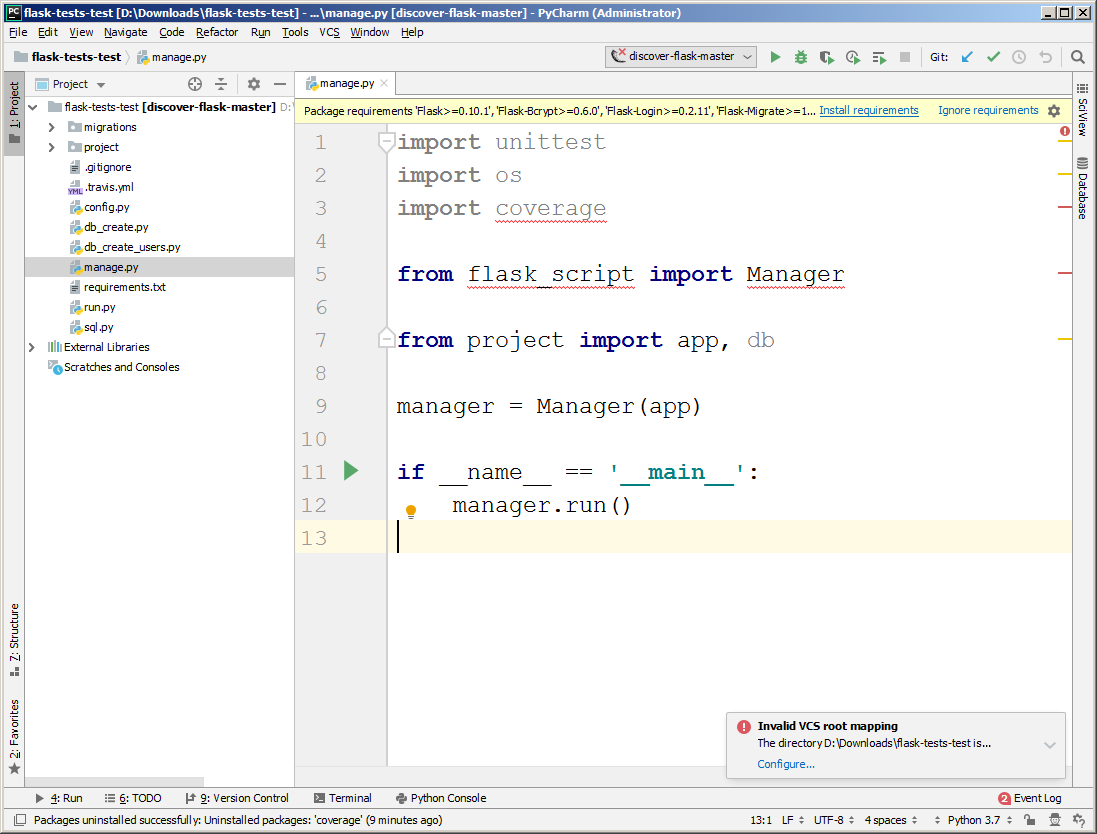
1. Скопируйте, распакуйте и откройте репозиторий:

**https://github.com/Burmakova/flask-tests**

2. Откройте любой Python file и установите недостающие зависимости:



3. Выполните создание и наполнение БД командами в терминале:

**python db\_create.py**

**python db\_create\_users.py**

4. Запустите проект, познакомьтесь с ним – перейдите по ссылке http://127.0.0.1:5000/, зарегистрируйтесь, оставьте пост с HTML-разметкой (картинкой и ссылкой с другого сайта), выйдите из системы, убедитесь, что форма для размещения постов исчезла, а Ваш пост – не исчез, войдите в систему, убедитесь, что форма для размещения постов стала снова доступна. Остановите проект и снова запустите его, убедитесь, что после отключения посты не исчезли, потому что были сохранены в БД.

5. Для создания тестов в конце файла **config.py** допишите тестовую конфигурацию запуска проекта, которая будет иметь упрощенную безопасность для тестовых запросов и хранить БД в оперативной памяти для обеспечения независимости тестов друг от друга – для каждого теста будет создаваться новая база и после теста – удаляться:

# тестовая конфигурация, используемая при запуске тестов,

# с адресом БД в памяти

class TestConfig(BaseConfig):

DEBUG = True

TESTING = True

WTF\_CSRF\_ENABLED = False

SQLALCHEMY\_DATABASE\_URI = 'sqlite:///:memory:'

6. Создйте в корне проекта папку **tests** и в ней – Python file с именем **base.py** для создания базового класса для будущих наборов тестов, который будет отвечать за запуск проекта в созданной тестовой конфигурации и за создание таблиц в БД при запуске:

from flask\_testing import TestCase

from project import app, db

from project.models import User, BlogPost

# базовый класс для наборов тестов -

# содержит методы запуска веб-приложения,

# создания БД в памяти и

# добавления пользователя и поста

class BaseTestCase(TestCase):

def create\_app(self):

app.config.from\_object('config.TestConfig')

return app

def setUp(self):

db.create\_all()

db.session.add(User("admin", "ad@min.com", "admin"))

db.session.add(

BlogPost("Test post",

"This is a test. Only a test.",

"admin"

)

)

db.session.commit()

def tearDown(self):

db.session.remove()

db.drop\_all()

7. Создйте в папке **tests** Python file с типом Python unit test и именем **test\_registration.py** для тестирования регистрации пользователей:

import unittest

from flask\_login import current\_user

from flask import request

from tests.base import BaseTestCase

from project.models import User

class RegistrationTests(BaseTestCase):

# проверка регистрации пользователя и

# входа в систему сразу при регистрации

def test\_user\_registeration(self):

with self.client:

response = self.client.post('register/', data=dict(

username='Michael', email='michael@realpython.com',

password='python', confirm='python'

), follow\_redirects=True)

self.assertNotIn(b'Sign in', response.data)

self.assertNotIn(b'Register', response.data)

self.assertIn(b'logout', response.data)

self.assertTrue(current\_user.name == "Michael")

self.assertTrue(current\_user.is\_active())

user = User.query.filter\_by(

email='michael@realpython.com'

).first()

self.assertTrue(

str(user) == '<name - Michael, anon - False>'

)

# проверка обнаружения неправильного e-mail при регистрации

def test\_wrong\_email\_user\_registeration(self):

with self.client:

response = self.client.post('register/', data=dict(

username='Michael', email='michael',

password='python', confirm='python'

), follow\_redirects=True)

self.assertIn(b'Invalid email address.', response.data)

self.assertIn('/register', request.url)

# проверка обнаружения попытки регистрации

# пользователя с существующим логином

def test\_existing\_login\_user\_registeration(self):

with self.client:

response = self.client.post('register/', data=dict(

username='admin', email='ad@min.com',

password='admin', confirm='admin'

), follow\_redirects=True)

self.assertIn('/register', request.url)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

8. В файле **manage.py** допишите команды для запуска всех тестов и расчета покрытия ими кода:

import unittest

import os

import coverage

from flask\_script import Manager

from project import app, db

manager = Manager(app)

@manager.command

def test():

"""Запуск тестов."""

tests = unittest.TestLoader().discover('tests')

unittest.TextTestRunner(verbosity=2).run(tests)

@manager.command

def cov():

"""Запуск тестов и расчет покрытия кода."""

cov = coverage.coverage(

branch=True,

include='project/\*'

)

cov.start()

"""Запуск тестов."""

tests = unittest.TestLoader().discover('tests')

unittest.TextTestRunner(verbosity=2).run(tests)

cov.stop()

cov.save()

# вывод оценки покрытия на консоль

print('Coverage Summary:')

cov.report()

# создание пути к папке coverage внутри проекта

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

covdir = os.path.join(basedir, 'coverage')

# вывод HTML-отчета о покрытии в папку coverage внутри проекта

cov.html\_report(directory=covdir)

print('Saved to: ' + covdir)

cov.erase()

if \_\_name\_\_ == '\_\_main\_\_':

manager.run()

9. Запустите созданные команды в терминале:

**python manage.py test**

**python manage.py cov**

10. В файловом менеджере откройте в папке **coverage** отчет **index.html**, откройте статистику по отдельным файлам кода, посмотрите, какие строки были протестированы, а какие – нет.

11. В папке **tests** создайте файл **test\_login.py** для тестирования входа в систему:

import unittest

from flask\_login import current\_user

from project import User, bcrypt

from tests.base import BaseTestCase

# набор тестов для проверки входа в систему

class LoginTestCase(BaseTestCase):

# тест - главная страница открывается нормально (код 200)

def test\_index(self):

response = self.client.get('/', content\_type='html/text')

self.assertEqual(response.status\_code, 200)

# проверка наличия кнопок входа и регистрации,

# и отсутствия кнопки выхода до входа в систему

def test\_page\_content\_before\_login(self):

response = self.client.get('/', follow\_redirects=True)

self.assertIn(b'Sign in', response.data)

self.assertIn(b'Register', response.data)

self.assertNotIn(b'logout', response.data)

# проверка отсутствия кнопок входа и регистрации,

# и наличия кнопки выхода после входа из системы

def test\_page\_content\_after\_login(self):

response = self.client.post(

'/login',

data=dict(username="admin", password="admin"),

follow\_redirects=True

)

self.assertNotIn(b'Sign in', response.data)

self.assertNotIn(b'Register', response.data)

self.assertIn(b'logout', response.data)

# проверка наличия заголовка на странице входа

def test\_login\_page\_loads(self):

response = self.client.get('/login')

self.assertIn(b'Please login', response.data)

# проверка по флагу is\_active сохранения в сессии

# сведений о вошедшем пользователе

def test\_correct\_login(self):

with self.client:

response = self.client.post(

'/login',

data=dict(username="admin", password="admin"),

follow\_redirects=True

)

self.assertTrue(current\_user.name == "admin")

self.assertTrue(current\_user.is\_active)

# проверка по флагу is\_active, что после неудачной попытки входа

# пользователь остается гостем

def test\_incorrect\_login(self):

response = self.client.post(

'/login',

data=dict(username="wrong", password="wrong"),

follow\_redirects=True

)

self.assertFalse(current\_user.is\_active)

# проверка по id успешного входа пользователя

def test\_get\_admin\_by\_id(self):

with self.client:

try:

self.client.post('/login', data=dict(

username="admin", password='admin'

), follow\_redirects=True)

self.assertTrue(current\_user.id == 1)

self.assertFalse(current\_user.id == 20)

except:

self.assertFalse(True)

# проверка обнаружения ошибки при входе с

# несуществующими учетными данными по id

# (id не существует до входа)

def test\_get\_anon\_by\_id(self):

with self.client:

try:

self.client.post('/login', data=dict(

username="Michael", password='python'

), follow\_redirects=True)

self.assertTrue(current\_user.id == 2)

self.assertFalse(current\_user.id == 20)

except:

self.assertFalse(False)

# проверка корректного хэширования

# пароля пользователя

def test\_check\_password(self):

user = User.query.filter\_by(

email='ad@min.com'

).first()

self.assertTrue(

bcrypt.check\_password\_hash(

user.password, 'admin'

)

)

self.assertFalse(

bcrypt.check\_password\_hash(

user.password, 'foobar'

)

)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

12. В папке **tests** создайте файл **test\_logout.py** для тестирования выхода из системы:

import unittest

from flask\_login import current\_user

from tests.base import BaseTestCase

# набор тестов для выхода из системы

class LogoutTests(BaseTestCase):

# тест - после входа и выхода из системы

# пользователь становится неактивным, гостем

def test\_logout(self):

with self.client:

self.client.post(

'/login',

data=dict(username="admin", password="admin"),

follow\_redirects=True

)

response = self.client.get('/logout', follow\_redirects=True)

self.assertFalse(current\_user.is\_active)

# тест - при попытке выхода из системы без входа

# отображается сообщение об ошибке

def test\_logout\_route\_requires\_login(self):

response = self.client.get('/logout', follow\_redirects=True)

self.assertIn(b'Please log in to access this page', response.data)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

13. В папке **tests** создайте файл **test\_post.py** для тестирования отправки сообщений:

import unittest

from tests.base import BaseTestCase

# набор тестов для отправки сообщений

class BlogPostTests(BaseTestCase):

# проверка возможности отправки

# постов пользователем

def test\_user\_can\_post(self):

with self.client:

self.client.post(

'/login',

data=dict(username="admin", password="admin"),

follow\_redirects=True

)

response = self.client.post(

'/',

data=dict(title="test", description="test"),

follow\_redirects=True

)

self.assertEqual(response.status\_code, 200)

# проверка, что гость не может отправлять посты

def test\_anon\_can\_post(self):

with self.client:

response = self.client.post(

'/',

data=dict(title="test", description="test"),

follow\_redirects=True

)

self.assertEqual(response.status\_code, 401)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

14. Запустите тестирование с подсчетом покрытия кода в терминале:

**python manage.py cov**

15. В файловом менеджере откройте в папке **coverage** отчет **index.html**, откройте статистику по отдельным файлам кода, посмотрите, какие строки были протестированы, а какие – нет.