Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет

имени первого Президента России Б. Н. Ельцина»

Институт фундаментального образования

Кафедра интеллектуальных информационных технологий

Дисциплина «Web программирование»

**СОЗДАНИЕ ПОРТАЛА ДЛЯ РАЗМЕЩЕНИЯ ПОСТОВ**

КУРСОВАЯ РАБОТА

Пояснительная записка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Преподаватель | |  | |  |  | Свинцов Д.В. |
|  |  | | подпись преподавателя | |  |  |
|  |  | |  | |  |  |
| Студент гр.ФО-331002 |  | |  | |  | Белкина Е.В. |
|  |  | | подпись студента | |  |  |

Екатеринбург - 2016

**Оглавление**

Введение 2

1. Описание проекта 3

2. Обоснование проекта 3

3. Инструкция пользователя 4

3.1. Главная страница 4

3.2. Страница авторизации 5

3.3. Страница регистрации 6

3.4. Страница пользователя 6

3.5. Страница добавления тега 6

3.6. Страница добавления типа тега 7

3.7. Страница добавления поста 7

3.8. Страница поста 8

4. Инструкция программиста 9

4.1. Установка 9

4.2. Запуск 9

4.3. Инициализация 9

4.4. Описание модулей 9

4.5. Описание ключевых функций, классов 9

Выводы 23

# Введение

В качестве курсового проекта было выбрано написание сервера для сайта, на котором пользователи могли бы добавлять посты, включающие в себя различные изображения. Эта тема была выбрана, потому что множество сайтов, на которых можно было бы добавлять посты, не позволяют вставить внутрь поста изображения.

# Описание проекта

Проект должен состоять из сервера для поддержки сайта. На этом сайте пользователь может создать пост, привязанный к различным тегам. Также, пользователь может добавить типы тегов и сами теги.

# Обоснование проекта

В качестве темы для сайта я решила выбрать сайт с постами различных пользователей, так как на таком сайте возможно реализовать все требования к курсовой и в тоже время он предоставляет простор для творчества во время разработки. Законченный проект может быть размещен на онлайн сервере.

# Инструкция пользователя

Сайт состоит из 8 страниц, переход на каждую из которых, производится на соответствующий элемент страницы.

## Главная страница

Данная страница открывается, когда пользователь заходит на сайт. Она содержит следующие элементы:

1. Список всех постов, доступных на сайте;
2. Элементы управления для фильтрации постов по тегам;
3. Общие, для всех страниц, элементы управления перехода на главную страницу и переход на страницу авторизации.

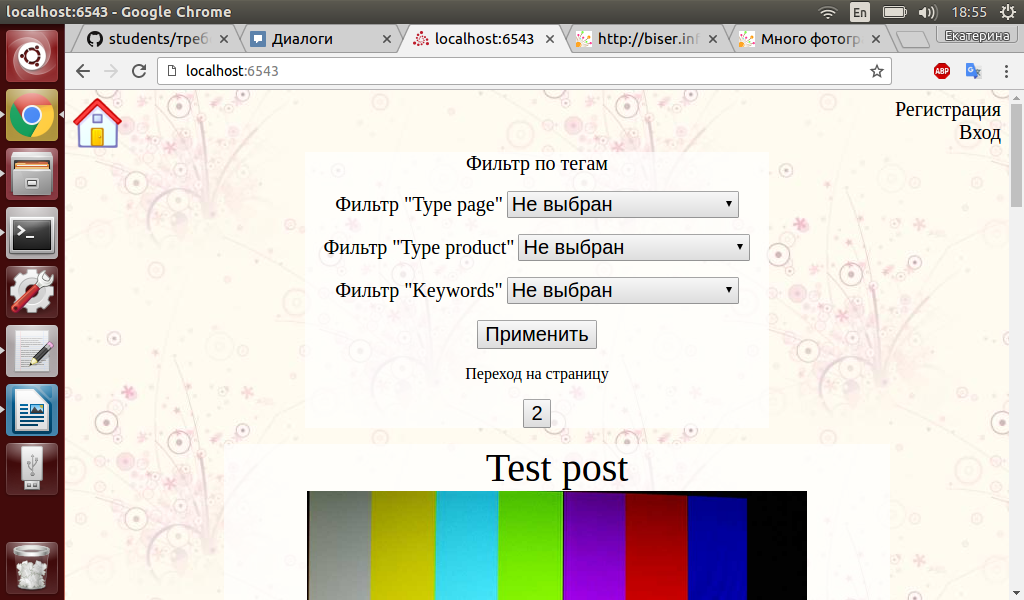
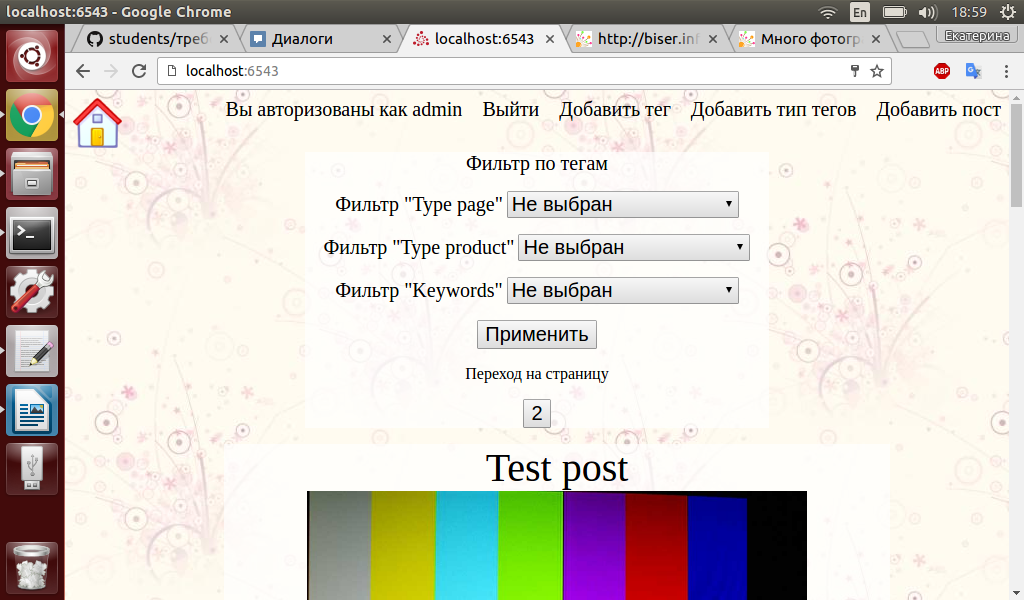


Рис.1. Главная страница неавторизированный пользователь

Рис.2. Главная страница авторизованный пользователь

## Страница авторизации

Открывается при нажатии на ссылку «вход» на главной странице. Содержит следующие элементы :

* Поле ввода логина
* Поле ввода пароля
* Кнопка подтверждения логина и пароля
* Переход на страницу регистрации

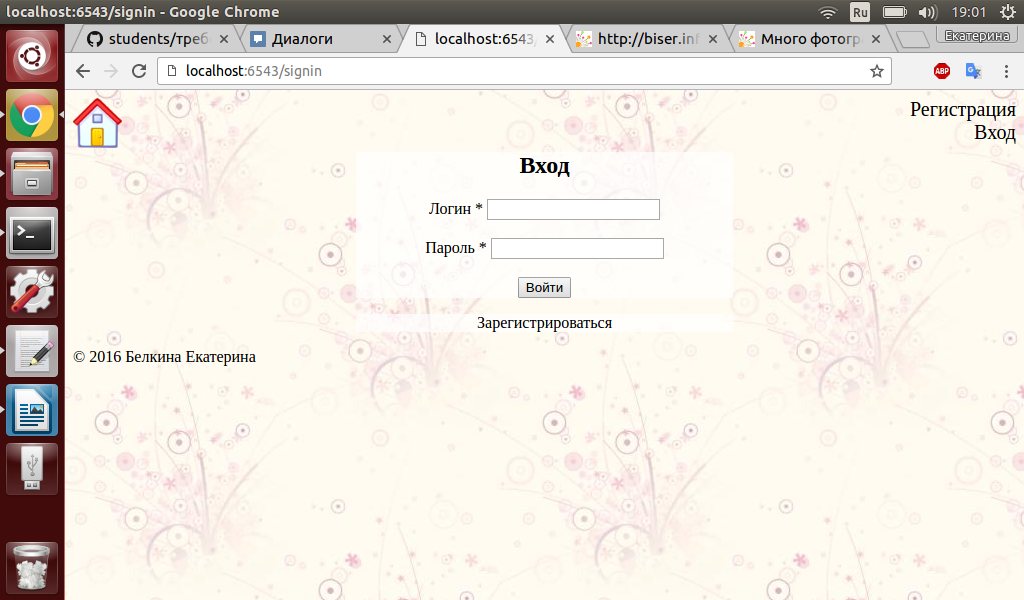


Рис.3. Страница авторизации

## Страница регистрации

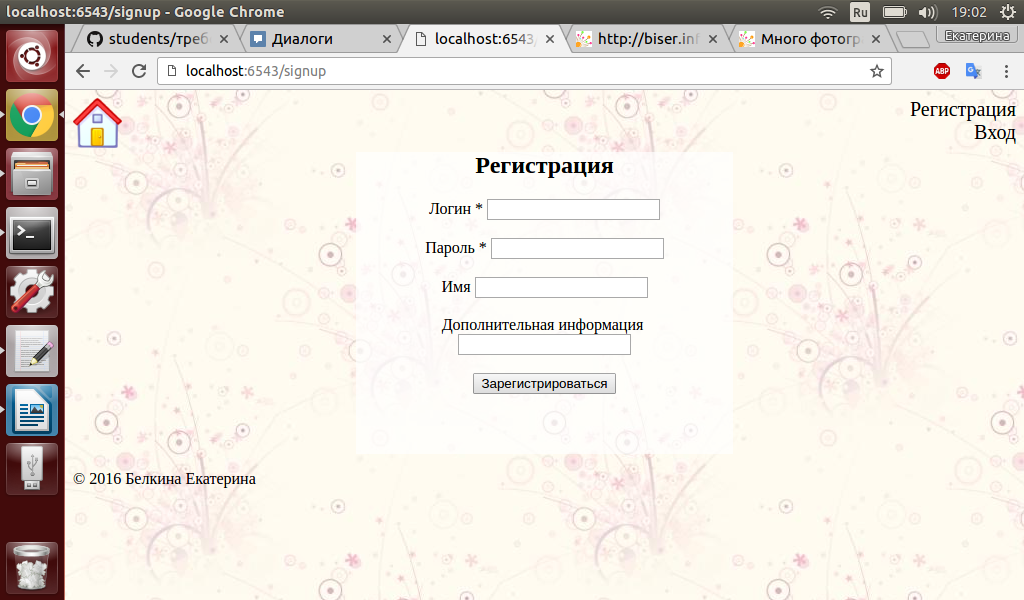
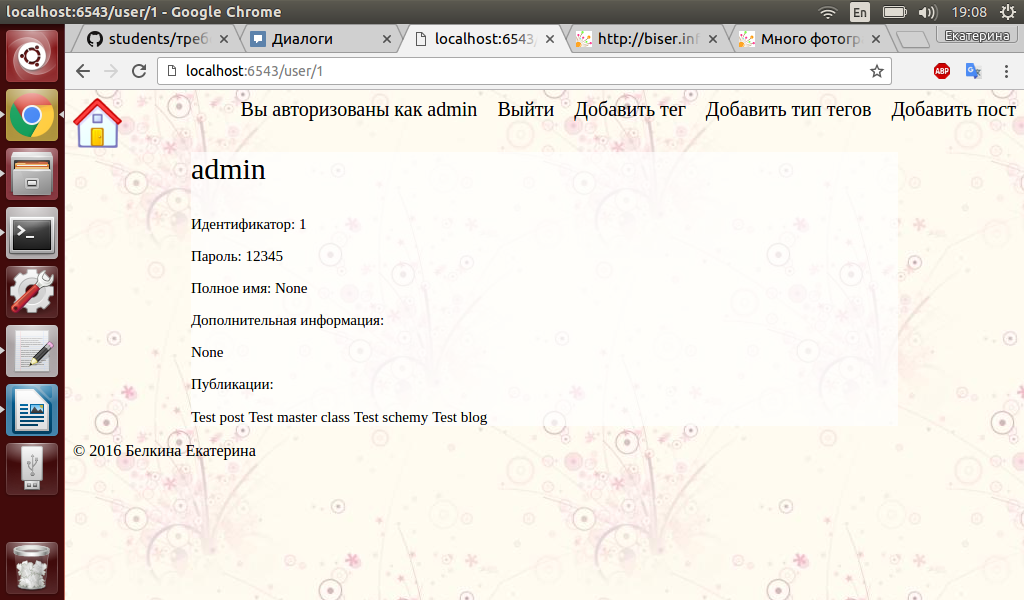


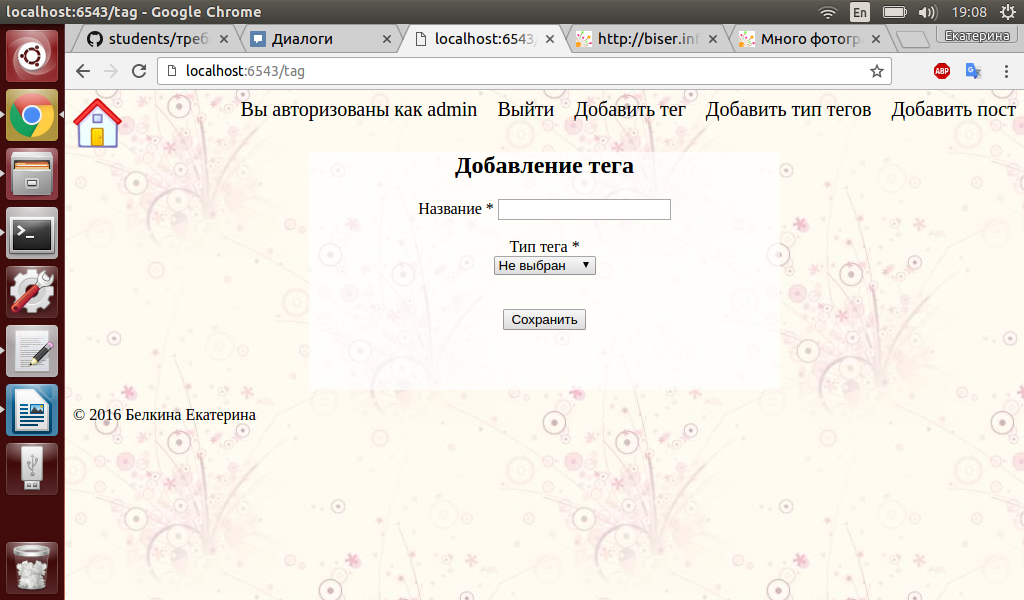
Рис.4. Страница регистрации

## Страница пользователя

На страницу пользователя отображена информация о нем, а также, если пользователь авторизован под тем же ником, то отображается его пароль.

Рис.5. Страница пользователя

## Страница добавления тега

Рис.6. Добавление тега

## Страница добавления типа тега

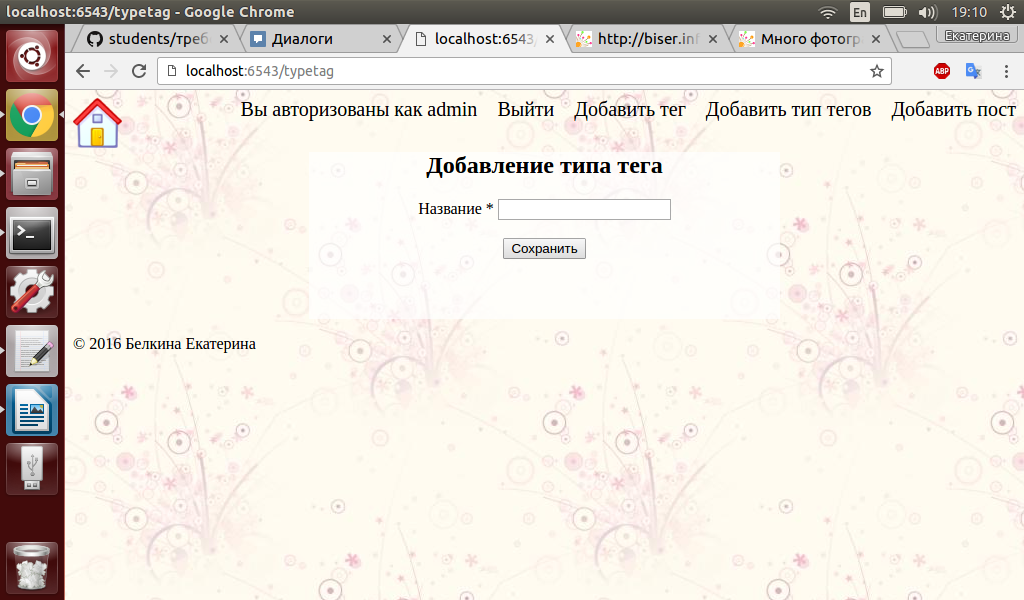


Рис. 7. Добавление типа тега

## Страница добавления поста

В текст поста можно вставить любое количество изображений.

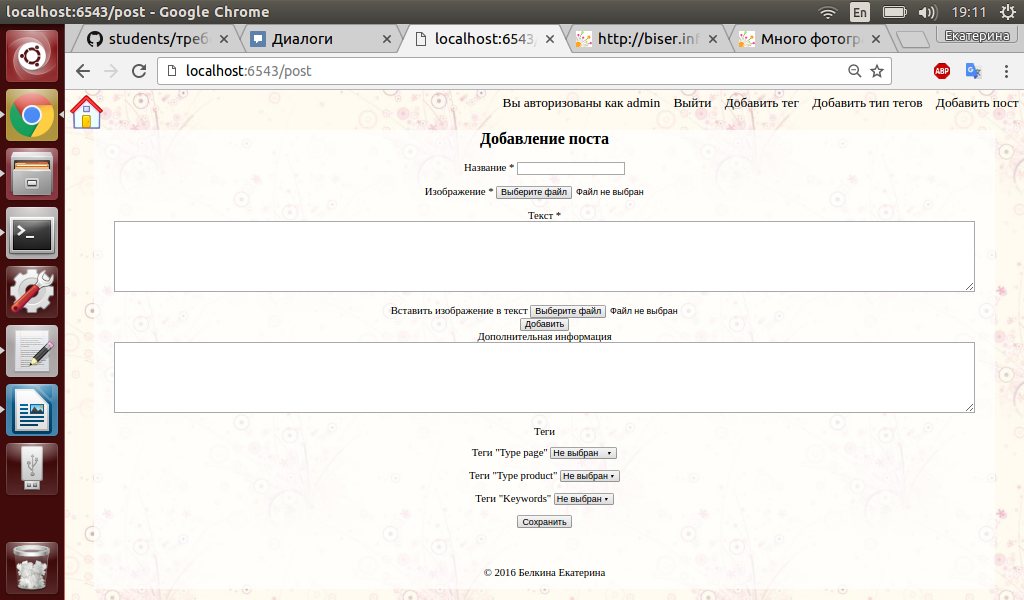


Рис. 8. Добавление поста

## Страница поста

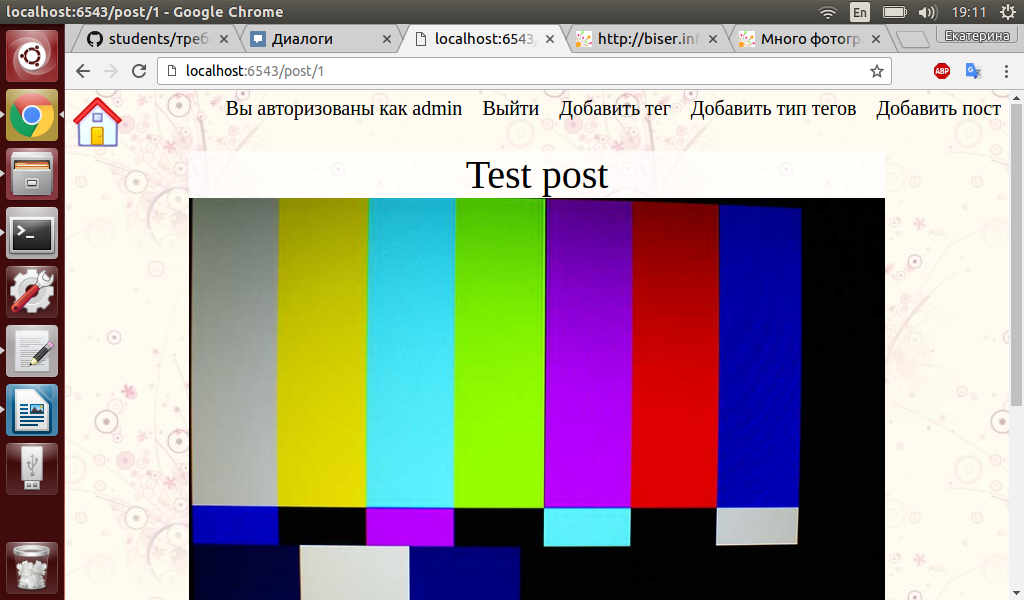


Рис. 9. Страница поста

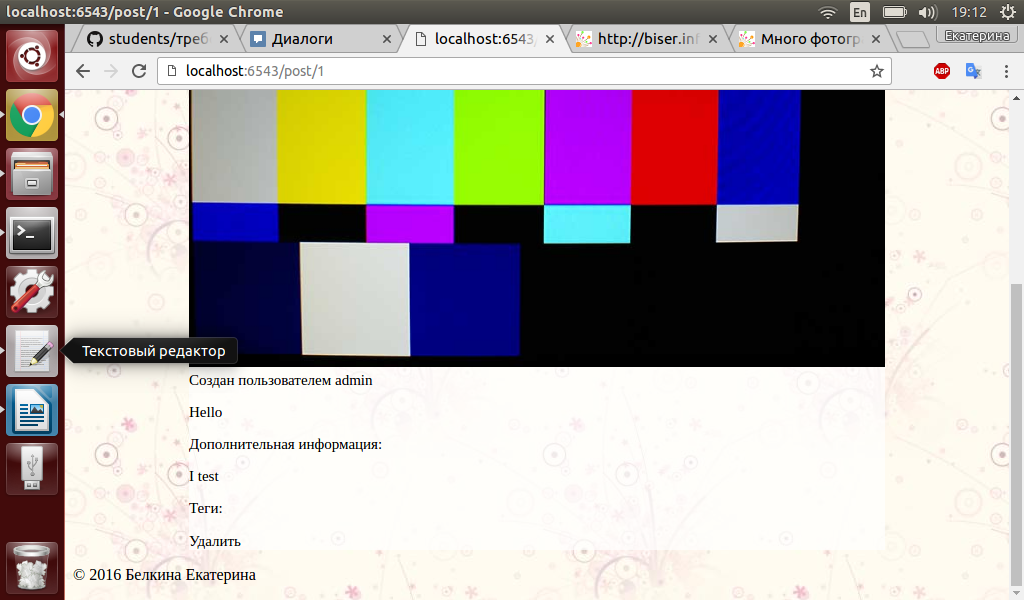


Рис. 10. Страница поста. продолжение

# Инструкция программиста

## Установка

Для установки требуется версия Python 2.7, наличие установленных пакетов Jinja2 и PyramidFramework, Alchemy.

В командной строке необходимо создать виртуальной пространство и перейти в папку проекта. Далее пишем команду:

python setup.py develop

## Запуск

Запустить проект можно следующей командой:

pserve development.ini

## Инициализация

Для инициализации баз данных прописываем в командной строке следующую команду:

initialize\_KPostServer\_db decelopment.ini

## Описание модулей

Файл \_\_init\_\_.py содержит создание wsgi сервера и добавление url путей сайта. Views.py содержит описание того, что будет возвращено сервером при переходах по сайту. Models.py описывает все таблицы, которые будут созданы в базе данных.

## Описание ключевых функций, классов

В файле models.py содержатся классы таблиц базы данных. С помощью этих классов можно изменять записи в базе данных.

from sqlalchemy import Table, Column, Integer, Text, create\_engine, String, ForeignKey, DateTime

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, backref, relationship

engine = create\_engine('sqlite:///kpost.db')

Session = sessionmaker()

Base = declarative\_base(bind=engine)

#Таблица для связи поста и тегов

association\_table = Table('association', Base.metadata,

Column('post\_id', Integer, ForeignKey('posts.id')),

Column('tag\_id', Integer, ForeignKey('tags.id'))

)

#Класс таблицы пользователя

class User(Base):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True)

name = Column(String)

password = Column(String)

fullname = Column(String, nullable = True)

info = Column(Text, nullable = True)

def \_\_repr\_\_(self):

return self.name + " " + self.password + " " + str(self.access)

#Класс таблицы поста

class Post(Base):

\_\_tablename\_\_ = 'posts'

id = Column(Integer, primary\_key=True)

image = Column(String)

name = Column(String)

text = Column(Text)

info = Column(Text, nullable = True)

user\_id = Column(Integer, ForeignKey('users.id'))

user = relationship('User', backref = backref('posts', lazy='dynamic'))

tags = relationship("Tag", secondary=association\_table, back\_populates="posts")

#Класс таблицы тега

class Tag(Base):

\_\_tablename\_\_ = 'tags'

id = Column(Integer, primary\_key=True)

name = Column(String)

posts = relationship("Post", secondary=association\_table, back\_populates="tags")

typetag\_id = Column(Integer, ForeignKey('typetags.id'))

typetag = relationship("TypeTag", backref = backref('tags', lazy='dynamic'))

#Класс таблицы типа тега

class TypeTag(Base):

\_\_tablename\_\_ = 'typetags'

id = Column(Integer, primary\_key=True)

name = Column(String)

#Класс таблицы изображения, вставляемого внутрь текста поста

class Image(Base):

\_\_tablename\_\_ = 'images'

id = Column(Integer, primary\_key=True)

name = Column(String)

Файл views.py определяет ответы серсера на различные html запросы пользователя.

from pyramid.view import view\_config

from pyramid.response import Response

from models import User, Session, Base, engine, Post, Tag, TypeTag, Image

from pyramid.security import (

remember,

forget,

)

from pyramid.httpexceptions import (

HTTPFound,

HTTPNotFound,

)

def get\_user(user\_name):

if user\_name!=None:

DBSession = Session(bind=engine)

return DBSession.query(User).filter(User.name==user\_name).first()

else: return None

#home

@view\_config(route\_name='home', renderer='templates/home.jinja2')

def my\_view(request):

DBSession = Session(bind=engine)

posts = DBSession.query(Post).all()

typetags = DBSession.query(TypeTag)

bd\_tags = DBSession.query(Tag)

tags = []

for typetag in typetags:

if str(typetag.id) in request.GET:

tag\_id = request.GET[str(typetag.id)]

tag = bd\_tags.filter\_by(id=tag\_id).first()

if tag != None:

tags.append(tag)

if len(tags)>0:

posts = posts.filter(Post.tags!=None).all()

if len(posts)>0:

result\_posts=[]

for post in posts:

post\_tags=post.tags

a=True

for tag in tags:

b=False

for t in post\_tags:

if t.id==tag.id: b=True

if a: a=b

if a: result\_posts.append(post)

posts = result\_posts

pages=None

if len(posts)>5:

pages=[]

page=1

if 'page' in request.GET:

page=int(request.GET['page'])

minpage=1

maxpage=10

countpage=len(posts) // 5

if len(posts) % 5>0:

countpage+=1

if maxpage>countpage:

maxpage=countpage

elif page>5:

minpage=page-4

maxpage=page+4

if maxpage>countpage:

minpage-=maxpage-countpage

maxpage=countpage

i=1

while i<=maxpage:

if i!=page:

pages.append(i)

i+=1

result\_posts=[]

i=(page-1)\*5

while i<len(posts) and i<page\*5:

result\_posts.append(posts[i])

i+=1

posts=result\_posts

return {'posts': posts,

'typetags': typetags,

'selected\_tags': tags,

'ruser': get\_user(request.authenticated\_userid),

'pages': pages

}

#post

@view\_config(route\_name='post', renderer='templates/post.jinja2')

def post\_view(request):

DBSession = Session(bind=engine)

post\_id = request.matchdict['id']

post = DBSession.query(Post).filter\_by(id=post\_id).first()

try:

post.user

post.tags

except BaseException: i=0

return {'post': post,

'ruser': get\_user(request.authenticated\_userid)

}

#post delete

@view\_config(route\_name='post\_delete', permission='add')

def post\_remove\_view(request):

DBSession = Session(bind=engine)

user = get\_user(request.authenticated\_userid)

post\_id = request.matchdict['id']

post = DBSession.query(Post).filter\_by(id=post\_id).first()

DBSession.delete(post)

DBSession.commit()

return HTTPFound(location = '/post/'+post\_id)

#user

@view\_config(route\_name='user', renderer='templates/user.jinja2')

def user\_view(request):

DBSession = Session(bind=engine)

user\_id = request.matchdict['id']

user = DBSession.query(User).filter\_by(id=user\_id).first()

try:

user.posts

except BaseException: i=0

return {'user': user,

'ruser': get\_user(request.authenticated\_userid)

}

#sign in

@view\_config(route\_name='signin', renderer='templates/signin.jinja2')

def singin\_view(request):

if (get\_user(request.authenticated\_userid)!=None):

headers = forget(request)

return HTTPFound(location = '/signin', headers = headers)

if 'POST' == request.method:

login = request.params['login']

password = request.params['password']

DBSession = Session(bind=engine)

user = DBSession.query(User).filter(login==User.name).first()

if user!=None and user.password == password:

headers = remember(request, login)

return HTTPFound(location = '/', headers = headers)

else:

return {'message': "Incorrect login or password",

'ruser': get\_user(request.authenticated\_userid)

}

else: return{'ruser': get\_user(request.authenticated\_userid)}

#sign out

@view\_config(route\_name='signout', permission='add')

def signout\_view(request):

headers = forget(request)

return HTTPFound(location = '/', headers = headers)

#sign up

@view\_config(route\_name='signup', renderer='templates/signup.jinja2')

def singup\_view(request):

if (get\_user(request.authenticated\_userid)!=None):

headers = forget(request)

return HTTPFound(location = '/signup', headers = headers)

headers = forget(request)

if 'POST' == request.method:

nameU = request.params['name']

passwordU = request.params['password']

fullnameU = request.params['fullname']

infoU = request.params['info']

DBSession = Session(bind=engine)

user=None

message=""

if DBSession.query(User).filter\_by(name=nameU).first()==None:

if passwordU!=None and passwordU!="" and nameU!=None and nameU!="":

user = User(name=nameU, password = passwordU, fullname=fullnameU, info=infoU)

DBSession.add(user)

DBSession.commit()

else: message="nofull"

else: message="login"

if user!=None:

headers = remember(request, nameU)

return HTTPFound(location = '/', headers = headers)

else:

return {'message': message,

'ruser': get\_user(request.authenticated\_userid)

}

else: return{'ruser': get\_user(request.authenticated\_userid)}

#add post

@view\_config(route\_name='addpost', renderer='templates/addpost.jinja2', permission='add')

def addpost\_view(request):

if (get\_user(request.authenticated\_userid)==None):

headers = forget(request)

return HTTPFound(location = '/addpost', headers = headers)

DBSession = Session(bind=engine)

if 'POST' == request.method:

textP = request.params['text']

infoP=None

nameP=request.params['name']

if 'info' in request.params:

infoP = request.params['info']

textim=False

try:

request.params["textimage"].value

textim=True

except Exception: textim=False

if textim:

im=Image(name="")

DBSession.add(im)

DBSession.commit()

im.name=str(im.id)+".jpg"

DBSession.commit()

with open("kpostserver/static/image/image\_in\_text/"+im.name,'wb') as f:

f.write(request.params["textimage"].value)

nameP=None

textP=textP + " <br><a><img style=\"margin-left:auto; margin-right:auto; width: 100%;\" src=\"http://localhost:6543/static/image/image\_in\_text/"+im.name+"\" alt=\"photo\"></a></br> "

if 'name' in request.params:

nameP=request.params['name']

return{'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag),

'nameP': nameP,

'textP': textP,

'infoP': request.params['info'] }

elif 'name' in request.params and request.params['name']!="" and 'image' in request.params and textP!="":

try:

request.params["image"].value

except Exception:

return {'message': "nofull",

'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag),

'textP': textP,

'nameP': nameP

}

nameP = request.params['name']

post = Post(name=nameP, image="", user\_id=get\_user(request.authenticated\_userid).id, text=textP, info=infoP)

DBSession.add(post)

DBSession.commit()

post.image="im"+str(post.id)+".jpg"

DBSession.commit()

with open("kpostserver/static/image/"+post.image,'wb') as f:

f.write(request.params["image"].value)

typetags = DBSession.query(TypeTag)

bd\_tags = DBSession.query(Tag)

for typetag in typetags:

if str(typetag.id) in request.params:

tag\_id = request.params[str(typetag.id)]

tag = bd\_tags.filter(Tag.id==tag\_id).first()

if tag != None:

post.tags.append(tag)

DBSession.commit()

return HTTPFound(location = '/post/'+str(post.id))

else:

return {'message': "nofull",

'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag),

'textP': textP,

'nameP': nameP

}

else: return{'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag) }

#add type tag

@view\_config(route\_name='addtypetag', renderer='templates/addtypetag.jinja2', permission='add')

def addtypetag\_view(request):

if (get\_user(request.authenticated\_userid)==None):

headers = forget(request)

return HTTPFound(location = '/addtypetag', headers = headers)

DBSession = Session(bind=engine)

if 'POST' == request.method:

name = request.params['name']

if name!="":

if DBSession.query(TypeTag).filter(TypeTag.name==name).first()==None:

typetag = TypeTag(name=name)

DBSession.add(typetag)

DBSession.commit()

return HTTPFound(location = '/')

else:

return {'message': "typetag",

'ruser': get\_user(request.authenticated\_userid)

}

else:

return {'message': "nofull",

'ruser': get\_user(request.authenticated\_userid)

}

else: return{'ruser': get\_user(request.authenticated\_userid)}

#add tag

@view\_config(route\_name='addtag', renderer='templates/addtag.jinja2', permission='add')

def addtag\_view(request):

if (get\_user(request.authenticated\_userid)==None):

headers = forget(request)

return HTTPFound(location = '/addtag', headers = headers)

DBSession = Session(bind=engine)

if 'POST' == request.method:

name = request.params['name']

typetag = request.params['typetag']

if name!="" and typetag!="":

typetag = DBSession.query(TypeTag).filter(TypeTag.id==typetag).first()

tag = Tag(name=name, typetag\_id=typetag.id)

DBSession.add(tag)

DBSession.commit()

return HTTPFound(location = '/')

else:

return {'message': "nofull",

'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag)

}

else: return{'ruser': get\_user(request.authenticated\_userid),

'typetags': DBSession.query(TypeTag) }

# Выводы

Во время работы над проектом я научилась создавать Web сервер при помощи фреймворка Piramid, работать с базами данных, используя ОРМ SQL Alchemy. Во время разработки сайта я использовала следующие технологии:

* Фреймворк Pyramid
* Шаблонизатор Jinja2
* ОРМ SQLAlchemy
* SQLite

А также научилась создавать группы пользователей и авторизовать пользователей на сайте.

Это существенно расширило мои знания и в будущем я смогу самостоятельно написать необходимый мне Web сервер.