GeekBrains

Разработка веб-приложения

на языке программирования python

с использованием фреймворка Django

Программа:

Программист Python. Цифровые Профессии

Лудинов В.О.

Архангельск

2024

**СОДЕРЖАНИЕ**

1. Введение 3
   1. Веб-приложение и его особенности 4
      1. Что такое веб-приложение и как оно работает? 4
      2. Веб-приложение и веб-сайт: в чем разница? 4
      3. Виды веб-приложений 5
      4. Веб-приложения для бизнеса 7
      5. Как создать свое веб-приложение? 8
   2. Django 10
      1. Что такое Django? 10
      2. Как он появился? 12
      3. Насколько популярен Django? 12
      4. Является ли Django гибким? 13
      5. Как выглядит код Django? 13
   3. Создание веб-приложения 15
      1. Создание механики аутентификации пользователей 15
      2. Создание основного приложения 20
      3. Представления Create 24
      4. Представления Read 29
      5. Представления Update 37
      6. Представления Delete 42
2. Заключение 50
3. Список используемых источников 51

**ВВЕДЕНИЕ**

Тема проекта: создание простого веб-приложения на языке python позволяющего делать записи о прочитанных и еще не прочитанных произведениях различных авторов с различных сайтов и из различных стран, чтобы структурировать и предотвратить забывание различных книг.

Цель проекта: научиться создавать веб-приложения на языке python.

Какие проблемы решает: книги различных авторов могут храниться на различных сайтах никак не связанных друг с другом, поэтому общий список с красивым и удобным графическим интерфейсом облегчит задачу запоминания множества книг и авторов.

Задачи: просмотр обучающих видео и чтение обучающей литературы по теме создания веб-приложений на языке python, работе с базами данных mysql и языки html и css.

В проекте использовано: фреймворк Django языка программирования python, база данных mysql, библиотека python mysql-python.

Состав команды: Лудинов В.О. (Программист).

1. **Веб-приложение и его особенности**
   1. **Что такое веб-приложение и как оно работает?**

Веб-приложение – это программа с определенными функциями, которая работает через браузер. Его можно использовать как во внутренних бизнес-процессах компании, так и в работе с клиентами. Во втором случае оно позволяет пользователям заказывать товары, оплачивать покупки, бронировать номера, оставлять отзывы и др.

Основные преимущества веб-приложений:

- взаимодействуют с любой операционной системой и браузером;

- не требуют одобрения сторонней платформы для выхода на рынок, как в случае с мобильными приложениями;

- не требуют установки на устройство — достаточно знать URL-адрес приложения, чтобы взаимодействовать с ним;

- нет необходимости в подписке или покупки лицензии, что очень выгодно для компаний.

Теперь давайте рассмотрим, как такое приложение работает. Пользователь формирует запрос, например, показать список друзей в социальной сети. Этот запрос отправляется на сервер, где хранятся все необходимые данные. Как только сервер получил запрос, он направляется в базу данных и формирует ответ на запрос пользователя. После этого, отправляет необходимую информацию ему на экран. Пользователь видит на своем устройстве общий список друзей и кто в данный момент онлайн.

Примерно такой принцип работы и преследуют веб-приложения. Однако стоит понимать, чем сложнее запрос пользователя, тем больше времени уйдет на формирование ответа. Поэтому главная задача всех веб приложений — обеспечить грамотное взаимодействие между сервером и пользователем и быстро формировать готовые страницы в ответ на запрос клиента.

* 1. **Веб-приложение и веб-сайт: в чем разница?**

Многие задаются вопросом — в чем разница между сайтом и приложением? С первого взгляда, эти два понятия одинаковы, но отличия заключаются во внутренней работе с ними и создании самого проекта.

Веб-сайт

Это одна или несколько веб страниц, которые демонстрируют пользователю информационный контент: статьи, видео, картинки, музыка. Как правило, эти данные статичны и пользователь не может взаимодействовать с ними, то есть менять что-то на странице. Обычно сайт используют для демонстрации продуктов, рассказа о бренде, его истории и достижениях, публикации полезных кейсов и статей. Зачастую тут не требуется аутентификация, а значит, вся информация доступна каждому пользователю.

Веб-приложение

Полноценная программа с расширенным функционалом. По сути это тот же сайт, только с интерактивными элементами, которые решают сразу несколько задач клиента. Здесь он уже сможет взаимодействовать с контентом и проводить разные манипуляции, в пределах допустимого: добавлять информацию, общаться с другими пользователями, заказывать товары, оплачивать покупки, добавлять или удалять продукты из корзины. Поэтому здесь важным моментом является регистрация себя, как пользователя и встречается она гораздо чаще, чем у веб-сайтов.

* 1. **Виды веб-приложений**

Теперь, когда мы разобрались с тем, что такое веб-приложение, как оно работает и чем отличается от сайта, рассмотрим их виды с точки зрения технической классификации:

SPA (Single Page Application) — одностраничное интерактивное приложение, которое не требует перезагрузки основной страницы во время своей работы. В отличие от традиционных веб-приложений, они работают быстрее, так как вся логика приложения реализуется в браузере, а не на сервере.

Плюсы:

Удобный интерфейс. Пользователю проще ориентироваться на одной странице, когда есть только один вариант движения — вниз. Никакие дополнительные ссылки не отвлекут его внимание и не позволят запутаться внутри программы.Высокая скорость. Во время работы, приложение не обновляет всю страницу, а лишь подгружает нужный пользователю контент. Поэтому, оно работает гораздо быстрее многостраничных сайтов — запрос на сервер делается один раз, после чего информация на странице просто дополняется.

Кэширование данных. Точно также работает и кэш — достаточного обратиться к серверу один раз и он сохранит все данные пользователя. Теперь клиент сможет работать в приложении даже в автономном режиме.

Минусы:

Недостаточная SEO-оптимизация. У одностраничных приложений всего один URL-адрес, который не меняется в процессе работы, в отличие от многостраничников. Поэтому разработчик может привязать к ней только один поисковый запрос. Такие сайты трудно оптимизировать под поисковые системы из-за того, что дополнительно подгруженную информацию поисковый бот просто не найдет.

Плохая безопасность. Одностраничные приложения больше подвержены атаке хакеров, которые крадут данные пользователей, с помощью межсайтовых сценариев. Они внедряют их со стороны клиента и легко могут получить конфиденциальную информацию. У MPA и PWA, которые мы рассмотрим ниже, безопасность настроена гораздо лучше.  
Когда использовать:

Подойдет для создания социальных сетей, различных динамических платформ, на которых небольшой объем информации и нет большой потребности в SEO-ключах.

MPA (Multi Page Application) — многостраничное веб-приложение. Оно позволяет пользователю переходить между отдельными страницами при совершении запроса. У таких ресурсов, как правило, более сложный интерфейс и большое количество дополнительных ссылок. При каждом новом запросе, например, при оплате продукта, у пользователя откроется новая страница.

Плюсы:

Отличное SEO. В отличие от SPA, у MPA каждая страница имеет свой URL-адрес, что дает возможность отлично контролировать SEO-запросы пользователей — на каждую из них, разработчик может добавить свой уникальный ключ и метатеги.

Неограниченная масштабируемость. MPA не ограничивает вас по количеству создания страниц и информации, которую нужно на ней разместить. В отличие от SPA, тут можно использовать множество функций на одной странице. Поэтому, если ваша задача — больше возможностей для клиента, выбирайте MPA.

Минусы:

Низкая скорость. При каждом взаимодействии с приложением, сервер перезагружает и обновляет все ресурсы — HTML, CSS, скрипты и все данные страницы. Поэтому скорость ожидания ответа на запрос, увеличивается, что негативно может повлиять на производительность приложения.

Более сложная разработка. При создании такого приложения, требуется полноценная разработка серверной и фронтальной части программы и использование фреймворков для внешнего и внутреннего интерфейса. Все это занимает гораздо больше времени, в сравнении с созданием SPA.

Когда использовать:

Отлично подойдет крупным компаниям, которые предлагают широкий спектр услуг или товаров, интернет-магазинам, каталогам и торговым площадкам.

PWA (Progressive Web Application) — прогрессивное веб-приложение. Разрабатывается с помощью специальных технологий и способно работать без подключения к интернету. Такие приложения доступны на всех устройствах и адаптируются под — любую ширину экрана телефона или компьютера. Их можно устанавливать на главный экран своего смартфона, отправлять уведомления и пользоваться им, как полноценным приложением.

Плюсы:

Быстрая разработка. Разработка PWA занимает меньше времени, чем создание нативных приложений, поэтому является более доступной.

Необязательная установка. Приложением можно пользоваться напрямую через браузер, не скачивая его из магазина приложений. Это значит, что при разработке, не нужно будет учитывать их правила и подгонять программу под стандарт.

Множество устройств. PWA доступен в браузере на компьютерах, телефонах и планшетах, что значительно повышает удобство работы для пользователя.

Автономный режим. Пользователям не нужно каждый раз загружать полностью все приложение — оно частично может работать в автономном режиме.

Минусы:

Низкая производительность. PWA уступает своей производительностью нативным приложениям.

Нет доступа к магазинам приложений. Прежде чем добавить значок приложения на экран своего телефона, пользователь должен сначала обязательно посетить этот веб-сайт.

Когда использовать:

Подходит сетевым крупным компаниям для того, чтобы контролировать систему работы, повышать уровень лояльности клиентов и работать на узнаваемость бренда; сервисам по предоставлению услуг; бизнесу для создания онлайн-курсов.

* 1. **Веб-приложения для бизнеса**

CRM-системы. Используются для автоматизации продаж компании и взаимодействия работы с клиентами. С помощью такой системы вы можете повысить уровень продаж, оптимизировать маркетинг, улучшить обслуживание клиентов, выставить и контролировать бизнес-процессы, анализировать результаты своей деятельности.

E-commerce системы. Системы для электронных онлайн-сделок. С их помощью пользователи совершают покупки, продают товары, переводят денежные средства без участия сторонних лиц. К E-commerce относятся: онлайн-магазины, маркетплейсы, курьерские службы, b2b-сервисы и платформы.

ERP-системы. Используются для управления бизнес-процессами в компании: от отдела продаж до бухгалтерского учета. С ее помощью вы можете отслеживать эффективность работы каждого отдела и ставить соответствующие задачи для ее повышения.

Корпоративные порталы. Веб-интерфейсы, которые позволяют сотрудникам компании видеть все корпоративные данные фирмы. С его помощью пользователи могут обмениваться информацией, различными документами, добавлять или менять информацию, работать совместно над проектами.

* 1. **Как создать свое веб-приложение?**

Чтобы разработать веб-приложение под ключ, нужна команда профессионалов. К их выбору стоит подойти очень серьезно, ведь именно от компетенций специалистов зависит будущее вашего приложения. Как же выбрать команду для создания проекта? Рассказываем.

В первую очередь — изучите портфолио, кейсы и отзывы о компании, которая вам понравилась. Опытные разработчики знают, на что клиенты обращают внимание и всегда делятся своими работами, грамотно презентуют их на официальном сайте. Внимательно посмотрите, что пишут о компании в интернете и поинтересуйтесь на их счет у коллег. Мы в LeanTech всегда делимся нашими кейсами и подробно прописываем этапы работы на своем сайте.

Допустим, вы нашли подходящий вариант и следующий шаг — это встреча с менеджером компании. Выделите достаточно времени и подготовьтесь рассказать о своем проекте, как можно подробнее. Убедитесь в том, верно ли менеджер понял вашу цель и идею веб-приложения и насколько вам приятно вести с ним диалог. Оцените сроки и стоимость проекта, после чего можете делать окончательный вывод о том, подходит вам эта команда или нет.

Отлично, подрядчик выбран, приступаем к созданию веб-приложения. Разработка такого проекта выглядит следующим образом:

Оценка и аналитика

Наша команда внимательно изучает идею и варианты достижения ваших целей. После этого менеджер предоставляет вам детальную оценку проекта, учитывая все функции и сроки его выполнения, а аналитик помогает точно оценить сроки и стоимость работы, создать подробный план каждого этапа разработки. Таким образом, вы всегда будете в курсе, над чем именно мы работаем сейчас и чего ждать от каждого этапа работы.

Создание прототипа

После оценки и аналитики, команда приступает к созданию прототипа — черновому варианту будущего веб-приложения. С его помощью, мы продемонстрируем вам базовую версию проекта и как пользователь сможет взаимодействовать с ней.

Дизайн

Подбираем цветовую гамму и разрабатываем фирменный стиль веб-приложения. На этом этапе мы формируем внешний вид элементов продукта с учетом всех требований и особенностей восприятия ваших пользователей.

Разработка и тестирование

Основная часть работы команды по разработке, где мы пишем код и реализуем необходимые функции — создаем все страницы вашего приложения с соблюдением логики и основной цели проекта. На каждом этапе происходит тестирование приложения — важно убедиться, что все элементы работают стабильно и выполняют свои основные функции.

Поддержка

После того как приложение успешно прошло проверку и вышло на рынок, мы сопровождаем его, дорабатываем существующие функции или добавляем новые, исходя из ваших бизнес-задач и анализа эффективности взаимодействия пользователя с сайтом.

Итог

Разработать веб-приложение для своего бизнеса — это отличная возможность достигнуть и реализовать все ваши идеи. Используя современный подход к его созданию, вы сможете улучшить бизнес-процессы, повысить уровень продаж и контролировать эффективность взаимодействия с пользователями внутри компании. Грамотная команда специалистов поможет вам достигнуть хорошего результата, сделать ваш бренд узнаваемым, повысить лояльность существующих клиентов и привлечь новых.

1. **Django**
   1. **Что такое Django?**

Django — это высокоуровневый Python веб-фреймворк, который позволяет быстро создавать безопасные и поддерживаемые веб-сайты. Созданный опытными разработчиками, Django берёт на себя большую часть хлопот веб-разработки, поэтому вы можете сосредоточиться на написании своего веб-приложения без необходимости изобретать велосипед. Он бесплатный и с открытым исходным кодом, имеет растущее и активное сообщество, отличную документацию и множество вариантов как бесплатной, так и платной поддержки.

Django помогает писать программное обеспечение, которое будет:

Полным

Django следует философии «Всё включено» и предоставляет почти всё, что разработчики могут захотеть сделать «из коробки». Поскольку всё, что вам нужно, является частью единого «продукта», всё это безупречно работает вместе, соответствует последовательным принципам проектирования и имеет обширную и актуальную документацию.

Разносторонним

Django может быть (и был) использован для создания практически любого типа веб-сайтов — от систем управления контентом и wiki до социальных сетей и новостных сайтов. Он может работать с любой клиентской средой и может доставлять контент практически в любом формате (включая HTML, RSS-каналы, JSON, XML и т. д.). Сайт, который вы сейчас читаете, создан с помощью Django!

Хотя Django предоставляет решения практически для любой функциональности, которая вам может понадобиться (например, для нескольких популярных баз данных, шаблонизаторов и т. д.), внутренне он также может быть расширен сторонними компонентами, если это необходимо.

Безопасным

Django помогает разработчикам избежать многих распространённых ошибок безопасности, предоставляя фреймворк, разработанный чтобы «делать правильные вещи» для автоматической защиты сайта. Например, Django предоставляет безопасный способ управления учётными записями пользователей и паролями, избегая распространённых ошибок, таких как размещение информации о сеансе в файлы cookie, где она уязвима (вместо этого файлы cookie содержат только ключ, а фактические данные хранятся в базе данных) или непосредственное хранение паролей вместо хэша пароля.

Хэш пароля – это значение фиксированной длины, созданное путём обработки пароля через криптографическую функцию Django может проверить правильность введённого пароля, пропустив его через хэш-функцию и сравнив вывод с сохранённым значением хэша. Благодаря «одностороннему» характеру функции, даже если сохранённое хэш-значение скомпрометировано, злоумышленнику будет сложно определить исходный пароль.

Django, по умолчанию, обеспечивает защиту от многих уязвимостей, включая SQL-инъекцию, межсайтовый скриптинг, подделку межсайтовых запросов и кликджекинг.

Масштабируемым

Django использует компонентную "shared-nothing" архитектуру (каждая её часть независима от других и, следовательно, может быть заменена или изменена, если это необходимо). Чёткое разделение частей означает, что Django может масштабироваться при увеличении трафика, путём добавления оборудования на любом уровне: серверы кеширования, серверы баз данных или серверы приложений. Одни из самых загруженных сайтов успешно масштабировали Django (например, Instagram и Disqus, если назвать только два из них).

Удобным в сопровождении

Код Django написан с использованием принципов и шаблонов проектирования, которые поощряют создание поддерживаемого и повторно используемого кода. В частности, в нём используется принцип «Don't Repeat Yourself» (DRY, «не повторяйся»), поэтому нет ненужного дублирования, что сокращает объём кода. Django также способствует группированию связанных функциональных возможностей в повторно используемые «приложения» и, на более низком уровне, группирует связанный код в модули (в соответствии с шаблоном Model View Controller (MVC)).

Переносным

Django написан на Python, который работает на многих платформах. Это означает, что вы не привязаны к какой-либо конкретной серверной платформе и можете запускать приложения на многих версиях Linux, Windows и Mac OS X. Кроме того, Django хорошо поддерживается многими веб-хостингами, которые часто предоставляют определённую инфраструктуру и документацию для размещения сайтов Django.

* 1. **Как он появился?**

Django был разработан в период с 2003 по 2005 год командой, которая занималась созданием и обслуживанием газетных веб-сайтов. После создания нескольких сайтов, команда начала повторно использовать множество общего кода и шаблонов проектирования. Этот общий код эволюционировал в веб-фреймворк, который превратился в проект "Django" с открытым исходным кодом в июле 2005 года.

Django продолжает расти и улучшаться с момента его первого релиза (1.0) в сентябре 2008 года до недавно выпущенной версии 3.1 (2020). В каждой версии добавлены новые функциональные возможности и исправлены ошибки, начиная от поддержки новых типов баз данных, шаблонизаторов и кеширования, до добавления «общих» функций просмотра и классов (уменьшающих объём кода, который разработчики должны писать для ряда программных задач).

Django — это процветающий совместный проект с открытым исходным кодом, в котором заняты многие тысячи пользователей и участников. Несмотря на то, что у него всё ещё есть некоторые особенности, которые отражают его происхождение, Django превратился в универсальный фреймворк, способный разрабатывать веб-сайты любого типа.

* 1. **Насколько популярен Django?**

Нет никаких доступных и окончательных оценок популярности серверных фреймворков (хотя сайты наподобие Hot Framework и пытаются оценить популярность, используя такие механизмы, как подсчёт количества проектов на GitHub и вопросов на StackOverflow для каждой платформы). Лучший вопрос — «достаточно ли Django популярен», чтобы избежать проблем непопулярных платформ. Продолжает ли он развиваться? Можете ли вы получить помощь, если вам нужно? Найдёте ли вы оплачиваемую работу, если изучите Django?

Основываясь на количестве крупных сайтов, которые используют Django, количестве участников и количестве людей, предоставляющих как бесплатную, так и платную поддержку, можно ответить: да, Django — популярный фреймворк!

Django используют такие крупные сайты, как Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest и Open Stack.

* 1. **Является ли Django гибким?**

Веб-фреймворки часто можно поделить на "гибкие" и "негибкие".

Негибкие - это те, у которых есть "правильный путь" для решения какой-либо конкретной задачи. Они часто поддерживают быстрое развёртывание в определённой области (решение проблем определённого типа), потому что правильный способ сделать что-либо обычно хорошо понимается и хорошо документируется. Однако они могут быть менее гибкими при решении проблем за пределами их основной сферы и, как правило, предлагают меньше вариантов того, какие компоненты и подходы они могут использовать.

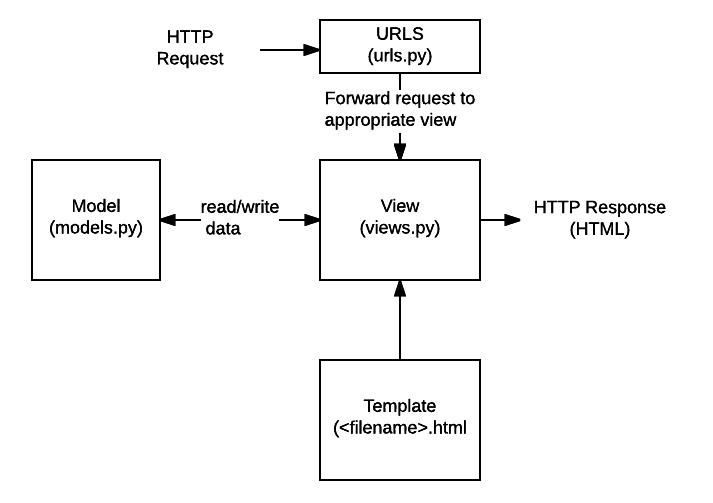
Напротив, у гибких фреймворков гораздо меньше ограничений на лучший способ склеивания компонентов для достижения цели или даже того, какие компоненты следует использовать. Они облегчают разработчикам использование наиболее подходящих инструментов для выполнения конкретной задачи, хотя и за счёт того, что вам нужно самим найти эти компоненты.

Django «умеренно гибкий» и, следовательно, обеспечивает «лучшее из обоих миров». Он предоставляет набор компонентов для обработки большинства задач веб-разработки и один (или два) предпочтительных способа их использования. Однако такая архитектура Django означает, что вы обычно можете выбирать из нескольких различных опций или при необходимости добавлять поддержку для совершенно новых.

* 1. **Как выглядит код Django?**

На традиционном информационном веб-сайте веб-приложение ожидает HTTP-запросы от веб-браузера (или другого клиента). Когда запрос получен, приложение разрабатывает то, что необходимо на основе URL-адреса и, возможно, данных в POST или GET запросах. В зависимости от того, что требуется, далее он может читать или записывать информацию из базы данных или выполнять другие задачи, необходимые для удовлетворения запроса. Затем приложение вернёт ответ веб-браузеру, часто динамически создавая HTML-страницу для отображения в браузере, вставляя полученные данные в HTML-шаблон.

Веб-приложения, написанные на Django, обычно группируют код, который обрабатывает каждый из этих шагов, в отдельные файлы:



URLs: Хотя можно обрабатывать запросы с каждого URL-адреса с помощью одной функции, гораздо удобнее писать отдельную функцию для обработки каждого ресурса. URL-маршрутизатор используется для перенаправления HTTP-запросов в соответствующее представление на основе URL-адреса запроса. Кроме того, URL-маршрутизатор может извлекать данные из URL-адреса в соответствии с заданным шаблоном и передавать их в соответствующую функцию отображения (view) в виде аргументов.

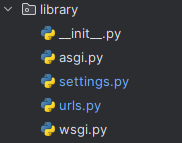
View: View (англ. «отображение») — это функция обработчика запросов, которая получает HTTP-запросы и возвращает ответы. Функция view имеет доступ к данным, необходимым для удовлетворения запросов, и делегирует ответы в шаблоны через модели.

Models: Модели представляют собой объекты Python, которые определяют структуру данных приложения и предоставляют механизмы для управления (добавления, изменения, удаления) и выполнения запросов в базу данных.

Templates: Template (англ. «шаблон») — это текстовый файл, определяющий структуру или разметку страницы (например HTML-страницы), с полями для подстановки, которые используются для вывода актуального содержимого. View может динамически создавать HTML-страницы, используя HTML-шаблоны и заполняя их данными из модели (model). Шаблон может быть использован для определения структуры файлов любых типов, не обязательно HTML.

1. **Создание веб-приложения**

Для начала был создан проект Django под названием library.



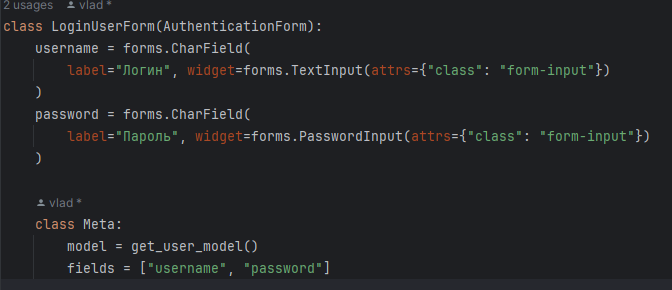
* 1. **Создание механики аутентификации пользователей**

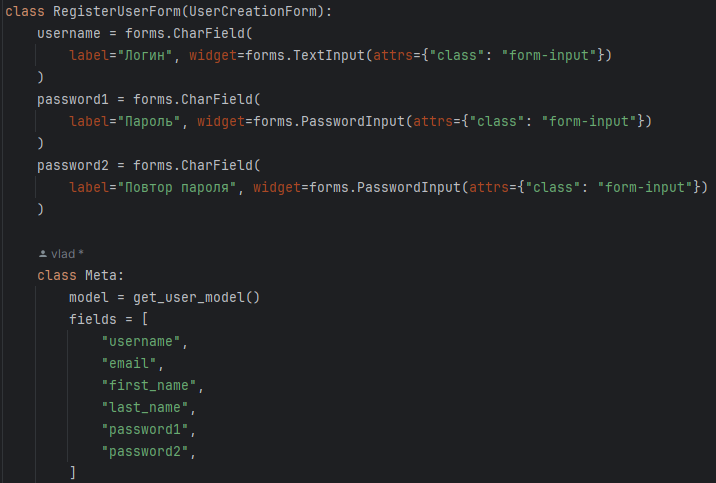
Первым делом было создано приложение users для обеспечения логики авторизации, регистрации и выхода из аккаунта пользователей моего сайта.



Сразу же добавил приложение в список INSTALLED\_APPS в файле settings проекта library.

Потом были созданы формы для авторизации и регистрации пользователей.







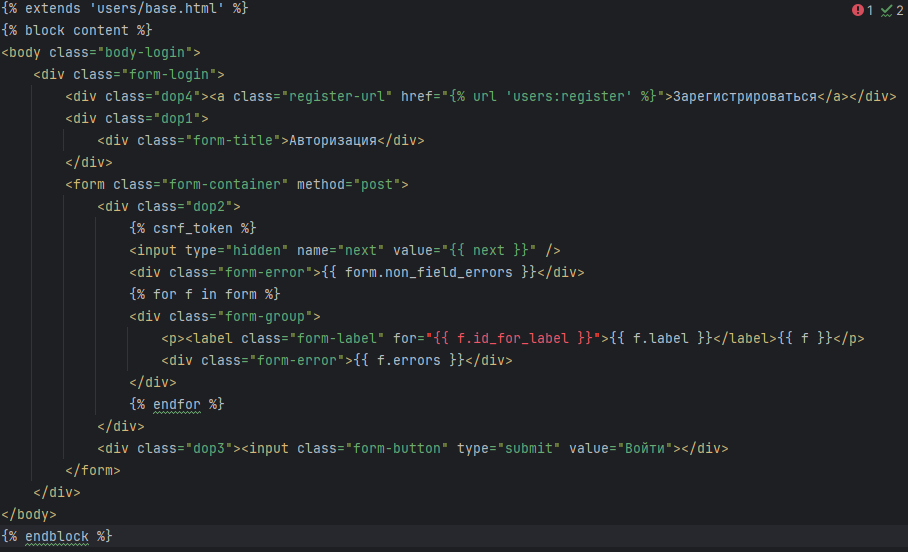
Потом были созданы html шаблоны для этих форм.

Сначала базовый шаблон base.

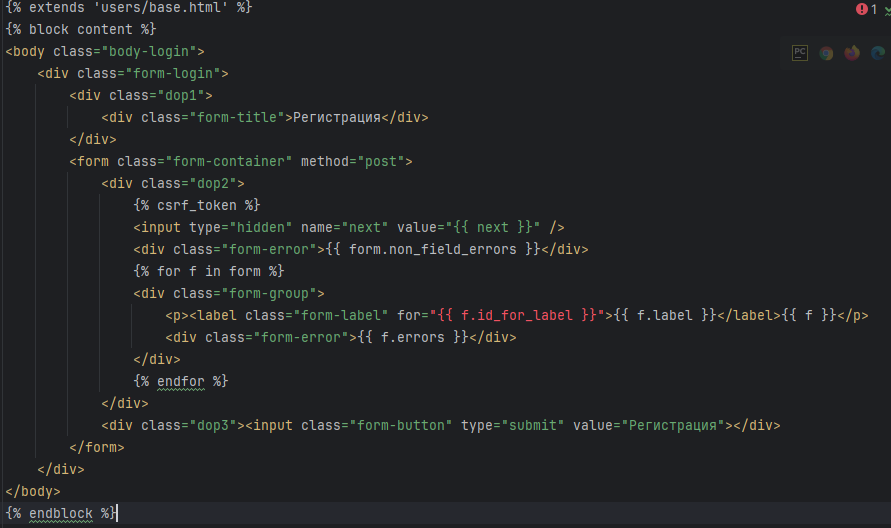


А потом и наследующиеся от него

login.



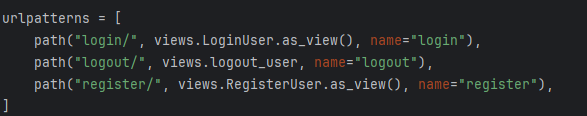
и register.



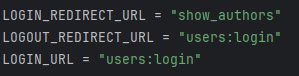
Далее были созданы классы представления LoginUser и RegisterUser, а также простая функция представление logout\_user.



Также были настроены url-адреса для этих представлений.



Кроме того в файле settings проекта library настроил url-адреса для авторизации и выхода из аккаунта.

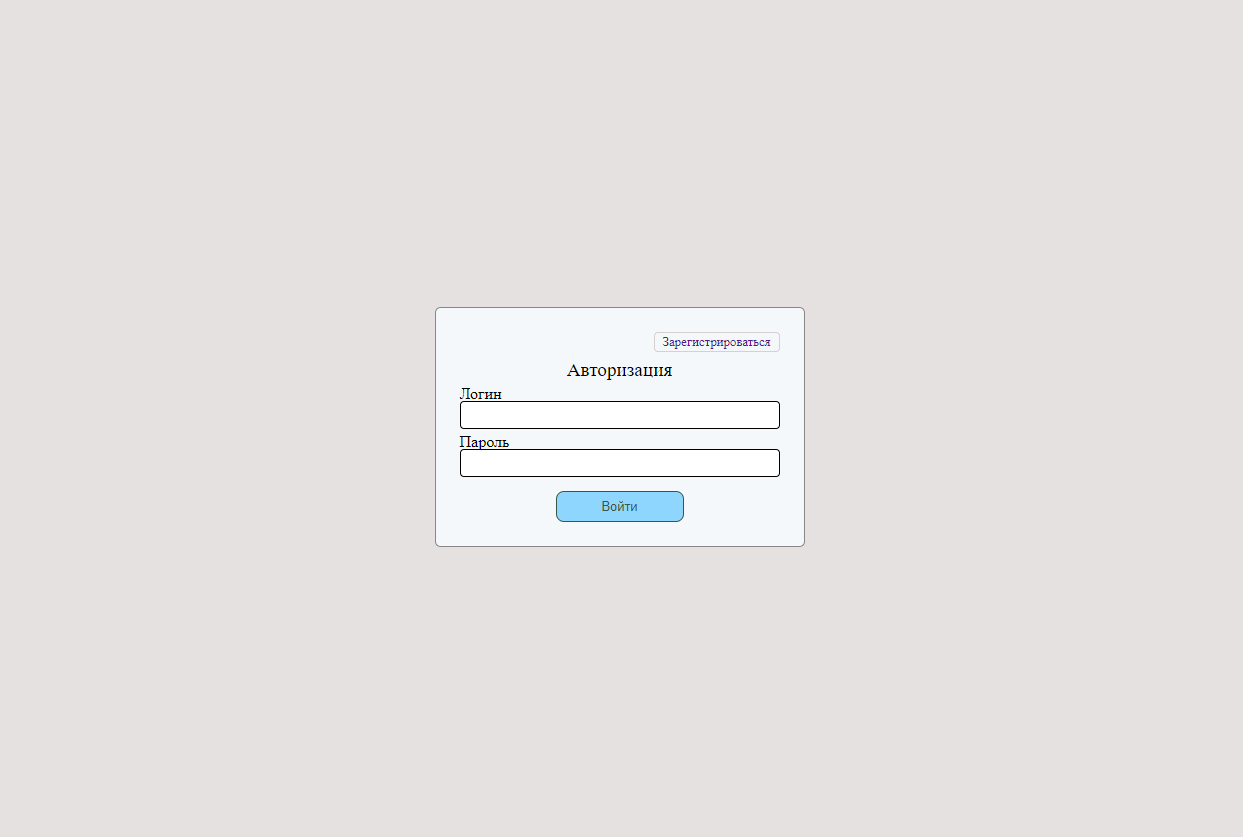


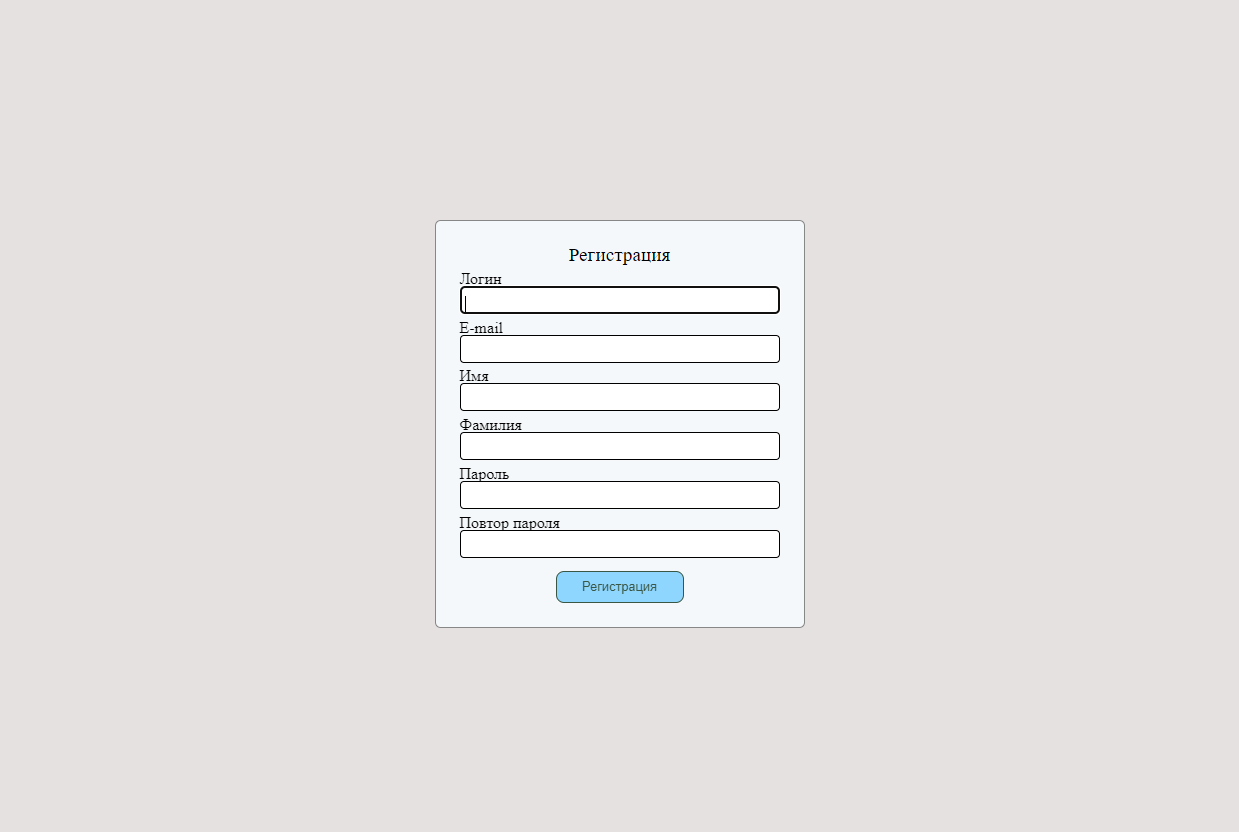
В последнюю очередь были настроены css стили для html шаблонов.

И указаны адреса статических файлов.



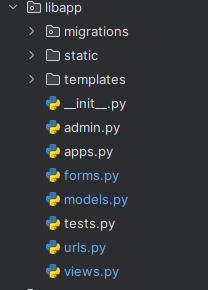
Если запустить проект и открыть его страницу в браузере, то увидим вот такие странички.





* 1. **Создание основного приложения**

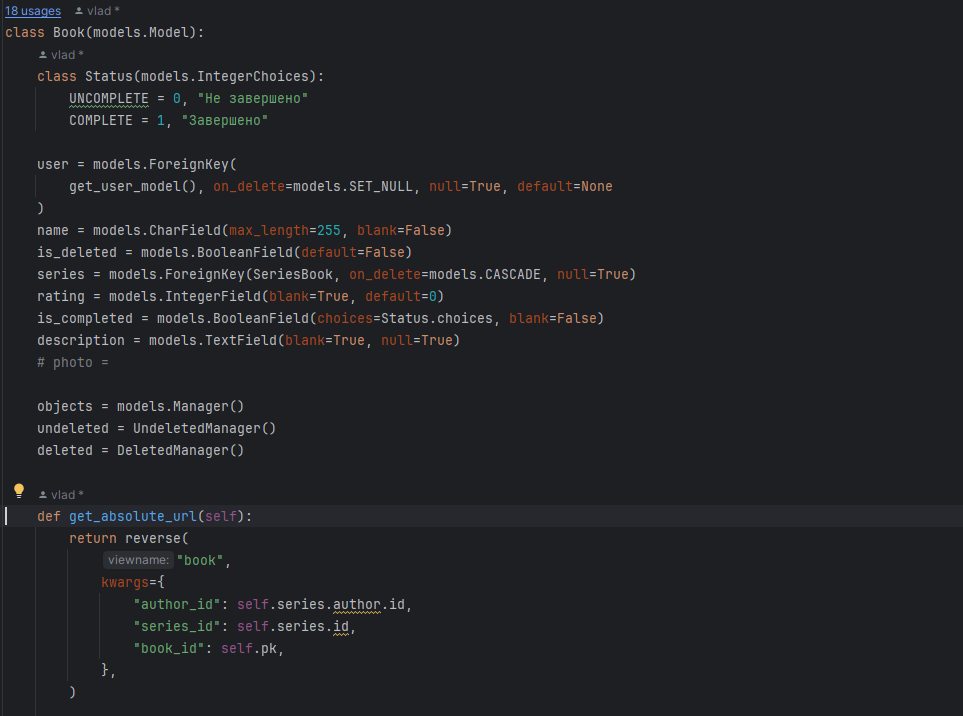
Сначала было создано приложение libapp в котором и будет основная часть.



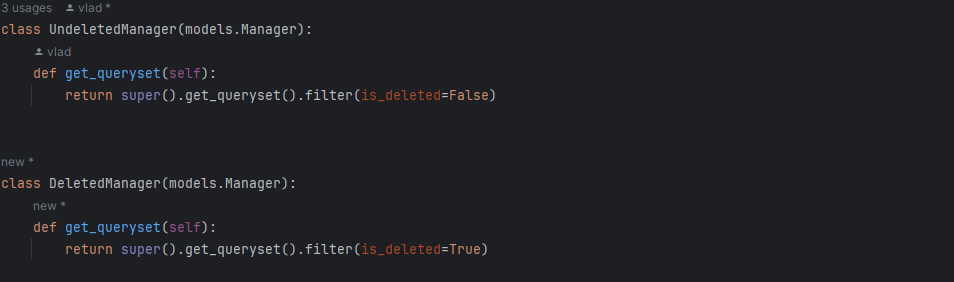
Первым делом были созданы модели для авторов, серий книг и самих книг.







А также менеджеры для этих моделей.

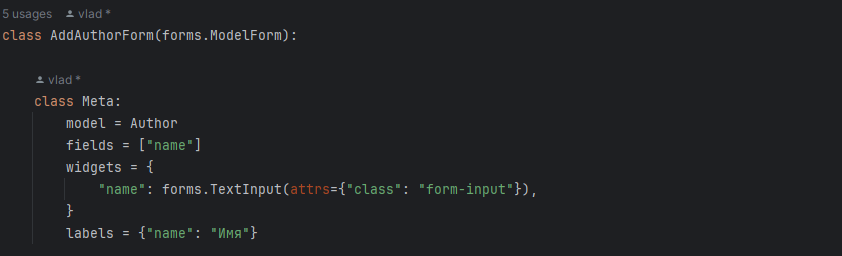


У модели автор есть только два поля, name – имя, и is\_deleted отвечающий за функцию удаления автора, is\_deleted есть у всех трех моделей, оно сделано ради того чтобы при удалении не удалять саму запись в таблице, а просто помечать ее удаленной, это позволило бы в будущем сделать функцию восстановления удаленных записей, (но пока в программе такой функции не будет, так как это только можно сказать бета версия сайта, после сдачи диплома, сайт будет дорабатываться еще больше). Также у модели есть менеджеры, отвечающие за “удаленные” и “не удаленные” записи, оно также есть у всех трех моделей.

У модели серия книг есть поля name – имя, rating – оценка, is\_completed – статус, description – описание, is\_deleted и author – внешний ключ к модели автор, а также менеджеры.

У модели книга похожие поля, name – имя, rating – оценка, is\_completed – статус, description – описание, is\_deleted и series – внешний ключ к модели серия книг, а также менеджеры.

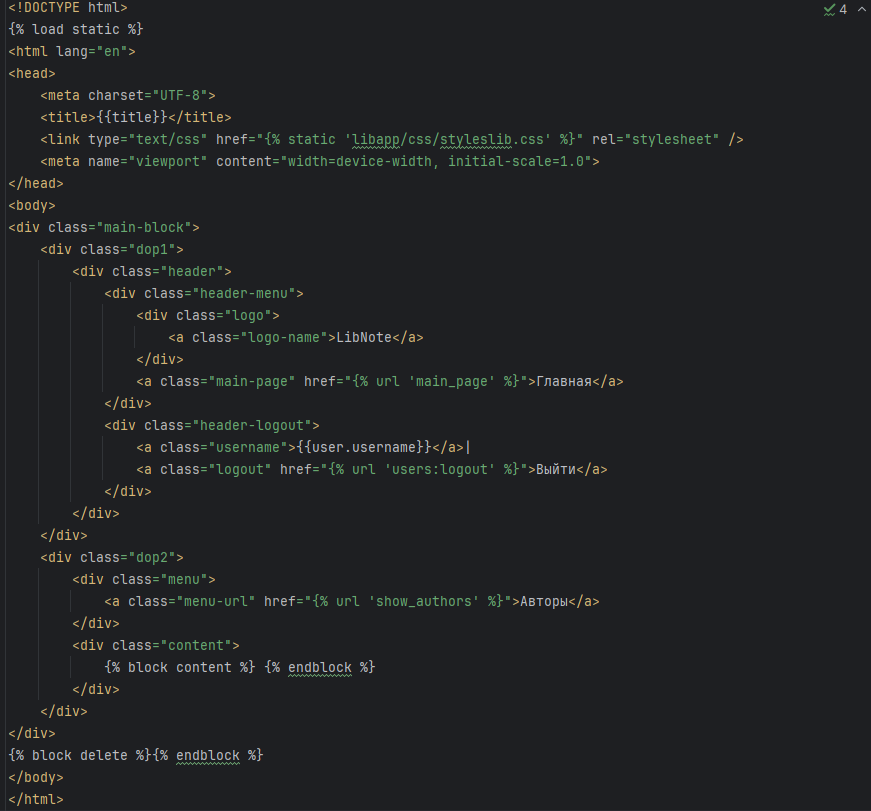
Потом для моделей я создал три формы.







Шаблон base для всех представлений приложения libapp.



* 1. **Представления Create**

(Примечание: этот диплом пишется уже после завершения работы над веб-приложением, по ходу разработки я много дорабатывал представления поэтому не смогу в точности расписать порядок выполнения работы, так что дальше будет описываться только результат)

CreateAuthor, CreateSeriesBook и CreateBook. Они наследуются от базового класса CreateView, а также от класса LoginRequiredMixin который проверяет авторизован пользователь или нет, (все остальные представления тоже унаследованы от этого миксина).



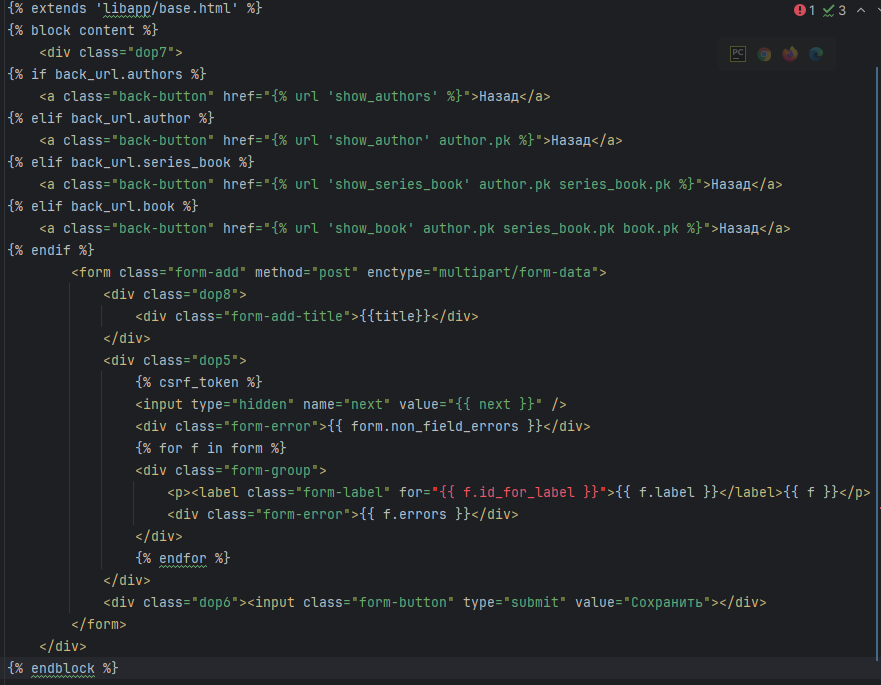




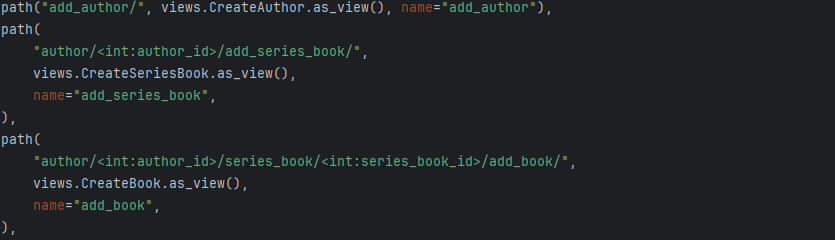


Здесь также есть метод get\_succes\_url который перенаправляет пользователя на страницы которые я еще не описал, которые показывают список с для которого создаются автор, серия или книга, и метод get\_context\_data который позволяет использовать в html файле дополнительные данные, из таких словарь back\_url который определяет на какую страницу будет перенаправлен пользователь при нажатии кнопки “назад” на странице создания автора, серии книг или книги, потому что для всех трех представлений используется один и тот же html шаблон, а также он используется для редактирования данных, также передаются author, series\_book и book, для работы ссылки у кнопки “назад”, также передается переменная title, которая отображается в теге <title></title> в html шаблоне (эта переменная есть во всех представлениях). Также в этом методе проверяется неправильность url-адреса.

И шаблон create для создания записей.

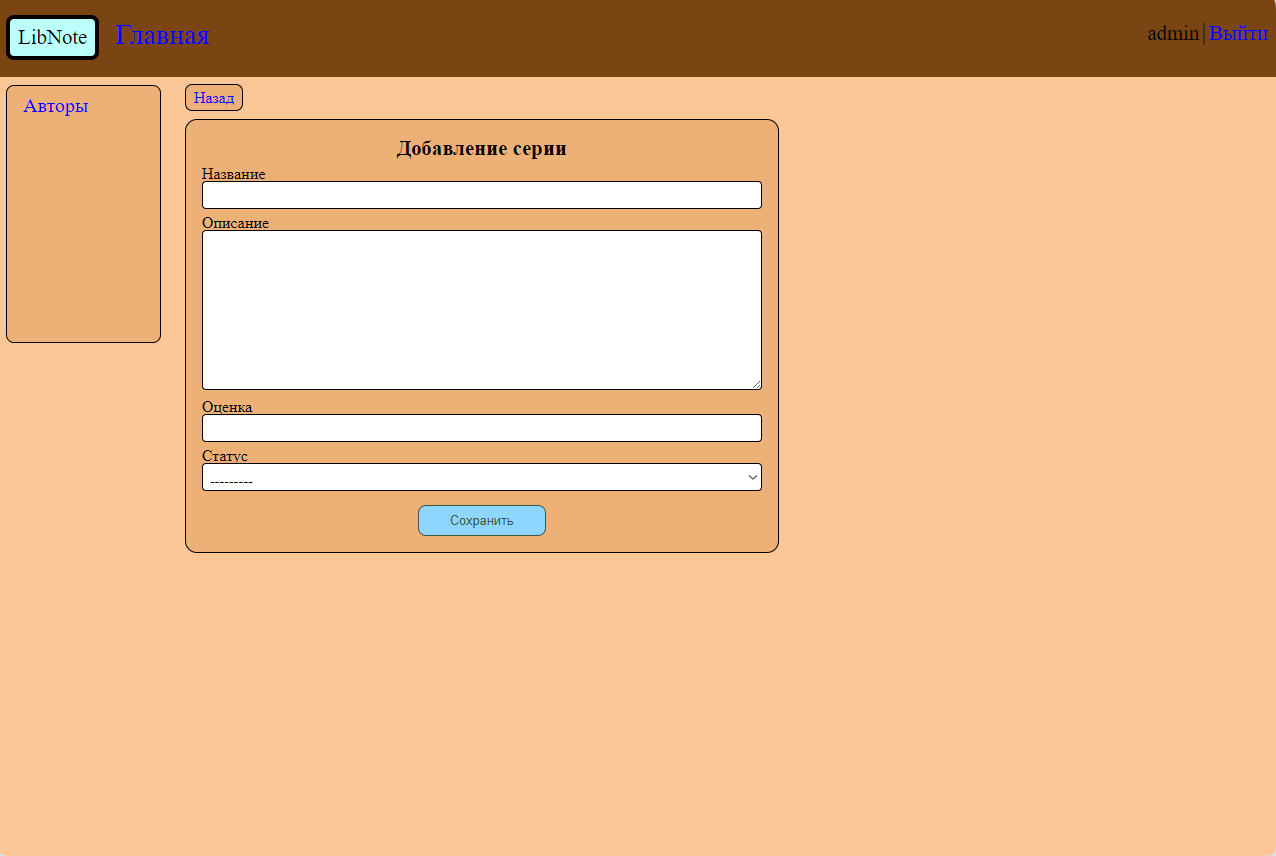


url-адреса для этих представлений.



Эти страницы в браузере выглядят так.



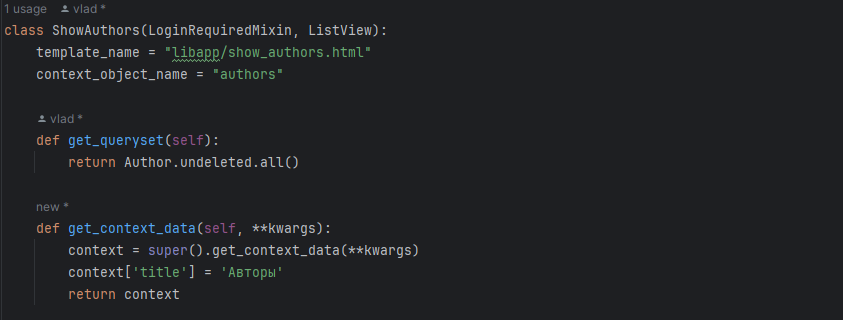


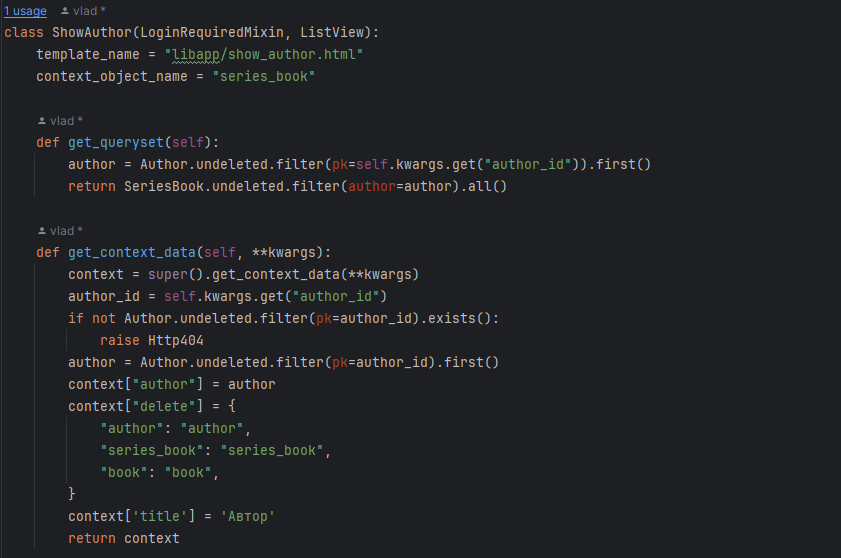


(Примечание: Маленькое поле с надписью авторы, это меню, оно добавлено с планами на будущее когда я добавлю функции избранное и корзина, но это будет потом).

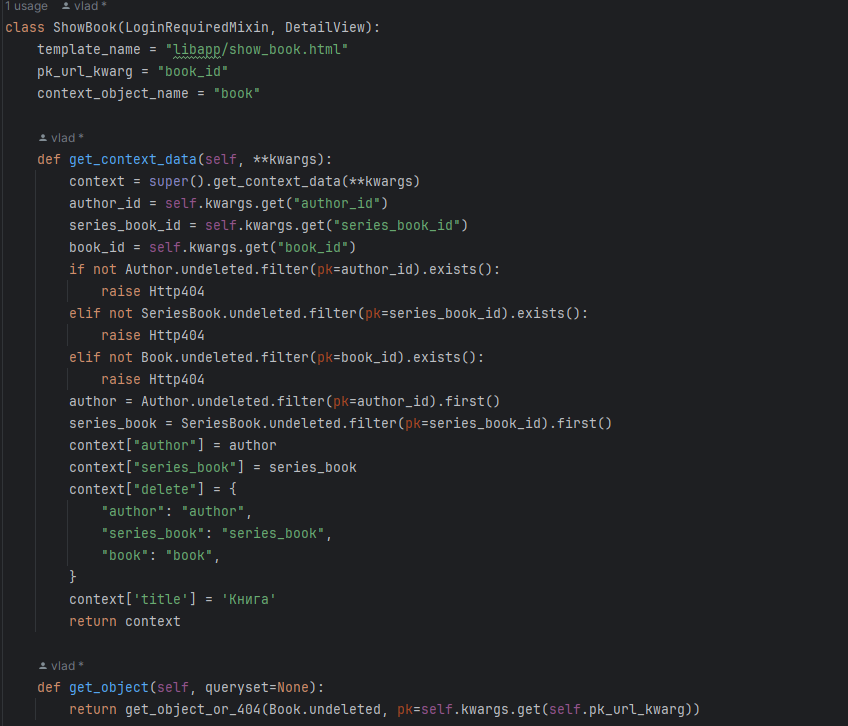
* 1. **Представления Read**

Классы представления ShowAuthors, ShowAuthor, ShowSeriesBook и ShowBook.









Первые три наследуются от класса ListView, а последний от класса DetailView.

ShowAuthors показывает список всех авторов.

ShowAuthor показывает выбранного автора и все его серии книг.

ShowSeriesBook показывает выбранного автора, выбранную серию и все книги этой серии.

ShowBook показывает выбранного автора, выбранную серию книг, и подробности о выбранной книге.

Во всех представлениях есть функция get\_context\_data, в которой есть словарь delete со значениями author, series\_book и book (кроме первого), они нужны для работы ссылок в соответствующих html шаблонах. Так как около имени автора, названия серии книг и названия книги, находится кнопки “отредактировать” и “удалить”, отсылающие на соответствующие страницы. При этом получается что на странице show\_book есть три кнопки “отредактировать” и три кнопки “удалить”, для автора, серии книг и книги, и также эти кнопки для автора можно найти на странице show\_series\_book и show\_author, так что возникает путаница с тем куда перенаправлять пользователя после того как будет произведено редактирование или удаление, поэтому добавление переменной delete в ссылку помогает в этом (подробнее при описании соответствующих представлений), также есть переменные author, series\_book и book, для осуществления работы ссылок.

Шаблоны для этих представлений.

show\_authors.

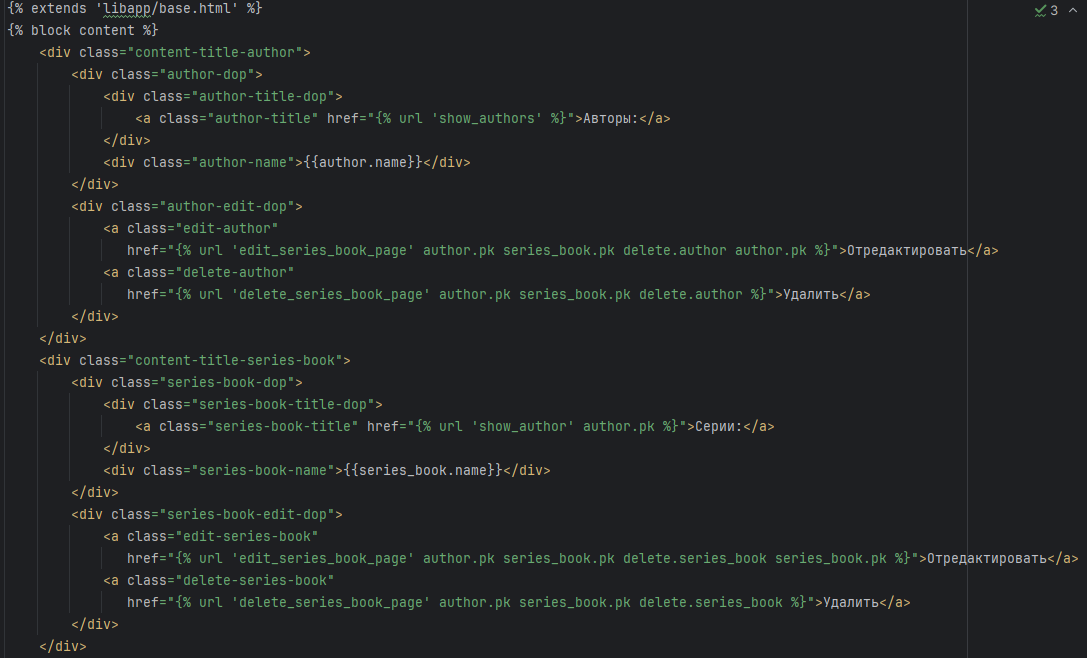


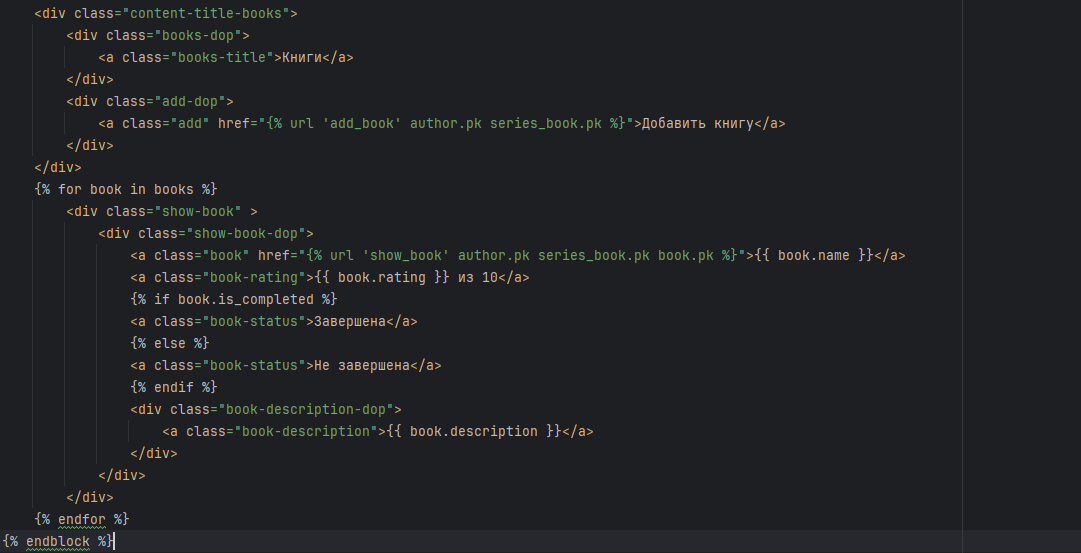
show\_author.





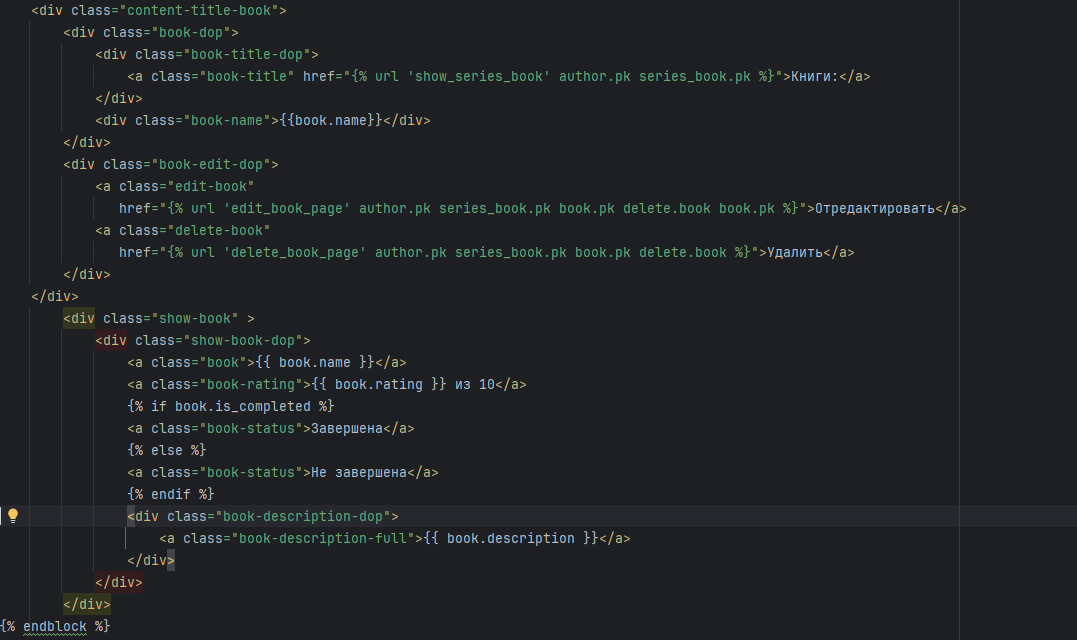
show\_series\_book.





show\_book.

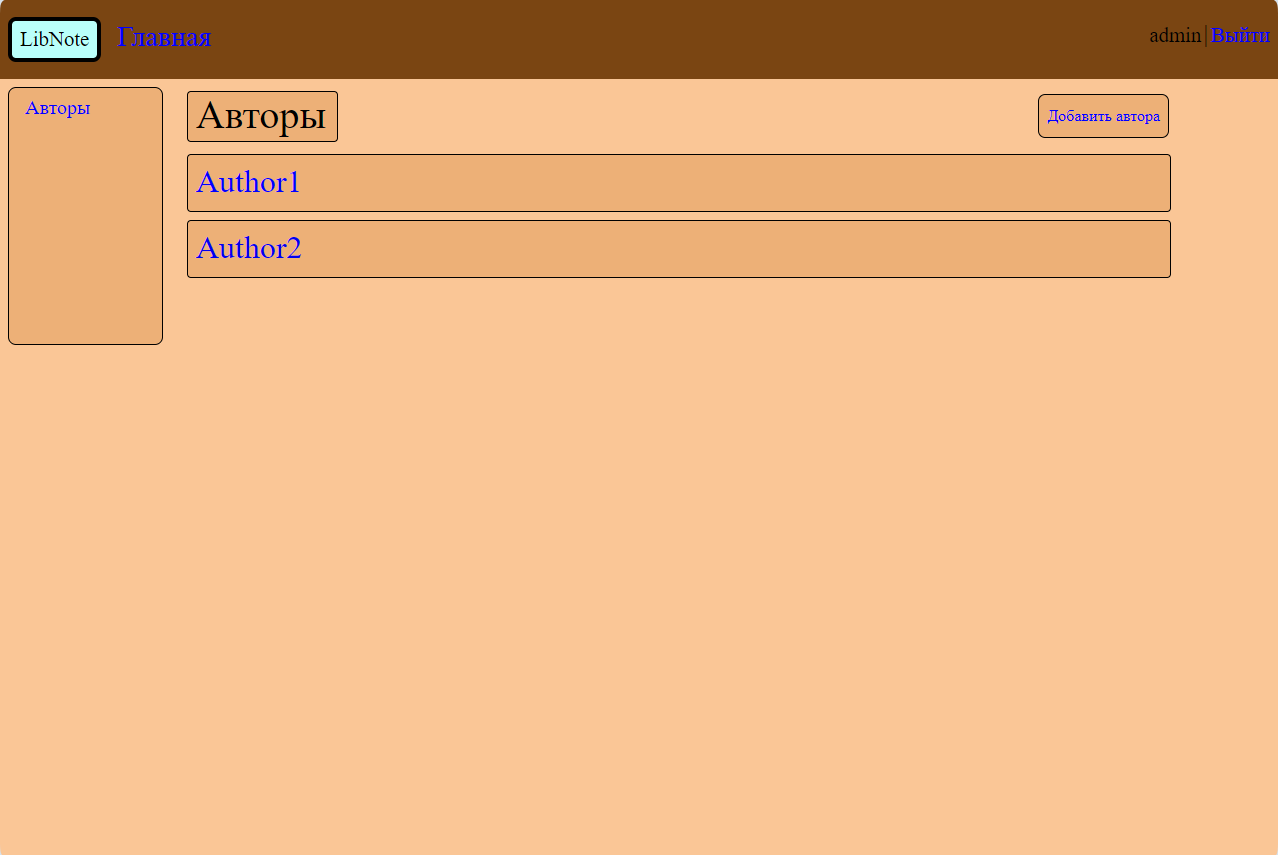




url-адреса.



В браузере эти страницы выглядят вот так.





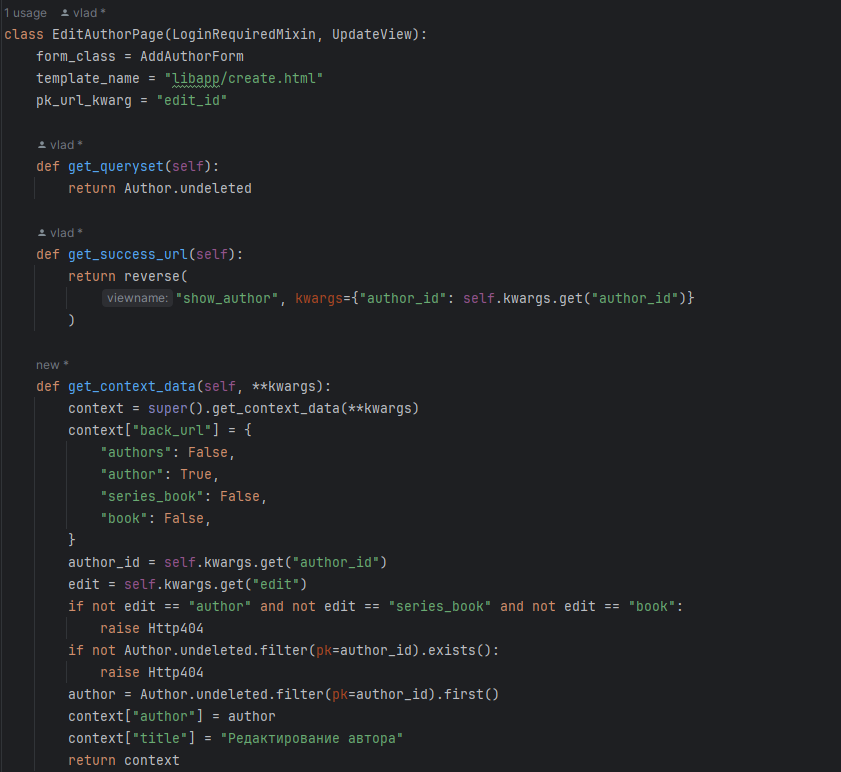




(Примечание: Это просто примеры, которые я добавил для демонстрации, для описаний взяты случайные страницы с википедии).

* 1. **Представления Update**

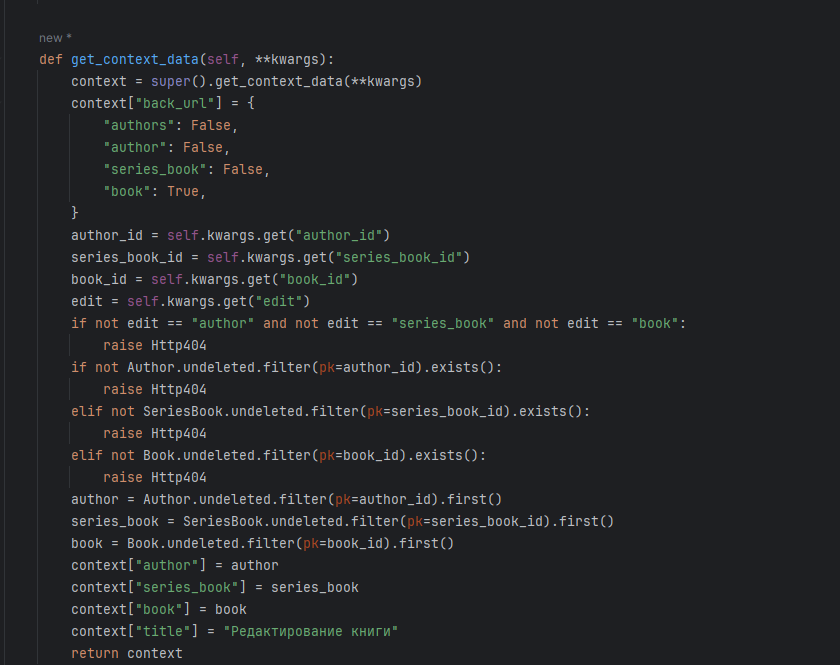
Классы представления EditAuthorPage, EditSeriesBookPage и EditBookPage.











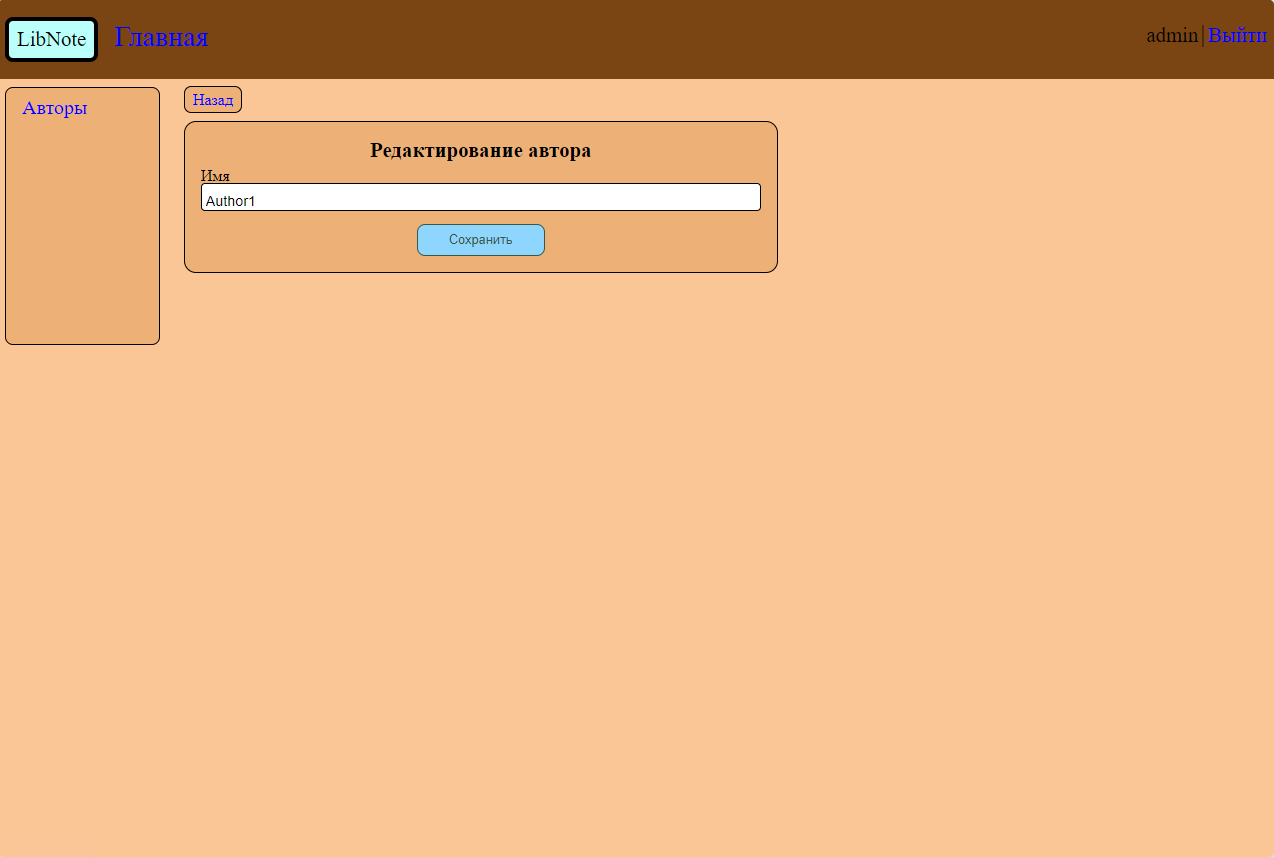
Название обозначает то с какой страницы открывается текущее представление, а то с чем оно работает определяется переменной edit в url-адресе, это тот самый словарь delete из Show представлений, так например на странице show\_book кнопки “отредактировать” работали с url-адресом под названием edit\_book\_page, просто передавая в переменную edit, разные значения, author, series\_book и book.

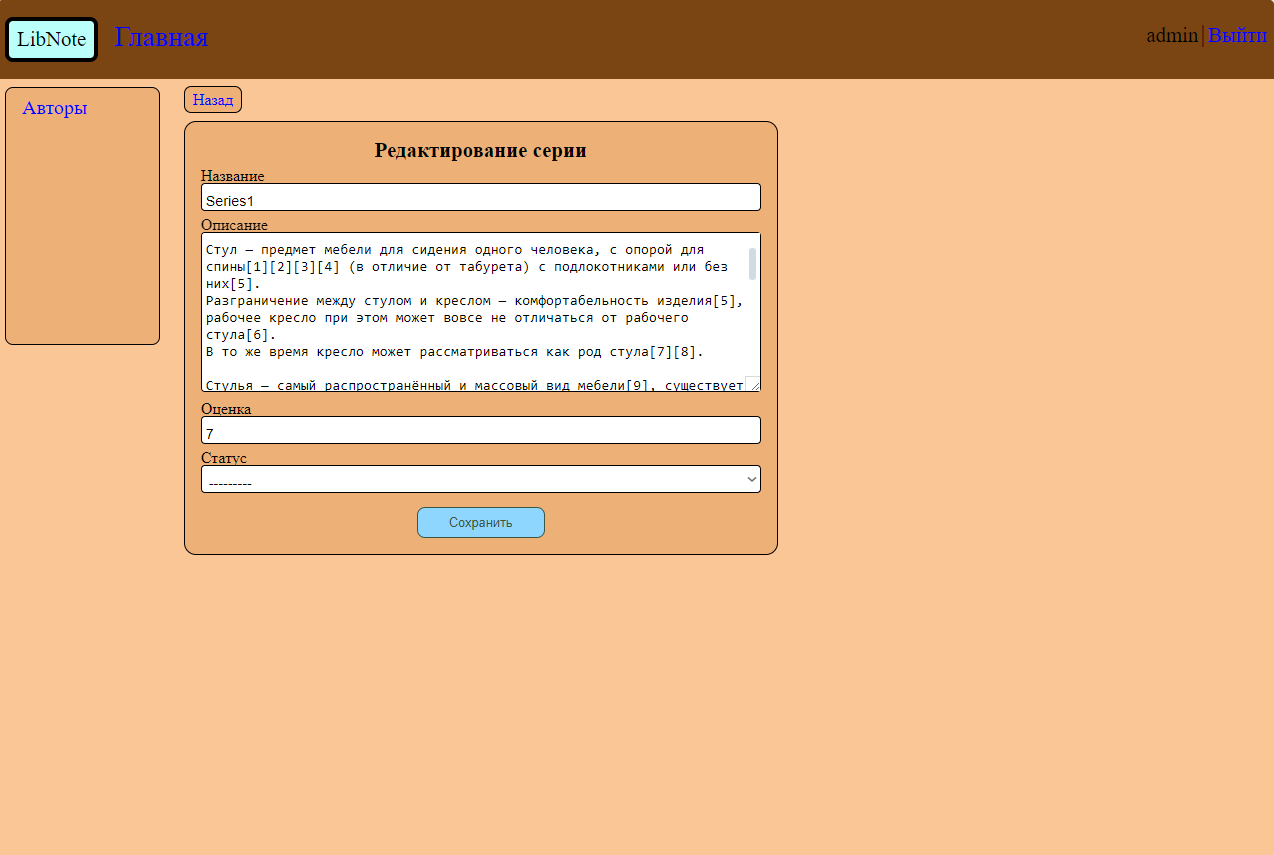
Так же как и в представлениях Create тут есть функция get\_context\_data со словарем back\_url для кнопки “назад”, author, series\_book и book для работы ссылок и title, и проверка правильности url адресов.

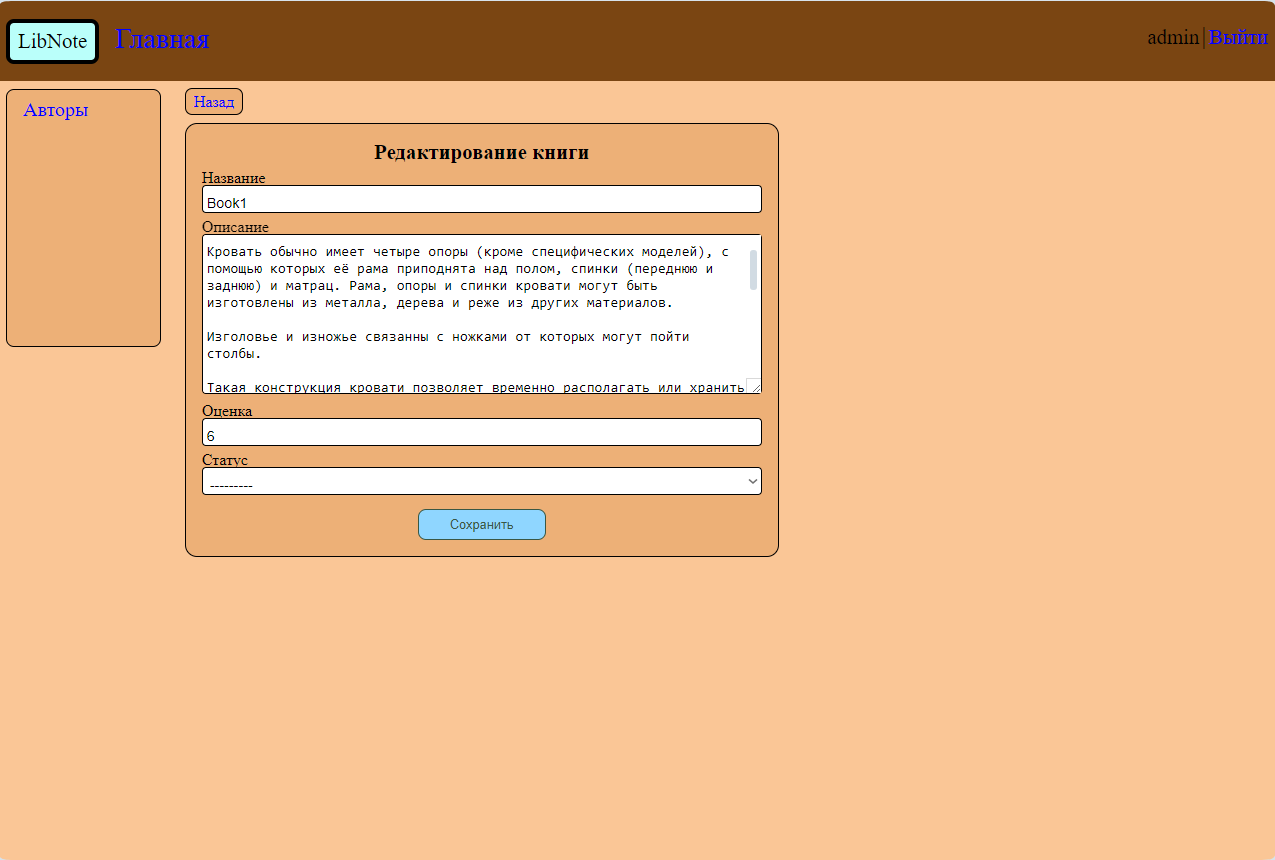
Вот url-адреса.



Это представление работает с тем же html шаблоном create с которым работали представления Create.



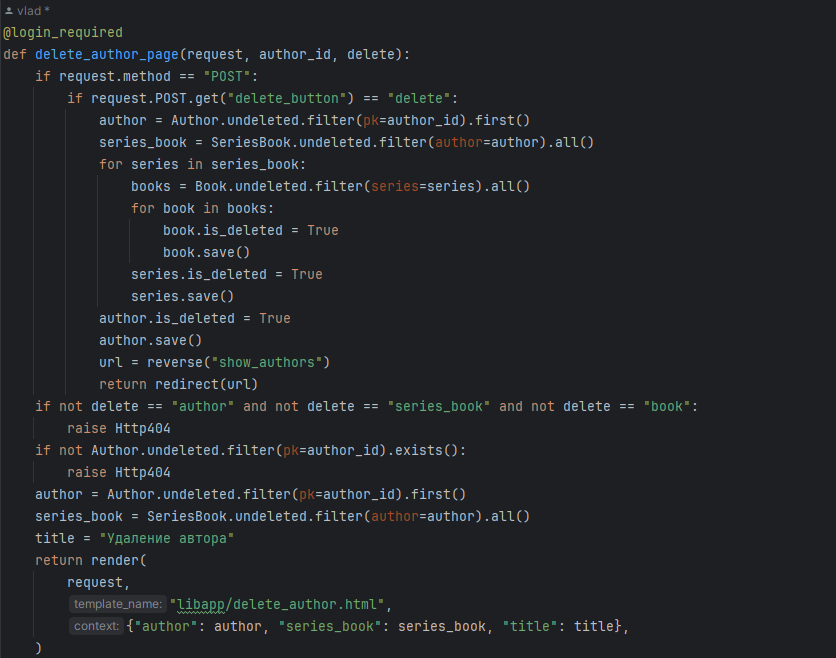


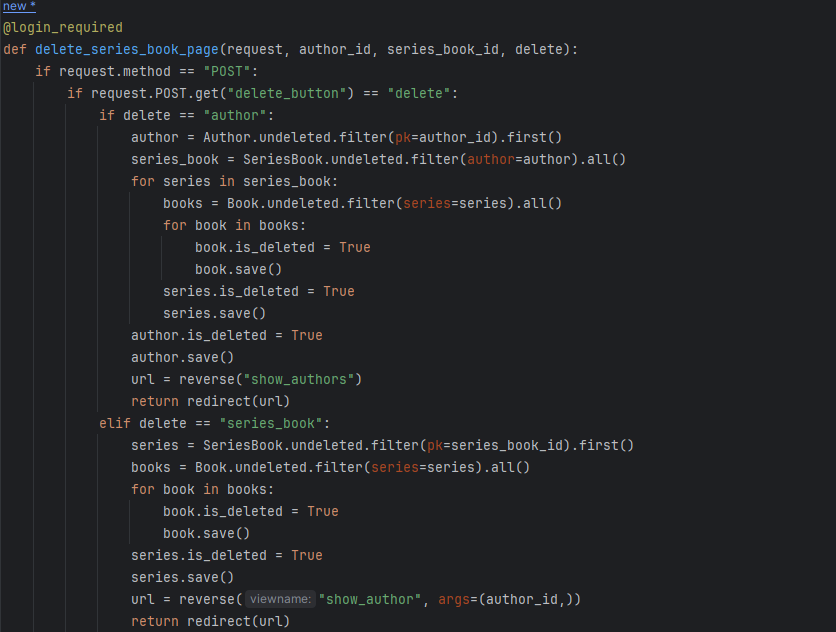


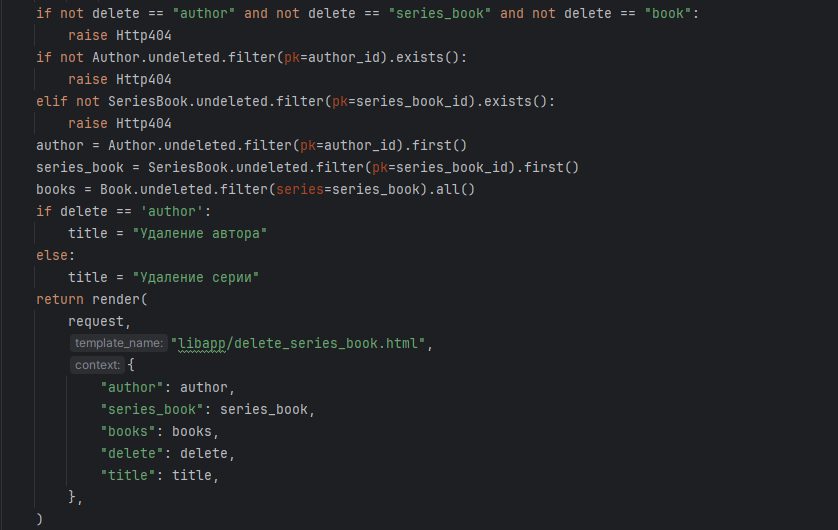
* 1. **Представления Delete**

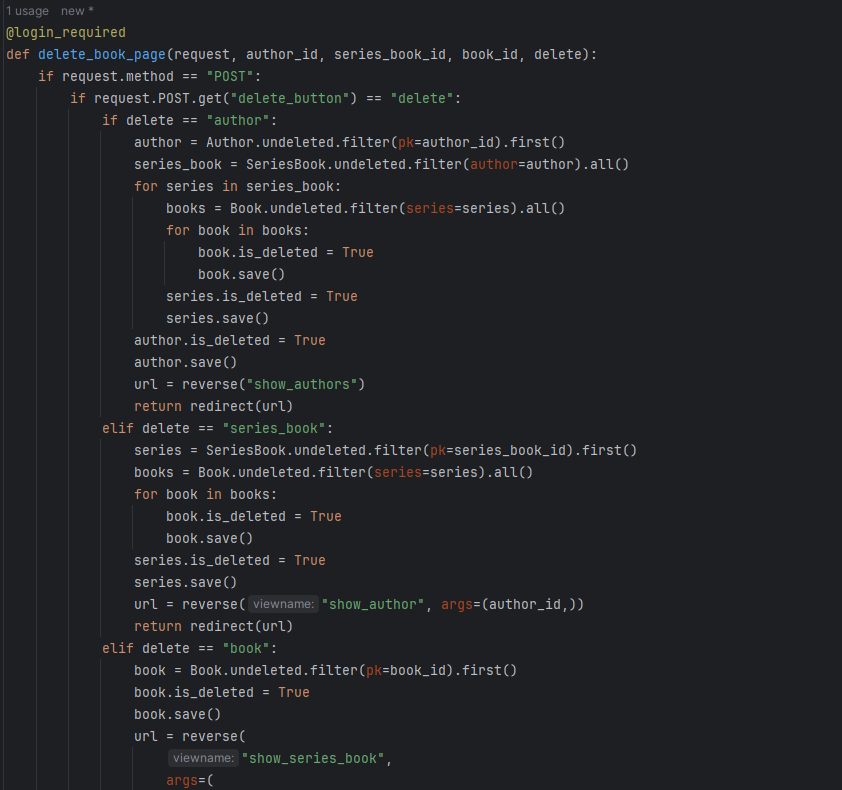
(Примечание: Представления delete, я сделал в виде функций, потому что недостаточно хорошо был знаком с классом DeleteView, и в начале разработки не совсем понимал как должен быть реализован механизм тот механизм удаления который я планировал в классе представлении, поэтому сделал его в виде функции, возможно в дальнейшем придется заменить их на классы).

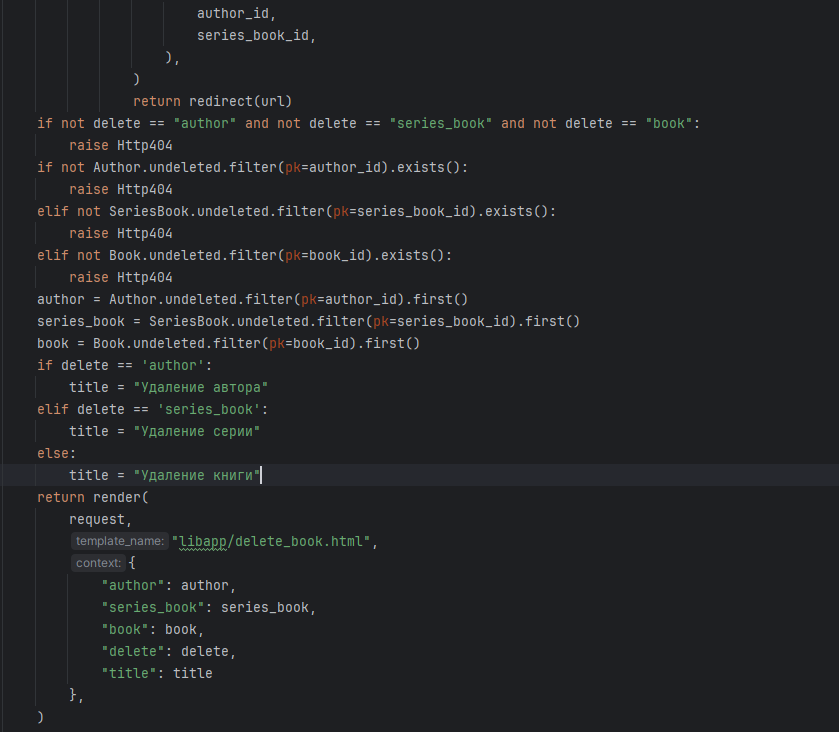
Функции представления delete\_author\_page, delete\_series\_book\_page и delete\_book\_page.







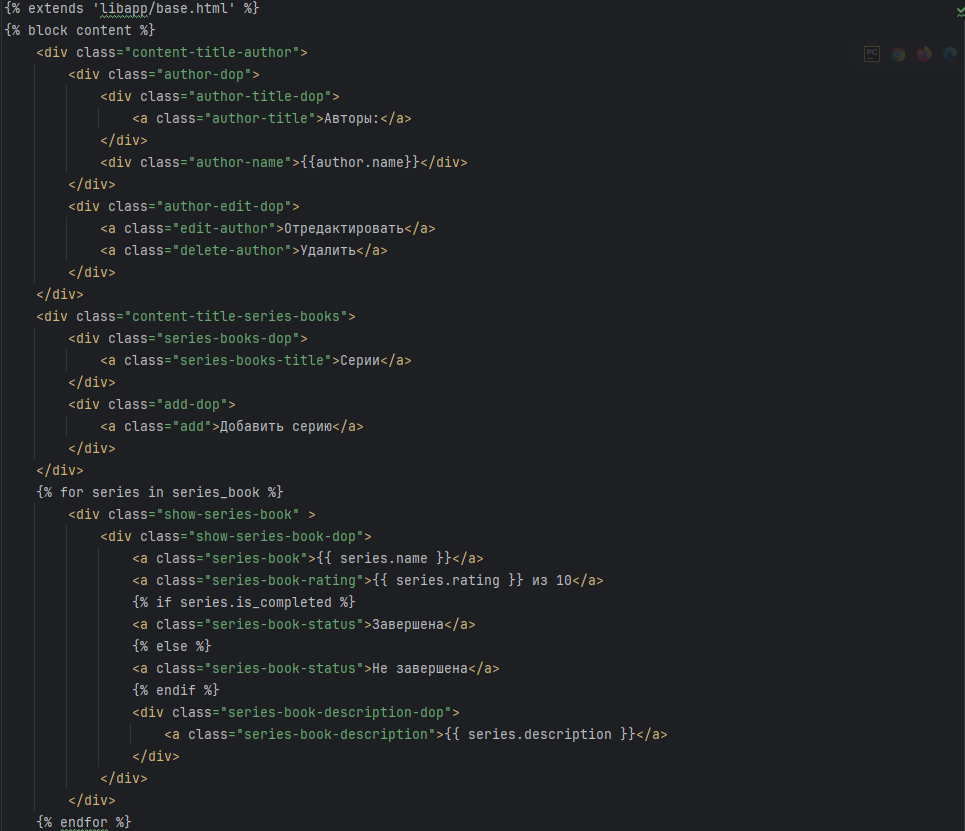


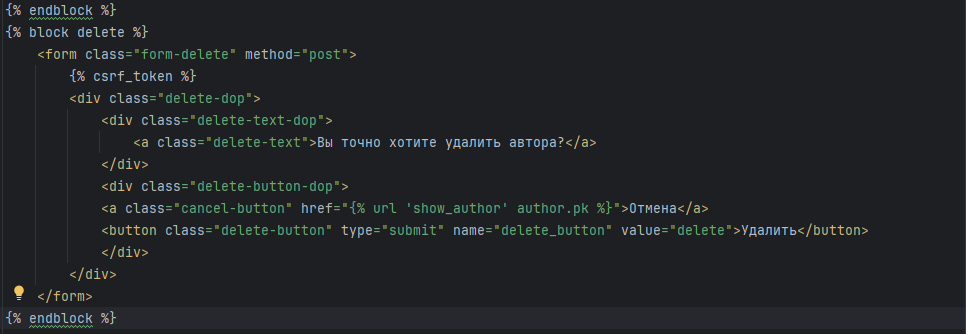


Смысл этих представлений в том что когда кто-то например нажимает кнопку “удалить” автора на странице show\_book, то должна открыться страница которая в точности повторяет данные на странице show\_book, только все ссылки и кнопки будут не активны, а по середине экрана появится окно с кнопками “отмена” и “удалить”, соответственно отменяющие удаление и удаляющие автора, поэтому так же как с представлениями Edit, названия представлений delete отражают то на какой странице была нажата кнопка удалить и на фоне какой страницы появится окно с удалением. Тут также есть проверка правильности url-адреса и дополнительные значения author, series\_book и book, для работы url-адресов и title.

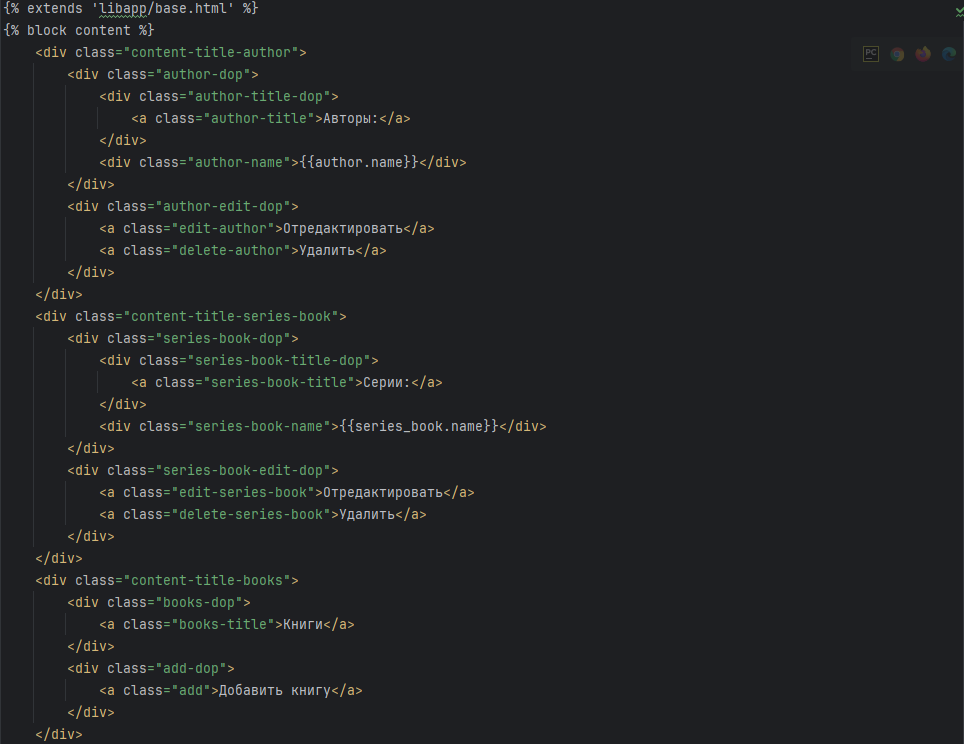
Вот html шаблоны для этих представлений.

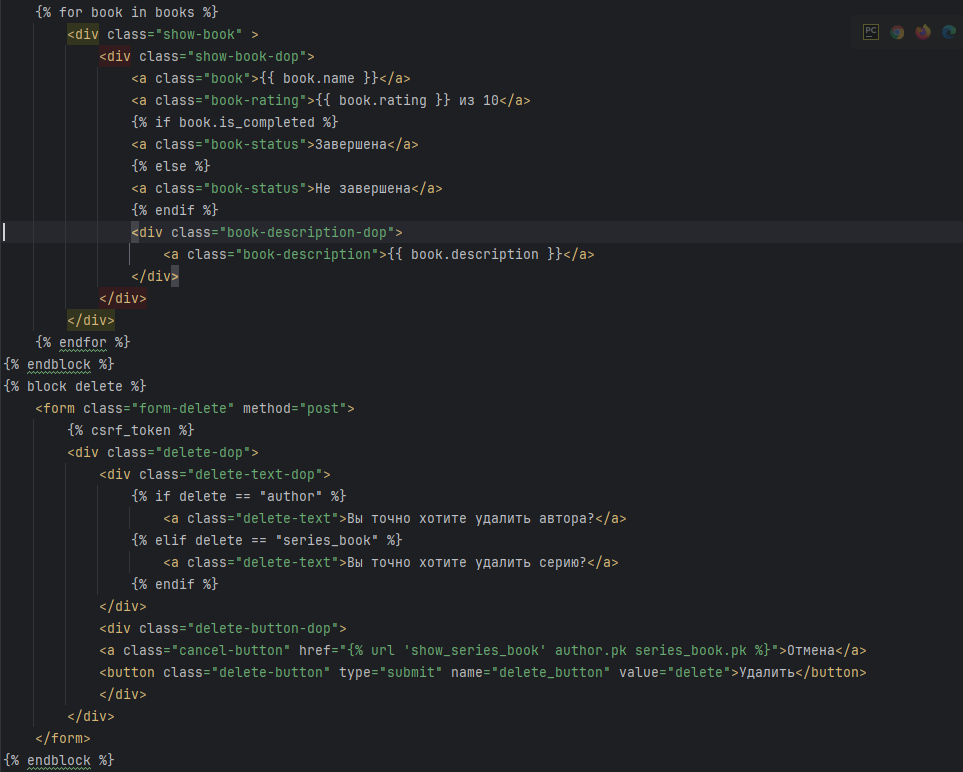
delete\_author.



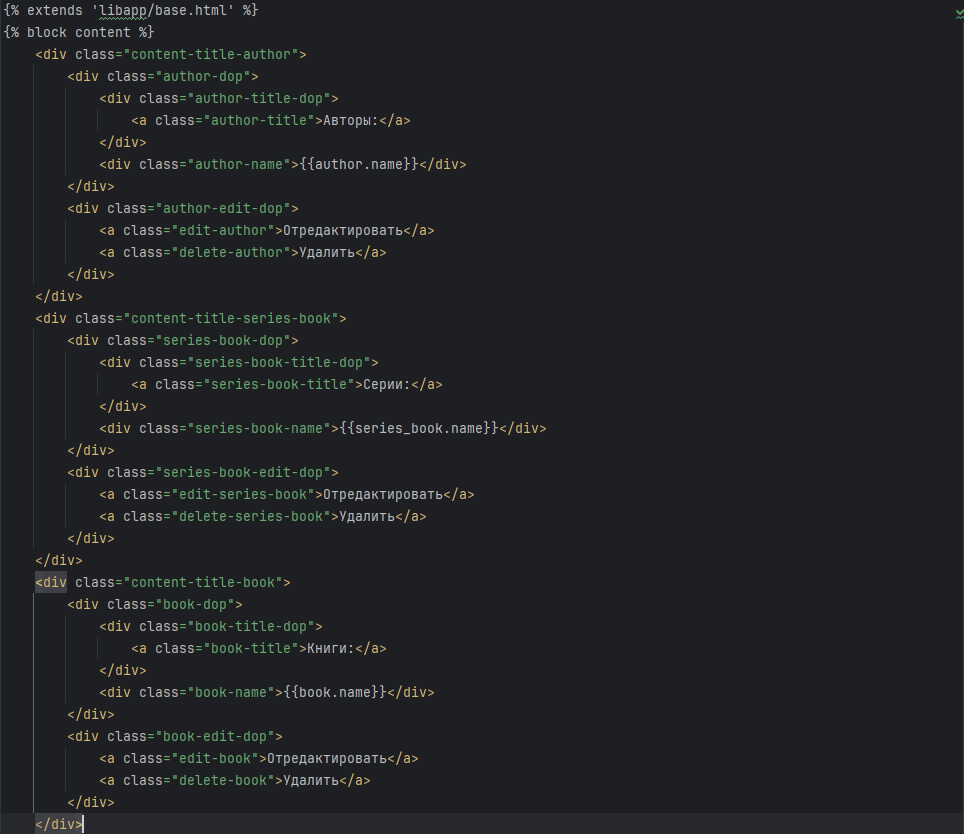


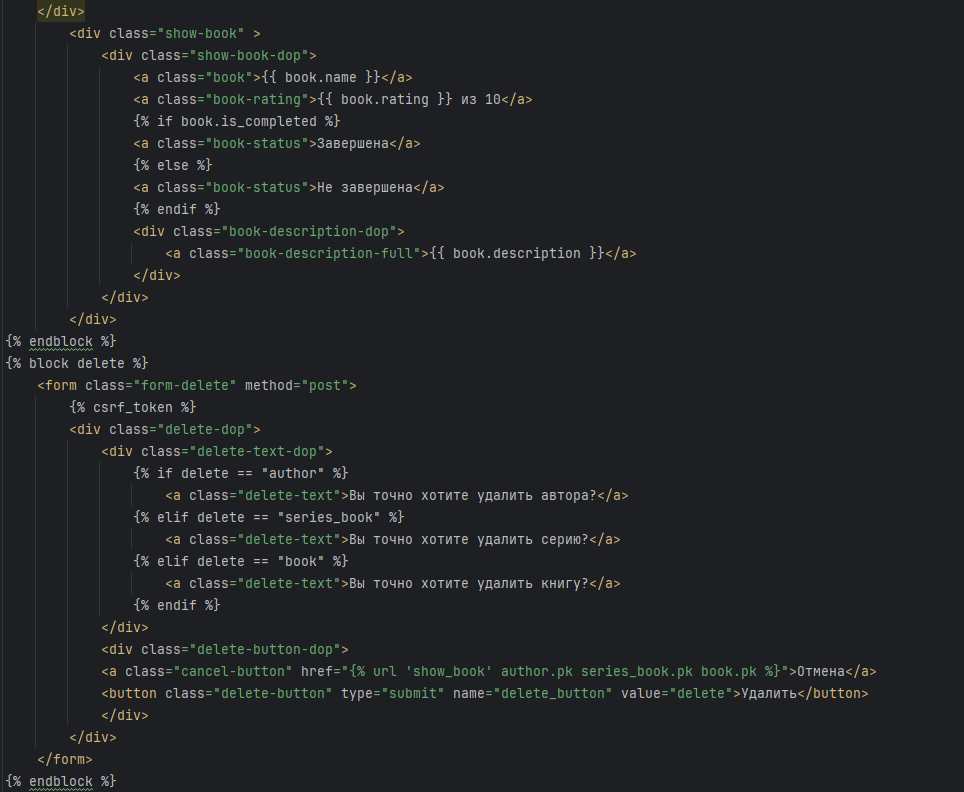
delete\_series\_book.



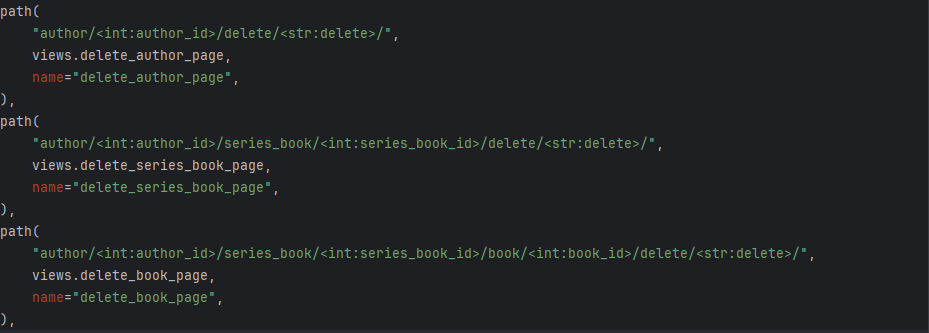


delete\_book.





Вот url-адреса.



Вот так работают эти представления.





**ЗАКЛЮЧЕНИЕ**

Создать простое веб-приложение оказалось довольно легко, легче чем я думал, хотя конечно все прошло не без проблем, но они решились с помощью помощи интернета, в частности различных обучающих видео, хотя конечно некоторые проблемы так и остались не решенными, все таки я делал проект в одиночку. У моего веб-приложения есть еще куда расти, я даже не сделал функцию позволяющую пометить какую-нибудь книгу как избранное или любимое. Но Django это и правда довольно гибкий фреймворк, изначально некоторые функции я думал будут трудно выполнимыми, но оказалось все проще. Было бы наверно интересно попытаться сделать из моего проекта реальное веб-приложение которым будут пользоваться другие люди.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

Видео: Сергей Балакирев (selfedu). «Django 4 (Уроки)» (73 видео).

Статья: Александр Юркин. «Виды веб-приложений». Дата: 14.10.2022.

Сайт: developer.mozilla.org. «Django введение - Изучение веб-разработки | MDN».

Сайт: docs.djangoproject.com. «Django documentation | Django documentation | Django».

Сайт: stackoverflow.com. «Stack Overflow - Where Developers Learn, Share, & Build...».