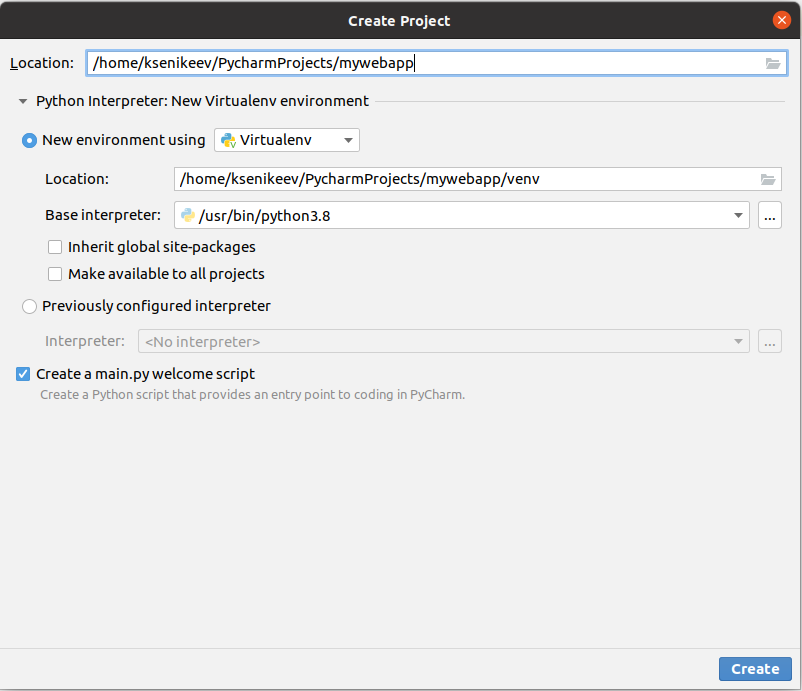
**Пример разработки web-приложения с использованием фреймворка Flask**

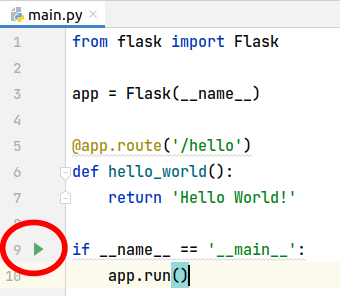
Мы будем пользоваться средой разработки PyCharm, и нам хватит бесплатной версии Community.

Для начала — новый проект:



Заменим (или создадим) содержимое файла main.py

from flask import Flask  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/')  
def hello\_world():  
 return 'Hello World!'  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

Теперь можно попробовать запустить это простейшее web-приложение:  
В логах приложения мы увидим:

\* Serving Flask app 'main' (lazy loading)

\* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

\* Debug mode: off

\* Running on **http://127.0.0.1:5000** (Press CTRL+C to quit)

127.0.0.1 - - [19/May/2022 13:34:05] "GET / HTTP/1.1" 200 -

Что мы сделали:

1) app = Flask(\_\_name\_\_) - инициировали библиотеку

2) @app.route('/') - создали функцию с декоратором, указывающим путь который будет обрабатывать наша функция

3) app.run() - запускаем приложение

4) приложение доступно по адресу **http://127.0.0.1:5000**

**Добавим в проект контроль версий** (будем использовать **git**):

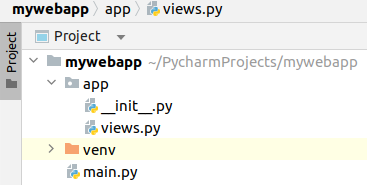
через интерфейс: в меню VCS — Enable version control Integration …

через коммандную строку в директории проекта: git init

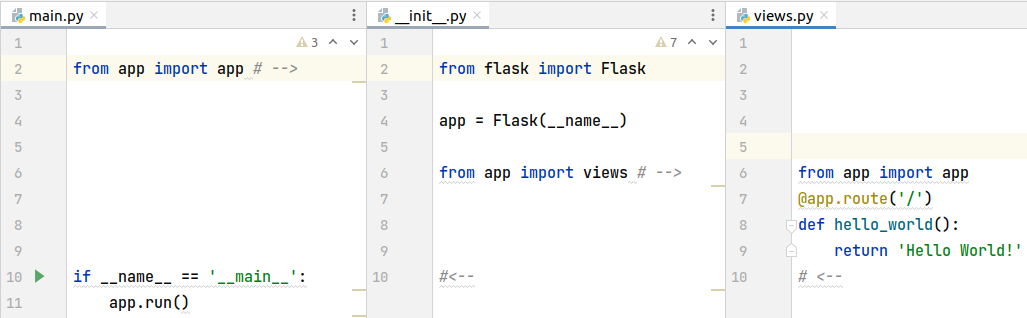


**Переструктурируем проект** для болеее удобной разработки в дальнейшем.

Создадим пакет ***app*** где будем размещать пакеты, коды, ресурсы

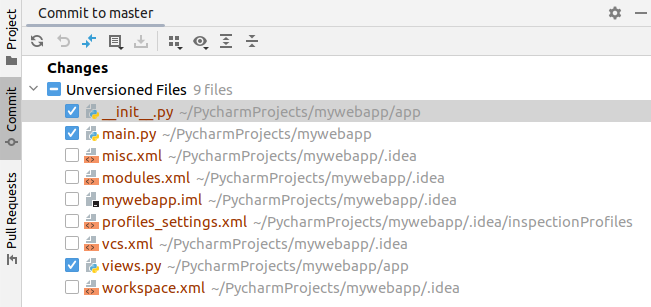


Код распределим по файлам, выделив функциональные блоки (запуск, инициализация, маршрутизация):



Здесь же виден порядок исполнения скрипта, стрелки (-->) изображают переход процесса исполнения от одного к другому файлу.

Мы провели изменения в проекте. Имеет смысл зафиксировать эти изменения в git.



При работе в команде не стоит фиксировать в общем репозитории файлы настроек IDE (у каждого участника проекта они будут своими).

Для отдачи клиенту полноценной html страницы удобно пользоваться шаблоном с набором переменных в которые мы на сервере подставим определенные значения.

В файле views.py отвечающему за маршрутизацию запросов импортируем модуль render\_template

В его задачи будет входить обработка шаблонов html-страниц. Сами шаблоны будем хранить в директории ***app/templates***.

Добавим в проект файл **index.html**

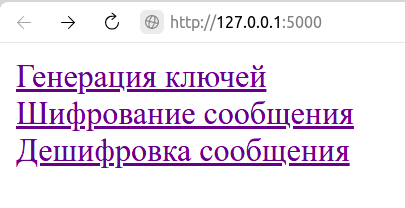
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
 <meta charset="utf-8"/>  
 <title>RSA calculator</title>  
</head>  
<body>  
 <div>  
 <a href="/keygen">Генерация ключей</a>  
 </div>  
  
 <div>  
 <a href="/sign">Шифрование сообщения</a>  
 </div>  
  
 <div>  
 <a href="/unsign">Дешифровка сообщения</a>  
 </div>  
</body>  
</html>

Далее нам надо вывести это при запросе корневого ресурса "/".

Редактируем views.py:

from app import app  
from flask import render\_template  
  
@app.route('/')  
def index():  
 return render\_template('index.html')

Попробуем запустить приложение и зайти на нашу страничку:



Ссылки есть, но они пока не работают — необходимо прописать функции с соответствующими шаблонами, и подготовить сами шаблоны с формами.

**views.py:**

from app import app  
from flask import render\_template  
  
@app.route('/')  
def index():  
 return render\_template('index.html')  
  
@app.route('/keygen')  
def keygen():  
 return render\_template('keygen.html')  
  
@app.route('/encode')  
def sign():  
 return render\_template('encode.html')  
  
@app.route('/decode')  
def unsign():  
 return render\_template('decode.html')

**keygen.html:**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
 <meta charset="utf-8"/>  
 <link rel='stylesheet' href='/static/css/style.css'>  
 <title>RSA calculator</title>  
</head>  
<body>  
 <h1>Генерация ключей</h1>  
  
 <div>  
 <form action="/startkeygen" method="post">  
 <div>  
 p:<input name="number\_p" type="text"/>  
 </div>  
 <div>  
 q:<input name="number\_q" type="text"/>  
 </div>  
  
 <input type="submit" value="Сгенерировать ключи"/>  
 </form>  
 </div>  
 <br/>  
 {% if privatekey %}  
 <div>  
 <h2>Секретный ключ {{ privatekey }}</h2>  
 </div>  
 {% endif %}  
  
 {% if publickey %}  
 <div>  
 <h2>Открытый ключ {{ publickey }}</h2>  
 </div>  
 {% endif %}  
  
 {% if error %}  
 <div>  
 <h2>Ошибка: <span class="errormsg">{{ error }}</span></h2>  
 </div>  
 {% endif %}  
  
 <div>  
 <a href="/">Вернуться на главную</a>  
 </div>  
</body>  
</html>

Форма (<**form** action="**/****startkeygen**" method="post">) будет отправляться на ресурс "**/startkeygen**". Нам надо его подготовить — создать процедуру генерации ключей и вернуть их на страницу **keygen.html** через параметры шаблона privatekey и publickey. Такой функционал лучше реализовать в отдельном файле **keygen.py** (ну и сослаться на него через import в файле **views.py:** from app import keygen):

from app import app  
from flask import render\_template  
from flask import request  
  
@app.route('/startkeygen', methods=['POST'])  
def startkeygen():

#объявляем переменные  
 public\_key = None  
 private\_key = None  
 error = None

try: # дальше возможны исключения  
 p = int(request.form.get('number\_p')) #преобразуем текст из формы в целое  
 q = int(request.form.get('number\_q')) #преобразуем текст из формы в целое  
 n = p \* q  
 phi = (p - 1) \* (q - 1)  
 e = 3  
 while (phi % e == 0):  
 e = e + 1  
 public\_key = [e, n]  
 d = 1  
 while ((d \* e) % phi != 1):  
 d = d + 1  
 private\_key = [d, n]  
 except ValueError:  
 error='Неверное значение параметра!'  
 return render\_template('keygen.html', publickey=public\_key, privatekey=private\_key, error=error)

Что бы сообщение об ошибке выделялось на странице красным цветом, мы стилизовали его (class="errormsg"), а сам класс стиля описали в статическом файле **style.css**:

.errormsg {  
 color: #FF0000;  
}  
  
input {  
 margin-top: 5pt;  
}

Доступ к статическим ресурсам, по умолчанию, осуществляется из директории **static**, в нее мы и положили наш файл со стилями (точнее, для удобства во вложенную директорию **css**).

Далее, по плану, мы должны реализовать страничку с интерфейсом для формирования подписи и дешифровки.

**encode.html**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
 <meta charset="utf-8"/>  
 <link rel='stylesheet' href='/static/css/style.css'>  
 <title>RSA calculator</title>  
</head>  
<body>  
 <h1>Шифрование сообщения:</h1>  
  
 <div>  
 <form action="/encodemsg" method="post">  
 <div>Параметры открытого ключа:</div>  
 <div>  
 e:<input name="param\_e" type="text"/> &nbsp; n:<input name="param\_n" type="text"/>  
 </div>  
 <div>  
 Сообщение (целое число < n):<input name="message" type="text"/>  
 </div>  
  
 <input type="submit" value="Зашифровать"/>  
 </form>  
 </div>  
 <br/>  
 {% if signed %}  
 <div>  
 <h2>Шифровка: {{ signed }}</h2>  
 </div>  
 {% endif %}  
  
 {% if error %}  
 <div>  
 <h2>Ошибка: <span class="errormsg">{{ error }}</span></h2>  
 </div>  
 {% endif %}  
  
 <div>  
 <a href="/">Вернуться на главную</a>  
 </div>  
</body>  
</html>

Реализация процедуры шифрования в методе, отрабатывающим запрос на ресурс (маршрут) '/encodemsg'

**encode.py:**

from app import app  
from flask import render\_template  
from flask import request  
  
@app.route('/encodemsg', methods=['POST'])  
def startkeygen():  
 error = None  
 c = None  
 try:  
 e = int(request.form.get('param\_e'))  
 n = int(request.form.get('param\_n'))  
 m = int(request.form.get('message'))  
  
 c = pow(m, e, n)  
  
 except ValueError:  
 error='Неверное значение параметра!'  
 return render\_template('encode.html', encoded=c, error=error)

В файл views.py необходимо добавить импорт (from app import encode)

Дешифровка, шаблон странички — **decode.html**:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
 <meta charset="utf-8"/>  
 <link rel='stylesheet' href='/static/css/style.css'>  
 <title>RSA calculator</title>  
</head>  
<body>  
 <h1>Дешифровка сообщения:</h1>  
  
 <div>  
 <form action="/decodemsg" method="post">  
 <div>Параметры закрытого ключа:</div>  
 <div>  
 d:<input name="param\_d" type="text"/> &nbsp; n:<input name="param\_n" type="text"/>  
 </div>  
 <div>  
 Шифровка:<input name="encoded" type="text"/>  
 </div>  
  
 <input type="submit" value="Дешифровать"/>  
 </form>  
 </div>  
 <br/>  
 {% if decoded %}  
 <div>  
 <h2>Сообщение: {{ decoded }}</h2>  
 </div>  
 {% endif %}  
  
 {% if error %}  
 <div>  
 <h2>Ошибка: <span class="errormsg">{{ error }}</span></h2>  
 </div>  
 {% endif %}  
  
 <div>  
 <a href="/">Вернуться на главную</a>  
 </div>  
</body>  
</html>

Реализация алгоритма дешифровки — **decode.py**:

from app import app  
from flask import render\_template  
from flask import request  
  
@app.route('/decodemsg', methods=['POST'])  
def decodemsg():  
 error = None  
 m = None  
 try:  
 d = int(request.form.get('param\_d'))  
 n = int(request.form.get('param\_n'))  
 c = int(request.form.get('encoded'))  
  
 m = pow(c, d, n)  
  
 except ValueError:  
 error='Неверное значение параметра!'  
 return render\_template('decode.html', decoded=m, error=error)

Прототип программы готов, можно протестировать и зафиксировать очередной commit.

Почему прототип? Остались нерешенными вопросы:

1. Проверка на простоту входных данных p и q.

2. Шифруется только число, а хотелось бы текст, и вообще — любые бинарные данные.