

Урок 4. Создаем приложение для работы с записями блога

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc99377007)

[ВВЕДЕНИЕ 3](#_Toc99377008)

[РЕАЛИЗУЕМ НЕОБХОДИМЫЕ ФОРМЫ 4](#_Toc99377009)

[форма создания нового сообщения 4](#_Toc99377010)

[РЕАЛИЗУЕМ КОМПОНЕНТЫ CRUD-МЕХАНИЗМА 4](#_Toc99377011)

[контроллер отображения всех записей блога 4](#_Toc99377012)

[выполняем регистрацию приложения 5](#_Toc99377013)

[проверяем работу нового приложения 6](#_Toc99377014)

[реализуем вывод списка постов текущего пользователя 7](#_Toc99377015)

[реализуем создание новой записи блога 8](#_Toc99377016)

[реализуем функционал отображения содержимого записи блога 10](#_Toc99377017)

[реализуем функционал изменения содержимого записи блога 11](#_Toc99377018)

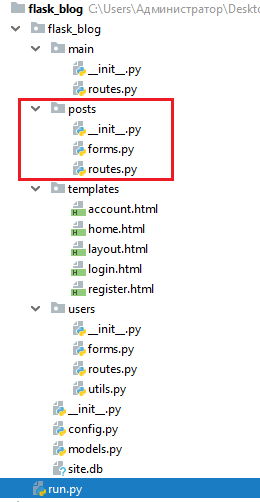
[реализуем функционал удаления поста блога 13](#_Toc99377019)

[ЗАКЛЮЧЕНИЕ 16](#_Toc99377020)



# ВВЕДЕНИЕ

Теперь в нашем проекте существует возможность регистрации новых пользователей. Теперь необходимо реализовать основной функционал – возможность создания, редактирования, удаления записей, а также просмотра общего списка записей и содержимого одной записи.

Добавим в пакет проекта flask\_blog новую директорию и назовем ее posts. Поскольку она будет являться пакетом, необходимо добавить файл \_\_init\_\_.py. Наше новое приложение обеспечит возможность пользователю вести свой блог, ведь это и есть основное назначение нашего проекта.

# РЕАЛИЗУЕМ НЕОБХОДИМЫЕ ФОРМЫ

## форма создания нового сообщения

Начнем с первой формы для создания нового сообщения.

**Листинг 1. flask\_blog/posts/forms.py**

**from** flask\_wtf **import** FlaskForm  
**from** wtforms **import** StringField, SubmitField, TextAreaField  
**from** wtforms.validators **import** DataRequired  
  
  
**class** PostForm(FlaskForm):  
 title = StringField(**'Title'**, validators=[DataRequired()])  
 content = TextAreaField(**'Content'**, validators=[DataRequired()])  
 submit = SubmitField(**'Post'**)

Код формы несложный. Похожий код мы уже разбирали ранее на одном из предыдущих уроков. Форма позволит указать название поста в блоге, заполнить пост текстом, а также выполнить подтверждение отправки данных (элемент формы submit = SubmitField(**'Post'**)).

Также для полей формы мы установили валидаторы, обеспечивающие проверку указанной пользователем даты.

# РЕАЛИЗУЕМ КОМПОНЕНТЫ CRUD-МЕХАНИЗМА

Нам предстоит реализовать возможность выполнения в блоге традиционных для таких проектов задач – создание записи, просмотр ее содержимого, редактирование, удаление, получение списка всех записей.

## компонент отображения всех записей блога

Добавим модуль routes.py в директорию posts.

**Листинг 2. flask\_blog/posts/routes.py**

**from** flask **import** (render\_template, url\_for, flash,  
 redirect, request, abort, Blueprint)  
**from** flask\_login **import** current\_user, login\_required  
**from** flask\_blog **import** db  
**from** flask\_blog.models **import** Post  
**from** flask\_blog.posts.forms **import** PostForm  
  
posts = Blueprint(**'posts'**, \_\_name\_\_)  
  
  
@posts.route(**"/allpost"**)  
@login\_required  
**def** allpost():  
 page = request.args.get(**'page'**, 1, type=int)  
 posts = Post.query.order\_by(Post.date\_posted.desc()).\  
 paginate(page=page, per\_page=5)  
 **return** render\_template(**'allpost.html'**, posts=posts)

Начнем с объявления нашего нового «блюпринта».

В приведенном листинге стоит отметить появление декоратора @login\_required, который означает, что переход к списку постов блога возможен только после прохождения пользователем авторизации.

Мы будем за счет ORM получать через класс-модель Post все записи, которые размещены в блоге. При этом настроим пагинацию – вывод на странице по пять постов.

Добавим в директорию с шаблонами файл allpost.html.

Дополним шаблон layout.html ссылкой на список всех записей.

**Листинг 3. flask\_blog/templates/layout.py**

<**div class="navbar-nav"**>  
{% if current\_user.is\_authenticated %}  
  
 <**a class="nav-item nav-link navigation-links" href="{{**

**url\_for('posts.allpost') }}"**><**i class="fa fa-th"**></**i**> Posts</**a**>  
  
  
 …

{% endif %}  
</**div**>

## выполняем регистрацию приложения

Наш «блюпринт» готов, теперь самое время его зарегистрировать.

Как всегда, перейдем в модуль \_\_init\_\_.py пакета проекта и выполним необходимые настройки.

**Листинг 4. flask\_blog/\_\_init\_\_.py**

**def** create\_app(config\_class=Config):  
 …  
 **from** flaskblog.posts.routes **import** posts  
 app.register\_blueprint(posts)  
 *…* **return** app

Сделаем еще несколько корректировок в нашем проекте, например, сделаем так, что при авторизации пользователя, он будет перенаправляться не на главную страницу блога, а на страницу со списком постов.

**Листинг 5. flask\_blog/users/routes.py**

@users.route(**"/login"**, methods=[**'GET'**, **'POST'**])  
**def** login():  
 **if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'posts.allpost'**))

form = LoginForm()

…

next\_page = request.args.get(**'next'**)

**return** redirect(next\_page) **if** next\_page

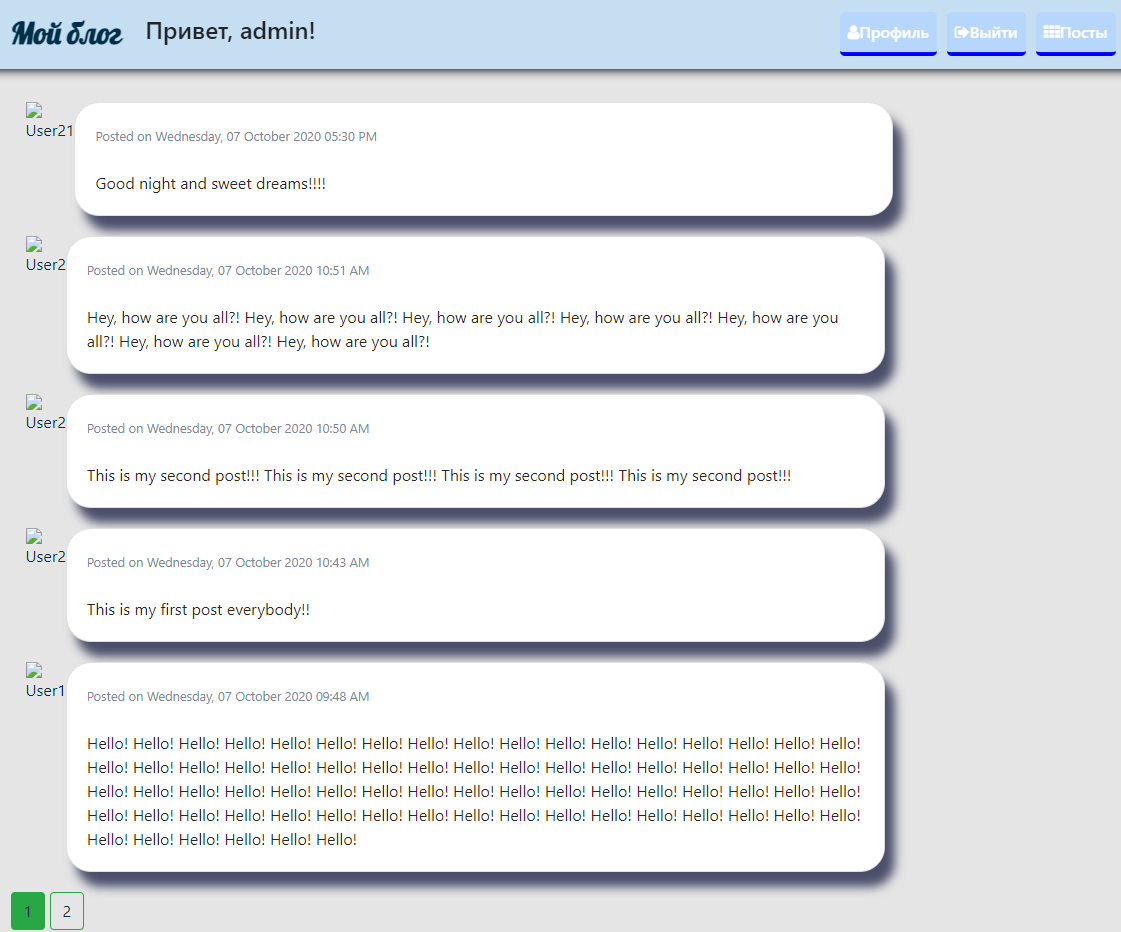
**else** redirect(url\_for(**'posts.allpost'**))

Теперь после авторизации пользователь нашего блога сможет сразу приступить к работе над постами.

## проверяем работу нового приложения

Выполним авторизацию ранее созданного пользователя.

Перейдем на страницу всех постов и увидим:

Кстати, у нас не выводятся фотографии пользователей – авторов постов. Но к этому мы еще вернемся.

## реализуем вывод списка постов текущего пользователя

Мы уже реализовали возможность отображения списка постов всех пользователей (контроллер allpost()), но нам также нужно добавить контроллер для визуализации списка постов только текущего авторизованного пользователя.

Для этого добавим в модуль routes.py приложения users контроллер:

**Листинг 6. flask\_blog/users/routes.py**

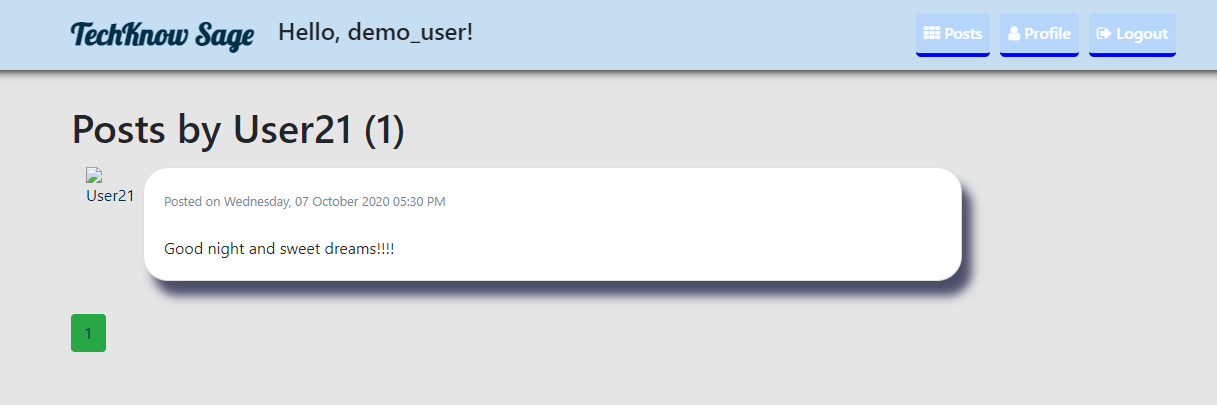
@users.route(**"/user/<string:username>"**)  
**def** user\_posts(username):  
 page = request.args.get(**'page'**, 1, type=int)  
 user = User.query.filter\_by(username=username).first\_or\_404()  
 posts = Post.query.filter\_by(author=user)\  
 .order\_by(Post.date\_posted.desc())\  
 .paginate(page=page, per\_page=5)  
 **return** render\_template(**'user\_posts.html'**, posts=posts, user=user)

Контроллер обеспечивает извлечение всех постов текущего пользователя и передачи списка в шаблон в виде контекста.

Ссылка, которую обрабатывает данный контроллер, находится в шаблоне all\_post.html.

<**a href="{{ url\_for('users.user\_posts', username=post.author.username) }}" id="author" style="**margin: 0 auto;**"**>{{ post.author.username }}</**a**>

Также нужно добавить шаблон **user\_posts.html** в директорию templates.

Перейдем в список постов конкретного пользователя:

## реализуем возможность создания новой записи блога

Теперь перейдем к следующему важному этапу – реализации функционала создания новой записи.

**Листинг 7. flask\_blog/posts/routes.py**

@posts.route(**"/post/new"**, methods=[**'GET'**, **'POST'**])  
@login\_required  
**def** new\_post():  
 form = PostForm()  
 **if** form.validate\_on\_submit():  
 post = Post(title=form.title.data, content=form.content.data,  
 author=current\_user)  
 db.session.add(post)  
 db.session.commit()  
 flash(**'Ваш пост создан!'**, **'success'**)  
 **return** redirect(url\_for(**'posts.allpost'**))  
 **return** render\_template(**'create\_post.html'**,  
 title=**'Новый пост'**, form=form, legend=**'Новый пост'**)

Код данного контроллера не должен вызвать сложностей. Мы создаем объект формы и при успешной ее валидации параметры пользователя в форме title=form.title.data, content=form.content.data будут передаваться через объект поста в базу данных. В поле с автором поста передается ссылка на объект текущего авторизованного пользователя.

При успешном добавлении поста мы будем перенаправлены на страницу со списком всех постов.

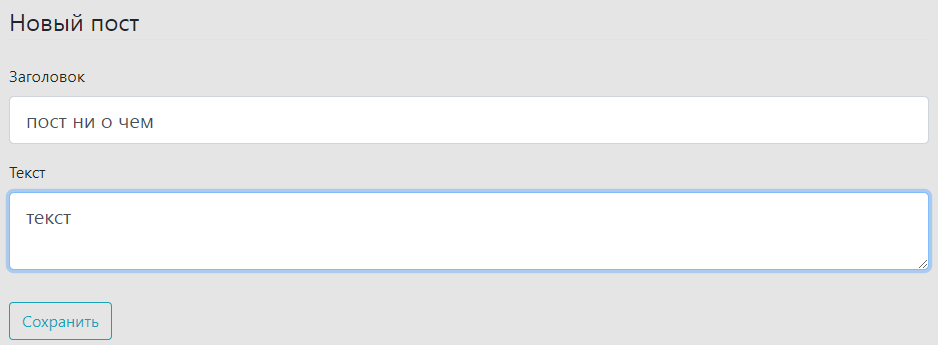
Добавим в директорию с шаблонами соответствующий файл create\_post.html.

Перейдем в шаблон layout.html и добавим ссылку на создание новой записи.

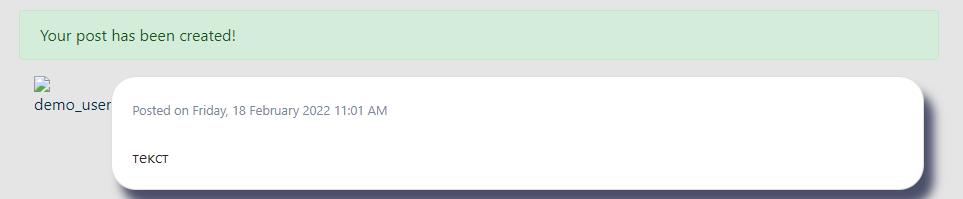
**Листинг 8. flask\_blog/templates/layout.html**

<**div class="navbar-nav"**>  
 {% if current\_user.is\_authenticated %}  
  
…  
 <**a class="nav-item nav-link navigation-links" href="{{**

**url\_for('posts.new\_post') }}"**><**i class="fa fa-plus"**></**i**> New Post</**a**>  
  
  
…  
 {% endif %}  
</**div**>

Теперь попытаемся создать новый пост.

Наш пост добавлен.



## реализуем функционал отображения содержимого записи блога

**Листинг 9. flask\_blog/posts/routes.py**

@posts.route(**"/post/<int:post\_id>"**)  
**def** post(post\_id):  
 post = Post.query.get\_or\_404(post\_id)  
 **return** render\_template(**'post.html'**, title=post.title, post=post)

В приведенном листинге нам важен маршрут, который указан для данного контроллера:

@posts.route(**"/post/<int:post\_id>"**)

У каждой записи блога существует свой идентификатор и когда мы будем переходить в карточку поста со страницы, содержащей все посты, в объект запроса будет передаваться идентификатор записи.

По этому идентификатору будет извлекаться запись из соответствующей таблицы базы данных и преобразовываться в объект класса-модели Post.

И этот объект мы будем передавать в контекст нашего шаблона (post=post).

Добавим в директорию с шаблонами соответствующий файл post.html.

Теперь нам нужно добавить ссылки для перехода к конкретному посту в шаблоны allpost.html и user\_posts.html.

**Листинг 10. flask\_blog/templates/allpost.py**

<**div class="card-body"**>  
 <**h4 class="card-title"**><**a class="title"**

**href="{{ url\_for('posts.post', post\_id=post.id) }}"**>

{{ post.title }}</**a**></**h4**>  
…

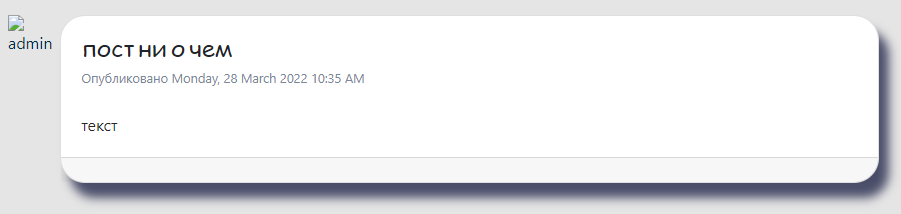
**Листинг 11. flask\_blog/templates/user\_posts.py**

<**div class="card-body"**>  
 <**h4 class="card-title"**>

<**a class="title"**

**href="{{ url\_for('posts.post', post\_id=post.id) }}"**>

{{ post.title }}</**a**></**h4**>  
…

Теперь попытаемся посмотреть содержимое поста.

## реализуем функционал изменения содержимого записи блога

**Листинг 12. flask\_blog/posts/routes.py**

@posts.route(**"/post/<int:post\_id>/update"**, methods=[**'GET'**, **'POST'**])  
@login\_required  
**def** update\_post(post\_id):  
 post = Post.query.get\_or\_404(post\_id)  
 **if** post.author != current\_user:  
 abort(403)  
 form = PostForm()  
 **if** form.validate\_on\_submit():  
 post.title = form.title.data  
 post.content = form.content.data  
 db.session.commit()  
 flash(**'Ваш пост обновлен!'**, **'success'**)  
 **return** redirect(url\_for(**'posts.post'**, post\_id=post.id))  
 **elif** request.method == **'GET'**:  
 form.title.data = post.title  
 form.content.data = post.content  
 **return** render\_template(**'create\_post.html'**, title=**'Обновление поста'**,  
 form=form, legend=**'Обновление поста'**)

Данный листинг несложный и позволит нам выполнить редактирование содержимого записи блога, в частности, поменять заголовок поста, его содержимое.

Важно, что при открытии страницы редактирования поста, мы увидим в форме его текущие данные:

**elif** request.method == **'GET'**:  
 form.title.data = post.title  
 form.content.data = post.content

Также стоит отметить, что доступ к редактированию поста будет только у автора.

**if** post.author != current\_user:  
 abort(403)

Добавим в шаблон post.html выражения для перехода на страницу изменения содержимого поста.

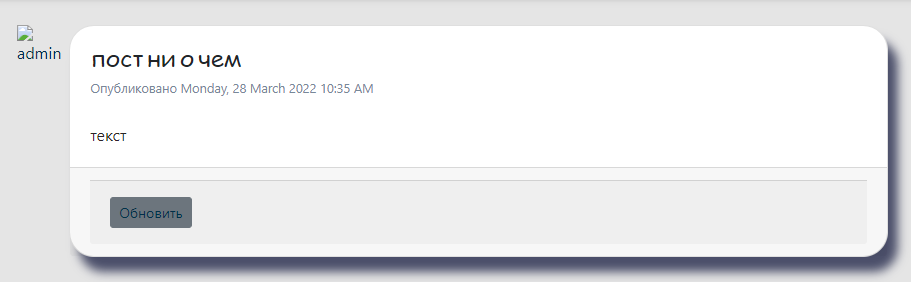
**Листинг 13. flask\_blog/templates/post.html**

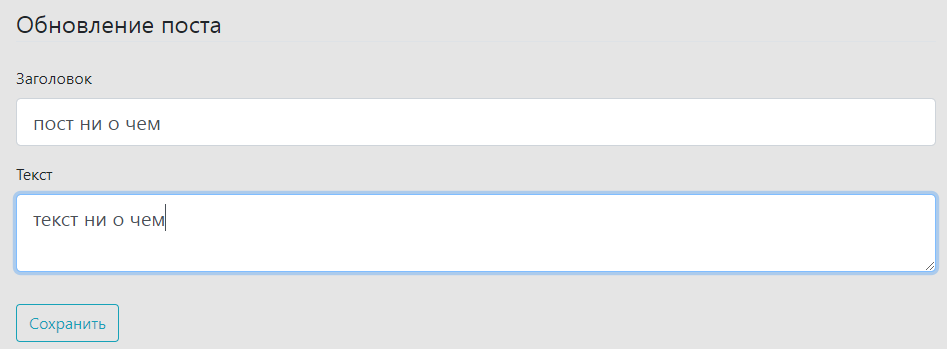
…

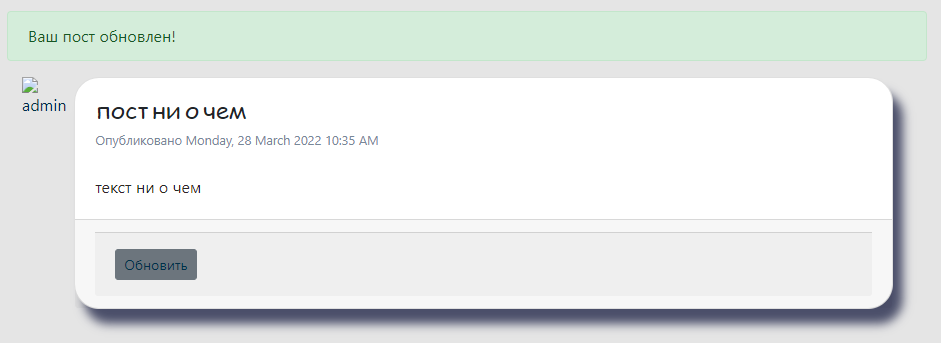
<**div class="card-footer" style="**font-size: 20px;**"**>{% if post.author == current\_user %}  
 <**a class="btn btn-secondary btn-sm mt-1 mb-1" href="{{**

**url\_for('posts.update\_post', post\_id=post.id) }}"**>Update</**a**>  
 {% endif %}  
</**div**>

Проверим как работает функционал изменения содержимого поста:

Перейдем к редактированию поста:

Результат:



## реализуем функционал удаления поста блога

**Листинг 14. flask\_blog/posts/routes.py**

@posts.route(**"/post/<int:post\_id>/delete"**, methods=[**'POST'**])  
@login\_required  
**def** delete\_post(post\_id):  
 post = Post.query.get\_or\_404(post\_id)  
 **if** post.author != current\_user:  
 abort(403)  
 db.session.delete(post)  
 db.session.commit()  
 flash(**'Ваш пост был удален!'**, **'success'**)  
 **return** redirect(url\_for(**'posts.allpost'**))

У контроллера простейший код, обеспечивающий удаление поста и перенаправление пользователя на страницу со списком постов.

Добавим в шаблон post.html выражения для перехода на страницу удаления содержимого поста.

**Листинг 15. flask\_blog/templates/post.html**

{% if post.author == current\_user %}  
 <**button type="button" class="btn btn-danger btn-sm m-1" data-toggle="modal"**

**data-target="#deleteModal"**>Удалить</**button**>

{% endif %}

А также блок с кодом модального окна для подтверждения удаления поста.

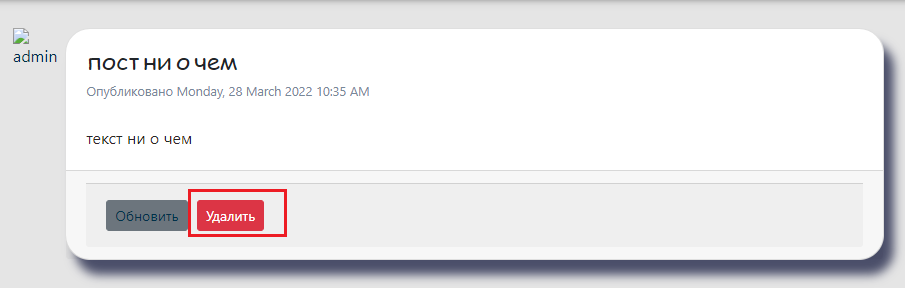
**Листинг 16. flask\_blog/templates/post.html**

<**div class="modal fade" id="deleteModal" tabindex="-1" role="dialog" aria-labelledby="deleteModalLabel" aria-hidden="true"**>  
 <**div class="modal-dialog" role="document"**>  
 <**div class="modal-content"**>  
 <**div class="modal-header"**>  
 <**h5 class="modal-title" id="deleteModalLabel"**>Удалить пост?</**h5**>  
 <**button type="button" class="close" data-dismiss="modal"**

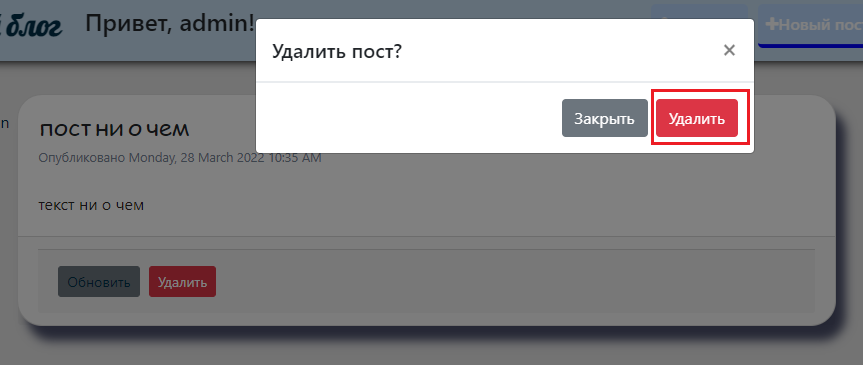
**aria-label="Close"**>  
 <**span aria-hidden="true"**>**&times;**</**span**>  
 </**button**>  
 </**div**>  
 <**div class="modal-footer"**>  
 <**button type="button" class="btn btn-secondary"**

**data-dismiss="modal"**>Закрыть</**button**>  
 <**form action="{{ url\_for('posts.delete\_post', post\_id=post.id) }}"**

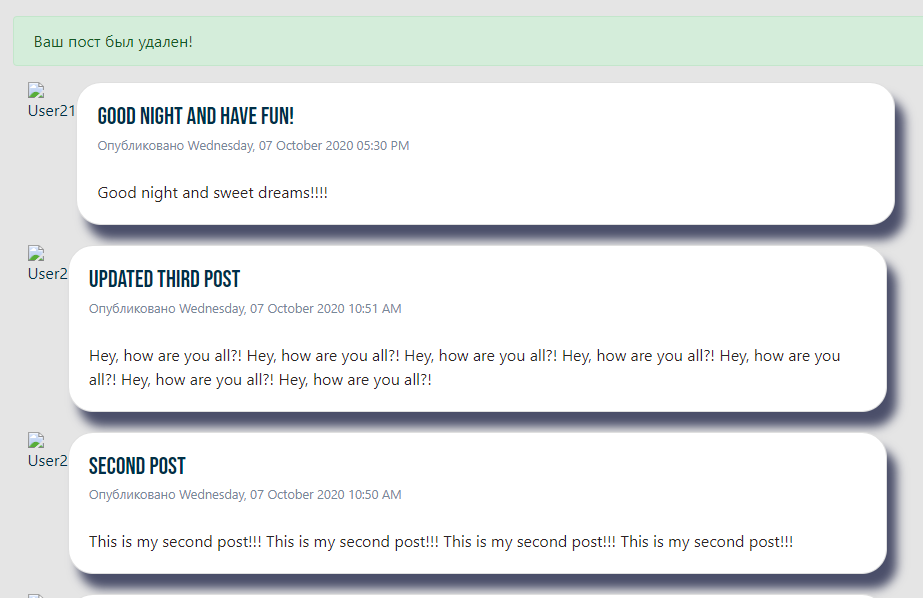
**method="POST"**>  
 <**input class="btn btn-danger" type="submit" value="Delete"**>  
 </**form**>  
 </**div**>  
 </**div**>  
 </**div**>  
</**div**>

Проверим удаление поста.

Подтверждаем удаление:



Результат:



# Вид сверху на рукопожатие над деловыми документамиВид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

По итогам урока 4 мы реализовали возможность создания, редактирования и отображения постов пользователя, а также отображения списка всех имеющихся постов и постов конкретного пользователя. Для этого мы подготовили соответствующие контроллеры и шаблоны.

Текущая структура проекта приведена на рис. 1.

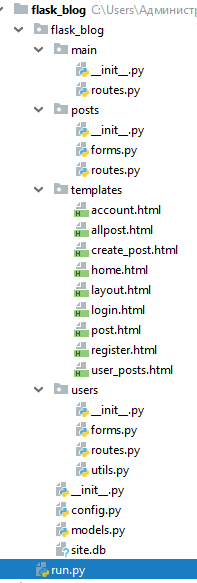


Рис. 1. Структура проекта