Автономная некоммерческая организация
Дополнительного профессионального
образования
Компьютерная Академия «ТОП» филиал
«Академия ТОП Уфа»

# Дипломная работа

по курсу «Веб-разработка на python»

на тему: «Разработка веб-сайта для образовательной платформы по программированию»

Выполнила студент Компьютерной Академии ТОП

Крохалева Аурика Алексеевна/_____
(ФИО) (подпись)

Дипломная работа допущена к защите и проверена на объем заимствования:

Директор филиала
АНО ДПО «Академия ТОП»
Игнатьева Азалия Фаритовна\ _____

Рук. учебной части филиала
АНО ДПО «Академия
Фатхинурова Светлана Форагатовна_____

Уфа, 2024г.

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

С развитием технологий и стремительным ростом спроса на специалистов в области программирования, кибербезопасности и машинного обучения, появляется потребность в онлайн-платформах, предлагающих качественное образование в этих направлениях. Важным аспектом является не только предоставление теоретических знаний, но и возможность практического применения навыков через интерактивные задания и проекты.

Цель данной дипломной работы — разработка веб-сайта для образовательной платформы, которая будет предоставлять курсы по Python, HTML, CSS, кибербезопасности и машинному обучению. Платформа позволит пользователям обучаться программированию и другим востребованным технологиям в удобном онлайн-формате, получая доступ к видеоурокам

Разработка веб-сайта будет осуществляться с использованием Python и фреймворка Django, а база данных PostgreSQL обеспечит надежность и высокую производительность. Платформа будет доступна пользователям с разным уровнем подготовки, от новичков до более опытных пользователей, стремящихся углубить свои знания.

# ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Обзор рынка образовательных платформ в области программирования

Рынок онлайн-образования в последние годы демонстрирует бурный рост, особенно в таких областях, как программирование, кибербезопасность и машинное обучение. Все больше людей стремятся освоить эти востребованные навыки, что обусловлено ростом числа технологических стартапов и компаний, нуждающихся в квалифицированных специалистах.

Образовательные платформы предлагают разнообразие курсов, от базового обучения программированию до более сложных тем, таких как машинное обучение и кибербезопасность. Однако часто пользователи сталкиваются с проблемой качества и актуальности контента, а также с нехваткой практических заданий, что затрудняет освоение материала.

Для успешного продвижения на этом рынке наша платформа должна предложить не только высококачественные курсы, но и поддержку студентов через онлайн-сообщество, регулярные вебинары. Кроме того, особое внимание будет уделено актуальности курсов, чтобы они соответствовали текущим требованиям и трендам в индустрии.

## 1.2. Целевая аудитория

Целевая аудитория платформы — это люди, желающие освоить или углубить свои знания в области программирования, кибербезопасности и машинного обучения. Среди них могут быть студенты, работающие специалисты, а также люди, решившие сменить профессию или повысить свою квалификацию.

Важно, чтобы платформа была доступна для пользователей с разным уровнем подготовки: от начинающих, которые хотят изучить основы программирования, до более опытных людей, желающих освоить сложные темы, такие как кибербезопасность или машинное обучение. Платформа будет предоставлять курсы с различным уровнем сложности.

Для привлечения аудитории важно создать удобный интерфейс, который позволит легко ориентироваться на сайте и быстро находить необходимые материалы.

Этот формат дипломной работы адаптирован для веб-платформы, которая предлагает курсы по Python, HTML, CSS, кибербезопасности и машинному обучению, и включает основные разделы, которые должны быть охвачены в дипломе.

# ГЛАВА 2. ОБЗОР ТЕХНОЛОГИЙ

## 2.1 Архитектура разрабатываемого программного обеспечения

Для реализации веб-сайта, на котором пользователи могут записываться на курсы, просматривать события, оставлять отзывы был выбран набор технологий, обеспечивающий эффективность, безопасность и удобство использования. Тщательно подобранные инструменты позволили быстро разработать масштабируемое и безопасное веб-приложение.

**1. Django:** Это высокоуровневый веб-фреймворк на языке Python, который предоставляет множество встроенных функций, таких как аутентификация, администрирование и работа с формами. Архитектура "MTV" (Model-Template-View) помогает разделять логику приложения на отдельные компоненты, что способствует упрощению разработки и последующей поддержки проекта.

**2.PostgreSQL:** Эта реляционная база данных была выбрана за её высокую производительность, надёжность и богатый функционал. PostgreSQL предоставляет поддержку сложных запросов, транзакций и масштабируемости, что идеально подходит для хранения и обработки большого объема данных, связанных с пользователями, курсами и событиями.

**3.HTML/CSS/JavaScript:** Для разработки интерфейса использованы стандартные веб-технологии. HTML обеспечивает структуру веб-страниц, CSS — визуальное оформление, а JavaScript добавляет интерактивность. В проекте также используется адаптивный дизайн для корректного отображения сайта на мобильных устройствах и планшетах, что улучшает удобство использования.

Таким образом, выбор этих технологий позволяет обеспечить надёжную и безопасную платформу, которая поддерживает высокий уровень взаимодействия с пользователем и может быть легко масштабирована.

Схема базы данных:

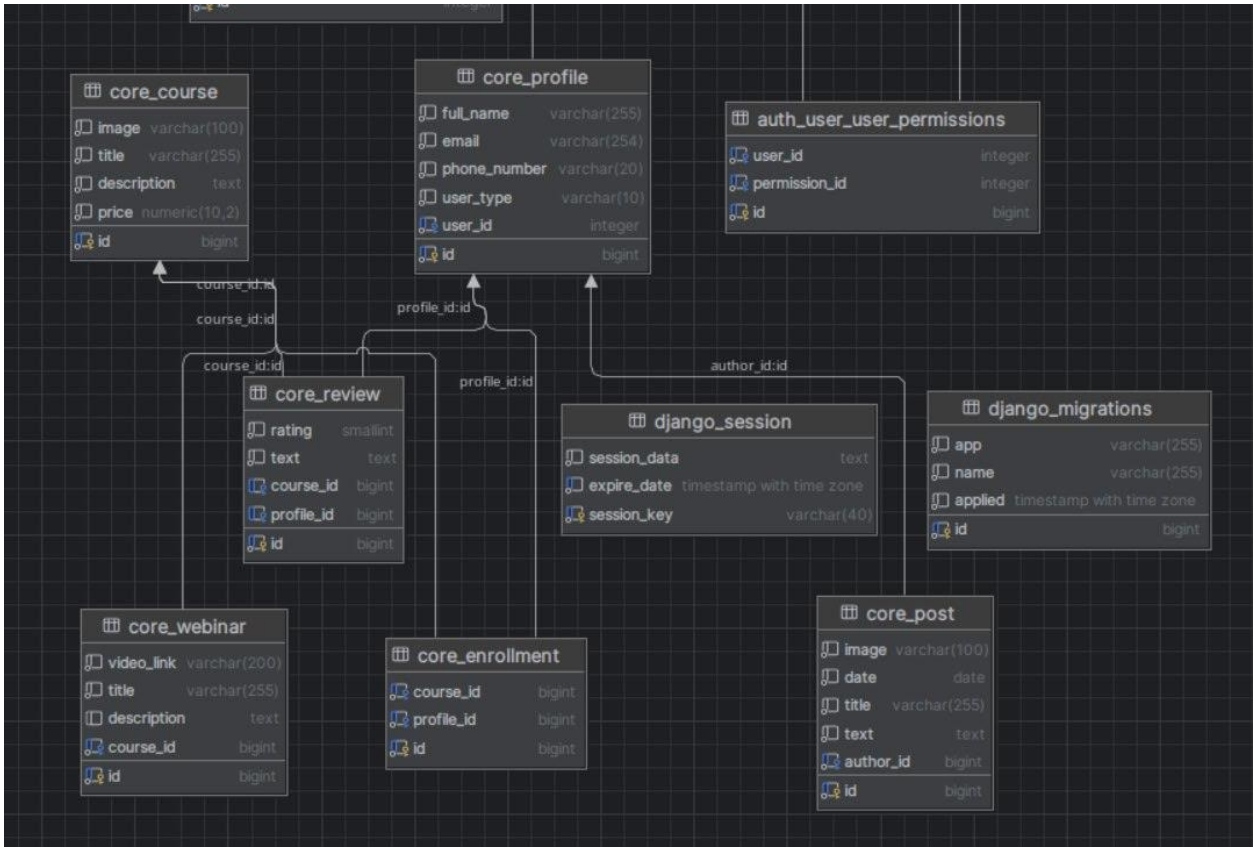


Рисунок 1 – Схема базы данных

1. **Profile**: Модель для профилей пользователей. Включает поля: full_name (полное имя), email (электронная почта), phone_number (номер телефона), user_type (тип пользователя). Связана с моделью User, предоставляющей функции аутентификации и управления пользователями.

2. **Course**: Модель для курсов. Включает поля: image (изображение курса), title (название курса), description (описание курса), price (стоимость курса). Хранит информацию о доступных курсах на платформе.

3. **Enrollment**: Модель для записи на курс. Связывает пользователя (через profile) с курсом (через course). Позволяет отслеживать, на какие курсы записались пользователи.

4. **Event**: Модель для хранения информации о событиях, таких как вебинары или мастер-классы. Включает поля: image (изображение события), date (дата события), title (название события), description (описание события).

5. **Review**: Модель для отзывов о курсах или преподавателях. Включает поля: profile (связь с пользователем, оставившим отзыв), course (связь с курсом), rating (оценка курса), text (текст отзыва).

6. **TeacherApplication**: Модель для анкет преподавателей. Включает поля: name (имя преподавателя), email (электронная почта), experience (опыт работы).

7. **Post**: Модель для публикаций на сайте (новости, объявления). Включает поля: image (изображение), date (дата публикации), author (автор поста), title (заголовок), text (текст публикации).

8. **Webinar**: Модель для хранения информации о вебинарах. Включает поля: video_link (ссылка на видео), title (название вебинара), description (описание вебинара), course (курс, к которому относится вебинар).

Эти модели используются для хранения и управления различной информацией на платформе, включая данные о пользователях, курсах, записях, событиях, отзывах и публикациях.

9.**index(request)**: Представление для главной страницы. Собирает данные о 5 самых последних курсах, постах, событиях и отзывах, а затем передает эти данные в шаблон index.html. Если пользователь аутентифицирован, проверяется его профиль и выводятся курсы, на которые он ещё не записан.

10.**about(request)**: Представление для страницы "О нас". Собирает 5 самых последних отзывов и передает эти данные в шаблон about.html.

11. **courses(request)**: Представление для страницы с курсами. Собирает все курсы, сортирует их по названию, и передает данные в шаблон courses.html. Также выводит 5 последних событий.

12. **course(request, course_id)**: Представление для страницы с деталями курса. Проверяет, записан ли текущий пользователь на курс. Если записан, отображает вебинары, связанные с курсом, и передает данные в шаблон course-details.html. Если не записан — выдает ошибку доступа.

13.**events(request)**: Представление для страницы с событиями. Собирает все события и сортирует их по дате. Передает данные в шаблон events.html.

14. **teacher_application(request)**: Представление для страницы с анкетой преподавателя. Обрабатывает форму с полями имени, email и опыта работы. После отправки формы сохраняет данные в модель TeacherApplication и перенаправляет на главную страницу.

15.**login_view(request)**: Представление для страницы входа. Обрабатывает форму входа, проверяет введенные данные, и если пользователь существует, аутентифицирует его и перенаправляет на главную страницу.

16.**register(request)**: Представление для страницы регистрации. Обрабатывает форму регистрации нового пользователя. Проверяет уникальность имени и email, создаёт нового пользователя, а затем аутентифицирует его и перенаправляет на главную страницу.

17.**logout_view(request)**: Представление для выхода из аккаунта. Осуществляет выход пользователя и перенаправляет на главную страницу.

18.**add_to_course(request)**: Представление для страницы записи на курс. Обрабатывает форму записи пользователя на курс, создавая соответствующую запись в модели Enrollment, и перенаправляет на главную страницу.

19.**profile(request)**: Представление для страницы профиля пользователя. Отображает профиль пользователя и курсы, на которые он записан. Также позволяет пользователю редактировать свои данные (имя, email, телефон).

20.**coaches(request)**: Представление для страницы с преподавателями. Собирает все записи о преподавателях и передает их в шаблон coaches.html.

21.**contacts(request)**: Представление для страницы с контактной информацией. Обрабатывает отправку контактной формы, сохраняя данные запросов в базе данных с помощью модели ContactForm.

22.**ticket(request, lesson_coach_id)**: Представление для страницы записи на занятие с преподавателем. Пользователь может указать свое имя и записаться на занятие, создавая запись в модели Enrollment.

23.**subscription(request, subscription_id)**: Представление для страницы оформления подписки. Пользователь выбирает курс и оформляет подписку, заполняя форму, после чего создается запись в модели Subscription.

24.**feedback(request, course_id)**: Представление для страницы с отзывами о курсе. Пользователь может оставить отзыв, который сохраняется в базе данных через модель Review. Отзывы добавляются без перезагрузки страницы с использованием JSON-ответа.

Эта архитектура позволяет пользователю легко взаимодействовать с платформой, записываться на курсы, получать информацию о преподавателях и оставлять отзывы. При этом технологии, такие как Django и PostgreSQL, обеспечивают безопасность, масштабируемость и стабильность системы.

## 2.2. Функциональные требования

Веб-сайт образовательной платформы должен включать следующие ключевые функциональные возможности, обеспечивающие удобство и эффективность использования для студентов, преподавателей и администраторов. Эти возможности помогут обеспечить качественный образовательный процесс, улучшат взаимодействие между пользователями и упростят администрирование платформы.

1. Возможность выбора курса и покупка
2. Возможность написать и посмотреть отзыв о курсах
3. Возможность просмотра ближайших мероприятий
4. Возможность подать заявку на вакансию учителя
5. Возможность создание личного аккаунта и вход
6. Возможность просмотра краткого описания курса

## 2.3. Нефункциональные требования

- Сайт должен быть удобен и интуитивно понятен для пользователей всех возрастов.
- Сайт должен иметь адаптивный дизайн для корректного отображения на всех устройствах.
- Сайт должен быть быстрым и отзывчивым для удобства пользователей.
- Сайт должен обеспечивать безопасное хранение персональных данных пользователей.

# ГЛАВА 3. РЕАЛИЗАЦИЯ

## 3.1. Структура проекта

Проект был организован с учётом принципов, заложенных в Django, что обеспечило структурированность и удобство разработки. Основная структура проекта выглядит следующим образом:

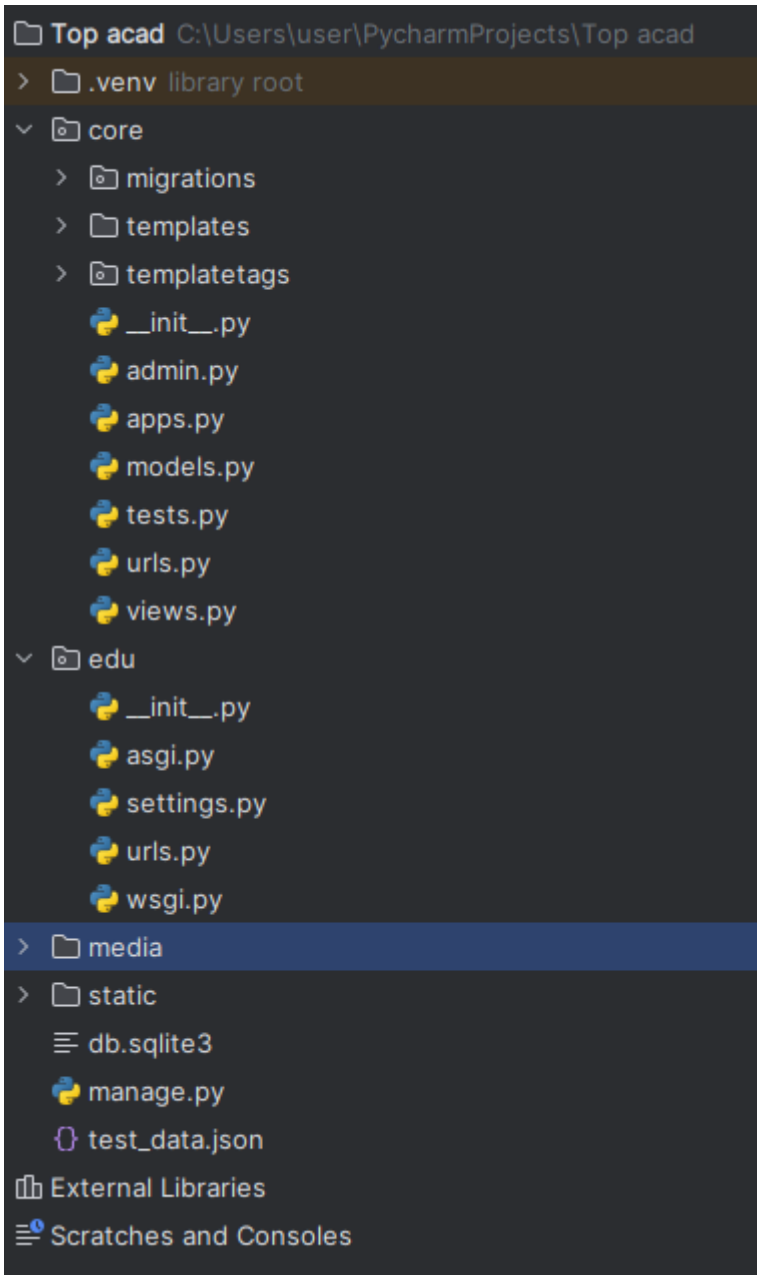


Рисунок 2 – Структура проекта

- **/core /**: Основная директория приложения, в которой реализуется функциональность платформы.

- **models.py**: Содержит определения моделей, которые представляют структуры данных, используемые в приложении. Здесь описаны такие сущности, как занятия, абонементы и отзывы.

- **views.py**: Логика обработки HTTP-запросов. Здесь определяются функции и классы, которые обрабатывают данные, передают их в шаблоны и возвращают ответы пользователю.

- **urls.py**: Определяет маршруты для приложения, связывая URL-адреса с соответствующими представлениями.

- **/templates/** и **/static/** на уровне корневой директории проекта используются для хранения глобальных шаблонов и статических файлов, которые могут использоваться во всех приложениях проекта.

- **settings.py**: Файл конфигурации, в котором определены настройки базы данных, статических файлов, приложений и другие параметры, влияющие на работу всего проекта.

- **urls.py**: Глобальный файл маршрутизации проекта, который объединяет маршруты всех приложений и направляет запросы к нужным представлениям.

## 3.2. Реализация функционала

Для обеспечения функционала, позволяющего пользователям оставлять свои данные для связи и выбирать абонементы, были реализованы следующие компоненты:

1.Покупка курса

Платформа предоставляет пользователям возможность выбрать и оформить покупку курса. Пользователи смогут выбрать курс, ознакомиться с его описанием и стоимостью, а затем пройти процесс оплаты, чтобы получить доступ к материалам курса.

2. Отзывы

- Модель отзывов: Для функционала отзывов была разработана модель, которая включает поля для имени пользователя, текста отзыва, рейтинга и связи с курсом (или преподавателем). Эта модель отвечает за хранение и отображение информации о полученных отзывах.
- Интерфейс добавления и отображения отзывов: Реализован интерфейс, позволяющий пользователям оставлять свои отзывы о занятиях. Отзывы отображаются на страницах курсов или преподавателей, что помогает новыми пользователям оценить качество обучения.

Эти функции значительно улучшают взаимодействие пользователей с веб-сайтом и помогают создавать более информативную и привлекательную платформу для образования.

## 3.3. Тестирование и отладка

Для обеспечения высокого качества работы веб-сайта была проведена комплексная процедура тестирования, охватывающая все аспекты функциональности приложения. Процесс включал как модульное, так и функциональное тестирование.

Модульное тестирование

Модульное тестирование сосредоточено на проверке отдельных компонентов приложения, таких как модели, представления и формы. Основные шаги включали:

- Тестирование моделей: Были написаны тесты для проверки корректности работы методов и полей моделей. Например, проверялась возможность правильного сохранения записей пользователей, отзывов, и записей на курсы в базе данных. Это позволило убедиться, что бизнес-логика, реализованная в моделях, работает как ожидается.
- Тестирование представлений: Разработаны тесты для проверки обработки запросов и правильного формирования ответов от сервера. Это включало проверку корректности отображения шаблонов (например, страницы с курсами, отзывами, событиями) в ответ на запросы пользователей.
- Тестирование форм: Были написаны тесты для проверки правильности работы форм. Включали тестирование валидации данных, таких как правильность введённых email-адресов, телефонов, текстов отзывов, и корректное сохранение данных в базе данных.

Функциональное тестирование

Функциональное тестирование проводилось для оценки работы всей системы в целом. Этот этап включал:

- Тестирование пользовательских сценариев: Были протестированы ключевые пользовательские сценарии, такие как:
    - Регистрация на курс.
    - Оставление отзывов.
    - Выбор и покупка курса.
    - Просмотр материала курса.

  Проверялась удобность и понятность интерфейса для пользователей.
- Тестирование интеграции: Проверено взаимодействие между различными компонентами системы. Например:
    - Правильность передачи данных о записях на курсы.
    - Корректная работа форм и моделей при сохранении данных в базу.

Тестирование показало, что все данные, введенные пользователем (например, в формах для записи на курсы), корректно передаются и сохраняются в базе данных.

- Проверка производительности: Проведено тестирование на время отклика и способности обрабатывать несколько одновременных запросов. Были проверены сценарии с высокой нагрузкой (например, большое количество пользователей, пытающихся записаться на курс одновременно). Результаты тестирования подтвердили, что система эффективно справляется с нагрузкой.

## Результаты тестирования

По итогам проведённого тестирования все основные функции веб-сайта продемонстрировали корректную работу. Ни одна критическая ошибка не была выявлена, что подтверждает высокое качество разработанной системы.

- Стабильность работы: Сайт прошёл все модульные и функциональные тесты. Все ключевые компоненты (запись на курсы, отзывы, форма обратной связи) работают корректно.
- Отсутствие критических ошибок: Во время тестирования не было найдено проблем, которые могли бы нарушить функциональность сайта.

Веб-сайт успешно прошёл все тесты и готов к запуску. Результаты тестирования также служат основой для дальнейшего мониторинга и улучшения приложения в будущем. На основании этих тестов можно будет уверенно заявить о надёжности и эффективности системы.Такой подход к реализации и тестированию помогает обеспечить высокое качество продукта и удобство для пользователей.

# ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы был успешно разработан веб-сайт для образования по программированию. Этот проект имеет все необходимые функциональные возможности, которые обеспечивают комфортное взаимодействие пользователей с сайтом:

1) **регистрация пользователей**. Позволяет клиентам создавать учётные записи для удобства работы с сайтом;
2) **покупка курса**. Интерфейс для покупки на различных курсов;
3) **отзывы о платформе и курсах**. Позволяет клиентам оставлять свои мнения и оценки, что способствует созданию обратной связи и улучшению качества услуг.

Реализация проекта с использованием Django и PostgreSQL обеспечила высокую производительность, безопасность данных и масштабируемость приложения. Django, как мощный веб-фреймворк, значительно упростил процесс разработки, а PostgreSQL обеспечил надёжное хранение данных и эффективную работу с ними.

Этот веб-сайт может стать эффективным инструментом для привлечения и удержания клиентов, а также для улучшения взаимодействия между студией и её посетителями. Удобный интерфейс и разнообразные функции обеспечивают положительный пользовательский опыт и способствуют росту популярности студии.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Django: docs.djangoproject.com
2. Официальная документация PostgreSQL.: postgresql.org/docs

# ПРИЛОЖЕНИЕ 1. СКРИНШОТЫ ИНТЕРФЕЙСА

Включены визуальные примеры пользовательского интерфейса для наглядного представления функциональности

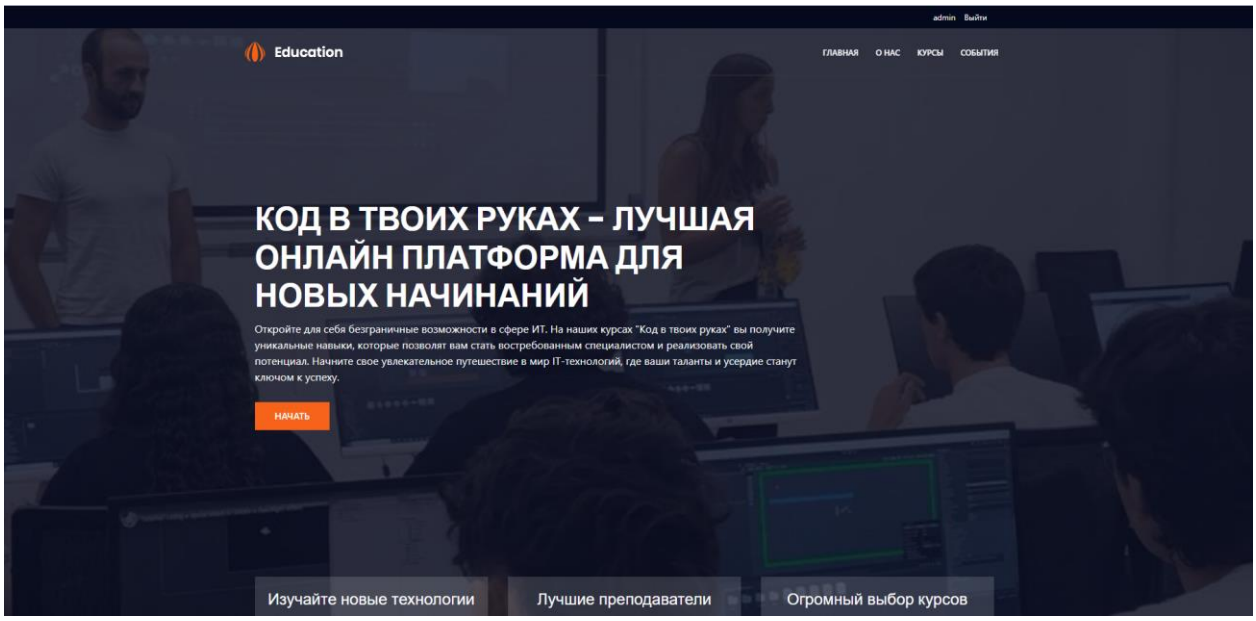


Рисунок 3 – Интерфейс сайта (главная страница)

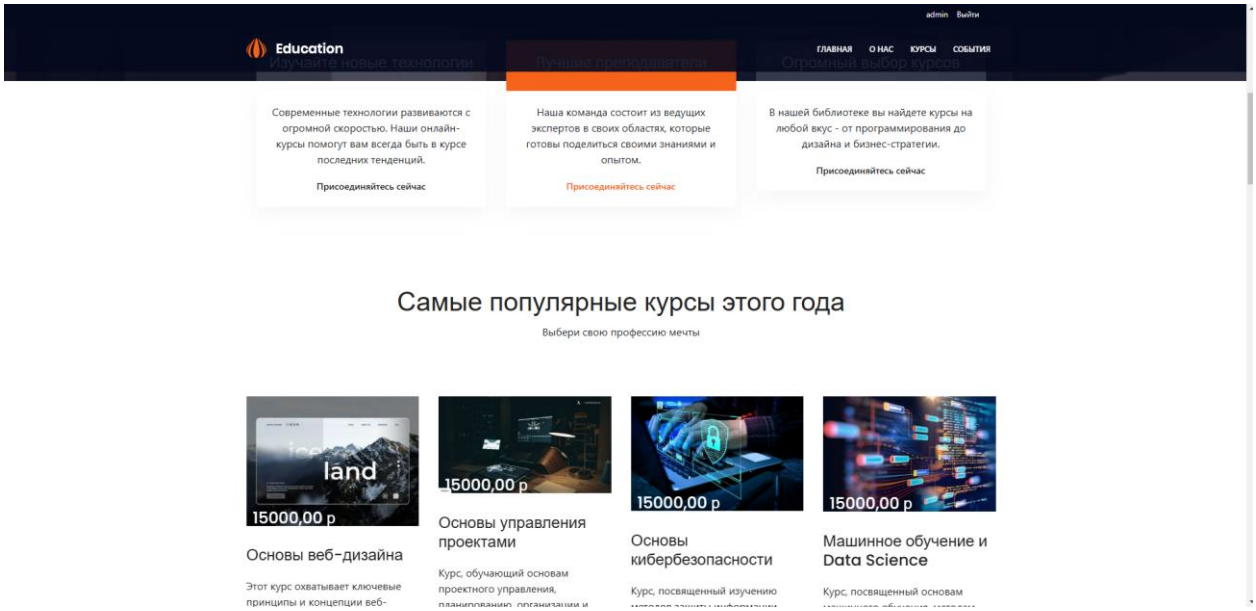


Рисунок 4 – Интерфейс сайта (главная страница)

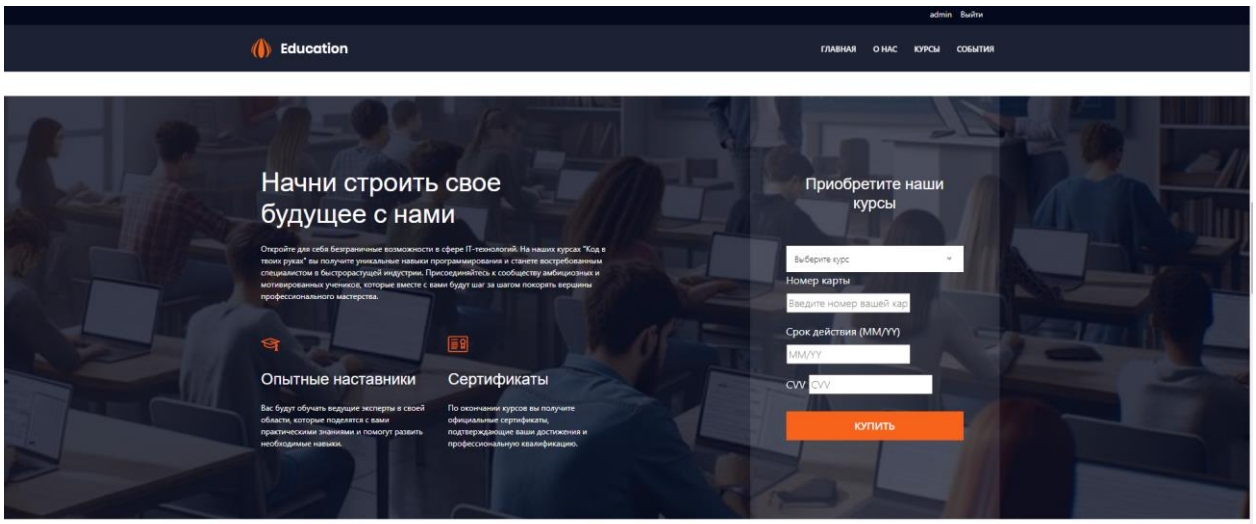


Рисунок 5 – Интерфейс сайта (главная страница)

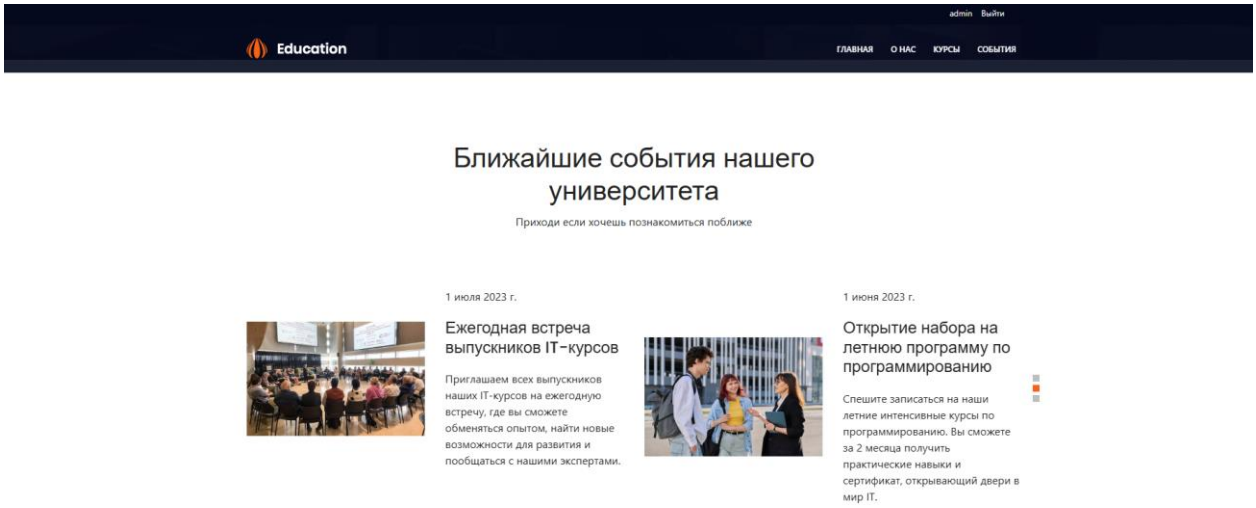


Рисунок 6 – Интерфейс сайта (главная страница)

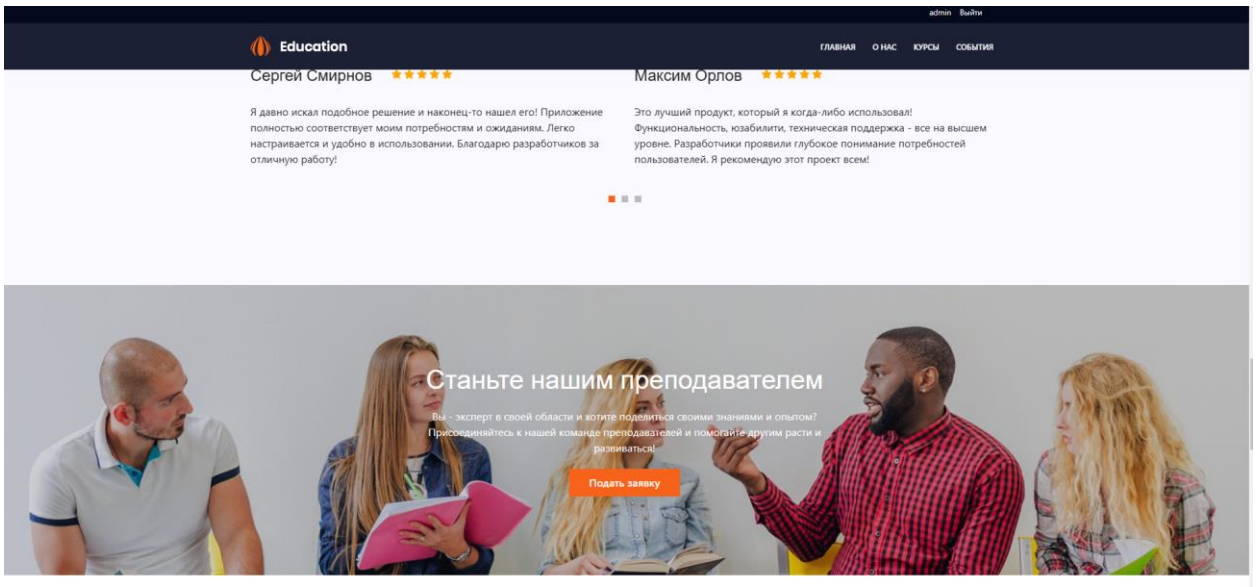


Рисунок 7 – Интерфейс сайта (главная страница)

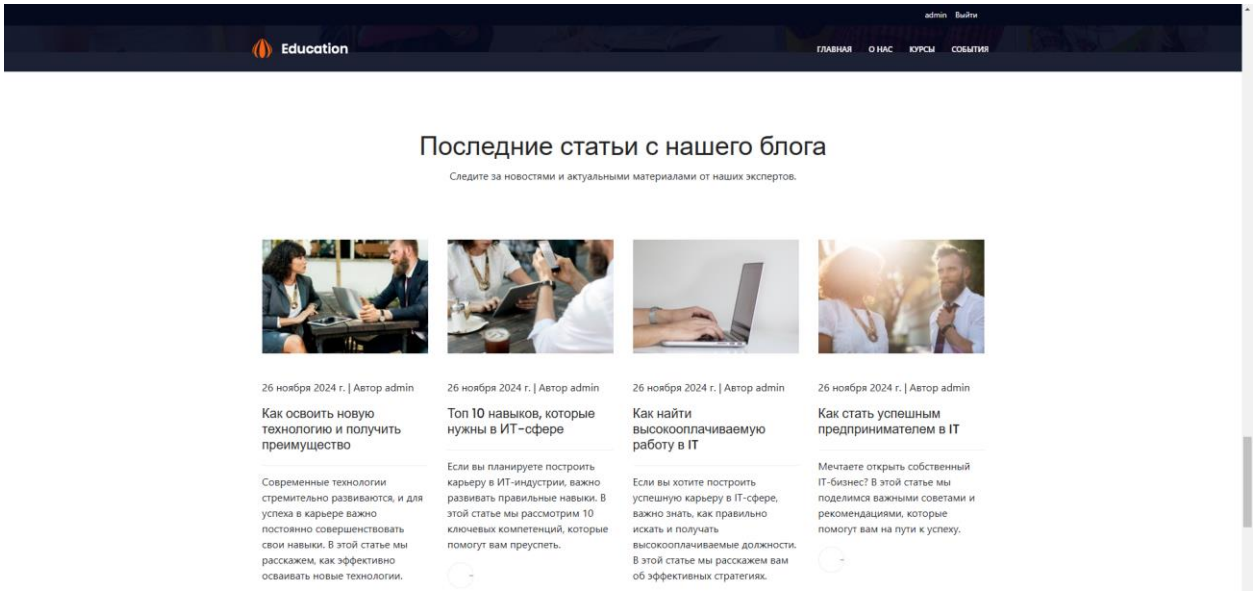


Рисунок 8 – Интерфейс сайта (главная страница)

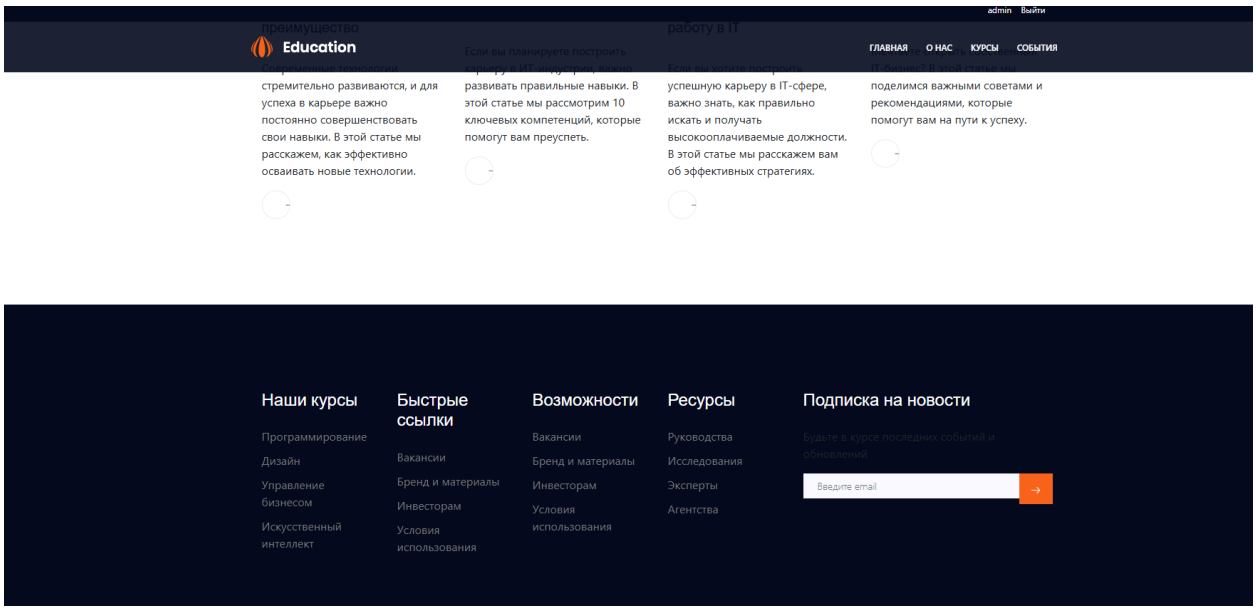


Рисунок 9 – Интерфейс сайта (главная страница)

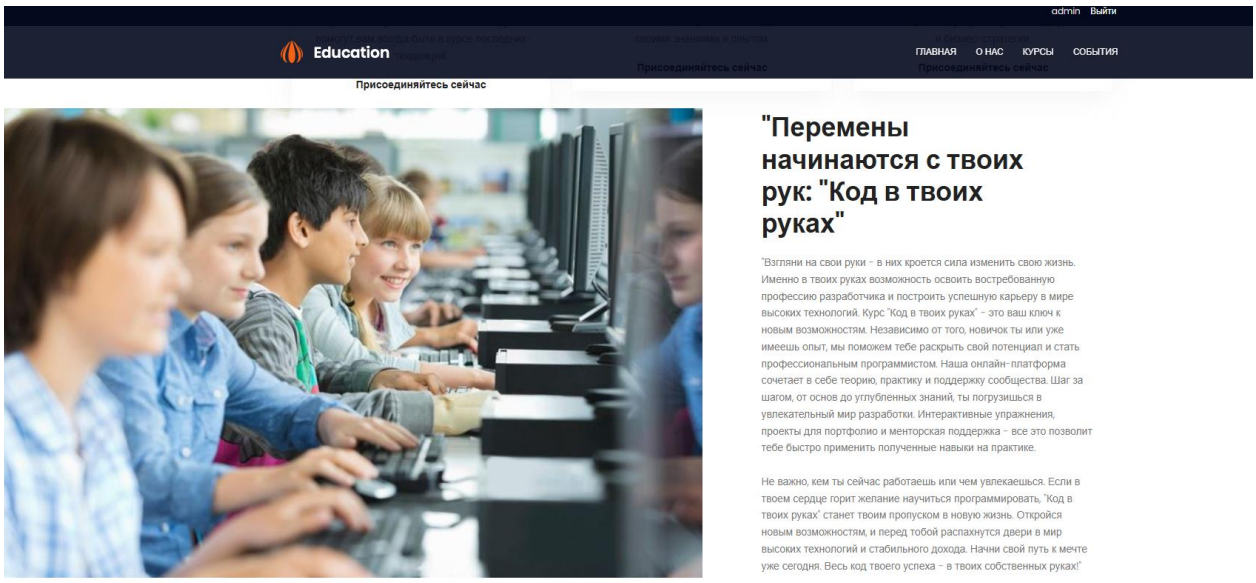


Рисунок 10 – Интерфейс сайта (О нас)

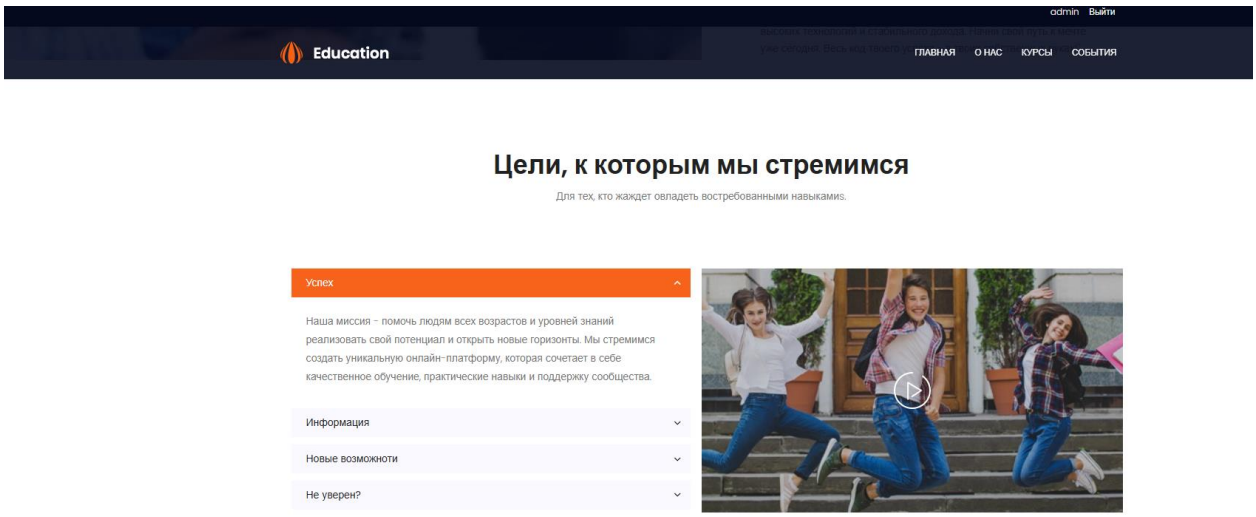


Рисунок 11 – Интерфейс сайта (О нас)

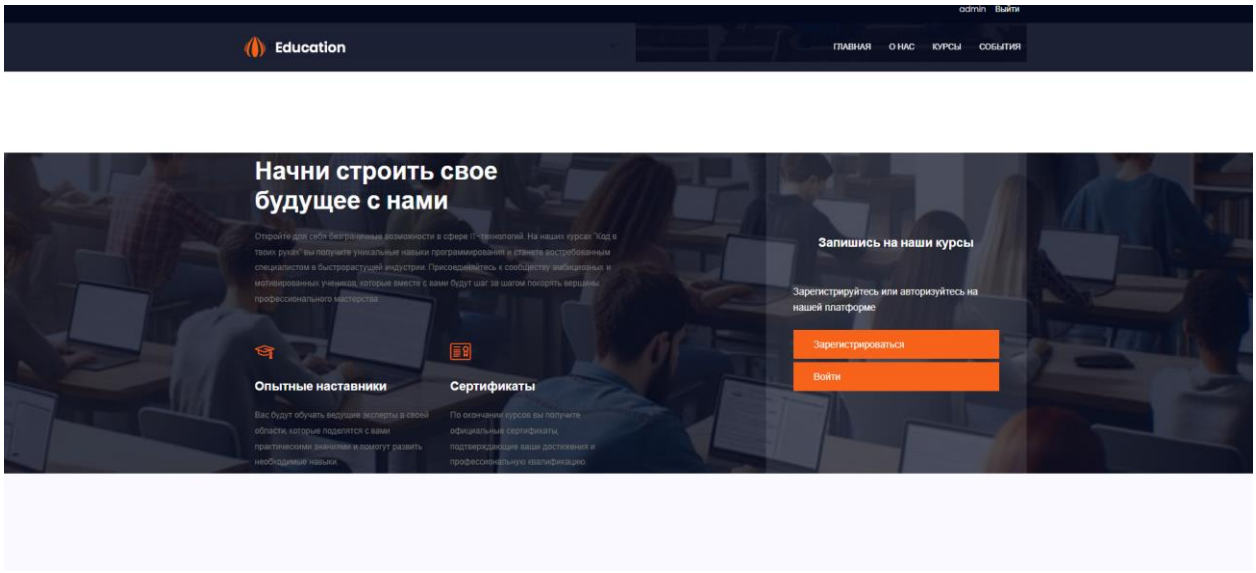


Рисунок 12 – Интерфейс сайта (О нас)

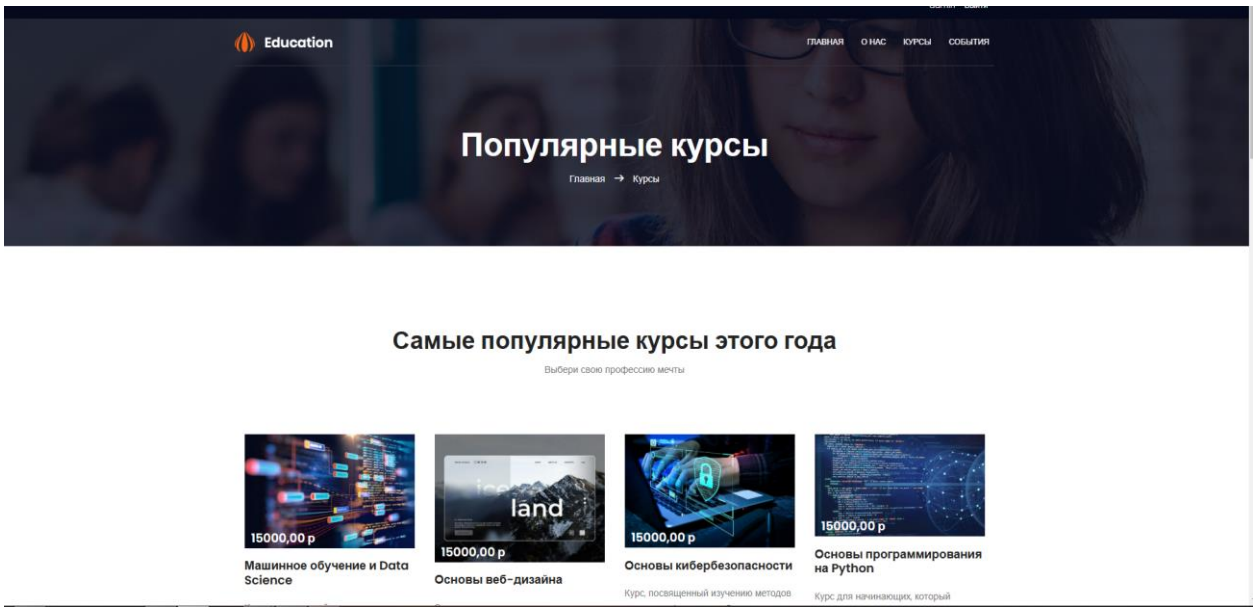


Рисунок 13 – Интерфейс сайта (Курсы)

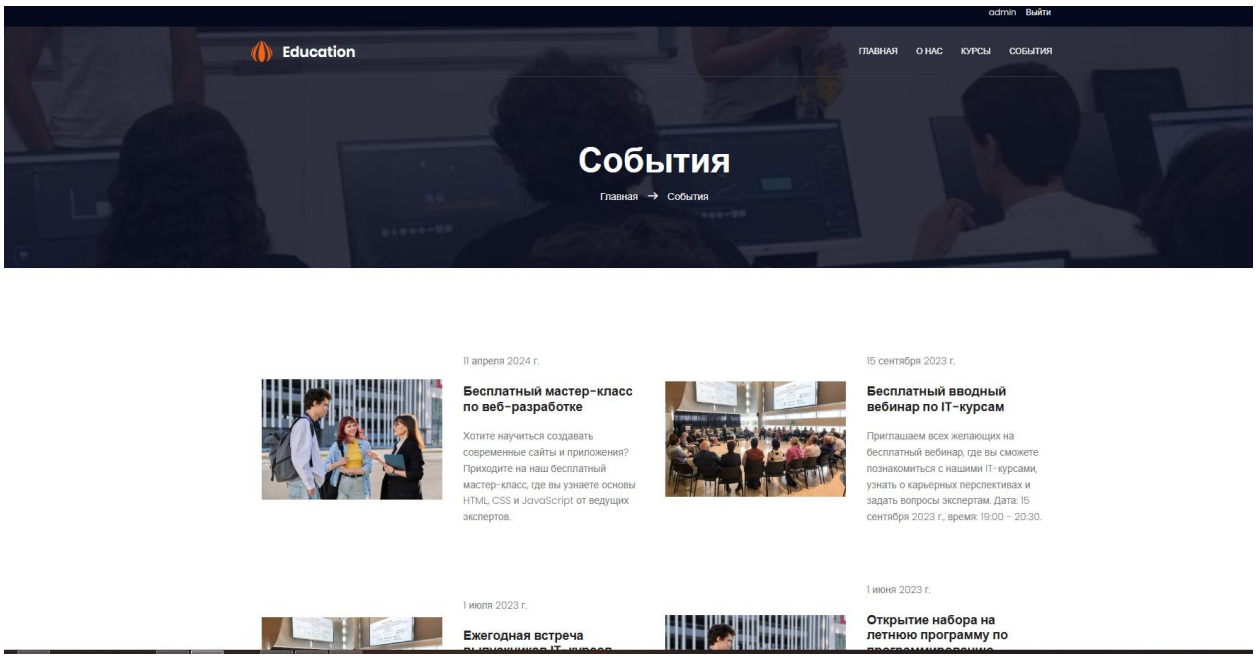


Рисунок 14 – Интерфейс сайта (События)

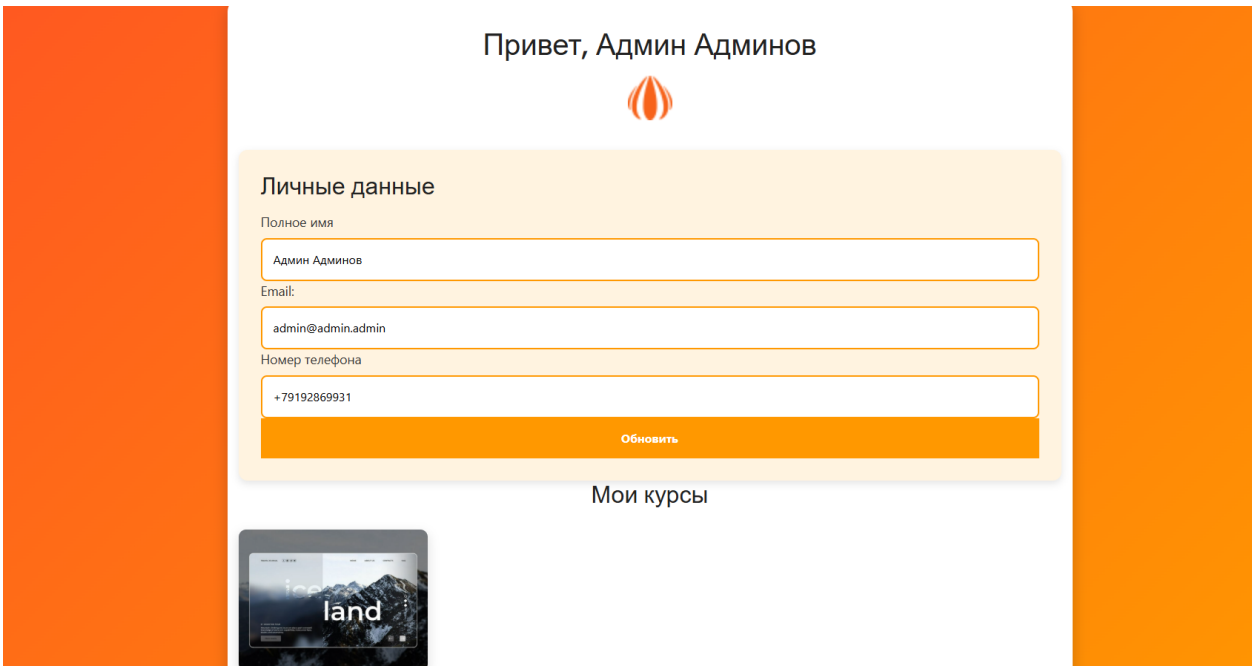


Рисунок 15 – Интерфейс сайта (Личный кабинет)

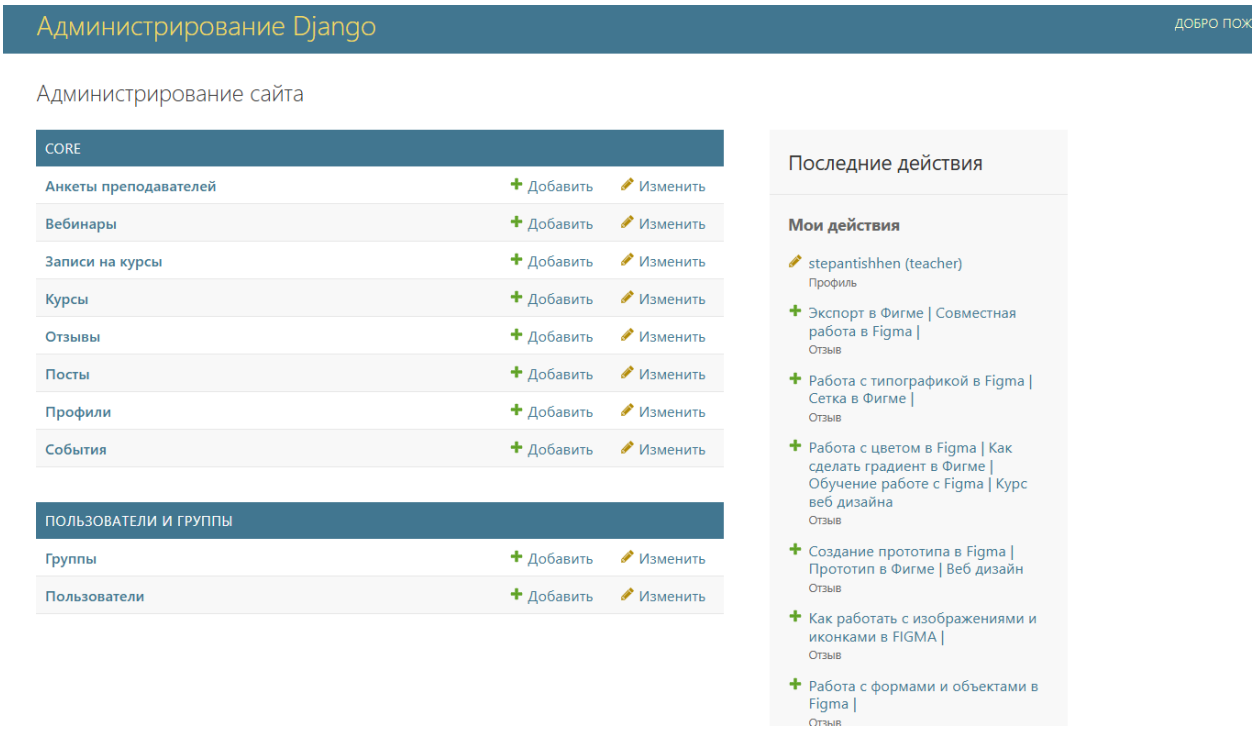


Рисунок 16 – Интерфейс сайта (Административная панель)

## ПРИЛОЖЕНИЕ 2 КОД ВЕБ-САЙТА:

- **Код веб-сайта**: В приложении представлен полный исходный код проекта. К сожалению код всех файлов представить не получится из-за большого объема, поэтому тут будет только самое важное:

**api/admin.py**

```
from django.contrib import admin
from .models import Profile, Course, Enrollment, Event, Review, TeacherApplication, Post, Webinar


@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ('user', 'full_name', 'email', 'phone_number', 'user_type')
    search_fields = ('user__username', 'full_name', 'email', 'phone_number')
    list_filter = ('user_type',)


@admin.register(Course)
class CourseAdmin(admin.ModelAdmin):
    list_display = ('title', 'price')
    search_fields = ('title',)
    list_filter = ('price',)


@admin.register(Enrollment)
class EnrollmentAdmin(admin.ModelAdmin):
    list_display = ('profile', 'course')
    search_fields = ('profile__full_name', 'course__title')
    list_filter = ('course',)


@admin.register(Event)
class EventAdmin(admin.ModelAdmin):
    list_display = ('title', 'date')
    search_fields = ('title',)
    list_filter = ('date',)


@admin.register(Review)
class ReviewAdmin(admin.ModelAdmin):
    list_display = ('profile', 'course', 'rating', 'text')
    search_fields = ('profile__full_name', 'course__title', 'text')
    list_filter = ('rating', 'course')


@admin.register(TeacherApplication)
class TeacherApplicationAdmin(admin.ModelAdmin):
```

```
    list_display = ('name', 'email')
    search_fields = ('name', 'email')


@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'date')
    search_fields = ('title', 'author__full_name')
    list_filter = ('date',)


@admin.register(Webinar)
class WebinarAdmin(admin.ModelAdmin):
    list_display = ('title', 'course')
    search_fields = ('title', 'course__title')
    list_filter = ('course',)
```

**api/models.py**

```
from django.utils.translation import gettext_lazy as _
from django.contrib.auth.models import User
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver


# Модель для профилей пользователей
class Profile(models.Model):
    USER_TYPE_CHOICES = [
        ('student', _('Студент')),
        ('teacher', _('Преподаватель')),
    ]
    full_name = models.CharField(max_length=255, verbose_name=_("Полное имя"))
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='profile',
verbose_name=_("Пользователь"))

    email = models.EmailField(verbose_name=_("Электронная почта"))
    phone_number = models.CharField(max_length=20, verbose_name=_("Номер телефона"))
    user_type = models.CharField(max_length=10, choices=USER_TYPE_CHOICES, default='student',
verbose_name=_("Тип пользователя"))

    class Meta:
        verbose_name = _("Профиль")
        verbose_name_plural = _("Профили")

    def __str__(self):
        return f"{self.user.username} ({self.user_type})"
```

```
# Модель курсов
class Course(models.Model):
    image = models.ImageField(upload_to='courses/', verbose_name=_("Изображение"))
    title = models.CharField(max_length=255, verbose_name=_("Название курса"))
```

```
    description = models.TextField(verbose_name=_("Описание курса"))
    price = models.DecimalField(max_digits=10, decimal_places=2, verbose_name=_("Стоимость курса"))

    class Meta:
        verbose_name = _("Курс")
        verbose_name_plural = _("Курсы")

    def __str__(self):
        return self.title


# Модель записи на курс
class Enrollment(models.Model):
    profile = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='profile',
verbose_name=_("Профиль"))
    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='enrollments',
verbose_name=_("Курс"))

    class Meta:
        verbose_name = _("Запись на курс")
        verbose_name_plural = _("Записи на курсы")

    def __str__(self):
        return f"{self.profile.full_name} - {self.course.title}"


# Модель событий
class Event(models.Model):
    image = models.ImageField(upload_to='events/', verbose_name=_("Изображение"))
    date = models.DateField(verbose_name=_("Дата события"))
    title = models.CharField(max_length=255, verbose_name=_("Название события"))
    description = models.TextField(verbose_name=_("Описание события"))

    class Meta:
        verbose_name = _("Событие")
        verbose_name_plural = _("События")

    def __str__(self):
        return self.title


# Модель отзывов
class Review(models.Model):
    profile = models.ForeignKey(Profile, null=True, blank=True, on_delete=models.SET_NULL,
related_name='reviews',
                                verbose_name=_("Профиль"))
    course = models.ForeignKey(Course, null=True, blank=True, on_delete=models.CASCADE,
related_name='reviews', verbose_name=_("Курс"))
    rating = models.PositiveSmallIntegerField(verbose_name=_("Оценка"))
    text = models.TextField(verbose_name=_("Текст отзыва"))

    class Meta:
        verbose_name = _("Отзыв")
```

```
        verbose_name_plural = _("Отзывы")

    def __str__(self):
        return f"{self.profile or _('Аноним')} - {self.rating}/5"


# Модель анкеты преподавателей
class TeacherApplication(models.Model):
    name = models.CharField(max_length=255, verbose_name=_("Имя"))
    email = models.EmailField(verbose_name=_("Электронная почта"))
    experience = models.TextField(verbose_name=_("Опыт работы"))

    class Meta:
        verbose_name = _("Анкета преподавателя")
        verbose_name_plural = _("Анкеты преподавателей")

    def __str__(self):
        return self.name


# Модель постов
class Post(models.Model):
    image = models.ImageField(upload_to='posts/', verbose_name=_("Изображение"))
    date = models.DateField(auto_now_add=True, verbose_name=_("Дата публикации"))
    author = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='posts',
verbose_name=_("Автор"))
    title = models.CharField(max_length=255, verbose_name=_("Заголовок"))
    text = models.TextField(verbose_name=_("Текст поста"))

    class Meta:
        verbose_name = _("Пост")
        verbose_name_plural = _("Посты")

    def __str__(self):
        return self.title


# Модель вебинаров
class Webinar(models.Model):
    video_link = models.URLField(verbose_name=_("Ссылка на видео"))
    title = models.CharField(max_length=255, verbose_name=_("Название вебинара"))
    description = models.TextField(blank=True, null=True, verbose_name=_("Описание вебинара"))
    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='webinars',
verbose_name=_("Курс"))

    class Meta:
        verbose_name = _("Вебинар")
        verbose_name_plural = _("Вебинары")

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

```
@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)
```

**api/urls.py**

```
from django.urls import path
from core.views import (
    index,
    about,
    courses,
    events,
    teacher_application,
    course,
    register,
    logout_view,
    login_view,
    add_to_course,
    profile
)

urlpatterns = [
    path('', index, name='index'),
    path('about/', about, name='about'),
    path('courses/', courses, name='course_all'),
    # single course
    path('courses/<int:course_id>/', course, name='course'),
    path('events/', events, name='events_all'),
    # single event
    path('teacher_application/', teacher_application, name='teacher_application'),
    path('accounts/login/', login_view, name='login'),
    path('register/', register, name='register'),
    path('accounts/logout/', logout_view, name='logout'),
    path('add_to_course/', add_to_course, name='add_to_course'),
    path('profile/', profile, name='profile'),

]
```

**api/views.py**

```
from django.contrib.auth import logout
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse, HttpResponseForbidden
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login
from django.shortcuts import render, redirect
from django.urls import reverse

from core.models import Course, Post, Event, Review, TeacherApplication, Profile, Enrollment, Webinar
```

```
def index(request):
    top_5_courses = Course.objects.order_by('-pk')[:5]
    top_5_posts = Post.objects.order_by('-date')[:5]
    top_5_events = Event.objects.order_by('-date')[:5]
    top_5_reviews = Review.objects.order_by('-rating')[:5]

    if request.user.is_authenticated:
        try:
            profile = request.user.profile  # Get the user's profile
            enrolled_courses = profile.profile.all().values_list('course_id',
                                                 flat=True)  # get all courses id where user is enrolled
            courses = Course.objects.exclude(pk__in=enrolled_courses)  # exclude courses where user is
enrolled
        except Profile.DoesNotExist:
            # Handle the case where the user doesn't have a profile (unlikely but possible)
            courses = Course.objects.none()  # return empty queryset
    else:
        courses = Course.objects.none()

    context = {
        'title': 'Главная',
        'top_5_courses': top_5_courses,
        'top_5_posts': top_5_posts,
        'top_5_events': top_5_events,
        'top_5_reviews': top_5_reviews,
        'courses': courses,
    }
    return render(request, 'index.html', context=context)


def about(request):
    top_5_reviews = Review.objects.order_by('-rating')[:5]
    context = {
        'title': 'О нас',
        'top_5_reviews': top_5_reviews,
    }
    return render(request, 'about.html', context=context)


def courses(request):
    courses = Course.objects.all().order_by('title')
    top_5_events = Event.objects.order_by('-date')[:5]
    context = {
        'title': 'Курсы',
        'courses': courses,
        'top_5_events': top_5_events,
    }
    return render(request, 'courses.html', context=context)


@login_required
def course(request, course_id):
    course = Course.objects.get(pk=course_id)
```

```
    user_profile = request.user.profile

    try:
        enrollment = Enrollment.objects.get(profile=user_profile, course=course)
    except Enrollment.DoesNotExist:
        # return render(request, 'courses.html', context=context)
        return HttpResponseForbidden("You are not enrolled in this course.")

    webinars = Webinar.objects.filter(course=course)
    context = {
        'title': course.title,
        'course': course,
        'webinars': webinars,
    }
    return render(request, 'course_details.html', context=context)


def events(request):
    events = Event.objects.all().order_by('-date')
    context = {
        'title': 'События',
        'events': events,
    }
    return render(request, 'events.html', context=context)


def teacher_application(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        experience = request.POST.get('experience')

        if name and email and experience:
            application = TeacherApplication(name=name, email=email, experience=experience)
            application.save()
            return redirect(reverse('index'))
        else:
            return HttpResponse("Please fill in all fields.")
    return render(request, 'teacher_application_form.html')


def login_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect(reverse('index'))
    return render(request, 'login.html')


def register(request):
```

```
    if request.method == 'POST':
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')

        # Проверка на существующего пользователя
        if User.objects.filter(username=username).exists():
            return render(request, 'registration.html', {
                'error': 'Пользователь с таким именем уже существует.'
            })

        if User.objects.filter(email=email).exists():
            return render(request, 'registration.html', {
                'error': 'Пользователь с таким email уже зарегистрирован.'
            })

        # Создаем пользователя
        user = User.objects.create_user(username=username, password=password, email=email)
        user.save()

        # Аутентификация и вход
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect(reverse('index'))

        return render(request, 'registration.html', {
            'error': 'Что-то пошло не так. Попробуйте снова.'
        })

    return render(request, 'registration.html')


def logout_view(request):
    logout(request)
    return redirect(reverse('index'))


def add_to_course(request):
    if request.method == 'POST':
        course_id = request.POST.get('course')
        course = Course.objects.get(pk=course_id)
        profile = Profile.objects.get(user=request.user)
        Enrollment.objects.create(course=course, profile=profile)
        return redirect(reverse("index"))
    return render(request, 'add_to_course.html')


@login_required
def profile(request):
    profile = Profile.objects.get(user=request.user)
    enroll_courses = Enrollment.objects.filter(profile=profile)
    context = {
```

```
        'profile': profile,
        'enroll_courses': enroll_courses
    }
    if request.method == 'POST':
        full_name = request.POST.get('full_name')
        email = request.POST.get('email')
        phone_number = request.POST.get('phone_number')
        profile.full_name = full_name
        profile.email = email
        profile.phone_number = phone_number
        profile.save()
    return render(request, 'profile.html', context=context)
```

**edu/urls.py**

```
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('core.urls')),
]
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```