет разработчикам организовать код и упростить его понимание, а также обеспечивает гибкость при обработке запросов.

**API (Application Programming Interface):**

набор определений и протоколов для создания и интеграции программного обеспечения. API позволяет разработчикам получать доступ к определённым функциям и данным другого приложения или сервиса, что упрощает разработку и обеспечивает стандартизированный способ взаимодействия между различными системами.

**URL (Uniform Resource Locator):**

**адрес ресурса в сети Интернет**. Он указывает местонахождение веб-страницы, картинки, видео или другого объекта и описывает способ его получения.

**Декораторы:**

это обертка вокруг функций (или классов) в Python, которая меняет способ работы этой функции. Декоратор абстрагирует свой собственный функционал.

**HTML (HyperText Markup Language): язык гипертекстовой разметки**, который используется для создания и структурирования веб-страниц. Он помогает определять, как содержимое страницы должно отображаться в браузерах. Иными словами, HTML — это своеобразный каркас, на котором строится страница, включая текст, изображения, ссылки и другие элементы.

**Рендеринг:**

автоматическая визуализация кода при помощи инструментов фреймворка

**Шаблонизатор:**

программа, которая, читая данные из одного источника (файла-шаблона), находит в нём специальные конструкции и согласно определённым правилам заменяет их на данные из другого источника.

**Jinja:**

**шаблонизатор для Python**. Он помогает создавать шаблоны с заполнителями, а затем заполнять заполнители данными для создания динамического контента

****CSRF (cross-site request forgery):****

**вид атак на сайт, при которой злоумышленник с помощью мошеннического сайта или скрипта заставляет браузер пользователя выполнять на доверенном сайте действия от его имени**.

**Валидация:**

это **процесс проверки, что продукт соответствует ожиданиям и потребностям пользователей**

**Хэш (Hash):**

это **строка фиксированной длины, которая создаётся из исходных данных с помощью хеш-функции.** Это уникальный цифровой отпечаток, который можно присвоить любому файлу.

1. **Описание фреймворков:**

**Общие положения по фреймворкам:**

Упрощение разработки. Фреймворки предоставляют готовые инструменты для выполнения типичных задач. Это ускоряет процесс разработки и уменьшает вероятность ошибок.

Повышение качества кода. Фреймворки следуют определённым стандартам и лучшим практикам, что помогает разработчикам писать более чистый и понятный код.

Масштабируемость. Фреймворки легко масштабируются, что позволяет разработчикам добавлять новые функции и поддерживать большие проекты.

Сообщество и поддержка. Фреймворки имеют большое сообщество разработчиков, которые создают плагины, расширения и документацию. Это облегчает поиск решений и поддержку.

Безопасность. Фреймворки часто включают в себя функции безопасности, такие как аутентификация, авторизация и защита от атак.

Тестирование. Фреймворки упрощают тестирование кода, предоставляя инструменты для написания тестов и проверки функциональности.

**Django:**

бесплатный и открытый веб-фреймворк, который позволяет создавать надёжные и масштабируемые веб-сайты. Он предоставляет разработчикам инструменты для быстрого создания динамических веб-приложений с использованием готовых компонентов и библиотек.

**Flask:**

микрофреймворк для создания веб-приложений. Он предоставляет базовые функции и инструменты, необходимые для разработки веб-сайтов и приложений, но не включает в себя множество дополнительных функций, которые есть в более крупных фреймворках.

**FastAPI:**

современный веб-фреймворк для создания API, использующий декораторы, аннотации типов и интроспекцию кода, что позволяет уменьшить количество шаблонного кода в веб-приложении.

1. **Веб-приложения на разных фреймворках:**

**Установка пакетов:**

В Python пакеты, такие как Django, Flask, FastAPI многие другие доступны в общедоступном репозитории, откуда любой может их скачать и установить. Официальный репозиторий пакетов называется PyPI (Python Package Index). Устанавливать пакеты из PyPI просто, так как Python поставляется с инструментом под названием pip в комплекте.

При помощи команды pip install нужный пакет загружается, устанавливается и с этого момента каждый скрипт Pyton имеет к нему доступ.

Дабы избежать конфликтов, например, связанных с различными версиями инструментария, на котором было написано ваше веб-приложение (работало на старой версии, но полностью сломалось на новой и т.п.) существуют виртуальные среды. Виртуальная среда - это полная копия интерпретатора Python, изолированная от основного, что делает её идеальной площадкой для экспериментов без страха затронуть работоспособность ваших старых приложений и инструментария (часто новые версии пакетов плохо взаимодействуют со старыми, особенно в первое время после релиза)

Создав новое виртуальное окружение, активировав его и установив в него все необходимые пакеты я приступаю к написанию веб приложений.

**4.1 Flask:**

В качестве образца для анализа я выбрал веб-приложение в виде небольшого блога, которое имеет страницы входа, регистрации, отображения профиля и написанных сообщений мной или, образными, другими пользователями.

В папке приложения, в файле \_\_init\_\_.py, который показывает Python, что подкаталог считается пакетом и может быть импортирован есть одна особенность , которая позволяет избежать циклического импорта, распространённой проблемы приложений с использованием Flask: импортирование модулей внизу файла, после всего написанного кода. Так как модулю routes необходимо импортировать переменную app, но у нас уже есть пакет под названием app, расположение импорта внизу позволяет избежать ошибки, возникающей в результате взаимных ссылок между этими двумя этими файлами.

В модуле routes находятся функции, которые являются обработчиками маршрутов разных URL-адресов. Они показывают, какую логику следует выполнять Flask в моменты обращения клиента к указанному URL.

Функции в этом модуле обёрнуты в декораторы, дабы создать связь между ними и URL, указанным в качестве аргумента (при запросе адреса, вызывается декорируемая фунция).

Перед запуском приложения, чтобы Flask знал, как его импортировать, устанавливаем переменную среды FLASK-APP. Для упрощения работы и автоматического её использования , существует пакет python-dotenv для регистрации FLASK-APP. В файл .flaskenv записывается FLASK-APP=blog.py (blog.py - название файла, которое определяет экземляр приложения)

Запускается приложение Flask командой flask run. Команда настраивает веб-сервер, который пересылает запросы приложению, указанному в FLASK-APP

В папке templates хранятся шаблоны HTML-страниц, которые нужны для отображения полноценных страниц в браузере. Для рендеринга шаблона Flask использует функцию render\_templates. Эта функция вызывает Jinja, который идёт в комплекте с фреймворком Flask. С его помощью можно подставлять аргументы, переданные с вызовом render\_template, в блоки HTML кода обозначенные как {{ ... }}

В Jinja есть функция наследования шаблонов, что позволяет создать один базовый шаблон и на его основе создавать общий. Это упрощает понимание кода и избавляет от его множественного копирования. Также Jinja поддерживает условные опреаторы и циклы, типичные для Python, такие как if, else, for. Они задаются внутри блоков {% ... %} и должны иметь завершающий блок, например {% endif %}

Базовым шаблоном в моём приложении является base.html, внутри которого используется оператор {% block ... %} позволяющий вставлять на своё место шаблоны с тем же именем из производных страниц.

Большинство расширений Flask используют соглашение flask\_имя\_объекта об именовании для объектов импорта верхнего уровня.

Для работы с веб-формами используется Flask-WTF. Flask-WTF - это расширение Flask, интегрирующее библиотеку WTForms, которая предоставляет полезные функции для простого создания и обработки форм в веб-приложении Flask. Подобные расширения являются важной частью экосистемы Flask и позволяют решать множество вопросов, на которые базовый фреймворк ответа не имеет. Расширения Flask являются обычными пакетами python и устанавливаются при помощи pip

Так как веб-приложения могут быть очень объемными, грамотная организация структуры и хранения переменных очень важна. Для этого задачи лучше всего разделять и именно поэтому в приложении вынесены в отдельные файлы и директории шаблоны, файлы с моделями, формами и т.п. По этой же причине создан файл config.py и в класс Config внутри этого файла для определения параметров конфигурации. Применение этого файла происходит по средствам метода app.config.from\_object().

Переменная конфигурации SECRET\_KEY является важной частью большинства приложений Flask. Расширение Flask-WTF использует его для защиты веб-форм от вредоносной атаки CSRF.

Расширение Flask-WTF использует классы Python для представления веб-форм. Класс Form определяет поля формы, как переменные класса. Они импортируются непосредственно из пакета WTForms, поскольку расширение Flask-WTF не предоставляет настраиваемых версий.

Поля определённые в классе LoginForm без сторонней помощи отображаются в формате HTML и аргумент form, созданный из этого самого класса будет отправлен функцией просмотра входа в систему

Параметр шаблона form.hidden\_tag() генерирует скрытое поле, включающее токен, который используется для защиты формы от CSRF-атак.

Для отправки данных формы на сервер используется метод POST

Функция flash() - полезный способ показать сообщение пользователю. Многие приложения используют этот метод, чтобы сообщить пользователю, было ли какое-то действие успешным или нет. Функция get\_flashed\_messages() исходит из Flask и возвращает список всех сообщений, которые были зарегистрированы ранее в flash(). Функция redirect() автоматически перенаправляет пользователя на страницу, указанную в качестве аргумента

Валидаторы, указанные в форме входа пользователя или в функции просмотра при получении данных на вход, предотвращают ошибочные действия и приём неверных данных, оповещая пользователя функцией flash о том, что он сделал неверно

Чтобы лучше контролировать ссылки, используемые в шаблонах, Flask предоставляет функцию с именем url\_for(), которая генерирует URL-адреса, используя свое внутреннее отображение URL-адресов для просмотра функций. Аргументом для url\_for() является имя конечной точки, которое является именем функции просмотра.

Flask изначально не поддерживает базы данных, что предоставляет разработчикам свободу выбора. Расширение Flask-SQLAlchemy предоставляет взаимодействие Flask для популярного пакета SQLAlchemy, который представляет собой ORM, поддерживающее такие движки баз данных, как MySQL, PostgreSQL или выбранный мной SQLite. Выбор на него пал из-за простоты и отсутствия необходимости в сервере, что отлично подходит для простого приложения.

При помощи расширения Flask-Migrate, что является оболочкой Flask для Alembic - платформы для миграции баз данных для SQLAlchemy.

Расширение Flask-SQLAlchemy определяет местоположение базы данных приложения из переменной конфигурации SQLALCHEMY\_DATABASE\_URI

Данные, которые будут храниться в базе данных, будут представлены набором классов, обычно называемых моделями базы данных. Уровень ORM в SQLAlchemy выполнит преобразования, необходимые для отображения объектов, созданных из этих классов, в строки в соответствующих таблицах базы данных.

Каждой единице в базе данных будет присвоено уникальное значение id, которое, обычно обозначается в качестве первичного ключа. Остальные же поля моделей определяются в зависимости от нужд приложения.

Модуль sqlalchemy включает в себя функции базы данных общего назначения и классы, такие как типы и помощники по построению запросов, в то время как sqlalchemy.orm обеспечивает поддержку использования моделей.

Созданные мной классы моделей наследуется от db.Model, базовый класс для всех моделей из Flask-SQLAlchemy. Объявление типа, такое как so.Mapped[int] или so.Mapped[str] определит тип столбца, а также указывает обязательность значения, а чтобы определить столбец, который может быть пустым, из Python добавлен модуль Optional

Репозиторий миграции для приложения создается при помощи команды flask db init. Команда flask db migrate генерирует автоматические миграции со всеми созданными ранее моделями. Alembic, на базе которой стоит Flask-Migrate внесет изменения таким образом, чтобы не требовалось воссоздавать базу данных с нуля каждый раз, когда вносятся изменения. Для примения изменений к базе данных используется команда flask db upgrade.

Связь между сущностями бывает нескольких типов: OneToOne, OneToMany, ManyToMany. Проявляется свзяь OneToMany при помощи добавления ссылки на id пользователя, для этого создается поле с внешним ключом (ForeignKey) и объявляется взаимосвязь методом relationship

Werkzeug - пакет, реализующий хеширование паролей для увеличения безопасности приложений

Расширение Flask-Login управляет состоянием входа пользователя в систему. Flask-Login предоставляет класс с именем UserMixin, который включает безопасные реализации, подходящие для большинства классов пользовательских моделей. Одним из обязательных методов для корректной работы Flask-Login является метод is\_authenticated, который я буду использовать для проверки входа пользователя в систему

Flask-Login отслеживает вошедшего в систему пользователя, сохраняя его уникальный идентификатор в сеансе пользователя Flask, хранилище, назначенное каждому пользователю, который подключается к приложению. Каждый раз, когда вошедший в систему пользователь переходит на новую страницу, Flask-Login извлекает идентификатор пользователя из сеанса, а затем загружает этого пользователя в память.

Flask-Login не умеет самостоятельно взаимодействовать с базами данных, поэтому для него нужно настроить функцию загрузки пользователя (@login.user\_loader)

Переменная current\_user берется из Flask-Login и используется в любое время для получения объекта user.

Для загрузки пользователя в систему необходимо достать его из базы данных. Используется поиск по имени, проверка пароля и затем, при помощи функиции login\_user() из Flask\_Login, пользователь регистрируется, как вошедший в систему. С помощью logout\_user() можно выйти из приложения.

Flask-Login предоставляет функцию, которая заставляет пользователей входить в систему, прежде чем они смогут просматривать определенные страницы приложения. Делается это при помощи декоратора @login\_required

Регистрационная форма строится по тому же принципу, что и форма входа с некоторыми отличиями: используется email-validator из WТForms для проверки корректности ввода пользователем имейла. Также добавляются функции имеющие форму validate\_<имя\_поля>, которые WTForms использует в качестве пользовательских проверок.

URL-компонент, который окружен знаками < и > является динамическим, что позволяет Flask при запросе подставить любое значение, указанное в клиентском браузере.

Оператор include в Jinja позволяет встроить один шаблон в другой целиком, что бывает полезно, если мы хотим отобразить одну и ту же (чаще всего небольшую) информацию на разных страницах без изменений.

Декоратор @before\_request из Flask регистрирует функцию, которая будет выполняться непосредственно перед функцией просмотра. Таким образом можно вставлять код, который необходимо выполнить перед любой функцией просмотра в приложении, что полезно для определения, например, последней активности пользователя.

Flask-SQLAlchemy изначально поддерживает разбивку на страницы с помощью функции db.paginate(), которая работает аналогично db.session.scalars(), но со встроенным разбиением на страницы. Возвращаемое значение из db.paginate() является объектом Pagination. Пагинация — это способ организации большого объема данных на веб-сайте путем разделения их на отдельные страницы для удобства навигации пользователей. В файле config.py можно записать общее значение страниц (постов) для всего приложения.

Объект класса Pagination имеет ещё несколько полезных атрибутов: has\_next: Возвращает True если после текущей страницы есть еще хотя бы одна страница, has\_prev: Возвращает True если перед текущей страницей есть хотя бы еще одна страница, next\_num: Возвращает номер страницы для следующей страницы, prev\_num: Возвращает номер страницы для предыдущей страницы

**4.2 Django:**

В качестве образца для анализа выступает простое веб-приложение с вопросами, которое имеет страницы входа, регистрации, страницы со списком вопросов, страницами конкретных вопросов и ответов на них.

После создания виртуального окружения, его активации и установки django через pip, создаётся проект командой django-admin startproject <имя\_проекта> и приложение командой python manage.py startapp <имя\_приложения>.

Все представления, которые обеспечивают логику приложения, пишутся в файле views.py



После создания функций необходимо не забыть указать маршруты к страницам, которые будут рендериться этими самыми функциями, в файле urls.py

В файле settings.py очень важно подключить наше приложение и указать путь к папке с шаблонами. Для создания миграций Django также использует Alembic. После процесса миграции можно создать учётную запись администратора, которая облегчит работу с данными нашего приложения, командой python manage.py createsuperuser. Чтобы модель для создания новых записей отображалась в панели администрирования, ее нужно зарегистрировать в файле admin.py.

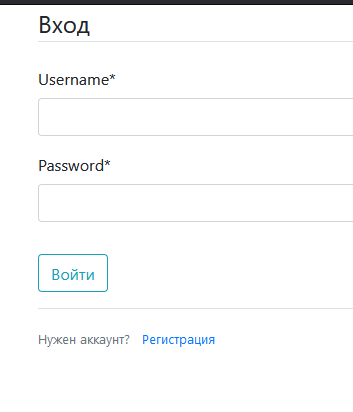
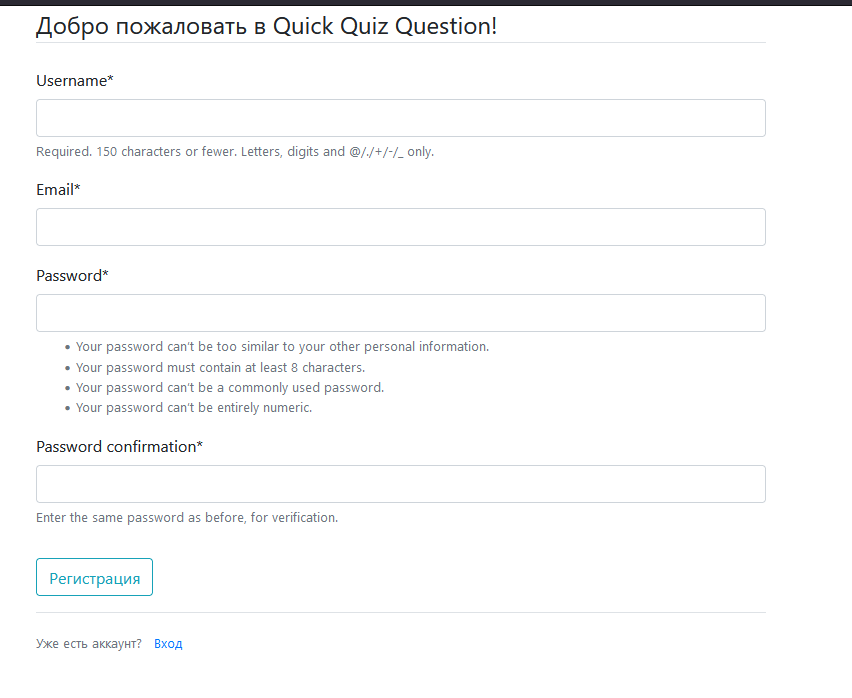
Все шаблоны HTML хранятся в папке templates и наследуются при помощи тега {% extends %} от base.html

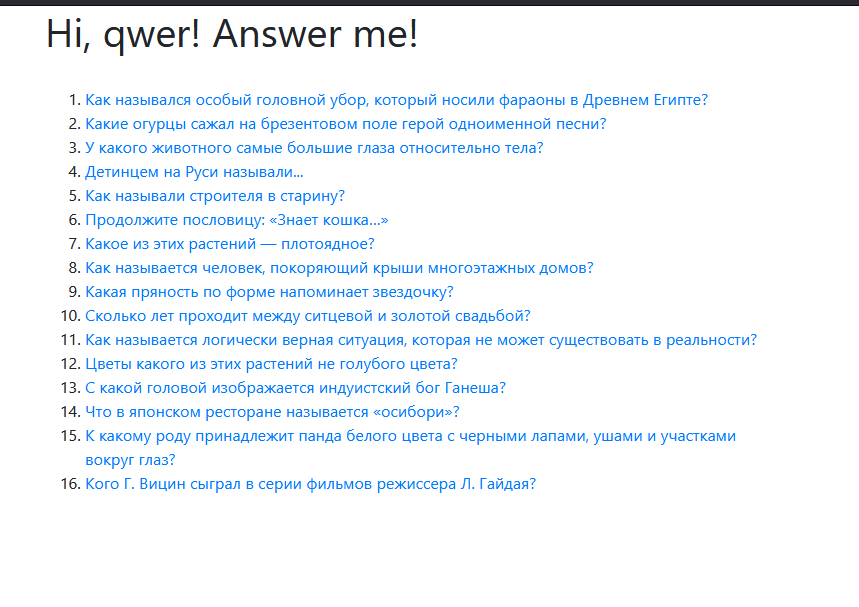
В models.py хранятся все модели при помощи которых будут создаваться объекты вопросов и ответов.

Единственная форма, использованная в приложении - форма для регистрации пользователя.

В папке static хранится файл со стилями, благодаря которым можно настроить внешний вид приложения по своему усмотрению.

Страницы регистрации и входа:



После регистрации и входа пользователь сохраняется в базе данных и ему даётся доступ к страницам, ранее закрытым при помощи декоратора @login\_requierd

Нажав на любой из вопросов пользователь перейдёт на страницу этого вопроса, где сможет увидеть варианты ответов и подсчёт количества этих самых ответов.

