Целью моего проекта было создание новостного сайта.

Ход работы:

1) Для начала я создала проект (*django-admin.py startproject newsite*)

2) Далее, в файле *newsite/settings.py* в разделе **Databases** в графе **‘Engine’** добавила строчку *‘django.db.backends.sqlite3’*; в графе **‘Name’** - ‘*database.db’*; и в конце раздела **Installed\_Apps** написала *'django.contrib.admin'*.

3) Далее я изменила файл *newsite/urls.py:*

*from django.conf.urls import patterns, include, url   
from django.contrib import admin   
admin.autodiscover()   
urlpatterns = patterns('', url(r'^admin/', include(admin.site.urls)), )*

4) После этого необходимо сохранить изменения в базе данных командой *python manage.py makemigrations* и *python manage.py migrate* и создать «супер пользователя» командой *python manage.py createsuperuser.*

5) Запустить сервер через команду *python manage.py runserver* и авторизироваться через созданного «супер пользователя» (в моем случае через *admin*).

6) Создаю основное приложение *links (python manage.py startapp links).*

7) Далее нужно внести изменения в файл links/models.py. Для этого я добавляю 2 класса:  
*class Link(models.Model):   
title = models.CharField("Headline", max\_length=100) - эта строчка отвечает за то, чтобы заголовок новости был виден всем;  
submitter = models.ForeignKey(User) – посетитель должен иметь значение внешнего ключа в базе данных;   
submitted\_on = models.DateTimeField (auto\_now\_add=True) – при посещении текущая дата/время ставится автоматически;   
rank\_score = models.FloatField(default=0.0) – оценка, которую получает новость;   
url = models.URLField("URL", max\_length=250, blank=True) – позволяет передавать каждой новости ссылку на источник;   
description = models.TextField(blank=True) – позволяет добавлять доп. описание новости   
def \_\_unicode\_\_(self): - функция для представления модели в строковом виде*

8) Создаю изменения в *links/admin.py*, необходимо для проверки условия, что для входа на сайт администратор должен быть зарегистрирован:

*class LinkAdmin(admin.ModelAdmin):   
pass   
admin.site.register(Link, LinkAdmin)   
class VoteAdmin(admin.ModelAdmin):   
pass   
admin.site.register(Vote, VoteAdmin)*

9) Ввожу в *links/views.py* :

*LinkListView(ListView):   
model = Link*

10) Изменяю файл *newsite/urls.py*, необходимо для определения «домашней страницы»:

*url(r'^$', LinkListView.as\_view(), name='home'),*

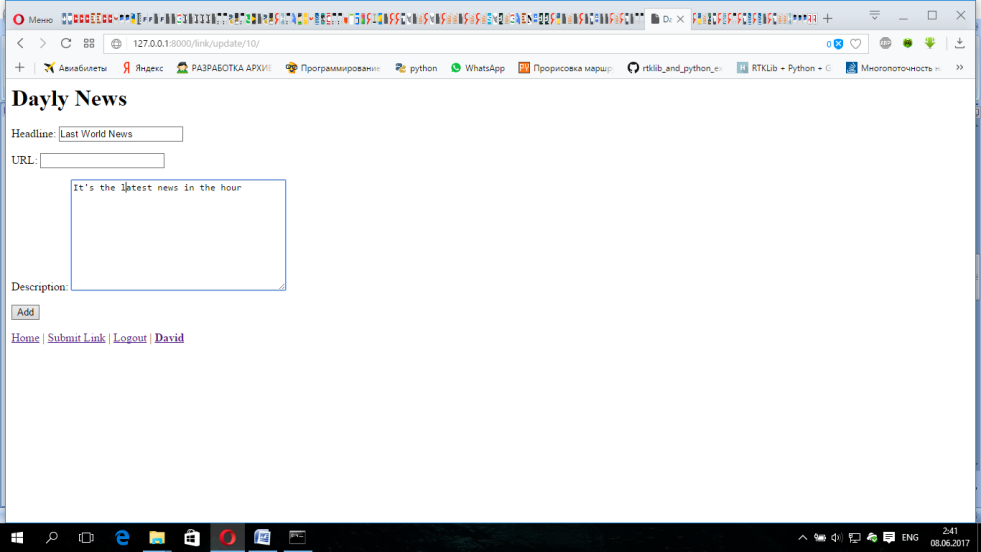
11) Создаю новый каталог шаблонов *newsite/templates/links/link\_list.html*:

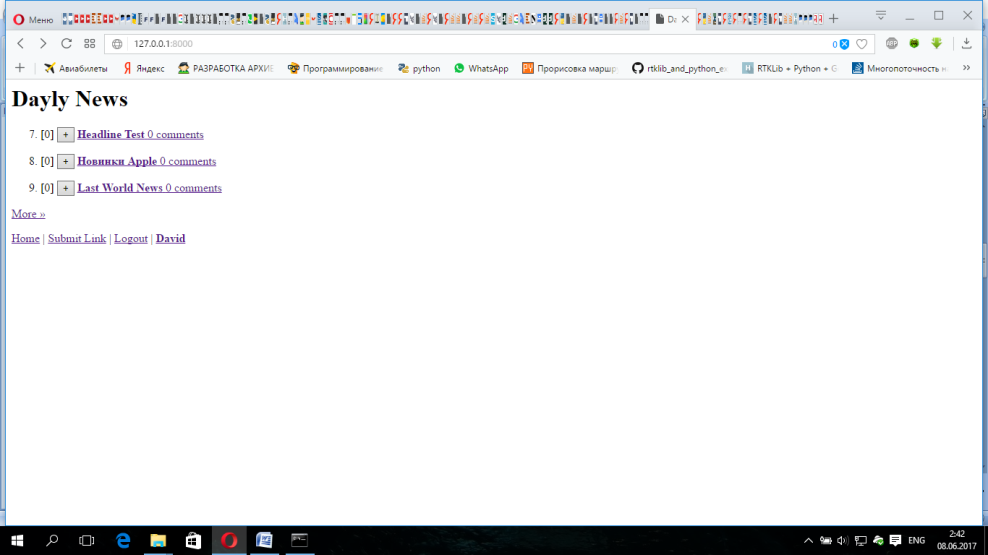
*<ol> {% for link in object\_list %} <li> <a href="{{ link.url }}"> <b>{{ link.title }}</b> </a> </li> {% endfor %} </ol>*

12) Добавляю приложения в *settings.py* в конец **INSTALLED\_APPS** и сохраняю изменения в базе данных:

*'links',   
'newsite'*

13) Запускаю сервер и добавляю несколько новостей





14) Создаю общий базовый шаблон в *newsite/templates/base.html*, для создания заголовка страницы :

*<html> <body> <h1>Dayly News</h1> {% block content %} {% endblock %} </body> </html>*

15) Добавляю классы в *links/models.py* :

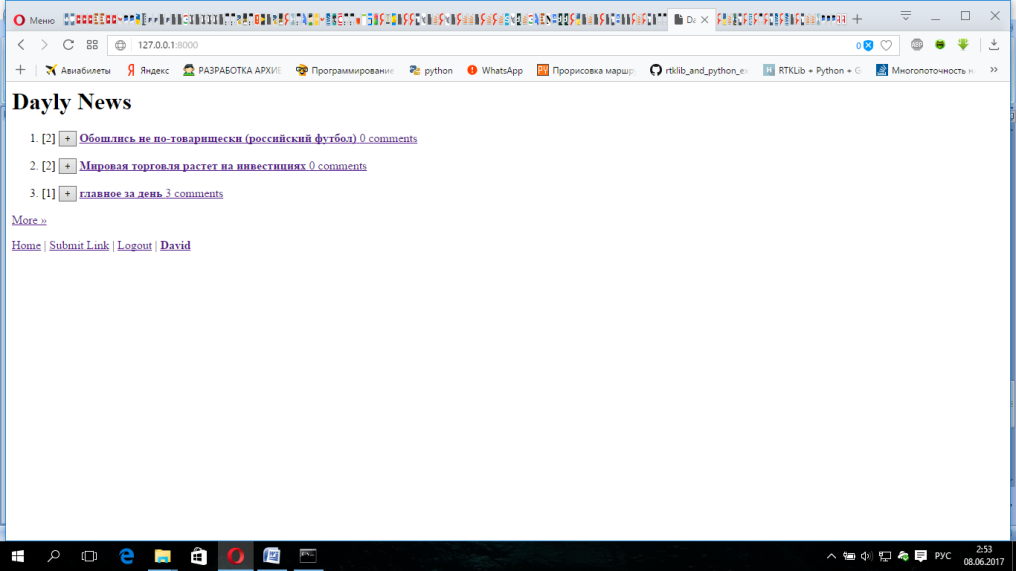
*class LinkVoteCountManager(models.Manager):   
def get\_query\_set(self):   
return super(LinkVoteCountManager, self).get\_query\_set().annotate( votes=Count('vote')).order\_by('-votes')* – данная строчка возвращает то, что было записано при создании «супер пользователя», также добавляет определенное поле, которое является учетной записью; метод annotate – является способом передачи запроса, в результате происходит подсчет голосов на каждой новости и сортировка по убыванию голосов

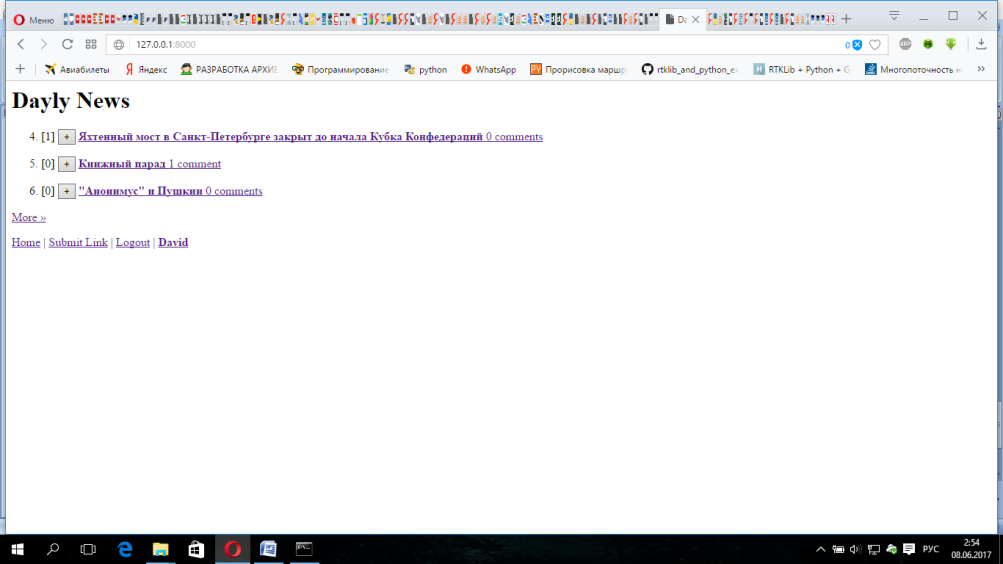
16) Вставьте эти две строки в класс **Link** в *links/models.py* :

*with\_votes = LinkVoteCountManager()   
objects = models.Manager()* – значение по умолчанию

17) Изменяю *links/views.py* и вставляю строки в класс **LinkListView**, определяет количество новостей на одной странице :

*queryset = Link.with\_votes.all()   
paginate\_by = 3*





Для создания пользовательской страницы входа необходимо:

1. Добавить в *newsite/urls.py*, ссылки для входа и выхода из аккаунта :

*url(r'^login/$', 'django.contrib.auth.views.login', { 'template\_name': 'login.html'}, name="login"), url(r'^logout/$', 'django.contrib.auth.views.logout\_then\_login', name="logout"),*

1. Добавляю адреса **URL входа / выхода** из системы в *newsite /settings.py*:

*LOGIN\_URL=reverse\_lazy('login') - reverse\_lazy необходим для определения URL еще до загрузки конфигурации  
LOGIN\_REDIRECT\_URL = reverse\_lazy('home')   
LOGOUT\_URL=reverse\_lazy('logout')*

1. Добавляю строку в *settings.py*, чтобы добавить приложение регистрации в конец **INSTALLED\_APPS**.

*'registration'*

1. Сохраняю изменения в базе данных
2. Добавляю в *urls.py* :

*url(r'^accounts/', include('registration.backends.simple.urls')),*

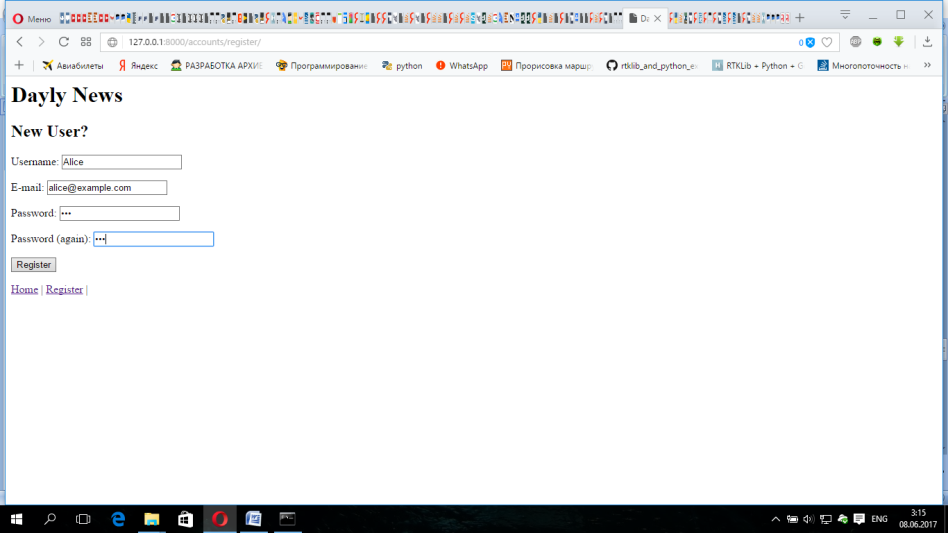
1. Добавьте класс **UserProfile** в *models.py* :

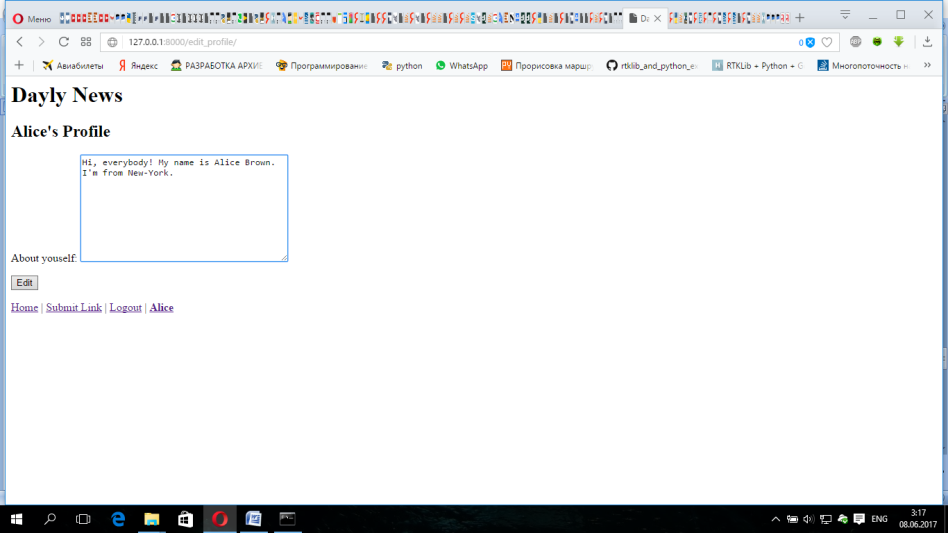
*class UserProfile(models.Model):   
user = models.OneToOneField(User, unique=True) – проверяет уникальность создаваемых аккаунтов  
about\_youself = models.TextField(null=True) – возможность написать о себе какую-либо информацию, значение null необходимо, чтобы поле возникло в тот момент, когда пользователь уже создал  
if created: - проверка наличия регистрации  
profile, created = UserProfile.objects.get\_or\_create(user=instance)   
post\_save.connect(create\_profile, sender=User) - каждый раз когда модель пользователя отправляет сообщение СОХРАНИТЬ, обработчик создаст профиль.*

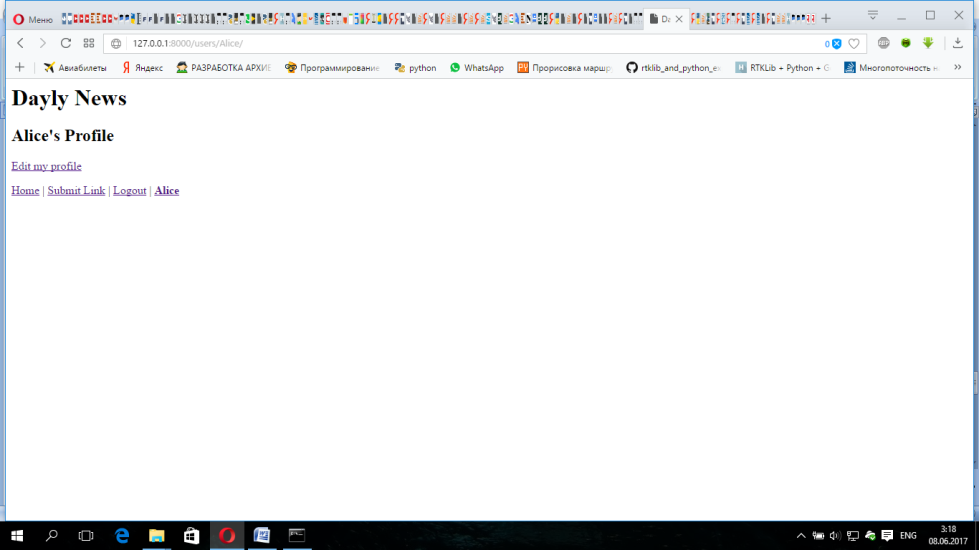
1. Синхронизирую изменения в базу данных
2. Добавляю в *admin.py* ссылки,:

*class UserProfileInline(admin.StackedInline):   
model = UserProfile can\_delete = False   
class UserProfileAdmin(UserAdmin):   
inlines=(UserProfileInline, )   
admin.site.unregister(get\_user\_model())* – Если пользователь не зарегистрирован, то получить форму регистрации, иначе получить профиль админа  
*admin.site.register(get\_user\_model(), UserProfileAdmin)*

1. Запускаю сервер и создаю некоторого пользователя







1. Добавляю в *views.py* ссылки:

*class UserProfileDetailView(DetailView):   
model = get\_user\_model()   
slug\_field = "username"   
template\_name = "user\_detail.html"   
def get\_object(self, queryset=None):   
user = super(UserProfileDetailView, self).get\_object(queryset) UserProfile.objects.get\_or\_create(user=user) return user*

1. Далее добавляю **URL-адреса** в *urls.py*, для возможности создать профиль :

*url(r'^users/(?P<slug>\w+)/$', UserProfileDetailView.as\_view(), name="profile"),*

1. Изменяю шаблон *base.html:*

*{% if user.is\_authenticated %} <a href="{% url 'logout' %}">Logout</a> |   
<a href="{% url 'profile' slug=user.username %}">  
<b>{{ user.username }}</b></a> {% else %}   
<a href="{% url 'registration\_register' %}">Register</a> |*

1. Добавляю класс **UserProfileEditView** в *views.py*, должно быть соблюдено условие безопасности создаваемых профилей. Это значит, что если пользователь зашел через администратора, то он не может изменять уже созданные другими людьми аккаунты. Конечная функция возвращает кортеж из зарегистрированных пользователей, в ачестве аргумента берется профиль пользователя:

*class UserProfileEditView(UpdateView):   
model = UserProfile form\_class = UserProfileForm   
template\_name = "edit\_profile.html"   
def get\_object(self, queryset=None):   
return UserProfile.objects.get\_or\_create(user=self.request.user)[0]   
def get\_success\_url(self):   
return reverse("profile", kwargs={'slug': self.request.user})*

1. Создаю шаблон *links/forms.py* :

*class UserProfileForm(forms.ModelForm):   
class Meta:   
model = UserProfile   
exclude = ("user")* – исключением являются поля пользователя

1. Изменяю *urls.py*. Декоратор**auth** необходим, чтобы предотвратить просмотр неактивными пользователями.

*url(r'^edit\_profile/$', auth(UserProfileEditView.as\_view()), name="edit\_profile"),*

1. В *templates/user\_detail* добавляю следующие строчки :

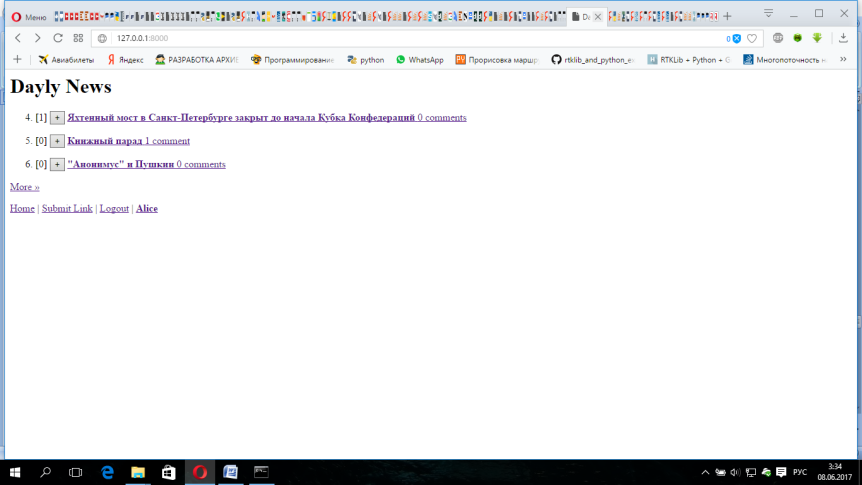
*{% if object.username == user.username %} <p>  
<a href='{% url "edit\_profile" %}'  
>Edit my profile</a></p> {% endif %}*

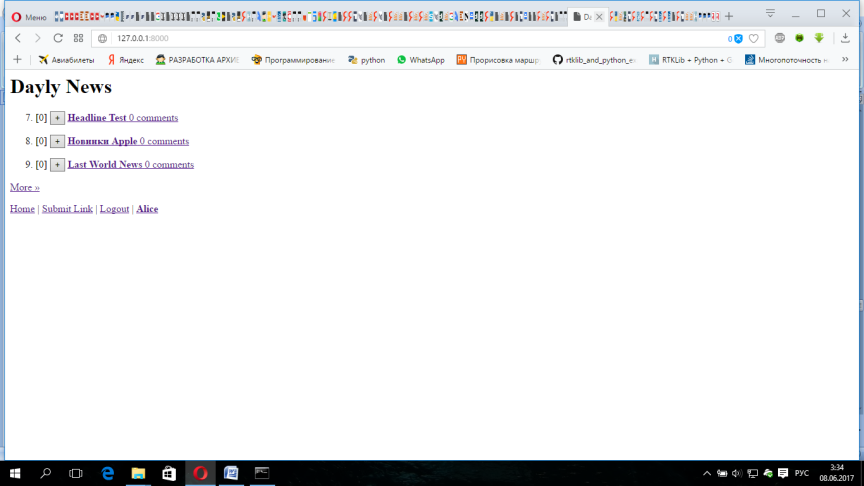
Для перехода на следующую страницу необходимо:

1. Добавить следующий фрагмент в *templates/links/link\_list.html*, для добавления кнопки **MORE**:

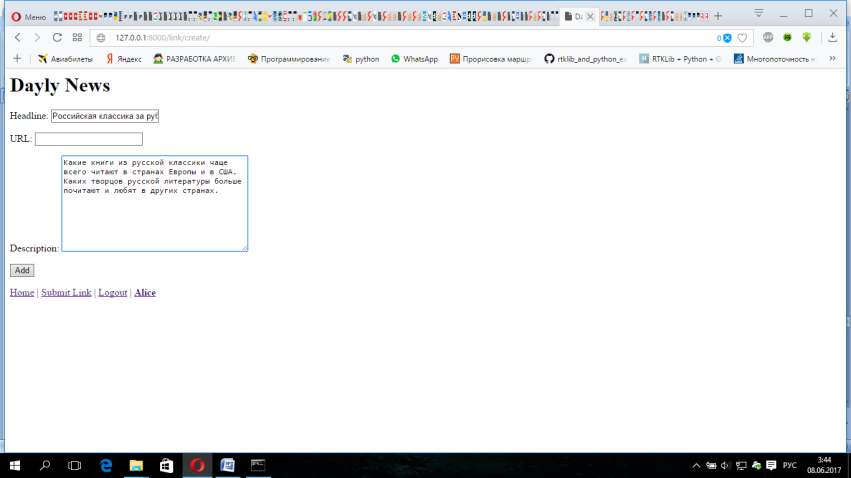
*{% if is\_paginated %} <div class="pagination"> {% if page\_obj.has\_next %} <a href="?page={{ page\_obj.next\_page\_number }}">More &raquo;</a> {% endif %} </div> {% endif %}*

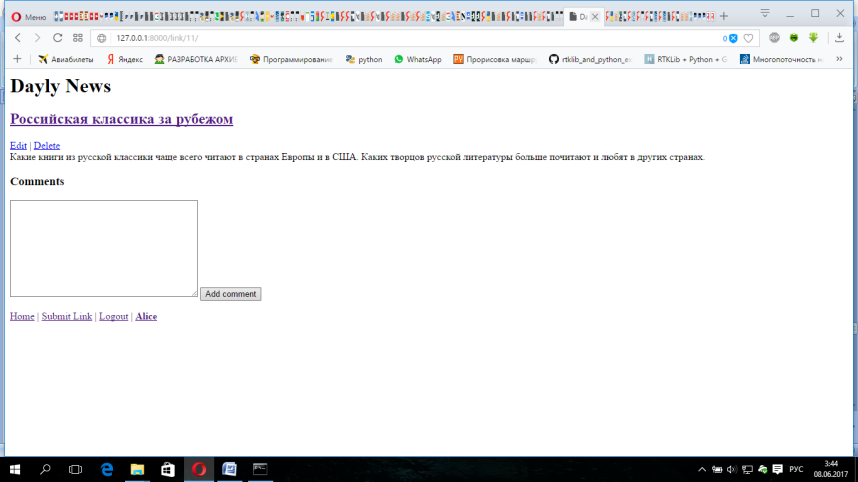
1. Запускаю сервер

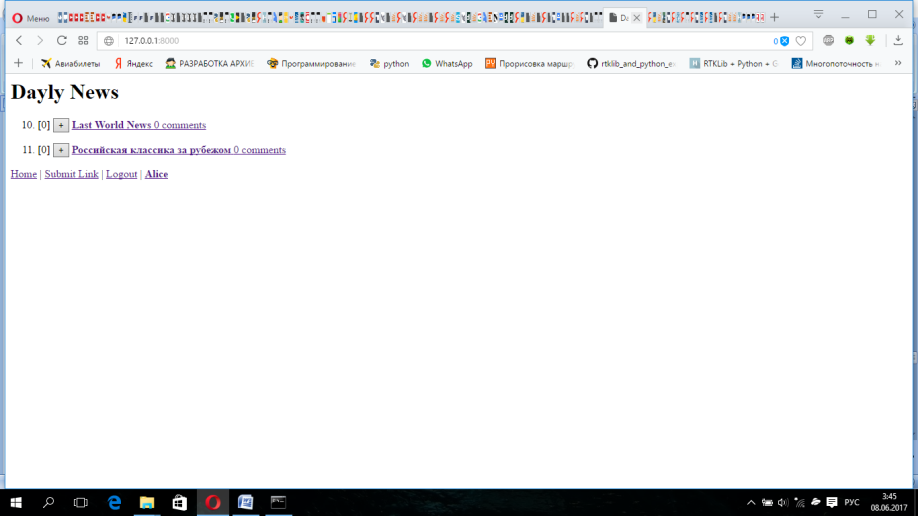




1. Чтобы пользователи могли отправлять ссылки, необходимо создание новой форму в *`links/forms.py`:   
   class LinkForm(forms.ModelForm):   
   class Meta: model = Link   
   exclude = ("submitter", "rank\_score")*
2. В *`links/views.py`:   
   class LinkCreateView(CreateView):   
   model = Link form\_class = LinkForm   
   def form\_valid(self, form):   
   f = form.save(commit=False)   
   f.rank\_score = 0.0   
   f.submitter = self.request.user   
   f.save()   
   return super(CreateView, self).form\_valid(form)*
3. Теперь чтобы загрузить ссылку необходимо войти в систему. Запускаю сервер и через пользователя создаю новую новость







1. Чтобы появилась возможность создавать комментарии нужно:
2. Сначала добавить приложения в  *settings.py* :

**INSTALLED\_APPS** *= ( 'django.contrib.admin',   
'django.contrib.comments'*,

1. Далее необходимо сохранить изменения в базе данных
2. Нужно показать количество комментариев на первой странице. Поэтому я добавляю следующие строки в  *templates/links/link\_list.html* в начале и в середине шаблона:

*<a href="{% url 'link\_detail' pk=link.pk %}">   
<b>{{ link.title }}</b>   
{% get\_comment\_count for link as comment\_count %}   
{{ comment\_count }} comment{{ comment\_count|pluralize }} </a>*

1. Далее добавляю в *urls.py*:

*url(r'^comments/', include('django.contrib.comments.urls')),*

1. Перехожу на сайт и создаю несколько комментариев к новости

