

Flask для начинающих — Часть 2 пишем landing page+admin panel с редактированием контента

 habr.com/ru/articles/784770

m_nikitin_dev

January 6, 2024

Простой

40 мин

37K

Тutorial

Цель статьи: В этой статье мы разработаем небольшой landing page с admin panel, оснащенной системой авторизации. Вы научитесь изменять контент фронтенда сайта через админ-панель. Основная цель - показать начинающим разработчикам, как можно быстро и эффективно создать функциональный сайт на Flask с возможностью редактирования контента.

Что вы научитесь:

1. Создание скелета landing page на Bootstrap 5.
2. Разработка admin panel с функционалом авторизации.
3. Реализация функционала для редактирования контента на лендинге.

Требуемые навыки:

- Основы **Python**: понимание основных конструкций языка и способность писать простые программы.
- Знакомство с **HTML**: базовое понимание тегов и структуры веб-страниц.
- Рекомендуется ознакомиться с первой статьей: Flask для начинающих - [Flask для начинающих — Часть 1 настройка](#)

Ориентировочное время чтения: Примерно 30-40 минут.

Введение

Прежде всего, мы создадим основу для нашего сайта, используя Bootstrap 5. Начнем с разработки статического HTML для landing page, а затем перейдем к структурированию проекта на Flask, разделив наш HTML на компоненты и подключив базу данных SQLite для управления контентом.

Для начала мы создадим заготовки html на bootstrap 5.

Далее создадим структуру Flask проекта и разобьем наш html на сегменты часть которые будем тянуть с базы данных SQLite

Шаг 1: Создание Landing Page (на основе бесплатного шаблона)

Начнем с создания landing page, опираясь на готовые шаблоны с официального сайта Bootstrap. В качестве примера возьмем шаблон [Carousel](#).

Создание HTML-страницы Landing Page на основе шаблона Carousel из Bootstrap 5

Для создания HTML-страницы landing page, мы воспользуемся шаблоном Carousel, доступным на официальном сайте Bootstrap. Вот примерный код, который вы можете использовать и адаптировать под свои нужды.

вы можете взять код через inspector кода со страницы шаблона <https://getbootstrap.com/docs/5.0/examples/carousel/>

или взять готовый упрощенный мой вариант :
просто создав html файл и вставить туда следующий код:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bootstrap Page</title>
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        .carousel-inner {
            max-width: 1024px;
            margin: auto;
            height: 500px; /* Высота карусели */
        }
        .carousel-inner img {
            width: 100%;
            height: 100%;
            object-fit: cover; /* Обрезка или масштабирование изображения */
        }
        .carousel-caption {
            text-align: left; /* Выравнивание текста по левому краю */
            bottom: 30%; /* Приподнять текст и кнопку вверх */
            right: 40%; /* Центрирование по правому краю */
            transform: translateX(0%); /* Корректировка положения */
            background-color: rgba(0, 0, 0, 0.1); /* Небольшой фон для улучшения
читаемости */
            padding: 10px;
        }
    </style> </head>
<body>
<!-- Header -->
<header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Navbar</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <!-- в href указываем id div block куда нам нужно сделать
скролл -->
                        <a class="nav-link"
href="#carouselExampleIndicators">Carousel</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="#cards">Cards</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="#featurette1">Featurette 1</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

```

```

                <li class="nav-item">
                    <a class="nav-link"
href="#featurette2">Featurette 2</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
</header> <!-- Carousel With Indicators -->
<div id="carouselExampleIndicators" class="carousel slide" data-bs-
ride="carousel">
    <!-- Indicators -->
    <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="0" class="active"
            aria-current="true" aria-label="Slide 1"></button>
        <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="1"
            aria-label="Slide 2"></button>
        <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="2"
            aria-label="Slide 3"></button>
    </div> <!-- Carousel items -->
    <div class="carousel-inner">
        <div class="carousel-item active">
            
            <div class="carousel-caption d-none d-md-block">
                <h5>First Slide</h5>
                <p>Description for the first slide.</p>
                <button class="btn btn-primary">Click Me</button>
            </div>
        </div>
        <div class="carousel-item">
            
            <div class="carousel-caption d-none d-md-block">
                <h5>Second Slide</h5>
                <p>Description for the second slide.</p>
                <button class="btn btn-primary">Click Me</button>
            </div>
        </div>
        <div class="carousel-item">
            
            <div class="carousel-caption d-none d-md-block">
                <h5>Third Slide</h5>
                <p>Description for the third slide.</p>
                <button class="btn btn-primary">Click Me</button>
            </div>
        </div>
    </div> <!-- Controls -->
    <button class="carousel-control-prev" type="button" data-bs-
target="#carouselExampleIndicators"
        data-bs-slide="prev">
        <span class="carousel-control-prev-icon" aria-hidden="true">

```

```

</span>
        <span class="visually-hidden">Previous</span>
    </button>
    <button class="carousel-control-next" type="button" data-bs-
target="#carouselExampleIndicators"
        data-bs-slide="next">
        <span class="carousel-control-next-icon" aria-hidden="true">
</span>
        <span class="visually-hidden">Next</span>
    </button>
</div> <!-- Cards in grid -->
<section id="cards" class="container mt-4">
    <div class="row">
        <!-- Card 1 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card Title 1</h5>
                    <p class="card-text">Some quick example
text to build on the card title and make up the bulk of the
                    card's content.</p>
                    <a href="#" class="btn btn-primary">Go
somewhere</a>
                </div>
            </div>
        </div>
        <!-- Card 2 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card Title 2</h5>
                    <p class="card-text">Some quick example
text to build on the card title and make up the bulk of the
                    card's content.</p>
                    <a href="#" class="btn btn-primary">Go
somewhere</a>
                </div>
            </div>
        </div>
        <!-- Card 3 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card Title 3</h5>
                    <p class="card-text">Some quick example
text to build on the card title and make up the bulk of the

```

```

card's content.</p>
<a href="#" class="btn btn-primary">Go
somewhere</a>

</div>

</div>

</div>

</div>
</section> <!-- Horizontal Card with image left and text right -->
<section id="featurette1" class="container mt-4">
  <div class="row align-items-center">
    <div class="col-md-6">
      
    </div>
    <div class="col-md-6">
      <!-- Text -->
      <h2 class="featurette-heading">Заголовок раздела</h2>
      <p class="lead">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sed nec elementum leo. Cras
tincidunt magna ut aliquam tincidunt. Nunc lacinia
turpis ac neque porta, a fermentum neque facilisis.
Curabitur in felis nec ipsum sollicitudin
fermentum. Sed ultrices, nunc vel mollis dapibus, lacus risus
fermentum massa, id condimentum justo sem eget
leo. Aenean efficitur, sapien in sodales cursus, ligula
magna cursus lorem, a elementum nulla orci nec
nunc.</p>
    </div>
  </div>
</section> <hr class="featurette-divider"> <!-- Horizontal Card with image right
and text left -->
<section id="featurette2" class="container mt-4">
  <div class="row align-items-center">
    <div class="col-md-6">
      <!-- Text -->
      <h2 class="featurette-heading">Заголовок раздела</h2>
      <p class="lead">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sed nec elementum leo. Cras
tincidunt magna ut aliquam tincidunt. Nunc lacinia
turpis ac neque porta, a fermentum neque facilisis.
Curabitur in felis nec ipsum sollicitudin
fermentum. Sed ultrices, nunc vel mollis dapibus, lacus risus
fermentum massa, id condimentum justo sem eget
leo. Aenean efficitur, sapien in sodales cursus, ligula
magna cursus lorem, a elementum nulla orci nec
nunc.</p>
    </div>
    <div class="col-md-6">
      
    </div>
  </div>
</section> <hr class="featurette-divider"> <!-- Repeat of the first horizontal
card -->
<section id="featurette3" class="container mt-4">
  <div class="row align-items-center">

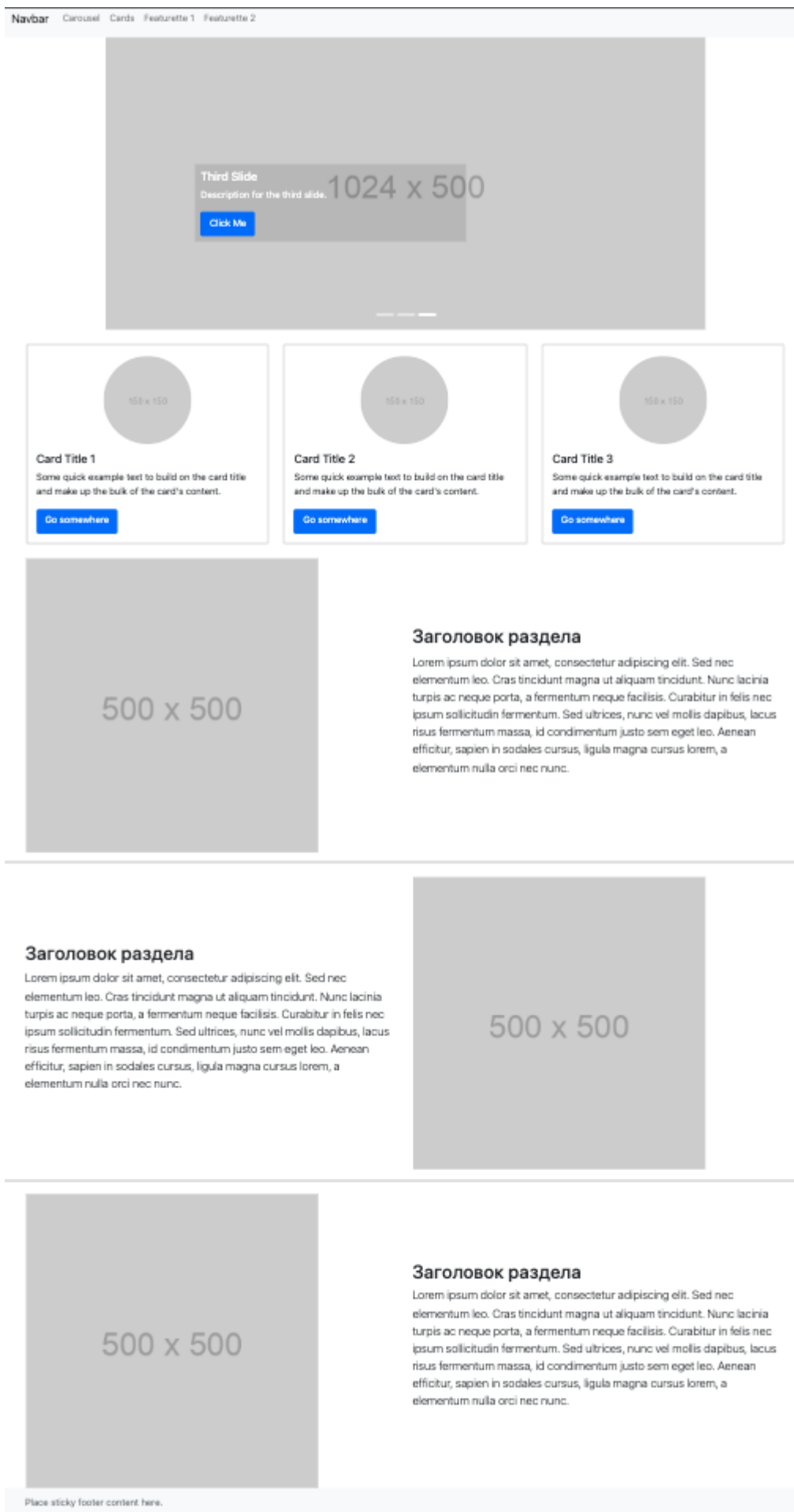
```

```

        <div class="col-md-6">
            
        </div>
        <div class="col-md-6">
            <!-- Text -->
            <h2 class="featurette-heading">Заголовок раздела</h2>
            <p class="lead">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sed nec elementum leo. Cras
                tincidunt magna ut aliquam tincidunt. Nunc lacinia
turpis ac neque porta, a fermentum neque facilisis.
                Curabitur in felis nec ipsum sollicitudin
fermentum. Sed ultrices, nunc vel mollis dapibus, lacus risus
                fermentum massa, id condimentum justo sem eget
leo. Aenean efficitur, sapien in sodales cursus, ligula
                magna cursus lorem, a elementum nulla orci nec
nunc.</p>
        </div>
    </div>
</section> <!-- Footer -->
<footer class="footer mt-auto py-3 bg-light">
    <div class="container">
        <span class="text-muted">Place sticky footer content here.</span>
    </div>
</footer> <!-- Bootstrap JS -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
></script> </body>
</html>

```

мы получаем примерный макет (стилистику вы можете доработать сами, главная цель показать как реализовать функционал):



Шаг 2: создание разметки для страницы логина в админ панель

В этом шаге мы разработаем страницу входа в административную панель. Эта страница будет содержать простую, но безопасную форму для входа, которая включает поля для ввода имени пользователя и пароля.

Важные аспекты страницы входа:

1. **Метод POST для Формы:** Для отправки данных формы мы будем использовать метод POST. Это критически важно для безопасности, поскольку метод POST не передает данные формы в URL, что предотвращает их видимость и логирование на сервере.

2. Различие между GET и POST:

- **GET:** Этот метод передает данные формы в URL. Он подходит для запросов безопасных данных, где конфиденциальность информации не является проблемой (например, при поиске на сайте).

Пример GET запроса

GET запросы часто используются для запроса данных с сервера. Данные, отправляемые через GET запрос, включаются в URL. Вот простой пример GET запроса, который может возникнуть при использовании поисковой строки на веб-сайте:

```
https://example.com/search?query=котики&sort=popular
```

В этом примере:

- `https://example.com/search` - это URL, на который отправляется запрос.
 - `?query=котики&sort=popular` - это строка запроса, содержащая данные. В данном случае есть два параметра: `query`, имеющий значение "котики", и `sort` со значением "popular".
- **POST:** В отличие от GET, POST передает данные в теле запроса и не отображает их в URL. Это делает POST более безопасным выбором для передачи конфиденциальных данных, таких как логины и пароли.

Пример POST запроса

POST запросы часто используются для отправки данных на сервер, например, при заполнении формы. В отличие от GET запросов, данные POST запроса включаются в тело запроса, а не в URL, что делает их более безопасными для конфиденциальной информации. Вот пример, как может выглядеть POST запрос на сервер при входе в систему:

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
username=alice&password=12345
```

В этом примере:

- `/login` - путь на сервере, куда отправляется запрос.
- `username=alice&password=12345` - данные, отправляемые на сервер. Они не видны в URL, а содержатся в теле запроса.

Структура формы входа:

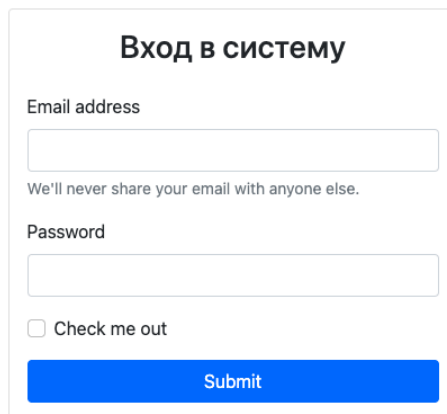
Форма входа будет включать следующие элементы:

- **Поле для ввода имени пользователя:** Стандартное текстовое поле для ввода имени пользователя.
- **Поле для ввода пароля:** Поле с маскировкой для ввода пароля.
- **Кнопка отправки формы:** Кнопка, которая активирует процесс проверки учетных данных и входа в систему.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Вход в систему</title>
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container d-flex align-items-center justify-content-center"
style="min-height: 100vh;">
    <div class="card w-100" style="max-width: 400px;">
        <div class="card-body">
            <h3 class="card-title text-center mb-4">Вход в
систему</h3>
            <form method="post">
                <div class="mb-3">
                    <label for="username" class="form-
label">Имя пользователя</label>
                    <input type="text" class="form-control"
id="username" name="username"
                    aria-describedby="usernameHelp">
                    <div id="usernameHelp" class="form-
text">Ваше имя пользователя останется конфиденциальным.</div>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-
label">Пароль</label>
                    <input type="password" class="form-
control" id="password" name="password">
                </div>
                <div class="mb-3 form-check">
                    <input type="checkbox" class="form-check-
input" id="rememberMe">
                    <label class="form-check-label"
for="rememberMe">Запомнить меня</label>
                </div>
                <button type="submit" class="btn btn-primary w-
100">Войти</button>
            </form>
        </div>
    </div> <!-- Bootstrap JS -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
></script>
</body>
</html>

```



Вход в систему

Email address

We'll never share your email with anyone else.

Password

☐ Check me out

Submit

Flask - login form to admin panel

Шаг 3: создание макета для админ панели

В этом шаге мы создадим функциональный и интуитивно понятный интерфейс административной панели. Основная цель — обеспечить удобное управление контентом и настройками сайта для администраторов. Вот ключевые элементы дизайна:

1. **Шапка (Header):** В верхней части страницы расположится шапка, отображающая имя текущего пользователя, что дает понимание, кто сейчас авторизован. Также в шапке будет кнопка "Выход", позволяющая безопасно завершить сеанс пользователя.
2. **Левое Навигационное Меню:** Слева от основного контента будет расположено навигационное меню, из которого администраторы смогут легко переходить к различным разделам административной панели, включая "Редактор контента". Это меню будет представлять собой список ссылок или кнопок, каждая из которых ведет на соответствующую страницу управления.

3. **Основная Область Контента:** Здесь будет отображаться интерфейс для управления контентом или другие административные инструменты в зависимости от выбранного раздела в навигационном меню.

```

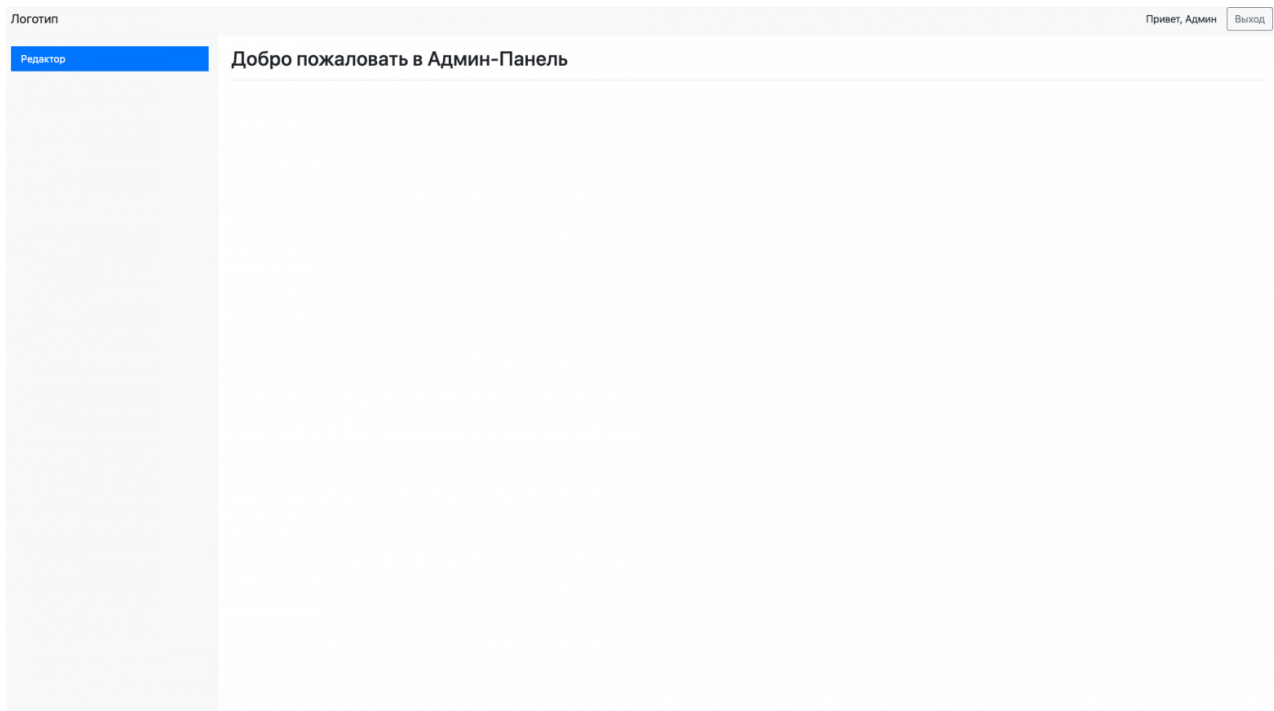
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Админ-Панель</title>
  <!-- Bootstrap CSS -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <style>
    /* Стили для бокового меню */
    #sidebarMenu {
      min-height: 100vh; /* Высота на всю высоту вьюпорта */
      box-shadow: 0 2px 5px rgba(0,0,0,0.1); /* Тень для бокового меню */
      z-index: 1000; /* Устанавливаем z-index, чтобы меню было поверх
основного контента */
    }
    .sidebar .nav-link {
      border-radius: 0; /* Убираем закругление углов */
      color: #333; /* Цвет текста */
    }
    .sidebar .nav-link:hover {
      background: #f8f9fa; /* Фон при наведении */
    }
    .sidebar .nav-link.active {
      background-color: #007bff; /* Фон для активной ссылки */
      color: white; /* Цвет текста для активной ссылки */
    }
  </style>
</head>
<body> <!-- Header -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Логотип</a>
    <div class="d-flex align-items-center">
      <span class="me-3">Привет, Админ</span>
      <button class="btn btn-outline-secondary" type="button">Выход</button>
    </div>
  </div>
</nav> <!-- Admin Panel Layout -->
<div class="container-fluid">
  <div class="row">
    <!-- Sidebar Menu -->
    <nav id="sidebarMenu" class="col-md-3 col-lg-2 d-md-block bg-light sidebar
collapse">
      <div class="position-sticky pt-3">
        <ul class="nav flex-column">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="#">
              <span data-feather="home"></span>
              Редактор
            </a>
          </li>
          <!-- More menu items -->
        </ul>
      </div>
    </nav>
    <!-- Main Content -->
    <main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">

```

```

        <div class="d-flex justify-content-between flex-wrap flex-md-nowrap
align-items-center pt-3 pb-2 mb-3 border-bottom">
            <h1 class="h2">Добро пожаловать в Админ-Панель</h1>
        </div>
        <!-- Content goes here -->
    </main>
</div>
</div> <!-- Bootstrap JS -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
></script>
</body>
</html>

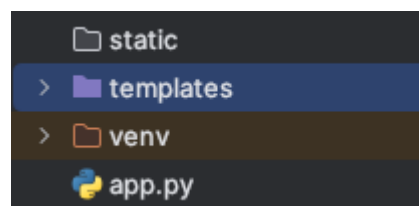
```



Flask - admin panel

Шаг 4: Приступаем к Разработке Приложения Flask

После того как мы подготовили макеты для административной панели, настало время перейти к разработке приложения Flask. Мы будем опираться на структуру Flask проекта, которую мы создали в статье "[Flask для начинающих: знакомство и настройка](#)". Перед началом разработки убедитесь, что ваша среда разработки настроена соответствующим образом.



- **static:** Эта папка предназначена для хранения статических файлов, таких как CSS и JavaScript. Здесь мы размещаем все вспомогательные ресурсы, которые делают наш веб-интерфейс привлекательным и функциональным, включая стиливые таблицы для определения визуальной составляющей и скрипты для интерактивных элементов интерфейса.
- **templates:** В этой директории мы создаем HTML-шаблоны нашего приложения. Это макеты, которые были разработаны на предыдущих шагах (1-3), и они определяют структуру и макет страниц нашего веб-приложения. Flask автоматически связывается с этой папкой для поиска и рендеринга HTML-шаблонов.
- **venv:** Это виртуальное окружение Python, которое позволяет нам управлять зависимостями проекта изолированно от других проектов. Это означает, что можно установить и использовать определенные версии библиотек (например, Python версии 3, SQLite и другие) без риска конфликтов с библиотеками, установленными на глобальном уровне системы.

- **app.py:** Этот файл является основным файлом веб-приложения, разработанного с использованием фреймворка Flask. В **app.py** происходит инициализация и конфигурация Flask-приложения, а также определение маршрутов и обработчиков запросов. Файл **app.py** служит входной точкой для запуска приложения и часто содержит основную логику взаимодействия с пользователем.

Шаги для создания venv:

1. Откройте Терминал или Командную Строку:

Находясь в корневой директории вашего проекта, откройте терминал (на Linux или macOS) или командную строку/PowerShell (на Windows).

2. Создание Виртуального Окружения:

Выполните следующую команду для создания виртуального окружения в папке **venv** внутри вашего проекта:

```
python -m venv venv
```

Эта команда создаст новую директорию **venv**, содержащую скрипты виртуального окружения и другие файлы.

3. Активация Виртуального Окружения:

- На Windows:

```
.\venv\Scripts\activate
```

- На Linux или macOS:

```
source venv/bin/activate
```

После активации виртуального окружения ваш командный интерпретатор изменит свой вывод, чтобы показать имя активированного окружения.

4. Установка Зависимостей:

Теперь, когда ваше виртуальное окружение активировано, вы можете установить необходимые зависимости, такие как Flask:

```
pip install flask
```

Все установки будут ограничены вашим виртуальным окружением.

5. Работа в Виртуальном Окружении:

Продолжайте разработку, используя это виртуальное окружение. Все Python-скрипты будут исполняться с использованием изолированных зависимостей.

6. Деактивация Виртуального Окружения:

Когда вы закончите работу, вы можете деактивировать виртуальное окружение, выполнив:

```
deactivate
```

Это вернет вас к системному Python и его пакетам.

Преимущества Использования venv:

- Изолирует зависимости проекта от глобальной установки Python.
- Позволяет управлять различными версиями библиотек для разных проектов.
- Упрощает развертывание проекта и его настройку на других машинах.

Настройка виртуального окружения:

1. Убедитесь, что на вашем компьютере установлен Python.
2. Откройте терминал и перейдите в корневую папку вашего проекта.
3. Выполните команду для создания виртуального окружения:

```
python -m venv venv
```

4. Активируйте виртуальное окружение:

- На Windows: `venv\Scripts\activate`
- На macOS/Linux: `source venv/bin/activate`

5. После активации установите необходимые библиотеки, используя `pip`:

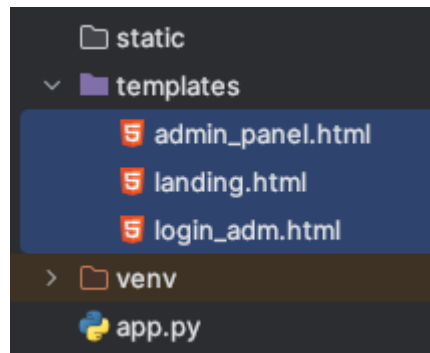
```
pip install flask
```

Следуя этим шагам, вы создадите надежную основу для разработки вашего Flask-приложения, с четким разделением ресурсов и изолированным управлением зависимостями.

создание App.py

Перед тем как приступить к написанию кода, убедимся, что структура вашего проекта Flask выглядит следующим образом:

- `/` - корневой маршрут, который будет отображать вашу лендинг-страницу.
- `/adm_login` - маршрут для страницы входа в админ-панель.
- `/admin_panel` - маршрут, который ведет непосредственно на страницу админ-панели.



Flask - структура проекта с макетами html

Теперь приступим к созданию файла `app.py` для нашего проекта. В этом файле мы подключим наши HTML-шаблоны и определим маршруты для их отображения. Вот примерный код файла `app.py` с комментариями, объясняющими каждый шаг:

```
from flask import Flask, render_template, redirect, url_for, request
# Flask - библиотека для запуска нашего приложения Flask - app
# render_template - нужен для то чтобы ваша страница html отобразилась корреткно
# redirect - нам понадобится для обработки запросы формы где мы перенаприм
пользователя на страницу админ панели
# url_for - вспомогательна библиотека для того чтобы сделать правильный переход по
ссылке в нашем случеш мы будем ссылаться на adm_panel
# request - обработчик запросов GET/POST и дргих app = Flask(__name__) # наша
корневая страиницу лендинда
@app.route('/')
def home():
    # Загрузка и отображение главной страницы (landing page)
    return render_template('landing.html') # страница формы логина в админ панель
@app.route('/adm_login', methods=['GET', 'POST'])
def admin_login():
    if request.method == 'POST':
        # Здесь должна быть логика аутентификации
        # Если аутентификация прошла успешно, перенаправляем на /admin_panel
        return redirect(url_for('admin_panel'))
    # Если GET запрос, показываем форму входа
    return render_template('login_adm.html') # страница админ панели
@app.route('/admin_panel')
def admin_panel():
    # Загрузка и отображение админ-панели
    return render_template('admin_panel.html') if __name__ == '__main__':
app.run(debug=True)
```

После написания кода запустим наше приложение и проверим, что все страницы отображаются корректно.

```
python app.py
```

После запуска вы должны увидеть следующее:

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 107-615-069
```

Проверяем наши страницы, что они отобразились:

- просто заходим на главную страницу <http://127.0.0.1:5000/> - landing page
- http://127.0.0.1:5000/adm_login - страница формы логина
- http://127.0.0.1:5000/admin_panel - страница админ панели

если ваши страницы отобразились то значит вы настроили всё корректно и можно переходить на следующий шаг 5

► пример Troubleshooting

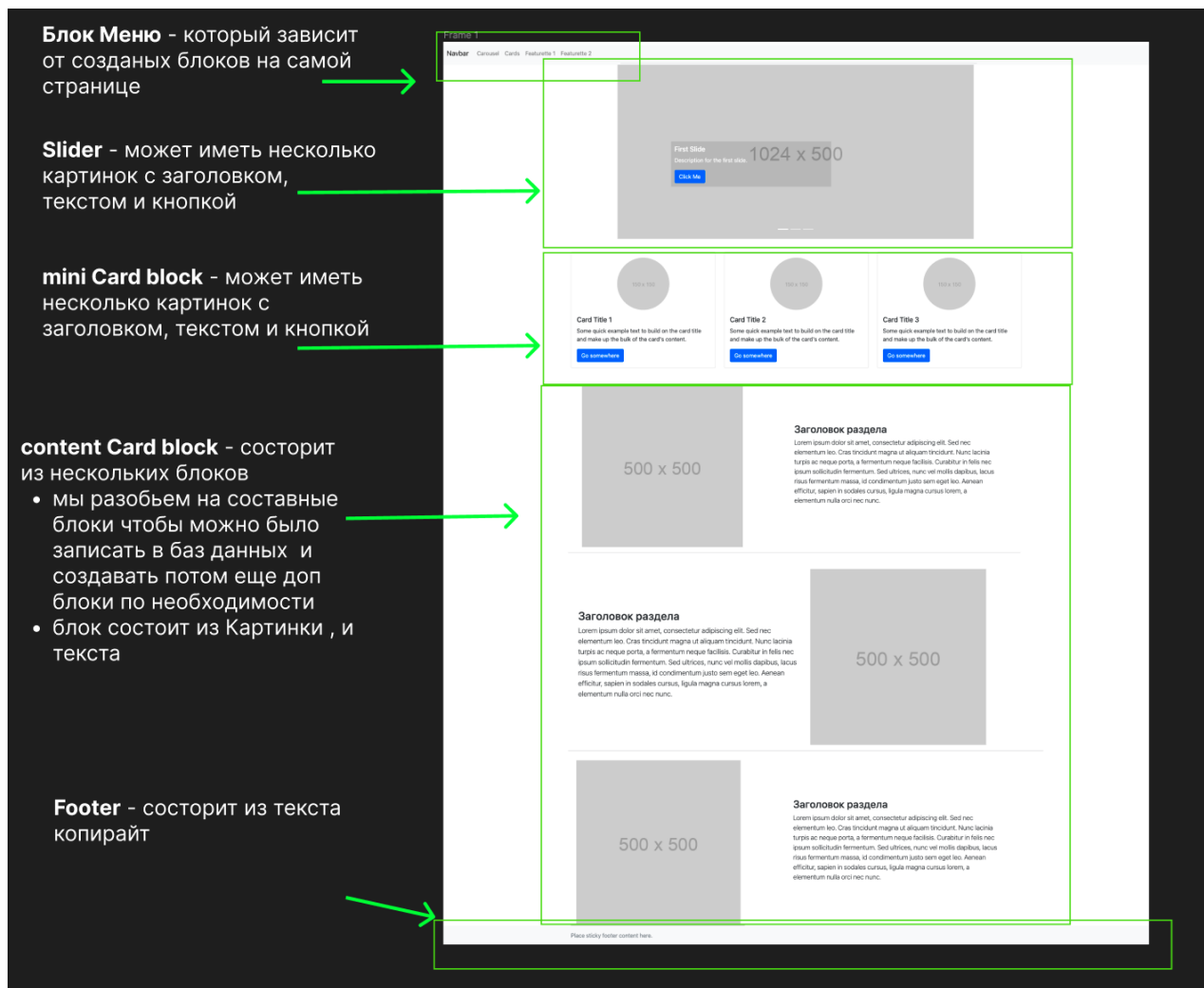
Шаг 5: Создаем Базу Данных:

Для упрощения настройки и минимизации необходимости дополнительной конфигурации, в нашем проекте мы будем использовать SQLite — легковесную, эффективную и самодостаточную базу данных, идеально подходящую для разработки и тестирования. Это позволит легко развернуть базу данных на любом локальном компьютере без сложностей, связанных с настройкой более крупных систем управления базами данных, таких как MySQL или PostgreSQL.

Структура базы данных:

Разбиение сайта на блоки:

Мы организуем структуру базы данных, разбивая наш сайт на отдельные блоки. Для простоты, сосредоточимся на разделении только главной страницы (landing page).



Разбиваем по блока структуру сайта

Таким образом: мы можем создать простую структуру базы данных

каждый блок должен иметь свой id + контент

пример Таблицы которой мы можем создать для базы данных:

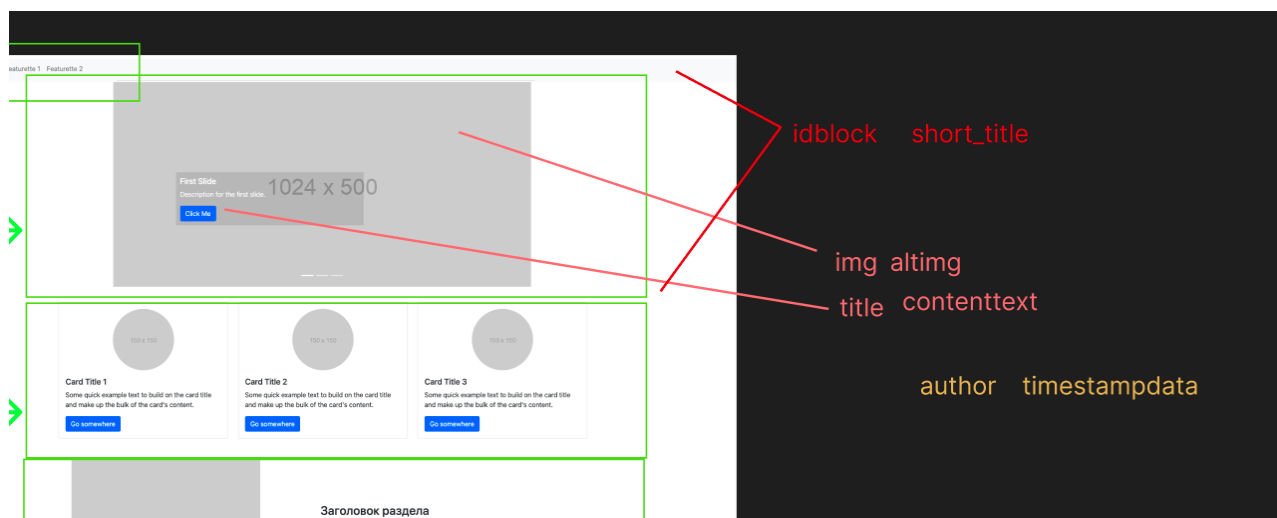
Таблицы базы данных:

Таблица Content:

- **id**: int, автоинкрементный - уникальный идентификатор каждой записи.
- **idblock**: text - идентификатор блока, используемый для навигации и связывания элементов внутри блока.
- **short_title**: text - короткое название блока для использования в меню или заголовках.
- **img**: text - путь к изображению.
- **altimg**: text - альтернативный текст изображения для доступности и SEO.

- **title**: text - заголовок элемента.
- **contenttext**: text - основной текст блока.
- **author**: text - информация об авторе изменений.
- **timestampdata**: datetime - время последнего изменения блока.

Примечание: **idblock** и **short_title** используются для группировки элементов в один логический блок (например, слайдер или карточку).



Описание структуры

idblock и **short_title** - это целый блок в котором может быть несколько блоков с картинкой и текстом

то есть для каждого слайда, где мы будем прописывать **img**, **altimg**, **title**, **contenttext** мы должны будем повторять название **idblock** и **short_title** для возможности их группировки в **slider block** или **mini card block** и тд

author и **timestampdata** - будем генерировать на бэкенде

для определения автора / пользователя и авторизации в админ часть нам нужна еще одна таблица

Таблица **Users**:

- **id**: int, автоинкрементный - уникальный идентификатор пользователя.
- **username**: text - имя пользователя, используемое для входа в систему.
- **password**: text - хешированный пароль пользователя (в данной статье мы не будем хэшировать пароль, чтобы упростить проект).

Сценарий развертывания базы данных:

Создадим скрипт на Python, который автоматически создаст эти таблицы в SQLite. Вот примерный скрипт для создания этих таблиц:

скрип давайте положим в наш проект создав новый файл **createdb.py**

```
import sqlite3 # Создание или подключение к базе данных
conn = sqlite3.connect('database.db') # Создание курсора
c = conn.cursor() # Создание таблицы Content
c.execute('''CREATE TABLE IF NOT EXISTS content (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            idblock TEXT,
            short_title TEXT,
            img TEXT,
            altimg TEXT,
            title TEXT,
            contenttext TEXT,
            author TEXT,
            timestampdata DATETIME)''') # Создание таблицы Users
c.execute('''CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT,
            password TEXT)''') # Закрытие соединения с базой данных
conn.close()
```

далее запускаем скрипт через консоль

```
python creatdb.py
```

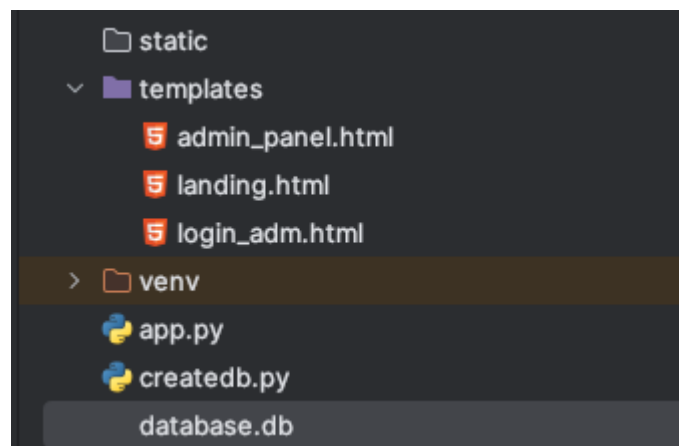
если у вас появились ошибки, то проблема может быть связана с не установленным sqlite

просто пропишите команду

```
pip install sqlite3
```

и повторить запуск скрипта

как результат вы должны будете увидеть созданную базу данных



Создаем пользователя В базу данных

Для создания пользователя в базу данных через скрипт `createuser.py`, вам необходимо выполнить следующие шаги. Этот скрипт позволит добавить пользователя в таблицу `users` вашей базы данных SQLite, что особенно полезно в случаях, когда у вас нет интерфейса для регистрации пользователей и вы хотите создать учетные записи администраторов заранее.

Для хеширования пароля мы будем использовать библиотеку `hashlib` это просто быстрый метод, чтобы защитить пароль. В этом примере используется простое хеширование SHA-256 для пароля. В реальных приложениях рекомендуется использовать более сложные методы хеширования и соль для обеспечения большей безопасности.

напишем код `createuser.py`, который будет заносить нового пользователя в базу данных

```
import sqlite3
import hashlib

def create_user(username, password):
    # Хеширование пароля
    hashed_password = hashlib.sha256(password.encode('utf-8')).hexdigest()

    # Подключение к нашей базе данных
    conn = sqlite3.connect('database.db')
    c = conn.cursor()
    # Добавление нового пользователя
    c.execute('INSERT INTO users (username, password) VALUES (?, ?)', (username, hashed_password))
    # Сохранение изменений и закрытие соединения с базой данных
    conn.commit()
    conn.close()

# Замените 'admin' и 'your_password' на желаемые имя пользователя и пароль
create_user('admin', 'your_password')
```

я сделал в качестве пример `admin admin`

и если посмотреть базу то пароль у нас будет зашифрован

Table:	users	Page:	0	Jump	<<	<	1-1	>	>>	Refresh
	id	username	password							
	1	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918							

Шаг 6: Подключаем Базу данных в наше приложение `app.py` и настраиваем авторизацию в админ часть

Нам понадобится внести следующие изменения в код

1. Импортировать дополнительные модули для работы с базой данных и хеширования паролей.

2. Создать функцию для подключения к базе данных SQLite.

3. Добавить логику обработки запросов входа/выхода в админ-панель

обновленный код с комметариями:

В данном коде мы добавили Sqlite и hashlib библиотеки
и обрабатываем форму логина

```
from flask import Flask, render_template, redirect, url_for, request, session
import sqlite3 # подключаем Sqlite в наш проект
import hashlib # библиотека для хеширования app = Flask(__name__)
app.secret_key = 'your_secret_key' # подставьте свой секретный ключ
# секретный ключ для хеширования данных сессии при авторизации # Устанавливаем
соединение с Базой Данных
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn @app.route('/')
def home():
    return render_template('landing.html') @app.route('/adm_login', methods=
['GET', 'POST'])
def admin_login():
    error = None # обнуляем переменную ошибок
    if request.method == 'POST':
        username = request.form['username'] # обрабатываем запрос с нашей формы
который имеет атрибут name="username"
        password = request.form['password'] # обрабатываем запрос с нашей формы
который имеет атрибут name="password"
        hashed_password = hashlib.sha256(password.encode('utf-8')).hexdigest() #
шифруем пароль в sha-256 # устанавливаем соединение с БД
        conn = get_db_connection()
        # создаем запрос для поиска пользователя по username,
        # если такой пользователь существует, то получаем все данные id, password
        user = conn.execute('SELECT * FROM users WHERE username = ?',
(username,)).fetchone()
        # закрываем подключение БД
        conn.close() # теперь проверяем если данные сходятся
формы с данными БД
        if user and user['password'] == hashed_password:
            # в случае успеха создаем сессию в которую записываем id пользователя
            session['user_id'] = user['id']
            # и делаем переадресацию пользователя на новую страницу -> в нашу
адимнку
            return redirect(url_for('admin_panel')) else:
            error = 'Неправильное имя пользователя или пароль' return
render_template('login_adm.html', error=error) @app.route('/admin_panel')
def admin_panel():
    # делаем доп проверку если сессия авторизации была создана
    if 'user_id' not in session:
        return redirect(url_for('admin_login'))
    return render_template('admin_panel.html') if __name__ == '__main__':
app.run(debug=True)
```

вопросы по обновлениям:

что такое **secret_key**

► Hidden text

зачем нам session

► Hidden text

кому интересно посмотреть, что находится в сессии:

► Hidden text

теперь после того как мы залогинились мы должны разлогиниться при нажатие на выход в самой админ панели. Добавляем еще изменения в код: сделаем дополнительную функцию очистки сессии для выхода из админки

```
@app.route('/logout')
def logout():
    # Удаление данных пользователя из сессии
    session.clear()
    # Перенаправление на главную страницу или страницу входа
    return redirect(url_for('home'))
```

► обновленный app.py

и обновляем наш html у страницы админ панели
нам нужно добавить:

```
<a href="{{ url_for('logout') }}" class="btn btn-outline-secondary">Выход</a>
```

► обновленный admin_panel.html

Готово! Теперь у нас есть Лендинг, страница логина и страницы админки с
возможностью sign in и sign out

Шаг 7: Добавляем возможность редактирования блоков через админ панель

Давайте разделим этот шаг на две основные части:

- 7.1 - редактирование контента через админку и обновление БД
- 7.2 - автоматическое обновление header и данных на фронте landing page

7.1 - Редактирование контента через админ-панель и обновление базы данных

Начнем с того, что необходимо отобразить информацию на админ-панели, для чего создадим шаблон `editland.html` и поместим его в папку `templates`. Этот файл будет отвечать за отображение информации для редактирования на странице `admin_panel.html`.

Для интеграции `editland.html` в `admin_panel.html` мы используем следующую вставку кода:

Вывод информации сделаем по группировки блоков и возьмем пример html разметки

And with vertical pills.

Home

Profile

Messages

Settings

This is some placeholder content the Settings tab's associated content. Clicking another tab will toggle the visibility of this one for the next. The tab JavaScript swaps classes to control the content visibility and styling. You can use it with tabs, pills, and any other `.nav`-powered navigation.

```
<div class="d-flex align-items-start">
  <div class="nav flex-column nav-pills me-3" id="v-pills-tab" role="tablist" aria-orienta
    <button class="nav-link active" id="v-pills-home-tab" data-bs-toggle="pill" data-bs-ta
    <button class="nav-link" id="v-pills-profile-tab" data-bs-toggle="pill" data-bs-target
    <button class="nav-link" id="v-pills-messages-tab" data-bs-toggle="pill" data-bs-targe
    <button class="nav-link" id="v-pills-settings-tab" data-bs-toggle="pill" data-bs-targe
  </div>
  <div class="tab-content" id="v-pills-tabContent">
    <div class="tab-pane fade show active" id="v-pills-home" role="tabpanel" aria-labelled
    <div class="tab-pane fade" id="v-pills-profile" role="tabpanel" aria-labelledby="v-pil
    <div class="tab-pane fade" id="v-pills-messages" role="tabpanel" aria-labelledby="v-pi
    <div class="tab-pane fade" id="v-pills-settings" role="tabpanel" aria-labelledby="v-pi
  </div>
</div>
```

Теперь разберемся с выводом информации. Мы сгруппируем блоки в соответствии с их структурой на лендинг-странице. Пример HTML-разметки для этих блоков:

- **Слайдер/Карусель:** `id = carouselExampleIndicators`
- **Мини-карточки:** `id = cards`
- **Разделы Featurette:** `id = featurette1, featurette2`, и так далее
- **Футер:** `id = Footerblock` (добавим вручную, так как отсутствует на лендинг-странице)

Для начала наполним базу данных примерами данных для этих блоков. Вы можете вручную добавить эти данные в БД.

id	idblock	short_title	img	altimg	title	contenttext	author	timestampdata
1	carouselExampleIndicators	Slider	https://via.placeholder.com/1024x500	image 1	Title Test 1	some text for slider 1	<null>	<null>
2	carouselExampleIndicators	Slider	https://via.placeholder.com/1024x500	image 2	Title Test 2	some text for slider 2	<null>	<null>
3	cards	miniCards	https://via.placeholder.com/150	mini img 2	mini card 2	text for mini card 2	<null>	<null>
4	cards	miniCards	https://via.placeholder.com/150	mini img 1	mini card 1	text for mini card 1	<null>	<null>

В app.py добавляем функционал для сбора данных из БД и передачи их на фронт

Теперь обновим код маршрута `/admin_login` для отображения актуальных данных из БД:

```
@app.route('/admin_panel')
def admin_panel():
    if 'user_id' not in session:
        return redirect(url_for('admin_login'))    conn = get_db_connection()
    blocks = conn.execute('SELECT * FROM content').fetchall() # Получаем все
записи из таблицы content
    conn.close()    # Преобразование данных из БД в список словарей
    blocks_list = [dict(ix) for ix in blocks]
    # print(blocks_list) [{строка 1 из бд},{строка 2 из бд},{строка 3 из бд},
строка 4 из бд]    # Теперь нужно сделать группировку списка в один словарь json
    # Группировка данных в словарь JSON
    json_data = {}
    for raw in blocks_list:
        # Создание новой записи, если ключ еще не существует
        if raw['idblock'] not in json_data:
            json_data[raw['idblock']] = []    # Добавление данных в
существующий ключ
        json_data[raw['idblock']].append({
            'id': raw['id'],
            'short_title': raw['short_title'],
            'img': raw['img'],
            'altimg': raw['altimg'],
            'title': raw['title'],
            'contenttext': raw['contenttext'],
            'author': raw['author'],
            'timestampdata': raw['timestampdata']
        })    # print(json_data)
    # передаем на json на фронт - далее нужно смотреть admin_panel.html и
обрабатывать там
    return render_template('admin_panel.html', json_data=json_data)
```

► output json_data:

теперь мы должны должны принять данные на фронте на самом html и создать функцию формирования данных на фронте

- Для правильного отображения информации из JSON на фронте, нужно учитывать структуру данных и соответствующим образом обрабатывать их в шаблоне. В данном случае, `json_data` представляет собой словарь, где ключи (`j_key`) являются идентификаторами блоков (например, 'carouselExampleIndicators', 'cards'), а значения (`j_value`) - списками словарей с данными для каждого блока.
- редактируем код **editland** - я сделал небольшой пример опираясь на пример выше из bootstrap

в данной разметки мы включаем код, который должен быть в между
{% тут код %}
таким образом, мы мы подставляем код питона в html для обработки
и вывода данных

```
<main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
    <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-
items-center pt-3 pb-2 mb-3 border-bottom">
        <h1 class="h2">Добро пожаловать в Админ-Панель</h1>
    </div> <!-- Content goes here -->
    <div class="d-flex align-items-start">
        <div class="nav flex-column nav-pills me-3" id="v-pills-tab"
role="tablist" aria-orientation="vertical">
            {% for j_key, j_value in json_data.items() %}
                <button class="nav-link" id="v-pills-{{ j_key }}-
tab" data-bs-toggle="pill"
                                data-bs-target="#v-pills-{{ j_key }}"
type="button" role="tab"
                                aria-controls="v-pills-{{ j_key }}" aria-
selected="false">
                                    {{ j_value[0].short_title
                                }}
                </button>
            {% endfor %}
        </div>
        <div class="tab-content" id="v-pills-tabContent">
            {% for j_key, j_value in json_data.items() %}
                <div class="tab-pane fade" id="v-pills-{{ j_key
}}}" role="tabpanel"
                                aria-labelledby="v-pills-{{ j_key }}-tab">
                    {% for item in j_value %}
                        <h3>{{ item.short_title }}</h3>
                        
                                <p>{{ item.title }}</p>
                                <p>{{ item.contenttext }}</p>
                    {% endfor %}
                </div>
            {% endfor %}
        </div>
    </div> </main>
```

и чтобы подтянуть **editland** в **admin_panel** делаем - под секцией *Main Content* добавляем оператор **IF** и проверяем если мы получили данные с бэка в случае успеха подключаем сегмент кода из **editland**, а если нет то показываем информацию что что-то произошло и не получилось получить данные или их нету:

```
<!-- Main Content -->
    {% if json_data %}
        {% include 'editland.html' %}
    {% else %}
        <main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
            <div class="d-flex justify-content-between flex-
wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-bottom">
                <h1 class="h2">Добро пожаловать в Админ-
Панель</h1>
            </div>
            <!-- Content goes
here -->
            <div class="d-flex align-items-start">
                <p>Данные пока отсутствуют или их не
получилось получить из БД</p>
            </div>
        </main>
    {% endif %}
```

► что такое Include?

Итог наш код **admin_panel.html** и **editland.html** показывает информацию из БД

Реализация возможности редактирования и добавления новых блоков

Чтобы добавить функцию редактирования информации и создания новых блоков, выполним следующие шаги:

1. Модификация **editland.html**:

- **Форма для отправки данных:** В **editland.html** добавим форму, которая позволит отправлять отредактированную информацию. Эта форма будет содержать все необходимые поля с текущими данными для удобства редактирования.
- **создание формы** - мы меняем отображения информации на код формы

```

<main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
    <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-
items-center pt-3 pb-2 mb-3 border-bottom">
        <h1 class="h2">Добро пожаловать в Админ-Панель</h1>
    </div> <!-- Content goes here -->
    <div class="d-flex align-items-start">
        <div class="nav flex-column nav-pills me-3" id="v-pills-tab"
role="tablist" aria-orientation="vertical">
            {% for j_key, j_value in json_data.items() %}
                <button class="nav-link" id="v-pills-{{ j_key }}-
tab" data-bs-toggle="pill"
                                data-bs-target="#v-pills-{{ j_key }}"
type="button" role="tab"
                                aria-controls="v-pills-{{ j_key }}" aria-
selected="false">
                                    {{ j_value[0].short_title
}}
                                </button>
            {% endfor %}
        </div> <!-- ОБНОВЛЕННЫЙ КОД - для отображения формы -->
        <div class="tab-content" id="v-pills-tabContent">
            {% for j_key, j_value in json_data.items() %}
                <div class="tab-pane fade" id="v-pills-{{ j_key
}}>
                    role="tabpanel"
                                aria-labelledby="v-pills-{{ j_key }}-tab"
style="position: absolute; left: 30%;> <!-- добавляем доп стиль чтобы данные
отображались правильно в столбик относительно каждого блока -->
                                {% for item in j_value %}
                                    <form action="update_content"
method="post"> <!-- Отправляем 1 сегмент блока через POST на endpoint
update_content -->
                                    <h3><input type="text" name="short_title" value="{{ item.short_title }}"></h3> <!--
- название блока в меню header -->
                                     <!-- показываем изображение которое сейчас доступно -->
                                    <input type="file"
name="img"> <!-- возможность загружать новое изображение -->
                                    <p><input type="text"
name="title" value="{{ item.title }}"></p> <!-- заголовок контента -->
                                    <p><textarea
name="contenttext">{{ item.contenttext }}</textarea></p> <!-- текс контента -->
                                    <input type="hidden"
name="id" value="{{ item.id }}"> <!-- id контента - поле не показывается на форме
-->
                                    <button type="submit"
class="btn btn-primary">Сохранить изменения</button>
                                </form>
                                {% endfor %}
                            </div>
                        {% endfor %}
                    </div>
                </div> </main>

```

код выше отправляет один сегмент блока на изменения через app.py

Создание маршрута /update_content:

Обработка данных формы: В `app.py` создадим маршрут `/update_content`, который будет принимать и обрабатывать данные, отправленные из формы на фронте.

Импортированные библиотеки:

- `import os`: Используется для работы с файловой системой и создания безопасных путей к файлам и директориям.
- `from werkzeug.utils import secure_filename`: Обеспечивает безопасность имён файлов, загруженных пользователями, предотвращая атаки через манипуляции с файловой системой.

```
# в самом верху добавить новые библиотеки
import os # отвечает за
from werkzeug.utils import secure_filename # отвечает за # Путь для сохранения
изображений
path_to_save_images = os.path.join(app.root_path, 'static',
'imgs') ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'} def
allowed_file(filename):
    return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS @app.route('/update_content', methods=['POST'])
def update_content():    content_id = request.form['id']
    short_title = request.form['short_title']
    title = request.form['title']
    altimg = request.form['altimg']
    contenttext = request.form['contenttext']    # Обработка загруженного файла
    file = request.files['img']    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        save_path = os.path.join(path_to_save_images, filename)
        imgpath = "/static/imgs/"+filename
        file.save(save_path)
        # Обновите путь изображения в вашей базе данных    # Обновление данных в
базе
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    if file:
        cursor.execute('UPDATE content SET short_title=?, img=?, altimg=?,
title=?, contenttext=? WHERE id=?',
                        (short_title, imgpath, altimg, title, contenttext, content_id))
    else:
        cursor.execute('UPDATE content SET short_title=?, altimg=?, title=?,
contenttext=? WHERE id=?',
                        (short_title, altimg, title, contenttext, content_id))
    conn.commit()
    conn.close()    return redirect(url_for('admin_panel'))
```

в нашем коде мы получаем данные по методу POST и записываем в переменные после этого мы проверяем если у нас был файл картинки

Обработка загрузки изображений:

Проверка и сохранение изображений: Если форма содержит изображение, обновляем информацию в БД с новым путем к файлу. В противном случае сохраняем данные без изменения изображения.

Важное замечание:

Создание папки `/imgs`: Убедитесь, что папка `/static/imgs` существует для хранения загруженных изображений. Эту папку необходимо создать вручную.

Расширение функционала:

Добавление даты, времени и автора: Вы можете расширить функционал, добавив обновление даты, времени и информации об авторе при каждом изменении. Это отличная возможность для практики!

► подсказка как получить обновления - автор и время

7.2 - автоматическое обновление header и данных на фронте landing page

мы уже знаем , получать данные из БД , чтобы показать что у нас есть

поэтому нам остается добавить только логику для endpoint корня сайта в app.py и добавить логику в landing.html

получаем данные из БД и отправляем на фронт - просто копируем часть кода из /admin_panel

```
conn = get_db_connection()
blocks = conn.execute('SELECT * FROM content').fetchall() # Получаем все
записи из таблицы content
conn.close()      # Преобразование данных из БД в список словарей
blocks_list = [dict(ix) for ix in blocks]
# print(blocks_list) [{строка 1 из бд},{строка 2 из бд},{строка 3 из бд},
строка 4 из бд]    # Теперь нужно сделать группировку списка в один словарь json
# Группировка данных в словарь JSON
json_data = {}
for raw in blocks_list:
    # Создание новой записи, если ключ еще не существует
    if raw['idblock'] not in json_data:
        json_data[raw['idblock']] = []          # Добавление данных в
существующий ключ
    json_data[raw['idblock']].append({
        'id': raw['id'],
        'short_title': raw['short_title'],
        'img': raw['img'],
        'altimg': raw['altimg'],
        'title': raw['title'],
        'contenttext': raw['contenttext'],
        'author': raw['author'],
        'timestampdata': raw['timestampdata']
    })
```

и добавляем его в наш корень '/'

```
@app.route('/')
def home():
    conn = get_db_connection()
    blocks = conn.execute('SELECT * FROM content').fetchall() # Получаем все
записи из таблицы content
    conn.close() # Преобразование данных из БД в список словарей
    blocks_list = [dict(ix) for ix in blocks]
    # print(blocks_list) [{строка 1 из бд},{строка 2 из бд},{строка 3 из бд},
строка 4 из бд] # Теперь нужно сделать группировку списка в один словарь json
    # Группировка данных в словарь JSON
    json_data = {}
    for raw in blocks_list:
        # Создание новой записи, если ключ еще не существует
        if raw['idblock'] not in json_data:
            json_data[raw['idblock']] = [] # Добавление данных в
существующий ключ
        json_data[raw['idblock']].append({
            'id': raw['id'],
            'short_title': raw['short_title'],
            'img': raw['img'],
            'alting': raw['alting'],
            'title': raw['title'],
            'contenttext': raw['contenttext'],
            'author': raw['author'],
            'timestampdata': raw['timestampdata']
        })
    # далее передаем json_data на фронт
    return render_template('landing.html', json_data=json_data)
```

и нам осталось обновить логику на фронте для landing.html

в каждом блоке сегмента у нас есть id div-а поэтому обновляем только внутреннюю часть каждого блока

начнем с простого - вывод header menu

из этого

```

<!-- Header -->
<header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Navbar</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link"
href="#carouselExampleIndicators">Carousel</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="#cards">Cards</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="#featurette1">Featurette 1</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link"
href="#featurette2">Featurette 2</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>

```

делаем обработку словаря items():

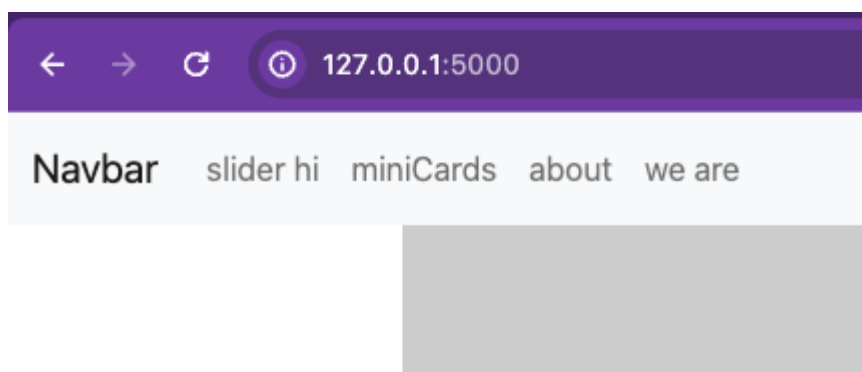
- в нашем json keys - это и есть href для наших ссылок их будем формировать {{ j_key }}
- а название это value первого элемента массива по значению short_title поэтому подставляем {{ j_value[0].short_title }}

```

<!-- Header -->
<header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Navbar</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
ОБНОВЛЕННАЯ ЧАСТЬ КОДА -->
                    {% for j_key, j_value in json_data.items()
%}
                        <li class="nav-item">
                            <a class="nav-link" href="#{{ j_key }}">{{
j_value[0].short_title }}</a>
                        </li>
                    {% endfor %}
                <!-- ОБНОВЛЕННАЯ ЧАСТЬ КОДА КОНЕЦ -->
            </ul>
        </div>
    </nav>
</header>

```

как результат мы должны увидеть сл



Приступаем к обработке slider

здесь мы будем искать значения из json где **key = carouselExampleIndicators**

В этом коде:

- Индикаторы карусели генерируются в цикле `{% for item in json_data['carouselExampleIndicators'] %}`. Атрибут `data-bs-slide-to` устанавливается в значение `loop.index0`, которое является индексом текущего элемента в цикле (начиная с 0). Активный класс добавляется только к первому элементу.

- Элементы карусели также генерируются в цикле с использованием тех же данных. Изображения, заголовки и текст берутся непосредственно из элементов `item` в `json_data['carouselExampleIndicators']`.
- Убедитесь, что `json_data['carouselExampleIndicators']` существует и содержит данные перед использованием этого кода в шаблоне.

```
<!-- Carousel With Indicators -->
<div id="carouselExampleIndicators" class="carousel slide" data-bs-
ride="carousel">
  <!-- Indicators -->
  <div class="carousel-indicators">
    {% for item in json_data['carouselExampleIndicators'] %}
      <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="{{ loop.index0 }}"
      class="{{ 'active' if loop.first else '' }}" aria-label="Slide
{{ loop.index }}"></button>
    {% endfor %}
  </div>
  <!-- Carousel items -->
  <div class="carousel-inner">
    {% for item in json_data['carouselExampleIndicators'] %}
      <div class="carousel-item {{ 'active' if loop.first else '' }}">
        
        <div class="carousel-caption d-none d-md-block">
          <h5>{{ item.title }}</h5>
          <p>{{ item.contenttext }}</p>
          <button class="btn btn-primary">Click Me</button>
        </div>
      </div>
    {% endfor %}
  </div>
  <!-- Controls -->
  <button class="carousel-control-prev" type="button" data-bs-
target="#carouselExampleIndicators" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-
target="#carouselExampleIndicators" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

Обработка блока Card

Используется цикл `{% for item in json_data['cards'] %}` для итерации по каждому элементу в списке `json_data['cards']`.

```

<!-- Cards in grid -->
<section id="cards" class="container mt-4">
  <div class="row">
    {% for item in json_data['cards'] %}
      <div class="col-md-4">
        <div class="card">
          
          <div class="card-body">
            <h5 class="card-title">{{ item.title }}</h5>
            <p class="card-text">{{ item.contenttext }}</p>
            <a href="#" class="btn btn-primary">Go somewhere</a>
          </div>
        </div>
      </div>
    {% endfor %}
  </div>
</section>

```

Обработка Cards с картинкой слева/справа

В этом коде:

- Цикл `{% for count in range(1, 10) %}` итерирует счетчик `count` от 1 до 9 (предполагая, что у вас максимум 9 featurette).
- Выражение `{% set feature_key = 'feature' ~ count %}` создает ключ для доступа к соответствующему featurette в `json_data`.
- Блок `{% if json_data.get(feature_key) %}` проверяет, существует ли ключ `feature_key` в `json_data`, и если да, выводит соответствующий раздел featurette.

```

<!-- Horizontal Card with image and text right -->
{% for count in range(1, 10) %} <!-- Предположим, что у нас не более 10
featurette -->
    {% set feature_key = 'featurette' ~ count %}
    {% if json_data.get(feature_key) %}
        <section id="featurette{{ count }}" class="container mt-4">
            <div class="row align-items-center">
                {% if count % 2 == 1 %}
                    <!-- Для нечетных featurette: изображение слева, текст справа
-->

                    <div class="col-md-6">
                        
                    </div>
                    <div class="col-md-6">
                        <!-- Текст -->
                        <h2 class="featurette-heading">{{ json_data[feature_key]
[0].title }}</h2>
                        <p class="lead">{{ json_data[feature_key][0].contenttext
}}</p>
                    </div>
                {% else %}
                    <!-- Для четных featurette: текст слева, изображение справа --
>

                    <div class="col-md-6">
                        <!-- Текст -->
                        <h2 class="featurette-heading">{{ json_data[feature_key]
[0].title }}</h2>
                        <p class="lead">{{ json_data[feature_key][0].contenttext
}}</p>
                    </div>
                    <div class="col-md-6">
                        
                    </div>
                {% endif %}
            </div>
        </section>
        <hr class="featurette-divider">
    {% endif %}
{% endfor %}

```

Обработка footer

в данном коде: проверяем если есть контент - в случае если нету, то ставим дефолтный текст


```

<!-- Footer -->
<footer class="footer mt-auto py-3 bg-light">
  <div id="Footerblock" class="container">
    {% for item in json_data['Footerblock'] %}
      {% if item.contenttext %}
        <span class="text-muted">{{ item.contenttext }}</span>
      {% else %}
        <span class="text-muted">Place sticky footer content here.
</span>
      {% endif %}
    {% endfor %}
  </div>
</footer>

```

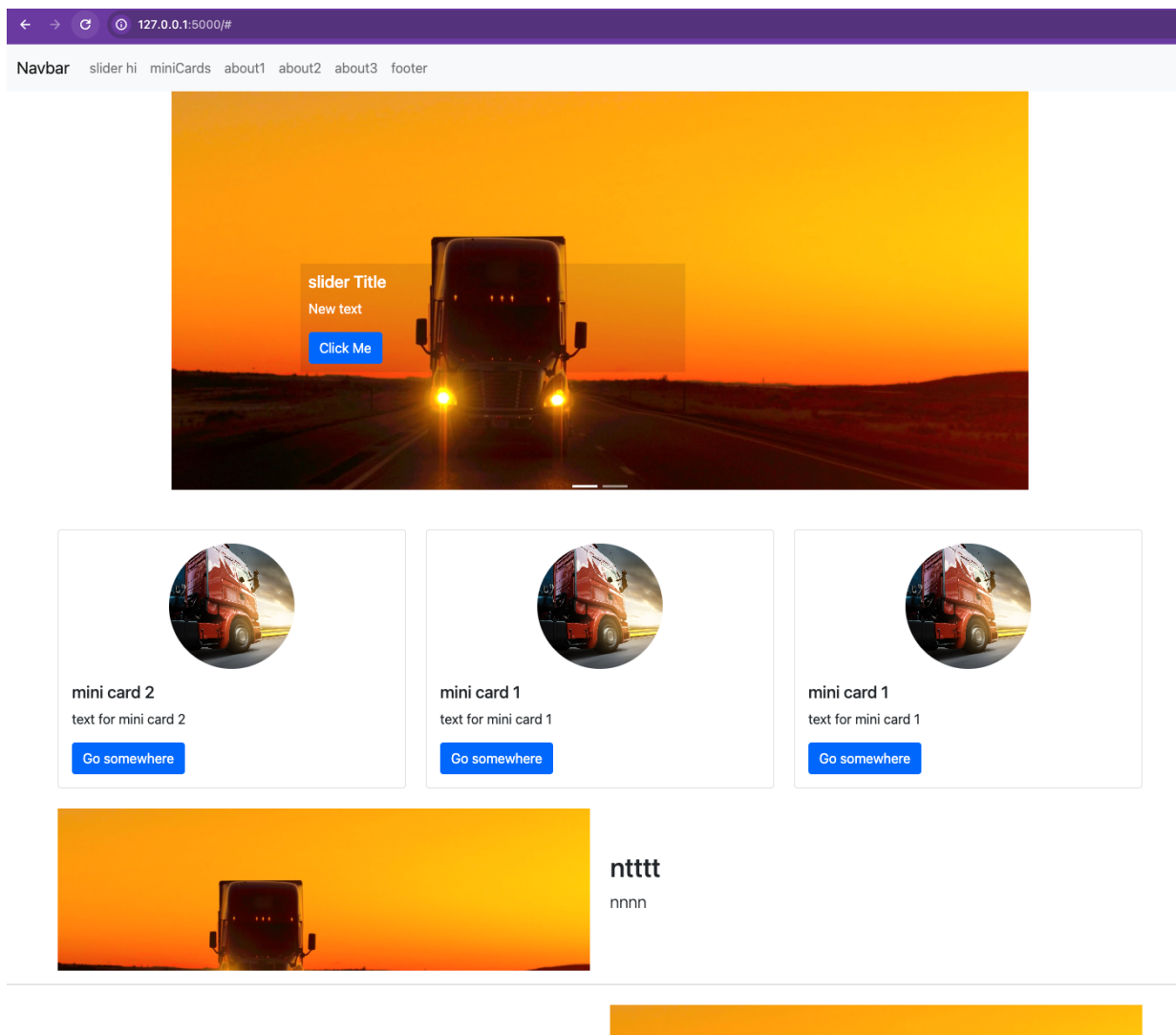
Итоги:

мы создали Веб страницу с мини админкой, где есть возможность редактировать блоки фронта напрямую из admin panel.

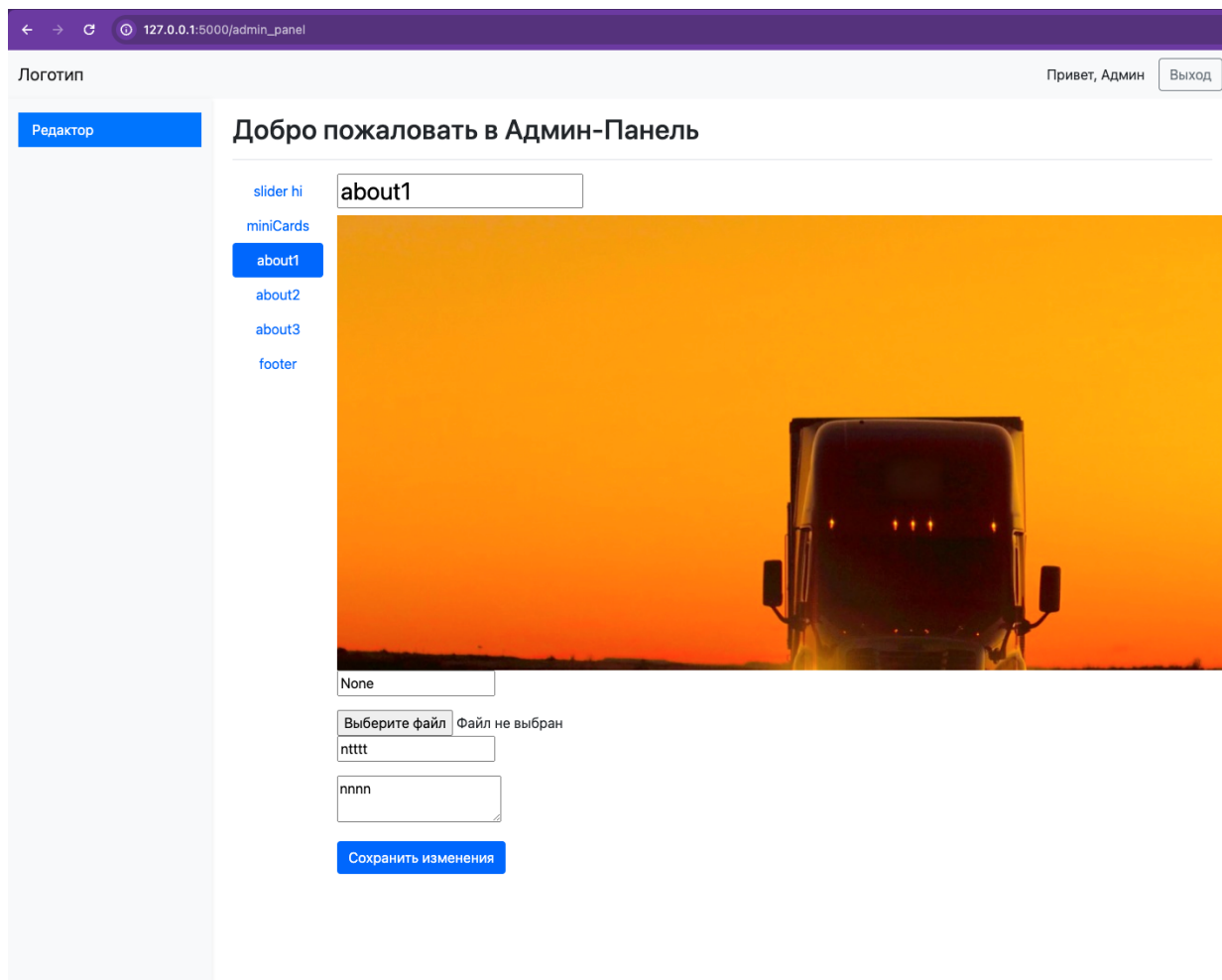
Дополнительно в тексте вы нашли задачку для доработки определения автора текста или кто изменял контент и когда - а также получили подсказку, как это сделать

знания, которые мы покрыли в данной статье:

- создание скелета сайта на основе bootstrap5
- разработка landing page и admin части
- работа с session на фронте и на бэке
- работа с БД sqlite
- шифрование пароля
- дешифровка информации сессии
- проверка получение файлов изображения в форму и обработка на бэке
- понимание передачи информации с backend на frontend



фронт сайта после редактирование в админке



редактирование контента в админ панели

P.S. надеюсь статья была для полезна и познавательна
и если вы Начинающий разработчик - то таким образом, вы уже можете начать
работать на freelance и создавать сайт где можно редактировать контент
Конечно вы можете доработать функционал и улучшить работу сайта, а также вы
сможете начать создавать, что-то новое свое)

Если у вас появились вопросы или есть предложения рад буду вас услышать в
комментариях

Финальная версия проекта:

Структура

```
/static                # эта папка для хранения JS или CSS данных или других
файлов
...../images         # папка для хранения всех картинок
/templates
..... admin_panel.html # админка
..... editland.html    # блок обработки контента (данный код является частью в
admin_panel.html)
..... landing.html     # главная ленгинг страница
..... login_adm.html   # страница входа в админку
app.py                # главный код обработки backend
database.db           # База Данных проекта createdb.py          # скрипт
развертывания БД
createuser.py         # скрипт для создания пользователя
decode-session.py     # бонус поинт - скрипт декодирования session value
```

Исходный код структуры:

[ВЫ МОЖЕТЕ ВЗЯТЬ ТУТ](#)