МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра веб-технологий и компьютерного моделирования

Курсовая работа

Разработка серверной часть с использованием веб-фреймворка Django

Выполнил:  
Студент 3-ого курса 3-ей группы  
Трифонов Вадим  
  
  
Научный руководитель:  
Доцент, кандидат физ.-мат. наук  
Кремень E.B.

Минск, 2020

Оглавление

[Оглавление 2](#_Toc41529047)

[Введение 3](#_Toc41529048)

[Веб-фреймворки 4](#_Toc41529049)

[Структура проекта 6](#_Toc41529050)

[Формирование веб-страницы 17](#_Toc41529051)

[Создание и реадиктирование Веб-страниц 20](#_Toc41529052)

[Авторизация пользователй и Разделение доступа 25](#_Toc41529053)

[Заключение 30](#_Toc41529054)

[Литература 31](#_Toc41529055)

Введение

Каждый день на просторах интернета появляется всё больше веб-ресурсов, различный платформ и информационных площадок, в связи с этим знания и умения работать с данными, уметь их обрабатывать, ограничивать доступ и защищать становятся всё актуальнее.  
  
Основная цель моей курсовой работы – максимально подробно изложить разработку серверной части на языке Python с помощью его фреймворка Django, на примере создания веб-сервиса, который будет иметь следующие функциональные особенности: авторизация, разделение доступа к взаимодействию с базой данных, размещение файлов. Каждая глава отделяет логические этапы разработки проекта, в каждой будут описаны основные методы и способы проектирования структуры вер-сервиса, опуская такие вещи как: установка Django и Python, запись необходимых директорий, мелки настройки проекта для правильной работы на операционной системе Windows, создание стилей для отображения веб-страницы.   
  
Выбор фреймворка Django связан с его функциональными возможностями, а так же большим инструментарием идущим “из коробки”, например: Миграции баз данных, формы, админ-панель, ORM (объектно-реляционное отображение). А также при создании проекта на Django нет необходимости реализовывать защиту от таких видов атак как SQL-инъекций (XSS) и подделки межсайтовых запросов (CSRF), механизмы осуществляющие защиту от таких атак встроены в структуру Django.

В ходе написания курсовой работы были использованы следующие средства:  
 Язык программирования Python версии 3.8  
 Веб-фреймворк Django версии 3.0.6  
 Язык гипертекстовой разметки HTML  
 CSS – стили Bootstrap версии 3.4.1

Веб-фреймворки

В настоящее время огромная часть информации расположена в интернете, более того, с каждым днём ее становится всё больше и больше. Эти данные могут иметь общий доступ (доступ к этим данным может получить любой желающий, умеющий пользоваться поисковиком), закрытый (защищённый от всех посетителей при помощи шифрования и дополнительных защитных служб) или же просто ограниченный (например, только для авторизовавшихся пользователей). Любой из упомянутых структур необходимо управлять, как-то защищать, передавать, получать и т.д. Данные аспекты проектируются и создаются на этапе разработки веб-ресурса, на котором будет располагаться эта информация. Данная ветвь разработки называется серверной, для её проектировки используются различные языки программирования, такие как: PHP, Python, Java, Ruby, Go и другие. Большее удобство привносят веб-фреймворки – специально созданные архитектуры, предназначенные только для одного язык программирования, позволяющие грамотно и быстро реализовать ваше веб-приложение. Они используются как для серверной части, так и для клиентской (часть, отвечающая за отображение и обработку информации на стороне конечного пользователя). Вот примеры таких:

Серверная часть:

* Django — Python;
* Zend — PHP;
* Ruby on Rails — Ruby.

Клиентская часть:

* Angular;
* Ember.js;
* Vue.js.

Так же существую многофункциональные веб-фреймворки, так называемые full-stack фреймворки, которые позволяют разработать клиентскую и серверную часть одновременно, один из таких – Meteor, написанный на языке JavaScript.   
  
Однако какими бы разными эти фреймворки не казались, архитектура заложенная в них одна, называется она MVC — Модель, Представление и Контроллер (Model-View-Controller) [1].   
Разберем каждую из составляющих:  
Модель содержит в себе все данные, её правила и функции, то есть является реализацией хранения, доступа, обработки данных  
Представление отвечает за отображение информации (чаще всего в виде HTTP ответа)  
Контроллер же “просит” две другие составляющее предоставить, обменяться или же обработать какие-то данные.  
Схема представлена на *Рис.1*

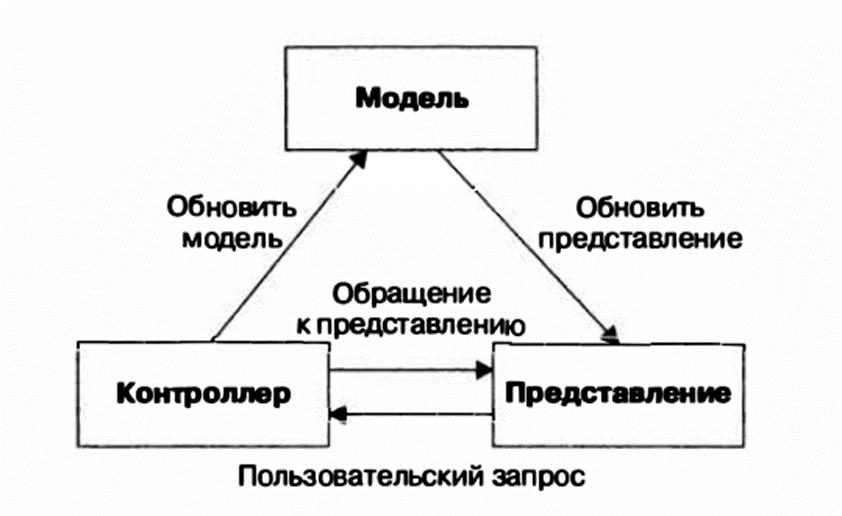


Рисунок 1

Структура проекта

Для начала разработки проекта необходимо создать рабочее пространство, сделать это можно следующим образом:

1. Открыть командную строку (для Windows)
2. Перейти в директорию в которой будет располагаться наш проект
3. Вызвать команду Django-admin startproject Website (название)

После выполнения этих шагов в нашем каталоге появится следующая структура *(Рис.2):*

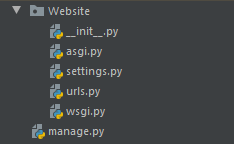


Рисунок 2

Разберём назначение каждого из них:

* Manage.py – файл утилита для создания миграций, запуска сервера, добавление “суперпользователя” и др.
* \_\_init\_\_ - сообщает интерпретатору Python, что данная директория является модулем
* Asgi.py и wsig.py – модули для связи проекта с веб-сервером
* Settings.py – файл, содержащий основные настройки рабочего пространства: подключения баз данных, пути к приложениям, языковые параметры и т.д.
* Urls.py – модуль содержащий маршруты проекта

Для проверки работоспособности созданного нами пространства, запустим встроенный тестовый сервер, сделать это можно следующим образом: в командной строке вызвать команду python manage.py runserver. После выполнения данной команды на вашем устройстве запустится тестовый сервер с локальным ip адресом на 8000 порту (для изменения адреса или порта необходимо внести правки в файл setting.py)

Если всё было установлено правильно тестовый сервер будет выглядеть следующим образом (*Рис.3*) :

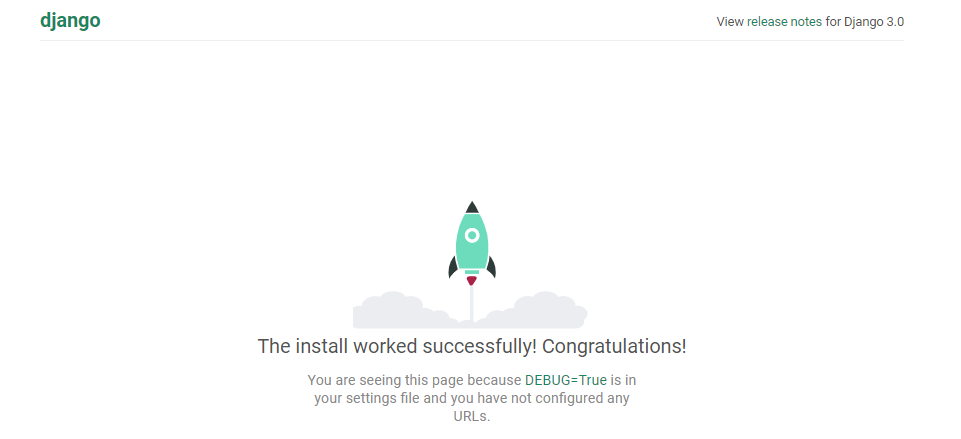


Рисунок 3

Также после запуска сервера в корневой папке проекта появится новый файл – db.sqlite3 – файл базы данный, по умолчанию Django использует SQLite.

За функциональную часть проекта отвечают “приложения”, “приложения” – фрагмент независимый от других приложений, реализующий работу с базами данных, их обработку и представление. Для узконаправленных сайтов достаточно одного приложения (например блог, мессенджер и т.д.), для более крупных веб-сервисов (например интернет-магазин со своей новостной лентой) одного будет недостаточно. [1]

Для создания приложения достаточно вызвать команду: python manage.py startapp WSApp (название приложения). В корневой директории утилита manage.py создала каталог, содержащий наше приложение (*Рис.4)*:

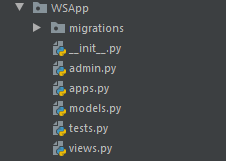


Рисунок 4

Рассмотрим их назначения:

* Migrations – папка вложенного модуля, в котором будут сохраняться пакеты сгенерированных миграций
* Admin.py – модуль настроек административной панели, автоматически подключаемая и работающая сразу с всеми зарегистрированными приложениями
* Apps.py – модуль с настройками текущего приложения
* Models.py – файл хранящий в себе модели проекта
* Tests.py – содержит тестирующие процедуры
* Views.py – модуль с контроллерами приложения

Для регистрации нашего приложения во всем проекте, необходимо объявить его в файле settings.py в блоке INSTALLED\_APPS (*Рис.5*):

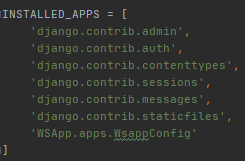


Рисунок 5

После регистрации приложения перейдем к формированию ответа на запрос пользователя, за это отвечают контроллеры. Поступивший запрос обрабатывается сервером, проверяется на соответствие с маршрутами (задаются в файле urls.py) и, если совпадение найдено, передаётся соответствующему контроллеру, а они, в свою очередь, описаны в файле views.py. Для более грамотной маршрутизации внутри проекта, желательно в приложении создать модуль urls.py и прописывать внутренние маршруты в нем, это позволяет легко и быстро переносить ваши приложения между различными проектами. После создания такого файла, перейдем к контроллеру, Django позволяет создать контроллер в двух вариантах: в виде функции (контроллер-функция) или же в виде класса (контроллер-класс).

Перейдем к редактированию файла views.py, опишем базовый контроллер-функцию, создадим ответ на пользовательский запрос и зарегистрируем маршрут запроса. В файле views.py опишем контроллер (*Рис.6*):

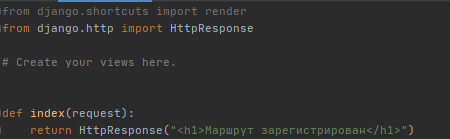


Рисунок 6

А также создаем маршрут для нашего контроллера в модуле urls.py (*Рис.7*):

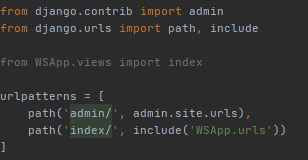


Рисунок 7

Запустим отладочный сервер для проверки правильности работы контроллера и маршрута (*Рис.8*):

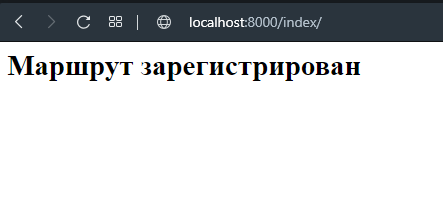


Рисунок 8

Далее перейдем к описанию моделей, благодаря которым создаются миграции (в папке migration) на их основе проектируется структура базы данных, а также отслеживаются изменения в файле models.py (работает как система контроля версий, но для базы данных). Модель в Django – средство представления таблицы базы данных с помощью средств языка Python, она представляется в виде класса, а отдельный его экземпляр - запись из советующей таблицы. Реализация моделей происходит в файле models.py. [4] При проектировке модели стоит учитывать, что все описанные поля, по умолчанию являются обязательными для того, чтобы указать, что поле необязательное к заполнению необходимо указать параметр: blank = True.

Создадим модель, обеспечивающая структуру данных статьям, которые будут добавляться в проект пользователем, для этого определим поля для формирования статьи, а так же тип данных, необходимый для их хранения:

* Id – поле, содержащее уникальный идентификатор
* Title – заголовок статьи
* Created\_date – дата создания статьи
* Content – содержимое статьи
* Files – прикреплённые к статье файлы

Опишем данную модель в файле models.py (*Рис.9*):

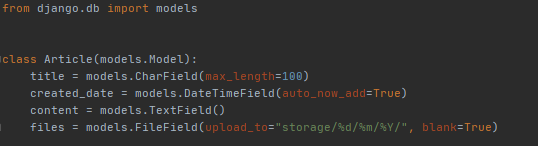


Рисунок 9

Модель мы представили в виде класса, который наследуется от импортируемого класса Model. Можем заметить, что мы не создавали поле id, при миграции это поле будет создано автоматически с типом данных Integer, также оно будет являться первичным ключом и авто инкрементироваться. Описываемые типы данных отличаются от типов данных языка Python, однако имеют некоторую схожесть с SQL.

Рассмотрим описанные поля: title имеет тип данных CharField ( на деле, это лишь вызов специальной функции, которая создаёт SQL код для создания поля), имеющая обязательный параметр max\_length, чаще всего используется для небольших строк; created\_date – DateTimeField не имеет обязательных параметров, однако существуют такие аргументы как auto\_now\_add, который заполняет поле датой и временем создания экземпляра, но при редактировании существующего объекта не меняется, auto\_now работает по тому же принципу, но обновляется при редактировании; content – TextField – поле может содержать большой объем текста; files – FileField – позволяет загружать файлы, обязательный параметр upload\_to – директория, куда будут загружаться файлы, также для того, чтобы через год существования нашего вебсайта данный каталог не был перегружен огромным количеством файлов, можно использовать специальные маркеры %d – текущий день, %m – месяц, %y – год, при такой записи, будет создаваться отдельная папка под каждый новый день, если в один день будут загружены файлы с один именем, Django автоматически их переименует;

Теперь, когда модель базы данных создана, перейдем к миграции. Для создания файла миграции необходимо воспользоваться manage.py, обратиться к нему через консоль: python manage.py makemigrations

После выполнение данной команды в папке migration появится следующий файл 0001\_initial.py со следующей структурой (*Рис.10*):

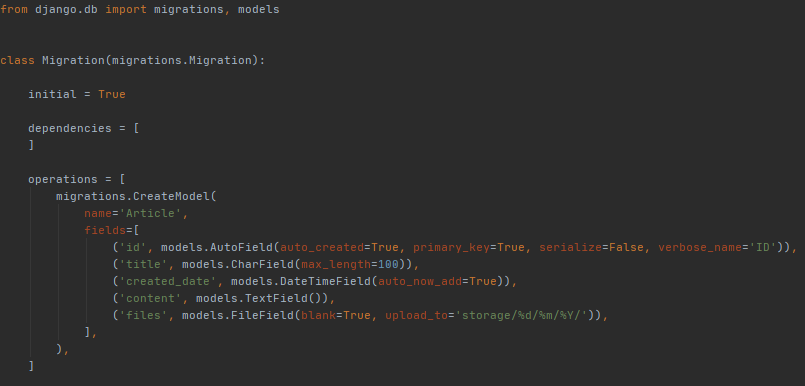


Рисунок 10

Теперь необходимо применить созданную миграцию, для этого вызываем команду: python manage.py migrate

При первом выполнении данной команды так же выполнится миграция встроенных файлов, необходимых для работы административной панели. Теперь файл в корневой директории проекта db.sqlite3 имеет созданную нами структуру (*Рис.11*):

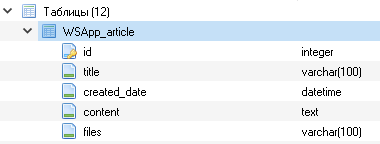


Рисунок 11

Для добавления объектов в базу данных можно использовать админ панель, для того, чтобы её запустить необходимо создать суперпользователя, для этого в командной строке пропишем: python manage.py createsuperuser, далее заполняем необходимые данные. После создания суперпользователя можем воспользоваться админ панелью, для этого необходимо запустить веб-сервер и к его адресу добавить /admin. Далее заполняем данные, которые использовали при создании суперпользователя [5].

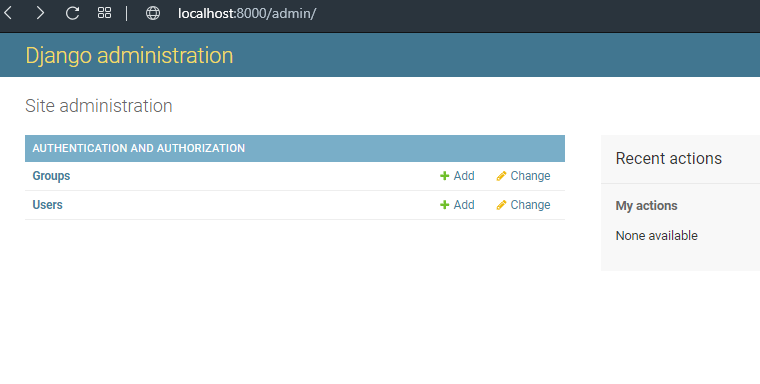


Рисунок 12

Административная панель (*Рис.12*) позволяет редактировать и добавлять новые информационные блоки в имеющиеся базы данных, однако, на данный момент, созданной нами модели Article нет в списке, необходимо ее зарегистрировать в модуле admin.py, добавив в него следующую строку admin.site.register(Article)

Теперь можем редактировать созданную базу данных (*Рис.13 и Рис.14*):

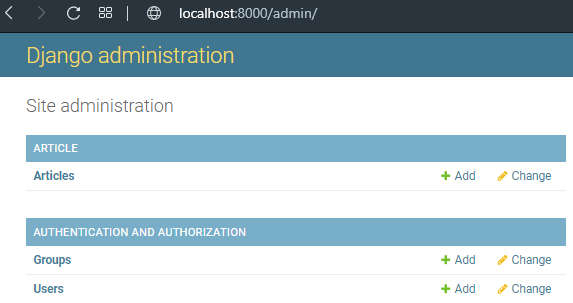


Рисунок 13

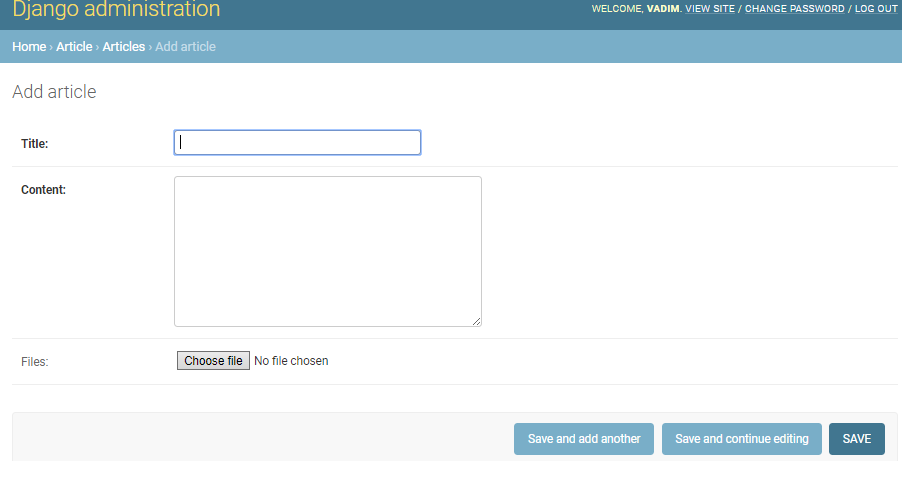


Рисунок 14

Для примера создадим несколько объектов для нашей базы данных, для некоторых прикрепим файл с один именем и посмотрим содержимое базы данных (*Рис.15 и Рис.16*):

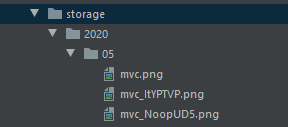


Рисунок 15

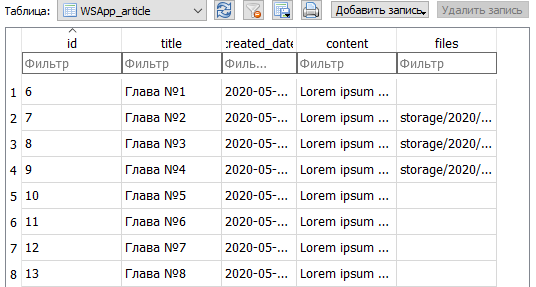


Рисунок 16

После проектировки моделей и создание контроллера необходимо отобразить пользователю по его запросу элементы базы данных, для этой цели используется шаблоны и шаблонизатор встроенный в Django. Ранее мы создали и связали маршрутом контроллер index, вернемся к нему и свяжем его с файлом index.html и дадим ему доступ к нашей базе данных, для этого создадим каталог templates и расположим в нем структуру обычной веб-странички(*Рис.18*):

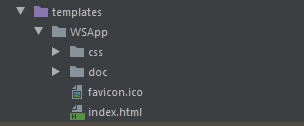


Рисунок 17

Обеспечим доступ к элементам базы данных(*Рис.18*):

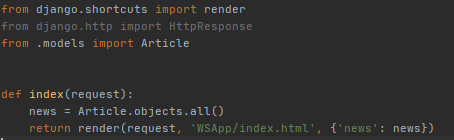


Рисунок 18

Доступ предоставляется не к всем таблицам и полям, а только тем, которые мы передадим при помощи словаря Python. Теперь в файле index.html возможно использовать специальные директивы и теги, которые обрабатываются шаблонизатором следующим образом: клиентский запрос, прошедший маршрутизацию по проекту и, согласно нашему котроллеру, запросит файл index.html, который перед передачей, в качестве ответа пользователю, обработается шаблонизатором, заполнит страницу данными и передаст серверу.

HTML файл index.htlm имеет следующую структуру (*Рис.19 и Рис.20*):



Рисунок 19

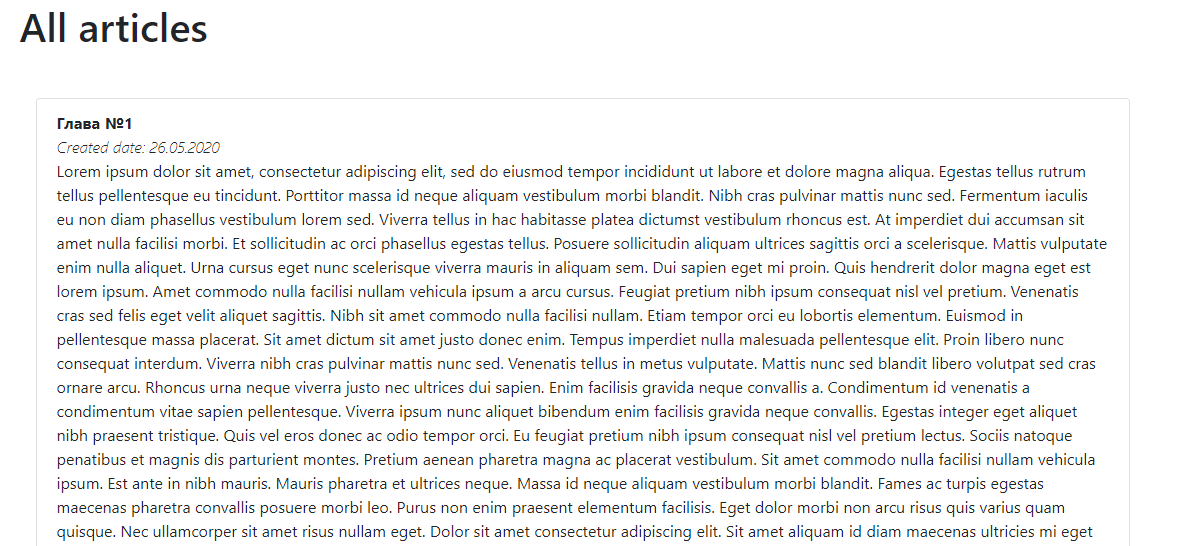


Рисунок 20

Разберем используемые команды для шаблонизатора:

* {% for article in articles %} – Цикл прохода по переданной переменной articles
* {{ article.title }} – Директива получающая данные из переменной, в данном случае получает заголовок
* {{ article.created\_date |date:"d.m.o" }} - Так же получаем дату, но используя фильтры, позволяющие форматировать дату в необходимый нам вид
* {{ article.content }}– Выводит содержание статьи
* {% if article.files %} – Оператор условия, проверяющий наличия прикрепленного файла, если файл не прикреплен, ничего не выводит
* {% endif %} – Конец условного оператора
* {% endfor %} – Конец оператора цикла

Формирование веб-страницы

Для взаимодействия с базой данных были созданы модели, миграции и их представление, также получили ответ в виде веб-страницы, однако на ней все статьи представлены с полным описанием, что не является удобным. Для решения этой проблемы, необходимо динамически создавать веб-страницы, конструируемые на основе статьи, а на основном проекте, выдавать только некоторую выдержку из содержания статьи и ссылку на генерируемую страницу.

Создадим в каждом блоке, содержащей статью, ссылку, а так же обрежем текст содержания, для этого перейдем в файл index.html (*Рис.21*) и внесём изменения:

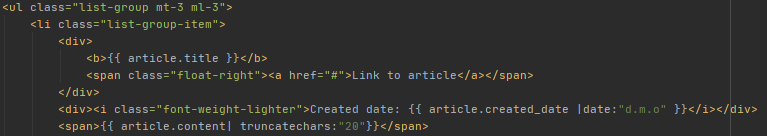


Рисунок 21

Символ “|” – показатель фильтра, в данном случае: truncatechars, который оставляет заданное количество символов; Также создали ссылку, пока что пустую (*Рис.22*):

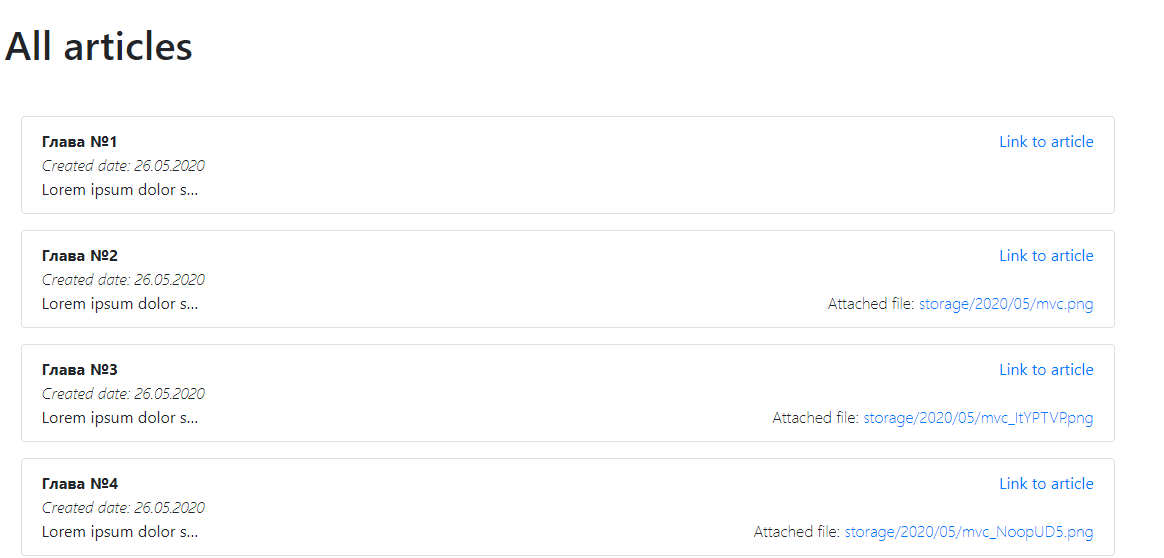


Рисунок 22

Для автоматической генерации страницы, необходимо создать еще один HTML шаблон, в который будут подставляться данные запрашиваемой статьи, создадим его в папке templates и назовем его description.html, пока что оставим его пустым, далее создадим контроллер, который будет принимать запрос пользователя и id статьи, страницу которой необходимо сгенерировать, а так пропишем маршрут, для вызова контроллера (*Рис.23 и Рис.24*) [5]:

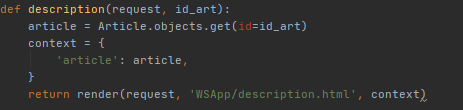


Рисунок 23

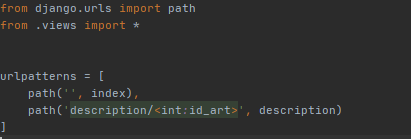


Рисунок 24

Для передачи данных используется следующая структура: в index.html ссылка будет иметь вид index/description/{{ article.id }}, т.е. при нажатии на эту ссылку будет формироваться запрос с id статьи, далее маршрутизатор свяжет это число с переменной id\_art, которая будет использована в контроллере, для поиска статьи в базе данных по его id, полученные данные передаются нашей новой странице description.html. Что бы не приходилось каждый раз создавать схожую структуру с index.html, можно воспользоваться следующий способом: ту часть веб-страницы которую мы хотим заменить необходимо обернуть в специальные теги:

{% block content %}

Заменяемая часть

{% endblock %}

А файл description.html заполнить следующим образом (*Рис.25*):

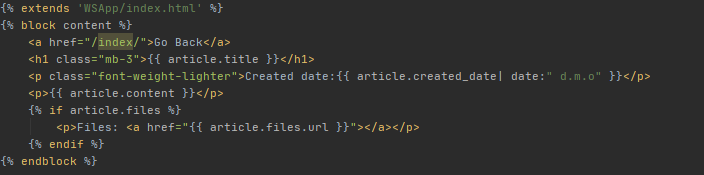


Рисунок 25

Теперь при изменении части, не входящую в block content, в файле index.html будет изменяться и структура файла description.html, то как работает данная функциональная особенность проверим в следующей главе.

Создание и реадиктирование Веб-страниц

Добавление статьи и её редактирование, на данный момент, реализовано только через админ панель, однако будет не достаточно, для полноценного веб-сайта, необходимо добавить возможность вносить правки в статьи, добавлять новые и удалять старые.   
Для создания новых статей, необходимо работать с формами и обрабатывать POST запросы. Создадим html документ – add\_post.html с следующей структурой(*Рис.26*):

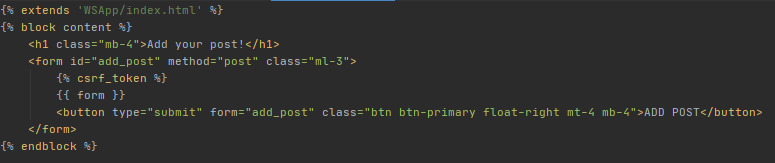


Рисунок 26

В документе в форме, запишем следующее: {% csrf\_token %}- встроенную защиту от перенаправленных запросов , которая генерирует специальный ключ на этапе отправления и сверяет его при получении. {{ form }} – форма сгенерированная Django, для ее реализации создадим в директории приложения файл forms.py и опишем форму(*Рис.27*):

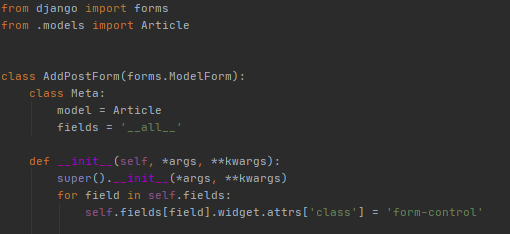


Рисунок 27

Создается новый класс, наследуемый от служебного forms, далее описывается Meta-класс содержащий в себе необходимые атрибуты для работы с базой данных. Поле model – принимает значение модели базы данных, fields – поля, которые необходимо импортировать (функция переопределения инициализации присутствует для применения стилей к форме).

После создание необходимой формы, передадим эту форму в контроллер, который будет обрабатывать страницу добавления новой статьи (регистрация маршрута здесь и в дальнейшем будет опускаться) (*Рис.28*):

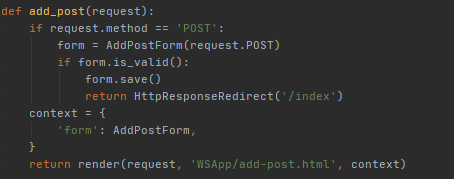


Рисунок 28

Контроллер с именем add\_post принимает запрос, проверяет является ли метод запроса - POST, если да, проверят на правильность заполненной формы и вносит изменения в базу данных. Если всё прошло успешно, производит перенаправление на начальную страницу [3].

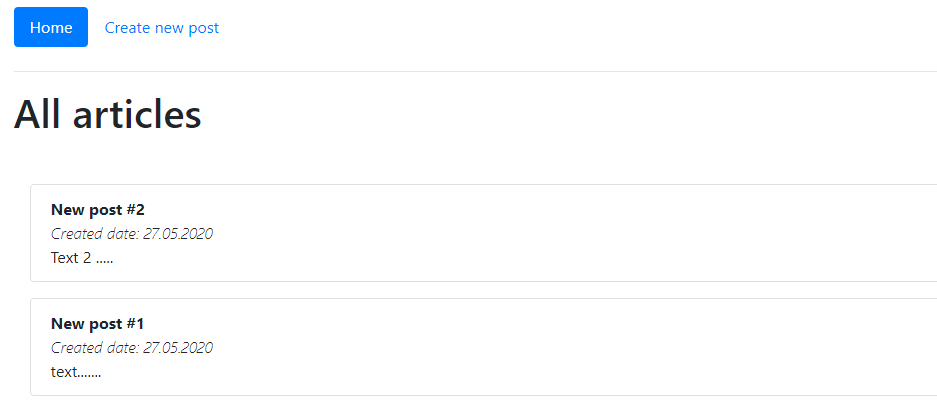


Рисунок 29

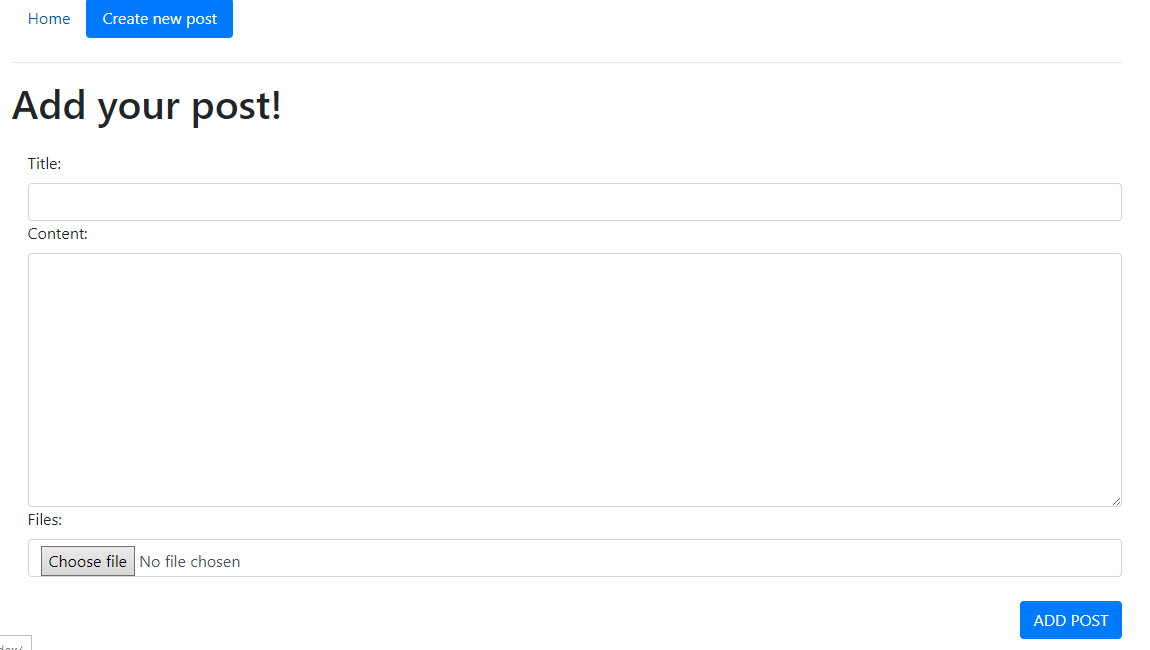


Рисунок 30

Для редактирования существующий форм, понадобится еще одна веб-страница. Создаем файл update-post.html, в нем достаточно описать схожую структуру как в add-post.html, форма для заполнения уже создана в предыдущем шаге, осталось описать работу обработки в новом контроллере update\_post (*Рис.31*):

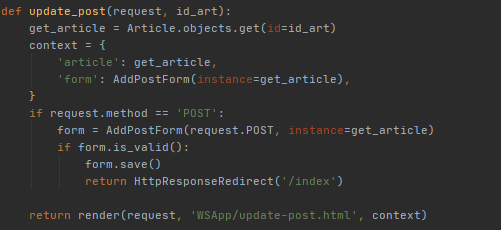


Рисунок 31

Контроллер принимает два параметра: запрос и id статьи (id передается через маршрут). Переменная get\_article хранит информацию о статье, найденную по её id, для того, чтобы форма на странице была заполнена уже имеющимися данными о статье, необходимо передать именной атрибут instance с полями статьи. Далее реализация схожа с давлением статьи.

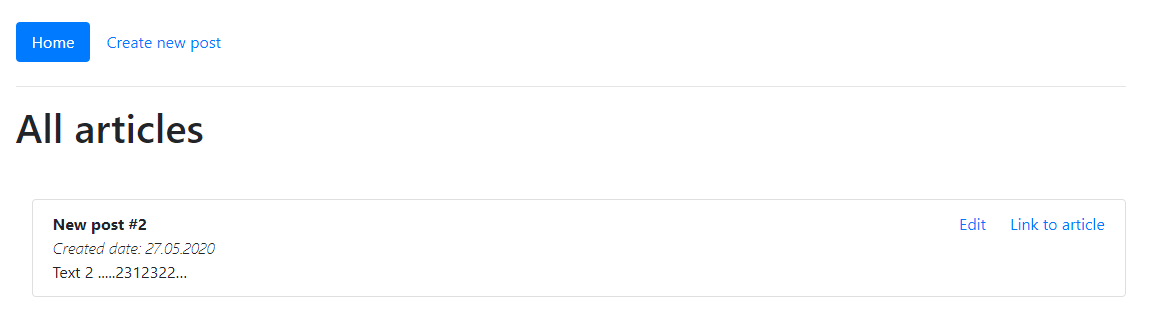


Рисунок 32

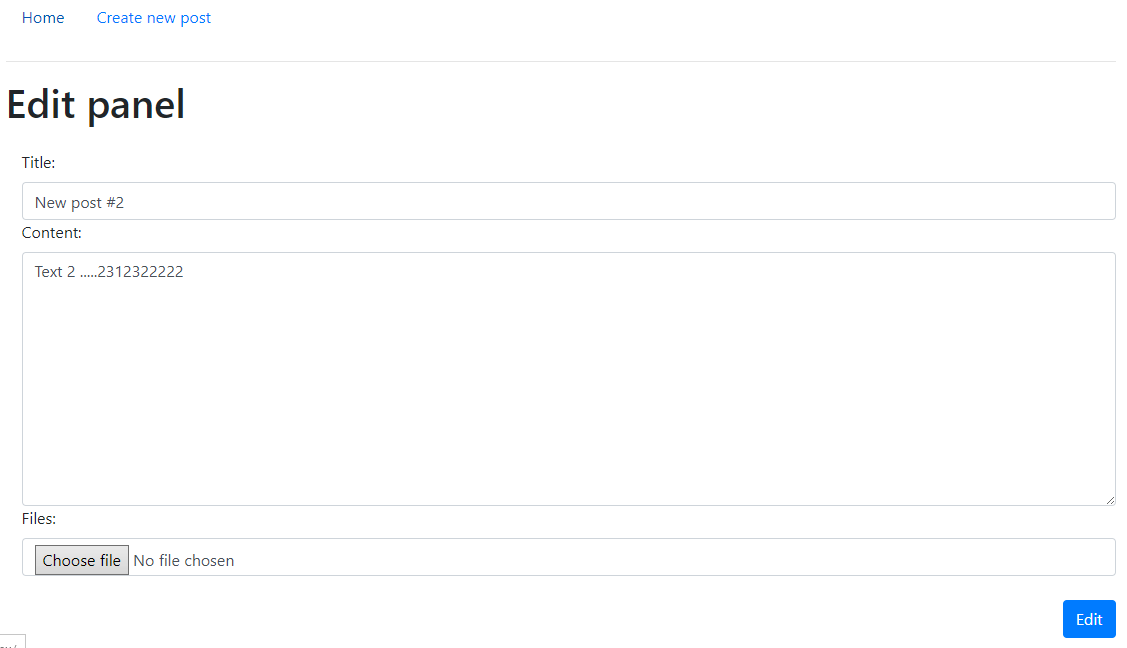


Рисунок 33

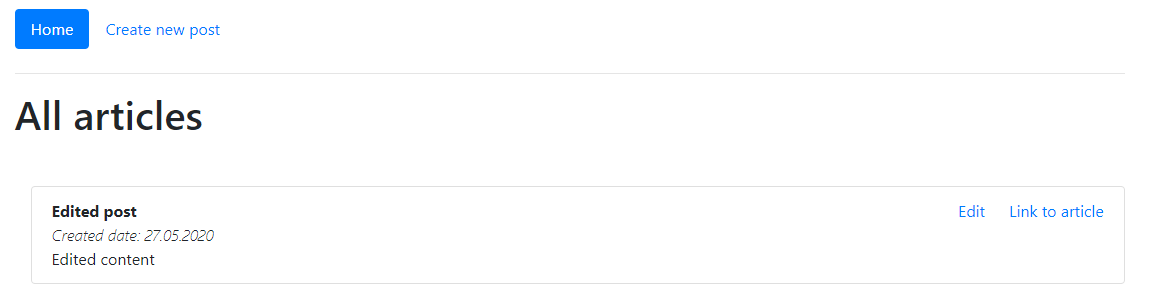


Рисунок 34

Для удаления элемента, необходимо передать id элемента контроллеру, который выполнит следующие инструкции(*Рис.35 и Рис.36*):



Рисунок 35

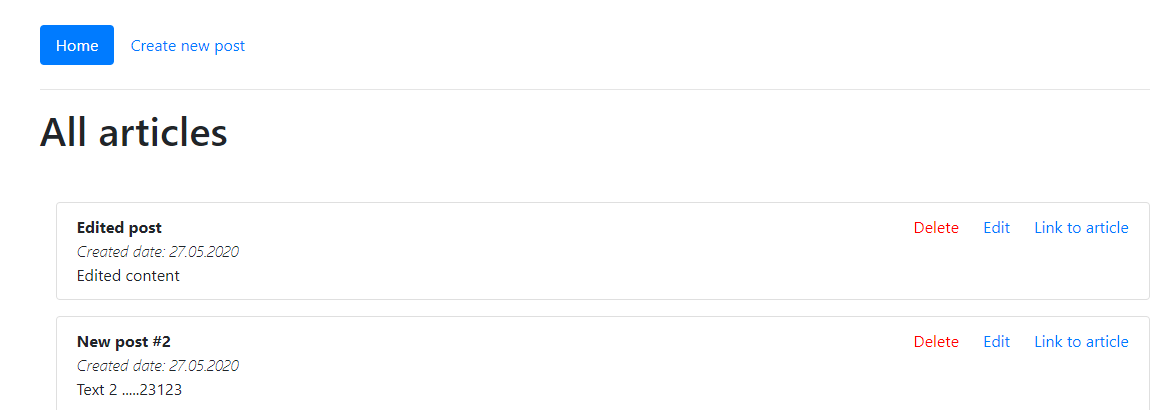


Рисунок 36

Авторизация пользователй и Разделение доступа

На данный момент абсолютно любой пользователь, который зайдет на сайт, имеет доступ изменять, удалять и добавлять статьи. Добавим авторизацию пользователей, для реализации можно воспользоваться встроенной системой авторизации, которая создаётся при инициализации рабочего пространства. Создадим два дополнительных HTML шаблона: login-page.html, auth-page.html, по своему содержанию они идентичны с предыдущими формами для создания статьи. Далее создадим необходимые формы для заполнения при авторизации в файле forms.py (*Рис.37*):



Рисунок 37

Описание форм создается на основе наследуемой формы AuthenticationForm, перезаписываем используемую модель на User, а также необходимые поля формы – username и password. Далее необходимо передать формы соответствующим контроллерам в файле views.py(*Рис.38*):

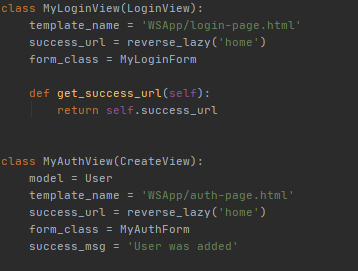


Рисунок 38

Контроллеры для авторизации создаются в виде классов, наследуемых от встроенных классов для создания контроллеров, атрибуты которых обеспечивают корректную работу: tamplate\_name – привязка к HTML шаблону, success\_url – перенаправление при успешной авторизации, form\_class – форма контроллера, model – используемая модель базы данных; Так же переопределяем метод get\_success\_url класса MyLoginView в связи с тем, что встроенное поведение, после успешной авторизации, перенаправляет на страницу с именем пользователя. Создадим пользователя и проверим его существования в админ панели(*Рис.39 и Рис.40*):

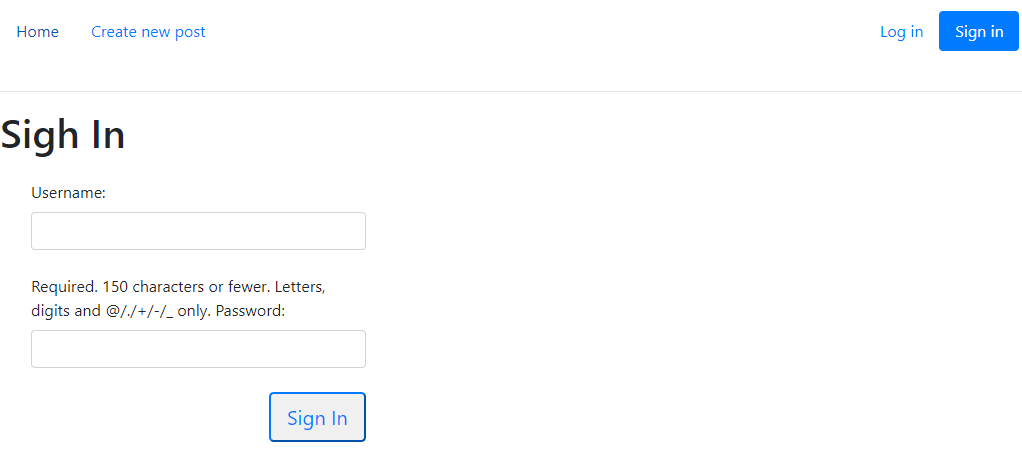


Рисунок 39

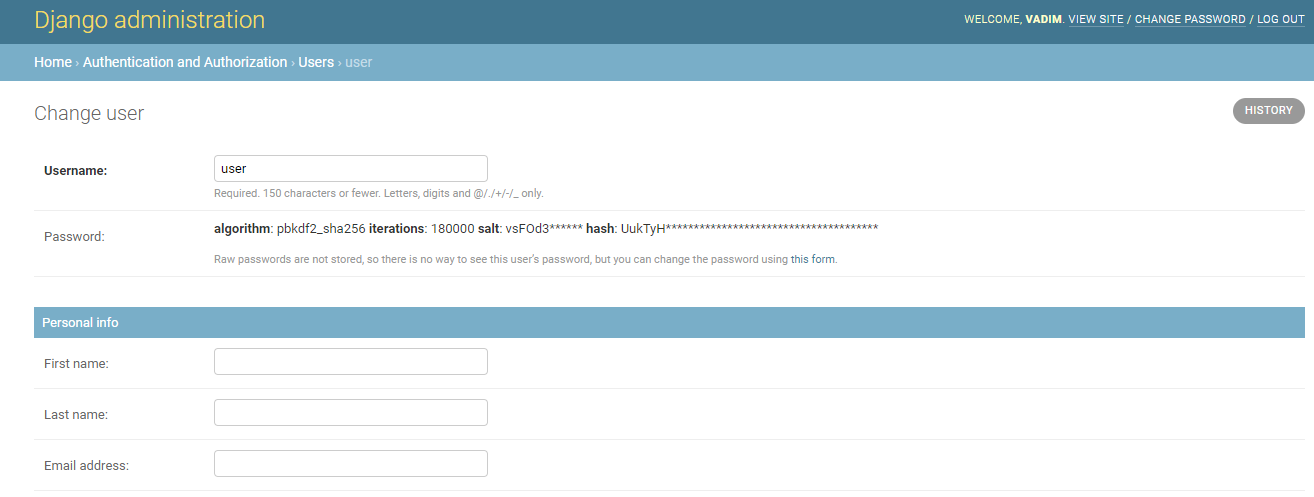


Рисунок 40

Для добавления кнопки выхода из аккаунта достаточно связать её с контроллером(*Рис.41*):



Рисунок 41

В файле index.html можем внедрить проверку, зарегистрирован ли пользователь, если да, то скрыть кнопки авторизации и добавить кнопку выхода(*Рис.42 и Рис.43*):

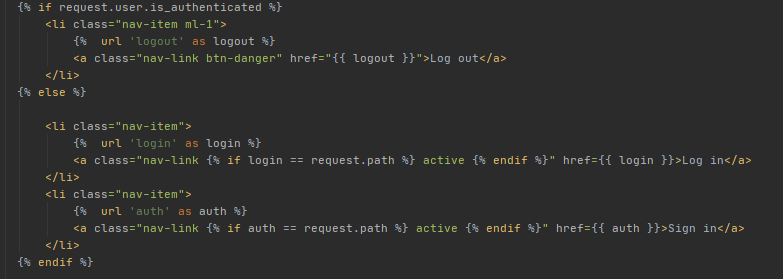


Рисунок 42

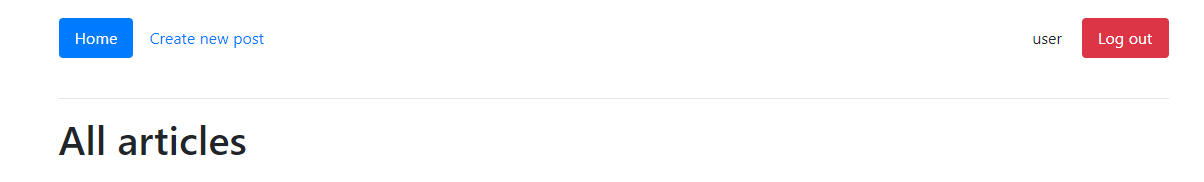


Рисунок 43

Теперь ограничим возможность добавлять статьи и сделаем их доступными только для авторизированных пользователей, для этого необходимо контроллеру add\_post добавить наследование от LoginRequiredMixin, а так же добавить поле

login\_url = reverse\_lazy('login')

Для переадресации на страницу регистрации. А так же скроем кнопку от не зарегистрированных пользователей (*Рис. 44):*

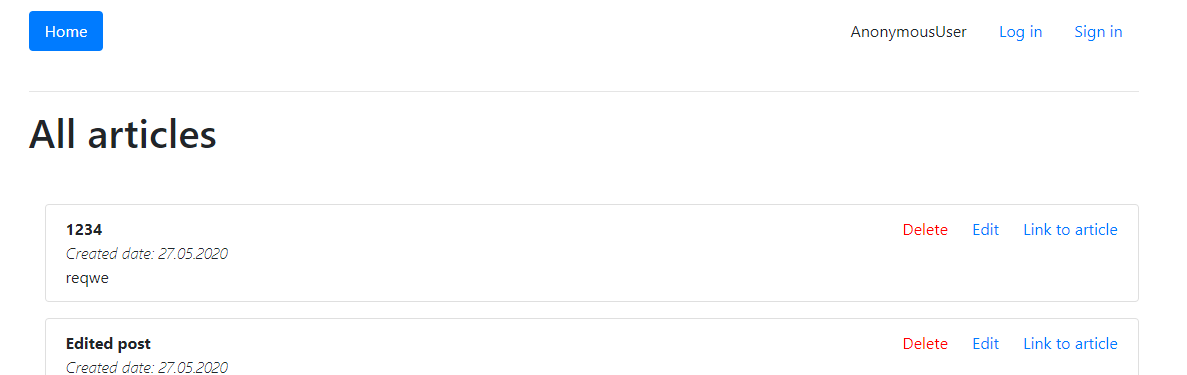


Рисунок 44

Теперь добавим наследование и для del\_post и update\_post, и скроем их(*Рис.45 и Рис.46*):

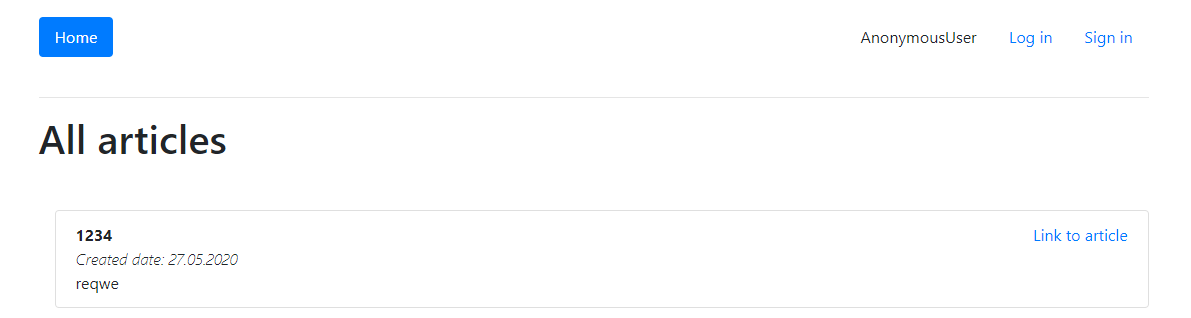


Рисунок 45

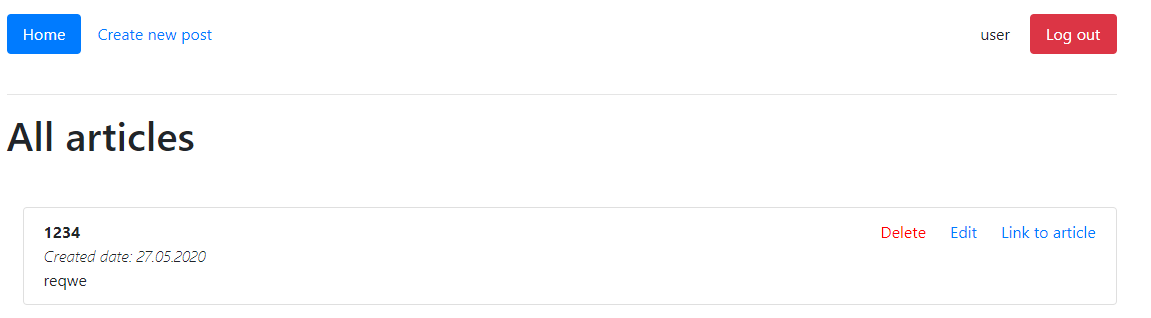


Рисунок 46

Заключение

В ходе разработки проекта, были реализованы следующие функциональные особенности: регистрация и авторизация пользователя, динамические веб-страницы, разграничение доступа, внесения изменений в базу данных существующего проекта. А также были подробно разобраны: создание рабочей среды, маршрутизация проекта, создание приложения, разработка контроллера, работа шаблонизатора, создание базы данных средствами Python Django. Данный фреймворк занимает лидирующие позиции по популярности использования, в ходе работы были показаны основные преимущества и особенности Django, по причине которых, его используют многие именитые компании.

В курсовой работе не были описаны принципы использования стилей Bootstrap, не было разобрано написание HTML разметки, подготовка среды Windows для начала работы с Django, а также не рассматривались принципы работы с ООП.

Данный фреймворк имеет очень большие возможности, про которые не было упомянуто, весь функционал Django наиболее полно описан в его официальной документации.

Проект расположен на Google Driver: https://drive.google.com/open?id=1eNXAsNGhZNqLgxsx7WabUBFNZGyFv8\_T

Литература

1. Django documentation [Электронный ресурс] Режим доступа: <https://docs.djangoproject.com/en/3.0/> - Дата доступа: 05.05.2020.
2. Python 3.8.3 documentation [Электронный ресурс] Режим доступа: <https://docs.python.org/3/> - Дата доступа: 07.05.2020.
3. Каплан-Мосс Д. Д. Django. Подробное руководство. 2-е издание / Д. Д. Каплан-Мосс. – 2012 г. – 549с.
4. Лутц. М. И. Python, 4-е издание / М. И. Лутц - 2019 г. – 1280с.
5. Дронов В. В. Django: практика создания Web-сайтов на Python / В. В. Дронов. – 2019г. – 530с.