

Урок 5. Реализуем механизм восстановления паролей

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc109640488)

[ВВЕДЕНИЕ 3](#_Toc109640489)

[ВЫПОЛНЯЕМ ДОРАБОТКУ ПРОЕКТА 3](#_Toc109640490)

[дорабатываем код моделей 3](#_Toc109640491)

[добавляем новые формы 5](#_Toc109640492)

[добавляем контроллер для запроса на изменение пароля 5](#_Toc109640493)

[добавляем специальную утилиту 6](#_Toc109640494)

[добавляем контроллер для изменения пароля 8](#_Toc109640495)

[подключаем шаблоны 10](#_Toc109640496)

[проверяем работу проекта 10](#_Toc109640497)

[выполняем доработку проекта 11](#_Toc109640498)

[проверяем работу проекта 12](#_Toc109640499)

[МОДИФИЦИРУЕМ СИСТЕМУ ВОССТАНОВЛЕНИЯ ПАРОЛЕЙ 13](#_Toc109640500)

[РЕАЛИЗУЕМ ОТПРАВКУ ПИСЕМ, ИСПОЛЬЗУЯ SMTP-СЕРВЕР YANDEX 17](#_Toc109640501)

[ЗАКЛЮЧЕНИЕ 18](#_Toc109640502)



# ВВЕДЕНИЕ

Базовый функционал нашего блога реализован. Мы можем уверенно добавлять новых пользователей, а также новые записи в блог, выполнять их просмотр, редактирование и удаление. Теперь необходимо реализовать дополнительный, но важный функционал, в частности, реализовать возможность восстановления пароля пользователя и т.д.

# ВЫПОЛНЯЕМ ДОРАБОТКУ ПРОЕКТА

## дорабатываем код моделей

Возможно вы уже слышали о правиле «толстая модель, тонкий контроллер». Это значит, что мы должны размещать бизнес-логику проекта в методах классов-моделей, а не в контроллерах.

Модифицируем модель пользователя User, добавив необходимые методы.

**Листинг 1. flask\_blog/models.py**

…

**from** itsdangerous **import** TimedJSONWebSignatureSerializer **as** Serializer  
**from** flask **import** current\_app

…

**class** User(db.Model, UserMixin):  
 …  
  
 **def** get\_reset\_token(self, expires\_sec=1800):  
 s = Serializer(current\_app.config[**'SECRET\_KEY'**], expires\_sec)  
 **return** s.dumps({**'user\_id'**: self.id}).decode(**'utf-8'**)  
  
 @staticmethod  
 **def** verify\_reset\_token(token):  
 s = Serializer(current\_app.config[**'SECRET\_KEY'**])  
 **try**:  
 user\_id = s.loads(token)[**'user\_id'**]  
 **except** Exception:  
 **return None  
 return** User.query.get(user\_id)

…

В чем идея представленного фрагмента. Когда мы хотим выполнить восстановление пароля, то нужна гарантия, что никто не подделал наш запрос, нужна своего рода подпись.

В данном случае у пользователя есть специальный секретный ключ, опираясь на который, он криптографически подписывает данные и передает их. Когда он получает данные обратно, то будет гарантия, что данные не подделаны.

В процессе подписи происходит генерация токена.

Срок действия токена мы установим в 1800 сек (expires\_sec=1800).

Для генерации токена мы используем библиотеку itsdangerous.

Секретный ключ мы указали в модуле config.py:

SECRET\_KEY = **'5791628bb0b13ce0c676dfde280ba245'**

Далее мы выполняем создание объекта-сериализатора:

s = Serializer(current\_app.config[**'SECRET\_KEY'**], expires\_sec)

И через него выполняем требуемую сериализацию словаря с идентификатором пользователя, чтобы зафиксировать, что запрос поступает именно от него.

Далее мы выполняем непосредственно сериализацию, переводим получившиеся байты в строку и возвращаем получившийся объект:

**return** s.dumps({**'user\_id'**: self.id}).decode(**'utf-8'**)

Во втором методе мы имеем обратную процедуру, когда выполняется десериализация и полученный идентификатор сравнивается с идентификатором пользователя.

## добавляем новые формы

Начнем с первой формы для регистрации нового пользователя.

**Листинг 2. 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'**)

Первая форма необходима для отправки запроса на изменение пароля.

**Листинг 3. 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'**)

Вторая форма обеспечивает непосредственно ввод нового пароля и подтверждение его изменения.

## добавляем контроллер для запроса на изменение пароля

Речь пойдет о контроллере, который позволяет вывести форму и выполнить изменения пароля.

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

**from** flask\_blog.users.utils **import** send\_reset\_email

@users.route(**"/reset\_password"**, methods=[**'GET'**, **'POST'**])  
**def** reset\_request():  
 **if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'posts.allposts'**))  
 form = RequestResetForm()  
 **if** form.validate\_on\_submit():  
 user = User.query.filter\_by(email=form.email.data).first()  
 send\_reset\_email(user)  
 flash(**'На почту отправлено письмо с '  
 'инструкциями по сбросу пароля.'**, **'info'**)  
 **return** redirect(url\_for(**'users.login'**))  
 **return** render\_template(**'reset\_request.html'**,  
 title=**'Сброс пароля'**, form=form)

Разберем отдельные фрагменты кода:

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

Если пользователь авторизован, то сразу обходим процедуру сброса пароля и перенаправляем пользователя на страницу со списком постов.

form = RequestResetForm()  
 **if** form.validate\_on\_submit():  
 user = User.query.filter\_by(email=form.email.data).first()

Создаем объект формы и при успешной валидации определяем объект пользователя на основе содержимого формы и посредством ORM.

send\_reset\_email(user)

Выполняем отправку пользователю письма для изменения пароля. Для этого используем специальную утилиту, информация о которой идет ниже.

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

Выполняем переход на страницу с авторизацией пользователя.

Нам также нужно добавить шаблон reset\_request.html в директорию templates.

## добавляем специальную утилиту

Также добавим еще одну вспомогательную функцию-утилиту. Непосредственно она отвечает за изменение пароля пользователя.

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

**from** flask\_mail **import** Message  
**from** flask\_blog **import** mail

**def** send\_reset\_email(user):  
 token = user.get\_reset\_token()  
 msg = Message(**'Запрос на сброс пароля'**,  
 sender=**'noreply@demo.com'**,  
 recipients=[user.email])  
 msg.body = **f'''Чтобы сбросить пароль,  
 перейдите по следующей ссылке: {**url\_for(**'users.reset\_token'**,  
 token=token, \_external=**True**)**}. Если  
 вы не делали этот запрос  
 тогда просто проигнорируйте это**

**письмо и никаких изменений**

**не будет.'''** mail.send(msg)

Разберем некоторые моменты кода:

**from** flask\_mail **import** Message

Импортируем класс-конструктор для создания объекта сообщения.

**from** flask\_blog **import** mail

А с этим компонентом не совсем все ясно. Похоже, что компонент пользовательский, но не встроенный.

Перейдем в модуль \_\_init\_\_.py пакета проекта и добавим:

**Листинг 6. flask\_blog/\_\_init\_\_.py**

**from** flask\_mail **import** Mail

…

mail = Mail()

…

**def** create\_app(config\_class=Config):  
 …  
 mail.init\_app(app)

Здесь мы работаем с классом-конструктором Mail, импортированным из библиотеки flask\_mail.

Идем далее по методу send\_reset\_email().

token = user.get\_reset\_token()

Запрашиваем токен, который по сути является той самой электронной подписью, которая гарантирует безопасность доставки письма.

Метод get\_reset\_token() мы разобрали выше.

msg = Message(**'Запрос на сброс пароля'**,  
 sender=**'noreply@demo.com'**,  
 recipients=[user.email])

Создаем объект сообщения, в котором определяем email-ы отправителя и получателя.

msg.body = **f'''Чтобы сбросить пароль,  
 перейдите по следующей ссылке: {**url\_for(**'users.reset\_token'**,  
 token=token, \_external=**True**)**}. Если  
 вы не делали этот запрос  
 тогда просто проигнорируйте это письмо  
 и никаких изменений не будет.'''**mail.send(msg)

Определяем тело сообщения и, что очень важно, указываем в нем ссылку, которая обеспечит запуск процедуры переустановки пароля. В конце выполняем отправку сообщения.

## добавляем контроллер для изменения пароля

Теперь добавим в список контроллеров приложения users новый контроллер.

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

@users.route(**"/reset\_password/<token>"**, methods=[**'GET'**, **'POST'**])  
**def** reset\_token(token):  
 **if** current\_user.is\_authenticated:  
 **return** redirect(url\_for(**'posts.allpost'**))  
 user = User.verify\_reset\_token(token)  
 **if** user **is None**:  
 flash(**'Это недействительный или просроченный токен'**, **'warning'**)  
 **return** redirect(url\_for(**'users.reset\_request'**))  
 form = ResetPasswordForm()  
 **if** form.validate\_on\_submit():  
 hashed\_password = bcrypt.\  
 generate\_password\_hash(form.password.data).decode(**'utf-8'**)  
 user.password = hashed\_password  
 db.session.commit()  
 flash(**'Ваш пароль был обновлен! '  
 'Теперь вы можете авторизоваться'**, **'success'**)  
 **return** redirect(url\_for(**'users.login'**))  
 **return** render\_template(**'reset\_token.html'**,  
 title=**'Сброс пароля'**, form=form)

Разберем отдельные выражения контроллера.

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

Если пользователь авторизован, пользователь будет перенаправлен на страницу со списком постов.

user = User.verify\_reset\_token(token)

Через переданный токен выполняем десериализацию того самого ключа, который мы использовали для подписи нашего запроса на изменение пароля.

**if** user **is None**:  
 flash(**'Это недействительный или просроченный токен'**, **'warning'**)  
 **return** redirect(url\_for(**'users.reset\_request'**))

Если срок действия токена (1800 сек) истек, то мы будем перенаправлены на повторный запрос изменения пароля.

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

Создаем объект формы для изменения пароля и выполняем валидацию формы.

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

Получаем пароль из формы и выполняем его хеширование, изменяем значение соответствующего поля строки таблицы базы данных и подтверждаем изменения.

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

Переходим на страницу с авторизацией.

## подключаем шаблоны

Нужно добавить шаблоны для страницы запроса восстановления пароля reset\_request.htmlи страницы непосредственно изменения пароля reset\_token.html.

Далее нужно добавить ссылку на кнопку с восстановлением пароля на страницу, через которую происходит авторизация пользователя.

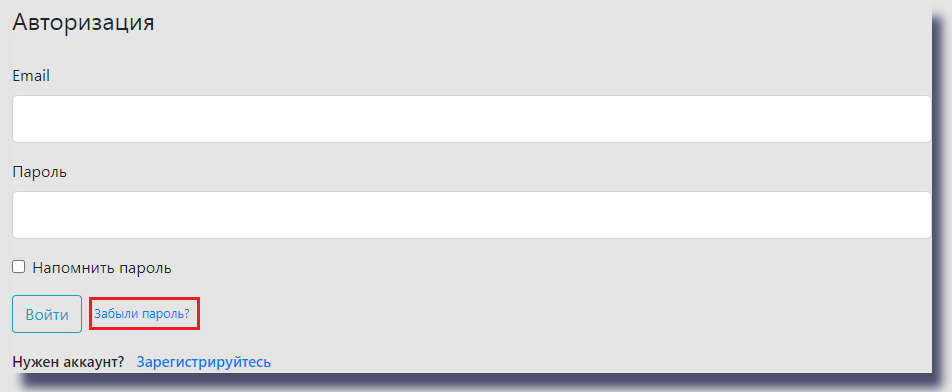
**Листинг 8. flask\_blog/templates/login.html**

<**div class="form-group"**>  
 {{ form.submit(class="btn btn-outline-info") }}  
 <**small class="text-muted ml-2"**>  
 <**a href="{{ url\_for('users.reset\_request') }}"**>Forgot Password?</**a**>  
 </**small**>

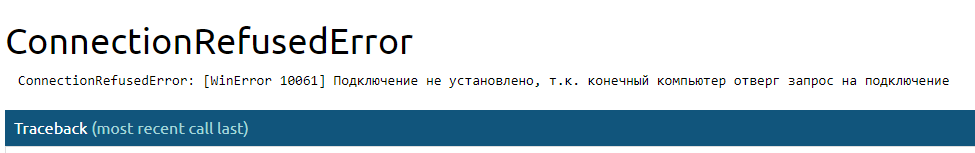
…

## проверяем работу проекта

Перейдем на страницу авторизации.

Перейдем на страницу запроса на изменение пароля.

Но если мы введем email-адрес и попытаемся отправить запрос на изменение пароля, получим такой ответ:



## выполняем доработку проекта

Перейдем в файл config.py проекта и пропишем в нем важные настройки.

**Листинг 9. flask\_blog/config.py**

**class** Config:  
 …MAIL\_SERVER = **'smtp.googlemail.com'** MAIL\_PORT = 587  
 MAIL\_USE\_TLS = **True** MAIL\_USERNAME = **"ваш e-mail"** MAIL\_PASSWORD = **"пароль"**

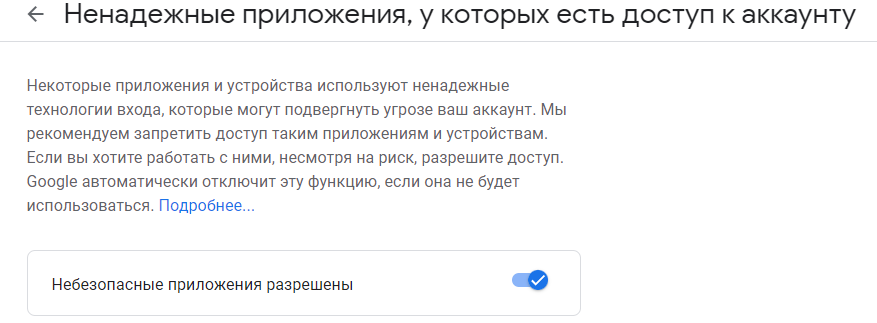
Именно от этого отправителя пользователь будет получать письма о восстановлении пароля.

Для корректной работы схемы вам нужно в Google разрешить доступ со стороны всех приложений.

Нужно перейти по ссылке:

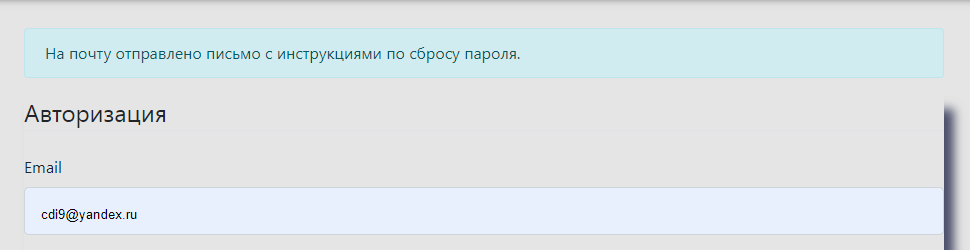
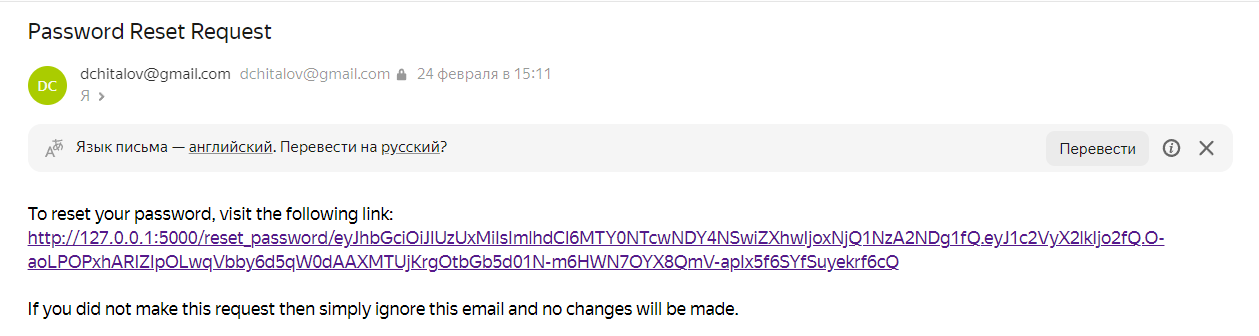
<https://myaccount.google.com/lesssecureapps?pli=1&rapt=AEjHL4PvUCH9kGZrtTmFiU8TpeQXAjx-2VZT5vmztdpiCWHXMS_l5_lE0p-VW7MLEl2UsGJ08ZKQILZOm_NdVrxCE_dS-RbWnQ>

И включить доступ со стороны небезопасных приложений.



## проверяем работу проекта

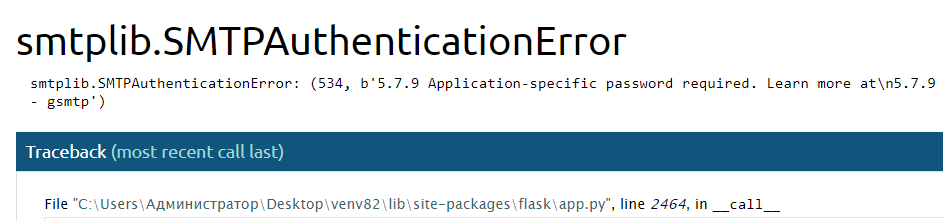
При запросе на изменение пароля мы получим сообщение подобного формата:

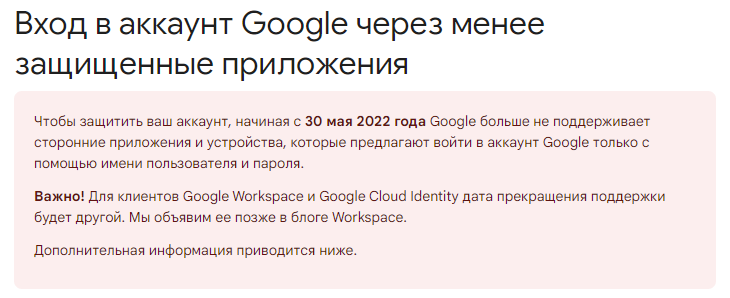
На электронный ящик пользователя, запросившего изменение пароля, придет такое письмо:

Переход по ссылке обеспечить переход на форму изменения пароля:



# МОДИФИЦИРУЕМ СИСТЕМУ ВОССТАНОВЛЕНИЯ ПАРОЛЕЙ

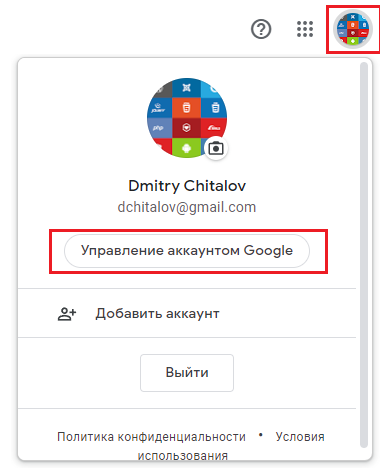
Если вы начали прохождение курса совсем недавно, то возможно при попытке реализовать отправку сообщений для восстановления паролей через SMTP-сервер GMAIL, столкнулись со следующей ошибкой:

Но в чем дело? Что об этом говорит справочная служба Google?

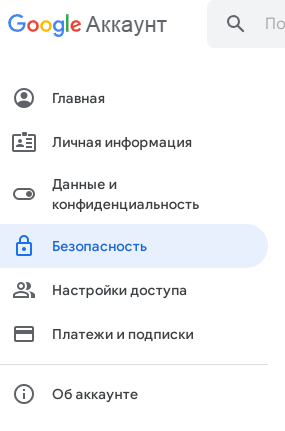
Все просто, Gmail запретил SMTP-авторизацию по паролю с 30 мая 2022 г., и мы не можем отправлять письма пользователям с своего почтового ящика Gmail.

Но решение есть. Необходимо использовать пароль приложения.

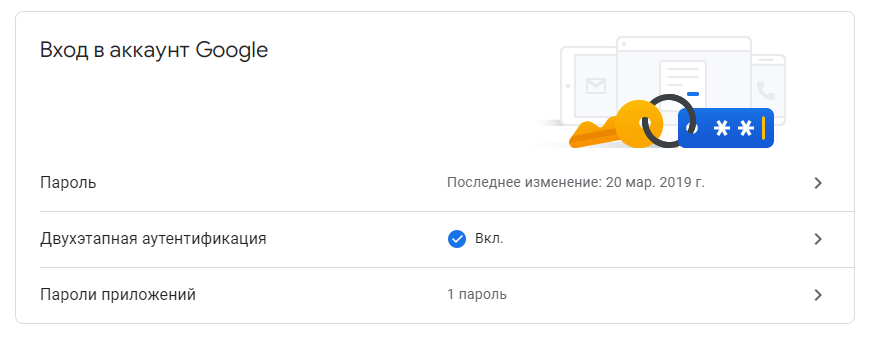
Перейдем в раздел управления аккаунтом Google:



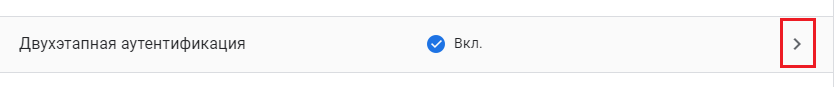
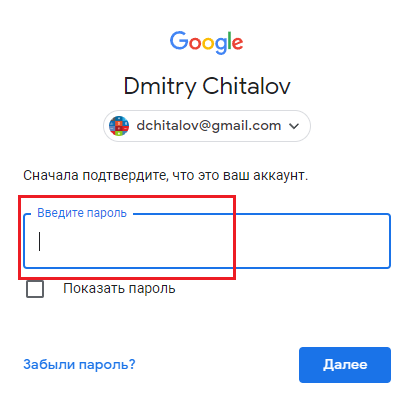
Далее в раздел «Безопасность»:

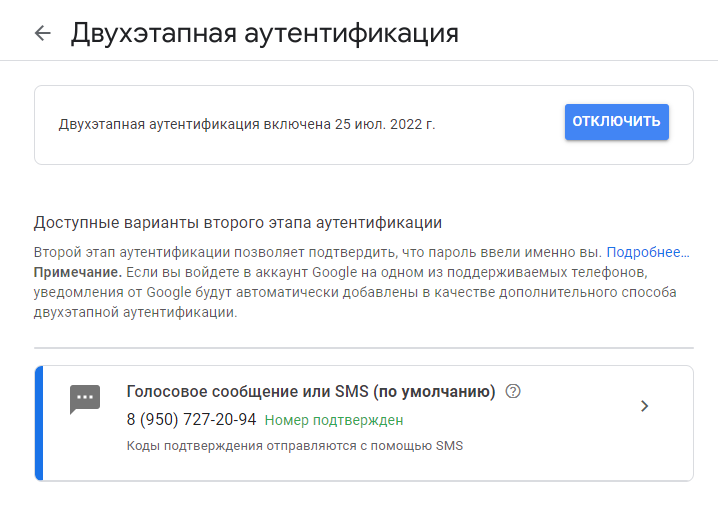


Нам нужно включить режим двухэтапной аутентификации и создать пароль приложения.

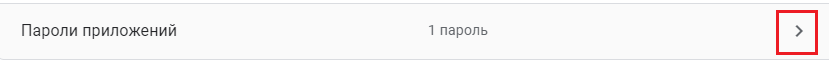


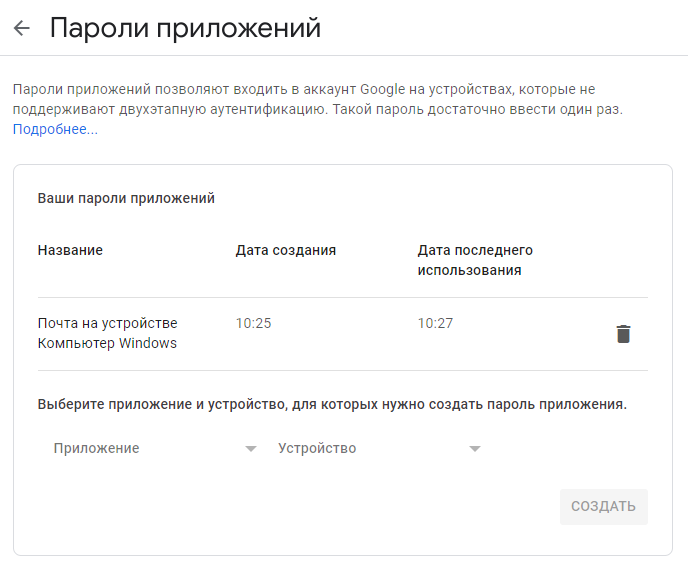
Начинаем с включения двухэтапной аутентификации:

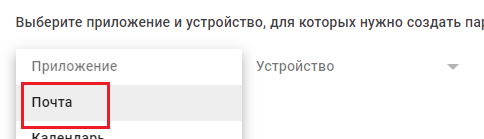


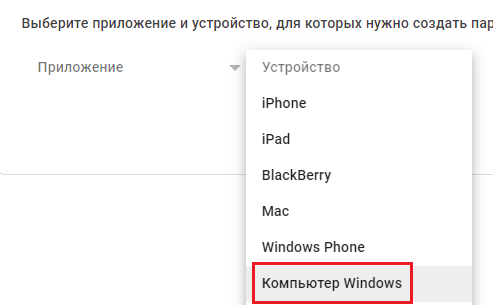
Включаем двухэтапную аутентификацию и подтверждаем номер телефона через SMS-код:

Теперь переходим к созданию паролей приложений:

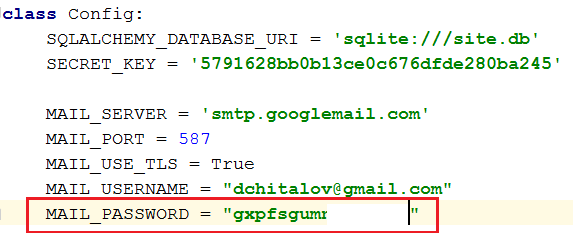




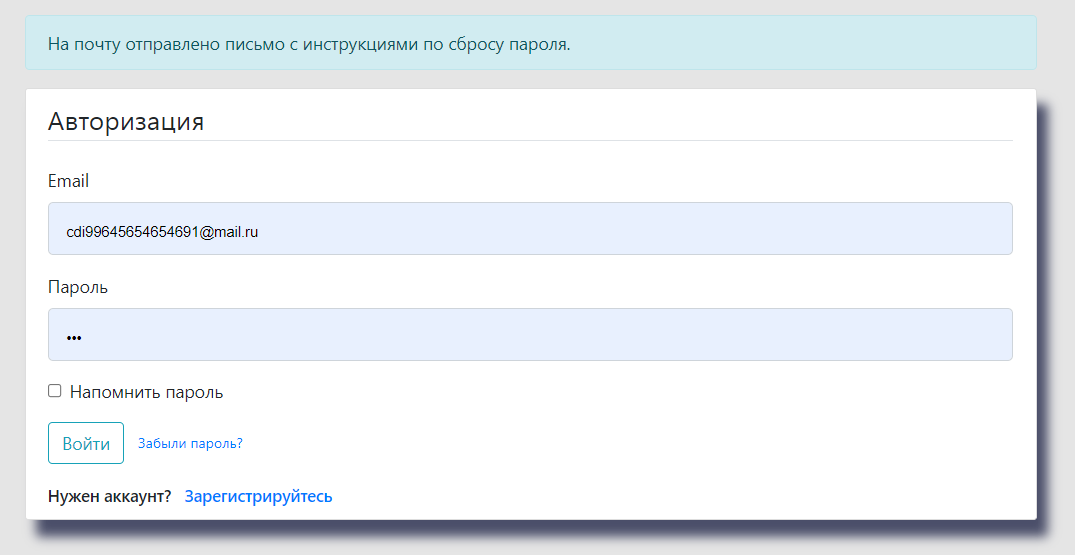




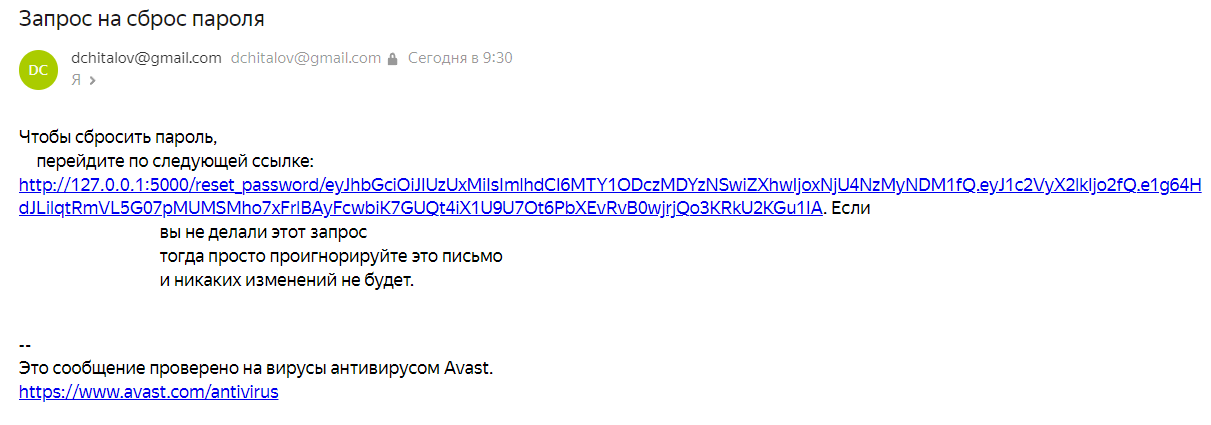
Теперь создаем сам пароль и сохраняем его в отдельном файле. Пароль выглядит примерно так: «gcpfsmgmnikikpkr». Его и нужно записать в модуль config.py.



Теперь, если мы захотим восстановить пароль и отправить письмо с ящика Gmail, ошибок больше возникать не будет.

Проверим, что получилось.

А вот и письмо:



# РЕАЛИЗУЕМ ОТПРАВКУ ПИСЕМ, ИСПОЛЬЗУЯ SMTP-СЕРВЕР YANDEX

Вы можете протестировать и этот способ. Он незначительно отличается от предыдущего. Изменятся лишь настройки в модуле config.py. Пример такого модуля:

**Листинг 10. flask\_blog/config.py**

**class Config:**

**SQLALCHEMY\_DATABASE\_URI = 'sqlite:///site.db'**

**SECRET\_KEY = '5791628bb0b13ce0c676dfde280ba245'**

**MAIL\_SERVER = 'smtp.yandex.ru'**

**MAIL\_PORT = 465**

**MAIL\_USE\_TLS = False**

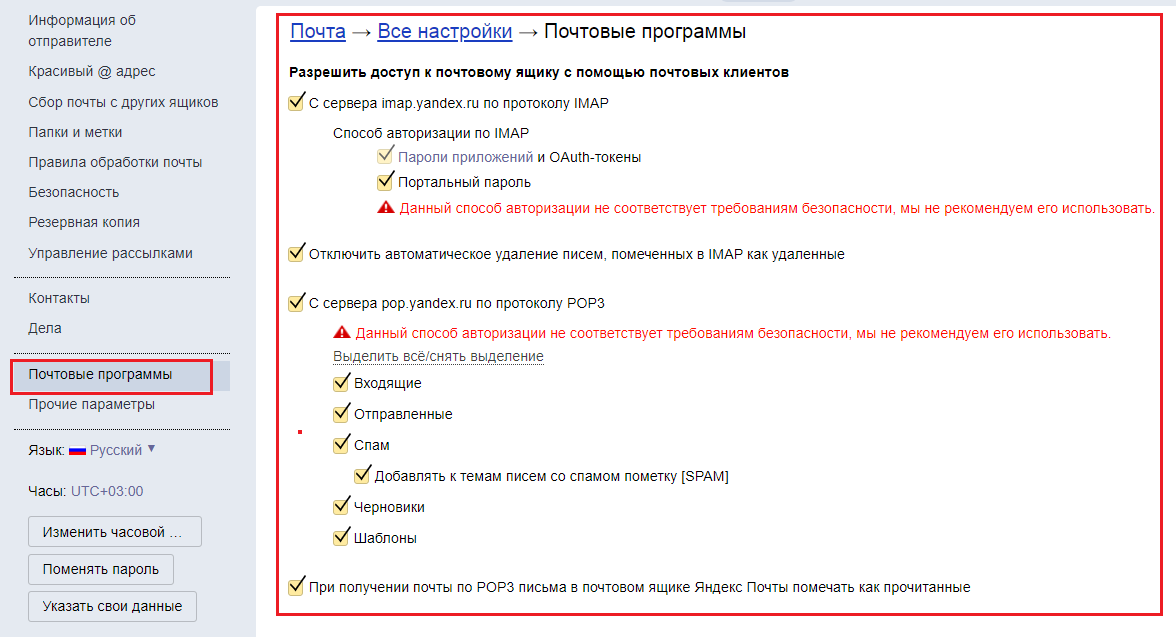
**MAIL\_USE\_SSL = True**

**MAIL\_USERNAME = "ramlir5201@yandex.ru"**

**MAIL\_PASSWORD = "t876543wqaz"**

Для своего проекта вам нужно поменять параметры: MAIL\_USERNAME и MAIL\_PASSWORD.

Далее нужно выполнить настройки почты:



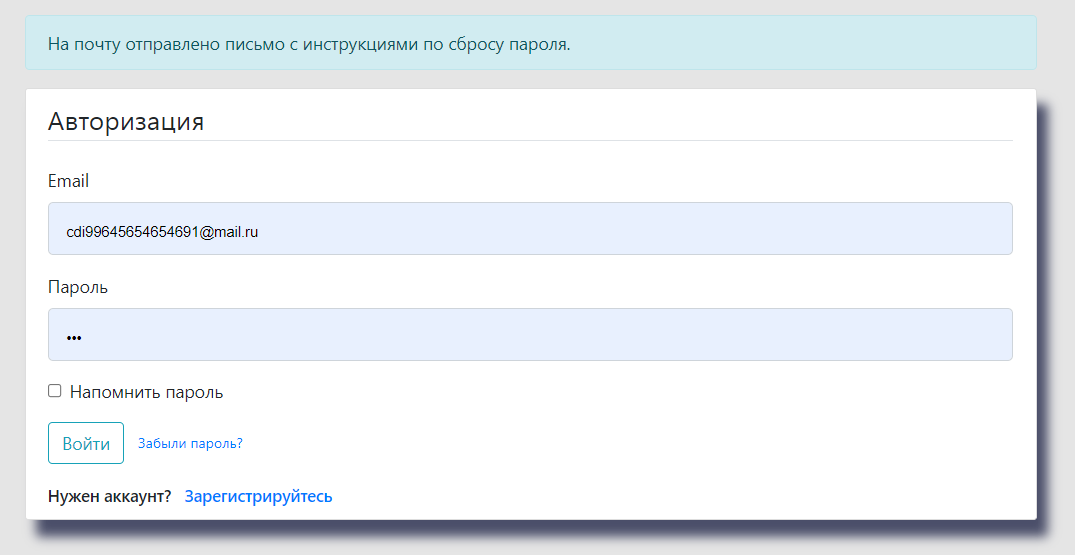
И еще один штрих, нужно менять адрес, с которого нам приходят письма при формировании сообщения:

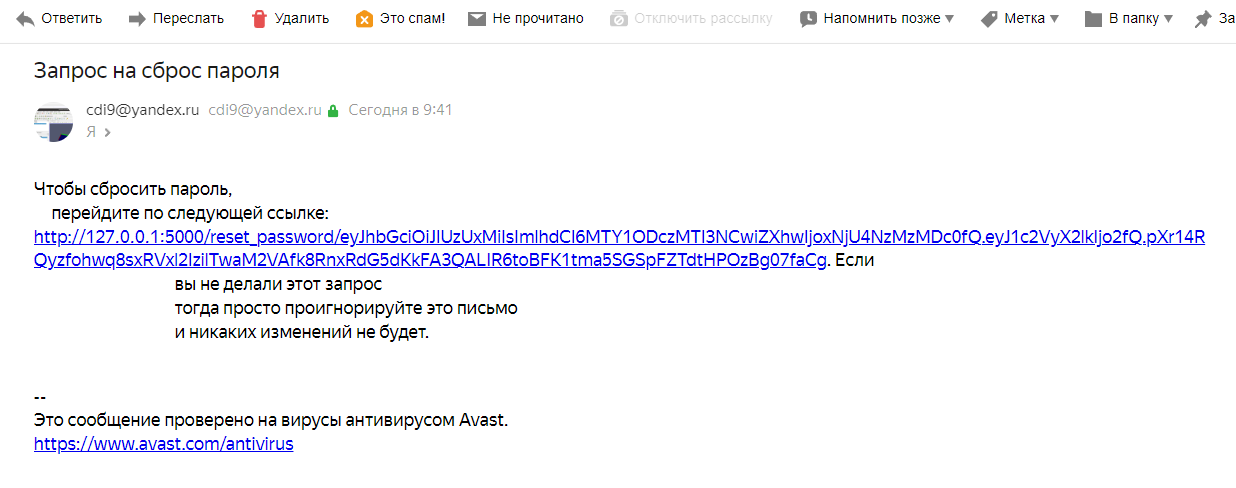
**Листинг 11. flask\_blog/users/utils.py**

**def** send\_reset\_email(user):  
 token = user.get\_reset\_token()  
 msg = Message(**'Запрос на сброс пароля'**,  
 sender=**'ramlir5201@yandex.ru'**,  
 recipients=[user.email])  
 msg.body = **f'''Чтобы сбросить пароль,  
 перейдите по следующей ссылке: {**url\_for(**'users.reset\_token'**,  
 token=token, \_external=**True**)**}. Если  
 вы не делали этот запрос  
 тогда просто проигнорируйте это**

**письмо и никаких изменений не**

**будет.'''** mail.send(msg)

Проверим, что получилось.

А вот и письмо:

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

По итогам урока 5 мы реализовали возможность восстановления пароля пользователя. Код проекта был существенно доработан, в том числе в части моделей, контроллеров и дополнительных утилит. Мы выполнили тестовый запуск, выявили и устранили возникшие ошибки и обеспечили приложению работоспособное состояние.

Текущая структура проекта приведена на рис. 1.

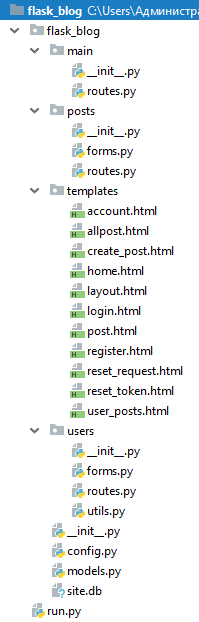


Рис. 1. Структура проекта