**Курс: Web-разработка**

**Дисциплина: Создание web-приложений с использованием фреймворка Django**

**Тема занятия № 1-4: Дипломное проектирование**

1. Описание проекта и Проектирование. Развёртывание проекта и описание библиотек

Для начала стоит определить, что за приложение, или вернее веб-приложение мы будем разрабатывать. Для финального проекта нам нужно использовать максимум навыков и знаний, что мы получили в течении курса. **Что нам нужно:**

1. Расширенная модель пользователей, т.к. стандартная уж очень «скудная». Подключение сигналов для моделей.
2. Использование backend + frontend и django MVT подходов. Т.е. django-rest-framework в связке с одним из популярных фронтент фреймворков обеспечить должный уровень «красочности», а более «утилитарные» функции можно поручить стандартному шаблонизатору django.
3. Стандартный шаблонизатор должен содержать как собственные теги и фильтры, так и пагинацию «из коробки».
4. Самый продвинутый из способов аутентификации для frontend-а, - токены.
5. Наличие загружаемых и выгружаемых файлов, будь то изображения или файлы pdf/xlsx.
6. Использование и контроллеров функций и контроллеров классов. Дополнение примитивных функций миксинами.
7. Сериализация и валидация данных.
8. Использование самого мощного из примеров SQL – PostgreSQL и подключение его к проекту.
9. Маршрутизация в формате регулярных выражений, ака «regex».
10. Настройка проекта к публикации, использование виртуальных сред. Подготовка к развёртыванию на «современной и свежей» Ubuntu. Настройка виртуальной машины, uvicorn / gunicorn и nginx.
11. И многое другое, что решим реализовать по ходу дела, либо Вы сможете доработать сами с минимальными изменениями.

**Список используемых библиотек с описанием:**

* **django**

Django – это высокоуровневый Python веб-фреймворк, который позволяет быстро создавать безопасные и поддерживаемые веб-сайты. Созданный опытными разработчиками, Django берёт на себя большую часть хлопот веб-разработки, поэтому вы можете сосредоточиться на написании своего веб-приложения. Именно его мы изучали на курсе.

* **pillow**

Pillow — это свободно распространяемая библиотека для работы с изображениями на Python, с открытым исходным кодом, которая добавляет вашему коду поддержку открытия, изменения и сохранения изображений в различных расширениях.

* **django-environ**

Django-environ — это пакет Python, который позволяет вам настроить Ваше приложение Django с переменными среды, если более конкретно, то «спрятать» чувствительные или опасные настройки, чтобы случайно не залить их в github. Настройки могут быть – секретный код django-проекта, пароль и логин от базы данных, данные от SMTP(почтового) клиента и многое другое.

* **django-grappelli**

Django-grappelli предоставляет красивый интерфейс управления контентом и удобные поиск с фильтрацией в стандартной «django-админке».

* **django-cors-headers**

Django-cors-headers — это приложение Django для обработки заголовков сервера, необходимых для совместного использования ресурсов между источниками (CORS). Эта библиотека будет выступать «щитом» от возможных попыток получить данные не из ресурсов, которые мы ожидаем.

* **djangorestframework**

Django REST framework — это мощный и гибкий инструментарий для создания веб-API. Эта мощная библиотека будет «ядром» для передачи данных к frontend-у.

* **djangorestframework-simplejwt**

Simple JWT предоставляет серверную часть аутентификации JSON Web Token для Django REST Framework. Он направлен на то, чтобы охватить наиболее распространенные варианты использования JWT.

* **openpyxl**

Openpyxl — это библиотека Python для чтения/записи файлов Excel xlsx/xlsm/xltx/xltm. С её помощью мы будем выгружать данные по статистике и логам сайта для тестировщиков или администраторов.

* **psycopg2**

Psycopg — самый популярный адаптер базы данных PostgreSQL для языка программирования Python. Его основными особенностями являются полная реализация спецификации Python DB API 2.0 и безопасность потоков. Именно его мы будем использовать для подключения Django к PostgreSQL.

* **requests**

Requests — это простая библиотека HTTP, и этим всё сказано. Мы будем её использовать для обращений к другим сервисам в «синхронном» режиме.

* **gunicorn**

Gunicorn – это HTTP-сервер Python WSGI для UNIX. Сервер Gunicorn в целом совместим с различными веб-фреймворками, прост в реализации, требует мало ресурсов сервера и довольно быстр.

Итак, общий план готов, библиотеки определены, что же будем разрабатывать? Исходя из функционала это напоминает либо интернет веб-приложение для продажи чего-либо, либо подобный сложный комплекс. Тогда решено, будем разрабатывать книжный сайт, в контексте того, что было озвучено выше.

Устанавливаем самую свежую версию Python с сайта <https://www.python.org/>

Все настройки стандартные, «галочки» можно проставлять кругом.

Начинаем разворачивать проект:

Создаём новую папку –

mkdir web

Входим в созданную директорию –

cd web

Создаём виртуальное окружение –

python -m venv env

Входим в виртуальное окружение –

call env/Scripts/activate.bat

Запускаем установку всех перечисленных выше библиотек –

pip install django pillow django-environ django-grappelli django-cors-headers djangorestframework djangorestframework-simplejwt openpyxl psycopg2 requests gunicorn

Замораживаем зависимости в текстовый файл –

pip freeze > requirements.txt

Развёртываем django-проект в текущей директории –

django-admin startproject backend\_settings .

Развёртываем 2 django-приложения в текущей директории –

django-admin startapp backend\_admin

django-admin startapp backend\_api

1. Настройка базы данных. Установка Postgresql и подключение к базе. Выполнение Миграций и создание суперпользователя

Устанавливаем самую свежую версию PostgreSQL с сайта <https://www.postgresql.org/>

Все настройки стандартные (ip:127.0.0.1 / port:5432), «галочки» можно проставлять кругом.

Запускаем установленный PgAdmin и устаналиваем ключ для суперпользователя, проверяем, что база «postgres» отвечает нам (достаточно кликнуть на нём мышкой, чтобы она «развернулась» и появилась её схема).

На этом этапе нужно создать ещё одну базу – «**django\_db**», также создаём пользователя «**django\_user**» c паролем «**django\_db\_password**». Ну и конечно же даём новому пользователю все права на эту базу.

Самая важная часть – подключение django к базе –

Заходим в файл **settings.py** и меняем строки **DATABASES –**

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql\_psycopg2',

'NAME': 'django\_db',

'USER': 'django\_user',

'PASSWORD': 'django\_db\_password',

'HOST': '127.0.0.1',

'PORT': '5432',

}

}

Сохраняем. Теперь создаём и выполняем миграции –

python manage.py makemigrations

python manage.py migrate

На данном этапе должны быть сообщения в командной строке в стиле «ОК». Если всё нормально, продолжаем. Создаём суперпользователя –

python manage.py createsuperuser

Следуя инструкциям командной строки, создаём пользователя с данными: admin [admin@gmail.com](mailto:admin@gmail.com) admin. Подтверждаем слабый пароль.

В качестве проверки можно уже запустить django и проверить –

python manage.py runserver 0.0.0.0:8000

Переходим в браузере на ссылку [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/) **.** Если видим страницу с django, то всё нормально.

1. Общие настройки проекта и статических файлов, настройка сборки статики. Запуск отладочного веб-сервера

Теперь нам нужно настроить большую часть django и установленных библиотек –

Входим в файл **settings.py** и, последовательно сверху – вниз меняем –

ALLOWED\_HOSTS = ['\*']

CORS\_ALLOW\_ALL\_ORIGINS = True

CORS\_URLS\_REGEX = r"^/api/.\*$"

Мы открыли все хосты для развёртывания django, и дали доступ к drf со всех «источников», но условием, что «маршрут» будет начинаться с префикса **/api/.**

Далее меняем список установленных приложений на –

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'backend\_admin',

'backend\_api',

'corsheaders',

'rest\_framework',

'rest\_framework\_simplejwt',

'rest\_framework\_simplejwt.token\_blacklist',

]

Тут мы добавили jwt, drf, corheaders и наши 2 собственных приложения. Описание библиотек уже было выше.

Далее нужно заменить middleware, т.е. промежуточный уровень обработки запросов и ответов, а конкретнее, добавить библиотеку corsheaders перед –

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'corsheaders.middleware.CorsMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

Далее изменяем настройки шаблонизаторов на –

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [BASE\_DIR / 'templates', BASE\_DIR / 'react'],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

'django.template.context\_processors.request',

],

},

},

]

На этом этапе нужно создать две папки в корне проекта: «**templates**» и «**react**». Первое будет хранить html-файлы, ака шаблоны для django MVT, во второй будет react-фронтенд. Также добавляем контекстный процессор для «досылания» к фронтенду запроса (**request**).

Базу данных мы заменили, ещё раз проверьте, что эти строчки –

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'database/db.sqlite3',

}

}

У Вас заменены на настройки PostgreSQL.

Меняем язык на русский и временную зону на GMT+6 –

LANGUAGE\_CODE = 'ru-RU'

TIME\_ZONE = 'Asia/Almaty'

Далее настраиваем статику и обработку медиа-файлов. Тут нужно создать папку «**static**» в корне проекта, там будут хранится и туда собираться все статические файлы. Также создаём папки «static\_external», «frontend/build/static», «frontend/public/static», «static/media».

STATIC\_URL = '/static/'

if DEBUG:

STATICFILES\_DIRS = [

Path(BASE\_DIR / 'static\_external'),

Path(BASE\_DIR / 'static'),

Path(BASE\_DIR / 'frontend/build/static'),

Path(BASE\_DIR / 'frontend/public/static'),

]

else:

STATIC\_ROOT = Path(BASE\_DIR / 'static')

STATICFILES\_DIRS = [

]

MEDIA\_URL = 'media/'

MEDIA\_ROOT = Path(BASE\_DIR, 'static/media')

Что тут особенного, в чём суть логического разделения? У нас есть 2 режима – режим разработки и режим публикации, тут мы используем специфичные настройки для каждого из режимов.

Теперь применяем настройки Resr Framework-а –

REST\_FRAMEWORK = {

'DEFAULT\_PERMISSION\_CLASSES': (

'rest\_framework.permissions.IsAuthenticated',

),

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

),

}

Т.е. стандартный настройки для доступа и аутентификации. Теперь мы используем JWT, далее нужно его настройки –

SIMPLE\_JWT = {

'ACCESS\_TOKEN\_LIFETIME': timedelta(days=1),

'REFRESH\_TOKEN\_LIFETIME': timedelta(days=2),

'ROTATE\_REFRESH\_TOKENS': True,

'BLACKLIST\_AFTER\_ROTATION': True,

'UPDATE\_LAST\_LOGIN': True,

'ALGORITHM': 'HS256',

'SIGNING\_KEY': SECRET\_KEY,

'VERIFYING\_KEY': None,

'AUDIENCE': None,

'ISSUER': None,

'JWK\_URL': None,

'LEEWAY': 0,

'AUTH\_HEADER\_TYPES': ('Bearer',),

'AUTH\_HEADER\_NAME': 'HTTP\_AUTHORIZATION',

'USER\_ID\_FIELD': 'id',

'USER\_ID\_CLAIM': 'user\_id',

'USER\_AUTHENTICATION\_RULE': 'rest\_framework\_simplejwt.authentication.default\_user\_authentication\_rule',

'AUTH\_TOKEN\_CLASSES': ('rest\_framework\_simplejwt.tokens.AccessToken',),

'TOKEN\_TYPE\_CLAIM': 'token\_type',

'TOKEN\_USER\_CLASS': 'rest\_framework\_simplejwt.models.TokenUser',

'JTI\_CLAIM': 'jti',

'SLIDING\_TOKEN\_REFRESH\_EXP\_CLAIM': 'refresh\_exp',

'SLIDING\_TOKEN\_LIFETIME': timedelta(days=1),

'SLIDING\_TOKEN\_REFRESH\_LIFETIME': timedelta(days=2),

}

Большая часть параметров интуитивно понятны, остальную можно всегда подсмотреть в официальной документации.

C файлом настройки всё, сохраняем и закрываем. А теперь немного забежим наперёд, скорректируем маршрутизацию под статику и библиотеки, файл **urls.py** –

from django.contrib import admin

from django.urls import path, include, re\_path

from django.conf import settings

from django.conf.urls.static import static

# from backend\_api.views import index

urlpatterns = [

path('grappelli/', include('grappelli.urls')), # grappelli URLS

# path('', index, name=''),

path('admin/', admin.site.urls),

# path('hide/', include('backend\_admin.urls')),

# path('api/', include('backend\_api.urls')),

path('api-auth/', include('rest\_framework.urls'))

]

urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Несколько закомментированных строк нам понадобятся в будущем.

Сохраняем и снова запускаем проект, желательно также выполнить миграции повторно (миграции для JWT) –

python manage.py makemigrations

python manage.py migrate

python manage.py runserver 0.0.0.0:8000

1. Расширение модели пользователей. Сигнал для автоматического создания профиля

Существует несколько способов расширить стандартную модель пользователя. Мы проходили в рамках курса через прокси модель, через наследование с миксинами от AbstractBaseUser, и самый безопасный из всех: через связь к дополнительной модели. Будем использовать последний вариант, он наиболее оптимальный, т.к. не «ломает» другие библиотеки, файл **model.py** в приложении **bakend\_api** –

from django.db import models

from django.contrib.auth.models import User

class Profile(models.Model):

user = models.OneToOneField(

primary\_key=True,

unique=True,

editable=True,

blank=True,

null=False,

default=None,

verbose\_name='Аккаунт',

help\_text='<small class="text-muted">Аккаунт</small><hr><br>',

to=User,

on\_delete=models.CASCADE,

)

email = models.EmailField(

unique=True,

editable=True,

blank=False,

null=False,

default="",

verbose\_name="Почта",

help\_text='<small class="text-muted">пример: bogdandrienko@gmail.com</small><hr><br>',

)

user\_email\_spam\_success = models.BooleanField('spam email', default=False)

phone = models.IntegerField('mobile phone', default=0)

class Meta:

app\_label = 'auth'

ordering = ('user',)

verbose\_name = 'Профиль пользователя'

verbose\_name\_plural = 'Профили пользователей'

def \_\_str\_\_(self):

return f'профиль: {self.user.username}...'

Тут мы связываем через OneToOne к модели стандартного пользователя дополнительную модель.

Прежде чем её настраивать в админке, реализуем сигнал, который будет автоматизировать создание модели профиля в дополнение к модели пользователя –

from django.db.models.signals import post\_save

from django.dispatch import receiver

@receiver(post\_save, sender=User)

def create\_profile(sender, instance, created, \*\*kwargs):

if created:

Profile.objects.get\_or\_create(user=instance, email=instance.username)

else:

Profile.objects.get\_or\_create(user=instance, email=instance.username)

Сигнал готов, можно настроить его и иначе, на Ваше усмотрение. Теперь настройка отображения модели профиля на админ панели, файл **admin.py** –

from django.contrib import admin

from backend\_api import models

admin.site.register(models.Profile)

Сохраняем всё, запускаем сервер, входим в админку и создадим руками пару пользователей. Если модель профиля автоматически создаётся, значит всё работает отлично.

1. Написание основных маршрутов. Регулярные выражения

В глобальное файле маршрутизации разделяем маршруты на две части, для административных функций и для внешних сервисов:

from django.conf import settings

from django.conf.urls.static import static

from django.contrib import admin

from django.shortcuts import redirect

from django.urls import path, include, re\_path

from backend\_api import views

urlpatterns = [

path('grappelli/', include('grappelli.urls')), # grappelli URLS

path('admin/', admin.site.urls),

path('', views.index, name='index'),

path('moderator/', include('backend\_admin.urls')),

path('api/', include('backend\_api.urls')),

path('api/', include('rest\_framework.urls')),

]

urlpatterns += [re\_path(r'^.\*$', lambda request: redirect('/', permanent=False), name='redirect')]

urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Теперь непосредственно определим маршруты для каждого из двух файлов, административные:

from django.conf import settings

from django.conf.urls.static import static

from django.contrib import admin

from django.shortcuts import redirect

from django.urls import path, include, re\_path

from backend\_admin import views

urlpatterns = [

path('', views.HomeView.as\_view(), name='home'),

path('home/', views.HomeView.as\_view(), name='HomeView'),

re\_path(r'^get\_active\_user\_list/$', views.GetActiveUserListView.as\_view(), name='get\_active\_user\_list'),

re\_path(r'^login/$', views.django\_login, name='login'),

re\_path(r'^logout/$', views.django\_logout, name='logout'),

re\_path(r'^receipt/(?P<receipt\_id>\d+)/$', views.receipt, name='receipt\_id'), # GET(one) / PUT (PUTCH) / DELETE

re\_path(r'^receipt/$', views.receipt, name='receipt'), # GET(all) / POST

]

Выше идёт комбинация регулярных выражений с обычных маршрутов, вызов функций-контроллеров и классов. Также всё это сочетает подход REST Api.

Последний файл с маршутизацией, сервисы:

from django.conf import settings

from django.conf.urls.static import static

from django.contrib import admin

from django.shortcuts import redirect

from django.urls import path, include, re\_path

from backend\_api import views

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

)

urlpatterns = [

path('', views.index),

path('registration/', views.registration),

path('token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

re\_path(r'^receipt/(?P<receipt\_id>\d+)/$', views.receipt, name='receipt\_id'), # GET(one) / PUT (PUTCH) / DELETE

re\_path(r'^receipt/$', views.receipt, name='receipt'), # GET(all) / POST

]

1. Написание Моделей и связей

После завершения маршрутов, можно приступить к моделям, которые и будут хранить и возвращать данные. Модель для связанных моделей «рецепта»:

from django.contrib.auth.models import User

from django.db import models

from django.core.validators import FileExtensionValidator, MinValueValidator, MaxValueValidator

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.utils import timezone

class ReceiptCategory(models.Model):

title = models.CharField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="Заголовок",

verbose\_name="Заголовок:",

help\_text='<small class="text-muted">это наш заголовок</small><hr><br>',

max\_length=250,

)

class Meta:

app\_label = 'backend\_api'

ordering = ('title',)

verbose\_name = 'Категория'

verbose\_name\_plural = 'Категории рецептов'

def \_\_str\_\_(self): # возвращает строкове представление объекта

return f'{self.title}'

class ReceiptIngredient(models.Model):

name = models.TextField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="Название",

verbose\_name="Название:",

help\_text='<small class="text-muted">это наше название</small><hr><br>',

)

class Meta:

app\_label = 'backend\_api'

ordering = ('name',)

verbose\_name = 'Ингредиент'

verbose\_name\_plural = 'Ингредиенты'

def \_\_str\_\_(self): # возвращает строкове представление объекта

return f'{self.name}'

class Receipt(models.Model):

title = models.CharField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="Заголовок",

verbose\_name="Заголовок:",

help\_text='<small class="text-muted">это наш заголовок</small><hr><br>',

max\_length=250,

)

image = models.ImageField(

unique=False,

editable=True,

blank=True,

null=True,

default="img/receipt/default/default\_receipt.jpg",

verbose\_name="Заставка:",

help\_text='<small class="text-muted">это наша заставка</small><hr><br>',

validators=[FileExtensionValidator(['jpg', 'png'])],

upload\_to='img/receipt',

max\_length=100,

)

time\_to\_cook = models.IntegerField( # BigIntegerField SmallIntegerField PositiveIntegerField ...

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="1",

verbose\_name="Время на приготовление(минуты)",

help\_text='<small class="text-muted">это наше время на приготовление</small><hr><br>',

validators=[MinValueValidator(1), MaxValueValidator(9999)],

)

category = models.ManyToManyField(

db\_column='country\_db\_column',

db\_index=True,

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

default=None,

verbose\_name='Категория блюда',

help\_text='<small class="text-muted">категория</small><hr><br>',

to=ReceiptCategory,

)

author = models.ForeignKey(

db\_index=True,

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Автор блюда',

help\_text='<small class="text-muted">автор</small><hr><br>',

to=User,

on\_delete=models.SET\_NULL, # CASCADE - удаляет всю запись, при удалении связанной записи

# SET\_NULL - зануляет всю запись, при удалении связанной записи DO\_NOTHING - ничего не делать

)

ingredients = models.ManyToManyField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

default=None,

verbose\_name='Ингредиенты блюда',

help\_text='<small class="text-muted">ингредиенты</small><hr><br>',

to=ReceiptIngredient,

)

description = models.TextField(

primary\_key=False,

unique=False,

editable=True,

blank=False,

null=False,

default="Описание",

verbose\_name="Описание:",

help\_text='<small class="text-muted">это наше Описание</small><hr><br>',

)

is\_completed = models.BooleanField(

default=False,

# editable=

)

instructions = models.FileField(

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name="Инструкция:",

help\_text='<small class="text-muted">Инструкция</small><hr><br>',

validators=[FileExtensionValidator(['PDF', 'XLSX'])],

upload\_to='file/pdf',

max\_length=100,

)

# is\_show = models.BooleanField(

# default=False,

# # editable=

# )

class Meta:

app\_label = 'backend\_api'

ordering = ('title', 'description')

verbose\_name = 'Рецепт'

verbose\_name\_plural = 'Рецепты'

def \_\_str\_\_(self): # возвращает строковое представление объекта

return f'{self.title}'

def return\_clear\_data(self):

title = self.title

return str(title).strip() + " banana"

def get\_rating(self):

ratings = ReceiptRating.objects.filter(

receipt=Receipt.objects.get(id=self.id),

)

like\_count = 0

dislike\_count = 0

for rating in ratings:

if rating.is\_liked:

like\_count += 1

else:

dislike\_count += 1

return dict(total=ratings.count, like\_count=like\_count, dislike\_count=dislike\_count)

class ReceiptComment(models.Model):

comment\_text = models.CharField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="Текст комментария",

verbose\_name="Заголовок:",

help\_text='<small class="text-muted">Текст комментария</small><hr><br>',

max\_length=500,

)

user = models.ForeignKey(

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Пользователь',

help\_text='<small class="text-muted">Пользователь</small><hr><br>',

to=User,

on\_delete=models.SET\_NULL, # CASCADE SET\_NULL DO\_NOTHING

)

receipt = models.ForeignKey(

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Рецепт',

help\_text='<small class="text-muted">Рецепт</small><hr><br>',

to=Receipt,

on\_delete=models.CASCADE, # CASCADE SET\_NULL DO\_NOTHING

)

time = models.DateTimeField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default=timezone.now,

verbose\_name="время создания:",

help\_text='<small class="text-muted">время создания</small><hr><br>',

auto\_now\_add=False,

auto\_now=False,

)

create = models.DateTimeField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

# default=timezone.now,

verbose\_name="время создания:",

help\_text='<small class="text-muted">время создания</small><hr><br>',

auto\_now\_add=True, # автоматически добавляет время в момент создания объекта (нельзя редактировать)

auto\_now=False, # автоматически добавляет время в момент создания и изменения объекта (нельзя редактировать)

)

update = models.DateTimeField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

# default=timezone.now,

verbose\_name="время создания:",

help\_text='<small class="text-muted">время создания</small><hr><br>',

auto\_now\_add=False, # автоматически добавляет время в момент создания объекта (нельзя редактировать)

auto\_now=True, # автоматически добавляет время в момент создания и изменения объекта (нельзя редактировать)

)

class Meta:

app\_label = 'backend\_api'

ordering = ('-time',)

verbose\_name = 'Комментарий к рецепту'

verbose\_name\_plural = 'Комментарии к рецептам'

def \_\_str\_\_(self): # возвращает строкове представление объекта

return f'{self.comment\_text[:10:1]}'

# сигнал на обновление времени в комментарии

@receiver(post\_save, sender=ReceiptComment)

def update\_comment\_time(sender, instance, created, \*\*kwargs):

if not created: # если объект не создан впервые

obj = ReceiptComment.objects.get(id=instance.id)

if obj.time != timezone.now():

obj.time = timezone.now()

obj.save()

class ReceiptRating(models.Model):

is\_liked = models.BooleanField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default=False,

verbose\_name="Лайк:",

help\_text='<small class="text-muted">Лайк</small><hr><br>',

)

rating\_value = models.IntegerField( # BigIntegerField SmallIntegerField PositiveIntegerField ...

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="0",

verbose\_name="Оценка",

help\_text='<small class="text-muted">Оценка</small><hr><br>',

validators=[MinValueValidator(0), MaxValueValidator(10)],

)

user = models.ForeignKey(

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Пользователь',

help\_text='<small class="text-muted">Пользователь</small><hr><br>',

to=User,

on\_delete=models.CASCADE, # CASCADE SET\_NULL DO\_NOTHING

)

receipt = models.ForeignKey(

error\_messages=False,

primary\_key=False,

unique\_for\_date=False,

unique\_for\_month=False,

unique\_for\_year=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Рецепт',

help\_text='<small class="text-muted">Рецепт</small><hr><br>',

to=Receipt,

on\_delete=models.CASCADE, # CASCADE SET\_NULL DO\_NOTHING

)

class Meta:

app\_label = 'backend\_api'

ordering = ('receipt',)

verbose\_name = 'Рейтинг рецепта'

verbose\_name\_plural = 'Рейтинги рецептов'

def \_\_str\_\_(self): # возвращает строкове представление объекта

return f'{self.receipt}'

Код выше нужно вставить в файл с моделями, вернее всего будет в приложение backend\_api.

Модель пользователя уже расширена. Напишем ещё модель для логирования:

class Log(models.Model):

path = models.URLField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="",

verbose\_name="Маршрут",

help\_text='<small class="text-muted">Маршрут</small><hr><br>',

max\_length=64,

)

method = models.SlugField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="",

verbose\_name="Метод",

help\_text='<small class="text-muted">Метод</small><hr><br>',

max\_length=8,

)

user = models.ForeignKey(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default=None,

verbose\_name='Пользователь',

help\_text='<small class="text-muted">Пользователь</small><hr><br>',

to=User,

on\_delete=models.SET\_NULL, # CASCADE SET\_NULL DO\_NOTHING

)

error = models.CharField(

primary\_key=False,

unique=False,

editable=True,

blank=True,

null=True,

default="нет ошибки",

verbose\_name="Ошибка:",

help\_text='<small class="text-muted">Ошибка или текст исключения</small><hr><br>',

max\_length=256,

)

created = models.DateTimeField(

unique=False,

editable=True,

blank=True,

null=True,

default=timezone.now,

verbose\_name='Дата и время создания',

help\_text='<small class="text-muted">datetime\_field</small><hr><br>',

auto\_now=False,

auto\_now\_add=False,

)

class Meta:

app\_label = 'auth'

ordering = ('-created', 'user')

verbose\_name = 'Лог'

verbose\_name\_plural = 'Логи'

def \_\_str\_\_(self):

return f'{self.created} {self.method} {self.path[:30]} {self.user.username[:30]}'

Модель лога хранить более правильно в «административном приложении».

1. Написание основных контроллеров. Контроллеры функции и классы

Начнём с простейших контроллеров-классов:

class HomeView(View):

def get(self, request):

return render(request, 'backend\_admin/Home.html', context={})

Далее административный функционал для выгрузки пользователей:

class GetActiveUserListView(View):

initial = {'key': 'value'}

template\_name = 'backend\_admin/GetUserList.html'

def get(self, request, \*args, \*\*kwargs):

users = User.objects.filter(is\_active=True) # только активные пользователи

# EXCEL GENERATOR

workbook = openpyxl.Workbook()

worksheet = workbook.active

kwargs\_list = ["username", "password", "id", "is\_superuser"]

users\_list = [x for x in users] # list comprehension (pre generator yeld)

for kwarg in kwargs\_list:

worksheet.cell(row=1, column=kwargs\_list.index(kwarg) + 1, value=kwarg)

for user\_obj in users\_list:

row\_index = users\_list.index(user\_obj)

for kwarg in kwargs\_list:

col\_index = kwargs\_list.index(kwarg)

worksheet.cell(row=row\_index + 2, column=col\_index + 1, value=getattr(user\_obj, kwarg))

path = f'temp/new\_{datetime.datetime.now().strftime("%m-%d-%Y %H-%M-%S-%f")}.xlsx'

directory = os.path.join("static", "temp")

if not os.path.exists(directory):

os.mkdir(directory)

if os.path.exists(settings.STATIC\_URL[1:] + path):

os.remove(settings.STATIC\_URL[1:] + path)

workbook.save(settings.STATIC\_URL[1:] + path)

# EXCEL GENERATOR

context = {

"title": "Абракадабра",

"headers": ["id", "username", "password", "password", "password", "is\_superuser"],

"user\_list": users,

"excel\_file": path

}

return render(request, self.template\_name, context)

Теперь функции-контроллеры:

def django\_login(request):

try:

if request.method == "GET":

context = {

"login\_form": forms.LoginUserForm()

}

return render(request, "backend\_admin/Login.html", context=context)

if request.method == "POST":

print(request.POST)

email = request.POST["email"]

password = request.POST["password"]

if email and password:

user = authenticate(username=email, password=password)

if user:

login(request, user)

return redirect(reverse('home', args=()))

return redirect(reverse('login', args=()))

except Exception as error:

return Logger(request=request, error=str(error)).log()

def django\_logout(request):

try:

logout(request)

return redirect(reverse('login', args=()))

except Exception as error:

return Logger(request=request, error=str(error)).log()

def receipt(request):

try:

if request.method == "GET":

categories = models\_api.ReceiptCategory.objects.all()

ingredients = models\_api.ReceiptIngredient.objects.all()

context = {"categories": categories, "ingredients": ingredients}

return render(request, "backend\_admin/CreateReceipt.html", context=context)

Пропущенные методы можно до заполнить согласно практическим заданиям, смысл тот же.

1. Дизайн. Вспомогательные веб-страницы. Шаблонизатор Django

Используем стандартный подход с «**extend**», т.е. расширением шаблонов один через другой, так мы будем писать минимум лишнего и дублируемого кода.

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport"

content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>

{% block title %}Магазинчик суши{% endblock title %}

</title>

<link rel="icon" href="{% static 'img/logo.png' %}" type="image/png">

<link rel="stylesheet" href="{% static 'css/bootstrap/bootstrap.css' %}">

<link rel="stylesheet" href="{% static 'fontawesome/css/all.css' %}">

<link rel="stylesheet" href="{% static 'font\_zen/style.css' %}">

<link rel="stylesheet" href="{% static 'css/my\_django.css' %}">

{% block head %}{% endblock head %}

</head>

<body class="d-flex flex-column h-100 custom\_body\_1 container" style="min-height: 100vh;">

{% block navbar %}{% include 'components/navbar.html' %}{% endblock navbar %}

<main class="flex-shrink-0" style="min-height: 80vh;">

{% block main %}...{% endblock main %}

</main>

{% block footer %}{% include 'components/footer.html' %}{% endblock footer %}

<script src="{% static 'js/bootstrap/bootstrap.js' %}"></script>

</body>

</html>

Для примера вспомогательной страницы, создадим страницу, которая будет выводить список логов (для этого должна быть создана модель логирования и настроено сохранение данных в неё). Пагинация, фильтрация, поиск – всё включено. Также можно будет скачать файл excel.

\*\*\*

Примером другой вспомогательной страницы будет служить служебная страница, с которой можно будет создать массив пользователей из загружаемого excel-файла. Также будет доступен для скачивания шаблон для заполнения пользователей.

**\*\*\***

Последним примером напишем страницу «ошибки», т.е. в случае ошибки на каком-либо этапе, будь то реакт или другие вспомогательные страницы, мы должны возвращать эту страницу с возможностью автоперезагрузки в случае ошибки.

\*\*\*

1. Django Rest framework. Контроллеры и сериализаторы. Пагинация

Примеры сервиса для возврата значения с разными методами:

@api\_view(http\_method\_names=["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"])

@permission\_classes([IsAuthenticated])

def receipts(request, book\_id=0):

time.sleep(1)

try:

if int(book\_id) > 0:

if request.method == "GET":

book = get\_object\_or\_404(models.ModelBook, id=book\_id)

serialized\_book = serializers.BookSerializer(instance=book, many=False).data

return Response(data={"object": serialized\_book}, status=status.HTTP\_200\_OK)

elif request.method == "DELETE":

book = models.ModelBook.objects.get(id=book\_id)

# book.delete()

return Response(data={"response": "Успешно удалено."}, status=status.HTTP\_200\_OK)

elif request.method == "PUT" or request.method == "PATCH":

book = models.ModelBook.objects.get(id=book\_id)

title = request.POST.get("title", "Шаблон заголовка")

description = request.POST.get("description", "Шаблон описания")

if book.title != title:

book.title = title

if book.description != description:

book.description = description

book.save()

book = models.ModelBook.objects.get(id=book\_id)

serialized\_book = serializers.BookSerializer(instance=book, many=False).data

return Response(data=serialized\_book, status=status.HTTP\_200\_OK)

else:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

else:

if request.method == "GET":

page = request.GET.get("page", 1)

limit = request.GET.get("limit", 5)

books\_queryset = models.ModelBook.objects.all()

count = len(books\_queryset)

paginator\_instanse = Paginator(books\_queryset, limit)

books\_queryset = paginator\_instanse.get\_page(number=page).object\_list

serialized\_books = serializers.BookSerializer(instance=books\_queryset, many=True).data

return Response(data={"object\_list": serialized\_books, "count": count}, status=status.HTTP\_200\_OK)

elif request.method == "POST": # создание книги

title = request.POST.get("title", "Шаблон заголовка")

description = request.POST.get("description", "Шаблон описания")

models.ModelBook.objects.create(

title=title,

description=description

)

return Response(data={"response": "Успешно создано."}, status=status.HTTP\_201\_CREATED)

else:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

except Exception as error:

if settings.DEBUG:

print(f"error {error}")

return Response(status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

Для возможных категорий:

@api\_view(http\_method\_names=["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"])

@permission\_classes([IsAuthenticated])

def categories(request, category\_id=0):

time.sleep(2)

if int(category\_id) >= 1:

if request.method == "GET": # получение книги

book = get\_object\_or\_404(models.ModelBook, id=category\_id)

serialized\_book = serializers.BookSerializer(instance=book, many=False).data

return Response(data=serialized\_book, status=status.HTTP\_200\_OK)

elif request.method == "DELETE":

book = models.ModelBook.objects.get(id=category\_id).delete()

book.delete()

return Response(data={"response": "Успешно удалено."}, status=status.HTTP\_200\_OK)

elif request.method == "PUT" or request.method == "PATCH":

book = models.ModelBook.objects.get(id=category\_id)

title = request.POST.get("title", "Шаблон заголовка")

description = request.POST.get("description", "Шаблон описания")

if book.title != title:

book.title = title

if book.description != description:

book.description = description

book.save()

book = models.ModelBook.objects.get(id=category\_id)

serialized\_book = serializers.BookSerializer(instance=book, many=False).data

return Response(data=serialized\_book, status=status.HTTP\_200\_OK)

else:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

else:

if request.method == "GET":

categories\_list = models.ModelBookCategory.objects.all()

serialized\_categories = serializers.BookCategorySerializer(instance=categories\_list, many=True).data

return Response(data={"object\_list": serialized\_categories}, status=status.HTTP\_200\_OK)

elif request.method == "POST": # создание книги

title = request.POST.get("title", "Шаблон заголовка")

description = request.POST.get("description", "Шаблон описания")

models.ModelBook.objects.create(

title=title,

description=description

)

return Response(data={"response": "Успешно создано."}, status=status.HTTP\_201\_CREATED)

else:

return Response(status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

Маршруты для сервисов используются для передачи данных к фронтенду и внешним связям.

1. Django JWT. использование токенов

Ранее мы уже произвели настройку и установку библиотеки. Теперь нужно задать ей маршруты, которые будут возвращать или обновлять токен по запросу от frontend-а.

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

)

urlpatterns = [

# …

path('token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

# …

]

Тут мы как раз и вытащили контроллеры-классы, уже написанные за нас, и подключили их к маршрутам приложения.

1. React

Весь дизайн мы будем разделять на две категории, согласно отображения данных:

Первое – это стандартный подход MVC(T), т.е. вспомогательные и дополнительные страницы (логирование, тестирование, выгрузка пользователей…) мы будем реализовывать обычным способом.

Второе – это подход, при котором разделяют backend и frontend части. В качестве основы backend- а будет выступать django-rest-framework, а frontend – react js.

Начнём с установки node js и развёртывания «шаблона» react-приложения.

Качаем последнюю LTS версию ноды с сайта <https://nodejs.org/en/>

Установка как обычно, всё по стандарту.

Теперь к библиотекам для react:

* **react**

React – это библиотека JavaScript для создания пользовательских интерфейсов. Она более сложная, но в то же время более гибкая и мощная в плане визуальных эффектов и отображения интерфейса, нежели django – шаблонизатор.

* **react-bootstrap**

Bootstrap – это свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения. Разрабатывать большую часть модулей может быть утомительно, а вот использовать уже чужие блоки относительно просто.

* **react-redux**

Redux – это библиотека, для построения дерева контейнеров, которые дают предсказуемое состояния для JS-приложений и компонентов. Если говорить совсем грубо, эта библиотека обеспечит нас 4-5 состояниями для каждого запроса, который мы будем делать к drf.

* **react-router-dom**

React Router – это облегченная полнофункциональная библиотека маршрутизации для библиотеки React JavaScript. React Router работает везде, где работает React.

* **axios**

Axios – это простой HTTP-клиент для браузера и node.js на основе Promise. Axios предоставляет простую в использовании библиотеку в небольшом пакете с очень расширяющимся интерфейсом. Именно его мы будем использовать для обращений к django-rest-framework.

* **typescript**

Typesript – это язык программирования, позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. Он добавит нам немного «типизации», а это, в свою очередь, даст возможность более гибко усложнять проект и отлавливать «баги».

Исходя из библиотек разворачиваем проект react в папке «**frontend**», прям в корне проекта. Для развёртывания будем использовать стандартный шаблон –

npx create-react-app . --template redux-typescript

Утилита попросит установить приложение CRA, соглашаемся. Далее пойдёт процесс развёртывания.

Устанавливаем библиотеки, которые описали выше:

npm i react-bootstrap react-router-dom axios

На этом этапе можно запустить сборку.

npm run build

Или сразу развёртывание сервера разработки react.

npm start

Автоматически откроется локальный хост на порту 3000.

Далее последовательно настраиваем проксирование, маршрутизацию и менеджер состояний:

**index.tsx:**

import React from 'react';

import { createRoot } from 'react-dom/client';

import { Provider } from 'react-redux';

import { store } from './app/store';

import App from './App';

import reportWebVitals from './reportWebVitals';

import './index.css';

import './css/bootstrap/bootstrap.min.css';

import './css/fontawesome/css/all.css';

const container = document.getElementById('root')!;

const root = createRoot(container);

root.render(

<React.StrictMode>

<Provider store={store}>

<App />

</Provider>

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

**App.tsx:**

import React from 'react';

import {

BrowserRouter,

Routes,

Route,

} from "react-router-dom";

import './App.css';

import './css/my.css';

import Home from './pages/Home';

import Login from './pages/Login';

import {Logout} from './pages/Logout';

import { Books } from './pages/Books';

import { Book } from './pages/Book';

function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />}>

</Route>

<Route path="/login" element={<Login />}>

</Route>

<Route path="/logout" element={<Logout />}>

</Route>

<Route path="/books" element={<Books />}>

</Route>

<Route path="/books/:id" element={<Book />}>

</Route>

</Routes>

</BrowserRouter>

);

}

export default App;

**store.ts:**

import { configureStore, ThunkAction, Action } from '@reduxjs/toolkit';

import { combineReducers } from "redux";

import thunk from "redux-thunk";

import \* as reducers from '../components/Reducers'

const globalReducer = combineReducers({

token: reducers.R\_Token,

books: reducers.R\_Books,

book: reducers.R\_Book,

});

const tokenObj = localStorage.getItem("token") ? localStorage.getItem("token") : null;

const initialState = {

token: {data: tokenObj }

};

export const store = configureStore({

reducer: globalReducer,

devTools: true,

middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(thunk),

preloadedState: initialState,

});

export type AppDispatch = typeof store.dispatch;

export type RootState = ReturnType<typeof store.getState>;

export type AppThunk<ReturnType = void> = ThunkAction<

ReturnType,

RootState,

unknown,

Action<string>

>;

Что теперь? Настраиваем шаблоны для компонентного «наследования», т.е. переиспользования кода: base.jsx:

export function Base1({ children }) {

return (

<div className="custom\_body">

<LoginComponent/>

<main className="custom\_main">

<Navbar4 />

{children}

</main>

<Footer2 />

</div>

);

}

Подобным же образом пишем футер и навбар:

export function Footer2() {

return (

<footer className="text-muted">

<div className="px-4 text-center">

<img className="d-block mx-auto mb-4 img img-fuild w-25" src="/static/bootstrap-themes.png" alt=""/>

<h1 className="display-5 fw-bold">Centered hero</h1>

<div className="col-lg-6 mx-auto">

<p className="lead mb-4">Quickly design and customize responsive mobile-first sites with Bootstrap, the world’s most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.</p>

</div>

</div>

</footer>

);

}

export const Navbar4 = () => {

const token = useSelector((state) => state.token);

return (

<Navbar bg="light" expand="md">

<Container fluid>

<Navbar.Brand href="/">Главная</Navbar.Brand>

<Navbar.Brand

href="https://www.livelib.ru/genre/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/best/"

className="lead fw-bold"

>

Эталон

</Navbar.Brand>

<Navbar.Toggle aria-controls="navbarScroll" />

<Navbar.Collapse id="navbarScroll">

<Nav

className="me-auto my-2 my-lg-0"

style={{ maxHeight: "100px" }}

navbarScroll

>

<NavDropdown title="Что почитать" id="navbarScrollingDropdown">

<Link

to="/news"

className="text-decoration-none text-dark m-1 p-1"

>

Новинки

</Link>

<NavDropdown.Item className="text-decoration-none">

<Link

to="/top"

className="text-decoration-none text-dark m-1 p-1 w-100"

>

Лучшие

</Link>

</NavDropdown.Item>

<NavDropdown.Item className="text-decoration-none">

<Link

to="/books"

className="text-decoration-none text-dark m-1 p-1 w-100"

>

Все книги

</Link>

</NavDropdown.Item>

<NavDropdown.Item href="#action4">Рекомендации</NavDropdown.Item>

<NavDropdown.Divider />

<NavDropdown.Item href="#action5">Жанры</NavDropdown.Item>

</NavDropdown>

<NavDropdown title="Лента" id="navbarScrollingDropdown">

<NavDropdown.Item href="#action3">Рецензии</NavDropdown.Item>

<NavDropdown.Item href="#action4">Цитаты</NavDropdown.Item>

</NavDropdown>

<NavDropdown title="Авторы" id="navbarScrollingDropdown">

<NavDropdown.Item href="#action3">

Список авторов

</NavDropdown.Item>

</NavDropdown>

<Nav.Link href="#" disabled>

Карта развития платформы

</Nav.Link>

<Link to={"/auth"} className={"btn btn-sm btn-outline-primary"}>

Авторизация

</Link>

</Nav>

<Form className="d-flex">

<div className=" input-group">

<Form.Control

type="search"

placeholder="что ищем"

className="w-25"

aria-label="Search"

/>

<Button variant="outline-success">Поиск</Button>

</div>

</Form>

<div className="input-group w-25 p-2">

<Link to={"/register"} className={"btn btn-sm btn-outline-warning"}>

Регистрация

</Link>

{!token.data ? (

<Link to={"/login"} className={"btn btn-sm btn-outline-primary"}>

Войти

</Link>

) : (

<Link

to={"/logout"}

className={"btn btn-sm btn-outline-danger"}

>

Выйти

</Link>

)}

</div>

</Navbar.Collapse>

</Container>

</Navbar>

);

};

Обёртка готова, приступаем к домашней странице:

function Home() {

return (

<Base1>

<div className="px-2 py-2 my-2 text-center">

<img

className="d-block mx-auto mb-4 img-fluid img-responsive img-thumbnail w-50"

src="/static/home\_hero.png"

alt="image"

/>

<h1 className="display-5 fw-bold">Наш сайт</h1>

<div className="col-lg-6 mx-auto">

<p className="lead mb-4">

Совсем перестали читать! Ну ка!

</p>

<div className="d-grid gap-2 d-sm-flex justify-content-sm-center input-group">

<Link

to="/news"

type="button"

className="btn btn-primary btn-md px-4 gap-3"

>

Перейти к

</Link>

<a

href="https://github.com/bogdandrienko"

className="btn btn-outline-secondary btn-md px-4"

>

Узнать об авторе побольше

</a>

</div>

</div>

</div>

</Base1>

);

}

export default Home;

Ну и напоследок напишем страницы с редюсерами и получением данных:

export function Books() {

const dispatch = useDispatch();

const books = useSelector((state) => state.books);

const [paginateObj, setPaginateObj] = useState({

page: 1,

limit: 3,

count: 1,

});

useEffect(() => {

console.log(books);

}, [books]);

useEffect(() => {

console.log(paginateObj);

}, [paginateObj]);

useEffect(() => {

getBooks();

}, [paginateObj.page]);

function updateData() {

dispatch({ type: constants.C\_Books.reset });

}

async function getBooks() {

if(books.load !== true){

dispatch(

utils.ConstructorActionRedux(

`/api/books/?page=${paginateObj.page}&limit=${paginateObj.limit}`,

"GET",

{},

5000,

constants.C\_Books

)

)

}

}

return (

<base.Base1>

<div>

<div className="container card p-0">

<div className="card-header lead">

Список всех книг

<button

onClick={updateData}

className="btn btn-sm btn-outline-secondary m-1 p-1"

>

<i className="fa-solid fa-rotate"></i>

</button>

</div>

{books.load === true && (

<div className="card-header">

<ui.Loader1 color="text-success" />

</div>

)}

<div className="d-flex justify-content-center text-center">

<ui.Paginator3

topBooks={books}

paginateObj={paginateObj}

setPaginateObj={setPaginateObj}

/>

</div>

<div className="card-body">

<div className="row row-cols-1 row-cols-lg-3 align-items-stretch g-4 py-5">

{books.data && books.data["object\_list"] ? (

books.data["object\_list"].map((item) => (

<div key={item.id} className="col">

<Link to={`/books/${item.id}`} className="text-decoration-none text-dark">

<div className="card card-cover h-100 overflow-hidden text-white bg-dark rounded-5 shadow-lg custom\_card">

<div className="d-flex flex-column h-100 p-5 pb-3 text-shadow-1">

{/\* {deleteBook.load === true ? (

<ui.Loader1 color="text-danger" />

) : (

<button

onClick={() => removeBook(item.id)}

className="btn btn-sm btn-outline-danger text-end w-25"

>

удалить

</button>

)} \*/}

<h2 className="pt-5 mt-5 mb-4 display-6 lh-1 fw-bold">

{item.title}

</h2>

<ul className="d-flex list-unstyled mt-auto">

<li className="me-auto">

<img

src="https://github.com/twbs.png"

alt="Bootstrap"

width="32"

height="32"

className="rounded-circle border border-white"

></img>

{item.category2.map((category) => (<button key={category.id} className="btn btn-sm btn-outline-secondary">{category.title}</button>))}

</li>

<li className="d-flex align-items-center me-3">

<svg

className="bi me-2"

width="1em"

height="1em"

></svg>

<small>{utils.SliceDescription(item.description)}</small>

</li>

<li className="d-flex align-items-center">

<svg

className="bi me-2"

width="1em"

height="1em"

></svg>

<small className="btn btn-sm btn-outline-secondary">{item.upload\_author1.username}</small>

</li>

</ul>

<div className="">

<Link to={`/books/${item.id}`} className="text-decoration-none text-light btn btn-md btn-primary w-100 m-1 p-1">подробнее</Link>

</div>

</div>

</div>

</Link>

</div>

))

) : (

<ui.Alert.Empty>...данных нет..</ui.Alert.Empty>

)}

</div>

<div className="d-flex justify-content-center text-center">

<ui.Paginator3

topBooks={books}

paginateObj={paginateObj}

setPaginateObj={setPaginateObj}

/>

</div>

</div>

<div className="card-footer">

{books.fail === true && (

<ui.Alert.Warning>{books.fail}</ui.Alert.Warning>

)}

{books.error === true && (

<ui.Alert.Danger>{books.error}</ui.Alert.Danger>

)}

</div>

</div>

</div>

</base.Base1>

);

}

Недостающие участки кода есть внутри предыдущих проектов на лекциях. Мы кое-что забыли? Конечно, авторизация и выход из приложения. Маршруты то уже готовы.

import React, { useState } from "react";

import { useSelector, useDispatch } from "react-redux";

import Base, {Base1} from "../components/Base";

import axios from "axios";

import {useNavigate} from "react-router-dom";

import \* as bases from '../components/Base'

import \* as constants from "../components/Constants";

function Login() {

const navigate = useNavigate();

const dispatch = useDispatch();

// localstorage

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

async function Formdata(form) {

form.preventDefault();

const formData = new FormData();

formData.append("username", username);

formData.append("password", password);

const response = await axios.post(`/api/login/`, formData);

console.log(`ВХОД ${response.data.response.access}`)

localStorage.setItem("token", response.data.response.access);

dispatch({ type: bases.CONST\_USER\_LOGIN.data, payload: response.data.response.access }); // ЗАГРУЗКА

dispatch({ type: constants.C\_Token.data, payload: response.data.response.access });

}

function TogglePasswordVisibility(uuid="") {

let x = document.getElementById(uuid);

if (x.type === "password") {

x.type = "text";

} else {

x.type = "password";

}

}

function GetData() {

const accessToken = localStorage.getItem('token')

axios.get("/backend\_api/get\_books/", { headers: {

'Authorization': `Bearer ${accessToken}`

}}).then(

(response)=> {

console.log(response);

}

).catch((error)=> {

localStorage.removeItem("token");

});

}

return (

<Base1>

<main className="custom\_main\_1">

<div className="container col-xl-10 col-xxl-8 px-4 py-5">

<div className="row align-items-center g-lg-5 py-5">

<div className="col-lg-7 text-center text-lg-start">

<h1 className="display-4 fw-bold lh-1 mb-3">

Вход

</h1>

<p className="col-lg-10 fs-4">

... тут будет умная цитата ...

</p>

</div>

<div className="col-md-10 mx-auto col-lg-5">

<form className="p-4 p-md-5 border rounded-3 bg-light" onSubmit={Formdata}>

<div className="form-floating mb-3">

<input

type="text"

className="form-control"

id="floatingInput"

placeholder="name"

min="8"

max="16"

required

value={username}

onChange={(event)=> setUsername(event.target.value)}

/>

<label for="floatingInput">Имя пользователя</label>

</div>

<div className="form-floating mb-3">

<input

type="password"

className="form-control"

id="floatingPassword"

placeholder="Password"

min="8"

max="16"

required

value={password}

onChange={(event)=> setPassword(event.target.value)}

/>

<label for="floatingPassword">Введите пароль от аккаунта</label>

</div>

<label>показать пароль</label>

<input onClick={()=> TogglePasswordVisibility("floatingPassword")} type="checkbox" id="vehicle1" name="vehicle1" value="Bike" placeholder="показать пароль"></input>

<button className="w-100 btn btn-lg btn-primary" type="submit">

Войти

</button>

</form>

</div>

</div>

</div>

</main>

</Base1>

);

}

export default Login;

Остатки добивайте по своему усмотрению.

1. Финальные настройки и публикация проекта на ubuntu server. Django-environ

Приведём наш проект к финальным настройкам: начать стоит с переключения в режим «Продакшн», сбора статики и настройки виртуальной-среды – основную документацию стоит почитать в интернете, но если быть кратким, то нужно создать в корне файл «.env», в настройках проекта применить что-то типа -

env = environ.Env(

DEBUG=(bool, False),

SECRET\_KEY=(str, "secret key")

)

environ.Env.read\_env()

SECRET\_KEY = env('SECRET\_KEY')

DEBUG = env('DEBUG')

В файле указываем:

DEBUG=off

SECRET\_KEY=0x!b#(1\*cd73w$&azzc6p+essg7v=g80ls#z&xcx\*mpemx&@9$

DATABASE\_NAME=db\_name

DATABASE\_USER=db\_user

DATABASE\_PASSWORD=password

DATABASE\_HOST=localhost

DATABASE\_PORT=5432

Данные стоит заменить на соответствующие.

Ubuntu.

Для развёртывания выбрали последнюю LTS версию Ubuntu, поэтому скачиваем её образ (.iso) с официального сайта: <https://ubuntu.com/>

Ставить мы её будем на виртуальную машину, под поддержкой бесплатного VBox, также качаем последнюю версию с сайта: <https://www.virtualbox.org/>

Производим стандартную установку и развёртывание операционной системы. Не забываем создать пользователя с правами доступа и пароль к нему.

Входим в систему и последовательно выполняем команды:

Обновляем пакеты и узнаём свой ip –адрес виртуальной машины –

sudo apt-get update -y

sudo apt upgrade -y

ip a

Дальше нужно установить ssh –сервер, чтобы подключиться со своей машины и удобнее было перекидывать файлы или вставлять команды –

sudo apt -y install openssh-server

sudo systemctl start ssh

sudo systemctl enable ssh

sudo reboot

Дожидаемся перезапуска системы, устанавливаем ssh-клиент <https://www.bitvise.com/>

Параметры подключения стандартные: логин и пароль те, что мы создали, ip-адрес, который мы узнали предыдущей командой и стандартный порт для ssh. Подключаемся и запускаем командную строку.

Теперь нужно установить стандартные библиотеки и зависимости –

sudo apt -y install net-tools htop git curl nginx

sudo apt -y install gunicorn python3-pip python3-dev python3-venv build-essential libpq-dev unixodbc-dev postgresql postgresql-contrib

Даём права пользователю –

sudo usermod -aG bogdan www-data

Теперь переходим к копированию проекта, можно прямо через bitwise ssh, скопировать папку проекта в корень. Не забудьте удалить виртуальное окружение и ноду.

«Воссоздаём» проект –

cd ~

mkdir web

cd web

python3 -m venv env

source env/bin/activate

pip install --upgrade pip

pip install wheel

pip install -r requirements.txt

Прежде чем запускать приложение или миграции, нужно создать PostgreSQL –

sudo su - postgres

createuser django\_user

createdb django\_db -O django\_user

psql django\_db

alter user django\_user with password 'django\_db\_password';

\q

exit

По аналогии ранее, выполняем миграции, создаём суперпользователя и запускаем отладочный веб-сервер. Проверяем его.

Теперь к настройке gunicorn и nginx –

Создаём «сокет», т.е. то, к чему будет подключать django для запросов –

sudo nano /etc/systemd/system/gunicorn.socket

Откроется файл для редактирования, нужно полностью заменить на:

[Unit]

Description=gunicorn socket

[Socket]

ListenStream=/run/gunicorn.sock

[Install]

WantedBy=sockets.target

Далее настраиваем «сервис», который и будет пересылать запросы –

sudo nano /etc/systemd/system/gunicorn.service

[Unit]

Description=Gunicorn for the Django example project

Requires=gunicorn.socket

After=network.target

[Service]

Type=notify

User=bogdan

Group=www-data

RuntimeDirectory=gunicorn

WorkingDirectory=/home/bogdan/web

ExecStart=/home/bogdan/web/env/bin/gunicorn --workers 3 --bind unix:/run/gunicorn.sock backend\_settings.wsgi:application

ExecReload=/bin/kill -s HUP $MAINPID

KillMode=mixed

TimeoutStopSec=5

PrivateTmp=true

[Install]

WantedBy=multi-user.target

Внимательно следите за папками, пользователей. Расчёт количества «воркеров», т.е. работников идёт из: количество ядер \* 2 + 1, для примера машины с 2 ядрами = 2 \* 2 + 1 = 5.

Запускаем и обновляем все сервисы и демоны.

sudo systemctl daemon-reload

sudo systemctl start gunicorn

sudo systemctl enable --now gunicorn.service

sudo systemctl daemon-reload

sudo systemctl restart gunicorn

sudo systemctl status gunicorn.service

Редактируем nginx – прокси сервер и то, что будет получать запрос изначально, плюсом он будет отдавать статику браузеру –

sudo nano /etc/nginx/sites-available/192-168-80-131-http.conf

Тут для примера идёт ip адрес 192.168.80.131.

Заставляем «слушать» nginx 80 порт и отдавать данные пользователям и 80 порт и отдавать данные пользователям и gunicorn.

server {

listen 80;

listen [::]:80;

server\_name localhost 127.0.0.1 192.168.80.131;

root /home/bogdan/web;

location /.well-known/acme-challenge/ {}

location /favicon.ico {

alias /home/bogdan/web/static/logo.png;

access\_log off; log\_not\_found off;

expires max;

}

location /robots.txt {

alias /home/bogdan/web/static/robots.txt;

access\_log off; log\_not\_found off;

expires max;

}

location /static/ {

alias /home/bogdan/web/static/;

expires max;

}

location /media/ {

alias /home/bogdan/web/static/media/;

expires max;

}

location / {

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header Host $http\_host;

proxy\_redirect off;

proxy\_buffering off;

proxy\_pass http://unix:/run/gunicorn.sock;

}

}

Финальные штрихи, создаём символьную ссылку, даём права и перезагружаем службы и системы –

sudo ln -s /etc/nginx/sites-available/192-168-80-131-http.conf /etc/nginx/sites-enabled/192-168-80-131-http.conf

sudo service nginx start

sudo systemctl status nginx.service

sudo ufw allow 'Nginx Full'

sudo systemctl reload nginx.service

sudo nginx -t

sudo reboot

Похоже, что на этом всё! Ждём перезагрузки, и входим по ip адресу машины в своём браузере. Если всё сделано верно, то Вам откроется последняя версия разработанного Вами веб-приложения.