

Урок 3. Создаем приложение для работы с пользователями

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc99367885)

[ВВЕДЕНИЕ 3](#_Toc99367886)

[СОЗДАЕМ ФОРМЫ 3](#_Toc99367887)

[форма регистрации нового пользователя 4](#_Toc99367888)

[форма аутентификации пользователя 6](#_Toc99367889)

[форма обновления параметров пользователя 6](#_Toc99367890)

[форма запроса на обновление параметров пользователя 7](#_Toc99367891)

[форма изменения пароля пользователя 8](#_Toc99367892)

[СОЗДАЕМ СЛУЖЕБНЫЕ УТИЛИТЫ ПРИЛОЖЕНИЯ USERS 8](#_Toc99367893)

[функция изменения аватарки пользователя 8](#_Toc99367894)

[НАСТРАИВАЕМ РЕГИСТРАЦИЮ ПОЛЬЗОВАТЕЛЯ 9](#_Toc99367895)

[функция регистрации нового пользователя 10](#_Toc99367896)

[шаблон регистрации нового пользователя 12](#_Toc99367897)

[ВЫПОЛНЯЕМ РЕГИСТРАЦИЮ ПРИЛОЖЕНИЯ 13](#_Toc99367898)

[ПРОВЕРЯЕМ РАБОТУ ПРИЛОЖЕНИЯ 14](#_Toc99367899)

[РЕАЛИЗУЕМ АВТОРИЗАЦИЮ ПОЛЬЗОВАТЕЛЯ 17](#_Toc99367900)

[ПРОВЕРЯЕМ РАБОТУ ПРИЛОЖЕНИЯ 19](#_Toc99367901)

[РЕАЛИЗУЕМ ПРОФИЛЬ ПОЛЬЗОВАТЕЛЯ 20](#_Toc99367902)

[РЕАЛИЗУЕМ ЗАВЕРШЕНИЕ СЕССИИ ПОЛЬЗОВАТЕЛЯ 23](#_Toc99367903)

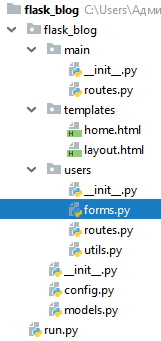
[ЗАКЛЮЧЕНИЕ 25](#_Toc99367904)



# ВВЕДЕНИЕ

В рамках второго урока мы создали наше первое приложение и для него – соответствующий макет («блюпринт»). Теперь в нашем проекте реализовано отображение содержимого главной страницы блога. Далее наша задача подготовить второй «блюпринт», функционал которого призван обеспечить работу подсистемы пользователей. Т.е. необходимо реализовать возможность регистрации пользователя блога, его авторизации и т.д.

# СОЗДАЕМ ФОРМЫ

Добавим в пакет проекта flask\_blog новую директорию users, оформим ее также в качестве пакета. Кроме того, добавим модуль, который будет содержать программный код экранных форм блога для работы с пользователями.

## форма регистрации нового пользователя

Начнем с первой формы для регистрации нового пользователя.

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

**from** flask\_wtf **import** FlaskForm  
**from** flask\_wtf.file **import** FileField, FileAllowed  
**from** wtforms **import** StringField, PasswordField, SubmitField, BooleanField  
**from** wtforms.validators **import** DataRequired, Length, Email, EqualTo, \  
 ValidationError  
**from** flask\_login **import** current\_user  
**from** test\_flask.models **import** User

**class** RegistrationForm(FlaskForm):  
 username = StringField(**'Имя пользователя:'**,  
 validators=[DataRequired(),  
 Length(min=2, max=20)])  
 email = StringField(**'Email:'**,  
 validators=[DataRequired(), Email()])  
 password = PasswordField(**'Пароль:'**, validators=[DataRequired()])  
 confirm\_password = PasswordField(**'Подтвердить пароль'**,  
 validators=[DataRequired(),  
 EqualTo(**'password'**)])  
 submit = SubmitField(**'Зарегистрироваться'**)  
  
  
 **def** validate\_username(self, username):  
 user = User.query.filter\_by(username=username.data).first()  
 **if** user:  
 **raise** ValidationError(  
 **'Это имя занято. Пожалуйста, выберите другое.'**)  
  
  
 **def** validate\_email(self, email):  
 user = User.query.filter\_by(email=email.data).first()  
 **if** user:  
 **raise** ValidationError(  
 **'Этот email занят. Пожалуйста, выберите другой.'**)

Класс формы наследует интерфейс встроенного класса FlaskForm. Это простая Flask-форма с возможностью валидации полей.

Совершенно новый для нас код. Разберем каждое его выражение.

username = StringField(**'Имя пользователя:'**,  
 validators=[DataRequired(),  
 Length(min=2, max=20)])

Это поле, через которое мы можем ввести текстовое значение минимальной длиной в два символа и максимальной в двадцать символов. Это имя пользователя.

Для данного поля установлен валидатор DataRequired(), за счет которого проверяется наличие содержимого в поле, иначе выводится сообщение об ошибке:

email = StringField(**'Email:'**,  
 validators=[DataRequired(), Email()])

Дополнительный валидатор Email() обеспечивает проверку указанного email-адреса:

password = PasswordField(**'Пароль:'**, validators=[DataRequired()])

Поле, предназначенное для указания пароля пользователя:

confirm\_password = PasswordField(**'Подтвердить пароль'**,  
 validators=[DataRequired(),  
 EqualTo(**'password'**)])

Поле, которое предназначено для подтверждения пароля с целью проверки его соответствия первоначально введенному паролю.

Проверку соответствия обеспечивает валидатор EqualTo('password').

submit = SubmitField(**'Зарегистрироваться'**)

Поле, соответствующее элементу управления – кнопке, для подтверждения регистрации пользователя.

Также добавим в класс формы два пользовательских метода валидации имени пользователя и адреса электронной почты.

**def** validate\_username(self, username):  
 user = User.query.filter\_by(username=username.data).first()  
 **if** user:  
 **raise** ValidationError(  
 **'Это имя занято. Пожалуйста, выберите другое.'**)

Данный метод обеспечивает получение записи из таблицы по логину пользователя и представление этой записи в виде объекта модели-класса User. Если запись с тем именем, которое ввел пользователь, уже существует в базе данных, возникает генерация сообщения об ошибке.

**def** validate\_email(self, email):  
 user = User.query.filter\_by(email=email.data).first()  
 **if** user:  
 **raise** ValidationError(  
 **'Этот email занят. Пожалуйста, выберите другой.'**)

Аналогичным образом работает метод для проверки возможности использования указанного адреса электронной почты пользователя.

## форма авторизации пользователя

После регистрации нового пользователя требуется процедура авторизации, которую мы реализуем через соответствующую форму.

**Листинг 2. flask\_blog/users/forms.py**

**class** LoginForm(FlaskForm):  
 email = StringField(**'Email:'**,  
 validators=[DataRequired(), Email()])  
 password = PasswordField(**'Пароль:'**, validators=[DataRequired()])  
 remember = BooleanField(**'Напомнить пароль'**)  
 submit = SubmitField(**'Войти'**)

Каждое из представленных полей нам уже знакомо за исключением поля:

remember = BooleanField(**'Напомнить пароль'**)

По сути это поле-флажок, через которое мы реализуем возможность напоминания пользователю о его забытом пароле.

## форма обновления параметров пользователя

Реализуем также форму редактирования параметров зарегистрированного пользователя.

**Листинг 3. flask\_blog/users/forms.py**

**class** UpdateAccountForm(FlaskForm):  
 username = StringField(**'Имя пользователя'**,  
 validators=[DataRequired(),  
 Length(min=2, max=20)])  
 email = StringField(**'Email'**,  
 validators=[DataRequired(), Email()])  
 picture = FileField(**'Обновить фото профиля'**,  
 validators=[FileAllowed([**'jpg'**, **'png'**])])  
 submit = SubmitField(**'Обновить'**)  
  
 **def** validate\_username(self, username):  
 **if** username.data != current\_user.username:  
 user = User.query.filter\_by(username=username.data).first()  
 **if** user:  
 **raise** ValidationError(**'Это имя занято. '  
 'Пожалуйста, выберите другой'**)  
  
 **def** validate\_email(self, email):  
 **if** email.data != current\_user.email:  
 user = User.query.filter\_by(email=email.data).first()  
 **if** user:  
 **raise** ValidationError(**'Этот email занят'  
 'Пожалуйста, выберите другой'**)

Класс-форма UpdateAccountForm содержит уже знакомые нам поля.

Новый атрибут-поле данного класса:

picture = FileField(**'Обновить фото профиля'**,  
 validators=[FileAllowed([**'jpg'**, **'png'**])])

Данное поле обеспечивает возможность выбора фото профиля пользователя. Контроль типа данных фото обеспечивает валидатор FileAllowed. В данном случае допустимы типы файлов: jpg и png.

Методы валидации имени пользователя и адреса электронной почты мы уже разобрали ранее.

## форма запроса на обновление параметров пользователя

На тот случай, если пользователь решит изменить свои регистрационные данные, в частности, пароль, нам нужно реализовать отдельную форму для запроса на редактирование.

**Листинг 4. flask\_blog/users/forms.py**

**class** RequestResetForm(FlaskForm):  
 email = StringField(**'Email'**,  
 validators=[DataRequired(), Email()])  
 submit = SubmitField(**'Изменить пароль'**)  
  
 **def** validate\_email(self, email):  
 user = User.query.filter\_by(email=email.data).first()  
 **if** user **is None**:  
 **raise** ValidationError(**'Аккаунт с данным email-адресом '  
 'отсутствует. '  
 'Вы можете зарегистрировать его'**)

Данная форма позволит пользователю указать email-адрес и выполнить запрос на редактирование параметров пользователя.

При этом, если мы введем email-адрес, который отсутствует в базе данных, будет отправлено предложение зарегистрировать новый аккаунт с этим адресом.

## форма изменения пароля пользователя

Итак, мы создали форму для того, чтобы отправить запрос на изменение пароля. Теперь добавим непосредственно форму редактирования пароля пользователя.

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

**class** ResetPasswordForm(FlaskForm):  
 password = PasswordField(**'Пароль:'**, validators=[DataRequired()])  
 confirm\_password = PasswordField(**'Подтвердите пароль'**,  
 validators=[DataRequired(),  
 EqualTo(**'password'**)])  
 submit = SubmitField(**'Переустановить пароль'**)

Данная форма позволит нам изменить пароль.

# СОЗДАЕМ СЛУЖЕБНЫЕ УТИЛИТЫ ПРИЛОЖЕНИЯ USERS

Создадим в директории users дополнительный модуль с кодом служебных утилит.

Он будет содержать код функции, которая обеспечивает изменение аватарки пользователя.

## функция изменения аватарки пользователя

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

**import** os  
**from** secrets **import** token\_hex  
**from** PIL **import** Image  
**from** flask **import** url\_for, current\_app**def** save\_picture(form\_picture):  
 random\_hex = token\_hex(8)  
 \_, f\_ext = os.path.splitext(form\_picture.filename)  
 picture\_fn = random\_hex + f\_ext  
 picture\_path = os.path.join(current\_app.root\_path,

**'static/profile\_pics'**, picture\_fn)  
  
 output\_size = (150, 150)  
 i = Image.open(form\_picture)  
 i.thumbnail(output\_size)  
 i.save(picture\_path)  
  
 **return** picture\_fn

В данном коде два важных импорта.

**from** secrets **import** token\_hex

Функция token\_hex() возвращает случайную строку в шестнадцатеричном формате. Модуль secrets содержит функции для создания безопасных токенов, используемых при процедурах сброса пароля, обновления важных параметров и т.д. Строка содержит 8 случайных байтов, каждый из которых трансформируется в две шестнадцатеричные цифры.

**from** PIL **import** Image

PIL это популярная библиотека для работы с изображениями. Основные функции для работы с изображениями находятся в ее модуле Image.

Функция принимает объект формы, через который мы получаем имя нового файла с аватаркой пользователя – filename.

# НАСТРАИВАЕМ РЕГИСТРАЦИЮ ПОЛЬЗОВАТЕЛЯ

Это большой раздел курса, в рамках которого мы создадим необходимые функции-контроллеры и настроим маршрутизацию.

## выполняем импорты

Нам предстоит в директории users создать новый модуль routes.py, в котором реализовать создание нового «блюпринта» и выполнить сопутствующие служебные импорты.

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

**from** flask **import** render\_template, url\_for, flash,

redirect, request, Blueprint  
**from** flask\_login **import** login\_user, current\_user,

logout\_user, login\_required  
**from** flask\_blog **import** db, bcrypt  
**from** flask\_blog.models **import** User, Post  
**from** flask\_blog.users.forms **import** (RegistrationForm, LoginForm, UpdateAccountForm, RequestResetForm, ResetPasswordForm)

**from** flask\_blog.users.utils **import** save\_picture  
  
users = Blueprint(**'users'**, \_\_name\_\_)

Назначение каждого импорта мы рассмотрим далее подробнее. Здесь есть импорты как стандартных встроенных компонентов, так и компонентов нашего проекта.

## функция регистрации нового пользователя

**Листинг 8. flask\_blog/users/routes.py**

@users.route(**"/register"**, methods=[**'GET'**, **'POST'**])  
**def** register():  
 **if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'main.home'**))  
 form = RegistrationForm()  
 **if** form.validate\_on\_submit():  
 hashed\_password = bcrypt.\  
 generate\_password\_hash(form.password.data).decode(**'utf-8'**)  
 user = User(username=form.username.data,  
 email=form.email.data, password=hashed\_password)  
 db.session.add(user)  
 db.session.commit()  
 flash(**'Ваша учетная запись была создана!'  
 ' Теперь вы можете войти в систему'**, **'success'**)  
 **return** redirect(url\_for(**'users.login'**))  
 **return** render\_template(**'register.html'**, title=**'Register'**, form=form)

Выражение:

@users.route(**"/register"**, methods=[**'GET'**, **'POST'**])

Означает, что мы будем работать с пространством маршрутов нашего текущего «блюпринта» users.

Переход по указанному в декораторе пути будет означать запуск соответствующего контроллера.

Параметр:

methods=[**'GET'**, **'POST'**]

Означает, что наш контроллер сможет одновременно работать с GET- и POST-запросами.

*Общий механизм работы контроллера*

*Если мы уже находимся в системе под одним из активных пользователей, то попытка перехода на страницу регистрации обеспечит перенаправление нас на главную страницу. Иначе, если вы не зарегистрированный пользователь, то в шаблоне регистрации пользователя будет создан и загружен объект регистрационной формы, через которую вы сможете ввести свои данные. Если форма с введенными данными пройдет валидацию, данные будут отправлены POST-запросом на сторону сервера, в базе данных появятся параметры нового пользователя и вас перенаправят на страницу авторизации.*

Теперь разберем отдельные выражения представленного кода.

Контроллер register() обеспечивает рендеринг шаблона регистрации с передачей соответствующего контекста, в том числе объекта формы регистрации и заголовка страницы регистрации:

**return** render\_template(**'register.html'**, title=**'Register'**, form=form)

Теперь подробнее разберем код контроллера:

**from** flask **import** url\_for

**from** flask\_login **import** current\_user

**if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'main.home'**))

Если пользователь является действующим пользователем нашего блога и прошел аутентификацию, то при попытке выполнить регистрацию, он будет перенаправлен на главную страницу.

Далее, при успешной валидации параметров формы:

**if** form.validate\_on\_submit():

…

На основе содержимого формы будет создаваться объект модели-класса User (пользователь), при этом пароль пользователь будет хешироваться.

hashed\_password = bcrypt.\  
 generate\_password\_hash(form.password.data).decode(**'utf-8'**)  
user = User(username=form.username.data,  
 email=form.email.data, password=hashed\_password)

Параметры созданного объекта-пользователя посредством ORM-библиотеки SQLAlchemy будут передаваться в базу данных и сохраняться.

db.session.add(user)  
db.session.commit()

Далее будет выводиться простое всплывающее сообщение для пользователя:

flash(**'Ваша учетная запись была создана!'  
 ' Теперь вы можете войти в систему'**, **'success'**)

Обратим внимание на функцию вычисления хеша пароля.

hashed\_password = bcrypt.\  
 generate\_password\_hash(form.password.data).decode(**'utf-8'**)

Расширение Flask-Bcrypt для Flask мы уже установили. Теперь необходимо создать объект главного класса-конструктора этого расширения и выполнить его регистрацию.

Для этого в модуле \_\_init\_\_.py пакета проекта добавить:

**Листинг 9. flask\_blog/\_\_init\_\_.py**

**…**  
**from** flask\_bcrypt **import** Bcrypt  
  
  
…  
bcrypt = Bcrypt()  
  
  
**def** create\_app():  
 …  
 bcrypt.init\_app(app)  
  
 **…**

## шаблон регистрации нового пользователя

На данный момент нам необходимо добавить в директорию templates шаблон register.html. Добавим его.

Шаблон имеет несложный код.

Также нам необходимо добавить ссылку на кнопку регистрации на главную страницу. В нашем случае, мы добавим их в базовый шаблон layout.html.

**Листинг 10. flask\_blog/templates/layout.html**

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

**url\_for('users.register')**

**}}"**>Зарегистрироваться</**a**>  
 {% endif %}  
</**div**>

Особого внимания заслуживают наши динамические ссылки на другие страницы проекта, например,

**href="{{ url\_for('users.register') }}**

Функция url\_for() понятна. Так мы обозначаем нашу динамическую ссылку. А дальше самое интересное – ее аргумент **'users.register'.**

Он означает, что мы работаем с «роутами» (маршрутами) блюпринта «users» и для обработки перехода на текущую ссылку используем маршрут «/register» именно этого блюпринта. Т.е. к строке запроса будет автоматически добавляться «/register».

Это чем-то похоже на маршрутизацию в Django, когда мы таким образом говорим, что работаем с маршрутами пространства имен, относящегося к конкретному приложению. Только в Django эта запись выглядит так:

**href="{% url 'users:register' %}**

# ВЫПОЛНЯЕМ РЕГИСТРАЦИЮ ПРИЛОЖЕНИЯ

Итак, наш «блюпринт» готов, а также реализован некоторые начальный функционал подсистемы работы с пользователями. Самое время зарегистрировать «блюпринт».

Перейдем в файл \_\_init\_\_.py пакета проекта и добавим соответствующие выражения.

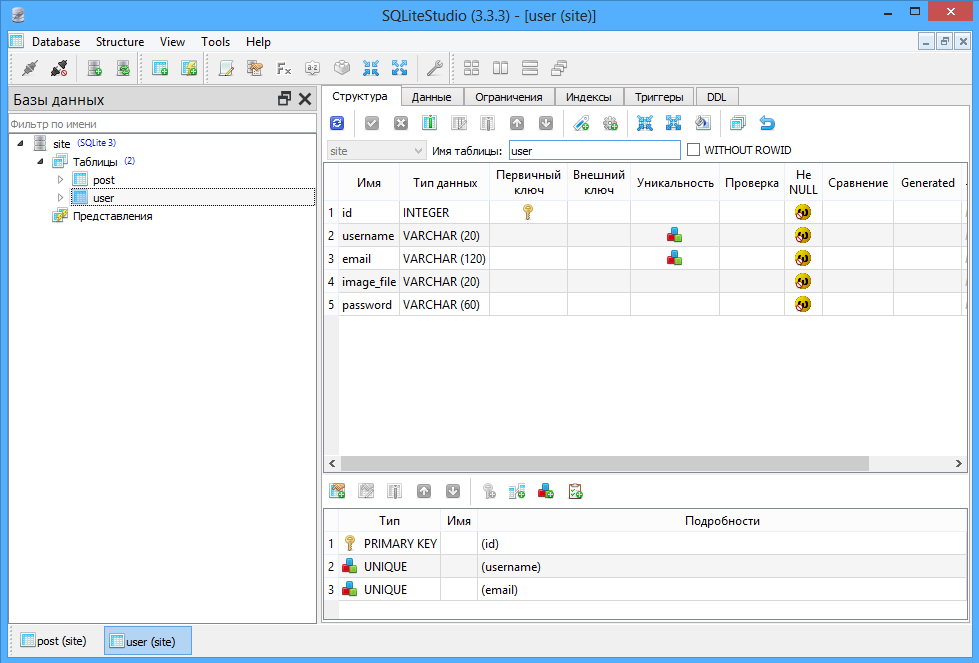
**Листинг 11. flask\_blog/\_\_init\_\_.py**

**def** create\_app():  
 …  
 **from** flask\_blog.users.routes **import** users  
  
 …  
 app.register\_blueprint(users)  
 **return** app

Казалось бы, работа на текущем шаге завершена, и можно выполнить тестовый запуск проекта, после чего попытаться зарегистрировать нового пользователя. Но если мы попытаемся выполнить запуск, то получим ошибку, связанную с отсутствием таблиц в базе данных.

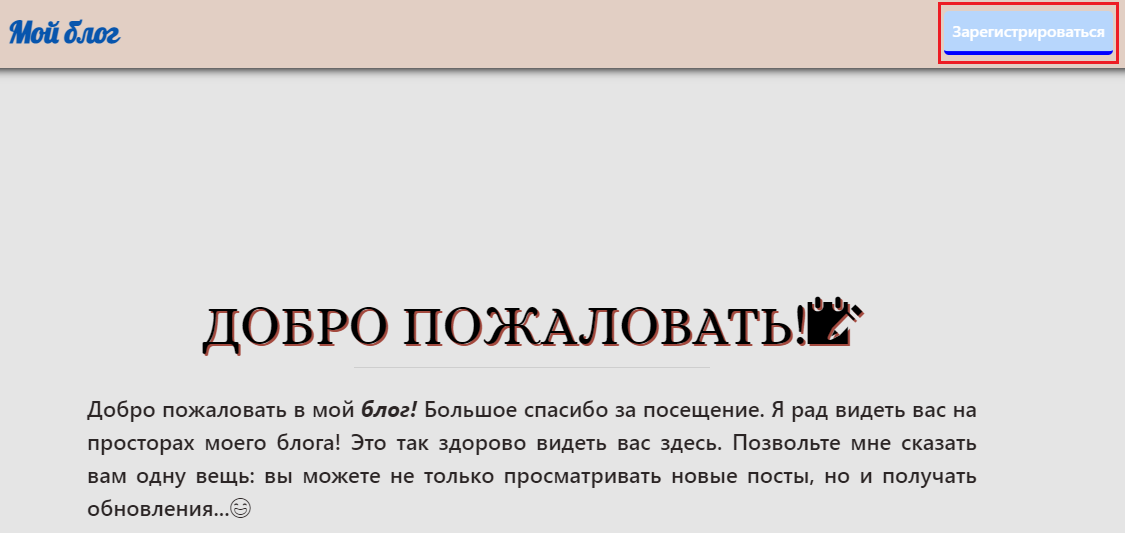
Действительно, при регистрации нового пользователя мы ведь проверяем существует ли он в базе данных. Но как это можно проверить, если база пуста.

Можно заранее через браузер баз данных SQLiteStudio заранее создать необходимые таблицы.

Вы можете просто скопировать в корень проекта заранее подготовленный файл базы данных. В нем есть все необходимое.

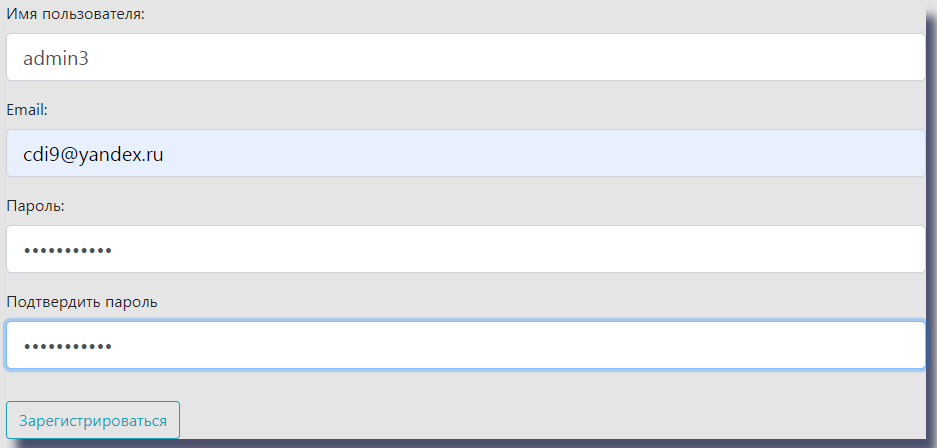
# ПРОВЕРЯЕМ РАБОТУ ПРИЛОЖЕНИЯ

Запустим файл run.py и перейдем на главную страницу.

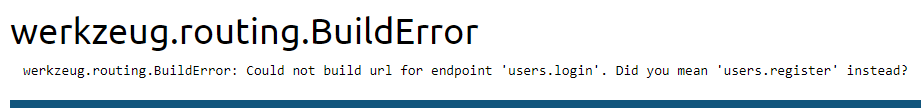
Перейдем к форме регистрации нового пользователя:

Откроем форму:

Заполним форму данными и выполним регистрацию нового пользователя:



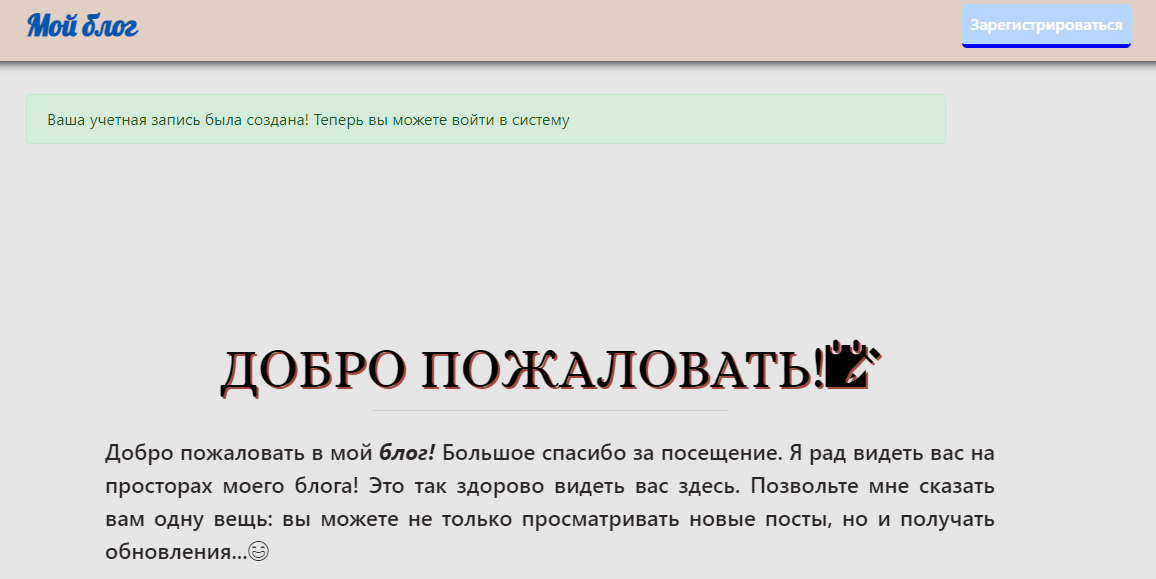
Сохраним данные, но в браузере последует ошибка:

Дело в том, что при успешной регистрации в контроллере register у нас есть такой редирект:

**return** redirect(url\_for(**'users.login'**))

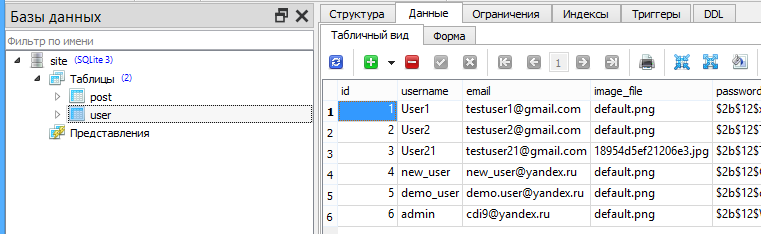
Но ведь мы еще не написали соответствующий обработчик!

Поэтому чтобы ошибки не было, изменим временно редирект на главную страницу:

**return** redirect(url\_for(**'main.home'**))

Все работает.

Проверим наличие нового пользователя в базе данных. Он добавлен



# РЕАЛИЗУЕМ АВТОРИЗАЦИЮ ПОЛЬЗОВАТЕЛЯ

После регистрации нового пользователя требуется процедура авторизации, которую мы реализуем через соответствующий класс-форму.

Перейдем непосредственно к контроллеру, отвечающему за авторизацию пользователя.

**Листинг 12. step\_3/flask\_blog/users/routes.py**

@users.route(**"/login"**, methods=[**'GET'**, **'POST'**])  
**def** login():  
 **if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'main.home'**))  
  
 form = LoginForm()  
 **if** form.validate\_on\_submit():  
 user = User.query.filter\_by(email=form.email.data).first()  
 **if** user **and** bcrypt.check\_password\_hash(user.password,  
 form.password.data):  
 login\_user(user, remember=form.remember.data)  
  
 **return** redirect(url\_for(**'main.home'**))  
 **else**:  
 flash(**'Войти не удалось. Пожалуйста, '  
 'проверьте электронную почту и пароль'**, **'внимание'**)  
 **return** render\_template(**'login.html'**, title=**'Аутентификация'**, form=form)

*Общий механизм работы контроллера*

*Если пользователь уже авторизован, то он будет перенаправлен на главную страницу. При отправке данных, указанных в форме, будет выполняться извлечение адреса электронной почты, указанного пользователем и по нему через ORM получены параметры соответствующего пользователя. После чего будут сверены хеш пароля, записанный в базу данных и хеш, вычисленный на основе пароля, введенного в форму. Если они совпадут, будет выполнена процедура аутентификации пользователя. И перенаправление его на главную страницу. Иначе выведено сообщение о том, что аутентификация не удалась.*

Большинство выражений приведенного листинга нам уже знакомы, кроме некоторых.

user = User.query.filter\_by(email=form.email.data).first()

Через класс-модель User получаем объект соответствующей записи базы данных. Это именно объект класса-модели, атрибуты которого будут ссылаться на значения соответствующих полей строки таблицы базы данных.

Если пользователь, которого нам необходимо «извлечь» из базы данных, существует и хеш пароля пользователя из базы данных совпадает с хешем пароля из формы авторизации, то выполняется процедура аутентификации.

login\_user(user, remember=form.remember.data)

Это встроенная функция, принимающая объект пользователя, устанавливающая с ним сессию при успешной аутентификации. Параметр remember позволяет пользователю быть авторизованным дольше и его параметры авторизации были бы сохранены.

Также нам нужно добавить в директорию templates шаблон login.html.

А в шаблон layout.html нам нужно добавить кнопку для перехода к авторизации.

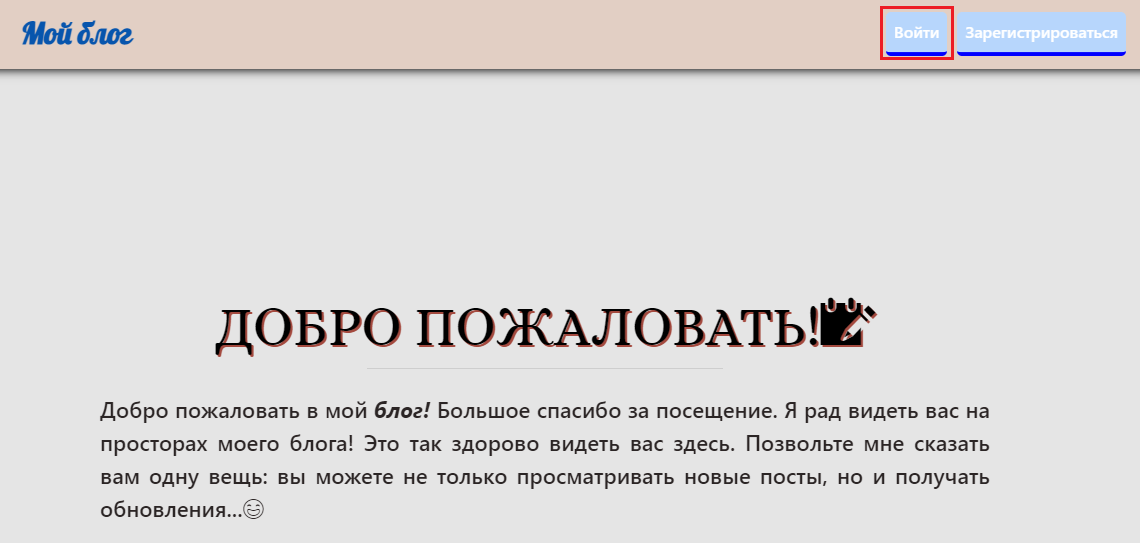
**Листинг 13. flask\_blog/templates/layout.html**

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

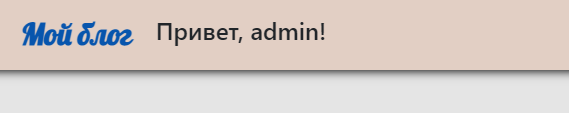
**url\_for('users.login') }}"**>Войти</**a**>  
 <**a class="nav-link nav-item navigation-links" href="{{**

**url\_for('users.register') }}"**>Зарегистрироваться</**a**>  
 {% endif %}  
</**div**>

# ПРОВЕРЯЕМ РАБОТУ ПРИЛОЖЕНИЯ

Запустим файл run.py и попробуем авторизовать нашего нового пользователя.

Перейдем к форме авторизации и укажем необходимые данные.

Процедура авторизации успешно завершена.

# РЕАЛИЗУЕМ ПРОФИЛЬ ПОЛЬЗОВАТЕЛЯ

Необходимо, чтобы после успешной авторизации у пользователя была бы возможность выполнить перехода в личный профиль для редактирования его параметров.

**Листинг 14. flask\_blog/users/forms.py**

**class** UpdateAccountForm(FlaskForm):  
 username = StringField(**'Имя пользователя'**,  
 validators=[DataRequired(),  
 Length(min=2, max=20)])  
 email = StringField(**'Email'**,  
 validators=[DataRequired(), Email()])  
 picture = FileField(**'Обновить фото профиля'**,  
 validators=[FileAllowed([**'jpg'**, **'png'**])])  
 submit = SubmitField(**'Обновить'**)  
  
 @staticmethod  
 **def** validate\_username(username):  
 **if** username.data != current\_user.username:  
 user = User.query.filter\_by(username=username.data).first()  
 **if** user:  
 **raise** ValidationError(**'Это имя занято. '  
 'Пожалуйста, выберите другой'**)  
  
 @staticmethod  
 **def** validate\_email(email):  
 **if** email.data != current\_user.email:  
 user = User.query.filter\_by(email=email.data).first()  
 **if** user:  
 **raise** ValidationError(**'Этот email занят'  
 'Пожалуйста, выберите другой'**)

Далее этого добавим в модуль routes.py нашего блюпринта users следующий контроллер.

**Листинг 15. flask\_blog/users/routes.py**

@users.route(**"/account"**, methods=[**'GET'**, **'POST'**])  
@login\_required  
**def** account():  
 form = UpdateAccountForm()  
 **if** form.validate\_on\_submit():  
 **if** form.picture.data:  
 picture\_file = save\_picture(form.picture.data)  
 current\_user.image\_file = picture\_file  
 current\_user.username = form.username.data  
 current\_user.email = form.email.data  
 db.session.commit()  
 flash(**'Ваш аккаунт был обновлен!'**, **'success'**)  
 **return** redirect(url\_for(**'users.account'**))  
 **elif** request.method == **'GET'**:  
 form.username.data = current\_user.username  
 form.email.data = current\_user.email  
 page = request.args.get(**'page'**, 1, type=int)  
 user = User.query.filter\_by(username=

form.username.data).first\_or\_404()  
 posts = Post.query.filter\_by(author=user) \  
 .order\_by(Post.date\_posted.desc()) \  
 .paginate(page=page, per\_page=5)  
 image\_file = url\_for(**'static'**, filename=**'profile\_pics/'** +  
 current\_user.image\_file)  
 **return** render\_template(**'account.html'**, title=**'Account'**,  
 image\_file=image\_file, form=form, posts=posts,  
 user=user)

Разберем отдельные выражения листинга.

form.username.data = current\_user.username  
form.email.data = current\_user.email

При переходе на страницу профиля в форме будут выведены имя и email текущего пользователя.

page = request.args.get(**'page'**, 1, type=int)

Мы добавим пагинацию, т.е. отображение списка постов текущего пользователя. По умолчанию будем выводить посты начиная с первой страницы.

user = User.query.filter\_by(username=form.username.data).first\_or\_404()

Через ORM из базы данных получаем объект пользователя.

posts = Post.query.filter\_by(author=user).

order\_by(Post.date\_posted.desc()).paginate(page=page, per\_page=5)

Получаем список постов текущего пользователя, упорядочиваем по дате и выводим по пять постов на странице.

image\_file = url\_for(**'static'**, filename=**'profile\_pics/'** +  
 current\_user.image\_file)

Формируем объект для вывода аватарки текущего пользователя..

**return** render\_template(**'account.html'**, title=**'Account'**,  
 image\_file=image\_file, form=form, posts=posts,  
 user=user)

Выполняем рендеринг шаблона, передаем в контекст шаблона заголовок страницы, объект-ссылку с аватаркой пользователя, объект формы с параметрами пользователя, список постов и объект текущего пользователя.

**if** form.validate\_on\_submit():

При успешной валидации объекта формы

**if** form.picture.data:

Если у пользователя есть аватарка,

picture\_file = save\_picture(form.picture.data)  
current\_user.image\_file = picture\_file

Сохраняем ее и привязываем к объекту нашего текущего пользователя в базе данных.

current\_user.username = form.username.data  
current\_user.email = form.email.data

Привязываем имя и email текущего пользователя к объекту нашего текущего пользователя в базе данных.

db.session.commit()  
flash(**'Ваш аккаунт был обновлен!'**, **'success'**)  
**return** redirect(url\_for(**'users.account'**))

Подтверждаем сохранение данных в базе, выполняем редирект на страницу с аккаунтом пользователя.

Теперь добавим шаблон account.html в директорию templates.

А также добавим ссылку на кнопку перехода к профилю пользователя в модуль в шаблон layout.html

**Листинг 16. 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('users.account') }}"**><**i class="fa fa-user"**></**i**>Профиль</**a**>  
  
  
 {% else %}  
 <**a class="nav-link nav-item navigation-links" href="{{**

**url\_for('users.login') }}"**>Войти</**a**>  
 <**a class="nav-link nav-item navigation-links" href="{{**

**url\_for('users.register') }}"**>Зарегистрироваться</**a**>  
 {% endif %}  
</**div**>

Проверим, что у нас получилось. Выполним авторизацию пользователя и перейдем в его профиль.

Результат:

# РЕАЛИЗУЕМ ЗАВЕРШЕНИЕ СЕССИИ ПОЛЬЗОВАТЕЛЯ

Добавим соответствующий контроллер.

**Листинг 17. flask\_blog/users/routes.py**

@users.route(**"/logout"**)  
**def** logout():  
 logout\_user()  
 **return** redirect(url\_for(**'main.home'**))

*Общий механизм работы контроллера.*

*Обеспечивает завершение работы сессии текущего пользователя и перенаправление на главную страницу.*

Контроллер имеет несложный код. Выход из сессии текущего пользователя обеспечивается встроенной функцией logout\_user().

А также добавим ссылку на кнопку завершения работы с пользователем в шаблон layout.html.

**Листинг 18. 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('users.account') }}"**><**i class="fa fa-user"**></**i**>Профиль</**a**>  
 <**a class="nav-item nav-link navigation-links" href="{{**

**url\_for('users.logout') }}"**><**i class="fa fa-sign-out"**></**i**>

Выйти</**a**>  
   
 {% else %}  
 <**a class="nav-link nav-item navigation-links" href="{{**

**url\_for('users.login') }}"**>Войти</**a**>  
 <**a class="nav-link nav-item navigation-links" href="{{**

**url\_for('users.register') }}"**>Зарегистрироваться</**a**>  
 {% endif %}  
</**div**>

Проверим, что у нас получилось. Выполним авторизацию пользователя и сразу выполним завершение работы сессии.



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

Урок 3 мы завершили созданием очередного блюпринта, в котором реализовали функционал подсистемы для работы с пользователями. Теперь в нашем проекте доступна возможность регистрации новых участников, авторизации, завершения сессии, а также возможность просмотра и редактирования содержимого профиля пользователя.

Текущая структура проекта приведена на рис. 1.

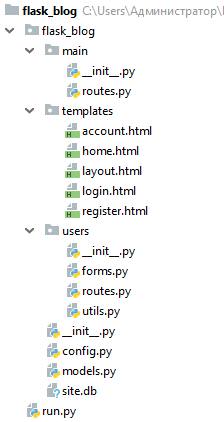


Рис. 1. Структура проекта