

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной
математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Программная инженерия»

Студент: А. У. Оскенбаева
Преподаватель: А. А. Кухтичев
Группа: М80-207М-23
Дата:
Оценка:
Подпись:

Москва, 2025

Задание курсового проекта

Задача: Требуется разработать современное веб-приложение. Основные требования:

- Поднять веб-сервер (nginx). Отдавать статические файлы (логотип и т.д.) по `/static/`. Настроить проксирование запросов на сервер-приложение по отдельному URL;
- В конфиге nginx создать `location`, которое будет ходить на Django-приложение;
- Поднять сервер-приложение;
- Создать базу данных в PostgreSQL; Написать классы-модели, мигрировать;
- Организовать приём и передачу сообщений с помощью формата JSON, используя REST;
- Реализовать метод API для загрузки файла, использовать для хранения файла облачное S3 хранилище, создать `location` в Nginx для раздачи загруженных файлов, реализовать обработчик в приложении для проверки прав доступа к файлу;
- Реализовать OAuth2-авторизацию для двух любых социальных сетей, навесить декоратор, проверяющий авторизацию при вызовах API;
- Покрыть тестами все вьюхи и по желанию другие функции; Написать selenium-тест (найти элемент + клик на элемент); Использовать mock-объект при тестировании; Использовать factory boy; Узнать степень покрытия тестами с помощью библиотеки coverage;
- Развернуть и наполнить тестовыми данными Elasticsearch; Реализовать поиск по пользователям, продуктам (сущностям); Реализовать метод API для поиска по указанным сущностям и создать страницу HTML с вёрсткой для поиска и отображения результатов;
- Установить и поднять centrifugo; Подключить centrifugo к проекту на стороне клиента и сервера; Организовать отправку/получение сообщений с помощью centrifugo;
- Установить docker и docker-compose; Создание Dockerfile для Django-приложения; Создание docker-compose для проекта:
 - nginx;
 - База данных;

- Django-приложение;
- elasticsearch;
- Создание Makefile для проекта.

Тема курсовой: Онлайн платформа Beauty Academy.

Вариант веб-сервера: nginx.

Вариант сервера-приложений: Django.

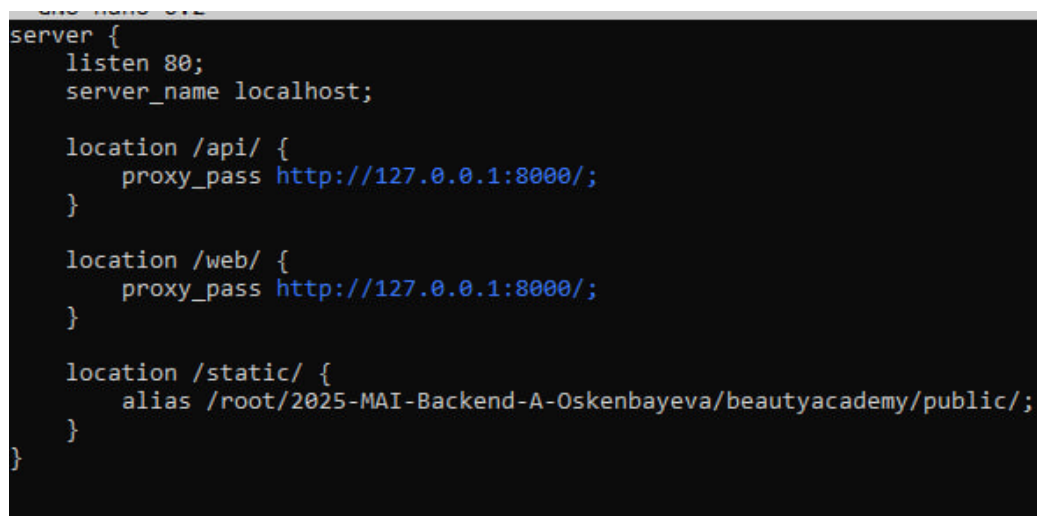
Вариант базы данных: PostgreSQL.

1 Веб-сервер

В проекте используется **Nginx** в качестве веб-сервера и обратного прокси для Django-приложения Beauty Academy.

Конфигурация включает:

- Проксирование API-запросов к Django-приложению:
 - * Все запросы к `/api/` направляются на порт 8000, где работает **runserver**.
 - * Настроен `proxy_pass` в конфиге `nginx`.
- Обслуживание статики:
 - * HTML и изображения отдаются через `/web/` с директории `public/`.
- Разделение логики через Nginx:
 - * `/api/` — для API-запросов.
 - * `/web/` — для HTML и фронтенда.
- Сервер слушает на порту 80, проксирует на порт 8000.
- Проверка API через `curl` и работу через Gunicorn + Nginx.



```
server {  
    listen 80;  
    server_name localhost;  
  
    location /api/ {  
        proxy_pass http://127.0.0.1:8000/  
    }  
  
    location /web/ {  
        proxy_pass http://127.0.0.1:8000/  
    }  
  
    location /static/ {  
        alias /root/2025-MAI-Backend-A-Oskenbayeva/beautyacademy/public/  
    }  
}
```

Рис. 1: Пример моделей Django для проекта Beauty Academy

2 Сервер-приложений

Модели данных:

- **Users** — информация о пользователях (логин, пароль, роль, дата рождения, номер телефона, адрес, и т.д.)

- **Courses** — курсы на платформе (название, описание, категория, режим, длительность)
- **Categories** — категории курсов (макияж, уход за кожей, брови и т.д.)

```

1  from django.db import models
2  from django.contrib.auth.models import User
3
4  class Profile(models.Model):
5      ROLE_CHOICES = [
6          ('student', 'Student'),
7          ('master', 'Master'),
8          ('coordinator', 'Coordinator'),
9      ]
10     user = models.OneToOneField(User, on_delete=models.CASCADE)
11     role = models.CharField(max_length=20, choices=ROLE_CHOICES)
12     bio = models.TextField(blank=True)
13     avatar = models.ImageField(upload_to='avatars/', blank=True, null=True)
14     phone = models.CharField(max_length=20, blank=True)
15     date_of_birth = models.DateField(blank=True, null=True)
16     address = models.CharField(max_length=255, blank=True)
17     created_at = models.DateTimeField(auto_now_add=True)
18     updated_at = models.DateTimeField(auto_now=True)
19
20     def __str__(self):
21         return f"{self.user.username} ({self.role})"
22
23     class Category(models.Model):
24         name = models.CharField(max_length=100, unique=True)
25         description = models.TextField()
26         parent = models.ForeignKey('self', null=True, blank=True, on_delete=models.SET_NULL)
27         icon = models.ImageField(upload_to='icons/', null=True, blank=True)
28
29         def __str__(self):
30             return self.name
31
32     class Course(models.Model):
33         DIFFICULTY_LEVELS = [
34             ('beginner', 'Beginner'),
35             ('intermediate', 'Intermediate'),
36             ('advanced', 'Advanced'),
37         ]
38
39         STATUS_CHOICES = [
40             ('planned', 'Planned'),
41             ('in_progress', 'In Progress'),
42             ('completed', 'Completed'),
43         ]
44
45         DELIVERY_MODES = [
46             ('online', 'Online'),
47             ('offline', 'Offline'),
48         ]
49
50         title = models.CharField(max_length=200)
51         description = models.TextField()

```

Рис. 2: Models.py

Эндпоинты API:

- `/api/profile/` — информация о пользователе (GET, POST)
- `/api/courses/` — список курсов (GET, POST)

- `/api/categories/` — категории курсов (GET, POST)
- Поиск: `/api/profile/?q=студент`

```

1  from django.urls import path
2  from core import views
3
4  urlpatterns = [
5      # Profile
6      path('profile/', views.profile_list_view),
7      path('profile/create', views.profile_create_view),
8      path('profile/search/', views.profile_search_view),
9
10     # Course
11     path('course/', views.course_list_view),
12     path('course/create', views.course_create_view),
13     path('course/search/', views.course_search_view),
14
15     # Category
16     path('category/', views.category_list_view),
17     path('category/create', views.category_create_view),
18     path('category/search/', views.category_search_view),
19 ]
20

```

Рис. 3: Маршруты API в файле `urls.py`

3 База данных и ORM

Конфигурация:

- Используется PostgreSQL 15.
- Подключение настроено в `settings.py`:
 - * `ENGINE = 'django.db.backends.postgresql'`
 - * `HOST = 'db'`
 - * `USER = 'beauty_user' NAME = 'beautyacademy_db'`
 - * Подключение проверено через `psql`, команды `\l` и `\du`.
- Суперпользователь: `admin / admin12345`

Миграции:

1. Первая миграция создаёт таблицы всех моделей:

- Profile, Course, Category, CourseSchedule, Attendance

2. Проверка миграций:

- python manage.py makemigrations
- python manage.py migrate

3. Проверка через PostgreSQL:

- sudo -u postgres psql -d beautyacademy_db\dt-
- SELECT * FROM core_profile LIMIT 5;

Генерация тестовых данных:

- Используется через Django-админку: /admin/
- Данные вводились вручную (имя, курс, категория)
- Проверено отображение всех моделей в админке:
 - * Profile, Course, Category, CourseSchedule
- Использовались реальные названия категорий (например: Макияж, Брови)

```

1 import django.db.models.deletion
2 from django.conf import settings
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7
8     initial = True
9
10     dependencies = [
11         migrations.swappable_dependency(settings.AUTH_USER_MODEL),
12     ]
13
14     operations = [
15         migrations.CreateModel(
16             name='Category',
17             fields=[
18                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
19                 ('name', models.CharField(max_length=100, unique=True)),
20                 ('description', models.TextField(blank=True)),
21                 ('icon', models.ImageField(blank=True, null=True, upload_to='category_icons/')),
22                 ('parent_category', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.SET_NULL, related_name='subcategories', to='core.category')),
23             ],
24         ),
25         migrations.CreateModel(
26             name='Course',
27             fields=[
28                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
29                 ('title', models.CharField(max_length=200)),
30                 ('description', models.TextField()),
31                 ('delivery_mode', models.CharField(choices=[('online', 'Online'), ('offline', 'Offline')], default='online', max_length=10)),
32                 ('location', models.CharField(blank=True, max_length=255)),
33                 ('duration', models.CharField(max_length=50)),
34                 ('num_lessons', models.PositiveIntegerField(default=0)),
35                 ('difficulty', models.CharField(choices=[('beginner', 'Beginner'), ('intermediate', 'Intermediate'), ('advanced', 'Advanced')], default='beginner', max_length=20)),
36                 ('price', models.DecimalField(decimal_places=2, default=0.0, max_digits=8)),
37                 ('presentation', models.URLField(blank=True, null=True)),
38                 ('status', models.CharField(choices=[('planned', 'Planned'), ('in_progress', 'In Progress'), ('completed', 'Completed')], default='planned', max_length=20)),
39                 ('created_at', models.DateTimeField(auto_now_add=True)),
40                 ('updated_at', models.DateTimeField(auto_now=True)),
41                 ('category', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='courses', to='core.category')),
42                 ('students', models.ManyToManyField(blank=True, related_name='courses_enrolled', to=settings.AUTH_USER_MODEL)),
43             ],
44         ),
45         migrations.CreateModel(
46             name='Profile',
47             fields=[
48                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
49                 ('role', models.CharField(choices=[('student', 'Student'), ('teacher', 'Teacher'), ('curator', 'Curator'), ('parent', 'Parent')], max_length=20)),
50             ],
51         ),
52     ]

```

```

47     },
48     ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
49     ('role', models.CharField(choices=({'student', 'Student'}, {'teacher', 'Teacher'}, {'curator', 'Curator'}, {'parent', 'Parent'})), max_length=20)),
50     ('bio', models.TextField(blank=True)),
51     ('avatar', models.ImageField(blank=True, null=True, upload_to='avatars/')),
52     ('phone', models.CharField(blank=True, max_length=20)),
53     ('date_of_birth', models.DateField(blank=True, null=True)),
54     ('address', models.CharField(blank=True, max_length=255)),
55     ('created_at', models.DateTimeField(auto_now_add=True)),
56     ('updated_at', models.DateTimeField(auto_now=True)),
57     ('user', models.OneToOneField(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL)),
58     },
59 ),
60 migrations.CreateModel(
61     name='CourseSchedule',
62     fields=[
63         ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
64         ('day_of_week', models.CharField(max_length=10)),
65         ('start_time', models.TimeField()),
66         ('end_time', models.TimeField()),
67         ('course', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='schedules', to='core.course')),
68         ('instructor', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.SET_NULL, related_name='schedule_entries', to='core.profile')),
69         ('replacement', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.SET_NULL, related_name='replacement_schedule_entries', to='core.profile')),
70     ],
71 ),
72 migrations.AddField(
73     model_name='course',
74     name='instructors',
75     field=models.ManyToManyField(blank=True, related_name='courses_teaching', to='core.profile'),
76 ),
77 migrations.CreateModel(
78     name='Attendance',
79     fields=[
80         ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
81         ('lesson_date', models.DateField()),
82         ('attendance_status', models.CharField(choices=({'attended_paid', 'Attended and Paid'}, {'attended_not_paid', 'Attended but Not Paid'}, {'not_attended_paid', 'Not Attended but Paid'}, {'not_attended_not_paid', 'Not Attended and Not Paid'})), max_length=20)),
83         ('absence_reason', models.TextField(blank=True, null=True)),
84         ('grade', models.DecimalField(blank=True, decimal_places=2, max_digits=4, null=True)),
85         ('student', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='attendances', to=settings.AUTH_USER_MODEL)),
86         ('course', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='attendances', to='core.course')),
87     ],
88     options={
89         'unique_together': (('course', 'student', 'lesson_date'),),
90     },
91 ),
92 ],
93 )

```

Рис. 4: Миграции и тестовые данные в файле admin.py

4 Контейнеризация

Проект развёрнут с использованием **Docker** и **Docker Compose**.

Сервисы:

1. db (PostgreSQL)

- Используется официальный образ `postgres:15`
- Переменные окружения задаются в `docker-compose.yml`:
 - * `POSTGRES_DB = beautyacademy_db` `POSTGRES_USER = beauty_user` `POSTGRES_PASSWORD = beautyacademy`
 - * Данные сохраняются в том `pg_data`
- Настроен `healthcheck` для проверки доступности сервиса

2. web (Django)

- Собирается с помощью `Dockerfile`
- При старте выполняются миграции и создаётся суперпользователь
- Подключается к БД через параметры окружения
- Зависит от PostgreSQL (через `depends_on`)
- Настроен `healthcheck`

3. nginx

- Используется официальный образ `nginx:latest`
- Проксирует запросы:
 - * `/api/` → `web:8000`

- * /web/ → директория public/
- Монтируются конфиги и статика
- Зависит от web-приложения
- Настроен healthcheck

```
1  version: '3.9'
2
3  services:
4    db:
5      image: postgres:15
6      container_name: beautyacademy_db
7      environment:
8        POSTGRES_DB: beautyacademy_db
9        POSTGRES_USER: beauty_user
10       POSTGRES_PASSWORD: beauty_pass
11      volumes:
12        - pg_data:/var/lib/postgresql/data
13      healthcheck:
14        test: ["CMD", "pg_isready", "-U", "beauty_user"]
15        interval: 10s
16        retries: 5
17
18    web:
19      build: .
20      container_name: beautyacademy_web
21      command: python manage.py runserver 0.0.0.0:8000
22      volumes:
23        - ./app
24      ports:
25        - "8000:8000"
26      depends_on:
27        db:
28          condition: service_healthy
29      environment:
30        DEBUG: "1"
31        DJANGO_ALLOWED_HOSTS: "*"
32        DB_NAME: beautyacademy_db
33        DB_USER: beauty_user
34        DB_PASSWORD: beauty_pass
35        DB_HOST: db
36        DB_PORT: 5432
37
38    volumes:
39      pg_data:
40
```

Рис. 5: Фрагмент файла `docker-compose.yml`, содержащий конфигурацию сервисов базы данных PostgreSQL

5 Выводы

В рамках реализации проекта Beauty Academy я научилась разворачивать полноценное веб-приложение с использованием современных технологий. Было освоено построение структуры Django-проекта, реализация моделей, API и связей между сущностями. Я закрепила навыки работы с PostgreSQL, настройки миграций и админ-панели, а также научилась использовать Docker и docker-compose для упрощения запуска и деплоя.

Особое внимание было уделено настройке Nginx как обратного прокси и разделению маршрутов по функциональности. Я научилась реализовывать взаимодействие между сервисами, настраивать зависимости и healthchecks. Полученные знания пригодятся мне как при разработке собственных проектов, так и при работе в команде над большими системами.

Список литературы

1. *Официальная документация Django*
URL: <https://docs.djangoproject.com/en/5.2/> (дата обращения: 25.04.2025).
2. *Официальная документация Docker Compose*
URL: <https://docs.docker.com/compose/> (дата обращения: 01.05.2025).
3. *Официальная документация Selenium WebDriver* URL: <https://www.selenium.dev/doc>
(дата ораени: 02.05.2025).
4. *Официальная документация NGINX*
URL: <https://nginx.org/en/docs/> (дата обращения: 15.04.2025).