

В этой работе вы напишите простое приложение для ведения заметок, покрывая каждую часть приложения тестами. Ваше приложение будет работать в нескольких докер-контейнерах (о которых вы вскоре узнаете). И в конце разработаете API и swagger-схему для этого приложения.

## Создание нового проекта

### Замечание

Эта работа основана на [замечательном руководстве](#) по Django от компании Six Feet Up.

Как обычно создайте новую ветку разработки, папку для вашего проекта `elevennote` и виртуальное окружение с именем `elevennote-env`, в котором установите следующие пакеты:

```
(elevennote-env) $ pip install Django==2.0.1
(elevennote-env) $ pip install psycpg2==2.7.3.2
(elevennote-env) $ pip install python-decouple==3.1
(elevennote-env) $ pip install uWSGI==2.0.15
```

Можете запустить команду `pip freeze` для просмотра установленных пакетов:

```
(elevennote-env) $ pip freeze
Django==2.0.1
psycpg2==2.7.3.2
uWSGI==2.0.15
python-decouple==3.1
pytz==2017.2
```

Все зависимости будем хранить в отдельном манифесте:

```
(elevennote-env) $ mkdir requirements
(elevennote-env) $ pip freeze > requirements/base.txt
(elevennote-env) $ echo "-r base.txt" >> requirements/dev.txt
(elevennote-env) $ echo "-r base.txt" >> requirements/prod.txt
(elevennote-env) $ echo "django-debug-toolbar==1.9.1" >> requirements/dev.txt
```

Теперь создадим новый проект с помощью команды `django-admin` (исходники проекта будем хранить в папке `src`). Обратите внимание, что имя проекта `config`, а все файлы проекта будут созданы в текущей рабочей директории (на что указывает `.`):

```
(elevennote-env) $ mkdir src && cd $_
(elevennote-env) $ django-admin startproject config .
(elevennote-env) $ ls
config manage.py
(elevennote-env) $ cd ..
```

В результате вы должны получить следующую структуру проекта:

```

elevennote/
├── requirements
│   ├── base.txt
│   ├── dev.txt
│   └── prod.txt
└── src
    ├── config
    │   ├── __init__.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── manage.py

```

Можете проверить, что проект запускается командой `python manage.py runserver` (если после запуска была создана БД `db.sqlite3`, а какие-то файлы закешированы `__pycache__`, то можете смело их удалить и добавить в `.gitignore`).

Конфигурацию нашего проекта разделим на:

- `base.py` - общая конфигурация для всех развертываний проекта;
- `local.py` - конфигурация, которая используется при разработке (включенный режим отладки, дополнительные пакеты для разработки, например, `django-debug-toolbar`);
- `production.py` - настройки, которые используются для production-сервера.

### Замечание

Если по каким-либо причинам приложение `tree` не найдено, [здесь](#) вы найдете информацию по его установке.

```

(elevennote-env) $ mkdir src/config/settings
(elevennote-env) $ mv src/config/settings.py src/config/settings/base.py
(elevennote-env) $ touch src/config/settings/__init__.py
(elevennote-env) $ touch src/config/settings/local.py
(elevennote-env) $ touch src/config/settings/production.py
(elevennote-env) $ touch src/config/settings/settings.ini
(elevennote-env) $ tree src/config
src/config
├── __init__.py
├── settings
│   ├── __init__.py
│   ├── base.py
│   ├── local.py
│   ├── production.py
│   └── settings.ini
├── urls.py
└── wsgi.py

```

Содержимое файла `src/config/settings/base.py`:

```

import os
from decouple import config

def root(*dirs):
    base_dir = os.path.join(os.path.dirname(__file__), '..', '..')
    return os.path.abspath(os.path.join(base_dir, *dirs))

BASE_DIR = root()

SECRET_KEY = config('SECRET_KEY')

INSTALLED_APPS = [
    # ...
]

MIDDLEWARE = [
    # ...
]

ROOT_URLCONF = 'config.urls'

TEMPLATES = [
    # ...
]

WSGI_APPLICATION = 'config.wsgi.application'

AUTH_PASSWORD_VALIDATORS = [
    # ...
]

STATIC_URL = '/static/'

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True

```

Содержимое файла `configs/settings/local.py`:

```

from .base import *

DEBUG = True

INSTALLED_APPS += [

```

```

'django.contrib.postgres',
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': config('DB_PORT', cast=int),
    }
}

```

В `config/settings/settings.ini` укажите секретный ключ для Django, имя БД, пользователя БД, его пароль, адрес хоста и порт, на котором будет запущена БД. Ниже представлены значения некоторых полей по умолчанию.

#### Замечание

Не добавляйте `settings.ini` в ваш репозиторий, так как он содержит пароли и ключи. Чтобы случайно его не закоммитить добавьте соответствующую запись в `.gitignore`.

#### Замечание

Если сгенерированный `SECRET_KEY` содержит символ `%`, и при этом на этапе запуска приложения возникает ошибка парсинга - продублируйте `%` везде в ключе, где этот символ встречается.

```

[settings]
SECRET_KEY=MY_SECRET_KEY
DB_NAME=postgres
DB_USER=postgres
DB_PASSWORD=
DB_HOST=db
DB_PORT=5432

```

В файле `manage.py` необходимо заменить имеющийся путь к конфигурации проекта на следующий:

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "config.settings.local")
```

## Запуск проекта в докере

#### Замечание

Для знакомства с докером можно почитать статью [What is docker and how to use it with python](#), а также занимательную статью в [блоге](#) Ивана Гришаева. Дополнительно можете посмотреть выступление Антона Егорова с Moscow Python: [Докеризация приложений на Python](#).

Создадим docker-окружение для разработки и запуска нашего проекта. Установите докер для вашей системы (инструкции можно получить [тут](#)) и создайте `Dockerfile` в корне вашего проекта:

```
$ touch Dockerfile
```

#### Dockerfile

```
FROM python:3.6
ENV PYTHONUNBUFFERED 1
RUN mkdir /config
COPY requirements /config/requirements
RUN pip install -r /config/requirements/dev.txt
RUN mkdir /src;
WORKDIR /src
```

Теперь в корне проекта создадим файл `docker-compose.yml`:

#### Замечание

Обратите внимание на `sleep 5`. Это очень наивное решение [проблемы зависимостей](#) между контейнерами.

```
$ touch docker-compose.yml
```

#### docker-compose.yml

```
version: '3'
services:
  nginx:
    image: nginx:latest
    container_name: ng01
    ports:
      - "8000:8000"
    volumes:
      - ./deploy/nginx:/etc/nginx/conf.d
    depends_on:
      - web
  web:
    build: .
    container_name: dg01
    command: bash -c "sleep 5 && python manage.py makemigrations && python manage.py migrate && uwsgi --ini /usr/local/etc/elevennote.ini"
    depends_on:
      - db
    volumes:
      - ./src:/src
      - ./deploy/uwsgi:/usr/local/etc/
    expose:
      - "8000"
  db:
    image: postgres:latest
```

```
container_name: ps01
```

```
$ mkdir -p deploy/uwsgi
$ touch deploy/uwsgi/elevennote.ini
```

```
deploy/uwsgi/elevennote.ini
```

```
[uwsgi]
chdir = /src
module = config.wsgi:application
master = true
processes = 5
socket = 0.0.0.0:8000
vacuum = true
die-on-term = true
env = DJANGO_SETTINGS_MODULE=config.settings.local
```

```
$ mkdir deploy/nginx
$ touch deploy/nginx/elevennote.conf
```

```
deploy/nginx/elevennote.conf
```

```
upstream web {
    ip_hash;
    server web:8000;
}

server {
    location / {
        include uwsgi_params;
        uwsgi_pass web;
    }
    listen 8000;
    server_name localhost;
}
```

На текущий момент структура вашего проекта должна быть следующей:

```
.
├── Dockerfile
├── deploy
│   ├── nginx
│   │   └── elevennote.conf
│   └── uwsgi
│       └── elevennote.ini
├── docker-compose.yml
└── requirements
```

```
|   |— base.txt
|   |— dev.txt
|   |— prod.txt
|   |
|   └─ src
|       |— config
|       |   |— __init__.py
|       |   |— settings
|       |   |   |— __init__.py
|       |   |   |— base.py
|       |   |   |— local.py
|       |   |   |— production.py
|       |   |   └─ settings.ini
|       |— urls.py
|       └─ wsgi.py
└─ manage.py
```

### Замечание

Если на данном этапе возникнет ошибка `UnicodeDecodeError`, выполните команду `rm -rf ~/.docker/config.json` для ее устранения.

```
$ docker-compose build
$ docker-compose up
```

Теперь приложение запущено и вы можете обратиться к нему по адресу <http://localhost:8000>, где должны увидеть стандартное приветствие Django:

**django**

[View release notes for Django 2.0](#)



**The install worked successfully! Congratulations!**

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



**Django Documentation**  
Topics, references, & how-to's



**Tutorial: A Polling App**  
Get started with Django



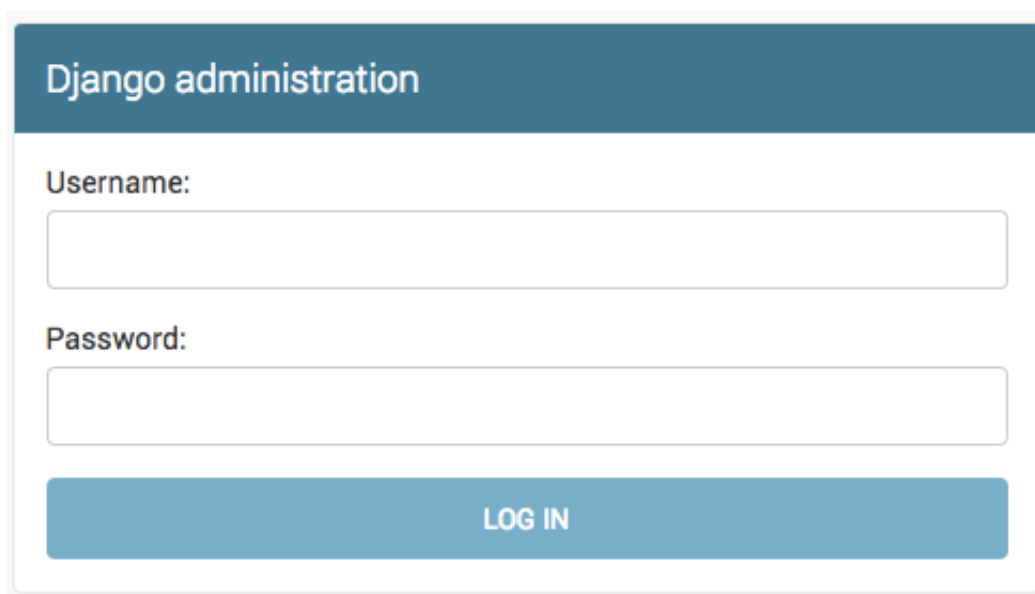
**Django Community**  
Connect, get help, or contribute

На текущий момент осталась одна нерешенная проблема - обработка статических файлов (css/js/и т.д.). Если пройти по адресу <http://localhost:8000/admin>, то мы увидим форму без применения стилей:

## Django administration

Username:   
Password:

Если же обработка статических файлов настроена верно, то мы должны увидеть следующую форму:



The image shows a styled version of the Django administration login page. It features a dark blue header with the text 'Django administration'. Below the header, there are two input fields for 'Username:' and 'Password:'. At the bottom, there is a large blue button with the text 'LOG IN' in white capital letters.

### Подсказка

Если у вас пока не получается справиться с этой проблемой, то вы можете найти решение в репозитории для этой работы.

**Задание:** Решите обозначенную проблему путем внесения соответствующих изменений в `docker-compose.yml`, `deploy/nginx/elevennote.conf` и `src/config/settings/base.py`.

На текущий момент нам больше не нужно виртуальное окружение, так как эту часть функций, а именно изоляции окружения, на себя взял докер.

Структура вашего проекта должна выглядеть следующим образом:

```
$ tree -L 2
.
├── Dockerfile
├── deploy
│   ├── nginx
│   └── uwsgi
├── docker-compose.yml
├── requirements
│   ├── base.txt
│   └── dev.txt
```



```
|   └─ prod.txt
|   └─ src
|       └─ config
|       └─ manage.py
|   └─ static
|       └─ admin
```

## Создание нового приложения

Создадим новое приложение `notes` (заметки) и добавим его в `INSTALLED_APPS`:

### Замечание

Убедитесь, что в папке `src` появилась папка `notes`, в которой нужно располагать все файлы, касающиеся приложения `notes`.

```
$ docker-compose run web python manage.py startapp notes
```

```
src/config/settings/base.py
```

```
INSTALLED_APPS = [
    # ...
    'notes',
]
```

Создадим первую модель `Note` со следующими полями:

- заголовок (`title`) с ограничением 200 символов;
- текст заметки (`body`);
- и датой создания (`pub_date`).

```
src/notes/models.py
```

```
class Note(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()
    pub_date = models.DateTimeField('date published', auto_now_add=True)

    def __str__(self):
        return self.title
```

И применим внесенные изменения к БД:

### Замечание

Миграцию необходимо выполнять всякий раз, когда изменяется структура базы данных (добавление/удаление полей, редактирование их свойств и т.д.), но при перезапуске контейнера `web` это будет происходить автоматически.

### Замечание

При необходимости вы можете перезапустить контейнер `web` командой `docker-compose restart web`.

```
$ docker-compose run web python manage.py makemigrations notes
$ docker-compose run web python manage.py migrate
$ docker-compose restart web
```

src/notes/tests.py

```
from django.test import TestCase

from notes.models import Note

class NoteModelTests(TestCase):

    def test_can_create_a_new_note(self):
        note = Note.objects.create(title='Note title', body='Note body')
        self.assertTrue(note)

    def test_string_representation(self):
        note = Note.objects.create(title='Note title', body='Note body')
        self.assertEqual(str(note), 'Note title')
```

```
$ docker-compose run web python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
-----
Ran 2 tests in 0.009s

OK
Destroying test database for alias 'default'...
```

```
$ mkdir src/notes/tests
$ mv src/notes/tests.py src/notes/tests/test_models.py
$ touch src/notes/tests/__init__.py
```

Ваши тесты должны по прежнему проходить.

```
$ echo "coverage==4.5.1" >> requirements/dev.txt
```

```
$ docker-compose run web coverage run --source='.' manage.py test notes
$ docker-compose run web coverage report
```

Name	Stmts	Miss	Cover
config/__init__.py	0	0	100%

config/settings/__init__.py	0	0	100%
config/settings/base.py	20	0	100%
config/settings/local.py	4	0	100%
config/settings/production.py	0	0	100%
config/urls.py	3	0	100%
config/wsgi.py	4	4	0%
manage.py	9	2	78%
notes/__init__.py	0	0	100%
notes/admin.py	1	0	100%
notes/apps.py	3	3	0%
notes/migrations/0001_initial.py	5	0	100%
notes/migrations/__init__.py	0	0	100%
notes/models.py	7	0	100%
notes/test_models.py	9	0	100%
notes/tests/__init__.py	0	0	100%
notes/views.py	1	1	0%
<hr/>			
TOTAL	66	10	85%

Добавим метод `was_published_recently`, который определяет была ли заметка опубликована за последние 24 часа, в модель `Note`:

src/notes/models.py

```
from django.db import models
from django.utils import timezone

class Note(models.Model):
    # ...

    def was_published_recently(self):
        return self.pub_date >= timezone.now() - timezone.timedelta(days=1)
```

Напишем тест, который будет проверять, что отложенная заметка не является недавно опубликованной:

src/notes/tests/test\_models.py

```

from django.test import TestCase
from django.utils import timezone

from notes.models import Note

class NoteModelTests(TestCase):
    # ...

    def test_was_published_recently(self):
        time = timezone.now() + timezone.timedelta(days=30)
        future_note = Note(pub_date=time)
        self.assertEqual(future_note.was_published_recently(), False)

```

```

$ docker-compose run web python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..F

=====
FAIL: test_was_published_recently (notes.tests.test_models.NoteMethodTests)
-----
Traceback (most recent call last):
  File "/src/notes/tests/test_models.py", line 14, in test_was_published_recently
    self.assertEqual(future_note.was_published_recently(), False)
AssertionError: True != False

-----

Ran 3 tests in 0.018s

FAILED (failures=1)
Destroying test database for alias 'default'...

```

Похоже была допущена ошибка в методе `was_published_recently`, давайте исправим ее:

`src/notes/models.py`

```

# ...
def was_published_recently(self):
    now = timezone.now()
    return now - timezone.timedelta(days=1) <= self.pub_date <= now

```

И снова запустим тесты:

```
$ docker-compose run web python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 3 tests in 0.012s

OK
Destroying test database for alias 'default'...
```

## Работа с панелью администратора

Создадим суперпользователя, чтобы иметь возможность заходить в панель администратора. Для этого подключимся к контейнеру:

```
$ docker exec -it dg01 bash
root@67a3bcc5c881:/src# python manage.py createsuperuser
Username (leave blank to use 'root'):
Email address:
Password:
Password (again):
Superuser created successfully.
root@67a3bcc5c881:/src# exit
```

Зарегистрируем созданную модель в панели администратора:

```
src/notes/admin.py
```

```
from django.contrib import admin
from .models import Note

admin.site.register(Note)
```

Теперь откройте панель администратора <http://localhost:8000/admin>. Вы должны увидеть новый раздел **Notes**, в котором вы можете создавать, редактировать и удалять заметки.



Сделаем отображение заметки в панели администратора более дружелюбным, отображая название заметки, дату публикации и проверку на то, была ли заметка опубликована в течении последних 24-х часов:

```
src/notes/admin.py
```

### Замечание

Узнать больше о расширении панели администратора можно в [официальной документации](#).

```
class NoteAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date', 'was_published_recently')
    list_filter = ['pub_date']

admin.site.register(Note, NoteAdmin)
```

<input type="checkbox"/> TITLE	DATE PUBLISHED	WAS PUBLISHED RECENTLY
<input type="checkbox"/> Заметка созданная в админке	Jan. 8, 2018, 4:18 p.m.	True

## Добавление выверов

src/config/urls.py

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('notes/', include('notes.urls', namespace='notes')),
    path('admin/', admin.site.urls),
]
```

```
$ touch src/notes/urls.py
```

src/notes/urls.py

```
from django.urls import path

from . import views

app_name = 'notes'

urlpatterns = [
    path('', views.index, name='index'),
    path('<int:note_id>', views.detail, name='detail'),
]
```

src/notes/views.py

```
from django.shortcuts import get_object_or_404, render

from .models import Note

def index(request):
    latest_note_list = Note.objects.order_by('-pub_date')[:5]
```

```

context = {
    'latest_note_list': latest_note_list,
}
return render(request, 'notes/index.html', context)

def detail(request, note_id):
    note = get_object_or_404(Note, pk=note_id)
    return render(request, 'notes/detail.html', {'note': note})

```

```
$ mkdir -p src/templates/notes
```

```
src/config/settings/base.py
```

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [root('templates')],
        # ...
    },
]

```

```
src/templates/notes/index.html
```

```
{% raw %}
```

```

{% if latest_note_list %}
    <ul>
    {% for note in latest_note_list %}
        <li><a href="{% url 'notes:detail' note.id %}">{{ note.title }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No notes available.</p>
{% endif %}

```

```
{% endraw %}
```

```
src/templates/notes/detail.html
```

```
{% raw %}
```

```

<h1>{{ note.title }}</h1>
<p>{{ note.body }}</p>

```

```
{% endraw %}
```

```
src/notes/tests/tests_views.py
```

**Замечание**

Обратите внимание на использование f-strings, которые были введены в Python 3.6.

```
from django.test import TestCase
from django.urls import reverse, resolve

from notes.models import Note
from notes.views import index, detail

class IndexTests(TestCase):

    def setUp(self):
        self.note = Note.objects.create(
            title="Note title", body="Note description")
        url = reverse('notes:index')
        self.response = self.client.get(url)

    def test_index_view_status_code(self):
        self.assertEqual(self.response.status_code, 200)

    def test_index_url_resolves_index_view(self):
        view = resolve('/notes/')
        self.assertEqual(view.func, index)

    def test_index_view_contains_link_to_details_page(self):
        note_detail_url = reverse('notes:detail', kwargs={
            'note_id': self.note.pk})
        self.assertContains(self.response, f'href="{note_detail_url}"')

class DetailTests(TestCase):

    def setUp(self):
        self.note = Note.objects.create(
            title="Note title", body="Note description")
        url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
        self.response = self.client.get(url)

    def test_detail_view_status_code(self):
        self.assertEqual(self.response.status_code, 200)

    def test_detail_url_resolves_detail_view(self):
        view = resolve(f'/notes/{self.note.pk}/')
        self.assertEqual(view.func, detail)
```



```
$ docker-compose run web python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 8 tests in 0.119s

OK
Destroying test database for alias 'default'...
```

## Создание кастомной модели пользователя

### Замечание

Не забудьте добавить созданное приложение в `INSTALLED_APPS` в файле `src/config/settings/base.py`.

Создадим новое приложение `accounts`, которое будет отвечать за хранение информации о пользователях. Также добавим возможность регистрации и авторизации.

Пример создания кастомной модели пользователя можно найти в [официальной документации Django](#).

```
$ docker-compose run web python manage.py startapp accounts
```

```
src/accounts/models.py
```

```
from django.db import models
from django.contrib.auth.base_user import AbstractBaseUser, BaseUserManager
from django.contrib.auth.models import PermissionsMixin

class UserManager(BaseUserManager):
    use_in_migrations = True

    def create_user(self, email, password=None):
        """
        Creates and saves a User with the given email and password.
        """
        if email is None:
            raise ValueError('User must have an email address.')
        user = self.model(email=self.normalize_email(email))
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password=None):
        """
        Creates and saves a superuser with the given email and password.
        """
```

```

        if password is None:
            raise ValueError('Superusers must have a password.')
        user = self.create_user(email, password)
        user.is_superuser = True
        user.is_staff = True
        user.save(using=self._db)
        return user

class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(db_index=True, unique=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    USERNAME_FIELD = 'email'

    objects = UserManager()

    def __str__(self):
        return self.email

```

src/config/settings/base.py

```

# ...
AUTH_USER_MODEL = 'accounts.User'

```

```

$ mkdir src/accounts/tests
$ mv src/accounts/tests.py src/accounts/tests/test_models.py
$ touch src/accounts/tests/__init__.py

```

src/accounts/tests/test\_models.py

```

from django.test import TestCase
from django.contrib.auth import get_user_model

User = get_user_model()

class TestUserModel(TestCase):

    def test_can_create_user(self):
        user = User.objects.create_user(email='user@example.com')
        self.assertTrue(user)

    def test_can_create_superuser(self):
        superuser = User.objects.create_superuser(
            email='superuser@example.com',
            password='secret'
        )

```

```
self.assertTrue(superuser)
```

```
def test_string_representation(self):  
    user = User.objects.create_user(email='user@example.com')  
    self.assertEqual(str(user), 'user@example.com')
```

```
...  
dg01 | django.db.migrations.exceptions.InconsistentMigrationHistory: Migration  
admin.0001_initial is applied before its dependency accounts.0001_initial on database  
'default'.
```

## Добавление авторизации

```
src/config/urls.py
```

### Замечание

Обратите внимание, что мы используем `RedirectView` для перенаправления запроса с `http://localhost:8000/` на `http://localhost:8000/notes/`.

```
#...  
from django.views.generic import RedirectView  
  
urlpatterns = [  
    # Handle the root url.  
    path('', RedirectView.as_view(url='notes/'), name='index'),  
  
    # Accounts app  
    path('accounts/', include('accounts.urls', namespace='accounts')),  
  
    # Notes app  
    path('notes/', include('notes.urls', namespace='notes')),  
  
    # Admin  
    path('admin/', admin.site.urls),  
]
```

```
$ touch src/accounts/urls.py
```

```
src/accounts/urls.py
```

```

from django.urls import path
from django.contrib.auth import views

app_name = 'accounts'

urlpatterns = [
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
]

```

```

$ mkdir src/templates/registration
$ touch src/templates/registration/login.html

```

```

src/templates/registration/login.html
{% raw %}

```

```

<form action="{% url 'accounts:login' %}" method="post" accept-charset="utf-8">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="hidden" name="next" value="{{ next }}" />
    <input class="button small" type="submit" value="Submit"/>
</form>

```

```

{% endraw %}

```

Email

Password

Отмечаем соответствующие страницы доступными только после авторизации добавлением декоратора `@login_required`:

```

src/notes/views.py

```

```

# ...
from django.contrib.auth.decorators import login_required

@login_required
def index(request):
    # ...

@login_required
def detail(request):
    # ...

```

```

src/notes/tests/test_views.py

```

```
# ...
from django.contrib.auth import get_user_model

User = get_user_model()

class IndexTests(TestCase):

    def setUp(self):
        self.user = User.objects.create_user(
            email="user@example.com", password="secret")
        self.client.login(email="user@example.com", password="secret")
        # ...

class DetailTests(TestCase):

    def setUp(self):
        self.user = User.objects.create_user(
            email="user@example.com", password="secret")
        self.client.login(email="user@example.com", password="secret")
        # ...
```

Далее предлагается загрузить набор стилевых компонентов [bootstrap](#) и поместить их в папку со статичными файлами.

```
$ mkdir src/static
$ cd src/static
$ wget https://github.com/twbs/bootstrap/releases/download/v4.0.0-beta.3/bootstrap-4.0.0-beta.3-dist.zip
$ unzip bootstrap-4.0.0-beta.3-dist.zip -d bootstrap
$ rm -rf bootstrap-4.0.0-beta.3-dist.zip
$ cd -
```

src/config/settings/base.py

```
# ...
STATIC_URL = '/static/'
STATIC_ROOT = '/static'
STATICFILES_DIRS = [
    root('static'),
]
```

```
$ docker exec -it dg01 bash
root@67a3bcc5c881:/src# pip install django-widget-tweaks
root@67a3bcc5c881:/src# exit
$ echo "django-widget-tweaks==1.4.2" >> requirements/base.txt
```

src/config/settings/base.py

```
INSTALLED_APPS = [  
    # ...  
    'widget_tweaks',  
]
```

```
$ cd src/templates  
$ wget https://github.com/Dementiy/pybook-  
assignments/raw/elevennote/homework07/elevennote/templates_archives/templates01.zip  
$ unzip templates01.zip # If prompted to replace, say (Y)es  
$ cp -r templates01/* .  
$ rm -rf templates01 templates01.zip  
$ cd ../..
```

## ElevenNote

Email:

Password:

Submit

[Create a new user](#)

Добавляем возможность регистрации.

src/accounts/views.py

```
from django.contrib.auth import authenticate, login  
from django.views.generic import FormView  
  
from .forms import UserCreationForm  
  
class RegisterView(FormView):  
    template_name = 'registration/register.html'  
    form_class = UserCreationForm  
    success_url = '/'  
  
    def form_valid(self, form):  
        form.save()
```

```
email = self.request.POST['email']
password = self.request.POST['password1']

user = authenticate(email=email, password=password)
login(self.request, user)
return super(RegisterView, self).form_valid(form)
```

```
$ touch src/accounts/forms.py
```

```
src/accounts/forms.py
```

```
from django import forms

from accounts.models import User

class UserCreationForm(forms.ModelForm):
    password1 = forms.CharField(label='Password', widget=forms.PasswordInput)
    password2 = forms.CharField(label='Password confirmation', widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ('email',)

    def clean_password2(self):
        password1 = self.cleaned_data.get("password1")
        password2 = self.cleaned_data.get("password2")
        if password1 and password2 and password1 != password2:
            raise forms.ValidationError("Passwords don't match")
        return password2

    def save(self, commit=True):
        user = super().save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        if commit:
            user.save()
        return user
```

```
src/accounts/urls.py
```

```

from django.urls import path, reverse_lazy
from django.contrib.auth import views as auth_views

from .views import RegisterView

app_name = 'accounts'

urlpatterns = [
    path('login/', auth_views.login, name='login'),
    path('logout/', auth_views.logout, {"next_page" : reverse_lazy('accounts:login')},
name='logout'),
    path('register/', RegisterView.as_view(), name='register'),
]

```

Далее настраивается адрес, на который будет осуществлен переход по факту успешной авторизации/регистрации.

```
config/settings/base.py
```

```

# ...
LOGIN_REDIRECT_URL = '/'

```

## ElevenNote

Email:

Password:

Password confirmation:

Submit

Already have a user? [Sign in](#)

```

src/templates/registration/login.html
{% raw %}

```



```
<a href="{% url 'accounts:register' %}">Create a new user</a>
```

```
{% endraw %}
```

```
accounts/tests/test_views.py
```

```
from django.test import TestCase
from django.urls import reverse, resolve
from django.contrib.auth import get_user_model

from accounts.views import RegisterView
from accounts.forms import UserCreationForm

User = get_user_model()

class RegisterViewTests(TestCase):

    def setUp(self):
        url = reverse('accounts:register')
        self.response = self.client.get(url)

    def test_signup_status_code(self):
        self.assertEqual(self.response.status_code, 200)

    def test_signup_url_resolves_signup_view(self):
        view = resolve('/accounts/register/')
        self.assertEqual(view.func.view_class, RegisterView)

    def test_csrf(self):
        self.assertContains(self.response, 'csrfmiddlewaretoken')

    def test_contains_form(self):
        form = self.response.context.get('form')
        self.assertIsInstance(form, UserCreationForm)

    def test_form_inputs(self):
        self.assertContains(self.response, '<input', 6)
        self.assertContains(self.response, 'type="email"', 1)
        self.assertContains(self.response, 'type="password"', 2)

class SuccessfulSignUpTests(TestCase):

    def setUp(self):
        url = reverse('accounts:register')
        data = {
            'email': 'user@example.com',
            'password1': 'secret',
```

```

        'password2': 'secret',
    }
    self.response = self.client.post(url, data)
    self.notes_page = reverse('notes:index')
    self.index_page = reverse('index')

    def test_redirects_to_index_page(self):
        self.assertRedirects(self.response, self.index_page,
                             target_status_code=302)

    def test_user_creation(self):
        self.assertTrue(User.objects.exists())

    def test_user_authentication(self):
        response = self.client.get(self.notes_page)
        user = response.context.get('user')
        self.assertTrue(user.is_authenticated)

class InvalidSingUpTests(TestCase):

    def setUp(self):
        url = reverse('accounts:register')
        self.response = self.client.post(url, {})

    def test_signup_status_code(self):
        self.assertEqual(self.response.status_code, 200)

    def test_form_errors(self):
        form = self.response.context.get('form')
        self.assertTrue(form.errors)

    def test_dont_create_user(self):
        self.assertFalse(User.objects.exists())

```

Добавим в модель заметки информацию об авторе - пользователе, создавшем ее - путем внесения внешнего ключа.

src/notes/models.py

```

# ...
from django.contrib.auth import get_user_model

User = get_user_model()

class Note(models.Model):
    # ...
    owner = models.ForeignKey(User, related_name='notes', on_delete=models.CASCADE)

```

Как говорилось ранее, вследствие изменения структуры базы данных необходимо выполнить миграцию.

**TODO:** выбор значения по умолчанию

```
$ docker exec -it dg01 bash
root@67a3bcc5c881:/src# python manage.py makemigrations
root@67a3bcc5c881:/src# python manage.py migrate
root@67a3bcc5c881:/src# exit
```

В выводимый список заметок добавим сортировку по убыванию даты публикации и ограничим выводимое количество (5 шт.). Позже можно убрать количественное ограничение для возможности страничного просмотра имеющихся заметок.

src/notes/views.py

```
latest_note_list = Note.objects.filter(owner=request.user).order_by('-pub_date')[:5]

# ...

note = get_object_or_404(Note, pk=note_id, owner=request.user)
```

src/notes/admin.py

```
list_display = ('title', 'owner', 'pub_date', 'was_published_recently')
```

src/notes/tests/test\_views.py

```
from django.test import TestCase
from django.urls import reverse, resolve
from django.contrib.auth import get_user_model

import datetime

from notes.models import Note
from notes.views import index, detail

User = get_user_model()

class IndexTests(TestCase):

    def setUp(self):
        self.test_user1 = User.objects.create_user(
            email="test_user1@example.com",
            password="secret")
        self.test_user2 = User.objects.create_user(
            email="test_user2@example.com",
```

```

        password="secret")

    now = datetime.datetime.now()
    self.notes = []
    self.n = 5
    for i in range(self.n):
        self.notes.append(Note.objects.create(
            title=f"Note title {i}",
            body="Note description",
            pub_date=now + datetime.timedelta(days=i),
            owner=self.test_user1))

    def test_redirect_if_not_logged_in(self):
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
        self.assertRedirects(response, "/accounts/login/?next=/notes/")

    def test_index_view_status_code(self):
        self.client.login(email="test_user1@example.com", password="secret")
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
        self.assertEqual(response.status_code, 200)

    def test_index_url_resolves_index_view(self):
        view = resolve('/notes/')
        self.assertEqual(view.func, index)

    def test_index_view_contains_link_to_details_page(self):
        self.client.login(email="test_user1@example.com", password="secret")
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
        for note in self.notes:
            note_detail_url = reverse('notes:detail',
                                      kwargs={'note_id': note.pk})
            self.assertContains(response, f'href="{note_detail_url}"')

    def test_notes_ordered_by_pub_dates(self):
        self.client.login(email="test_user1@example.com", password="secret")
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
        notes = response.context["latest_note_list"]
        self.assertEqual(len(notes), self.n)

        pub_date = notes[0].pub_date
        for note in notes[1:]:
            self.assertTrue(pub_date >= note.pub_date)
            pub_date = note.pub_date

    def test_only_owned_notes_in_list(self):

```

```

self.client.login(email="test_user2@example.com", password="secret")
index_page_url = reverse('notes:index')
response = self.client.get(index_page_url)
notes = response.context["latest_note_list"]
self.assertEqual(len(notes), 0)

class DetailTests(TestCase):

    def setUp(self):
        self.test_user1 = User.objects.create_user(
            email="test_user1@example.com",
            password="secret")
        self.test_user2 = User.objects.create_user(
            email="test_user2@example.com",
            password="secret")
        self.note = Note.objects.create(
            title="Note title", body="Note description", owner=self.test_user1)

    def test_redirect_if_not_logged_in(self):
        detail_page_url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
        response = self.client.get(detail_page_url)
        self.assertRedirects(response, f"/accounts/login/?next=/notes/{self.note.pk}/")

    def test_detail_view_status_code(self):
        self.client.login(email="test_user1@example.com", password="secret")
        url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)

    def test_detail_url_resolves_detail_view(self):
        view = resolve(f'/notes/{self.note.pk}/')
        self.assertEqual(view.func, detail)

    def test_only_owner_can_see_detail_page(self):
        self.client.login(email="test_user2@example.com", password="secret")
        url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
        response = self.client.get(url)
        self.assertEqual(response.status_code, 404)

```

src/notes/tests/test\_models.py

```

from django.test import TestCase
from django.utils import timezone
from django.contrib.auth import get_user_model

from notes.models import Note

User = get_user_model()

```

```

class NoteModelTests(TestCase):

    def setUp(self):
        self.test_user = User.objects.create_user(
            email="test_user@example.com",
            password="secret")

    def test_can_create_a_new_note(self):
        note = Note.objects.create(title='Note title', body='Note body', owner=self.test_user)
        self.assertTrue(note)

    def test_string_representation(self):
        note = Note.objects.create(title='Note title', body='Note body', owner=self.test_user)
        self.assertEqual(str(note), 'Note title')

    def test_was_published_recently(self):
        time = timezone.now() + timezone.timedelta(days=30)
        future_note = Note(pub_date=time)
        self.assertEqual(future_note.was_published_recently(), False)

```

## Class-based views

notes/views.py

```

from django.utils.decorators import method_decorator
from django.contrib.auth.decorators import login_required
from django.views.generic import ListView, DetailView

from .models import Note

class NoteList(ListView):
    paginate_by = 5
    template_name = 'notes/index.html'
    context_object_name = 'latest_note_list'

    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super(NoteList, self).dispatch(*args, **kwargs)

    def get_queryset(self):
        return Note.objects.filter(owner=self.request.user).order_by('-pub_date')

class NoteDetail(DetailView):
    model = Note
    template_name = 'notes/detail.html'

```

```

context_object_name = 'note'

@method_decorator(login_required)
def dispatch(self, *args, **kwargs):
    return super(NoteDetail, self).dispatch(*args, **kwargs)

def get_queryset(self):
    return Note.objects.filter(owner=self.request.user)

```

notes/urls.py

```

from django.urls import path

from .views import NoteList, NoteDetail

app_name = 'notes'

urlpatterns = [
    path('', NoteList.as_view(), name='index'),
    path('<int:pk>', NoteDetail.as_view(), name='detail'),
]

```

src/tests/test\_views.py

```

-from notes.views import index, detail
+from notes.views import NoteList, NoteDetail

User = get_user_model()

@@ -43,7 +43,7 @@ class IndexTests(TestCase):

    def test_index_url_resolves_index_view(self):
        view = resolve('/notes/')
        - self.assertEqual(view.func, index)
        + self.assertEqual(view.func.view_class, NoteList)

    def test_index_view_contains_link_to_details_page(self):
        self.client.login(email="test_user1@example.com", password="secret")
@@ -51,7 +51,7 @@ class IndexTests(TestCase):
        response = self.client.get(index_page_url)
        for note in self.notes:
            note_detail_url = reverse('notes:detail',
            - kwargs={'note_id': note.pk})
            + kwargs={'pk': note.pk})
            self.assertContains(response, f'href="{note_detail_url}"')

    def test_notes_ordered_by_pub_dates(self):
@@ -87,23 +87,23 @@ class DetailTests(TestCase):

```

```

        title="Note title", body="Note description", owner=self.test_user1)

    def test_redirect_if_not_logged_in(self):
-         detail_page_url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
+         detail_page_url = reverse('notes:detail', kwargs={'pk': self.note.pk})
        response = self.client.get(detail_page_url)
        self.assertRedirects(response, f"/accounts/login/?next=/notes/{self.note.pk}/")

    def test_detail_view_status_code(self):
        self.client.login(email="test_user1@example.com", password="secret")
-         url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
+         url = reverse('notes:detail', kwargs={'pk': self.note.pk})
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)

    def test_detail_url_resolves_detail_view(self):
        view = resolve(f'/notes/{self.note.pk}/')
-         self.assertEqual(view.func, detail)
+         self.assertEqual(view.func.view_class, NoteDetail)

    def test_only_owner_can_see_detail_page(self):
        self.client.login(email="test_user2@example.com", password="secret")
-         url = reverse('notes:detail', kwargs={'note_id': self.note.pk})
+         url = reverse('notes:detail', kwargs={'pk': self.note.pk})
        response = self.client.get(url)
        self.assertEqual(response.status_code, 404)

```

```

@@ -22,7 +22,8 @@ class IndexTests(TestCase):

    now = datetime.datetime.now()
    self.notes = []
-     self.n = 5
+     self.n = 10
+     self.paginate_by = 5
    for i in range(self.n):
        self.notes.append(Note.objects.create(
            title=f"Note title {i}",

@@ -49,7 +50,7 @@ class IndexTests(TestCase):
        self.client.login(email="test_user1@example.com", password="secret")
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
-         for note in self.notes:
+         for note in response.context["latest_note_list"]:
            note_detail_url = reverse('notes:detail',
                kwargs={'pk': note.pk})
            self.assertContains(response, f'href="{note_detail_url}"')

@@ -59,7 +60,7 @@ class IndexTests(TestCase):
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)

```



```

        notes = response.context["latest_note_list"]
-       self.assertEqual(len(notes), self.n)
+       self.assertEqual(len(notes), self.paginate_by)

        pub_date = notes[0].pub_date
        for note in notes[1:]:
@@ -73,6 +74,15 @@ class IndexTests(TestCase):
        notes = response.context["latest_note_list"]
        self.assertEqual(len(notes), 0)

+       def test_pagination_is_five(self):
+           self.client.login(email="test_user1@example.com", password="secret")
+           index_page_url = reverse('notes:index')
+           response = self.client.get(index_page_url)
+           notes = response.context["latest_note_list"]
+           self.assertTrue('is_paginated' in response.context)
+           self.assertTrue(response.context['is_paginated'] == True)
+           self.assertEqual(len(notes), self.paginate_by)

```

### Замечание

В [руководстве](#) от Six Feet Up объясняется как создать собственный миксин, который бы решал аналогичную задачу.

notes/views.py

```

from django.contrib.auth.mixins import LoginRequiredMixin

class NoteList(LoginRequiredMixin, ListView):
    # ...

class NoteDetail(LoginRequiredMixin, DetailView):
    # ...

```

notes/forms.py

```

from django import forms

from .models import Note

class NoteForm(forms.ModelForm):

    class Meta:
        model = Note
        fields = ['title', 'body']

```

notes/views.py

```

from django.views.generic import ListView, DetailView, CreateView
from django.utils import timezone
from django.urls import reverse_lazy

from .forms import NoteForm
...

class NoteCreate(LoginRequiredMixin, CreateView):
    form_class = NoteForm
    template_name = 'notes/form.html'
    success_url = reverse_lazy('notes:index')

    def form_valid(self, form):
        form.instance.owner = self.request.user
        form.instance.pub_date = timezone.now()
        return super(NoteCreate, self).form_valid(form)

```

src/notes/urls.py

```

path('new/', NoteCreate.as_view(), name='create'),

```

Добавляем шаблон формы, используемый для создания и редактирования заметок. Убедитесь в наличии шаблона `base.html`, от которого ряд других шаблонов будет наследоваться. Используемая логика: `base.html` реализует базовый интерфейс в виде шапки ресурса, в которой отображается имя авторизованного пользователя и кнопка выхода, в то время как наследники реализуют интерфейс отдельного модуля в блоке `content`

src/templates/notes/form.html

```

{% raw %}

```

```

{% extends "base.html" %}

{% block content %}

<form action="{% url 'notes:create' %}" method="post" accept-charset="utf-8">
    {% csrf_token %}
    {% include 'includes/form.html' %}
    <input type="hidden" name="next" value="{{ next }}" />
    <input class="btn btn-success" type="submit" value="Submit"/>
</form>

{% endblock %}

```

```

{% endraw %}

```

Title:

Body:

Добавим возможность создания новой заметки в интерфейс общего списка заметок.

```
src/templates/notes/index.html
```

```
{% raw %}
```

```
<a href="{% url 'notes:create' %}">Create a new note</a>
```

```
{% endraw %}
```

```
src/notes/tests/test_forms.py
```

```
from django.test import TestCase
from django.contrib.auth import get_user_model

from notes.models import Note
from notes.forms import NoteForm

User = get_user_model()

class NoteFormTests(TestCase):
```

```

def test_form_save(self):
    data = {'title': 'Note Title', 'body': 'Note body'}
    form = NoteForm(data=data)
    self.assertTrue(form.is_valid())

    user = User.objects.create_user(email='user@example.com', password='secret')
    form.instance.owner = user
    note = form.save()
    self.assertEqual(note, Note.objects.first())

```

src/notes/tests/test\_views.py

```

# ...
class IndexTests(TestCase):
    # ...
    def test_index_view_contains_link_to_create_page(self):
        self.client.login(email="test_user1@example.com", password="secret")
        index_page_url = reverse('notes:index')
        response = self.client.get(index_page_url)
        self.assertContains(response, 'href="{0}"'.format(reverse('notes:create'))))

# ...
class CreateViewTest(TestCase):

    def setUp(self):
        self.user = User.objects.create_user(
            email="user@example.com",
            password="secret")

    def test_redirect_if_not_logged_in(self):
        create_page_url = reverse('notes:create')
        response = self.client.get(create_page_url)
        self.assertRedirects(response, f"/accounts/login/?next=/notes/new/")

    def test_create_view_status_code(self):
        self.client.login(email="user@example.com", password="secret")
        create_page_url = reverse('notes:create')
        response = self.client.get(create_page_url)
        self.assertEqual(response.status_code, 200)

    def test_uses_correct_template(self):
        self.client.login(email="user@example.com", password="secret")
        create_page_url = reverse('notes:create')
        response = self.client.get(create_page_url)
        self.assertTemplateUsed(response, 'notes/form.html')

    def test_redirects_to_index_page(self):
        self.client.login(email="user@example.com", password="secret")
        index_page_url = reverse('notes:index')

```

```

create_page_url = reverse('notes:create')
response = self.client.post(create_page_url,
    {'title': 'Note Title', 'body': 'Note body'})
self.assertRedirects(response, index_page_url)

def test_form_success(self):
    self.client.login(email="user@example.com", password="secret")
    create_page_url = reverse('notes:create')
    self.client.post(create_page_url,
        {'title': 'Note title', 'body': 'Note body'})
    note = Note.objects.first()
    self.assertEqual(note.title, 'Note title')
    self.assertEqual(note.body, 'Note body')
    self.assertEqual(note.owner, self.user)
    self.assertTrue(note.was_published_recently())

def test_form_invalid(self):
    self.client.login(email="user@example.com", password="secret")
    create_page_url = reverse('notes:create')
    response = self.client.post(create_page_url,
        {'title': '', 'body': ''})
    self.assertFormError(response, "form", "title", "This field is required.")
    self.assertFormError(response, "form", "body", "This field is required.")

```

Добавим возможность постраничного просмотра имеющихся заметок.

```

src/template/notes/index.html
{% raw %}

```

```

{% if is_paginated %}
<div class="pagination">
    <span class="step-links">
        {% if page_obj.has_previous %}
            <a href="?page={{ page_obj.previous_page_number }}">previous</a>
        {% endif %}

        <span class="current">
            Page {{ page_obj.number }} of {{ page_obj.paginator.num_pages }}.
        </span>

        {% if page_obj.has_next %}
            <a href="?page={{ page_obj.next_page_number }}">next</a>
        {% endif %}
    </span>
</div>
{% endif %}

```

```

{% endraw %}

```

Для возможности обширного редактирования заметок (функции для выделения текста, использование различных шрифтов, вставка ссылок, изображений и т. д.) предлагается к использованию модуль [django-wysiwyg](#).

```
$ docker exec -it dg01 bash
root@67a3bcc5c881:/src# python -m pip install django-wysiwyg
root@67a3bcc5c881:/src# exit
$ echo "django-wysiwyg==0.8.0" >> requirements/base.txt
```

src/config/settings/base.py

```
INSTALLED_APPS = [
    # ...
    'django_wysiwyg',
    # ...
]

#...
DJANGO_WYSIWYG_FLAVOR = 'ckeditor'
```

Далее нужно загрузить предпочитаемый редактор текста.

```
$ cd src/static
$ wget
https://download.cksource.com/CKEditor/CKEditor/CKEditor%204.7.2/ckeditor_4.7.2_full.zip
$ unzip ckeditor_4.7.2_full.zip
$ rm -rf ckeditor_4.7.2_full.zip
$ cd -
```

src/notes/views.py

```
from django.views.generic import (
    ListView, DetailView, CreateView, UpdateView
)

# ...

class NoteUpdate(LoginRequiredMixin, UpdateView):
    model = Note
    form_class = NoteForm
    template_name = 'notes/form.html'

    def get_queryset(self):
        return Note.objects.filter(owner=self.request.user)

    def get_success_url(self):
        return reverse('notes:update', kwargs={
```

```
        'pk': self.object.pk
    })
```

src/notes/urls.py

```
# ...
from .views import NoteList, NoteDetail, NoteCreate, NoteUpdate
# ...

path('<int:pk>/edit/', NoteUpdate.as_view(), name='update'),
```

src/templates/notes/form.html

```
{% raw %}
```

```
{% if object %}
<form action="{% url 'notes:update' object.pk %}" method="post" accept-charset="utf-8">
{% else %}
<form action="{% url 'notes:create' %}" method="post" accept-charset="utf-8">
{% endif %}
```

```
{% endraw %}
```

src/templates/notes/detail.html

```
{% raw %}
```

```
<a href="{% url 'notes:update' note.id %}">Edit</a>
```

```
{% endraw %}
```

src/templates/notes/detail.html

```
{% raw %}
```

```
<p>
  <a href="{% url 'notes:update' note.id %}">{{ note.title }}</a><br />
  {{ note.body | safe }}
</p>
<hr />
```

```
{% endraw %}
```

src/notes/mixins.py

```

from .models import Note

class NoteMixin(object):
    def get_context_data(self, **kwargs):
        context = super(NoteMixin, self).get_context_data(**kwargs)

        context.update({
            'notes': Note.objects.filter(owner=self.request.user).order_by('-pub_date'),
        })

    return context

```

src/notes/views.py

```

# ...
class NoteCreate(LoginRequiredMixin, NoteMixin, CreateView):
# ...
class NoteUpdate(LoginRequiredMixin, NoteMixin, UpdateView):
# ...

```

```

@@ -175,6 +175,12 @@ class CreateViewTest(TestCase):
    self.assertFormError(response, "form", "title", "This field is required.")
    self.assertFormError(response, "form", "body", "This field is required.")

+   def test_response_contains_notes_list(self):
+       self.client.login(email="user@example.com", password="secret")
+       create_page_url = reverse('notes:create')
+       response = self.client.get(create_page_url)
+       self.assertIn('notes', response.context)
+
    class UpdateViewTest(TestCase):

@@ -240,3 +246,12 @@ class UpdateViewTest(TestCase):
    self.assertEqual(note.body, 'Note description')
    self.assertEqual(note.owner, self.user)

+   def test_response_contains_notes_list(self):
+       self.client.login(email="user@example.com", password="secret")
+       update_page_url = reverse('notes:update', kwargs={'pk': self.note.pk})
+       response = self.client.get(update_page_url)
+       self.assertIn('notes', response.context)
+       self.assertQuerysetEqual(
+           response.context['notes'],
+           ['<Note: Note title>'])
+

```



```
$ cd src/templates
$ wget https://github.com/Dementiy/pybook-
assignments/raw/elevennote/homework07/elevennote/templates_archives/templates02.zip
$ unzip templates02.zip
$ cp -r templates02/* .
$ rm -rf templates02 templates02.zip
$ cd -
```

```
$ cd src/static
$ wget https://github.com/Dementiy/pybook-
assignments/raw/elevennote/homework07/elevennote/templates_archives/static02.zip
$ unzip static02.zip
$ rm -rf static02.zip
$ cd -
```

The screenshot displays the ElevenNote application interface. At the top, the header bar contains the application name "ElevenNote" on the left and the user status "Signed in as root. Profile | Logout" on the right. The main content area is divided into two sections. The left section, titled "Список покупок в магазине" (Shopping list in the store), contains a single note with the text "Что надо купить в магазине?" (What to buy in the store?). Below this list is a section titled "Заметка созданная в админке" (Note created in the admin panel), which contains the text "Проверка всех полей" (Check all fields). The right section, titled "Заметка созданная в админке" (Note created in the admin panel), shows a detailed view of a note. It features a rich text editor with a toolbar containing various icons for text formatting (bold, italic, underline, strikethrough, text color, background color), list creation, indentation, and other editing functions. The note content in this view is "Проверка всех полей". At the bottom of the right section, there is a "Update Note" button.

```
src/notes/views.py
```

```
# ...
from django.views.generic import (
    ListView, DetailView, CreateView, UpdateView, DeleteView
)
# ...

class NoteDelete(LoginRequiredMixin, DeleteView):
    model = Note
    success_url = reverse_lazy('notes:create')

    def get_queryset(self):
        return Note.objects.filter(owner=self.request.user)
```

```
src/notes/urls.py
```

```
# ...
from .views import NoteList, NoteDetail, NoteCreate, NoteUpdate, NoteDelete
# ...
path('<int:pk>/delete/', NoteDelete.as_view(), name='delete'),
```

Предоставление возможности удаления заметки.

```
src/templates/base.html
```

```
{% raw %}
```

```
<link rel="stylesheet" media="all" href="{% static "font-awesome/css/font-awesome.css" %}">
```

```
{% endraw %}
```

```
src/templates/notes/form.html
```

```
{% raw %}
```

```
{% if object %}
<form action="{% url 'notes:delete' object.pk %}" method="post" id="delete-note-form">
    {% csrf_token %}
    <a class="btn btn-outline-dark" id="delete-note">
        <i class="fa fa-trash" aria-hidden="true"></i>
    </a>
</form>
{% endif %}
```

```
{% endraw %}
```

```
src/notes/tests/test_views.py
```

```
# ...
class DeleteViewTest(TestCase):

    def setUp(self):
        self.test_user1 = User.objects.create_user(
            email="test_user1@example.com",
            password="secret")
        self.test_user2 = User.objects.create_user(
            email="test_user2@example.com",
            password="secret")
        self.note = Note.objects.create(
            title="Note title", body="Note description", owner=self.test_user1)

    def test_can_delete_note(self):
        self.client.login(email="test_user1@example.com", password="secret")
        delete_page_url = reverse('notes:delete', kwargs={'pk': self.note.pk})
        response = self.client.post(delete_page_url)
        self.assertEqual(Note.objects.count(), 0)
```

```

self.assertRedirects(response, reverse('notes:create'))

def test_only_owner_can_delete_note(self):
    self.client.login(email="test_user2@example.com", password="secret")
    delete_page_url = reverse('notes:delete', kwargs={'pk': self.note.pk})
    response = self.client.post(delete_page_url)
    self.assertEqual(Note.objects.count(), 1)
    self.assertEqual(response.status_code, 404)

```

## Добавляем API с помощью DRF

```

$ echo "djangorestframework==3.8.2" >> requirements/base.txt
$ echo "djangorestframework-jwt==1.11.0" >> requirements/base.txt

```

```

$ docker-compose run web python manage.py startapp api

```

src/config/settings/base.py

```

@@ -19,8 +19,10 @@ INSTALLED_APPS = [
    'django.contrib.staticfiles',
    'widget_tweaks',
    'django_wysiwyg',
+   'rest_framework',
    'notes',
    'accounts',
+   'api',
]

MIDDLEWARE = [
@@ -68,6 +70,15 @@ AUTH_PASSWORD_VALIDATORS = [
    },
]

+REST_FRAMEWORK = {
+   'DEFAULT_PERMISSION_CLASSES': (
+       'rest_framework.permissions.IsAuthenticated',
+   ),
+   'DEFAULT_AUTHENTICATION_CLASSES': (
+       'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
+   ),
+}

```

src/api/serializers.py

```

from rest_framework import serializers

from notes.models import Note

class NoteSerializer(serializers.ModelSerializer):

    class Meta:
        model = Note
        fields = ('id', 'title', 'body', 'pub_date')

```

src/api/views.py

```

from rest_framework import viewsets

from notes.models import Note
from .serializers import NoteSerializer

class NoteViewSet(viewsets.ModelViewSet):
    serializer_class = NoteSerializer
    queryset = Note.objects.all()

    def filter_queryset(self, queryset):
        queryset = Note.objects.filter(owner=self.request.user)
        return queryset

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

```

src/api/urls.py

```

from django.urls import path, include
from rest_framework_jwt.views import obtain_jwt_token
from rest_framework.routers import DefaultRouter

from .views import NoteViewSet

app_name = 'api'

router = DefaultRouter(trailing_slash=False)
router.register(r'notes', NoteViewSet)

urlpatterns = [
    path('jwt-auth/', obtain_jwt_token),
    path('', include(router.urls)),
]

```

src/config/urls.py

```
# ...
path('api/', include('api.urls', namespace='api')),
# ...
```

src/api/tests/test\_views.py

```
from django.test import TestCase
from django.urls import reverse
from django.contrib.auth import get_user_model

from rest_framework.test import APIClient

from notes.models import Note

User = get_user_model()

class NoteViewTest(TestCase):

    def setUp(self):
        self.test_user1 = User.objects.create_user(
            email="test_user1@example.com",
            password="secret")
        self.test_user2 = User.objects.create_user(
            email="test_user2@example.com",
            password="secret")

        self.n = 5
        self.notes = []
        for i in range(self.n):
            self.notes.append(Note.objects.create(
                title=f"Note title {i}",
                body="Note body",
                owner=self.test_user1))

        self.test_user2.note = Note.objects.create(
            title="Note title",
            body="Note body",
            owner=self.test_user2)

        self.client = APIClient()

    def _authenticate(self):
        response = self.client.post('/api/jwt-auth/', {
            "email": "test_user1@example.com",
            "password": "secret"
        }, format="json")
```

```

token = response.json()["token"]
self.client.credentials(HTTP_AUTHORIZATION='JWT ' + token)

def test_api_can_get_note_list(self):
    self._authenticate()
    response = self.client.get(reverse('api:note-list'))
    self.assertEqual(len(response.json()), self.n)

def test_api_can_get_note_detail(self):
    self._authenticate()
    pk = self.notes[0].id
    response = self.client.get(reverse('api:note-detail', kwargs={"pk": pk}))
    note = response.json()
    self.assertEqual(note["id"], self.notes[0].id)
    self.assertEqual(note["title"], self.notes[0].title)
    self.assertEqual(note["body"], self.notes[0].body)

def test_api_can_create_note(self):
    self._authenticate()
    response = self.client.post(reverse('api:note-list'),
    {
        "title": "New note title",
        "body": "New note body"
    }, format="json")
    note = Note.objects.filter(owner=self.test_user1).last()
    self.assertEqual(Note.objects.filter(owner=self.test_user1).count(),
        self.n + 1)
    self.assertEqual(note.title, "New note title")
    self.assertEqual(note.body, "New note body")

def test_api_can_update_note(self):
    self._authenticate()
    pk = self.notes[0].id
    response = self.client.put(reverse('api:note-detail', kwargs={"pk": pk}),
    {
        "title": "Note title updated",
        "body": "Note body updated"
    }, format="json")
    note = Note.objects.get(pk=pk)
    self.assertEqual(note.title, "Note title updated")
    self.assertEqual(note.body, "Note body updated")

def test_api_can_delete_note(self):
    self._authenticate()
    pk = self.notes[0].id
    response = self.client.delete(reverse('api:note-detail', kwargs={"pk": pk}))
    self.assertEqual(Note.objects.filter(owner=self.test_user1).count(),
        self.n-1)

```

```

def test_api_only_owner_can_get_note_detail(self):
    self._authenticate()
    response = self.client.get(reverse('api:note-detail',
        kwargs={"pk": self.test_user2_note.id}))
    self.assertEqual(response.json()["detail"], "Not found.")

def test_api_only_owner_can_update_note(self):
    self._authenticate()
    pk = self.test_user2_note.id
    response = self.client.put(reverse('api:note-detail', kwargs={"pk": pk}),
        {
            "title": "Note title updated",
            "body": "Note body updated"
        }, format="json")
    note = Note.objects.get(pk=pk)
    self.assertEqual(response.json()["detail"], "Not found.")
    self.assertEqual(note.title, "Note title")
    self.assertEqual(note.body, "Note body")

def test_api_only_owner_can_delete_note(self):
    self._authenticate()
    pk = self.test_user2_note.id
    response = self.client.delete(reverse('api:note-detail', kwargs={"pk": pk}))
    self.assertEqual(response.json()["detail"], "Not found.")
    self.assertEqual(Note.objects.filter(owner=self.test_user2).count(), 1)

```

## Задание

1. Добавьте возможность указывать теги у каждой заметки (можете использовать [Bootstrap Tags Input](#) для отображения и управления тегами на стороне клиента, см. скриншот ниже). При нажатии на тег должен выводиться список только тех заметок, у которых указан этот тег.
2. Продолжая предыдущее задание добавьте возможность поиска не только по тегам, но и по названию заметок.
3. Добавьте возможность расшаривания заметок.
4. Пользователю на электронную почту должна приходить ссылка с подтверждением регистрации.
5. Добавить API ко всем пунктам.
6. **Бонусное задание:** Используйте vue для написания фронтенда к вашему API.

2

The image shows a rich text editor interface. At the top is a toolbar with various icons for text formatting, alignment, and editing. The icons include: Source, Save, Undo, Redo, Bold, Italic, Underline, Strikethrough, Subscript, Superscript, Text Color, Background Color, Bulleted List, Numbered List, Decrease Indent, Increase Indent, Link, Unlink, Image, Video, Table, Horizontal Line, Smiley, Omega, and a globe icon. Below the toolbar is a text area with the text "Тело заметки". The text area is empty except for the text "Тело заметки".

