

**КОЛЛЕДЖ КОСМИЧЕСКОГО МАШИНОСТРОЕНИЯ И ТЕХНОЛОГИЙ**

«Допущен к защите»

Председатель цикловой комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Эшанов

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г.

**Дипломный проект**

**РАЗРАБОТКА WEB-КЛИЕНТА ДЛЯ ОБРАЗОВАТЕЛЬНОЙ ПЛАТФОРМЫ**

**Пояснительная записка**

**ДП.09.02.03.22.04ПЗ**

Обучающийся группы П1-18 Герасимов. Д. А.

Руководитель дипломного проекта Шаповалов Е.П.

Результат защиты\_\_\_\_\_\_\_\_\_\_\_\_

**Королёв, 2022**



**КОЛЛЕДЖ КОСМИЧЕСКОГО МАШИНОСТРОЕНИЯ И ТЕХНОЛОГИЙ**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | «УТВЕРЖДАЮ»  Заместитель директора по учебной работе  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Антропова  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. | |

**ЗАДАНИЕ**

**на дипломную работу/проект**

обучающемуся специальности 09.02.03 «Программирование в компьютерных системах»

группы П1-18 **Герасимов Дмитрий Александрович**

Тема дипломного проекта **«Разработка WEB-клиента для образовательной платформы»**

Утверждена приказом от «15» апреля 2022 г. № 01-05/820

Руководитель дипломного проекта **Шаповалов Е.П.**

**Содержание проекта:**

**Введение:** Во введении обосновывается актуальность выполняемой работы и раскрывается краткое содержание разделов.

**Теоретическая часть**: Описание предметной области, постановка задачи.

**Аналитическая часть:** Анализ имеющихся программных решений, маркетинговый анализ

**Проектная часть**: Описание ER-модели; описание алгоритмов решения; обоснование выбора инструментов, сред, языков; описание структуры программного комплекса; описание процедур и функций.

**Организационно-экономическая часть**: Эксплуатационный раздел; раздел техники безопасности; расчёт затрат на разработку программного комплекса.

**Заключение:** Основные выводы и предложения по применению и дальнейшему совершенствованию программного комплекса.

**Список использованных источников информации:** составляется в соответствии с правилами библиографического оформления.

**Приложения**: листинги процедур и функций; руководство пользователя (оператора).

**Презентация**

Срок выполнения дипломного проекта 15 июня 2022 года

Председатель цикловой комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Эшанов

Руководитель дипломной работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Шаповалов подпись

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Герасимов

подпись студента

Дата выдачи задания «15» апреля 2022 года

СОДЕРЖАНИЕ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

4

**ДП.09.02.04.22.04ПЗ**

Разраб.

Герасимов Д.А.

Провер.

Шаповалов Е.П.

Н. Контр.

Бобкова Н.Ю.

Рецензент

Теодорович Н.Н

**РАЗРАБОТКА WEB-КЛИЕНТА ДЛЯ ОБРАЗОВАТЕЛЬНОЙ ОРГАНИЗАЦИИ**

Лит.

Листов

153

Группа П1-18

[СОДЕРЖАНИЕ 4](#_Toc106644557)

[ВВЕДЕНИЕ 6](#_Toc106644558)

[1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 8](#_Toc106644559)

[1.1 Постановка задачи 8](#_Toc106644560)

[1.2 Описание предметной области 8](#_Toc106644561)

[2. АНАЛИТИЧЕСКАЯ ЧАСТЬ 12](#_Toc106644562)

[2.1 Анализ существующих программных решений 12](#_Toc106644563)

[2.2 Маркетинговый анализ 18](#_Toc106644564)

[3. ПРОЕКТНАЯ ЧАСТЬ 20](#_Toc106644567)

[3.1 Описание алгоритмов решения 20](#_Toc106644568)

[3.2 Возможности приложения 21](#_Toc106644569)

[3.3 Обоснование выбора инструментов, сред, языков 23](#_Toc106644570)

[3.4 Описание базы данных 25](#_Toc106644571)

[3.5 Описание структуры программы 34](#_Toc106644572)

[3.6 Описание структуры программного комплекса 45](#_Toc106644575)

[3.7 Описание процедур и функций 46](#_Toc106644576)

[3.8 Методы проектирования и разработки приложения 52](#_Toc106644577)

[4. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ 54](#_Toc106644578)

[4.1 Эксплуатационный раздел 54](#_Toc106644579)

[4.2 Раздел техники безопасности 55](#_Toc106644581)

[4.3 Расчет затрат на разработку программного комплекса 59](#_Toc106644582)

[ЗАКЛЮЧЕНИЕ 63](#_Toc106644583)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 65](#_Toc106644584)

[ПРИЛОЖЕНИЯ 68](#_Toc106644585)

ВВЕДЕНИЕ

Человек всегда задавал вопросы и искал ответ на них. Например, человек ещё в древности стал объяснять то, что он не понимает сущностью и проявлением воли Бога.

Существует целая наука, именуемая «Философией». Смысл её заключается в поиске ответов на главный вопрос «В чем смысл жизни?» и другие вопросы. Люди по сей день пытаются найти на него ответ. Возможно, вы задавали этот вопрос себе, и может быть смогли найти удовлетворяющий ответ для себя.

Но чтобы правильно отвечать на вопросы, нужны знания. Такие знания передаются из поколения в поколения, в форме книг в которых записан уже пройденный человеческий опыт. Поэтому существуют детские сады, школы, институты, которые созданы для того, чтобы обучать детей быть полноценной ячейкой общества.

Время идет, технологии меняются, и обучение в том числе. С появлением цифровых устройств стало возможным носить целую библиотеку книг в одном телефоне. А обучение населения стало перетекать в онлайн-курсы и обучающие видеозаписи. Также появились такие платформы, как Zoom, позволяющие созваниваться школьнику с учителем, студенту с преподавателем.

Такие технологии делают возможность обучения намного удобней и эффективней.

Онлайн-курсы – это определенный формат обучения, позволяет получать знания из любой точки мира с подключением в интернет. Обычно, такие курсы дают знания посредством видеороликов, текстовых лекций, письменных заданий и различных тестов интерактивного типа.

Цель выпускной квалификационной работы – предоставить возможность пользователям проходить и создавать обучающие курсы. Для реализации этой цели необходимо спроектировать WEB-сайт, который даст возможность проходить эти курсы удалённо.

В ходе выполнения дипломного проекта были поставлены следующие задачи:

1. Разработать алгоритмы решения для разработки WEB-сайта и написания серверной части программы.
2. Изучить и проанализировать инструменты для разработки.
3. Разработать архитектуру программы для упрощения дальнейшей разработки.
4. Разработать ER-модель базы данных для хранения данных.
5. Изучить и проанализировать существующие разработки
6. Выбрать язык программирования
7. Спроектировать и разработать программный продукт.

В теоретической части дипломного проекта сделана постановка задачи и определена предметная область.

В аналитической части дипломного проекта проведен анализ существующих программных решений, а также их маркетинговый анализ.

В проектной части дипломного проекта описаны возможности и компоненты обеспечивающие эти возможности: алгоритмы решения; инструменты, среда, языки разработки; структура программного комплекса; процедуры и функции.

В организационно-экономической части дипломного проекта были рассмотрены: правила техники безопасности, показатели затрат на разработку программного комплекса.

В заключении приведены основные результаты дипломного проекта и отмечены перспективные направления для развития приложения.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ
   1. Постановка задачи

Цель работы – разработка WEB-сайта для взаимодействия с программным интерфейсом цифровой образовательной платформы.

Задачи разрабатываемой системы:

* Регистрация пользователей.
* Авторизация пользователей.
* Просмотр существующих курсов.
* Просмотр своих курсов.
* Запись на прохождение курса.
* Прохождение курса.
* Добавление курсов в свои подборки.
* Просмотр чужих подборок.
* Просмотр своих подборок.
  1. Описание предметной области

Веб-сайт

Чтобы объяснить, что такое веб-сайт нужно объяснить, что такое веб-страница и браузер.

**Браузер** – это программное обеспечение для просмотра веб-страниц. Самые популярные браузера Яндекс Браузер [6], Google Chrome [12], Firefox [10], Safari [26].

**Веб-страница** — это документ, отображаемый браузером. Такой документ написан языком HTML [13], и может содержать следующую информацию:

* *Стилевая информация* – отвечает за внешний вид страницы.
* *Скрипты* – отвечают за динамичность страницы.
* *Медиа* – изображения, видео и аудио.

**Веб-сайт** или **сайт** – это коллекция веб-страниц, взаимодействующие между собой каким-либо способом.

**Веб-сервер** – это компьютер, хранящий веб-страницы и предоставляющий необходимое программное обеспечение для функционирования веб-сайта.

REST API

**REST API** это стандарт передачи данных. Данные передаются от сервера к клиентскому приложению.

REST API связывает клиент и сервер посредством протокола HTTP или же его защищенной версии HTTPS. По средством этих протоколов передаётся наибольшая часть информации в интернете.

Система REST API делится на 4 метода:

* + 1. **GET** – метод получения данных.
    2. **POST** – метод создания новых записей.
    3. **PUT** – метод обновления записей.
* **DELETE** – метод удаления записей.

Цифровая образовательная платформа

Цифровая образовательная платформа — информационное пространство, объединяющее участников процесса обучения, которое дает возможность для удаленного образования, обеспечивает доступ к методическим материалам и информации, а также позволяет осуществлять тестирование для контроля уровня знаний обучающихся.

Популярность электронных образовательных платформ с интерактивными возможностями обучения растет, особенно в условиях, продиктованных современной реальностью. Переход на удаленное обучение требует внедрения в процесс современных методов обучения с использованием не только компьютеров, но и другой современной техники.

Можно выделить три основные задачи, которые помогают решить цифровые онлайн платформы для обучения:

* Дистанционное обучение и контроль знаний для учебного заведения.
* Автоматизация процесса повышения квалификации и обучения сотрудников компании.
* Организация обучающих курсов в разных сферах бизнеса.

Наиболее популярные платформы для обучения в основном нацелены на повышение квалификации или получение новой профессии. На данный момент стали появляться платформы, которые можно использовать и для дистанционного обучения детей и подростков.

Основные направления курсов обучения, касаются тем программирования, маркетинга, дизайн и тестирования ПО, также стремительными темпами стали развиваться всевозможные онлайн-курсы, касающиеся иностранных языков.

Все платформы объединяют задачи, которые они позволяют решать, разница лишь в интерфейсе самой платформы, тарифных планах, информационном наполнении и выборе методик преподавания.

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ
   1. Анализ существующих программных решений

На данный момент существует множество уже разработанных приложений по этой тематике. Практически все из них предоставляют похожий функционал:

* Реализация прохождения курсов.
* Разделение прав доступа для использования приложения (Пользователь, Администратор).

Примеры сайтов:

1. WEB-сайт «Лекториум [2] ».

Это платформа с курсами, где курсы имеют право создать:

* Университеты.
* Авторская команда компании «Лекториум».

Пример страницы сайта курса (рисунок 1).

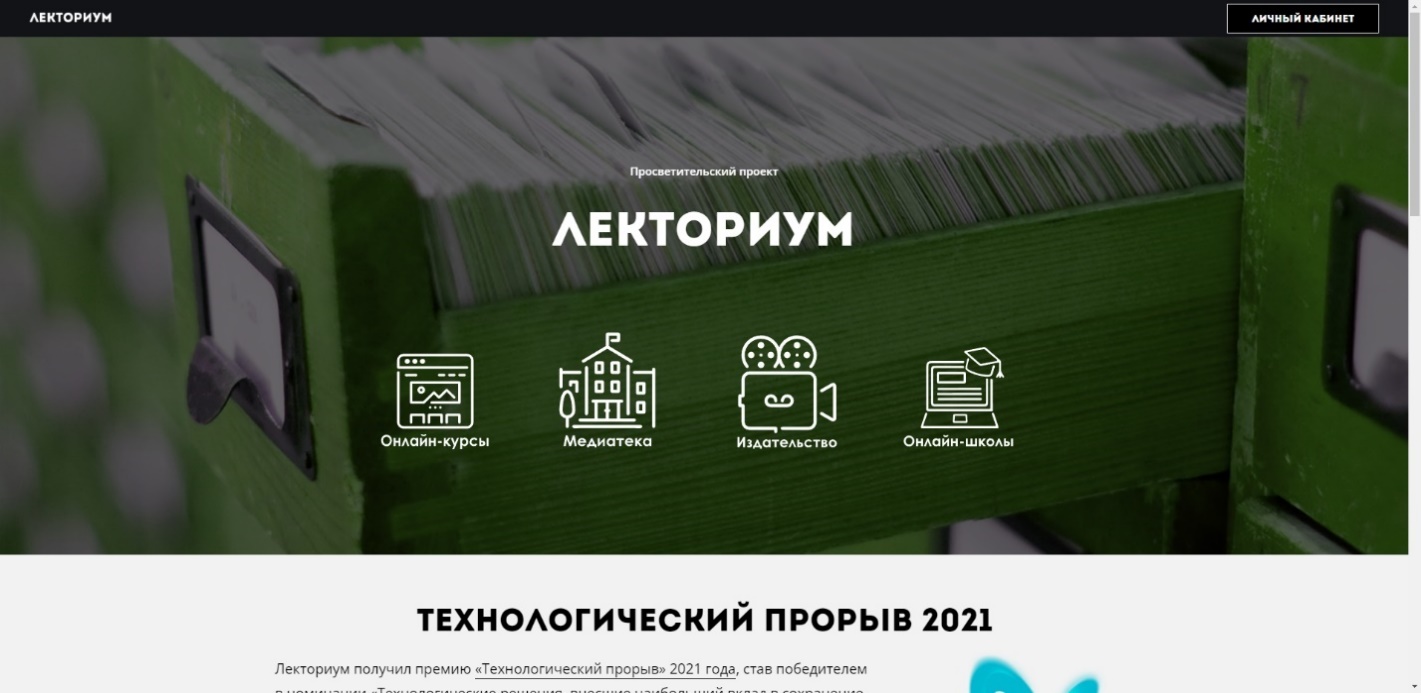


Рисунок 1. Лекториум. Главная страница

1. WEB-сайт «Stepik [29] » .

Отличается от других платформ тем, что есть возможность создания своего курса (рисунок 2).

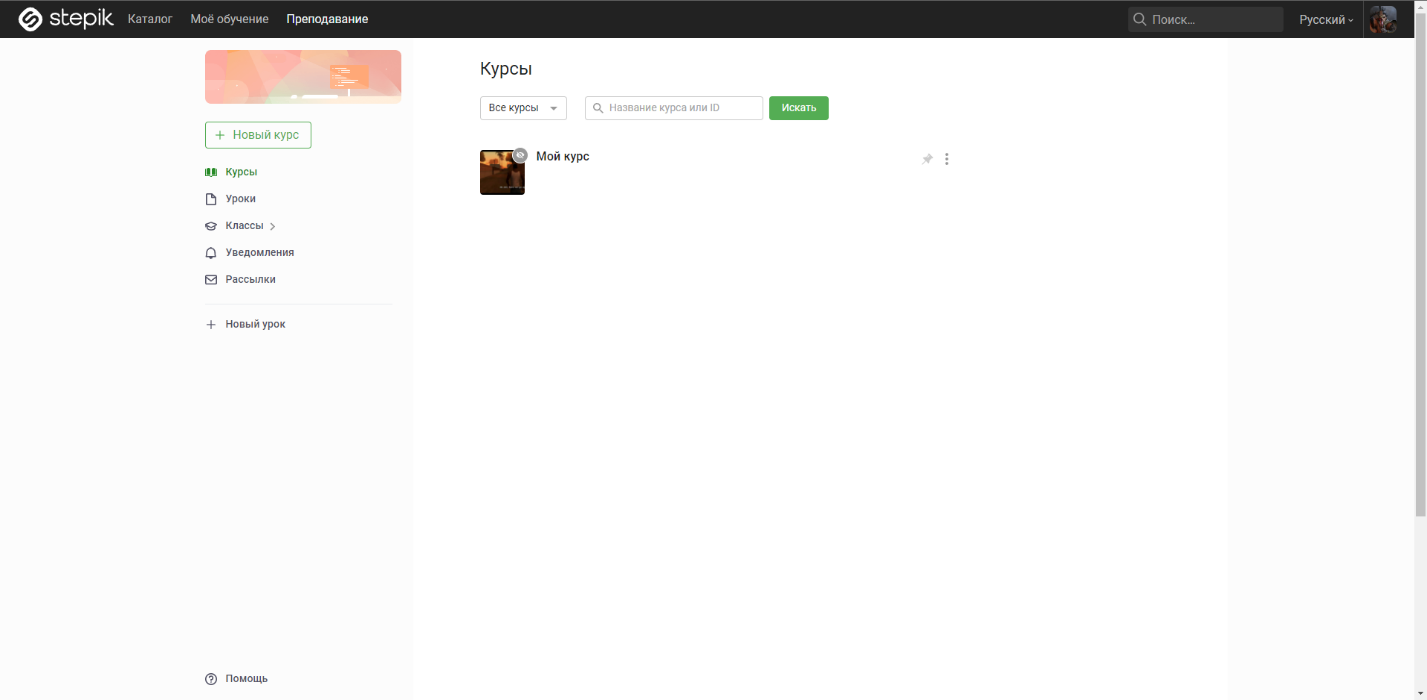


Рисунок 2. Stepik. Создание курса.

Также можно увидеть активность вашего изучения курсов (рисунок 3).

Рисунок 3. Stepik. Активность вашего прохождения.

1. WEB-сайт «ЯЮниор [5] ».

В отличие от других платформ с курсами, имеет лишь один курс (рисунок 4).

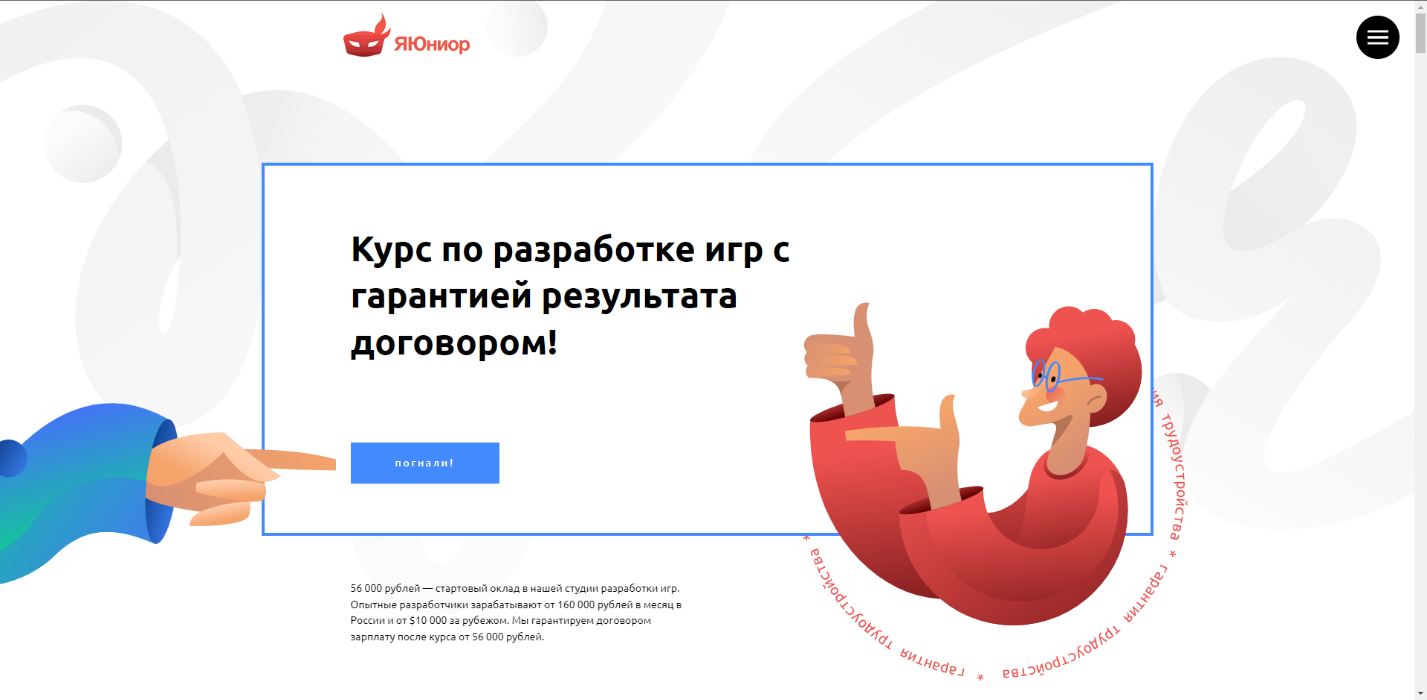


Рисунок 4. ЯЮниор. Главная страница

1. WEB-сайт «Яндекс Практикум [7] ».

Сайт с курсами, где курсы создаёт сама компания. После прохождения некоторых курсов предоставляется возможность трудоустройства (рисунок 5).

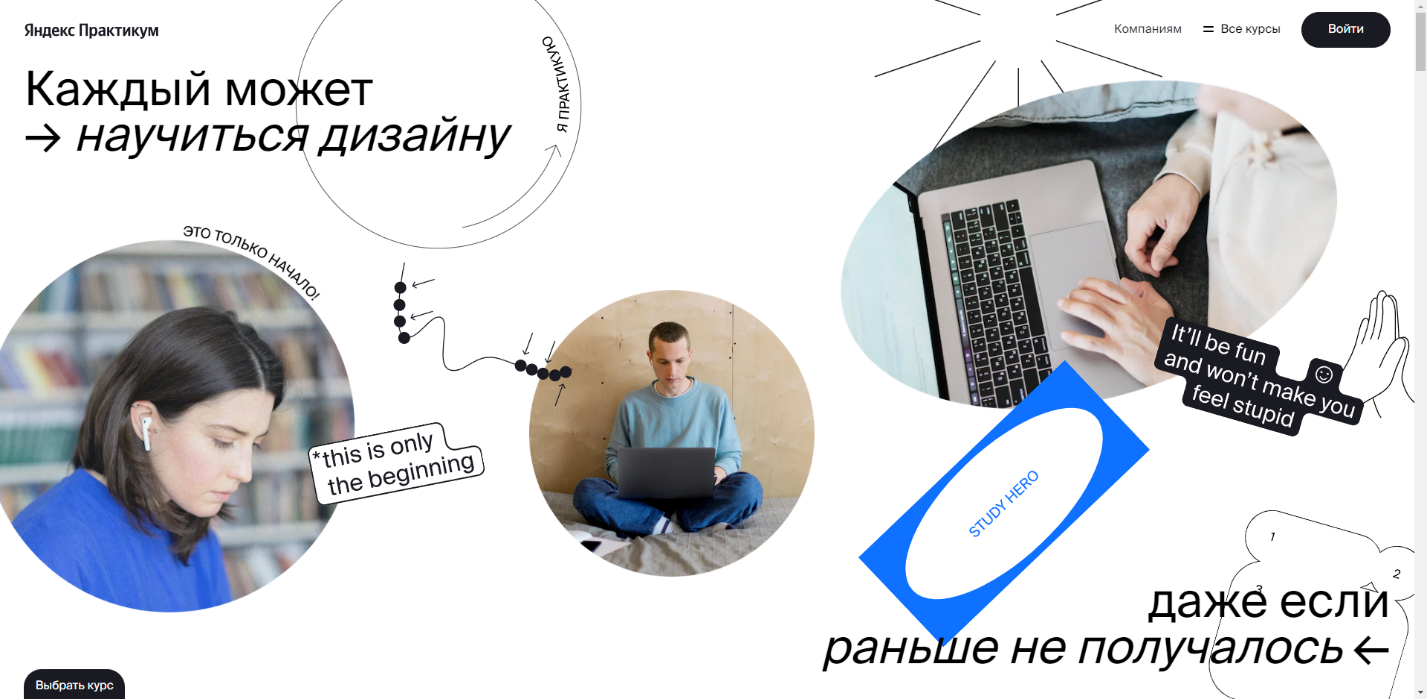


Рисунок 5. Яндекс Практикум. Главная страница

1. WEB-сайт «Skillbox [27] ».

Эта платформа предоставляет не только курсы, но и профессии, где тебя обучают всему необходимому для получения профессии. Так же, как и другие платформы, описанные выше, предоставляет возможность трудоустройства после прохождения курсов и профессий (рисунок 6).

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 6. Skillbox. Страница каталога

На платформе «Skillbox» присутствуют вебинары (рисунок 7)

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 7. Skillbox. Страница с вебинарами

Также, данная платформа предоставляет возможность изучения статей (рисунок 8).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8. Skillbox. Страница со статьями

Можно провести анализ этих разработок и оценить целесообразность разработки собственного аналога «FUJI». (оценки по 10-бальной шкале)

Таблица

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | UI Дизайн | UX Дизайн | Скорость работы сайта | Разнообразие задач | Наличие подборок |
| Лекториум | 7 | 8 | 6 | 1 | 0 |
| Stepik | 5 | 10 | 8 | 10 | 0 |
| ЯЮниор | 8 | 8 | 8 | 1 | 0 |
| Яндекс Практикум | 10 | 10 | 10 | 10 | 0 |
| Skillbox | 10 | 10 | 8 | 1 | 0 |
| FUJI | 10 | 10 | 8 | 5 | 10 |

Анализ существующих программных решений

Из данного анализа можно вынести, что в разработке присутствует новая система подборок, которая позволяет систематизировать и обобщить курсы по общим темам, принципам и делиться ими с другими пользователями. Также они упрощают поиск и навигацию по курсам.

* 1. Маркетинговый анализ

На сегодняшний день онлайн-образование пользуется всё большим спросом и в такое образование инвестируется все больше и больше денежных средств. Такой формат обучения становится на столько популярным, что его выделили в отдельную индустрию **EdTech**, или же **Education technology**.

Перспективы рынка EdTech, можно оценить по реакции инвесторов. Так, в 2020 году глобальный объём венчурных инвестиций в EdTech составил 16,1 млрд долларов, что в 32 раза больше, чем в 2010 году [4]. Безусловно Пандемийные ограничения повлияли на рост инвестиций в данную индустрию.

На данный момент в мировой структуре EdTech Россия занимает около 1%. Тем не менее темпы роста национального рынка высоки (20-25% в год). В 2019 году емкость российского рынка EdTech можно оценить в 45-50 млрд рублей.

Наиболее популярный сегмент в российской индустрии EdTech касается **дополнительного образования** или **ДО** же **ДПО**. Объем этого рынка учитывая лишь взрослый контингент, к 2020 году составил около 22-25 млрд рублей — это целых 50% всего российского рынка EdTech.

Основные сегменты ДО и ДПО для взрослых это:

* Обучение digital-профессиям. Основные лидеры: онлайн-университет Skillbox [27] (оценочная доля рынка ≈ 6,5%), "Нетология-групп" [18] (по 5,3% доли рынка в сегменте).
* Прикладным профессиям. Основные лидеры: GetCourse [11] (3,5-4%) и "Инфоурок" [1] (≈3%).
* Изучение иностранных языков. Основные лидеры: SkyEng [28] (≈19-21%), Lingualeo [15] (4,3-5%) и PuzzleEnglish [22] (≈3%).
* Разработки методов и средств корпоративного обучения. Основные лидеры: “Мираполис” [3] с долей ≈7,2%), также eQueo [31] и iSpring [32] с долями по 5,3%.
  + 1. Глобальные тренды

На данный момент рынок EdTech один из самых динамично развивающихся рынков. Образование уже невозможно представить вне развития технологий. Так, например, чат-боты берут на себя часть рутинной работы преподавателей, значительно упрощая общение со студентами. А искусственный интеллект, VR и другие новшества помогают лучшему усвоению материала.

Мы живем в мире, который меняется с невероятной скоростью. Теперь возможность обучить человека навыкам на всю жизнь, является невозможной задачей, поэтому современному специалисту приходится постоянно повышать свою квалификацию. Такому феномену даже дали определение – непрерывное обучение (Lifelong learning).

* + 1. Использование подборок для обучения персонала

Наличие подборок на платформе позволит руководителям составлять подборки для обучения персонала, что также повысит спрос платформы на рынке и приведет крупных клиентов. Также данная функция привлечет и обычных пользователей за счет своей уникальности и удобства.

1. ПРОЕКТНАЯ ЧАСТЬ
   1. Описание алгоритмов решения

Для решения поставленной задачи были составлен алгоритм ее решения, согласно которому проводилась разработка проекта. Алгоритм делится на следующий задачи:

* + 1. Постановка задачи.

Формулирование условий работы и эксплуатации программы с описанием входных и выходных данных.

* + 1. Составление алгоритма работы программы.

Алгоритм составляется для разбития приложения на более простые шаги, которые помогут определить процессы перехода входных данных в выходные.

* + 1. Выбор Языка программирования, среды и инструментов разработки.

Анализ актуальных средств разработки программного обеспечения и подбора подходящего набора инструментов для решения поставленной задачи

* + 1. Составление документации.

Подготовка документов относящихся к разработке ПО для информирования пользователей и заказчика о функциональных возможностях ПО.

* + 1. Разработка клиентской части приложения.

Разработка интерфейса клиентской части включающая в себя возможность управления и отображения данных, приходящих сервером.

* + 1. Разработка Базы Данных.

Разработка базы данных для хранения всей нужной информации.

* + 1. Разработка серверной части приложения.

Разработка классов сериализаторов для соединения с Базой Данных.

* + 1. Тестирование и отладка.

Поиск и устранение проблем и ошибок в работе приложения.

* + 1. Эксплуатация и сопровождение приложения.

Введение программы в эксплуатацию с дальнейшей поддержкой в течение периода эксплуатации.

* 1. Возможности приложения

WEB-сайт должен содержать:

* Экран Регистрации.
* Экран Входа.
* Экран Каталога. С выводом популярных подборок.
* Экран Моё обучение. Экран со статистикой обучения, а также пройденными и текущими курсами текущего пользователя.
* Экран Архив. Экран с подборками, который пользователь добавил себе; курсами, которые текущий пользователь проходит или прошёл; подписчиками и людьми, на которых подписан текущий пользователь.
* Страница Поиска.
* Страница Пользователя. Содержит курсы, которые проходит и которые завершил пользователь, а также содержит общую статистику по всем завершенным курсам пользователя. Также подборки и курсы, которые проходит пользователь. А также возможность подписаться на пользователя.
* Личная Страница. Содержит курсы, которые проходит и которые завершил пользователь, а также содержит общую статистику по всем завершенным курсам пользователя. Также подборки и курсы, которые проходит пользователь. А также Настройки пользователя.
* Страница создания курса, где будут продемонстрированы все опубликованные и не выложенные курсы текущим пользователем. Также есть возможность создать курс.
* Детальный экран подборки. С возможностью оценивания подборки и изменения данных в нем, если эта подборка принадлежит вам.
* Страница курса. Где есть общая информация о курсе.
* Страница редактирования страницы курса.
* Страница создания темы курса.
* Страница создания урока курса.
* Страница создания шага курса.
* Страница с темами курса.
* Страница с уроками курса.
* Страница с шагами курса.
  1. Обоснование выбора инструментов, сред, языков

Задача состоит в проектировании и создании образовательного портала для прохождения курсов.

Необходимо разработать Web-сайт для прохождения курсов. Курсы должны состоять из шагов, которые будут проходить пользователи. Пользователь может просматривать прогресс в курсах, которые он проходит или проходил.

При выборе инструментов обращалось внимание на определенные критерии:

* Наличие необходимых библиотек.
* Возможность подключений сторонних систем.
* Наличие документации на русском языке.
* Возможность работы на различных платформах.
* Актуальность.
* Доступность.
* Опыт разработки.

После некоторых поисков были выбраны следующие инструменты:

1. Выбор языка стилизации.

Для разработки WEB-сайта используются язык гипертекстовой разметки, а также язык описания внешнего документа. Описания внешнего вида сайта в основном пишется на языке таблиц стилей CSS [8], или же на предпроцессорах SASS [25]. Был проведен их анализ и выбран лучший вариант.

Таблица 2

Анализ языков стилизации

|  |  |  |  |
| --- | --- | --- | --- |
|  | Опыт разработки | Скорость разработки | Лаконичность |
| CSS | 10 | 8 | 5 |
| SASS | 5 | 10 | 10 |

После анализа критериев был выбран препроцессор SASS. Т.к. набрал наибольшее количество баллов.

1. Выбор фреймворка.

Для разработки динамического WEB-сайта рассматривались фреймворки ReactJS [24], VueJS [30], NuxtJS [19] и Django [9]. Был проведен их анализ и выбран лучший вариант.

Таблица 3

Анализ фреймворков

|  |  |  |  |
| --- | --- | --- | --- |
|  | Опыт разработки | Скорость разработки | SEO |
| ReactJS | 1 | 5 | 4 |
| VueJS | 0 | 6 | 5 |
| NuxtJS | 7 | 8 | 8 |
| Django | 8 | 7 | 7 |

После анализа критериев был выбран фреймворк NuxtJS. Т.к. он набрал наибольшее количество баллов.

1. Выбор СУБД.

WEB-сайт не сможет нормально функционировать без СУБД, т.к. курсы и информацию о пользователях надо где-то хранить. Наибольшей популярностью пользуются PostgreSQL [21], MsSQL [16], MySQL [17]. Был проведен их анализ и выбран лучший вариант.

Таблица 4

Анализ СУБД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Опыт разработки | Скорость работы | Надежность | Наличие хостинга |
| PostgreSQL | 7 | 8 | 8 | 0 |
| MsSQL | 5 | 8 | 7 | 0 |
| MySQL | 7 | 7 | 7 | 10 |

После анализа критериев была выбрана СУБД MySQL. Т.к. он набрал наибольшее количество баллов.

1. Выбор серверного языка.

Для связи БД и WEB-сайта, требуется реализация REST API. Зачастую для его реализации используют серверный языки Java [14], PHP [20], Python [23]. Был проведен их анализ и выбран лучший вариант.

Таблица 5

Анализ серверного языка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Опыт разработки | Скорость работы | Надежность | Наличие хостинга |
| Java | 7 | 7 | 10 | 0 |
| PHP | 1 | 6 | 5 | 10 |
| Python | 10 | 10 | 10 | 10 |

После анализа критериев был выбран серверный язык Python. Т.к. выигрывает по всем критериям.

* 1. Описание базы данных

Для данной программы была спроектирована и разработана база данных для возможность хранить и отсылать данные нужные пользователю.

База данных содержит информацию о курсах, подборках с курсами, пройдённых шагах пользователя и информация о пользователе.

ER-диаграмма представлена на рисунке (рисунок 9):

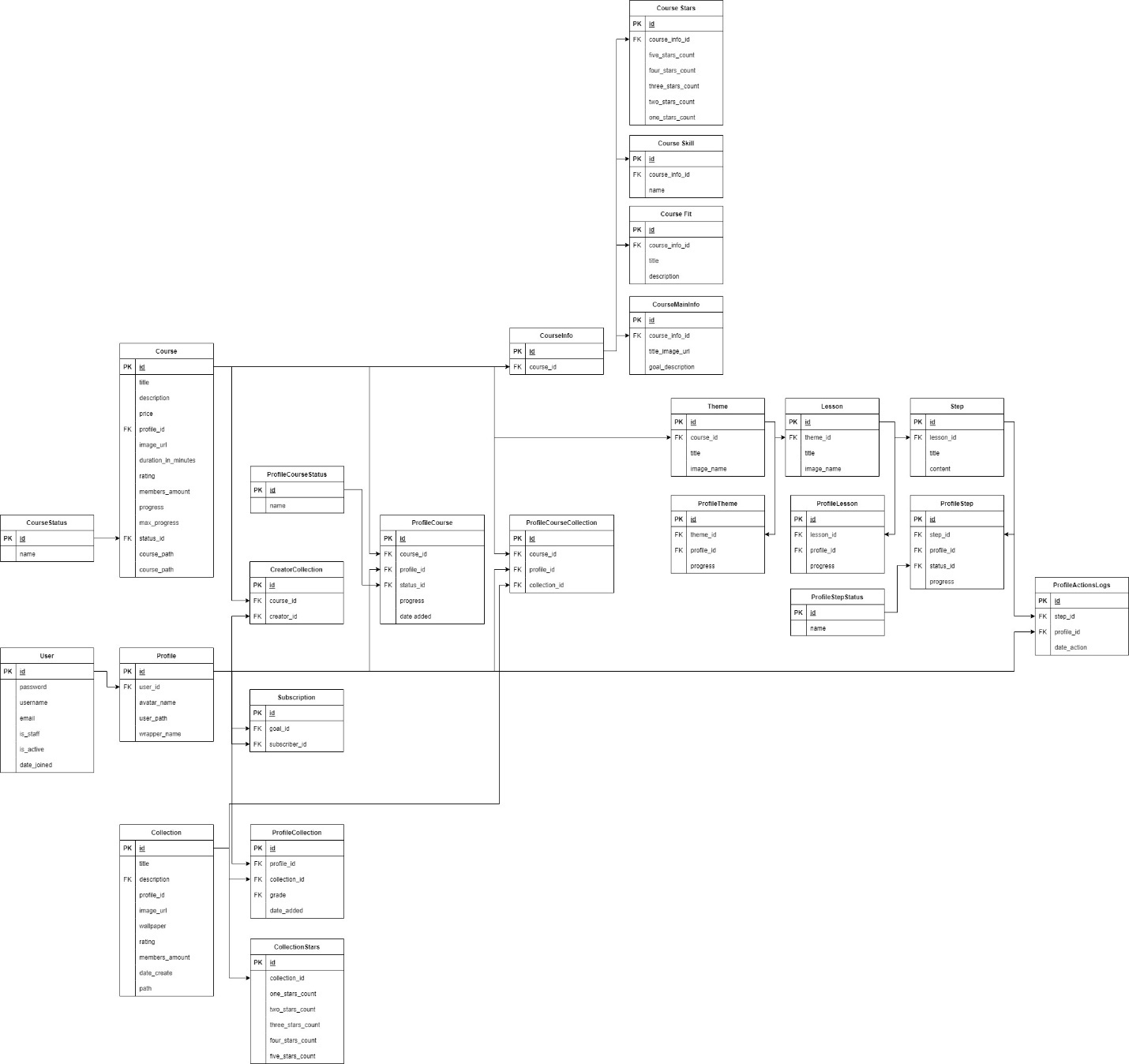


Рисунок 9.ER-диаграмма БД

Данный раздел содержит таблицы из ER-диаграммы Базы Данных и содержит следующие таблицы:

1. Фрагмент таблиц, связанных с пользователем:

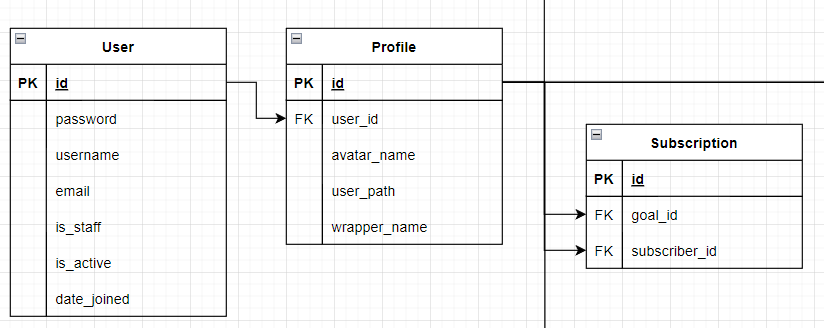


Рисунок 10. Фрагмент таблиц связанных с пользователем.

Данный фрагмент таблиц (рисунок 10) содержит данные пользователя, и состоит из следующих таблиц:

* **Таблица «User»**. Содержит информацию пользователя.
* **Таблица «Profile»**. Содержит расширенную информацию о пользователе.
* **Таблица «Subscription»**. Содержит информацию о подписавшихся.

1. Фрагмент таблиц, связанный с подборками:

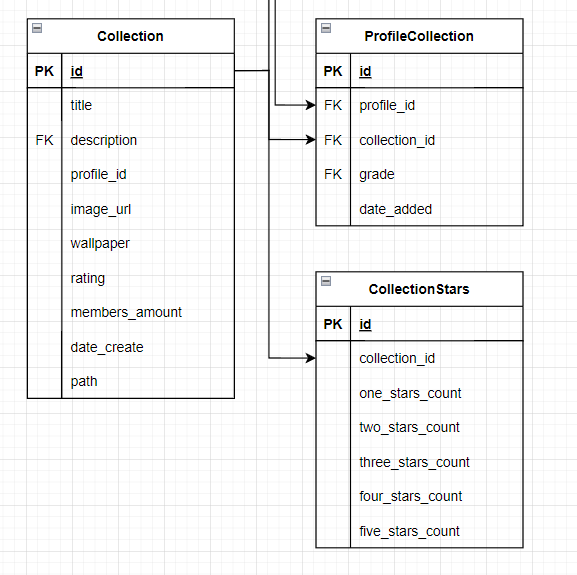


Рисунок 11. Фрагмент таблиц, связанный с подборками

Данный фрагмент таблиц (рисунок 11) содержит данные подборок, и состоит из следующих таблиц:

* **Таблица «Collection»**. Содержит информацию о подборках.
* **Таблица «ProfileCollection»**. Содержит информацию об отношении пользователя к подборке.
* **Таблица «CollectionStars»**. Содержит оценки пользователей к подборкам.

1. Фрагмент таблиц, связанный с курсами:

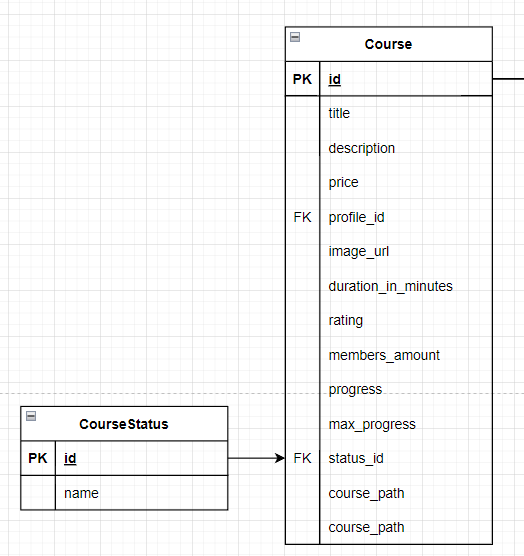


Рисунок 12. Фрагмент таблиц, связанный с курсами

Данный фрагмент таблиц (рисунок 12) содержит данные курса, и состоит из следующих таблиц:

* **Таблица «CourseStatus»**. Содержит информацию о статусе курса («В разработке» или «В публикации»).
* **Таблица «Course»**. Содержит информацию о курсе.

1. Фрагмент таблиц, связанный со страницей курса:

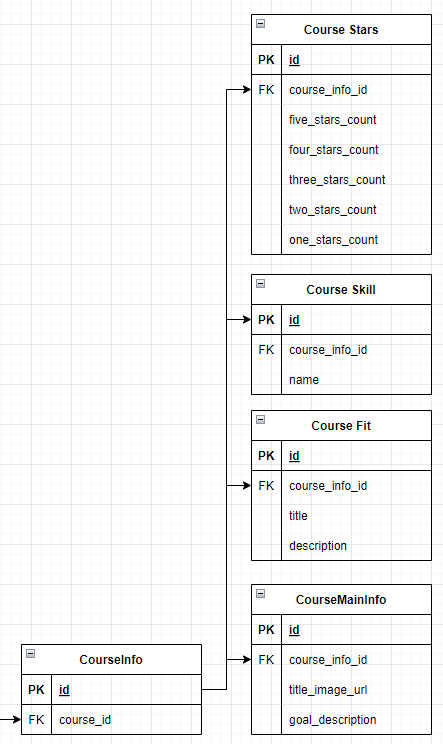


Рисунок 13. Фрагмент таблиц, связанный со страницей курса

Данный фрагмент таблиц (рисунок 13) содержит данные о странице курса, и состоит из следующих таблиц:

* **Таблица «CourseInfo»**. Содержит код страницы.
* **Таблица «CourseMainInfo»**. Содержит главную информацию страницы курса.
* **Таблица «CourseFit»**. Содержит информация о целевой аудитории курса.
* **Таблица «CourseSkill»**. Содержит информацию о навыках, которые приобретёт пользователь после прохождения курса.
* **Таблица «CourseStars»**. Содержит оценки пользователей к курсу.

1. Фрагмент таблиц, связанный с контентом курса:

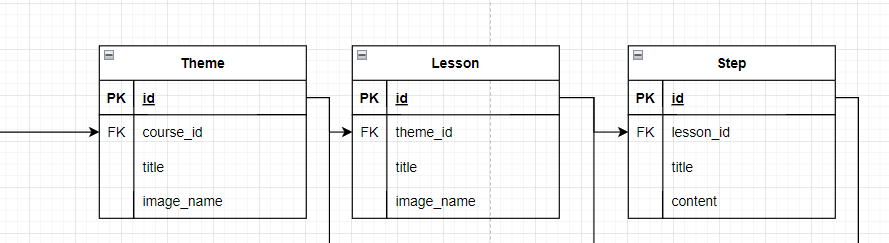


Рисунок 14. Фрагмент таблиц, связанный с контентом курса

Данный фрагмент таблиц (рисунок 14) содержит контент курса, и состоит из следующих таблиц:

* **Таблица «Theme»**. Содержит темы курса.
* **Таблица «Lesson»**. Содержит уроки курса.
* **Таблица «Step»**. Содержит шаги курса.

1. Фрагмент таблиц, со связью курс к пользователю:

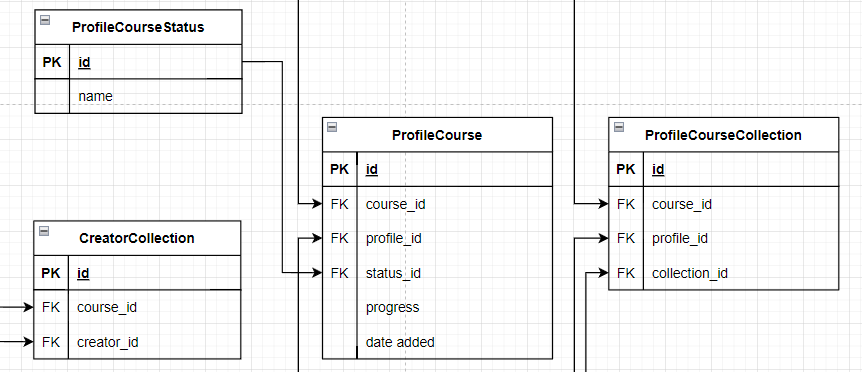


Рисунок 15. Фрагмент таблиц, со связью курс к пользователю

Данный фрагмент таблиц (рисунок 15) содержит информацию со связью пользователя к курсу, и состоит из следующих таблиц:

* **Таблица «ProfileCourseStatus»**. Содержит статус прохождения курса («Изучается» или «Завершен»).
* **Таблица «ProfileCourse»**. Содержит главную информацию о прохождении курса.
* **Таблица «CreatorCourse»**. Содержит связь создателя курса с самим курсом.
* **Таблица «ProfileCourseCollection»**. Демонстрирует связь пользователя в случае добавлении курса в подборку.

1. Фрагмент таблиц, олицетворяющий прохождение курса:

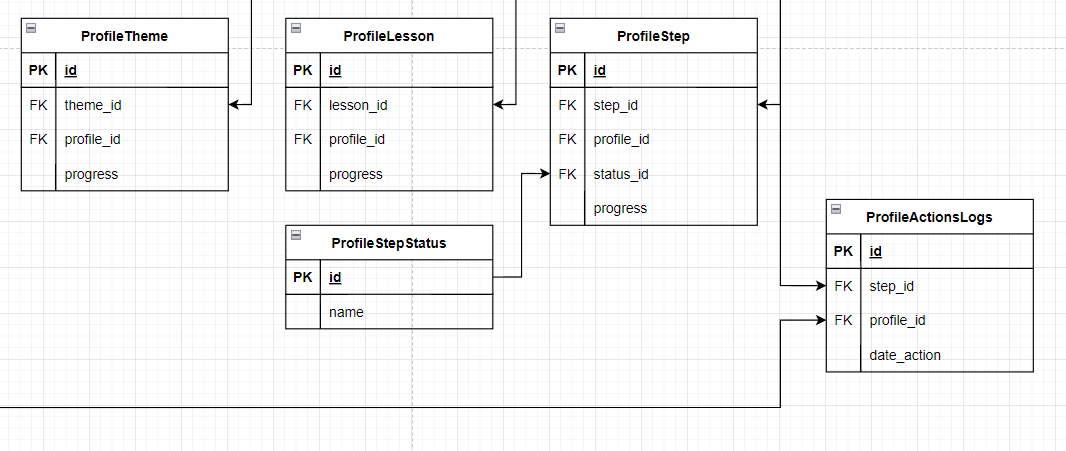


Рисунок 16. Фрагмент таблиц, олицитворяющий прохождения курса

Данный фрагмент таблиц (рисунок 16) показывает всю информацию связанной с прохождением курса, и состоит из следующих таблиц:

* **Таблица «ProfileTheme»**. Содержит информацию о прохождении темы пользователем.
* **Таблица «ProfileLesson»**. Содержит информацию о прохождении урока пользователем.
* **Таблица «****ProfileStepStatus»**. Содержит статус прохождения шага пользователем.
* **Таблица «ProfileStep»**. Содержит информацию прохождения шага пользователем.
* **Таблица «ProfileActionsLogs»**. Является отчетной таблицей по прохождению курса пользователем.
  1. Описание структуры программы
     1. Front-end модуль

Ниже представлена схема WEB-сайта (рисунок 17):

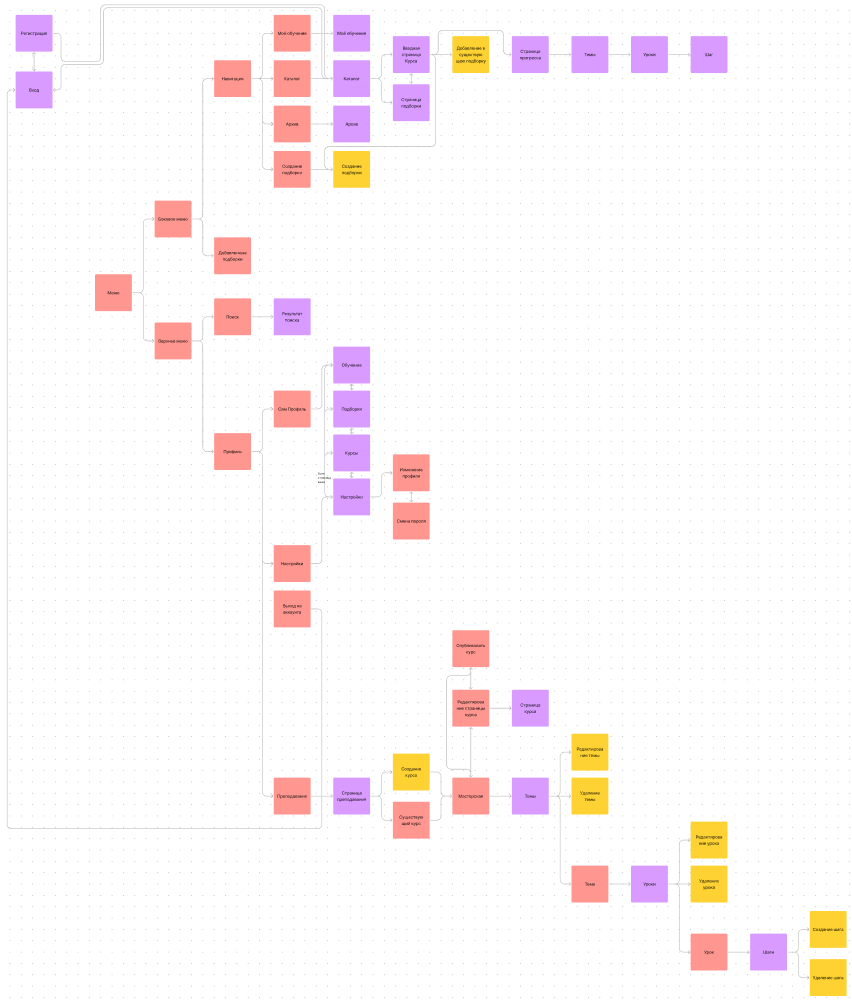


Рисунок 17. Схема WEB-сайта

Данная схема имеет блоки 3-х типов:

* + 1. ***Веб-страница*** – фиолетовый блок
    2. ***Компонент***– красный блок
    3. ***Промежуточное окно*** – желтый блок

Ниже представлены данные типы блоков (рисунок 18)



Рисунок 18. Типы блоков в схеме

Ниже будут описаны данные блоки по фрагментам:

1. Фрагмент схемы, связанный с пользователем:

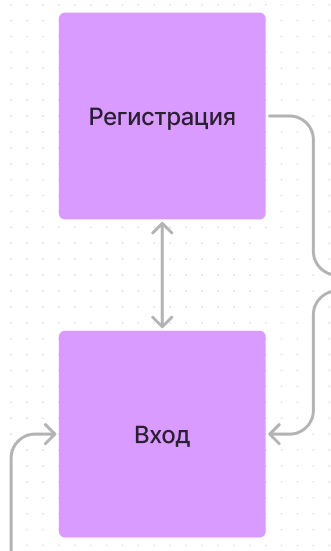


Рисунок 19. Фрагмент схемы, связанный с пользователем

Данный фрагмент схемы (рисунок 19) отвечает за регистрацию и вход пользователя, и состоит из следующих блоков:

* **Регистрация**. Регистрация пользователя.
* **Вход**. Аутентификация пользователя.

1. Фрагмент схемы, демонстрирующий меню web-сайта:

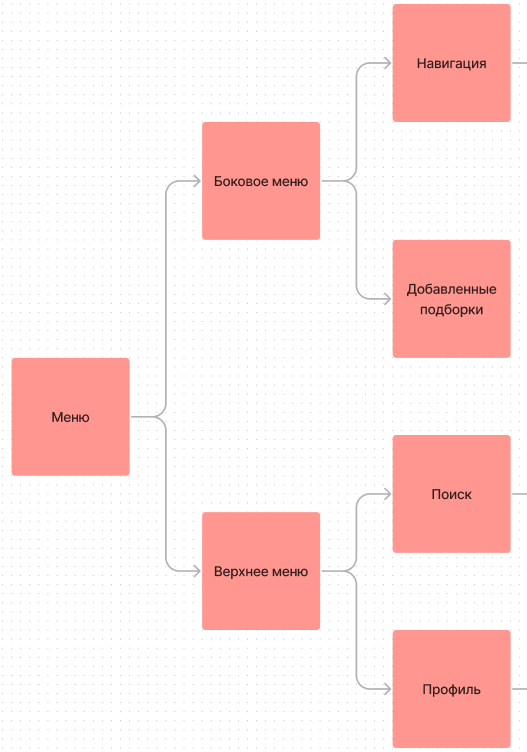


Рисунок 20. Фрагмент схемы, демонстрирующий меню web-сайта

Данный фрагмент схемы (рисунок 20) демонстрирует меню, которое состоит из Бокового меню и Верхнего меню.

* ***Боковое меню*** – состоит из компонента навигации и добавления подборки.
* ***Верхнее меню*** – состоит из компонента поиска из профиля.

1. Фрагмент схемы, демонстрирующий основную навигацию:

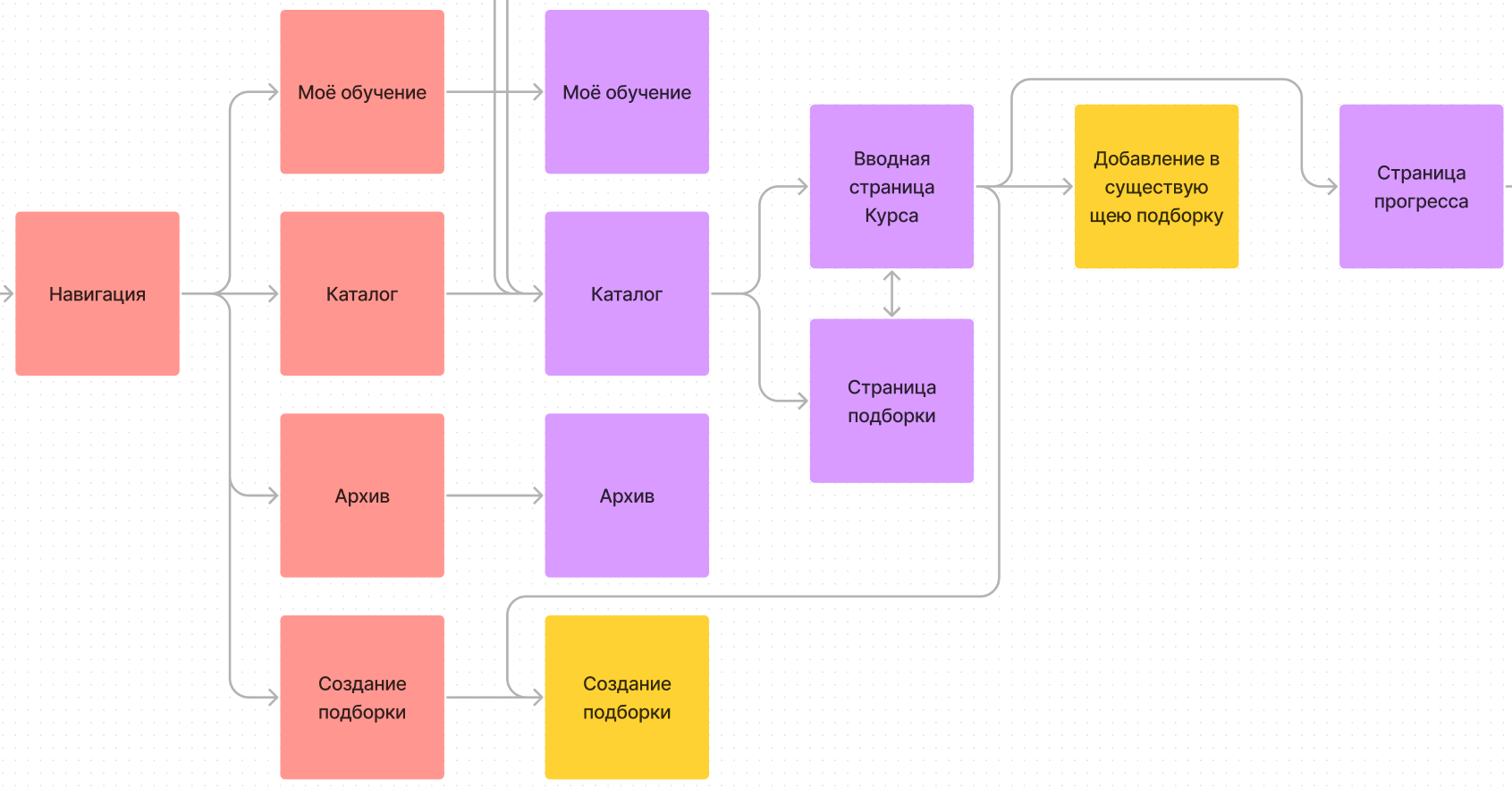


Рисунок 21. Фрагмент схемы, демонстрирующий основную навигацию

Данный фрагмент схемы (рисунок 21) демонстрирует навигацию, которая состоит из фрагментов:

* ***Моё обучение*** – это кнопка, при нажатии на которую переходишь на станицу «Моё обучение».
* ***Каталог*** – это кнопка, при нажатии на которую переходишь на станицу «Каталог».
* ***Архив*** – это кнопка, при нажатии на которую переходишь на станицу «Архив».
* ***Создание подборки*** – это кнопка, при нажатии на которую создаётся подборка.

На странице «Каталог», можно открыть страницу подборки или перейти на страницу курса. На странице курса, можно добавить себе курс или продолжить изучение.

1. Фрагмент схемы, демонстрирующий прохождение курса:

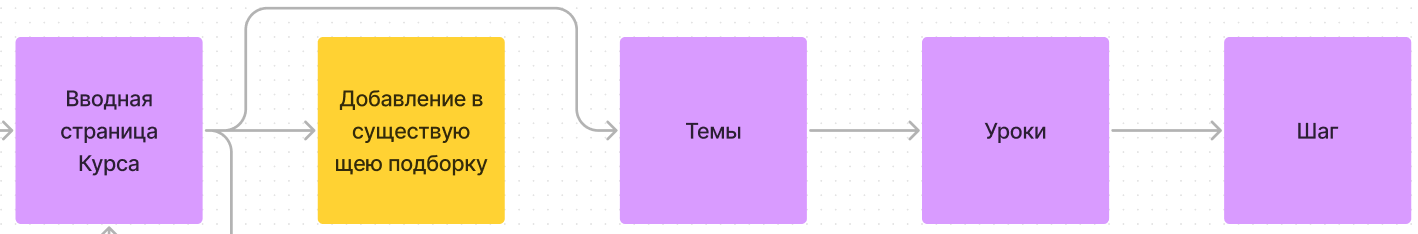


Рисунок 22. Фрагмент схемы, демонстрирующий прохождение курса

Данный фрагмент схемы (рисунок 22) демонстрирует возможность прохождения курса, который состоит из следующих блоков:

* ***Темы*** – WEB-страница, отображающая темы курса.
* ***Уроки*** – WEB-страница, отображающая уроки курса.
* ***Шаг*** – WEB-страница, отображающая контент текущего курса.

1. Фрагмент схемы, демонстрирующий поиск:

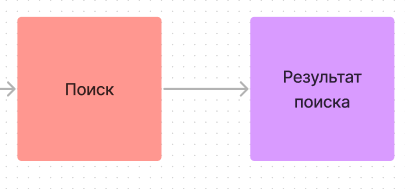


Рисунок 23. Фрагмент схемы, демонстрирующий поиск

Данный фрагмент схемы (рисунок 23) демонстрирует компонент поиск, где после ввода данных отображается страница с результатом.

1. Фрагмент схемы, демонстрирующий компонент профиль:

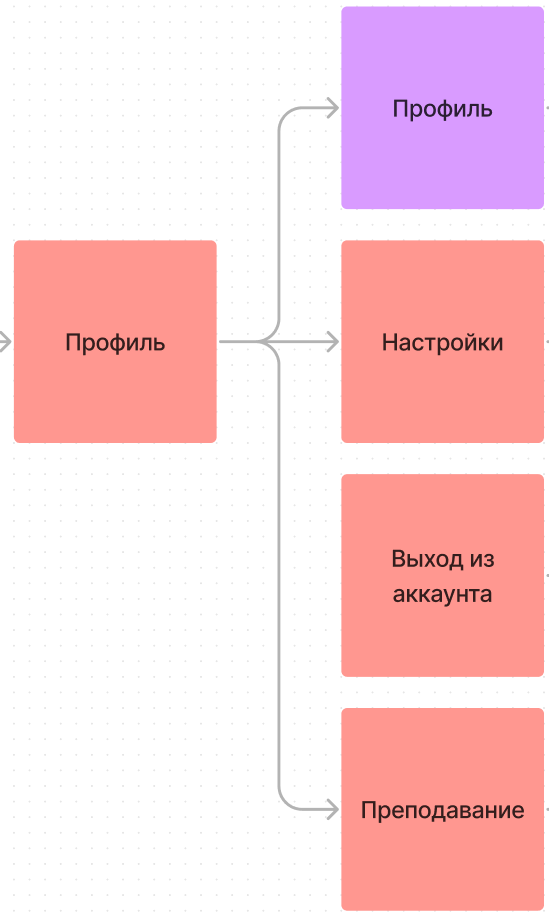


Рисунок 24. Фрагмент схемы, демонстрирующий компонент профиль

Данный фрагмент схемы (рисунок 24) демонстрирует компонент профиль, который делится на:

* ***Профиль*** – WEB-страница, отображающая страницу профиля.
* ***Настройки*** – кнопка, перемещающая на страницу «Настройки».
* ***Выход из аккаунта*** – кнопка, при нажатии на которую, пользователь выйдет из аккаунта.
* ***Преподавание*** – кнопка, перемещающая на страницу «Преподавания».

1. Фрагмент схемы, демонстрирующий страницу профиля:

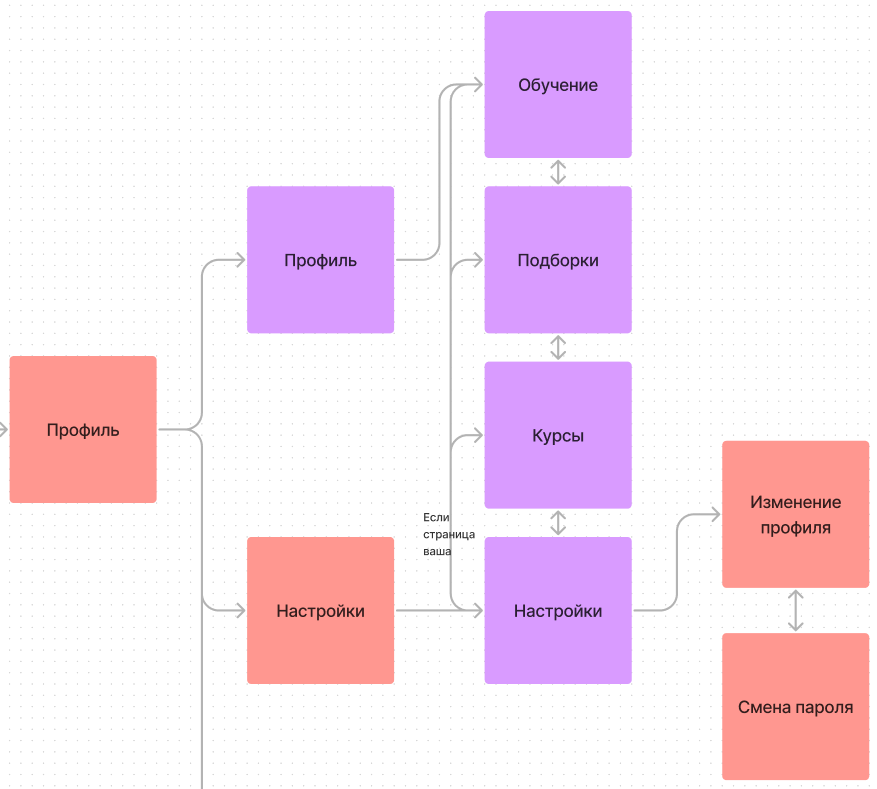


Рисунок 25. Фрагмент схемы, демонстрирующий страницу профиля

Данная схема (рисунок 25) демонстрирует страницу профиля, которая делится на следующие страницы:

* ***Обучение*** – WEB-страница, отображающая статистику пользователя.
* ***Подборки*** – WEB-страница, отображающая добавленные и созданные подборки.
* ***Курсы*** – WEB-страница, отображающая добавленные и созданные курсы.
* ***Настройки*** – WEB-страница, состоит из двух компонентов, а именно, «Изменение профиля», где меняются основная информация о пользователе, и компонент «Смена пароля», где меняется пароль.

1. Фрагмент схемы, демонстрирующий схему преподавания:

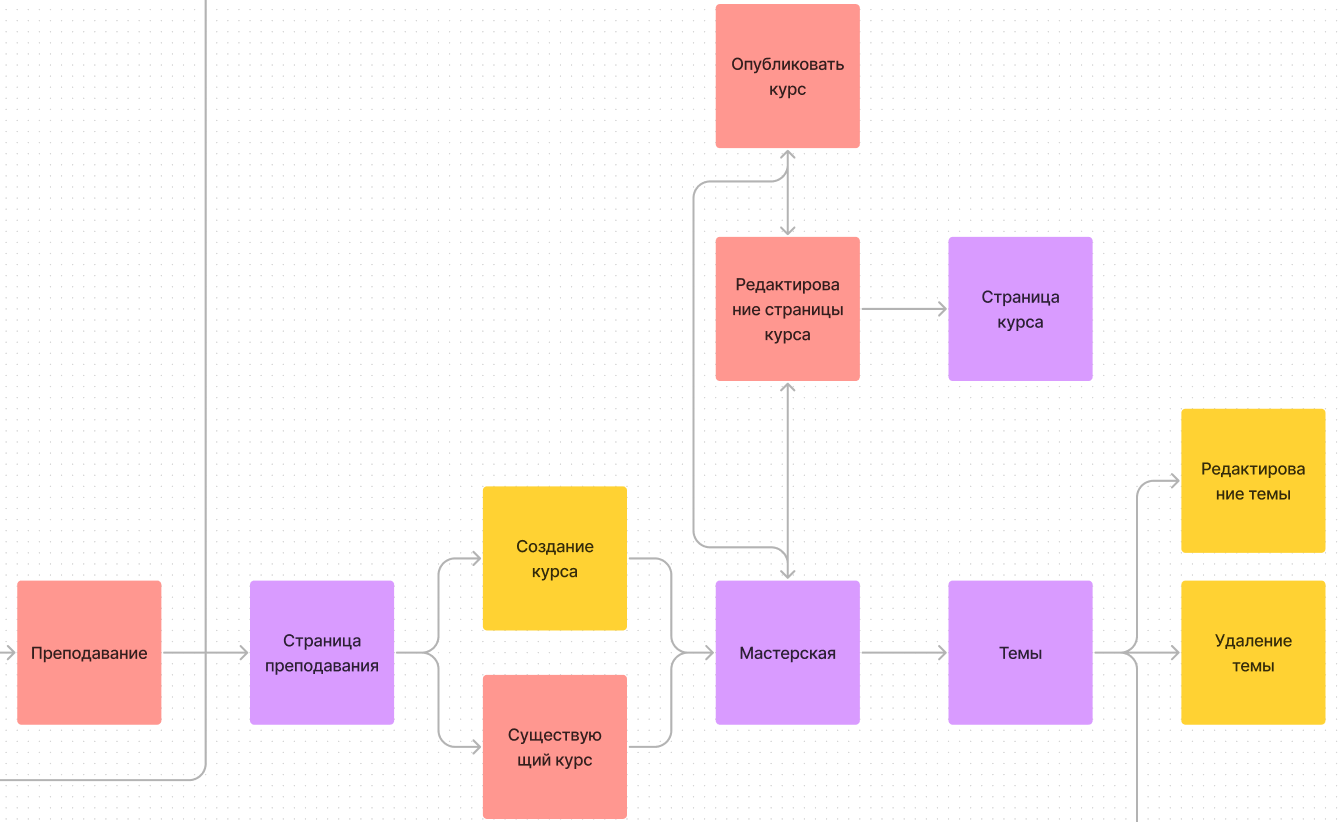


Рисунок 26. Фрагмент схемы, демонстрирующий схему преподавания

Данная схема (рисунок 26) демонстрирует схему преподавания. В этой схеме есть страница преподавания, которая выводит созданные вами курсы, а также даёт возможность создать курс. Из данных компонентов можно перейти на страницу «Мастерская».

WEB-страница «Мастерская» предоставляет возможность, редактирования титульной страницы курса, а также возможность опубликовать курс.

Из WEB-страницы «Мастерская», можно перейти на страницу «Темы», на ней выводятся все темы курса, а также предоставлена возможность редактирования и удаления отдельных тем.

1. Фрагмент схемы, демонстрирующий углубленную схему преподавания:

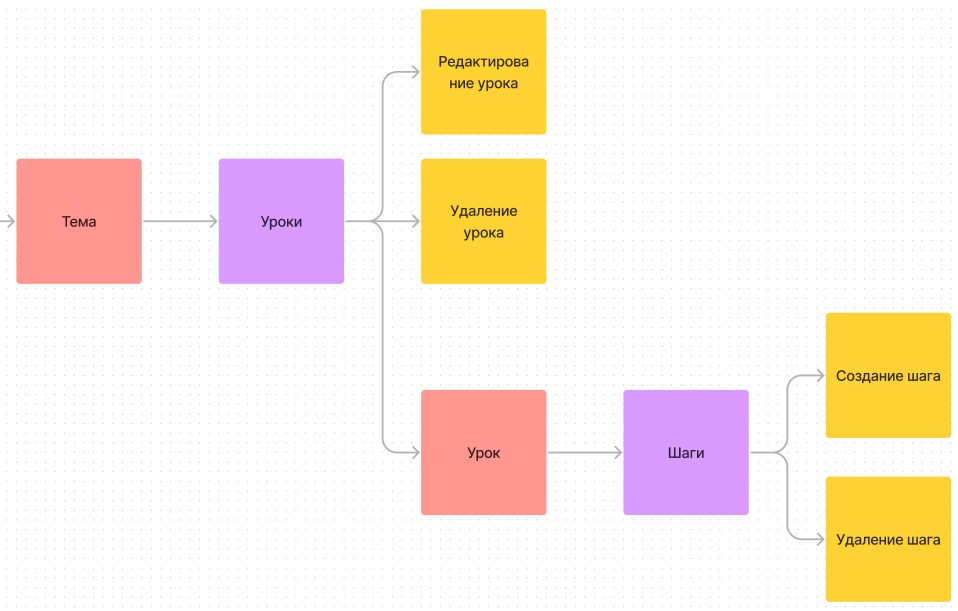


Рисунок 27. Фрагмент схемы, демонстрирующий углубленную схему преподавания

Данная схема (рисунок 27) демонстрирует схему преподавания. В располагается страница «Уроки», которая выводит все уроки текущей темы, а также предоставляет возможность изменять или удалять отдельный урок.

Из WEB-страницы «Уроки», можно перейти на страницу «Шаг». На ней выводятся контент шага. Также предоставлена возможность редактирования и удаления шагов.

* + 1. Back-end модуль

Back-end модуль включает в себя 15 файлов программного кода, разработанных студентов. В соответствии с принципами ООП, каждый класс выполняет отведённую ему роль. Также классы чётко разделены по собственному функционалу, что и отображено в самих названиях классов.

Файлы кода распределены так, есть ядро back-end модуля, а есть приложение back-end модуля, которое содержит файлы модели, представления и сериализации, а также дополнительные файлы к проекту.

1. Модуль ядра:

***settings.py*** – модуль, содержащий все настройки проекта.

***urls.py*** – модуль или файл, который описывает правила маршрутизации запросов. Файл с url адресами.

***wsgi.py*** и ***asgi.py*** – точки входа в wsgi и asgi – приложения.

1. Модуль приложения. Не основные модули:

***models.py*** – содержит переделанные системные модели. Отвечает за хранение моделей для таблиц в базе данных.

***urls.py*** – модуль или файл, который объединяет все файлы маршрутизаторы.

***utils.py*** – модуль или файл, хранящий все классы «помощники».

***test.py*** – модуль, содержащий в себе тесты приложения.

***admin.py*** – модуль содержит в себе настройки административной части.

***manage.py*** – этот файл находится в корневой папке проекта. Необходим для выполнения различных действий над проектом.

1. Модуль auth:

***views\_auth.py*** – это представления, занимающиеся основной работой с пользователем (регистрация, авторизация). Представления запрашивают данные из модели, и передают эти данные клиенту.

***serializers\_auth.py*** – это серилизаторы, отвечающие за преобразование из сложных представлений модели пользователя в текстовый документ формата JSON. Также обеспечивает и обратный процесс преобразования данных из файла формата JSON в сложные типы данных.

***urls\_auth.py*** – это модуль маршрутизации представлений пользователя.

1. Модуль collection:

***models\_collection.py*** – это модели, отвечающие за перевод классов связанных с подборками в таблицы располагающиеся в Базе Данных.

***views\_collection.py*** – это представления, занимающиеся основной работой с подборками. Представления запрашивают данные из модели, и передают эти данные клиенту.

***serializers\_collection.py*** – это серилизаторы, отвечающие за преобразование из сложных представлений модели подборок в текстовый документ формата JSON. Также обеспечивает и обратный процесс преобразования данных из файла формата JSON в сложные типы данных.

***urls\_collection.py*** – это модуль маршрутизации представлений подборок.

1. Модуль course:

***models\_course.py*** – это модели, отвечающие за перевод классов связанных с курсами в таблицы располагающиеся в Базе Данных.

***views\_course.py*** – это представления, занимающиеся основной работой с курсами. Представления запрашивают данные из модели, и передают эти данные клиенту.

***serializers\_course.py*** – это серилизаторы, отвечающие за преобразование из сложных представлений модели курсов в текстовый документ формата JSON. Также обеспечивает и обратный процесс преобразования данных из файла формата JSON в сложные типы данных.

***urls\_course.py*** – это модуль маршрутизации представлений курсов.

1. Модуль profile:

***models\_profile.py*** – это модели, отвечающие за перевод классов связанных с профилями пользователей в таблицы располагающиеся в Базе Данных.

***views\_profile.py*** – это представления, занимающиеся основной работой с профилями пользователей. Представления запрашивают данные из модели, и передают эти данные клиенту.

***serializers\_profile.py*** – это серилизаторы, отвечающие за преобразование из сложных представлений модели профилей пользователей в текстовый документ формата JSON. Также обеспечивает и обратный процесс преобразования данных из файла формата JSON в сложные типы данных.

***urls\_ profile.py*** – это модуль маршрутизации представлений профилей пользователей.

* 1. Описание структуры программного комплекса

Программный комплекс представляет собой клиент-сервер (рисунок 28).

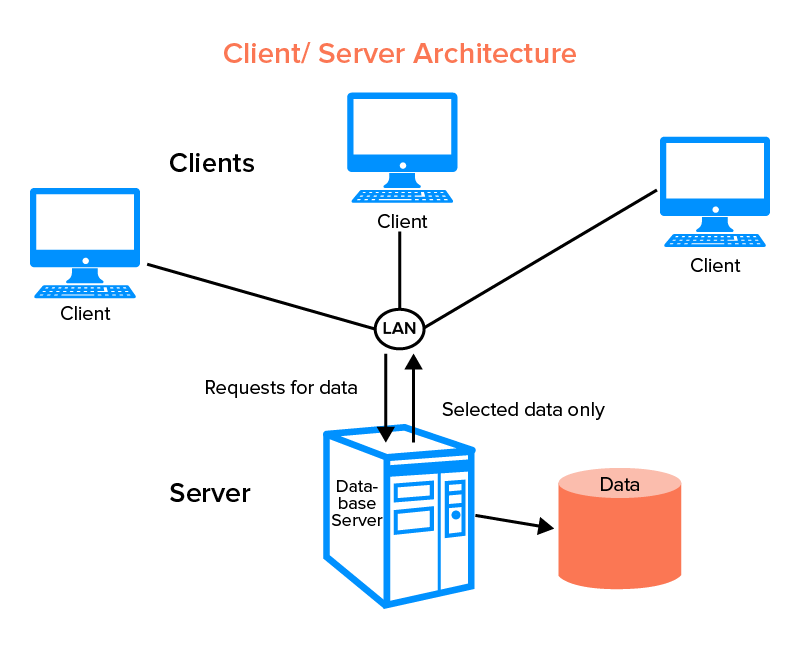


Рисунок 28. Схема архитекруты клиент-сервер

Клиент взаимодействует с сервером, посредством протокола HTTP, для получения и отправки данных.

Клиент отправляет данные на сервер в формате JSON. Сервер принимает этот файл JSON, сериализирует его, и обновляет Базу Данных.

При отправке данных с сервера происходит аналогично обратный сценарий.

* 1. Описание процедур и функций

В этом подразделе будет рассматриваться модуль ***view\_course.py***.

1. Класс «CourseView».

Это представление, которое связанно с моделью «Course». В нем реализовывались основные запросы к API типа GET.

Методы:

* exists\_access\_page – Есть ли доступ к данной странице у текущего пользователя.
* exists\_path – Определяет существует ли необходимый путь к странице.
* exists\_profile\_path – Определяет существует ли такой путь к профилю.
* get\_frame\_pagination - возвращает каркас к пагинации.
* swap\_filters\_field - Смена типа фильтрации.
* get\_courses – GET запрос к API. Вернет все курсы учитывая пагинацию и фильтрацию.
* get\_mini\_courses – GET запрос к API. Вернет все курсы в мини-форме, учитывая пагинацию и фильтрацию.
* get\_all\_profile\_courses – GET запрос к API. Вернет все добавленные и созданные курсы пользователем по path, в мини-форме, учитывая пагинацию и фильтрацию.
* get\_added\_courses – GET запрос к API. Вернет все добавленные курсы пользователем по path, в мини-форме, учитывая пагинацию и фильтрацию.
* get\_created\_courses – GET запрос к API. Вернет все созданные курсы пользователем по path, в мини-форме, учитывая пагинацию и фильтрацию.
* get\_page\_course – GET запрос к API. Вернет страницу курса.

1. Класс «ActionCourseView».

Это представление, которое связанно с моделью «Course». В нем реализовывались запросы, связанные с действием над курсом.

Методы:

* exists\_path – Определяет существует ли необходимый путь к странице.
* create\_course – Cоздание курса.
* delete\_course – Удаление курса.
* publish\_course – Опубликовать курс.
* development\_course – Перевод курса в разработку.

1. Класс «CoursePageView».

Это представление, которое связанно с моделью «CourseInfo». В нем реализовывались запросы, связанные со страницей курса.

Методы:

* get\_page – Вернет страницу курса.
* update\_course – Обновит курс.
* update\_main\_info – Обновит главную информацию о курсе.
* update\_fits – Обновит представителей.
* update\_skills – Обновит навыки.
* save\_page – Сохранение страницы.

1. Класс «CourseFitView».

Это представление, которое связанно с моделью «CourseFit». В нем реализовывались запросы, связанные с представителями страницы курса.

Методы:

* exists – Существует ли такой представитель.
* create\_fit – Создать представителя.
* update\_fit – Обновить представителя.
* delete\_fit – Удалить представителя.

1. Класс «CourseSkillView».

Это представление, которое связанно с моделью «CourseFit». В нем реализовывались запросы, связанные с навыками страницы курса.

Методы:

* exists – Существует ли такой навык.
* create\_skill– Создать навык.
* update\_skill – Обновить навык.
* delete\_skill – Удалить навык.

1. Класс «CourseCompletionPageView».

Это представление, которое связанно с моделями «Course», «Theme», «Lesson», «Step». В нем реализовывались запросы, связанно с прохождением курса.

Методы:

* get\_title\_course – Вернет название курса.
* get\_themes – Вернет темы курса.
* get\_title\_theme – Вернет название темы.
* get\_lessons – Вернет уроки темы.
* get\_steps – Вернет шаги урока
* add\_profile\_action\_logs – Ведение логов по изучению курса.
* add\_profile\_step – Добавить связь профиля с шагом.
* get\_detail\_step – Вернет детальную страницу шага.
* exists\_profile\_step – Изучали ли вы данный шаг.
* complete\_step – Завершить шаг.
* start\_learn\_course – Начать изучение курса.
* complete\_learn\_course – Завершить изучение курса.

1. Класс «ThemeView».

Это представление, которое связанно с моделью «Theme». В нем реализовывались запросы, влияния на модель «Theme».

Методы:

* create\_theme – Создание темы.
* get\_update\_info – Вернуть данные для обновления.
* update\_theme – Обновить тему.
* delete\_theme – Удалить тему.

1. Класс «LessonView».

Это представление, которое связанно с моделью «Lesson». В нем реализовывались запросы, влияния на модель «Lesson».

Методы:

* create\_lesson – Создание урока.
* get\_update\_info – Вернуть данные для обновления.
* update\_lesson – Обновить урок.
* delete\_lesson – Удалить урок.

1. Класс «StepView».

Это представление, которое связанно с моделью «Step». В нем реализовывались запросы, влияния на модель «Step».

Методы:

* create\_step – Создание шага.
* get\_update\_info – Вернуть данные для обновления.
* update\_step – Обновить шаг.
* delete\_step – Удалить шаг.

1. Класс «ActionProfileCourseView».

Это представление, которое связанно с моделью «Course». В нем реализовывались запросы, действия профиля к курсу.

Методы:

* exists\_course – Существует ли такой курс.
* exists\_collection – Существует ли такая подборка.
* get\_course – Вернет курс.
* added\_courses – Добавить курс.
* popped\_courses – Удалить курс из добавленных.

1. Класс «GradeCourseView».

Это представление, которое связанно с моделью «Course». В нем реализовывались запросы, действия профиля к курсу.

Методы:

* exists\_path – Существует ли такой путь.
* set\_grade – Установить оценку.
* update\_grade – Обновить оценку.
* delete\_grade – Удалить оценку.

1. Класс «PathValidator».

Класс проверки валидности данных.

Методы:

* get\_exists – Существует ли такая страница.
* is\_valid – Полная проверка на валидность.
* exists – Все ли страницы существуют.
* exists\_course – Существует ли курс.
* exists\_theme– Существует ли тема.
* exists\_lesson – Существует ли урок.
* exists\_step – Существует ли шаг.
* is\_access – Есть ли доступ.
  1. Методы проектирования и разработки приложения

Для разрабатываемого проекта в качестве метода проектирования было выбрано структурное проектирование, так как для более эффективной разработки проекта и его последующего поддержания необходимо четко структурировать и разбить на модули программу (рисунок 29).



Рисунок 29. Структурное проектирование проекта.

В качестве метода разработки была выбрана инкрементная модель разработки (рисунок 30), так как в последствии выполнения проекта и после его выполнения может потребоваться расширить возможности ПО, не затрагивая основные функции ПО.

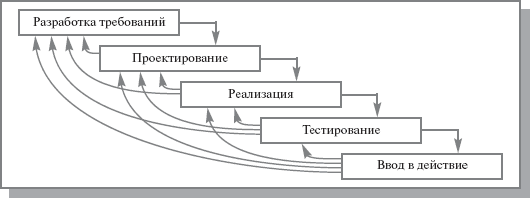


Рисунок 30. Инкрементная модель.

1. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ
   1. Эксплуатационный раздел

В данном разделе приведено руководство по применению и эксплуатации WEB-клиента.

В разделе «Назначение программы» указаны сведения о назначении WEB-клиента и информация, которая необходима для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия для выполнения программы. То есть указан минимальный состав аппаратных и программных средств.

В разделе «Выполнение программы» указана последовательность действий оператора для запуска, выполнения некоторых задач и завершение программы. Также приведено описание функций и варианты запросов, с помощью которых и проводится отладка WEB-клиента.

Раздел по эксплуатации WEB-клиента содержит сведения о назначении, составе, принципе действия, а также и другие сведения, необходимые для правильной эксплуатации back-end модуля.

Данный раздел по эксплуатации содержит сведения о назначении, составе, принципе действия, а также другие сведения, необходимые для правильной эксплуатации программного обеспечения.

* + 1. Рекомендации по эксплуатации программного продукта

Функциональным назначением программы является возможность прохождения курсов.

Программа не предназначена для эксплуатации на какой-то либо определённой операционной системе. Для запуска программы самое главное — это подключение к интернету.

Минимальный состав используемых технических (аппаратных) средств:

Процессор: Intel Pentium 4 или мощнее;

* ОЗУ: 2ГБ или больше.
* Место на жёстком диске: 10 ГБ (для 64-разрядной системы) или больше.
* Подключение к интернету.

Минимальный состав средств нужен для тестирования WEB-клиента, все равно программа будет записываться на хост. Запускаться программа должна на хосте.

* 1. Раздел техники безопасности

При эксплуатации ПК на работника могут оказывать действия, следующие опасные и вредные производственные факторы:

Повышенный уровень электромагнитных излучений;

Повышенный уровень статического электричества;

Повышенная ионизация воздуха;

Статические физические перегрузки;

Перенапряжение зрительных анализаторов;

Работник обязан:

Выполнять только ту работу, которая определена его должностной инструкцией.

Содержать в чистоте рабочее место.

Соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности.

Соблюдать меры пожарной безопасности.

К работе с ПК допускаются лица: имеющие персональные навыки, изучившие руководство по эксплуатации персонального компьютера и знающие порядок включения и отключения электронных устройств, а также программ; прошедшие вводный инструктаж, а также инструктаж по безопасности труда непосредственно на рабочем месте. Профессиональные пользователи должны проходить обязательные предварительные (при поступлении на работу) и периодические медосмотры. К непосредственной работе с персональным компьютером допускаются лица, не имеющие медицинских противопоказаний.

Также следует иметь в виду, что:

Расстояние между рабочими столами с компьютерами в направлении тыла поверхности одного видеомонитора и экрана другого видеомонитора.

Площадь на одно рабочее место ПК должна составлять не менее 6,0 кв. м, а объем - не менее 20,0 куб. м.

Верхний край экрана следует располагать на уровне глаз или чуть ниже.

Оптимальное расстояние от глаз до экрана 600-700 мм, но не ближе 500 мм.

Высота клавиатуры должна быть отрегулирована так, чтобы кисти рук держались прямо (возможно применение подставок для кистей рук).

С целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления необходимо, чтобы рабочий стул кресло) позволял изменять позу, был подъемно - поворотным и регулируемым по высоте и углам наклона сидения и спинки от переднего сидения.

В производственных помещениях, в которых работа на ПК является основной должны обеспечиваться оптимальные параметры микроклимата. Для нормализации аэроионного фактора помещений с компьютерами необходимо использовать устройства автоматического регулирования ионного режима воздушной среды.

Рабочие места с ПК при выполнении творческой работы, требующей значительного умственного напряжения или высокой концентрации внимания, следует изолировать друг от друга перегородками высотой 1,5-2,0 м.

В помещениях с ПК ежедневно должна проводится влажная уборка.

Помещения с ПК должны быть оснащены аптечкой первой помощи и углекислотными огнетушителями.

Кабели компьютера должны располагаться так, чтобы их нельзя было повредить неосторожным движением.

Для предотвращения образовании и защиты от статического электричества, в помещении, где установлен компьютер, необходимо применять нейтрализаторы и увлажнители. Должна быть эффективная вентиляция и поддерживаться относительная влажность воздуха на уровне 40-60 % (можно разместить вблизи компьютера цветы или аквариум).

Для обеспечения оптимальной работоспособности и сохранности здоровья пользователей, на протяжении рабочей смены должны устанавливаться регламентированные перерывы. Продолжительность непрерывной работы с персональным компьютером без регламентированного перерыва не должна превышать 2 часов.

Невыполнение требований является нарушением производственной дисциплины. Виновные в этом несут ответственность, в порядке, установленном действующим законодательством.

При начале работы с ПК, работник обязан:

Осмотреть и привести в порядок рабочее место.

Отрегулировать освещенность на рабочем месте, убедиться в достаточности освещенности, отсутствии бликов на экране.

Проверить правильность подключения оборудования в электросеть.

Проверить исправность проводящих проводов и отсутствие оголённых участков проводов.

Убедиться в наличии защитного заземления.

Протереть салфеткой поверхность экрана.

Убедиться в отсутствии дисков в дисководах ПК.

Проверить правильность установки стола, стула, положения оборудования, угла наклона экрана, положение клавиатуры, положение мыши на специальном коврике, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения длительных напряжений тела.

Работнику запрещается приступать к работе с ПК при:

* Отсутствии специальной вилки с подключением заземления.
* Обнаружении неисправности оборудования.

При размещении ПК в ряд на расстоянии менее 1,2 м, при расположении рабочих мест с компьютерами в колонку на расстоянии менее 2,0 м, при рядном расположении дисплеев экранами друг к другу.

Работнику запрещается протирать влажной или мокрой салфеткой электрооборудование, которое находится под напряжением. Влажную или любую другую уборку производить только при отключенном оборудовании.

Работник обязан сообщить руководителю об обнаруженной неисправности оборудования. Не пользоваться неисправным оборудованием. Приступить к работе только после устранения нарушений в работе или неисправностей оборудования.

Работник не должен разрешать работать лицам, не имеющим допуска к работе с опасным оборудованием или с ПК.

* 1. Расчет затрат на разработку программного комплекса

Основная заработная плата студента-разработчика рассчитывается по формуле 1:

, где О – оклад, 1,15 – областной коэффициент (1)

Оклад Full stack разработчика равен: 70000 рублей. Его основная месячная заработная плата составит:

70000 \* 1,15 = 80 500 рублей.

Основная заработная плата разработчика за весь период разработки программного продукта составит:

80 500 \* (240ч. / (8ч. \* 22 дня)) = 109 772 руб.

Дополнительная заработная плата рассчитывается в процентах от основной заработной платы и составляет 12%.

Дополнительная заработная плата разработчика за весь период разработки программного продукта составит:

109 772 руб. \* 0,12 = 13 172 руб.

Суммарная заработная плата разработчика за весь период разработки программного продукта составит:

109 772 руб. + 13 172 руб. = 122 944 руб.

Страховые взносы во внебюджетные фонды для IT-организаций в 2021 году составляют 7,6% от суммы основной и дополнительной заработной платы.

Суммарные страховые взносы во внебюджетные фонды за весь период разработки программного продукта составят:

122 944 руб. \* 7,6% = 9 343 руб.

Содержание и эксплуатация вычислительного комплекса считается следующим образом по формуле 2:

, где *См-ч* - стоимость машино-часа (2)

Число часов отладки составляет:

Стоимость машино-часа рассчитывается по формуле 3:

(3)

где *Фвр* - действительный фонд времени работы вычислительного комплекса.

Стоимость 1 КВТ/час электроэнергии составляет, например, 1,71 руб.

Один компьютер потребляет, например, 350 ВТ в час.

За месяц расходы на электроэнергию, потребляемую одной ЭВМ составляет:

8 ч. \* 22 дня \* 0,35 КВТ/ч. \* 1,71 руб. \* ч. / КВТ = 105,33 руб.

*Фвт* = 0,85 \* 240 час. = 204 часа.

Стоимость машино-часа составляет:

(105,33 руб. + 1300 руб. + 400 руб.)/ 204 часа = 8,8 руб.

Содержание и эксплуатация вычислительного комплекса составляет:

240ч. \* 8,8 руб. = 2123 руб.

Накладные расходы рассчитываются, как 60% от расходов на содержание и эксплуатацию вычислительного комплекса:

2088 руб.\*0,6 = 1252,8 руб.

Смета затрат на разработку программного продукта приведена в таблице 1.

Таблица 6

Смета затрат на разработку программного продукта

|  |  |  |
| --- | --- | --- |
| № | Наименование статьи расходов | Затраты, руб. |
| 1 | Основная заработная плата производственного персонала. | 109 772 |
| 2 | Дополнительная заработная плата производственного персонала. | 13 172 |
| 3 | Страховые взносы во внебюджетные фонды | 9 343 |
| 4 | Содержание и эксплуатация вычислительного комплекса. | 2 123 |
| 5 | Накладные расходы. | 1252,80 |
| ИТОГО: | 135 662 руб. |  |

Применение данной разработки позволит в самые кратчайшие сроки компенсировать затраты на разработку и эксплуатацию, получить экономический эффект от использования данного комплекса.

В ходе вышеприведённых вычислений были получены следующие результаты:

* Определена трудоёмкость разработки программного продукта, которая составила: 303 чел.-час.
* Рассчитана смета затрат на разработку программного продукта, итоговая сумма которой: 135 662 рублей.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта был разработан front-end модуль, в виде WEB-сайта, и back-end модуль, в виде REST API, по теме образовательного платформы.

Была изучена предметная область цифрового обучения на онлайн курсах и рассмотрены следующие представители:

* Лекториум.
* Stepik.
* ЯЮниор.
* Яндекс Практикум.
* Skillbox.

Затем были проанализированы существующие разработки в области цифрового обучения и выявлены критерии, по которым было выведено целесообразность разработки.

В ходе разработки выпускной квалификационной работы были выполнены следующие задачи:

* Разработка алгоритмов решения для разработки WEB-сайта и написания серверной части программы.
* Изучение и анализ инструментов для разработки.
* Разработка архитектуры программы, с учетом удобства и упрощения дальнейшей разработки.
* Разработана ER-модель базы данных для хранения данных.
* Разработаны модули с процедурами и функциями для разделения программы на функциональные части.

В организационной и экономической части представлены рекомендации по эксплуатации программного продукта, раздел техники безопасности, расчёт затрат на разработку программного комплекса.

В расчётах затрат на разработку программы были достигнуты следующие результаты в вычислениях:

* Определена трудоёмкость разработки программного продукта.
* Рассчитано, что понадобится для выполнения разработки.
* Рассчитана смета затрат на разработку back-end модуля.

В дальнейшем планируется добавить следующий функционал:

* Разделение пользователей и авторов.
* Создание курсов пользователем
* Добавление организаций.
* Возможность создания уникальной страницы для каждого курса.
* Добавление кейсов для решения проблем.
* Возможность разработки курсов совместно несколькими людьми.
* Возможность ведения подборок несколькими авторами.
* Возможность выкладывания платных курсов.
* Добавление новых типов заданий.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

* + 1. ИнфоУрок. [Электронный источник]. Режим доступа: https://infourok.ru/ (Дата обращения: 22.05.2022)
    2. Лекториум. [Электронный источник]. Режим доступа: https://www.lektorium.tv/ (Дата обращения: 20.05.2022)
    3. Мираполис. [Электронный источник]. Режим доступа: https://www.mirapolis.ru/ (Дата обращения: 22.05.2022)
    4. Статья инвестиций в образовательные проекты EdTech. [Электронный источник]. Режим доступа: https://inventure.com.ua/news/world/investicii-v-obrazovatelnye-proekty-edtech-v-2020-godu-dostigli-dollar161-mlrd (Дата обращения: 21.05.2022)
    5. ЯЮниор. [Электронный источник]. Режим доступа: https://ijunior.ru/unity-start (Дата обращения: 20.05.2022)
    6. Яндекс Браузера. [Электронный ресурс]. Режим доступa: https://browser.yandex.ru/ (Дата обращения: 17.05.2022)
    7. Яндекс Практикум. [Электронный источник]. Режим доступа: https://practicum.yandex.ru/ (Дата обращения: 20.05.2022)
    8. CSS. [Электронный источник]. Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS (Дата обращения: 23.05.2022)
    9. Django. [Электронный источник]. Режим доступа: https://www.djangoproject.com/ (Дата обращения: 23.05.2022)
    10. Firefox. [Электронный ресурс]. Режим доступа: https://www.mozilla.org/ru/firefox/ (Дата обращения: 17.05.2022)
    11. GetCourse. [Электронный источник]. Режим доступа: https://getcourse.ru/ (Дата обращения: 22.05.2022)
    12. Google Chrome. [Электронный ресурс]. Режим доступа: https://www.google.com/intl/ru/chrome/ (Дата обращения: 17.05.2022)
    13. HTML. [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTML (Дата обращения: 17.05.2022)
    14. Java. [Электронный источник]. Режим доступа: https://www.java.com/ru/ (Дата обращения: 23.05.2022)
    15. LingeuaLeo. [Электронный источник]. Режим доступа: https://lingualeo.com/ru (Дата обращения: 22.05.2022)
    16. MsSQL. [Электронный источник]. Режим доступа: https://www.microsoft.com/ru-ru/sql-server/ (Дата обращения: 23.05.2022)
    17. MySQL. [Электронный источник]. Режим доступа: https://www.mysql.com/ (Дата обращения: 23.05.2022)
    18. Netology. [Электронный источник]. Режим доступа: https://netology.ru/ (Дата обращения: 22.05.2022)
    19. NuxtJS. [Электронный источник]. Режим доступа: https://nuxtjs.org/ (Дата обращения: 23.05.2022)
    20. PHP. [Электронный источник]. Режим доступа: https://www.php.net/ (Дата обращения: 23.05.2022)
    21. PostgreSQL. [Электронный источник]. Режим доступа: https://www.postgresql.org/ (Дата обращения: 23.05.2022)
    22. PuzzleEnglish. [Электронный источник]. Режим доступа: https://puzzle-english.com/ (Дата обращения: 22.05.2022)
    23. Python. [Электронный источник]. Режим доступа: https://www.python.org/ (Дата обращения: 23.05.2022)
    24. ReactJS. [Электронный источник]. Режим доступа: https://reactjs.org/ (Дата обращения: 23.05.2022)
    25. SASS. [Электронный источник]. Режим доступа: https://sass-lang.com/ (Дата обращения: 23.05.2022)
    26. Safari. [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/safari/ (Дата обращения: 17.05.2022)
    27. Skillbox. [Электронный источник]. Режим доступа: https://skillbox.ru/ (Дата обращения: 20.05.2022)
    28. SkyEng. [Электронный источник]. Режим доступа: https://skyeng.ru/ (Дата обращения: 22.05.2022)
    29. Stepik. [Электронный источник]. Режим доступа: https://stepik.org/catalog (Дата обращения: 20.05.2022)
    30. VueJS. [Электронный источник]. Режим доступа: https://ru.vuejs.org/ (Дата обращения: 23.05.2022)
    31. eQueo. [Электронный источник]. Режим доступа: https://e-queo.com/ (Дата обращения: 22.05.2022)
    32. iSpring. [Электронный источник]. Режим доступа: https://www.ispring.ru/ (Дата обращения: 22.05.2022)

ПРИЛОЖЕНИЯ

Приложение 1. Листинг 1. manage.py

#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

"""Run administrative tasks."""

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'api.settings')

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Приложение 1. Листинг 2. urls.py

from django.contrib import admin

from django.urls import path, include

from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('', include('website.urls')),

path('api/', include('core.urls')),

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

path('api/token/', TokenObtainPairView.as\_view(), name="token"),

# path('api/auth/', include('rest\_auth.urls')),

path('api/refresh\_token/', TokenRefreshView.as\_view(), name="refresh\_token"),

path('ckeditor/', include('ckeditor\_uploader.urls')),

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

Приложение 1. Листинг 3. urls.py

from django.contrib import admin

from django.urls import path, include

from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('', include('website.urls')),

path('api/', include('core.urls')),

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

path('api/token/', TokenObtainPairView.as\_view(), name="token"),

# path('api/auth/', include('rest\_auth.urls')),

path('api/refresh\_token/', TokenRefreshView.as\_view(), name="refresh\_token"),

path('ckeditor/', include('ckeditor\_uploader.urls')),

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

Приложение 1. Листинг 4.models.py

import datetime

import jwt

from django.apps import apps

from django.contrib.auth.hashers import make\_password

from django.contrib.auth.models import UserManager, AbstractBaseUser, PermissionsMixin

from django.contrib.auth.validators import UnicodeUsernameValidator

from django.utils.translation import gettext\_lazy as \_

from django.utils import timezone

from .helpers.model import TrackingModel

from django.db import models

from django.conf import settings

class MyUserManager(UserManager):

"""

Custom user model manager where email is the unique identifiers

for authentication instead of usernames.

"""

def \_create\_user(self, username, email, password, \*\*extra\_fields):

"""

Create and save a user with the given username, email, and password.

"""

if not username:

raise ValueError("The given username must be set")

email = self.normalize\_email(email)

GlobalUserModel = apps.get\_model(self.model.\_meta.app\_label, self.model.\_meta.object\_name)

username = GlobalUserModel.normalize\_username(username)

user = self.model(username=username, email=email, \*\*extra\_fields)

user.password = make\_password(password)

user.save(using=self.\_db)

return user

def create\_user(self, username, email=None, password=None, \*\*extra\_fields):

"""Создание пользователя"""

extra\_fields.setdefault("is\_staff", False)

extra\_fields.setdefault("is\_superuser", False)

return self.\_create\_user(username, email, password, \*\*extra\_fields)

def create\_superuser(self, username, email=None, password=None, \*\*extra\_fields):

"""Создание админа"""

extra\_fields.setdefault("is\_staff", True)

extra\_fields.setdefault("is\_superuser", True)

if extra\_fields.get("is\_staff") is not True:

raise ValueError("Superuser must have is\_staff=True.")

if extra\_fields.get("is\_superuser") is not True:

raise ValueError("Superuser must have is\_superuser=True.")

return self.\_create\_user(username, email, password, \*\*extra\_fields)

class User(AbstractBaseUser, PermissionsMixin, TrackingModel):

"""Модель пользователя"""

username\_validator = UnicodeUsernameValidator()

username = models.CharField(

\_("username"),

max\_length=150,

help\_text=\_(

"Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only."

),

validators=[username\_validator],

error\_messages={

"unique": \_("A user with that username already exists."),

},

)

email = models.EmailField(\_("email address"), blank=False, unique=True)

is\_staff = models.BooleanField(

\_("staff status"),

default=False,

help\_text=\_("Designates whether the user can log into this admin site."),

)

is\_active = models.BooleanField(

\_("active"),

default=True,

help\_text=\_(

"Designates whether this user should be treated as active. "

"Unselect this instead of deleting accounts."

),

)

date\_joined = models.DateTimeField(\_("date joined"), default=timezone.now)

objects = MyUserManager()

# email\_verified = models.BooleanField(

# \_("email\_verified"),

# default=False,

# help\_text=\_(

# "Designates whether this user email is verified"

# ),

# )

EMAIL\_FIELD = "email"

USERNAME\_FIELD = "email"

REQUIRED\_FIELDS = []

@property

def token(self):

"""Выдача токена"""

token = jwt.encode({

'username': self.username,

'email': self.email,

'exp': datetime.datetime.now() + datetime.timedelta(hours=24)

}, settings.SECRET\_KEY, algorithm='HS256')

return token

Приложение 1. Листинг 5. utils.py

import os

from django.contrib.sites.shortcuts import get\_current\_site

from django.core.files.images import ImageFile

# from django.core.mail import send\_mail

from django.core.mail import EmailMessage

from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage

from django.urls import reverse

from rest\_framework.utils.urls import replace\_query\_param

from rest\_framework\_simplejwt.tokens import RefreshToken

class Util:

"""Класс с утилитами"""

PROFILE\_COURSE\_STATUS\_SEE\_NAME = 'Наблюдающий'

PROFILE\_COURSE\_STATUS\_STUDIED\_NAME = 'Завершен'

PROFILE\_COURSE\_STATUS\_STUDYING\_NAME = 'Изучается'

PROFILE\_COURSE\_ROLE\_ADMIN\_NAME = 'Admin'

PROFILE\_COURSE\_ROLE\_USER\_NAME = 'User'

COURSE\_STATUS\_DEV\_NAME = 'В разработке'

COURSE\_STATUS\_RELEASE\_NAME = 'Опубликован'

DEFAULT\_IMAGES = {

'profile': "!default-profile.jpg",

'collection': "!default-collection.png",

'course': "!default-course.png",

'theme': "!default-theme.png",

'lesson': "!default-lesson.png",

}

PROTOCOL = 'http'

@staticmethod

def get\_absolute\_url(request):

"""Возвращает абсолютный путь сайта"""

return f"{Util.PROTOCOL}://{get\_current\_site(request).domain}"

@staticmethod

def send\_email(data):

"""Отправку письма на email"""

email = EmailMessage(

subject=data['email\_subject'],

body=data['email\_body'],

to=[data['to\_email']],

from\_email=data['from\_email'],

)

email.send()

@staticmethod

def get\_absolute\_url\_token(request, to, user):

"""Вернет абсолютный путь сайта с токеном"""

token = RefreshToken.for\_user(user).access\_token

relative\_link = reverse(to)

return f"{Util.get\_absolute\_url(request)}{relative\_link}?token={token}"

@staticmethod

def exists\_path(model, validated\_data):

"""Существует ли путь к странице"""

return len(model.objects.filter(\*\*validated\_data)) != 0

@staticmethod

def get\_image(old, new, default):

"""Вернет новое изображение"""

try:

if new is None or len(new) == 0:

path\_image = "\\".join(old.path.split('\\')[:-1]) + "\\" + str(default)

new = ImageFile(open(path\_image, "rb"))

except FileNotFoundError:

path\_image = "\\".join(old.path.split('\\')[:-1]) + "\\" + str(default)

new = ImageFile(open(path\_image, "rb"))

return new

@staticmethod

def get\_update\_image(old, new):

"""Обновить изображение"""

if old.path != new.name:

try:

if (old is not None) and (old.name not in Util.DEFAULT\_IMAGES.values()):

os.remove(old.path)

except ValueError:

pass

return new

return old

@staticmethod

def get\_update\_path(new\_path):

"""Обновит путь"""

return "-".join(new\_path.split())

@staticmethod

def get\_new\_path(new\_path, old\_path, model):

"""Вернет новый путь"""

if new\_path is not None:

if len(new\_path) == 0:

raise ValueError("path не может быть пустым")

all\_path = model.objects.filter(path=new\_path)

if (len(all\_path) == 0) or (len(all\_path) == 1 and new\_path == old\_path):

return new\_path

else:

raise ValueError("Такой path уже существует")

return old\_path

@staticmethod

def get\_max\_path(queryset):

"""Вернет максимальный путь"""

max\_path = 0

for item in queryset:

if (type(item.path) == int) and (item.path > max\_path):

max\_path = item.path

return max\_path

class HelperFilter:

"""Фильтрация моделей"""

# COLLECTION

COLLECTION\_TYPE = 1

COLLECTION\_FILTER\_FIELDS = ('title', 'profile\_\_user\_\_username')

COLLECTION\_SEARCH\_FIELDS = ('title', 'profile\_\_user\_\_username')

COLLECTION\_ORDERING\_FIELDS = ('rating', 'title', 'date\_create')

PROFILE\_COLLECTION\_TYPE = 2

PROFILE\_COLLECTION\_FILTER\_FIELDS = ('collection\_\_title', 'collection\_\_profile\_\_user\_\_username')

PROFILE\_COLLECTION\_SEARCH\_FIELDS = ('collection\_\_title', 'collection\_\_profile\_\_user\_\_username')

PROFILE\_COLLECTION\_ORDERING\_FIELDS = ('collection\_\_rating', 'collection\_\_title')

# COURSE

COURSE\_TYPE = 3

COURSE\_FILTER\_FIELDS = ('title', 'profile\_\_user\_\_username')

COURSE\_SEARCH\_FIELDS = ('title', 'profile\_\_user\_\_username')

COURSE\_ORDERING\_FIELDS = ('rating', 'title')

PROFILE\_COURSE\_TYPE = 4

PROFILE\_COURSE\_FILTER\_FIELDS = ('course\_\_title', 'course\_\_profile\_\_user\_\_username')

PROFILE\_COURSE\_SEARCH\_FIELDS = ('course\_\_title', 'course\_\_profile\_\_user\_\_username')

PROFILE\_COURSE\_ORDERING\_FIELDS = ('course\_\_rating', 'course\_\_title')

# PROFILE

PROFILE\_TYPE = 5

PROFILE\_FILTER\_FIELDS = ('path', 'user\_\_username')

PROFILE\_SEARCH\_FIELDS = ('path', 'user\_\_username')

PROFILE\_ORDERING\_FIELDS = ('path', 'user\_\_username')

# GOAL Subscription

GOAL\_TYPE = 6

GOAL\_FILTER\_FIELDS = ('goal\_\_path', 'goal\_\_user\_\_username')

GOAL\_SEARCH\_FIELDS = ('goal\_\_path', 'goal\_\_user\_\_username')

GOAL\_ORDERING\_FIELDS = ('goal\_\_path', 'goal\_\_user\_\_username')

# SUBSCRIBER

SUBSCRIBER\_TYPE = 7

SUBSCRIBER\_FILTER\_FIELDS = ('subscriber\_\_path', 'subscriber\_\_user\_\_username')

SUBSCRIBER\_SEARCH\_FIELDS = ('subscriber\_\_path', 'subscriber\_\_user\_\_username')

SUBSCRIBER\_ORDERING\_FIELDS = ('subscriber\_\_path', 'subscriber\_\_user\_\_username')

@staticmethod

def get\_filters\_collection\_field(type\_filter):

"""Вернет подходящий фильтр для подборок"""

if type\_filter == HelperFilter.COLLECTION\_TYPE:

filter\_fields = HelperFilter.COLLECTION\_FILTER\_FIELDS

search\_fields = HelperFilter.COLLECTION\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.COLLECTION\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

elif type\_filter == HelperFilter.PROFILE\_COLLECTION\_TYPE:

filter\_fields = HelperFilter.PROFILE\_COLLECTION\_FILTER\_FIELDS

search\_fields = HelperFilter.PROFILE\_COLLECTION\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.PROFILE\_COLLECTION\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

@staticmethod

def get\_filters\_course\_field(type\_filter):

"""Вернет подходящий фильтр для курсов"""

if type\_filter == HelperFilter.COURSE\_TYPE:

filter\_fields = HelperFilter.COURSE\_FILTER\_FIELDS

search\_fields = HelperFilter.COURSE\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.COURSE\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

elif type\_filter == HelperFilter.PROFILE\_COURSE\_TYPE:

filter\_fields = HelperFilter.PROFILE\_COURSE\_FILTER\_FIELDS

search\_fields = HelperFilter.PROFILE\_COURSE\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.PROFILE\_COURSE\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

@staticmethod

def get\_filters\_profile\_field(type\_filter):

"""Вернет подходящий фильтр для профилей"""

if type\_filter == HelperFilter.PROFILE\_TYPE:

filter\_fields = HelperFilter.PROFILE\_FILTER\_FIELDS

search\_fields = HelperFilter.PROFILE\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.PROFILE\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

@staticmethod

def get\_filters\_subscription\_field(type\_filter):

"""Вернет подходящий фильтр для подписанных пользователей"""

if type\_filter == HelperFilter.GOAL\_TYPE:

filter\_fields = HelperFilter.GOAL\_FILTER\_FIELDS

search\_fields = HelperFilter.GOAL\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.GOAL\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

elif type\_filter == HelperFilter.SUBSCRIBER\_TYPE:

filter\_fields = HelperFilter.SUBSCRIBER\_FILTER\_FIELDS

search\_fields = HelperFilter.SUBSCRIBER\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.SUBSCRIBER\_ORDERING\_FIELDS

return filter\_fields, search\_fields, ordering\_fields

class HelperPaginatorValue:

"""Значения пагинаций моделей"""

COLLECTION\_MAX\_PAGE = 10

MINI\_COLLECTION\_MAX\_PAGE = 40

COURSE\_MAX\_PAGE = 20

MINI\_COURSE\_MAX\_PAGE = 40

PROFILE\_MAX\_PAGE = 20

MINI\_PROFILE\_MAX\_PAGE = 40

PAGE\_QUERY\_PARAM = 'page'

class HelperPaginator:

"""Пагинация моделей"""

def \_\_init\_\_(self, request, queryset, max\_page):

self.paginator = Paginator(queryset, max\_page)

self.link = request.build\_absolute\_uri()

self.current\_page\_num = self.get\_current\_page\_num(request=request)

self.page\_obj = self.get\_page()

@staticmethod

def get\_current\_page\_num(request):

"""Вернет текущий номер пагинации"""

return request.GET.get('page', 1)

def get\_page(self):

"""Вернет страницу"""

try:

return self.paginator.page(self.current\_page\_num)

except PageNotAnInteger:

return self.paginator.page(1)

except EmptyPage:

return self.paginator.page(self.paginator.num\_pages)

def get\_num\_pages(self):

"""Вернет количество страниц"""

return self.paginator.num\_pages

def get\_count(self):

"""Количество записей"""

return self.paginator.count

def get\_link\_next\_page(self):

"""Следующая страница"""

if not self.page\_obj.has\_next():

return None

page\_number = self.page\_obj.next\_page\_number()

return replace\_query\_param(self.link, HelperPaginatorValue.PAGE\_QUERY\_PARAM, page\_number)

def get\_link\_previous\_page(self):

"""Предыдущая страница"""

if not self.page\_obj.has\_previous():

return None

page\_number = self.page\_obj.previous\_page\_number()

return replace\_query\_param(self.link, HelperPaginatorValue.PAGE\_QUERY\_PARAM, page\_number)

Приложение 1. Листинг 6. urls.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

router = DefaultRouter()

urlpatterns = [

path('', include(router.urls)),

path('', include('core.auth.urls\_auth')),

path('', include('core.course.urls\_course')),

path('', include('core.collection.urls\_collection')),

path('', include('core.profile.urls\_profile')),

]

Приложение 1. Листинг 7. admin.py

from django.contrib import admin

from .models import User

from .profile.models\_profile import Profile, Subscription

from .course.models\_course import CourseStatus, Course, CourseInfo, CourseMainInfo, CourseFit, CourseStars, CourseSkill, \

Theme, Lesson, Step, \

ProfileCourseStatus, ProfileCourse, ProfileTheme, ProfileLesson, ProfileStepStatus, ProfileStep, \

CreatorCollection, ProfileCourseCollection, ProfileActionsLogs

from .collection.models\_collection import Collection, ProfileCollection, CollectionStars

# Register your models here.

# PROFILE

admin.site.register(User)

admin.site.register(Profile)

admin.site.register(Subscription)

# COURSE

admin.site.register(CourseStatus)

admin.site.register(CreatorCollection)

admin.site.register(Course)

# Page COURSE

admin.site.register(CourseInfo)

admin.site.register(CourseMainInfo)

admin.site.register(CourseFit)

admin.site.register(CourseStars)

admin.site.register(CourseSkill)

# Content COURSE

admin.site.register(Theme)

admin.site.register(Lesson)

admin.site.register(Step)

# Profile to Course START

admin.site.register(ProfileActionsLogs)

admin.site.register(ProfileCourseStatus)

admin.site.register(ProfileCourse)

admin.site.register(ProfileCourseCollection)

admin.site.register(ProfileTheme)

admin.site.register(ProfileLesson)

admin.site.register(ProfileStepStatus)

admin.site.register(ProfileStep)

# COLLECTION

admin.site.register(Collection)

admin.site.register(ProfileCollection)

admin.site.register(CollectionStars)

Приложение 1. Листинг 8. urls\_auth.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views\_auth import RegisterView, VerifyEmailView, RequestPasswordResetEmailView, PasswordTokenCheckAPI, \

SetNewPasswordAPIView, AuthView

router = DefaultRouter()

urlpatterns = [

path('', include(router.urls)),

path('register/', RegisterView.as\_view(), name='register'),

# path('login/', LoginView.as\_view(), name='login'),

path('user/', AuthView.as\_view(), name='auth'),

path('email-verify/', VerifyEmailView.as\_view(), name="email-verify"),

path('request-reset-email/', RequestPasswordResetEmailView.as\_view(), name="request-reset-email"),

path('password-reset/<uidb64>/<token>/', PasswordTokenCheckAPI.as\_view(), name="password-reset-confirm"),

path('password-reset-complete/', SetNewPasswordAPIView.as\_view(), name="password-reset-complete"),

]

Приложение 1. Листинг 9. views\_auth.py

import jwt

from django.conf import settings

from django.contrib.auth import authenticate

from ..models import User

from django.contrib.auth.tokens import PasswordResetTokenGenerator

from django.urls import reverse

from django.utils.encoding import smart\_str, smart\_bytes, DjangoUnicodeDecodeError

from django.utils.http import urlsafe\_base64\_encode, urlsafe\_base64\_decode

from rest\_framework import permissions, generics, status

from rest\_framework.response import Response

from .serializers\_auth import RegisterSerializer, RequestPasswordResetEmailSerializer, SetNewPasswordSerializer, \

LoginSerializer

from ..profile.models\_profile import Profile

from ..profile.serializers\_profile import UserSerializer, ProfileSerializer

from ..utils import Util

# class LoginView(generics.GenericAPIView):

# """"""

# permission\_classes = [permissions.AllowAny]

# serializer\_class = LoginSerializer

#

# def post(self, request, \*args, \*\*kwargs):

# email = request.data.get('email', None)

# password = request.data.get('password', None)

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

#

# if user:

# serializer = self.serializer\_class(user)

# return Response({

# 'path': serializer.data.get('path'),

# 'username': serializer.data.get('username'),

# 'email': serializer.data.get('email'),

# 'access': serializer.data.get('token'),

# }, status=status.HTTP\_200\_OK)

# return Response({'error': "Не правильная почта или пароль"}, status=status.HTTP\_401\_UNAUTHORIZED)

class AuthView(generics.GenericAPIView):

"""Получение данных пользователя"""

serializer\_class = ProfileSerializer

permission\_classes = [permissions.IsAuthenticated]

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

"""Вернет данные пользователя"""

user = request.user

profile = Profile.objects.get(user=user)

return Response(ProfileSerializer(profile, context=self.get\_serializer\_context()).data, status=status.HTTP\_200\_OK)

# Create your views here.

class RegisterView(generics.GenericAPIView):

"""View. Регистрация"""

permission\_classes = [permissions.AllowAny]

serializer\_class = RegisterSerializer

def post(self, request):

"""POST запрос на регистрацию пользователя"""

serializer = self.serializer\_class(data=request.data)

serializer.is\_valid(raise\_exception=True)

user = serializer.save()

absolute\_url = Util.get\_absolute\_url\_token(request, to='email-verify', user=user)

email\_body = f"Здравствуйте, {user.username}!\n" \

f"Спасибо, что зарегистрировались на нашей платформе!\n" \

f"Теперь вы примкнули к кузне, где каждый Кузнец, который стремится стать мастером жадно поглощает " \

f"курсы, постигая всё то, что до нас сделали другие...\n\n" \

f"Для подтверждения аккаунта перейдите по этой ссылке: {absolute\_url}\n\n" \

f"--\n" \

f"С уважением, от \"FUJI\""

data = {

'email\_subject': 'Успешная регистрация',

'email\_body': email\_body,

'to\_email': user.email,

'from\_email': 'teamfuji@yandex.ru',

}

Util.send\_email(data)

return Response({

'user': UserSerializer(user, context=self.get\_serializer\_context()).data,

'message': "Пользователь успешно создан"

}, status=status.HTTP\_200\_OK)

class VerifyEmailView(generics.GenericAPIView):

"""View. Верификация пользователя"""

permission\_classes = [permissions.AllowAny]

def get(self, request):

"""GET. Активация аккаунта"""

token = request.GET.get('token')

try:

payload = jwt.decode(token, settings.SECRET\_KEY)

profile = Profile.objects.get(user=payload['user\_id'])

if not profile.is\_verified:

profile.is\_verified = True

profile.save()

return Response({'email': "Аккаунт успешно активирован"}, status=status.HTTP\_200\_OK)

return Response({'email': "Ваш аккаунт уже активирован"}, status=status.HTTP\_200\_OK)

except jwt.ExpiredSignatureError as ex:

return Response({'error': "Не удалось активировать аккаунт"}, status=status.HTTP\_400\_BAD\_REQUEST)

except jwt.exceptions.DecodeError as ex:

return Response({'error': "Invalid token"}, status=status.HTTP\_400\_BAD\_REQUEST)

class RequestPasswordResetEmailView(generics.GenericAPIView):

"""

Восстановление пароля по Email.

Сверяет наличие такого email'a и отправляет на него закодированную

ссылку на восстановление аккаунта

"""

permission\_classes = [permissions.AllowAny]

serializer\_class = RequestPasswordResetEmailSerializer

def post(self, request):

"""POST. Cброс пароля по email"""

attrs = request.data

email = attrs.get('email', '')

users = User.objects.filter(email=email)

if users.exists():

user = users[0]

uidb64 = urlsafe\_base64\_encode(smart\_bytes(user.pk))

token = PasswordResetTokenGenerator().make\_token(user)

relative\_link = reverse('password-reset-confirm', kwargs={

'uidb64': uidb64,

'token': token

})

absolute\_url = f"{Util.get\_absolute\_url(request)}{relative\_link}"

email\_body = f"Здравствуйте {user.username}!\n" \

f"Для восстановления пароля перейдите по этой ссылке: {absolute\_url}\n" \

f"Если это были не вы, то проигнорируйте данное письмо\n\n" \

f"--\n" \

f"С уважением, от \"FUJI\""

data = {

'email\_subject': 'Сброс пароля',

'email\_body': email\_body,

'to\_email': user.email,

'from\_email': 'teamfuji@yandex.ru',

}

Util.send\_email(data)

return Response({

'message': 'Мы отправили вам на почту ссылку для сброса пароля',

}, status=status.HTTP\_200\_OK,)

class PasswordTokenCheckAPI(generics.GenericAPIView):

"""Декодирование ссылки"""

permission\_classes = [permissions.AllowAny]

def get(self, request, uidb64, token):

"""GET. Вернёт валиден ли токен на сброс пароля"""

try:

user\_pk = smart\_str(urlsafe\_base64\_decode(uidb64))

user = User.objects.get(pk=user\_pk)

if not PasswordResetTokenGenerator().check\_token(user, token):

return self.request\_error()

return Response({

'message': 'Token is valid',

'uidb64': uidb64,

'token': token,

},

status=status.HTTP\_200\_OK,

)

except DjangoUnicodeDecodeError as ex:

print(ex)

return self.request\_error()

@staticmethod

def request\_error():

"""Запрос с ошибкой"""

return Response({

'message': 'Token is not valid, please request a new one',

},

status=status.HTTP\_400\_BAD\_REQUEST

)

class SetNewPasswordAPIView(generics.GenericAPIView):

"""Создание нового пароля"""

permission\_classes = [permissions.AllowAny]

serializer\_class = SetNewPasswordSerializer

def patch(self, request):

"""PATCH. Обновление пароля"""

serializer = self.serializer\_class(data=request.data)

serializer.is\_valid(raise\_exception=True)

return Response({

'message': 'Password reset success',

},

status=status.HTTP\_200\_OK,

)

Приложение 1. Листинг 10. serializers\_auth.py

import re

from ..models import User

from django.contrib.auth.tokens import PasswordResetTokenGenerator

from django.utils.encoding import force\_str

from django.utils.http import urlsafe\_base64\_decode

from rest\_framework import serializers

from ..profile.models\_profile import Profile

class LoginSerializer(serializers.ModelSerializer):

"""Сериализация. Login'a"""

path = serializers.SerializerMethodField()

password = serializers.CharField(max\_length=128, min\_length=8, write\_only=True)

class Meta:

model = User

fields = ('path', 'username', 'email', 'password', 'token')

read\_only\_fields = ('token',)

def get\_path(self, user):

"""Вернет путь до пользователя"""

return Profile.objects.get(user=user).path

class RegisterSerializer(serializers.ModelSerializer):

"""Сериализация. Регистрация пользователя"""

password = serializers.CharField(max\_length=128, min\_length=8, write\_only=True)

class Meta:

model = User

fields = ('username', 'email', 'password',)

extra\_kwargs = {

'password': {'write\_only': True}

}

@staticmethod

def password\_is\_valid(password):

"""Проверка пароля на валидность"""

reg = r'^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*#?&])[A-Za-z\d@$!#%\*?&]{8,}'

if re.fullmatch(reg, password):

return True

return False

def create(self, validated\_data):

"""Создание пользователя"""

username = validated\_data['username']

email = validated\_data['email']

password = validated\_data['password']

len\_username = len(username)

if 1 > len\_username or len\_username > 32:

raise serializers.ValidationError({'error': 'username должно быть не больше 32 символов и не меньше 3'})

if len(User.objects.filter(email=email)) != 0:

raise serializers.ValidationError({'error': 'Пользователь с таким email уже существует'})

if not self.password\_is\_valid(password):

raise serializers.ValidationError({'error': 'Некорректный пароль'})

user = User(username=username, email=email)

user.set\_password(password)

user.save()

return user

class RequestPasswordResetEmailSerializer(serializers.ModelSerializer):

"""Восстановление пароля по Email."""

class Meta:

model = User

fields = 'email'

class SetNewPasswordSerializer(serializers.ModelSerializer):

"""Создание нового пароля"""

password = serializers.CharField(write\_only=True)

repeat\_password = serializers.CharField(write\_only=True)

token = serializers.CharField(min\_length=1, write\_only=True)

uidb64 = serializers.CharField(min\_length=1, write\_only=True)

class Meta:

model = User

fields = ('password', 'repeat\_password', 'token', 'uidb64')

def validate(self, attrs):

"""Проверка на валидность токена"""

try:

password = attrs.get('password')

repeat\_password = attrs.get('repeat\_password')

token = attrs.get('token')

uidb64 = attrs.get('uidb64')

user\_pk = force\_str(urlsafe\_base64\_decode(uidb64))

user = User.objects.get(pk=user\_pk)

if not PasswordResetTokenGenerator().check\_token(user, token):

raise Exception("AuthenticationFailed", "The reset link is invalid", 401)

if password != repeat\_password:

raise serializers.ValidationError({'password': 'Пароли не совпадают'})

user.set\_password(password)

user.save()

except Exception:

raise Exception("AuthenticationFailed", "The reset link is invalid", 401)

return super().validate(attrs)

Приложение 1. Листинг 11. models\_profile.py

from django.db import models

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

from ..utils import Util

from ..models import User

class Profile(models.Model):

"""Advanced User"""

user = models.OneToOneField(User, on\_delete=models.CASCADE)

description = models.TextField(blank=True, null=True)

path = models.CharField(max\_length=64)

avatar\_url = models.ImageField(default=Util.DEFAULT\_IMAGES.get('profile'))

wrapper\_url = models.ImageField(blank=True)

is\_verified = models.BooleanField(default=False)

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

return f'{self.user}'

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

"""When a user is created, a profile is also created"""

if kwargs['created']:

profile = Profile.objects.create(user=kwargs['instance'])

profile.path = profile.pk

profile.save()

post\_save.connect(create\_profile, sender=User)

class Subscription(models.Model):

"""

goal - the GOAL one who subscribed

subscriber - one who subscribes

"""

goal = models.ForeignKey(Profile, related\_name="goal", on\_delete=models.CASCADE)

subscriber = models.ForeignKey(Profile, related\_name="subscriber", on\_delete=models.CASCADE)

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

return f'{self.subscriber.user.username} => {self.goal.user.username}'

Приложение 1. Листинг 12. urls\_profile.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views\_profile import ProfileView, SubscriptionProfileView, CourseProfileView, ActionProfileView

router = DefaultRouter()

urlpatterns = [

path('', include(router.urls)),

# GET

path('profile/', ProfileView.as\_view({'get': 'get\_profile\_data'})),

path('profiles/', ProfileView.as\_view({'get': 'get\_list\_profile'})),

path('mini-profiles/', ProfileView.as\_view({'get': 'get\_list\_mini\_profile'})),

path('profiles/<slug:path>/header/', ProfileView.as\_view({'get': 'get\_header\_profile'})),

# PROFILE to COURSE

path('profiles/<slug:path>/studying/courses/', CourseProfileView.as\_view({'get': 'get\_studying\_courses'})),

path('profiles/<slug:path>/studied/courses/', CourseProfileView.as\_view({'get': 'get\_studied\_courses'})),

path('profiles/<slug:path>/study-percent/', CourseProfileView.as\_view({'get': 'get\_statistic\_study\_courses'})),

# SUBSCRIPTION

path('profiles/<slug:path>/goals-subscription/',

SubscriptionProfileView.as\_view({'get': 'get\_goals\_subscription\_profile'})),

path('profiles/<slug:path>/subscribers/', SubscriptionProfileView.as\_view({'get': 'get\_subscribers\_profile'})),

path('profiles/<slug:path>/create/subscription/',

SubscriptionProfileView.as\_view({'post': 'create\_goal\_subscription'})),

path('profiles/<slug:path>/delete/subscription/',

SubscriptionProfileView.as\_view({'delete': 'delete\_goal\_subscription'})),

# path('profile/<slug:path>/', ProfileView.as\_view()),

# PROFILE UPDATE

path('profiles/<slug:path>/get/info/', ActionProfileView.as\_view({'get': 'get\_info'})),

path('profiles/<slug:path>/update/info/', ActionProfileView.as\_view({'put': 'update\_info'})),

path('profiles/<slug:path>/update/password/', ActionProfileView.as\_view({'put': 'update\_password'})),

]

Приложение 1. Листинг 13. views\_profile.py

from django\_filters.rest\_framework import DjangoFilterBackend  
from rest\_framework import permissions, viewsets, status  
from rest\_framework.decorators import action  
from rest\_framework.filters import SearchFilter, OrderingFilter  
from rest\_framework.response import Response  
  
from .models\_profile import Profile, Subscription  
from .serializers\_profile import ProfileSerializer, MiniProfileSerializer, HeaderProfileSerializer, \  
 ActionProfileSerializer, ActionUserSerializer, ActionUserPasswordSerializer  
from ..course.models\_course import ProfileCourse, ProfileCourseStatus  
from ..course.serializers\_course import MiniCourseSerializer  
from ..utils import Util, HelperFilter, HelperPaginatorValue, HelperPaginator  
  
  
class ProfileView(viewsets.ModelViewSet):  
 *"""View. Геттеры на модель Profile"""* lookup\_field = 'slug'  
 queryset = Profile.objects.all()  
 permission\_classes = [permissions.IsAuthenticated]  
 serializer\_class = ProfileSerializer  
  
 filter\_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)  
 filter\_fields = HelperFilter.PROFILE\_FILTER\_FIELDS  
 search\_fields = HelperFilter.PROFILE\_SEARCH\_FIELDS  
 ordering\_fields = HelperFilter.PROFILE\_ORDERING\_FIELDS  
  
 pagination\_max\_page = HelperPaginatorValue.PROFILE\_MAX\_PAGE  
  
 def exists\_path(self, path):  
 *"""Существует ли такой путь к странице"""* return len(self.queryset.filter(path=path)) != 0  
  
 def get\_frame\_pagination(self, request, queryset, max\_page=None):  
 *"""GET. Вернет форму для пагинации"""* if max\_page is None:  
 max\_page = self.pagination\_max\_page  
 pagination = HelperPaginator(request=request, queryset=queryset, max\_page=max\_page)  
 return {  
 "count": pagination.get\_count(),  
 "pages": pagination.get\_num\_pages(),  
 "next": pagination.get\_link\_next\_page(),  
 "previous": pagination.get\_link\_previous\_page(),  
 "results": pagination.page\_obj  
 }  
  
 @action(methods=['get'], detail=False)  
 def get\_profile\_data(self, request):  
 *"""GET. Вернёт страницу профиля"""* auth = Profile.objects.get(user=self.request.user)  
 return Response(ProfileSerializer(auth).data, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_list\_profile(self, request):  
 *"""GET. Вернёт все профили"""* auth = Profile.objects.get(user=self.request.user)  
  
 queryset = self.filter\_queryset(self.queryset)  
 frame\_pagination = self.get\_frame\_pagination(request, queryset)  
 serializer = ProfileSerializer(frame\_pagination.get('results'), many=True, context={'auth': auth})  
 frame\_pagination['results'] = serializer.data  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_list\_mini\_profile(self, request):  
 *"""GET. Вернёт все профили в мини-формах"""* queryset = self.filter\_queryset(self.queryset)  
 frame\_pagination = self.get\_frame\_pagination(request, queryset,  
 max\_page=HelperPaginatorValue.MINI\_PROFILE\_MAX\_PAGE)  
 serializer = MiniProfileSerializer(frame\_pagination.get('results'), many=True)  
 frame\_pagination['results'] = serializer.data  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_header\_profile(self, request, path):  
 *"""GET. Верхняя информация на странице любого пользователя"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 auth = Profile.objects.get(user=self.request.user)  
 profile = Profile.objects.get(path=path)  
 return Response(HeaderProfileSerializer(profile, context={'auth': auth}).data, status=status.HTTP\_200\_OK)  
  
  
class ActionProfileView(viewsets.ModelViewSet):  
 *"""View. Действия над моделью профиля"""* lookup\_field = 'slug'  
 queryset = Profile.objects.all()  
 permission\_classes = [permissions.IsAuthenticated]  
 serializer\_class = ProfileSerializer  
  
 @staticmethod  
 def get\_profile\_or\_error(path):  
 *"""GET. Вернёт профиль или ошибку"""* profile\_list = Profile.objects.filter(path=path)  
 if len(profile\_list) == 0:  
 error\_text = "Такого пользователя не существует"  
 return {'error': Response({'error': error\_text}, status=status.HTTP\_404\_NOT\_FOUND), }  
 return {'profile': profile\_list[0]}  
  
 def is\_valid(self, request, path):  
 *"""Проверка на валидность"""* profile\_dict = self.get\_profile\_or\_error(path=path)  
 if profile\_dict.get('error', None) is not None:  
 return profile\_dict.get('error', None)  
  
 profile = profile\_dict.get('profile')  
 auth = Profile.objects.get(user=self.request.user)  
 if profile != auth:  
 error\_text = "У вас нет доступа для изменения данных от имени этого пользователя"  
 return Response({'error': error\_text}, status=status.HTTP\_200\_OK)  
 return {'profile': auth}  
  
 @action(methods=['get'], detail=False)  
 def get\_info(self, request, path):  
 *"""GET. Вернет информацию по пользователю для обновления"""* profile\_dict = self.get\_profile\_or\_error(path=path)  
 if profile\_dict.get('error', None) is not None:  
 return profile\_dict.get('error', None)  
  
 profile = profile\_dict.get('profile')  
 serializer = ActionProfileSerializer(profile)  
 return Response(serializer.data, status=status.HTTP\_200\_OK)  
  
 @staticmethod  
 def update\_profile(data, profile):  
 *"""Обновление профиля"""* serializer\_profile = ActionProfileSerializer(data=data, instance=profile)  
 serializer\_profile.is\_valid(raise\_exception=True)  
 try:  
 serializer\_profile.save()  
 except ValueError as ex:  
 return {'error': Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)}  
 return {'serializer': serializer\_profile.data}  
  
 @staticmethod  
 def update\_user(data, user):  
 *"""Обновление пользователя"""* serializer\_user = ActionUserSerializer(data=data, instance=user)  
 serializer\_user.is\_valid(raise\_exception=True)  
 serializer\_user.save()  
 return serializer\_user.data  
  
 @action(methods=['put'], detail=False)  
 def update\_info(self, request, path):  
 *"""PUT. Обновление информации о пользователе"""* profile\_dict = self.is\_valid(request=request, path=path)  
 if profile\_dict.get('error', None) is not None:  
 return profile\_dict.get('error', None)  
 profile = profile\_dict.get('profile')  
  
 result\_profile = self.update\_profile(data=request.data, profile=profile)  
 if result\_profile.get('error', None) is not None:  
 return result\_profile.get('error')  
 serializer\_profile = result\_profile.get('serializer')  
  
 serializer\_user = self.update\_user(data=request.data, user=profile.user)  
 return Response({  
 'username': serializer\_user.get('username'),  
 'email': serializer\_user.get('email'),  
 'avatar\_url': serializer\_profile.get('avatar\_url'),  
 'path': serializer\_profile.get('path'),  
 }, status=status.HTTP\_200\_OK)  
  
 @action(methods=['put'], detail=False)  
 def update\_password(self, request, path):  
 *"""PUT. Обновление пароля"""* profile\_dict = self.is\_valid(request=request, path=path)  
 if profile\_dict.get('error', None) is not None:  
 return profile\_dict.get('error', None)  
 profile = profile\_dict.get('profile')  
  
 context = {'new\_password': request.data.get('new\_password')}  
 serializer\_user = ActionUserPasswordSerializer(data=request.data, instance=profile.user, context=context)  
 serializer\_user.is\_valid(raise\_exception=True)  
 serializer\_user.save()  
 return Response({'message': "password успешно обновлен"}, status=status.HTTP\_200\_OK)  
  
  
class CourseProfileView(viewsets.ModelViewSet):  
 *"""View. Связь профиля к курсу"""* lookup\_field = 'slug'  
 queryset = Profile.objects.all()  
 permission\_classes = [permissions.IsAuthenticated]  
 serializer\_class = ProfileSerializer  
  
 filter\_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)  
 filter\_fields = HelperFilter.COURSE\_FILTER\_FIELDS  
 search\_fields = HelperFilter.COURSE\_SEARCH\_FIELDS  
 ordering\_fields = HelperFilter.COURSE\_ORDERING\_FIELDS  
  
 pagination\_max\_page = HelperPaginatorValue.PROFILE\_MAX\_PAGE  
  
 def exists\_path(self, path):  
 *"""Существует ли такой путь"""* return len(self.queryset.filter(path=path)) != 0  
  
 def swap\_filters\_field(self, type\_filter):  
 *"""Смена типов фильтрации"""* (self.filter\_fields, self.search\_fields, self.ordering\_fields) = HelperFilter.get\_filters\_course\_field(  
 type\_filter)  
  
 def get\_frame\_pagination(self, request, queryset, max\_page=None):  
 *"""Вернет каркас для пагинации"""* if max\_page is None:  
 max\_page = self.pagination\_max\_page  
 pagination = HelperPaginator(request=request, queryset=queryset, max\_page=max\_page)  
 return {  
 "count": pagination.get\_count(),  
 "pages": pagination.get\_num\_pages(),  
 "next": pagination.get\_link\_next\_page(),  
 "previous": pagination.get\_link\_previous\_page(),  
 "results": pagination.page\_obj  
 }  
  
 @action(methods=['get'], detail=False)  
 def get\_studying\_courses(self, request, path):  
 *"""GET. Какие курсы ИЗУЧАЕТ студент"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 auth = Profile.objects.get(user=self.request.user)  
 profile = Profile.objects.get(path=path)  
  
 self.swap\_filters\_field(HelperFilter.PROFILE\_COURSE\_TYPE)  
 status\_studying = ProfileCourseStatus.objects.filter(name=Util.PROFILE\_COURSE\_STATUS\_STUDYING\_NAME)  
 profile\_course\_list = ProfileCourse.objects.filter(profile=profile, status\_\_in=status\_studying)  
 queryset = self.filter\_queryset(profile\_course\_list)  
 self.swap\_filters\_field(HelperFilter.COURSE\_TYPE)  
  
 frame\_pagination = self.get\_frame\_pagination(request, queryset,  
 max\_page=HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)  
 serializer\_list = list()  
 for profile\_course in frame\_pagination.get('results'):  
 serializer\_list.append(  
 MiniCourseSerializer(profile\_course.course, context={'profile': profile, 'auth': auth}).data)  
 frame\_pagination['results'] = serializer\_list  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_studied\_courses(self, request, path):  
 *"""GET. Какие курсы ИЗУЧИЛ студент"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 auth = Profile.objects.get(user=self.request.user)  
 profile = Profile.objects.get(path=path)  
  
 self.swap\_filters\_field(HelperFilter.PROFILE\_COURSE\_TYPE)  
 status\_studied = ProfileCourseStatus.objects.filter(name=Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME)  
 profile\_course\_list = ProfileCourse.objects.filter(profile=profile, status\_\_in=status\_studied)  
 queryset = self.filter\_queryset(profile\_course\_list)  
 self.swap\_filters\_field(HelperFilter.COURSE\_TYPE)  
  
 frame\_pagination = self.get\_frame\_pagination(request, queryset,  
 max\_page=HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)  
  
 serializer\_list = list()  
 for profile\_course in frame\_pagination.get('results'):  
 serializer\_list.append(  
 MiniCourseSerializer(profile\_course.course, context={'profile': profile, 'auth': auth}).data)  
 frame\_pagination['results'] = serializer\_list  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_statistic\_study\_courses(self, request, path):  
 *"""GET. Статистика по студенту по изученным курсам"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 profile = Profile.objects.get(path=path)  
 profile\_course\_list = ProfileCourse.objects.filter(profile=profile)  
  
 studying\_quantity = 0  
 studied\_quantity = 0  
 for profile\_course in profile\_course\_list:  
 if profile\_course.status is None:  
 continue  
 if profile\_course.status.name == Util.PROFILE\_COURSE\_STATUS\_STUDYING\_NAME:  
 studying\_quantity += 1  
 elif profile\_course.status.name == Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME:  
 studied\_quantity += 1  
  
 percent = 0  
 if studied\_quantity != 0:  
 percent = studied\_quantity / (studying\_quantity + studied\_quantity) \* 100  
  
 return Response({  
 'studying\_quantity': studying\_quantity,  
 'studied\_quantity': studied\_quantity,  
 'percent': percent,  
 }, status=status.HTTP\_200\_OK)  
  
  
class SubscriptionProfileView(viewsets.ModelViewSet):  
 *"""VIEW. Подписчики к профилю"""* lookup\_field = 'slug'  
 profiles = Profile.objects.all()  
 queryset = Subscription.objects.all()  
 permission\_classes = [permissions.IsAuthenticated]  
 serializer\_class = ProfileSerializer  
  
 filter\_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)  
 filter\_fields = HelperFilter.SUBSCRIBER\_FILTER\_FIELDS  
 search\_fields = HelperFilter.SUBSCRIBER\_SEARCH\_FIELDS  
 ordering\_fields = HelperFilter.SUBSCRIBER\_ORDERING\_FIELDS  
  
 pagination\_max\_page = HelperPaginatorValue.PROFILE\_MAX\_PAGE  
  
 def exists\_path(self, path):  
 *"""Существует ли такой путь"""* return len(self.profiles.filter(path=path)) != 0  
  
 def is\_subscribe(self, goal, subscriber):  
 *"""Подписан ли пользователь"""* return len(Subscription.objects.filter(goal=goal, subscriber=subscriber)) != 0  
  
 def swap\_filters\_field(self, type\_filter):  
 *"""Смена типов фильтраций"""* (self.filter\_fields, self.search\_fields, self.ordering\_fields) = HelperFilter.get\_filters\_subscription\_field(  
 type\_filter)  
  
 def get\_frame\_pagination(self, request, queryset, max\_page=None):  
 *"""Вернет каркас для пагинации"""* if max\_page is None:  
 max\_page = self.pagination\_max\_page  
 pagination = HelperPaginator(request=request, queryset=queryset, max\_page=max\_page)  
 return {  
 "count": pagination.get\_count(),  
 "pages": pagination.get\_num\_pages(),  
 "next": pagination.get\_link\_next\_page(),  
 "previous": pagination.get\_link\_previous\_page(),  
 "results": pagination.page\_obj  
 }  
  
 @action(methods=['get'], detail=False)  
 def get\_goals\_subscription\_profile(self, request, path):  
 *"""GET. На кого подписан"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 auth = Profile.objects.get(user=self.request.user)  
 profile = Profile.objects.get(path=path)  
 queryset = self.filter\_queryset(self.queryset.filter(subscriber=profile))  
  
 frame\_pagination = self.get\_frame\_pagination(request, queryset)  
 serializer\_list = list()  
 for subscription in frame\_pagination.get('results'):  
 serializer\_list.append(ProfileSerializer(subscription.goal, context={'auth': auth}).data)  
 frame\_pagination['results'] = serializer\_list  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(methods=['get'], detail=False)  
 def get\_subscribers\_profile(self, request, path):  
 *"""GET. Кто подписан на профиль"""* if not self.exists\_path(path):  
 return Response({'path': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 auth = Profile.objects.get(user=self.request.user)  
 profile = Profile.objects.get(path=path)  
 self.swap\_filters\_field(HelperFilter.GOAL\_TYPE)  
 queryset = self.filter\_queryset(self.queryset.filter(goal=profile))  
 self.swap\_filters\_field(HelperFilter.SUBSCRIBER\_TYPE)  
  
 frame\_pagination = self.get\_frame\_pagination(request, queryset)  
 serializer\_list = list()  
 for subscriber in frame\_pagination.get('results'):  
 serializer\_list.append(ProfileSerializer(subscriber.subscriber, context={'auth': auth}).data)  
 frame\_pagination['results'] = serializer\_list  
 return Response(frame\_pagination, status=status.HTTP\_200\_OK)  
  
 @action(detail=False, methods=['post'])  
 def create\_goal\_subscription(self, request, path):  
 *"""POST. Подписаться на профиль"""* if not self.exists\_path(path):  
 return Response({'error': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 profile = self.profiles.get(path=path)  
 auth = self.profiles.get(user=self.request.user)  
 if profile == auth:  
 return Response({'error': 'Вы не можете подписаться или отписаться от себя'},  
 status=status.HTTP\_400\_BAD\_REQUEST)  
  
 if self.is\_subscribe(goal=profile, subscriber=auth):  
 return Response({'error': 'Вы уже подписались на этого пользователя'}, status=status.HTTP\_400\_BAD\_REQUEST)  
  
 serializer = Subscription.objects.create(goal=profile, subscriber=auth)  
 serializer.save()  
 return Response({  
 'success': f'Успешно подписались на {profile.user.username}'  
 }, status=status.HTTP\_200\_OK)  
  
 @action(detail=False, methods=['delete'])  
 def delete\_goal\_subscription(self, request, path):  
 *"""DELETE. Удалить подписку на профиль"""* if not self.exists\_path(path):  
 return Response({'error': "Пути к такому пользователю не существует"}, status=status.HTTP\_404\_NOT\_FOUND)  
  
 profile = self.profiles.get(path=path)  
 auth = self.profiles.get(user=self.request.user)  
 if profile == auth:  
 return Response({'error': 'Вы не можете подписаться или отписаться от себя'},  
 status=status.HTTP\_400\_BAD\_REQUEST)  
  
 if not self.is\_subscribe(goal=profile, subscriber=auth):  
 return Response({'error': 'Вы уже отписались на этого пользователя'}, status=status.HTTP\_400\_BAD\_REQUEST)  
  
 serializer = Subscription.objects.get(goal=profile, subscriber=auth)  
 serializer.delete()  
 return Response({  
 'success': f'Успешно отписались от {profile.user.username}'  
 }, status=status.HTTP\_200\_OK)

Приложение 1. Листинг 14. serializers\_profile.py

from ..models import User

from rest\_framework import serializers

from .models\_profile import Profile, Subscription

from ..auth.serializers\_auth import RegisterSerializer

from ..utils import Util

class HelperSerializer(serializers.ModelSerializer):

"""Serializer. Помощник сериализаций"""

@staticmethod

def is\_subscribed(goal, subscriber):

"""

goal: на кого подписались

subscriber: кто подписался

:return: True, если subscriber подписан на goal. False если нет

"""

subscription\_list = Subscription.objects.filter(goal=goal, subscriber=subscriber)

if len(subscription\_list):

return True

return False

class ProfileSerializer(serializers.ModelSerializer):

"""Serializer. Профиль"""

username = serializers.SerializerMethodField()

is\_subscribed = serializers.SerializerMethodField()

class Meta:

model = Profile

fields = ('path', 'username', 'avatar\_url', 'description', 'is\_subscribed')

def get\_username(self, profile):

"""Вернет имя профиля"""

return profile.user.username

def get\_is\_subscribed(self, profile):

"""Вернет подписан ли пользователь на профиль"""

auth = self.context.get('auth')

if auth != profile:

return HelperSerializer.is\_subscribed(goal=profile, subscriber=auth)

return None

class MiniProfileSerializer(serializers.ModelSerializer):

"""Serializer. Форма мини профиля"""

username = serializers.SerializerMethodField()

class Meta:

model = Profile

fields = ('path', 'username', 'avatar\_url')

@staticmethod

def get\_username(profile):

"""Вернет имя пользователя"""

return profile.user.username

class HeaderProfileSerializer(serializers.ModelSerializer):

"""Serializer. Шапка профиля"""

username = serializers.SerializerMethodField()

communications = serializers.SerializerMethodField(default=None)

is\_subscribed = serializers.SerializerMethodField()

class Meta:

model = Profile

fields = ('username', 'avatar\_url', 'wrapper\_url', 'communications', 'is\_subscribed')

@staticmethod

def get\_username(profile):

"""Вернет имя пользователя"""

return profile.user.username

@staticmethod

def get\_communications(profile):

"""Вернет количество подписчиков и подписок"""

return {

'goal\_quantity': len(Subscription.objects.filter(subscriber=profile)),

'subscribers\_quantity': len(Subscription.objects.filter(goal=profile)),

}

def get\_is\_subscribed(self, profile):

"""Подписаны ли вы на этот профиль"""

auth = self.context.get('auth', None)

if (auth is not None) and (auth != profile):

return HelperSerializer.is\_subscribed(goal=profile, subscriber=auth)

return None

class ActionProfileSerializer(serializers.ModelSerializer):

"""Serializer. Действия над профилем"""

class Meta:

model = Profile

fields = ('avatar\_url', 'path')

def update(self, instance, validated\_data):

"""Обновление данных"""

new\_path = validated\_data.get('path', -1)

if new\_path != -1:

instance.path = Util.get\_new\_path(new\_path=new\_path, old\_path=instance.path, model=Profile)

new\_image = validated\_data.get('avatar\_url', -1)

if new\_image != -1:

default\_image = Util.DEFAULT\_IMAGES.get('profile')

update\_image = Util.get\_image(old=instance.avatar\_url, new=new\_image, default=default\_image)

instance.avatar\_url = Util.get\_update\_image(old=instance.image\_url, new=update\_image)

instance.save()

return instance

class ActionUserSerializer(serializers.ModelSerializer):

"""Serializer. Действия над пользователем"""

class Meta:

model = User

fields = ('username', 'email')

def update(self, instance, validated\_data):

"""Обновление"""

instance.username = validated\_data.get('username', instance.username)

instance.email = validated\_data.get('email', instance.email)

instance.save()

return instance

class ActionUserPasswordSerializer(serializers.ModelSerializer):

"""Serializer. Действия над паролем пользователя"""

new\_password = serializers.SerializerMethodField()

class Meta:

model = User

fields = ('password', 'new\_password')

extra\_kwargs = {

'password': {'write\_only': True},

'new\_password': {'write\_only': True}

}

def update(self, instance, validated\_data):

"""Обновление"""

password = validated\_data.get('password')

new\_password = self.context.get('new\_password')

if instance.check\_password(password):

if not RegisterSerializer.password\_is\_valid(new\_password):

raise serializers.ValidationError({"error": "Новый пароль не корректен"})

instance.set\_password(new\_password)

instance.save()

return instance

else:

raise serializers.ValidationError({"error": "Не правильный password"})

class UserSerializer(serializers.ModelSerializer):

"""Serializer. Пользователь"""

class Meta:

model = User

fields = ('id', 'username', 'email', 'password')

class ProfileAsAuthor(serializers.ModelSerializer):

"""Serializer. Профиль как автор"""

username = serializers.SerializerMethodField()

class Meta:

model = Profile

fields = ('path', 'username', 'avatar\_url')

@staticmethod

def get\_username(profile):

"""Вернет имя пользователя"""

return profile.user.username

Приложение 1. Листинг 15. models\_collection.py

import datetime

from django.core.validators import validate\_image\_file\_extension

from django.db import models

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

from ..profile.models\_profile import Profile

from ..utils import Util

# ########### COLLECTION START ##############

class Collection(models.Model):

"""Collection"""

title = models.CharField(max\_length=40)

description = models.TextField(max\_length=512, blank=True)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

image\_url = models.ImageField(validators=[validate\_image\_file\_extension], blank=True, null=True,

default=Util.DEFAULT\_IMAGES.get('collection'))

wallpaper = models.ImageField(blank=True, null=True)

rating = models.FloatField(default=0)

members\_amount = models.IntegerField(default=0)

date\_create = models.DateField(default=datetime.date.today)

path = models.CharField(max\_length=64, blank=True, unique=True)

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

return f'{self.profile.user.username}: {self.title} [Collection]'

def create\_collection(sender, \*\*kwargs):

"""When a Collection is created, autofill fields"""

if kwargs['created']:

collection = kwargs['instance']

collection.path = collection.pk

collection.save()

profile\_collection = ProfileCollection.objects.create(collection=collection, profile=collection.profile)

profile\_collection.save()

collection\_stars = CollectionStars.objects.create(collection=collection)

collection\_stars.save()

post\_save.connect(create\_collection, sender=Collection)

class ProfileCollection(models.Model):

"""ProfileCollection"""

collection = models.ForeignKey(Collection, on\_delete=models.CASCADE)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

grade = models.IntegerField(blank=True, null=True)

date\_added = models.DateField(default=datetime.date.today, blank=True, null=True)

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

return f"\"{self.profile.user.username}\" to \"{self.collection.title}\" [Profile to Collection]"

class CollectionStars(models.Model):

"""CollectionStars"""

collection = models.OneToOneField(Collection, on\_delete=models.CASCADE)

one\_stars\_count = models.IntegerField(default=0)

two\_stars\_count = models.IntegerField(default=0)

three\_stars\_count = models.IntegerField(default=0)

four\_stars\_count = models.IntegerField(default=0)

five\_stars\_count = models.IntegerField(default=0)

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

return f"{self.collection.profile.user.username}: {self.collection.title} [CollectionStars]"

# ############ COLLECTION END ###############

Приложение 1. Листинг 16. urls\_collection.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views\_collection import CollectionView, GradeCollectionView, ActionCollectionView, ActionProfileCollectionView

router = DefaultRouter()

urlpatterns = [

path('', include(router.urls)),

path('collections/', CollectionView.as\_view({'get': 'get\_collections'})),

path('mini-collections/', CollectionView.as\_view({'get': 'get\_mini\_collections'}), name="mini-collections"),

path('collections/catalog/', CollectionView.as\_view({'get': 'get\_catalog\_collections'})),

path('collections/all/<slug:path>/', CollectionView.as\_view({'get': 'get\_all\_profile\_collections'})),

path('collections/added/<slug:path>/', CollectionView.as\_view({'get': 'get\_added\_collections'})),

path('collections/created/<slug:path>/', CollectionView.as\_view({'get': 'get\_created\_collections'})),

path('collections/get/<slug:path>/', CollectionView.as\_view({'get': 'get\_detail\_collection'})),

path('collections/create/grade/<slug:path>/', GradeCollectionView.as\_view({'post': 'set\_grade'})),

path('collections/update/grade/<slug:path>/', GradeCollectionView.as\_view({'put': 'update\_grade'})),

path('collections/delete/grade/<slug:path>/', GradeCollectionView.as\_view({'delete': 'delete\_grade'})),

path('collections/create/', ActionCollectionView.as\_view({'post': 'create\_collection'})),

path('collections/get-update/<slug:path>/', ActionCollectionView.as\_view({'get': 'get\_update\_info'})),

path('collections/update/<slug:path>/', ActionCollectionView.as\_view({'put': 'update\_info'})),

path('collections/delete/<slug:path>/', ActionCollectionView.as\_view({'delete': 'delete\_collection'})),

path('collections/add/<slug:path>/', ActionProfileCollectionView.as\_view({'post': 'added\_collections'})),

path('collections/pop/<slug:path>/', ActionProfileCollectionView.as\_view({'delete': 'popped\_collections'})),

]

Приложение 1. Листинг 17. views\_collection.py

from django\_filters.rest\_framework import DjangoFilterBackend

from rest\_framework import permissions, status, viewsets

from rest\_framework.decorators import action

from rest\_framework.filters import SearchFilter, OrderingFilter

from rest\_framework.response import Response

from .models\_collection import Profile, Collection, ProfileCollection

from .serializers\_collection import DetailCollectionSerializer, CollectionSerializer, WindowDetailCollectionSerializer, \

GradeCollectionSerializer, MiniCollectionSerializer

from ..utils import HelperFilter, HelperPaginatorValue, HelperPaginator

# #########################################

# ######## GET ########

# #########################################

class CollectionView(viewsets.ModelViewSet):

"""VIEW. Коллекция"""

lookup\_field = 'slug'

queryset = Collection.objects.all()

serializer\_class = CollectionSerializer

permission\_classes = [permissions.IsAuthenticated]

filter\_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)

filter\_fields = HelperFilter.COLLECTION\_FILTER\_FIELDS

search\_fields = HelperFilter.COLLECTION\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.COLLECTION\_ORDERING\_FIELDS

pagination\_max\_page = HelperPaginatorValue.COLLECTION\_MAX\_PAGE

def exists\_course\_path(self, path):

"""Существует ли такой путь к курсу"""

return len(self.queryset.filter(path=path)) != 0

@staticmethod

def exists\_profile\_path(path):

"""Существует ли такой путь к профилю"""

return len(Profile.objects.filter(path=path)) != 0

def swap\_filters\_field(self, type\_filter):

"""Смена типа фильтраций"""

(self.filter\_fields, self.search\_fields, self.ordering\_fields) = HelperFilter.get\_filters\_collection\_field(type\_filter)

def get\_frame\_pagination(self, request, queryset, max\_page=None):

"""Вернет каркас пагинации"""

if max\_page is None:

max\_page = self.pagination\_max\_page

pagination = HelperPaginator(request=request, queryset=queryset, max\_page=max\_page)

return {

"count": pagination.get\_count(),

"pages": pagination.get\_num\_pages(),

"next": pagination.get\_link\_next\_page(),

"previous": pagination.get\_link\_previous\_page(),

"results": pagination.page\_obj

}

@action(detail=False, methods=['get'])

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

"""GET. Вернет все подборки"""

queryset = self.filter\_queryset(self.queryset)

frame\_pagination = self.get\_frame\_pagination(request, queryset)

auth = Profile.objects.get(user=self.request.user)

serializer = CollectionSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

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

"""GET. Вернет все подборки в мини-формах"""

auth = Profile.objects.get(user=self.request.user)

queryset = self.filter\_queryset(self.queryset)

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COLLECTION\_MAX\_PAGE)

serializer = MiniCollectionSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@staticmethod

def get\_not\_profile\_collections(profile):

"""GET. Вернет все подборки, которые пользователь не добавлял себе и не создавал"""

data = dict()

for item in ProfileCollection.objects.all():

collection = item.collection

if data.get(collection, None) is None:

data[collection] = list()

data[collection].append(item.profile)

collection\_list = list()

for key, value in data.items():

if profile not in value:

collection\_list.append(key)

return ProfileCollection.objects.filter(collection\_\_in=collection\_list)

@action(detail=False, methods=['get'])

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

"""GET. Подборки которые неизвестны пользователю по path"""

auth = Profile.objects.get(user=self.request.user)

self.swap\_filters\_field(HelperFilter.PROFILE\_COLLECTION\_TYPE)

queryset = self.filter\_queryset(self.get\_not\_profile\_collections(profile=auth))

self.swap\_filters\_field(HelperFilter.COLLECTION\_TYPE)

frame\_pagination = self.get\_frame\_pagination(request, queryset)

serializer\_list = list()

for profile\_collection in frame\_pagination.get('results'):

serializer\_list.append(CollectionSerializer(profile\_collection.collection, context={'profile': auth}).data)

frame\_pagination['results'] = serializer\_list

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_all\_profile\_collections(self, request, path, \*args, \*\*kwargs):

"""GET. Добавленные и созданные подборки пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

profile = Profile.objects.get(path=path)

auth = Profile.objects.get(user=self.request.user)

self.swap\_filters\_field(HelperFilter.PROFILE\_COLLECTION\_TYPE)

queryset = self.filter\_queryset(ProfileCollection.objects.filter(profile=profile))

self.swap\_filters\_field(HelperFilter.COLLECTION\_TYPE)

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COLLECTION\_MAX\_PAGE)

serializer\_list = list()

for profile\_collection in frame\_pagination.get('results'):

serializer\_list.append(

MiniCollectionSerializer(profile\_collection.collection, context={'profile': auth}).data)

frame\_pagination['results'] = serializer\_list

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

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

"""GET. Добавленные подборки пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

profile = Profile.objects.get(path=path)

auth = Profile.objects.get(user=self.request.user)

self.swap\_filters\_field(HelperFilter.PROFILE\_COLLECTION\_TYPE)

added\_queryset = self.filter\_queryset(ProfileCollection.objects.filter(profile=profile))

self.swap\_filters\_field(HelperFilter.COLLECTION\_TYPE)

# Исключаем созданные подборки

queryset = list()

for item in added\_queryset:

if item.collection.profile != profile:

queryset.append(item.collection)

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COLLECTION\_MAX\_PAGE)

serializer = MiniCollectionSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

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

"""GET. Созданные подборки пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

profile = Profile.objects.get(path=path)

auth = Profile.objects.get(user=self.request.user)

queryset = self.filter\_queryset(self.queryset.filter(profile=profile))

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COLLECTION\_MAX\_PAGE)

serializer = MiniCollectionSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_detail\_collection(self, request, path=None, \*args, \*\*kwargs):

"""GET. Детальная страница подборки"""

if not self.exists\_course\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

auth = Profile.objects.get(user=request.user)

serializer = DetailCollectionSerializer(collection, context={'profile': auth})

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

# #########################################

# ######## ACTIONS ########

# #########################################

class ActionCollectionView(viewsets.ModelViewSet):

"""VIEW. Действия над подборками"""

lookup\_field = 'slug'

queryset = Collection.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_path(self, path):

"""Существует ли такой путь"""

return len(self.queryset.filter(path=path)) != 0

@action(detail=False, methods=['post'])

def create\_collection(self, request):

"""POST. Создание подборки"""

profile = Profile.objects.get(user=self.request.user)

number\_new\_collection = len(ProfileCollection.objects.filter(profile=profile)) + 1

serializer = WindowDetailCollectionSerializer(data={'title': f"Подборка #{number\_new\_collection}"},

context={'profile': profile})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'title': serializer.data['title'],

'path': serializer.data['path'],

'image\_url': serializer.data['image\_url'],

"message": "Подборка успешно создана",

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_update\_info(self, request, path=None, \*args, \*\*kwargs):

"""GET. Вернет информацию для обновления данных в подборке"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=request.user)

if collection.profile != profile:

return Response({"error": "У вас нет доступа для изменения коллекции"},

status=status.HTTP\_400\_BAD\_REQUEST)

return Response(WindowDetailCollectionSerializer(collection).data, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['put'])

def update\_info(self, request, path=None, \*args, \*\*kwargs):

"""PUT. Обновление информации о подборке"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=request.user)

if collection.profile != profile:

return Response({"error": "У вас нет доступа для изменения подборки от имени этого аккаунта"},

status=status.HTTP\_400\_BAD\_REQUEST)

serializer = WindowDetailCollectionSerializer(data=request.data, instance=collection)

serializer.is\_valid(raise\_exception=True)

try:

serializer.save()

except ValueError as ex:

return Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)

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

@action(detail=False, methods=['delete'])

def delete\_collection(self, request, path):

"""DELETE. Удаление подборки"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=request.user)

if collection.profile != profile:

return Response({"error": "У вас нет доступа для удаления подборки от имени этого аккаунта"},

status=status.HTTP\_400\_BAD\_REQUEST)

collection\_title = collection.title

collection.delete()

return Response({

"title": collection\_title,

"path": path,

"message": "Подборка успешно удалилась",

}, status=status.HTTP\_200\_OK)

# #########################################

# ######## ACTIONS PROFILE ########

# #########################################

class ActionProfileCollectionView(viewsets.ModelViewSet):

"""VIEW. Действия между профилем и подборками"""

lookup\_field = 'slug'

queryset = Collection.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_path(self, path):

"""Существует ли такой путь"""

return len(self.queryset.filter(path=path)) != 0

@action(detail=False, methods=['post'])

def added\_collections(self, request, path):

"""POST. Добавить подборку"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

profile = Profile.objects.get(user=self.request.user)

collection = Collection.objects.get(path=path)

profile\_collection\_list = ProfileCollection.objects.filter(profile=profile, collection=collection)

if len(profile\_collection\_list) != 0:

return Response({'error': "Вы уже добавили эту подборку"}, status=status.HTTP\_404\_NOT\_FOUND)

profile\_collection = ProfileCollection.objects.create(profile=profile, collection=collection)

profile\_collection.save()

return Response({

'collection': collection.title,

'message': "Подборка добавлена"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def popped\_collections(self, request, path):

"""DELETE. Удалить подборку из своей коллекции подборок"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

profile = Profile.objects.get(user=self.request.user)

collection = Collection.objects.get(path=path)

profile\_collection\_list = ProfileCollection.objects.filter(profile=profile, collection=collection)

if len(profile\_collection\_list) == 0:

return Response({'error': "Вы уже удалили эту подборку"}, status=status.HTTP\_404\_NOT\_FOUND)

profile\_collection = profile\_collection\_list[0]

profile\_collection.delete()

return Response({

'collection': collection.title,

'message': "Подборка удалена"

}, status=status.HTTP\_200\_OK)

# #########################################

# ######## GRADE ########

# #########################################

class GradeCollectionView(viewsets.ModelViewSet):

"""VIEW. Оценка подборки"""

lookup\_field = 'slug'

queryset = Collection.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_path(self, path):

"""Существует ли такой путь"""

return len(self.queryset.filter(path=path)) != 0

@action(detail=False, methods=['post'])

def set\_grade(self, request, path):

"""POST. Установить оценку"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

serializer = GradeCollectionSerializer(data=request.data,

context={'profile': profile, 'collection': collection})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'collection': collection.title,

'path': collection.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['put'])

def update\_grade(self, request, path):

"""PUT. Обновление оценки"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

profile\_collection = ProfileCollection.objects.get(profile=profile, collection=collection)

serializer = GradeCollectionSerializer(data=request.data, instance=profile\_collection)

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'collection': collection.title,

'path': collection.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def delete\_grade(self, request, path):

"""DELETE. Убрать оценку"""

if not self.exists\_path(path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

collection = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

if collection.profile != profile:

return Response({"error": "У вас нет доступа для удаления оценки у подборки от имени этого аккаунта"},

status=status.HTTP\_400\_BAD\_REQUEST)

profile\_collection = ProfileCollection.objects.get(profile=profile, collection=collection)

serializer = GradeCollectionSerializer(context={'profile\_collection': profile\_collection})

serializer.delete\_grade()

return Response({

'collection': collection.title,

'path': collection.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

Приложение 1. Листинг 18. serializers\_collection.py

from django.db.models import Q

from rest\_framework import serializers

from .models\_collection import Collection, ProfileCollection, CollectionStars

from ..course.models\_course import ProfileCourseCollection

from ..course.serializers\_course import MiniCourseSerializer

from ..profile.serializers\_profile import ProfileAsAuthor

#####################################

# ## COLLECTION ##

#####################################

from ..utils import Util

class HelperCollectionSerializer:

"""SERIALIZER. Помощник при сериализации"""

@staticmethod

def get\_is\_added(collection, profile):

profile\_to\_collection = ProfileCollection.objects.filter(collection=collection, profile=profile)

if len(profile\_to\_collection) == 1:

return True

return False

class CollectionSerializer(serializers.ModelSerializer):

"""

Collection.

Есть на странице каталога.

Содержит: Подборки; Курсы в этой подборке; Добавил ли себе пользователь эту подборку

"""

author = serializers.SerializerMethodField()

courses = serializers.SerializerMethodField()

is\_added = serializers.SerializerMethodField(default=False)

added\_number = serializers.SerializerMethodField()

class Meta:

model = Collection

fields = ('path', 'title', 'author', 'image\_url', 'rating', 'courses', 'is\_added', 'added\_number')

@staticmethod

def get\_author(collection):

"""Вернет автора подборки"""

return ProfileAsAuthor(collection.profile).data

def get\_courses(self, collection):

"""Вернет первые 5 курсов которые есть в подборке"""

courses\_to\_collection = ProfileCourseCollection.objects.filter(collection=collection)

courses = list()

for item in courses\_to\_collection:

if item.course.status.name == Util.COURSE\_STATUS\_RELEASE\_NAME:

courses.append(MiniCourseSerializer(item.course, context={'profile': self.context.get('profile')}).data)

courses = sorted(courses, key=lambda x: x['rating'])[:5]

return courses

def get\_is\_added(self, collection):

"""Добавлена ли эта подборка"""

return HelperCollectionSerializer.get\_is\_added(collection=collection, profile=self.context.get('profile'))

def get\_added\_number(self, collection):

"""Сколько людей добавило эту подборку"""

queryset = ProfileCollection.objects.filter(collection=collection, )

return len(queryset.filter(~Q(profile=self.context.get('profile'))))

class MiniCollectionSerializer(serializers.ModelSerializer):

"""

Item Mini Collection.

Подборка в малой форме с малым количеством информации.

"""

author = serializers.SerializerMethodField()

is\_added = serializers.SerializerMethodField(default=False)

class Meta:

model = Collection

fields = ('path', 'title', 'author', 'image\_url', 'is\_added')

@staticmethod

def get\_author(collection):

"""Вернет автора подборки"""

return collection.profile.user.username

def get\_is\_added(self, collection):

"""Добавлена ли подборка"""

return HelperCollectionSerializer.get\_is\_added(collection=collection, profile=self.context.get('profile'))

class DetailCollectionSerializer(serializers.ModelSerializer):

"""SERIALIZER. Дательная страница подборки"""

author = serializers.SerializerMethodField()

courses = serializers.SerializerMethodField()

is\_added = serializers.SerializerMethodField(default=False)

class Meta:

model = Collection

fields = ('title', 'author', 'description', 'wallpaper', 'image\_url', 'members\_amount', 'rating', 'courses',

'is\_added')

@staticmethod

def get\_author(collection):

"""Вернет автора"""

return ProfileAsAuthor(collection.profile).data

def get\_courses(self, collection):

"""Вернет все курсы"""

courses\_to\_collection = ProfileCourseCollection.objects.filter(collection=collection)

courses = list()

for item in courses\_to\_collection:

if item.course.status.name == Util.COURSE\_STATUS\_RELEASE\_NAME:

courses.append(MiniCourseSerializer(item.course, context={'profile': self.context.get('profile')}).data)

return courses

def get\_is\_added(self, collection):

"""Добавлена ли подборка"""

return HelperCollectionSerializer.get\_is\_added(collection=collection, profile=self.context.get('profile'))

class WindowDetailCollectionSerializer(serializers.ModelSerializer):

"""SERIALIZER. Окно изменения данных подборки"""

class Meta:

model = Collection

fields = ('title', 'description', 'wallpaper', 'image\_url', 'path')

def create(self, validated\_data):

"""Создание"""

return Collection.objects.create(\*\*validated\_data, profile=self.context.get('profile'))

def update(self, instance, validated\_data):

"""Обновление"""

instance.title = validated\_data.get('title', instance.title)

instance.description = validated\_data.get('description', instance.description)

new\_path = validated\_data.get('path', None)

if new\_path is not None:

new\_path = Util.get\_update\_path(new\_path=new\_path)

instance.path = Util.get\_new\_path(new\_path=new\_path, old\_path=instance.path, model=Collection)

# Изменение картинок

instance.wallpaper = Util.get\_update\_image(old=instance.image\_url,

new=validated\_data.get('wallpaper', instance.image\_url))

new\_image = validated\_data.get('image\_url', -1)

if new\_image != -1:

update\_image = Util.get\_image(old=instance.image\_url, new=new\_image,

default=Util.DEFAULT\_IMAGES.get('lesson'))

instance.image\_url = Util.get\_update\_image(old=instance.image\_url, new=update\_image)

instance.save()

return instance

#####################################

# #########################################

# ######## GRADE ########

# #########################################

class GradeCollectionSerializer(serializers.ModelSerializer):

"""SERIALIZER. Оценка подборки"""

class Meta:

model = ProfileCollection

fields = ('grade',)

@staticmethod

def add\_collection\_star(collection, grade):

"""Добавить оценку"""

stars = CollectionStars.objects.get(collection=collection)

if grade == 1:

stars.one\_stars\_count += 1

elif grade == 2:

stars.two\_stars\_count += 1

elif grade == 3:

stars.three\_stars\_count += 1

elif grade == 4:

stars.four\_stars\_count += 1

elif grade == 5:

stars.five\_stars\_count += 1

stars.save()

@staticmethod

def difference\_collection\_star(collection, grade):

"""Убрать оценку"""

stars = CollectionStars.objects.get(collection=collection)

if grade == 1:

stars.one\_stars\_count -= 1

elif grade == 2:

stars.two\_stars\_count -= 1

elif grade == 3:

stars.three\_stars\_count -= 1

elif grade == 4:

stars.four\_stars\_count -= 1

elif grade == 5:

stars.five\_stars\_count -= 1

stars.save()

@staticmethod

def update\_rating\_collection(collection):

"""Обновить оценку"""

stars = CollectionStars.objects.get(collection=collection)

sum\_grade = stars.one\_stars\_count + stars.two\_stars\_count \* 2 + stars.three\_stars\_count \* 3 \

+ stars.four\_stars\_count \* 4 + stars.five\_stars\_count \* 5

count = stars.one\_stars\_count + stars.two\_stars\_count + stars.three\_stars\_count + stars.four\_stars\_count \

+ stars.five\_stars\_count

rating = 0

if count != 0:

rating = sum\_grade / count

collection.rating = rating

collection.save()

def create(self, validated\_data):

"""Создание оценки"""

profile\_collection = ProfileCollection.objects.get(profile=self.context.get('profile'),

collection=self.context.get('collection'))

if profile\_collection.grade is not None:

raise serializers.ValidationError({'grade': 'Вы уже оценивали подборку'})

profile\_collection.grade = validated\_data['grade']

self.add\_collection\_star(collection=profile\_collection.collection, grade=profile\_collection.grade)

self.update\_rating\_collection(collection=profile\_collection.collection)

profile\_collection.save()

return profile\_collection

def update(self, instance, validated\_data):

"""Обновление оценки"""

grade\_list = (1, 2, 3, 4, 5)

new\_grade = validated\_data.get('grade')

if (type(new\_grade) == int) and (new\_grade in grade\_list) and (instance.grade != new\_grade):

self.difference\_collection\_star(collection=instance.collection, grade=instance.grade)

instance.grade = validated\_data.get('grade')

instance.save()

self.add\_collection\_star(collection=instance.collection, grade=instance.grade)

self.update\_rating\_collection(collection=instance.collection)

return instance

def delete\_grade(self):

"""Удаление оценки"""

profile\_collection = self.context.get('profile\_collection')

if profile\_collection.grade is not None:

self.difference\_collection\_star(collection=profile\_collection.collection, grade=profile\_collection.grade)

self.update\_rating\_collection(collection=profile\_collection.collection)

profile\_collection.grade = None

profile\_collection.save()

return profile\_collection

Приложение 1. Листинг 19. models\_course.py

import datetime

from ckeditor\_uploader.fields import RichTextUploadingField

from django.db import models

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

from ..collection.models\_collection import Collection

from ..profile.models\_profile import Profile

# ############## COURSE START ###############

from ..utils import Util

class CourseStatus(models.Model):

"""Status Course: Dev, Release"""

name = models.CharField(max\_length=64)

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

return f'{self.name}'

class Course(models.Model):

"""Course"""

title = models.CharField(max\_length=64)

description = models.TextField(max\_length=175, blank=True)

price = models.IntegerField(default=0)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

image\_url = models.ImageField(blank=True, null=True, default=Util.DEFAULT\_IMAGES.get('course'))

duration\_in\_minutes = models.IntegerField(default=0)

rating = models.FloatField(default=0)

members\_amount = models.IntegerField(default=0)

max\_progress = models.IntegerField(default=0)

progress = models.IntegerField(default=0)

status = models.ForeignKey(CourseStatus, blank=True, null=True, on\_delete=models.SET\_NULL)

date\_create = models.DateField(default=datetime.date.today)

path = models.CharField(max\_length=64, blank=True, unique=True)

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

return f'{self.profile.user.username}: {self.title} [Course]'

def create\_course(sender, \*\*kwargs):

"""When a course is created, autofill fields"""

if kwargs['created']:

course = kwargs['instance']

course.path = course.pk

course\_status = CourseStatus.objects.filter(name=Util.COURSE\_STATUS\_DEV\_NAME)[0]

course.status = course\_status

course.save()

creator\_collection = CreatorCollection.objects.create(course=course, creator=course.profile)

creator\_collection.save()

course\_info = CourseInfo.objects.create(course=course)

course\_info.save()

course\_main\_info = CourseMainInfo.objects.create(course\_info=course\_info)

course\_main\_info.save()

course\_stars = CourseStars.objects.create(course=course)

course\_stars.save()

post\_save.connect(create\_course, sender=Course)

class CreatorCollection(models.Model):

"""CreatorCollection"""

creator = models.ForeignKey(Profile, on\_delete=models.CASCADE)

course = models.ForeignKey(Course, on\_delete=models.CASCADE)

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

return f"{self.creator}: \"{self.course.title}\""

# -------------- Page Course START ---------------

class CourseInfo(models.Model):

"""CourseInfo"""

course = models.OneToOneField(Course, on\_delete=models.CASCADE)

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

return f"{self.course.profile.user.username}: {self.course.title} [Info]"

class CourseMainInfo(models.Model):

"""CourseMainInfo"""

course\_info = models.OneToOneField(CourseInfo, on\_delete=models.CASCADE)

title\_image\_url = models.ImageField(blank=True, null=True)

goal\_description = models.TextField(blank=True, null=True)

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

return f"{self.course\_info.course.profile.user.username}: {self.course\_info.course.title} [MainInfo]"

class CourseFit(models.Model):

"""CourseFit"""

course\_info = models.ForeignKey(CourseInfo, on\_delete=models.CASCADE)

title = models.CharField(max\_length=32, blank=True, null=True)

description = models.TextField(max\_length=256, blank=True, null=True)

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

return f"{self.course\_info.course.profile.user.username}: {self.course\_info.course.title}: {self.title} [Fit]"

class CourseStars(models.Model):

"""CourseStars"""

course = models.OneToOneField(Course, on\_delete=models.CASCADE)

one\_stars\_count = models.IntegerField(default=0)

two\_stars\_count = models.IntegerField(default=0)

three\_stars\_count = models.IntegerField(default=0)

four\_stars\_count = models.IntegerField(default=0)

five\_stars\_count = models.IntegerField(default=0)

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

return f"{self.course.profile.user.username}: {self.course.title} [Course Stars]"

class CourseSkill(models.Model):

"""CourseSkill"""

course\_info = models.ForeignKey(CourseInfo, on\_delete=models.CASCADE)

name = models.CharField(max\_length=64)

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

return f"{self.course\_info.course.profile.user.username}: {self.course\_info.course.title}: {self.name} [Skill]"

# -------------- Page Course END -----------------

# ------------ Content Course START --------------

class Theme(models.Model):

"""The Course consists of Theme"""

course = models.ForeignKey(Course, on\_delete=models.CASCADE)

title = models.CharField(max\_length=64)

image\_url = models.ImageField(blank=True, null=True, default=Util.DEFAULT\_IMAGES.get('theme'))

max\_progress = models.IntegerField(default=0)

progress = models.IntegerField(default=0)

path = models.CharField(max\_length=64, blank=True, null=True)

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

return f"{self.course.profile.user.username}: {self.course.title} => {self.title} [Theme]"

def create\_theme(sender, \*\*kwargs):

"""When a course is created, autofill fields"""

if kwargs['created']:

theme = kwargs['instance']

theme.path = theme.pk

theme.save()

post\_save.connect(create\_theme, sender=Theme)

class Lesson(models.Model):

"""The Theme consists of Lesson"""

theme = models.ForeignKey(Theme, on\_delete=models.CASCADE)

title = models.CharField(max\_length=64)

image\_url = models.ImageField(blank=True, null=True, default=Util.DEFAULT\_IMAGES.get('lesson'))

max\_progress = models.IntegerField(default=0)

progress = models.IntegerField(default=0)

path = models.CharField(max\_length=64, blank=True, null=True)

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

return f"{self.theme.course.profile.user.username}: {self.theme.course.title}: {self.theme.title}: {self.title} [Lesson]"

def create\_lesson(sender, \*\*kwargs):

"""When a course is created, autofill fields"""

if kwargs['created']:

lesson = kwargs['instance']

lesson.path = lesson.pk

lesson.save()

post\_save.connect(create\_lesson, sender=Lesson)

class Step(models.Model):

"""The Lesson consists of Step"""

lesson = models.ForeignKey(Lesson, on\_delete=models.CASCADE)

title = models.CharField(max\_length=64)

content = RichTextUploadingField(blank=True)

max\_progress = models.IntegerField(default=0)

number = models.IntegerField(default=0)

path = models.CharField(max\_length=64, blank=True, null=True)

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

return f"{self.lesson.theme.course.profile.user.username} => {self.lesson.theme.course.title}: {self.lesson.theme.title} => {self.lesson.title} => {self.title} [Step]"

def create\_step(sender, \*\*kwargs):

"""When a course is created, autofill fields"""

if kwargs['created']:

step = kwargs['instance']

step.path = step.pk

step.save()

post\_save.connect(create\_step, sender=Step)

# ------------ Content Course END ----------------

# ------------ Profile to Course START --------------

class ProfileActionsLogs(models.Model):

"""Logs to Step"""

step = models.ForeignKey(Step, on\_delete=models.SET\_NULL, blank=True, null=True)

profile = models.ForeignKey(Profile, on\_delete=models.SET\_NULL, blank=True, null=True)

date\_action = models.DateTimeField(default=datetime.datetime.now)

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

return f"{self.profile.user.username} => {self.step.lesson.theme.course.title}: {self.step.title} [{self.date\_action}]"

class ProfileCourseStatus(models.Model):

"""Status Profile to Course: Being studied, Completed"""

name = models.CharField(max\_length=64)

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

return f"{self.name} [Status Profile to Course]"

class ProfileCourse(models.Model):

"""ProfileCourse"""

course = models.ForeignKey(Course, on\_delete=models.CASCADE)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

status = models.ForeignKey(ProfileCourseStatus, blank=True, null=True, on\_delete=models.SET\_NULL)

progress = models.IntegerField(default=0)

grade = models.IntegerField(blank=True, null=True)

date\_added = models.DateField(default=datetime.date.today)

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

status = None

if self.status is not None:

status = self.status.name

return f"\"{self.profile.user.username}\" => \"{self.course.profile}: {self.course.title}\" [{status}]"

def create\_profile\_to\_course(sender, \*\*kwargs):

"""When a ProfileCourse is created, autofill fields"""

if kwargs['created']:

profile\_course = kwargs['instance']

if profile\_course.status is None:

profile\_course.status = ProfileCourseStatus.objects.get(name=Util.PROFILE\_COURSE\_STATUS\_STUDYING\_NAME)

profile\_course.save()

post\_save.connect(create\_profile\_to\_course, sender=ProfileCourse)

class ProfileCourseCollection(models.Model):

"""Добавление курсы в подборки"""

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

course = models.ForeignKey(Course, on\_delete=models.CASCADE)

collection = models.ForeignKey(Collection, on\_delete=models.CASCADE, blank=True, null=True)

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

course\_title = "Добавленные"

if self.collection is not None:

course\_title = f"{self.collection.profile}: {self.collection.title}"

return f"\"{self.profile.user.username}\" => \"{self.course.profile}: {self.course.title}\" => \"{course\_title}\""

class ProfileTheme(models.Model):

"""ProfileTheme"""

theme = models.ForeignKey(Theme, on\_delete=models.CASCADE)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

progress = models.IntegerField(default=0)

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

return f"\"{self.profile.user.username}\" to \"{self.theme.course.title}: {self.theme.title}\""

class ProfileLesson(models.Model):

"""ProfileLesson"""

lesson = models.ForeignKey(Lesson, on\_delete=models.CASCADE)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

progress = models.IntegerField(default=0)

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

return f"\"{self.profile.user.username}\" to \"{self.lesson.theme.course.title}: {self.lesson.theme.title}: {self.lesson.title}\""

class ProfileStepStatus(models.Model):

"""Status Profile to Step course: Being studied, Completed"""

name = models.CharField(max\_length=64)

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

return f"{self.name} [Status Profile to Step course]"

class ProfileStep(models.Model):

"""ProfileStep"""

step = models.ForeignKey(Step, on\_delete=models.CASCADE)

profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

status = models.ForeignKey(ProfileStepStatus, blank=True, null=True, on\_delete=models.SET\_NULL)

progress = models.IntegerField(default=0)

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

return f"\"{self.profile.user.username}\" to \"{self.step.lesson.theme.course.title}: {self.step.lesson.theme.title}: {self.step.lesson.title}: {self.step.title}\""

def create\_profile\_to\_course(sender, \*\*kwargs):

"""When a ProfileStep is created, autofill fields"""

if kwargs['created']:

profile\_step = kwargs['instance']

profile\_step.status = ProfileStepStatus.objects.filter(name=Util.PROFILE\_COURSE\_STATUS\_STUDYING\_NAME)[0]

profile\_step.save()

post\_save.connect(create\_profile\_to\_course, sender=ProfileStep)

# -------- Profile to Course END ------------

# ############## COURSE END #################

Приложение 1. Листинг 20. urls\_course.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views\_course import CourseView, GradeCourseView, ActionProfileCourseView, ActionCourseView, ThemeView, LessonView, \

StepView, CourseCompletionPageView, CourseFitView, CourseSkillView, CoursePageView

router = DefaultRouter()

urlpatterns = [

path('', include(router.urls)),

# GET

path('courses/', CourseView.as\_view({'get': 'get\_courses'})),

path('mini-courses/', CourseView.as\_view({'get': 'get\_mini\_courses'})),

path('courses/all/<slug:path>/', CourseView.as\_view({'get': 'get\_all\_profile\_courses'})),

path('courses/added/<slug:path>/', CourseView.as\_view({'get': 'get\_added\_courses'})),

path('courses/created/<slug:path>/', CourseView.as\_view({'get': 'get\_created\_courses'})),

path('courses/page/<slug:path>/', CourseView.as\_view({'get': 'get\_page\_course'})),

# GRADE

path('courses/create/grade/<slug:path>/', GradeCourseView.as\_view({'post': 'set\_grade'})),

path('courses/update/grade/<slug:path>/', GradeCourseView.as\_view({'put': 'update\_grade'})),

path('courses/delete/grade/<slug:path>/', GradeCourseView.as\_view({'delete': 'delete\_grade'})),

# COURSE ACTION

path('courses/create/', ActionCourseView.as\_view({'post': 'create\_course'})),

path('courses/delete/<slug:path>/', ActionCourseView.as\_view({'delete': 'delete\_course'})),

path('courses/publish/<slug:path>/', ActionCourseView.as\_view({'post': 'publish\_course'})),

path('courses/development/<slug:path>/', ActionCourseView.as\_view({'post': 'development\_course'})),

path('courses/start-learn/<slug:path>/', CourseCompletionPageView.as\_view({'post': 'start\_learn\_course'})),

path('courses/complete-learn/<slug:path>/', CourseCompletionPageView.as\_view({'post': 'complete\_learn\_course'})),

# COURSE PAGE

path('courses/creating/<slug:path>/get/page/', CoursePageView.as\_view({'get': 'get\_page'})),

path('courses/creating/<slug:path>/save/page/', CoursePageView.as\_view({'put': 'save\_page'})),

# COURSE FIT

path('courses/creating/<slug:path>/create/fit/', CourseFitView.as\_view({'post': 'create\_fit'})),

path('courses/creating/<slug:path>/update/fit/', CourseFitView.as\_view({'put': 'update\_fit'})),

path('courses/creating/<slug:path>/delete/fit/', CourseFitView.as\_view({'delete': 'delete\_fit'})),

# COURSE SKILL

path('courses/creating/<slug:path>/create/skill/', CourseSkillView.as\_view({'post': 'create\_skill'})),

path('courses/creating/<slug:path>/update/skill/', CourseSkillView.as\_view({'put': 'update\_skill'})),

path('courses/creating/<slug:path>/delete/skill/', CourseSkillView.as\_view({'delete': 'delete\_skill'})),

# COURSE completion

path('courses/learn/<slug:path\_course>/title/',

CourseCompletionPageView.as\_view({'get': 'get\_title\_course'})),

path('courses/learn/<slug:path\_course>/themes/',

CourseCompletionPageView.as\_view({'get': 'get\_themes'})),

path('courses/learn/<slug:path\_course>/themes/<slug:path\_theme>/title/',

CourseCompletionPageView.as\_view({'get': 'get\_title\_theme'})),

path('courses/learn/<slug:path\_course>/themes/<slug:path\_theme>/lessons/',

CourseCompletionPageView.as\_view({'get': 'get\_lessons'})),

path(

'courses/learn/<slug:path\_course>/themes/<slug:path\_theme>/lessons/<slug:path\_lesson>/steps/<slug:path\_step>/list/',

CourseCompletionPageView.as\_view({'get': 'get\_steps'})),

path('courses/learn/<slug:path\_course>/themes/<slug:path\_theme>/lessons/<slug:path\_lesson>/steps/<slug:path\_step>/',

CourseCompletionPageView.as\_view({'get': 'get\_detail\_step'})),

# THEME

path('courses/creating/<slug:path>/create/theme/',

ThemeView.as\_view({'post': 'create\_theme'})),

path('courses/creating/<slug:path\_course>/get-update/theme/<slug:path\_theme>/',

ThemeView.as\_view({'get': 'get\_update\_info'})),

path('courses/creating/<slug:path\_course>/update/theme/<slug:path\_theme>/',

ThemeView.as\_view({'put': 'update\_theme'})),

path('courses/creating/<slug:path\_course>/delete/theme/<slug:path\_theme>/',

ThemeView.as\_view({'delete': 'delete\_theme'})),

# LESSON

path('courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/create/lesson/',

LessonView.as\_view({'post': 'create\_lesson'})),

path('courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/get-update/lesson/<slug:path\_lesson>/',

LessonView.as\_view({'get': 'get\_update\_info'})),

path('courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/update/lesson/<slug:path\_lesson>/',

LessonView.as\_view({'put': 'update\_lesson'})),

path('courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/delete/lesson/<slug:path\_lesson>/',

LessonView.as\_view({'delete': 'delete\_lesson'})),

# STEP

path('courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/lesson/<slug:path\_lesson>/create/step/',

StepView.as\_view({'post': 'create\_step'})),

path(

'courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/lesson/<slug:path\_lesson>/get-update/step/<slug:path\_step>/',

StepView.as\_view({'get': 'get\_update\_info'})),

path(

'courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/lesson/<slug:path\_lesson>/update/step/<slug:path\_step>/',

StepView.as\_view({'put': 'update\_step'})),

path(

'courses/creating/<slug:path\_course>/theme/<slug:path\_theme>/lesson/<slug:path\_lesson>/delete/step/<slug:path\_step>/',

StepView.as\_view({'delete': 'delete\_step'})),

path(

'courses/learn/<slug:path\_course>/themes/<slug:path\_theme>/lessons/<slug:path\_lesson>/steps/<slug:path\_step>/complete/',

CourseCompletionPageView.as\_view({'put': 'complete\_step'})),

# COURSE ADD/POP

path('courses/add/<slug:path>/', ActionProfileCourseView.as\_view({'post': 'added\_courses'})),

path('courses/pop/<slug:path>/', ActionProfileCourseView.as\_view({'delete': 'popped\_courses'})),

]

Приложение 1. Листинг 21. views\_course.py

from django.http import Http404

from django\_filters.rest\_framework import DjangoFilterBackend

from rest\_framework import permissions, status, viewsets

from rest\_framework.decorators import action

from rest\_framework.exceptions import ValidationError

from rest\_framework.filters import SearchFilter, OrderingFilter

from rest\_framework.response import Response

from .models\_course import Course, CourseInfo, ProfileCourse, CourseStatus, ProfileCourseCollection, Theme, Lesson, \

Step, ProfileCourseStatus, CourseFit, CourseSkill, CourseMainInfo, ProfileActionsLogs, ProfileStep, \

ProfileStepStatus

from .serializers\_course import GradeCourseSerializer, PageCourseSerializer, PageInfoCourseSerializer, CourseSerializer, \

MiniCourseSerializer, ActionThemeSerializer, ActionLessonSerializer, ActionStepSerializer, ProfileThemeSerializer, \

CourseTitleSerializer, ThemeTitleSerializer, ProfileLessonSerializer, GetStepSerializer, StepSerializer, \

MaxProgressUpdater, CourseFitSerializer, CourseSkillSerializer, EditPageInfoCourseSerializer, ProfileStepSerializer

from ..collection.models\_collection import Collection

from ..profile.models\_profile import Profile

from ..utils import Util, HelperFilter, HelperPaginator, HelperPaginatorValue

class CourseView(viewsets.ModelViewSet):

"""VIEW. Курсы"""

lookup\_field = 'slug'

queryset = Course.objects.all()

permission\_classes = [permissions.IsAuthenticated]

filter\_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)

filter\_fields = HelperFilter.COURSE\_FILTER\_FIELDS

search\_fields = HelperFilter.COURSE\_SEARCH\_FIELDS

ordering\_fields = HelperFilter.COURSE\_ORDERING\_FIELDS

pagination\_max\_page = HelperPaginatorValue.COURSE\_MAX\_PAGE

@staticmethod

def exists\_access\_page(course, profile):

"""Есть ли доступ к данной странице"""

status\_development = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_DEV\_NAME)

status\_released = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_RELEASE\_NAME)

if (course.status == status\_released) or (course.status == status\_development and profile == course.profile):

return True

return False

def exists\_path(self, path):

"""Существует ли такой путь"""

return len(self.queryset.filter(path=path)) != 0

@staticmethod

def exists\_profile\_path(path):

"""Существует ли такой путь к профилю"""

return len(Profile.objects.filter(path=path)) != 0

def get\_frame\_pagination(self, request, queryset, max\_page=None):

"""Вернет каркас к пагинации"""

if max\_page is None:

max\_page = self.pagination\_max\_page

pagination = HelperPaginator(request=request, queryset=queryset, max\_page=max\_page)

return {

"count": pagination.get\_count(),

"pages": pagination.get\_num\_pages(),

"next": pagination.get\_link\_next\_page(),

"previous": pagination.get\_link\_previous\_page(),

"results": pagination.page\_obj

}

def swap\_filters\_field(self, type\_filter):

"""Смена типа фильтрации"""

(self.filter\_fields, self.search\_fields, self.ordering\_fields) = HelperFilter.get\_filters\_course\_field(

type\_filter)

@action(methods=['get'], detail=False)

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

"""GET. Вернет все курсы"""

auth = Profile.objects.get(user=self.request.user)

status\_release = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_RELEASE\_NAME)

queryset = self.filter\_queryset(self.queryset.filter(status=status\_release))

frame\_pagination = self.get\_frame\_pagination(request, queryset)

serializer = CourseSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(methods=['get'], detail=False)

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

"""GET. Вернет все мини-курсы"""

auth = Profile.objects.get(user=self.request.user)

status\_release = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_RELEASE\_NAME)

queryset = self.filter\_queryset(self.queryset.filter(status=status\_release))

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)

serializer = MiniCourseSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_all\_profile\_courses(self, request, path, \*args, \*\*kwargs):

"""GET. Добавленные и созданные курсы пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

auth = Profile.objects.get(user=self.request.user)

# Получаем созданные и добавленные курсы

profile = Profile.objects.get(path=path)

added\_profile\_course = ProfileCourse.objects.filter(profile=profile)

created\_queryset = self.queryset.filter(profile=profile)

queryset\_list = [item.course.pk for item in added\_profile\_course]

for item in created\_queryset:

queryset\_list.append(item.pk)

new\_queryset = self.queryset.filter(pk\_\_in=queryset\_list)

queryset = self.filter\_queryset(new\_queryset)

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)

serializer = MiniCourseSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

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

"""GET. Добавленные курсы пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

auth = Profile.objects.get(user=self.request.user)

profile = Profile.objects.get(path=path)

self.swap\_filters\_field(HelperFilter.PROFILE\_COURSE\_TYPE)

profile\_queryset = self.filter\_queryset(ProfileCourse.objects.filter(profile=profile))

self.swap\_filters\_field(HelperFilter.COURSE\_TYPE)

queryset = [item.course for item in profile\_queryset]

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)

serializer = MiniCourseSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

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

"""GET. Созданные курсы пользователем по path"""

if not self.exists\_profile\_path(path):

return Response({'error': "Такого пользователя не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

auth = Profile.objects.get(user=self.request.user)

profile = Profile.objects.get(path=path)

queryset = self.filter\_queryset(self.queryset.filter(profile=profile))

frame\_pagination = self.get\_frame\_pagination(request, queryset, HelperPaginatorValue.MINI\_COURSE\_MAX\_PAGE)

serializer = MiniCourseSerializer(frame\_pagination.get('results'), many=True, context={'profile': auth})

frame\_pagination['results'] = serializer.data

return Response(frame\_pagination, status=status.HTTP\_200\_OK)

@action(methods=['get'], detail=True)

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

"""GET. Страница с информацией о курсе"""

if not self.exists\_path(path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

course = self.queryset.get(path=path)

profile = Profile.objects.get(user=request.user)

if not self.exists\_access\_page(course=course, profile=profile):

return Response({'error': "У вас нет доступа к этой странице"}, status=status.HTTP\_404\_NOT\_FOUND)

serializer\_course = PageCourseSerializer(course, context={'profile': profile})

course\_info = CourseInfo.objects.get(course=course)

serializer\_course\_info = PageInfoCourseSerializer(course\_info, context={'profile': profile})

return Response(data={

'course': serializer\_course.data,

'info': serializer\_course\_info.data

}, status=status.HTTP\_200\_OK)

# #########################################

# ######## ACTIONS ########

# #########################################

class ActionCourseView(viewsets.ModelViewSet):

"""VIEW. Действия над подборками"""

lookup\_field = 'slug'

queryset = Course.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_path(self, path):

"""Существует ли такой путь"""

return len(self.queryset.filter(path=path)) != 0

@action(detail=False, methods=['post'])

def create\_course(self, request):

"""POST. Cоздание курса"""

course\_title = request.data.get('title', None)

if course\_title is None:

return Response({

'error': "Вы не ввели название курса",

}, status=status.HTTP\_400\_BAD\_REQUEST)

auth = Profile.objects.get(user=self.request.user)

course = Course.objects.create(title=course\_title, profile=auth)

course.save()

return Response({

'title': course.title,

'path': course.path,

'message': "Курс успешно создан",

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def delete\_course(self, request, path):

"""DELETE. Удаление курса"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = self.queryset.get(path=path)

MaxProgressUpdater.update\_max\_progress(old=course.max\_progress, new=0, course=course)

course.delete()

return Response({

'title': course.title,

'path': course.path,

'message': "Курс успешно удален"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['post'])

def publish\_course(self, request, path):

"""POST. Опубликовать курс"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = self.queryset.get(path=path)

course.status = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_RELEASE\_NAME)

course.save()

return Response({

'path': course.path,

'title': course.title,

'status': course.status.name,

'message': "Курс успешно опубликован"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['post'])

def development\_course(self, request, path):

"""POST. Перевести курс в разработку"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = self.queryset.get(path=path)

course.status = CourseStatus.objects.get(name=Util.COURSE\_STATUS\_DEV\_NAME)

course.save()

return Response({

'path': course.path,

'title': course.title,

'status': course.status.name,

'message': "Курс успешно отправлен в разработку"

}, status=status.HTTP\_200\_OK)

class CoursePageView(viewsets.ModelViewSet):

"""VIEW. Страница курса"""

lookup\_field = 'slug'

permission\_classes = [permissions.IsAuthenticated]

@action(detail=False, methods=['get'])

def get\_page(self, request, path):

"""GET. Вернет страницу курса"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path)

course\_info = CourseInfo.objects.get(course=course)

serializer = EditPageInfoCourseSerializer(course\_info)

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

@staticmethod

def update\_course(request, course\_info):

"""Обновит курс"""

course = course\_info.course

validated\_data = request.data.get('course', None)

course.title = validated\_data.get('title', course.title)

course.description = validated\_data.get('description', course.description)

new\_image = validated\_data.get('image\_url', -1)

if new\_image != -1:

update\_image = Util.get\_image(old=course.image\_url, new=new\_image,

default=Util.DEFAULT\_IMAGES.get('course'))

course.image\_url = Util.get\_update\_image(old=course.image\_url, new=update\_image)

course.save()

return course

@staticmethod

def update\_main\_info(request, course\_info):

"""Обновит главную информацию о курсе"""

main\_info = CourseMainInfo.objects.get(course\_info=course\_info)

validated\_data = request.data.get('main\_info', None)

main\_info.goal\_description = validated\_data.get('goal\_description', main\_info.goal\_description)

new\_image = validated\_data.get('title\_image\_url', -1)

if new\_image != -1:

main\_info.image\_url = Util.get\_update\_image(old=main\_info.title\_image\_url, new=new\_image)

main\_info.save()

return main\_info

@staticmethod

def update\_fits(request, course\_info):

"""Обновит сообщество"""

fit\_list = CourseFit.objects.filter(course\_info=course\_info)

validated\_data = request.data.get('fits', None)

for new\_fit in validated\_data:

curr\_fit = fit\_list.filter(pk=new\_fit.get('pk'))[0]

curr\_fit.title = new\_fit.get('title', curr\_fit.title)

curr\_fit.description = new\_fit.get('description', curr\_fit.description)

curr\_fit.save()

return fit\_list

@staticmethod

def update\_skills(request, course\_info):

"""Обновит навыки"""

skill\_list = CourseSkill.objects.filter(course\_info=course\_info)

validated\_data = request.data.get('skills', None)

for new\_skill in validated\_data:

curr\_skill = skill\_list.filter(pk=new\_skill.get('pk'))[0]

curr\_skill.name = new\_skill.get('name', curr\_skill.name)

curr\_skill.save()

return skill\_list

@action(detail=False, methods=['put'])

def save\_page(self, request, path):

"""PUT. Сохранение страницы"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path)

course\_info = CourseInfo.objects.get(course=course)

self.update\_course(request=request, course\_info=course\_info)

self.update\_main\_info(request=request, course\_info=course\_info)

self.update\_fits(request=request, course\_info=course\_info)

self.update\_skills(request=request, course\_info=course\_info)

return Response({

'message': "Успешно обновлено"

}, status=status.HTTP\_200\_OK)

class CourseFitView(viewsets.ModelViewSet):

"""VIEW. Курс к представителю"""

lookup\_field = 'slug'

queryset = CourseFit.objects.all()

permission\_classes = [permissions.IsAuthenticated]

@staticmethod

def exists(pk):

"""Существует ли такой представитель"""

return PathValidator.get\_exists(data={'pk': pk}, model=CourseFit,

error\_text="Такого представителя не существует")

@action(detail=False, methods=['post'])

def create\_fit(self, request, path):

"""POST. Создать представителя"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path)

course\_info = CourseInfo.objects.get(course=course)

number\_new = len(self.queryset.filter(course\_info=course\_info)) + 1

data = {

'title': f"Представитель #{number\_new}",

'description': f"Причина почему",

}

serializer = CourseFitSerializer(data=data, context={'course\_info': course\_info})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'pk': serializer.data['pk'],

'title': serializer.data['title'],

'description': serializer.data['description'],

'message': "Представитель успешно создан",

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['put'])

def update\_fit(self, request, path):

"""PUT. Обновление представителя"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

exists\_fit = self.exists(pk=request.data.get('pk', None))

if exists\_fit.get('error', None) is not None:

return exists\_fit.get('error')

fit = self.queryset.get(pk=request.data.get('pk'))

serializer = CourseFitSerializer(data=request.data, instance=fit)

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'pk': request.data.get('pk'),

'title': fit.title,

'description': fit.description,

'message': "Представитель успешно обновлен"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def delete\_fit(self, request, path):

"""DELETE. Удалить представителя"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

exists\_fit = self.exists(pk=request.data.get('pk', None))

if exists\_fit.get('error', None) is not None:

return exists\_fit.get('error')

fit = self.queryset.get(pk=request.data.get('pk'))

fit.delete()

return Response({

'pk': request.data.get('pk'),

'title': fit.title,

'description': fit.description,

'message': "Представитель успешно удален"

}, status=status.HTTP\_200\_OK)

class CourseSkillView(viewsets.ModelViewSet):

"""VIEW. Курс к навыку"""

lookup\_field = 'slug'

queryset = CourseSkill.objects.all()

permission\_classes = [permissions.IsAuthenticated]

@staticmethod

def exists(pk):

"""Существует ли такой навык"""

return PathValidator.get\_exists(data={'pk': pk}, model=CourseSkill,

error\_text="Такого представителя не существует")

@action(detail=False, methods=['post'])

def create\_skill(self, request, path):

"""POST. Создать навык"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path)

course\_info = CourseInfo.objects.get(course=course)

number\_new = len(self.queryset.filter(course\_info=course\_info)) + 1

data = {'name': f"Умение #{number\_new}"}

serializer = CourseSkillSerializer(data=data, context={'course\_info': course\_info})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'pk': serializer.data['pk'],

'name': serializer.data['name'],

'message': "Умение успешно создано",

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['put'])

def update\_skill(self, request, path):

"""PUT. Обновить подборку"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

exists = self.exists(pk=request.data.get('pk', None))

if exists.get('error', None) is not None:

return exists.get('error')

skill = self.queryset.get(pk=request.data.get('pk'))

serializer = CourseSkillSerializer(data=request.data, instance=skill)

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'pk': serializer.data['pk'],

'name': serializer.data['name'],

'message': "Умение успешно обновлено",

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def delete\_skill(self, request, path):

"""DELETE. Удалить подборку"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

exists = self.exists(pk=request.data.get('pk', None))

if exists.get('error', None) is not None:

return exists.get('error')

skill = self.queryset.get(pk=request.data.get('pk'))

skill.delete()

return Response({

'pk': request.data.get('pk'),

'name': skill.name,

'message': "Умение успешно удалено"

}, status=status.HTTP\_200\_OK)

class CourseCompletionPageView(viewsets.ModelViewSet):

"""VIEW. Страница мастерской"""

lookup\_field = 'slug'

permission\_classes = [permissions.IsAuthenticated]

# PAGE THEMES

@action(detail=False, methods=['get'])

def get\_title\_course(self, request, path\_course):

"""GET. Вернет название курса"""

exists = PathValidator.exists(path\_course=path\_course)

if exists.get('error', None) is not None:

return exists.get('error')

course = Course.objects.get(path=path\_course)

serializer = CourseTitleSerializer(course)

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

@action(detail=False, methods=['get'])

def get\_themes(self, request, path\_course):

"""GET. Вернет темы курса"""

exists = PathValidator.exists(path\_course=path\_course)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

course = Course.objects.get(path=path\_course)

theme\_list = Theme.objects.filter(course=course)

serializer = ProfileThemeSerializer(theme\_list, many=True, context={'profile': auth})

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

# ###########

# PAGE LESSONS

@action(detail=False, methods=['get'])

def get\_title\_theme(self, request, path\_course, path\_theme):

"""GET. Вернет название темы"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme)

if exists.get('error', None) is not None:

return exists.get('error')

theme = Theme.objects.get(path=path\_theme)

serializer = ThemeTitleSerializer(theme)

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

@action(detail=False, methods=['get'])

def get\_lessons(self, request, path\_course, path\_theme):

"""GET. Вернет уроки темы"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

theme = Theme.objects.get(path=path\_theme)

lesson\_list = Lesson.objects.filter(theme=theme)

serializer = ProfileLessonSerializer(lesson\_list, many=True, context={'profile': auth, 'request': request})

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

# ###########

# PAGE STEPS

@action(detail=False, methods=['get'])

def get\_steps(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""GET. Вернет шаги урока"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme, path\_lesson=path\_lesson,

path\_step=path\_step)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

lesson = Lesson.objects.get(path=path\_lesson)

step\_list = Step.objects.filter(lesson=lesson)

current\_step = step\_list.get(path=path\_step)

serializer = GetStepSerializer(step\_list, many=True,

context={'profile': auth, 'current\_step': current\_step})

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

@staticmethod

def add\_profile\_action\_logs(profile, step):

"""ЛОГИ по изучению курса"""

if len(ProfileCourse.objects.filter(profile=profile, course=step.lesson.theme.course)) != 0:

if len(ProfileStep.objects.filter(profile=profile, step=step)) != 0:

if len(ProfileActionsLogs.objects.filter(profile=profile, step=step)) == 0:

profile\_action\_logs = ProfileActionsLogs.objects.create(profile=profile, step=step)

profile\_action\_logs.save()

return True

return False

@staticmethod

def add\_profile\_step(profile, step):

"""Добавить связь профиля с шагом"""

profile\_step\_list = ProfileStep.objects.filter(profile=profile, step=step)

if len(profile\_step\_list) == 0:

profile\_step = ProfileStep.objects.create(profile=profile, step=step)

profile\_step.save()

CourseCompletionPageView.add\_profile\_action\_logs(profile=profile, step=step)

@action(detail=False, methods=['get'])

def get\_detail\_step(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""GET. Вернет детальную страницу шага"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme, path\_lesson=path\_lesson,

path\_step=path\_step)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

step = Step.objects.get(path=path\_step)

serializer = StepSerializer(step, context={'profile': auth, 'request': request})

self.add\_profile\_step(profile=auth, step=step)

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

@staticmethod

def exists\_profile\_step(profile, step):

"""Изучали ли вы данный шаг"""

profile\_step\_list = ProfileStep.objects.filter(profile=profile, step=step)

if len(profile\_step\_list) == 0:

raise ValidationError({'error': 'Вы не изучаете данный шаг'})

return True

@action(detail=False, methods=['put'])

def complete\_step(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""PUT. Завершить шаг"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme, path\_lesson=path\_lesson,

path\_step=path\_step)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

step = Step.objects.get(path=path\_step)

self.exists\_profile\_step(profile=auth, step=step)

profile\_step = ProfileStep.objects.get(profile=auth, step=step)

new\_status = ProfileStepStatus.objects.filter(name=Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME)[0]

serializer = ProfileStepSerializer(instance=profile\_step, data={}, context={'status': new\_status})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({'message': "Вы успешно изучили step!"}, status=status.HTTP\_200\_OK)

# ###########

# COMPLETE COURSE

@action(detail=False, methods=['post'])

def start\_learn\_course(self, request, path):

"""POST. Начать изучение курса"""

exists = PathValidator.exists(path\_course=path)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

course = Course.objects.get(path=path)

profile\_course\_list = ProfileCourse.objects.filter(course=course, profile=auth)

if len(profile\_course\_list) == 0:

profile\_course = ProfileCourse.objects.create(course=course, profile=auth)

else:

profile\_course = profile\_course\_list[0]

profile\_course.status = ProfileCourseStatus.objects.get(name=Util.PROFILE\_COURSE\_STATUS\_STUDYING\_NAME)

profile\_course.save()

return Response({

'profile': auth.user.username,

'course': course.path,

'status': profile\_course.status.name,

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['post'])

def complete\_learn\_course(self, request, path):

"""POST. Завершить изучение курса"""

exists = PathValidator.exists(path\_course=path)

if exists.get('error', None) is not None:

return exists.get('error')

auth = Profile.objects.get(user=self.request.user)

course = Course.objects.get(path=path)

profile\_course\_list = ProfileCourse.objects.filter(course=course, profile=auth)

if len(profile\_course\_list) == 0:

return Response({'error': "Вы не поступили на этот курс, чтобы завершить его"},

status=status.HTTP\_400\_BAD\_REQUEST)

profile\_course = profile\_course\_list[0]

profile\_course.status = ProfileCourseStatus.objects.get(name=Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME)

profile\_course.save()

return Response({

'profile': auth.user.username,

'course': course.path,

'status': profile\_course.status.name,

}, status=status.HTTP\_200\_OK)

# ###########

class ThemeView(viewsets.ModelViewSet):

"""VIEW. Тема курса"""

lookup\_field = 'slug'

queryset = Theme.objects.all()

permission\_classes = [permissions.IsAuthenticated]

@action(detail=False, methods=['post'])

def create\_theme(self, request, path):

"""POST. Создание темы"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path)

number\_new\_theme = len(self.queryset.filter(course=course)) + 1

serializer = ActionThemeSerializer(data={'title': f"Тема #{number\_new\_theme}"}, context={'course': course})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'title': serializer.data['title'],

'path': serializer.data['path'],

'message': "Тема успешно создана"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_update\_info(self, request, path\_course, path\_theme):

"""GET. Вернуть данные для обновления"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path\_course)

theme = self.queryset.get(path=path\_theme)

serializer = ActionThemeSerializer(theme, context={'course': course})

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

@action(detail=False, methods=['put'])

def update\_theme(self, request, path\_course, path\_theme):

"""PUT. Обновить тему"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

course = Course.objects.get(path=path\_course)

theme = self.queryset.get(path=path\_theme)

serializer = ActionThemeSerializer(data=request.data, instance=theme, context={'course': course})

serializer.is\_valid(raise\_exception=True)

try:

serializer.save()

except ValueError as ex:

return Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)

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

@action(detail=False, methods=['delete'])

def delete\_theme(self, request, path\_course, path\_theme):

"""DELETE. Удалить тему"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

theme = self.queryset.get(path=path\_theme)

MaxProgressUpdater.update\_max\_progress(old=theme.max\_progress, new=0, theme=theme)

theme.delete()

return Response({

'title': theme.title,

'path': theme.path,

'message': "Тема успешно удалена"

}, status=status.HTTP\_200\_OK)

class LessonView(viewsets.ModelViewSet):

"""VIEW. Урок"""

lookup\_field = 'slug'

queryset = Lesson.objects.all()

permission\_classes = [permissions.IsAuthenticated]

@action(detail=False, methods=['post'])

def create\_lesson(self, request, path\_course, path\_theme):

"""POST. Создание урока"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

theme = Theme.objects.get(path=path\_theme)

number\_new\_lesson = len(self.queryset.filter(theme=theme)) + 1

path\_lesson = Util.get\_max\_path(self.queryset.filter(theme=theme)) + 1

serializer = ActionLessonSerializer(data={'title': f"Урок #{number\_new\_lesson}", 'path': path\_lesson},

context={'theme': theme})

serializer.is\_valid(raise\_exception=True)

try:

serializer.save()

except ValueError as ex:

return Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)

return Response({

'title': serializer.data['title'],

'path': serializer.data['path'],

'message': "Урок успешно создан"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_update\_info(self, request, path\_course, path\_theme, path\_lesson):

"""GET. Вернуть данные для обновления"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme,

path\_lesson=path\_lesson)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

theme = Theme.objects.get(path=path\_theme)

lesson = self.queryset.get(path=path\_lesson)

serializer = ActionLessonSerializer(lesson, context={'theme': theme})

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

@action(detail=False, methods=['put'])

def update\_lesson(self, request, path\_course, path\_theme, path\_lesson):

"""PUT. Обновить урок"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

theme = Theme.objects.get(path=path\_theme)

lesson = self.queryset.get(path=path\_lesson)

serializer = ActionLessonSerializer(data=request.data, instance=lesson, context={'theme': theme})

serializer.is\_valid(raise\_exception=True)

try:

serializer.save()

except ValueError as ex:

return Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)

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

@action(detail=False, methods=['delete'])

def delete\_lesson(self, request, path\_course, path\_theme, path\_lesson):

"""DELETE. Удалить урок"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

lesson = self.queryset.get(path=path\_lesson)

MaxProgressUpdater.update\_max\_progress(old=lesson.max\_progress, new=0, lesson=lesson)

lesson.delete()

return Response({

'title': lesson.title,

'path': lesson.path,

'message': "Урок успешно удален"

}, status=status.HTTP\_200\_OK)

class StepView(viewsets.ModelViewSet):

"""VIEW. Шаг"""

lookup\_field = 'slug'

queryset = Step.objects.all()

permission\_classes = [permissions.IsAuthenticated]

@action(detail=False, methods=['post'])

def create\_step(self, request, path\_course, path\_theme, path\_lesson):

"""POST. Создание шага"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme,

path\_lesson=path\_lesson)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

lesson = Lesson.objects.get(path=path\_lesson)

number\_new\_step = len(self.queryset.filter(lesson=lesson)) + 1

serializer = ActionStepSerializer(data={'title': f"Шаг #{number\_new\_step}"},

context={'lesson': lesson, 'number': number\_new\_step})

serializer.is\_valid(raise\_exception=True)

try:

serializer.save()

except ValueError as ex:

return Response({'error': str(ex)}, status=status.HTTP\_400\_BAD\_REQUEST)

return Response({

'title': serializer.data['title'],

'path': serializer.data['path'],

'message': "Шаг успешно создан"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['get'])

def get\_update\_info(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""GET. Вернуть данные для обновления"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme,

path\_lesson=path\_lesson, path\_step=path\_step)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

lesson = Lesson.objects.get(path=path\_lesson)

step = self.queryset.get(path=path\_step)

serializer = StepSerializer(step, context={'lesson': lesson, 'request': request})

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

@action(detail=False, methods=['put'])

def update\_step(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""PUT. Обновить шаг"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme,

path\_lesson=path\_lesson, path\_step=path\_step)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

step = self.queryset.get(path=path\_step)

serializer = ActionStepSerializer(data=request.data, instance=step, context={'step': step})

serializer.is\_valid(raise\_exception=True)

serializer.save()

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

@action(detail=False, methods=['delete'])

def delete\_step(self, request, path\_course, path\_theme, path\_lesson, path\_step):

"""DELETE. Удалить шаг"""

is\_valid = PathValidator.is\_valid(user=self.request.user, path\_course=path\_course, path\_theme=path\_theme,

path\_lesson=path\_lesson, path\_step=path\_step)

if is\_valid.get('error', None) is not None:

return is\_valid.get('error')

step = self.queryset.get(path=path\_step)

MaxProgressUpdater.update\_max\_progress(old=step.max\_progress, new=0, step=step)

step.delete()

ActionStepSerializer.update\_numbers(step\_list=Step.objects.filter(lesson=step.lesson))

return Response({

'title': step.title,

'path': step.path,

'message': "Шаг успешно удален"

}, status=status.HTTP\_200\_OK)

# #########################################

# ######## ACTIONS PROFILE ########

# #########################################

class ActionProfileCourseView(viewsets.ModelViewSet):

"""VIEW. Действия профиля к курсу"""

lookup\_field = 'slug'

queryset = Course.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_course(self, path):

"""Cуществует ли такой курс"""

return len(self.queryset.filter(path=path)) != 0

@staticmethod

def exists\_collection(path):

"""Cуществует ли такая подборка"""

return len(Collection.objects.filter(path=path)) != 0

def get\_course(self, path):

"""Вернет курс"""

course\_list = self.queryset.filter(path=path)

if len(course\_list) == 0:

raise Http404("Такого курса не существует")

return course\_list[0]

@action(detail=False, methods=['post'])

def added\_courses(self, request, path):

"""POST. Добавить курс"""

collection\_path = request.data.get('collection\_path', None)

if not self.exists\_course(path=path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

if not self.exists\_collection(path=collection\_path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

auth = Profile.objects.get(user=self.request.user)

course = Course.objects.get(path=path)

collection = Collection.objects.get(path=collection\_path)

if collection.profile != auth:

return Response({

'error': f"Вы не являетесь создателем подборки, поэтому не имеете право её изменять"

}, status=status.HTTP\_400\_BAD\_REQUEST)

profile\_course\_collection\_list = ProfileCourseCollection.objects.filter(profile=auth, course=course,

collection=collection)

if len(profile\_course\_collection\_list) != 0:

return Response({'error': "Вы уже добавили этот курс"}, status=status.HTTP\_404\_NOT\_FOUND)

profile\_course\_collection = ProfileCourseCollection.objects.create(profile=auth, course=course,

collection=collection)

profile\_course\_collection.save()

return Response({

'course': course.title,

'message': "Курс добавлен в подборку"

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def popped\_courses(self, request, path):

"""DELETE. Выкинуть подборку"""

collection\_path = request.data.get('collection\_path', None)

if not self.exists\_course(path=path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

if not self.exists\_collection(path=collection\_path):

return Response({'error': "Такой подборки не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

auth = Profile.objects.get(user=self.request.user)

course = self.get\_course(path=path)

collection = Collection.objects.get(path=collection\_path)

if collection.profile != auth:

return Response({

'error': f"Вы не являетесь создателем подборки, поэтому не имеете право её изменять"

}, status=status.HTTP\_400\_BAD\_REQUEST)

profile\_course\_collection\_list = ProfileCourseCollection.objects.filter(profile=auth, course=course,

collection=collection)

if len(profile\_course\_collection\_list) == 0:

return Response({'error': "Вы уже удалили этот курс"}, status=status.HTTP\_404\_NOT\_FOUND)

profile\_course\_collection = profile\_course\_collection\_list[0]

profile\_course\_collection.delete()

return Response({

'course': course.title,

'message': "Курс удален из подборки"

}, status=status.HTTP\_200\_OK)

# #########################################

# ######## GRADE ########

# #########################################

class GradeCourseView(viewsets.ModelViewSet):

"""VIEW. Оценка курса"""

lookup\_field = 'slug'

queryset = Course.objects.all()

permission\_classes = [permissions.IsAuthenticated]

def exists\_path(self, path):

return len(self.queryset.filter(path=path)) != 0

@action(detail=False, methods=['post'])

def set\_grade(self, request, path):

"""POST.Установить оценку """

if not self.exists\_path(path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

course = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

serializer = GradeCourseSerializer(data=request.data, context={'profile': profile, 'course': course})

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'course': course.title,

'path': course.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['put'])

def update\_grade(self, request, path):

"""PUT. Обновить оценку"""

if not self.exists\_path(path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

course = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

profile\_course = ProfileCourse.objects.get(profile=profile, course=course)

serializer = GradeCourseSerializer(data=request.data, instance=profile\_course)

serializer.is\_valid(raise\_exception=True)

serializer.save()

return Response({

'course': course.title,

'path': course.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

@action(detail=False, methods=['delete'])

def delete\_grade(self, request, path):

"""DELETE. Удалить оценку"""

if not self.exists\_path(path):

return Response({'error': "Такого курса не существует"}, status=status.HTTP\_404\_NOT\_FOUND)

course = self.queryset.get(path=path)

profile = Profile.objects.get(user=self.request.user)

if course.profile != profile:

return Response({"error": "У вас нет доступа для удаления оценки курса от имени этого аккаунта"},

status=status.HTTP\_400\_BAD\_REQUEST)

profile\_course = ProfileCourse.objects.get(profile=profile, course=course)

serializer = GradeCourseSerializer(context={'profile\_course': profile\_course})

serializer.delete\_grade()

return Response({

'course': course.title,

'path': course.path,

'grade': serializer.data['grade']

}, status=status.HTTP\_200\_OK)

class PathValidator:

"""Валидатор путей и доступа"""

@staticmethod

def get\_exists(data, model, error\_text):

"""Существет ли"""

if data is not None:

if not Util.exists\_path(model, validated\_data=data):

return {

'valid': False,

'error': Response({'error': error\_text}, status=status.HTTP\_404\_NOT\_FOUND),

}

return {'valid': True}

@staticmethod

def is\_valid(user=None, path\_course=None, path\_theme=None, path\_lesson=None, path\_step=None):

"""Валиден ли"""

exists = PathValidator.exists(path\_course=path\_course, path\_theme=path\_theme, path\_lesson=path\_lesson,

path\_step=path\_step)

if exists.get('error', None) is not None:

return exists

if user is not None:

is\_access = PathValidator.is\_access(user=user, path\_course=path\_course)

if is\_access.get('error', None) is not None:

return is\_access

return {'valid': True}

@staticmethod

def exists(path\_course=None, path\_theme=None, path\_lesson=None, path\_step=None):

"""Все поля существуют?"""

if path\_course is not None:

exists\_course = PathValidator.exists\_course(path\_course=path\_course)

if exists\_course.get('error', None) is not None:

return exists\_course

if path\_theme is not None:

exists\_theme = PathValidator.exists\_theme(path\_theme=path\_theme)

if exists\_theme.get('error', None) is not None:

return exists\_theme

if path\_lesson is not None:

exists\_lesson = PathValidator.exists\_lesson(path\_lesson=path\_lesson)

if exists\_lesson.get('error', None) is not None:

return exists\_lesson

if path\_step is not None:

exists\_step = PathValidator.exists\_step(path\_step=path\_step)

if exists\_step.get('error', None) is not None:

return exists\_step

return {'valid': True}

@staticmethod

def exists\_course(path\_course):

"""Существует ли курс"""

return PathValidator.get\_exists(data={'path': path\_course}, model=Course,

error\_text="Такого курса не существует")

@staticmethod

def exists\_theme(path\_theme):

"""Существует ли тема"""

return PathValidator.get\_exists(data={'path': path\_theme}, model=Theme, error\_text="Такой темы не существует")

@staticmethod

def exists\_lesson(path\_lesson):

"""Существует ли урок"""

return PathValidator.get\_exists(data={'path': path\_lesson}, model=Lesson,

error\_text="Такого урока не существует")

@staticmethod

def exists\_step(path\_step):

"""Существует ли шаг"""

return PathValidator.get\_exists(data={'path': path\_step}, model=Step, error\_text="Такого шага не существует")

@staticmethod

def is\_access(user, path\_course):

"""Есть ли доступ"""

auth = Profile.objects.get(user=user)

course = Course.objects.get(path=path\_course)

if course.profile != auth:

error\_text = "У вас нет доступа для создания/изменения/удаления шага от имени этого аккаунта"

return {

'valid': False,

'error': Response({'error': error\_text}, status=status.HTTP\_404\_NOT\_FOUND),

}

return {'valid': False}

Приложение 1. Листинг 22. serializers\_course.py

from rest\_framework import serializers

from .models\_course import Course, ProfileCourse, Theme, Lesson, \

Step, ProfileStep, \

CourseInfo, CourseMainInfo, CourseFit, CourseSkill, CourseStars, ProfileTheme, ProfileLesson, ProfileActionsLogs, \

ProfileCourseCollection

#####################################

# ## COURSE ##

#####################################

from ..utils import Util

class HelperCourseSerializer:

"""Помощник сериализации """

@staticmethod

def get\_is\_added(course, profile):

"""Добавлен ли"""

profile\_to\_course = ProfileCourse.objects.filter(course=course, profile=profile)

if profile\_to\_course:

return True

return False

@staticmethod

def get\_status\_progress(course, profile):

"""Вернуть статус курса"""

profile\_to\_course = ProfileCourse.objects.filter(course=course, profile=profile)

if len(profile\_to\_course) != 0:

status = profile\_to\_course[0].status

if status is not None:

return status.name

return None

class MaxProgressUpdater:

"""Обновитель максимального прогресса"""

@staticmethod

def update\_max\_progress(old, new, step=None, lesson=None, theme=None, course=None):

"""Обновить максимальный прогресс"""

diff\_max\_progress = new - old

generate\_lesson = None

if step is not None:

MaxProgressUpdater.update\_max\_progress\_step(step=step, diff\_max\_progress=diff\_max\_progress)

generate\_lesson = step.lesson

generate\_theme = None

if lesson is None:

lesson = generate\_lesson

if lesson is not None:

MaxProgressUpdater.update\_max\_progress\_lesson(lesson=lesson, diff\_max\_progress=diff\_max\_progress)

generate\_theme = lesson.theme

generate\_course = None

if theme is None:

theme = generate\_theme

if theme is not None:

MaxProgressUpdater.update\_max\_progress\_theme(theme=theme, diff\_max\_progress=diff\_max\_progress)

generate\_course = theme.course

if course is None:

course = generate\_course

if course is not None:

MaxProgressUpdater.update\_max\_progress\_course(course=course, diff\_max\_progress=diff\_max\_progress)

@staticmethod

def update\_max\_progress\_course(course, diff\_max\_progress):

"""Обновить максимальный прогресс курса"""

course.max\_progress += diff\_max\_progress

course.save()

@staticmethod

def update\_max\_progress\_theme(theme, diff\_max\_progress):

"""Обновить максимальный прогресс темы"""

theme.max\_progress += diff\_max\_progress

theme.save()

@staticmethod

def update\_max\_progress\_lesson(lesson, diff\_max\_progress):

"""Обновить максимальный урока"""

lesson.max\_progress += diff\_max\_progress

lesson.save()

@staticmethod

def update\_max\_progress\_step(step, diff\_max\_progress):

"""Обновить максимальный прогресс шага"""

step.max\_progress += diff\_max\_progress

step.save()

class ProgressUpdater:

"""Обновитель прогресса"""

@staticmethod

def get\_or\_create(model, validation\_data):

"""Вернуть или создать"""

queryset = model.objects.filter(\*\*validation\_data)

if len(queryset) == 0:

return model.objects.create(\*\*validation\_data)

return queryset[0]

@staticmethod

def update\_progress(old, new, profile\_step=None, profile\_lesson=None, profile\_theme=None, profile\_course=None):

"""Обновить прогресс"""

diff\_progress = new - old

generate\_lesson = None

if profile\_step is not None:

ProgressUpdater.update\_progress\_step(profile\_step=profile\_step, diff\_progress=diff\_progress)

data = {'lesson': profile\_step.step.lesson, 'profile': profile\_step.profile}

generate\_lesson = ProgressUpdater.get\_or\_create(model=ProfileLesson, validation\_data=data)

generate\_theme = None

if profile\_lesson is None:

profile\_lesson = generate\_lesson

if profile\_lesson is not None:

ProgressUpdater.update\_progress\_lesson(profile\_lesson=profile\_lesson, diff\_progress=diff\_progress)

data = {'theme': profile\_step.step.lesson.theme, 'profile': profile\_step.profile}

generate\_theme = ProgressUpdater.get\_or\_create(model=ProfileTheme, validation\_data=data)

generate\_course = None

if profile\_theme is None:

profile\_theme = generate\_theme

if profile\_theme is not None:

ProgressUpdater.update\_progress\_theme(profile\_theme=profile\_theme, diff\_progress=diff\_progress)

data = {'course': profile\_step.step.lesson.theme.course, 'profile': profile\_step.profile}

generate\_course = ProgressUpdater.get\_or\_create(model=ProfileCourse, validation\_data=data)

if profile\_course is None:

profile\_course = generate\_course

if profile\_course is not None:

ProgressUpdater.update\_progress\_course(profile\_course=profile\_course, diff\_progress=diff\_progress)

@staticmethod

def update\_progress\_course(profile\_course, diff\_progress):

"""Обновить прогресс курса"""

profile\_course.progress += diff\_progress

profile\_course.save()

@staticmethod

def update\_progress\_theme(profile\_theme, diff\_progress):

"""Обновить прогресс темы"""

profile\_theme.progress += diff\_progress

profile\_theme.save()

@staticmethod

def update\_progress\_lesson(profile\_lesson, diff\_progress):

"""Обновить прогресс урока"""

profile\_lesson.progress += diff\_progress

profile\_lesson.save()

@staticmethod

def update\_progress\_step(profile\_step, diff\_progress):

"""Обновить прогресс шага"""

profile\_step.progress += diff\_progress

profile\_step.save()

class CourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Курс"""

author = serializers.SerializerMethodField()

quantity\_in\_collection = serializers.SerializerMethodField()

status\_progress = serializers.SerializerMethodField(default=None)

progress = serializers.SerializerMethodField(default=None)

class Meta:

model = Course

fields = (

'path', 'title', 'description', 'author', 'image\_url', 'duration\_in\_minutes', 'rating', 'members\_amount',

'quantity\_in\_collection', 'status\_progress', 'progress')

@staticmethod

def get\_author(course):

"""Вернуть автора"""

return course.profile.user.username

def get\_quantity\_in\_collection(self, course):

"""Вернуть количество добавлений курса в подборки"""

if self.context.get('profile', None) is None:

return None

return len(ProfileCourseCollection.objects.filter(course=course, profile=self.context.get('profile')))

def get\_status\_progress(self, course):

"""Вернуть статус прогресса"""

if self.context.get('profile', None) is None:

return None

return HelperCourseSerializer.get\_status\_progress(course=course, profile=self.context.get('profile'))

def get\_progress(self, course):

"""Вернуть прогресс"""

if self.context.get('profile', None) is None:

return None

profile\_course\_list = ProfileCourse.objects.filter(course=course, profile=self.context.get('profile'))

if len(profile\_course\_list) == 0:

return None

return {

'progress': profile\_course\_list[0].progress,

'max\_progress': course.max\_progress,

}

class MiniCourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Мини курс"""

author = serializers.SerializerMethodField()

status\_progress = serializers.SerializerMethodField(default=None)

progress = serializers.SerializerMethodField(default=None)

class Meta:

model = Course

fields = (

'path', 'title', 'description', 'author', 'image\_url', 'duration\_in\_minutes', 'rating', 'members\_amount',

'status\_progress', 'progress')

@staticmethod

def get\_author(course):

"""Вернуть автора"""

return course.profile.user.username

def get\_status\_progress(self, course):

"""Вернуть статус прогресса"""

if self.context.get('profile', None) is None:

return None

return HelperCourseSerializer.get\_status\_progress(course=course, profile=self.context.get('profile'))

def get\_progress(self, course):

"""Вернуть прогресс"""

if self.context.get('profile', None) is None:

return None

profile\_course\_list = ProfileCourse.objects.filter(course=course, profile=self.context.get('profile'))

if len(profile\_course\_list) == 0:

return None

return {

'progress': profile\_course\_list[0].progress,

'max\_progress': course.max\_progress,

}

class PageCourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Детальная страница курса"""

author = serializers.SerializerMethodField()

status\_progress = serializers.SerializerMethodField(default=None)

class Meta:

model = Course

fields = ('title', 'author', 'image\_url', 'price', 'duration\_in\_minutes', 'rating', 'members\_amount',

'status\_progress')

@staticmethod

def get\_author(course):

"""Вернуть автора"""

return course.profile.user.username

def get\_status\_progress(self, course):

"""Вернуть статус прогресса"""

return HelperCourseSerializer.get\_status\_progress(course=course, profile=self.context.get('profile'))

class PageInfoCourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Страница курса """

main\_info = serializers.SerializerMethodField()

fits = serializers.SerializerMethodField()

skills = serializers.SerializerMethodField()

stars = serializers.SerializerMethodField()

class Meta:

model = CourseInfo

fields = ('main\_info', 'fits', 'skills', 'stars')

@staticmethod

def get\_main\_info(course\_info):

"""Вернуть главную информацию"""

main\_info\_list = CourseMainInfo.objects.filter(course\_info=course\_info)

if len(main\_info\_list) == 0:

return None

main\_info = main\_info\_list[0]

title\_image\_url = None

if main\_info.title\_image\_url:

title\_image\_url = main\_info.title\_image\_url.url

return {

'title\_image\_url': title\_image\_url,

'goal\_description': main\_info.goal\_description

}

@staticmethod

def get\_fits(course\_info):

"""Вернуть представителей"""

filter\_fits = CourseFit.objects.filter(course\_info=course\_info)

fits = list()

for fit in filter\_fits:

fits.append({

'title': fit.title,

'description': fit.description

})

return fits

@staticmethod

def get\_skills(course\_info):

"""Вернуть навыки"""

filter\_skills = CourseSkill.objects.filter(course\_info=course\_info)

skills = list()

for skill in filter\_skills:

skills.append(skill.name)

return skills

@staticmethod

def get\_stars(course\_info):

"""Вернуть рейтинг"""

stars = CourseStars.objects.get(course=course\_info.course)

stars\_dict = {

'five': stars.five\_stars\_count,

'four': stars.four\_stars\_count,

'three': stars.three\_stars\_count,

'two': stars.two\_stars\_count,

'one': stars.one\_stars\_count,

}

stars\_dict['total\_number'] = sum(stars\_dict.values())

return stars\_dict

class CourseEditSerializer(serializers.ModelSerializer):

"""SERIALIZER. Окно изменения курса"""

class Meta:

model = Course

fields = ('title', 'image\_url', 'description')

class EditPageInfoCourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Изменение страницы курса"""

course = serializers.SerializerMethodField()

main\_info = serializers.SerializerMethodField()

fits = serializers.SerializerMethodField()

skills = serializers.SerializerMethodField()

class Meta:

model = CourseInfo

fields = ('course', 'main\_info', 'fits', 'skills')

@staticmethod

def get\_course(course\_info):

"""Вернет курс"""

return CourseEditSerializer(course\_info.course).data

@staticmethod

def get\_main\_info(course\_info):

"""Вернуть главную информацию"""

main\_info\_list = CourseMainInfo.objects.filter(course\_info=course\_info)

if len(main\_info\_list) == 0:

return None

main\_info = main\_info\_list[0]

title\_image\_url = None

if main\_info.title\_image\_url:

title\_image\_url = main\_info.title\_image\_url.url

return {

'title\_image\_url': title\_image\_url,

'goal\_description': main\_info.goal\_description

}

@staticmethod

def get\_fits(course\_info):

"""Вернуть представителей"""

filter\_fits = CourseFit.objects.filter(course\_info=course\_info)

fits = list()

for fit in filter\_fits:

fits.append({

'pk': fit.pk,

'title': fit.title,

'description': fit.description

})

return fits

@staticmethod

def get\_skills(course\_info):

"""Вернуть навыки"""

filter\_skills = CourseSkill.objects.filter(course\_info=course\_info)

skills = list()

for skill in filter\_skills:

skills.append({

'pk': skill.pk,

'name': skill.name

})

return skills

class CourseFitSerializer(serializers.ModelSerializer):

"""SERIALIZER. Представитля"""

class Meta:

model = CourseFit

fields = ('pk', 'title', 'description')

def create(self, validated\_data):

"""Создание представителя"""

course\_info = self.context.get('course\_info')

title = validated\_data.get('title', None)

description = validated\_data.get('description', None)

return CourseFit.objects.create(course\_info=course\_info, title=title, description=description)

def update(self, instance, validated\_data):

"""Обновление представителя"""

instance.title = validated\_data.get('title', instance.title)

instance.description = validated\_data.get('description', instance.description)

instance.save()

return instance

class CourseSkillSerializer(serializers.ModelSerializer):

"""SERIALIZER. Навык"""

class Meta:

model = CourseSkill

fields = ('pk', 'name')

def create(self, validated\_data):

"""Создание навыка"""

course\_info = self.context.get('course\_info')

name = validated\_data.get('name', None)

return self.Meta.model.objects.create(course\_info=course\_info, name=name)

def update(self, instance, validated\_data):

"""Обновление навыка"""

instance.name = validated\_data.get('name', instance.name)

instance.save()

return instance

#####################################

#####################################

# ## COURSE COMPLETION PAGE ##

#####################################

# PAGE THEMES

class CourseTitleSerializer(serializers.ModelSerializer):

"""SERIALIZER. Титульник курса"""

class Meta:

model = Theme

fields = ('path', 'title', 'image\_url')

class ProfileThemeSerializer(serializers.ModelSerializer):

"""SERIALIZER. Профиль к теме"""

count\_lesson = serializers.SerializerMethodField()

progress = serializers.SerializerMethodField()

is\_complete = serializers.SerializerMethodField(default=False)

class Meta:

model = Theme

fields = ('path', 'title', 'image\_url', 'count\_lesson', 'progress', 'is\_complete')

@staticmethod

def get\_count\_lesson(theme):

"""Вернуть количество уроков в теме"""

return len(Lesson.objects.filter(theme=theme))

def get\_progress(self, theme):

"""Вернуть прогресс над темой"""

profile\_theme\_list = ProfileTheme.objects.filter(theme=theme, profile=self.context.get('profile'))

if len(profile\_theme\_list) == 0:

return None

return {

'progress': profile\_theme\_list[0].progress,

'max\_progress': theme.max\_progress,

}

def get\_is\_complete(self, theme):

"""Завершена ли тема"""

progress = self.get\_progress(theme=theme)

if (progress is not None) and (progress.get('progress') == progress.get('max\_progress')):

return True

return False

class ActionThemeSerializer(serializers.ModelSerializer):

"""SERIALIZER. Действие над темой"""

class Meta:

model = Theme

fields = ('title', 'image\_url', 'max\_progress', 'path')

def create(self, validated\_data):

"""Создать действие над темой"""

return Theme.objects.create(\*\*validated\_data, course=self.context.get('course'))

def update(self, instance, validated\_data):

"""Обновить действие над темой"""

instance.title = validated\_data.get('title', instance.title)

new\_path = validated\_data.get('path', None)

if new\_path is not None:

new\_path = Util.get\_update\_path(new\_path=new\_path)

instance.path = Util.get\_new\_path(new\_path=new\_path, old\_path=instance.path, model=Theme)

new\_image = validated\_data.get('image\_url', -1)

if new\_image != -1:

update\_image = Util.get\_image(old=instance.image\_url, new=new\_image,

default=Util.DEFAULT\_IMAGES.get('lesson'))

instance.image\_url = Util.get\_update\_image(old=instance.image\_url, new=update\_image)

instance.save()

return instance

# PAGE LESSONS

class ThemeTitleSerializer(serializers.ModelSerializer):

"""SERIALIZER. Титульник темы"""

class Meta:

model = Theme

fields = ('path', 'title', 'image\_url')

class ProfileLessonSerializer(serializers.ModelSerializer):

"""SERIALIZER. Профиль к уроку"""

count\_step = serializers.SerializerMethodField()

progress = serializers.SerializerMethodField()

is\_complete = serializers.SerializerMethodField(default=False)

current\_step = serializers.SerializerMethodField()

class Meta:

model = Lesson

fields = ('path', 'title', 'image\_url', 'current\_step', 'count\_step', 'progress', 'is\_complete')

@staticmethod

def get\_count\_step(lesson):

"""Вернуть количество шагов в уроке"""

return len(Step.objects.filter(lesson=lesson))

def get\_progress(self, lesson):

"""Вернуть прогресс"""

profile\_lesson\_list = ProfileLesson.objects.filter(lesson=lesson, profile=self.context.get('profile'))

if len(profile\_lesson\_list) == 0:

return None

return {

'progress': profile\_lesson\_list[0].progress,

'max\_progress': lesson.max\_progress,

}

def get\_is\_complete(self, lesson):

"""Завершен ли урок"""

progress = self.get\_progress(lesson=lesson)

if (progress is not None) and (progress.get('progress') == progress.get('max\_progress')):

return True

return False

def get\_link(self):

"""Вернуть ссылку на урок"""

link\_old = self.context.get('request').build\_absolute\_uri()

return "/".join(link\_old.split('/')[:-1])

def get\_current\_step(self, lesson):

"""Текуший шаг"""

step\_list = Step.objects.filter(lesson=lesson)

if len(step\_list) == 0:

return None

logs = ProfileActionsLogs.objects.filter(step\_\_in=step\_list)

if len(logs) == 0:

return f"{self.get\_link()}/{lesson.path}/steps/{step\_list[0].path}"

current\_step = logs[0]

last\_date = current\_step.date\_action

for log in logs:

if log.date\_action > last\_date:

last\_date = log.date\_action

current\_step = log

return f"{self.get\_link()}/{lesson.path}/steps/{current\_step.step.path}"

class ActionLessonSerializer(serializers.ModelSerializer):

"""SERIALIZER. Действия над уроком"""

count\_step = serializers.SerializerMethodField()

class Meta:

model = Lesson

fields = ('title', 'image\_url', 'max\_progress', 'count\_step', 'path')

def get\_count\_step(self, lesson):

"""Вернуть количество шагов в уроке"""

return len(Step.objects.filter(lesson=lesson))

def create(self, validated\_data):

"""Создать урок"""

return Lesson.objects.create(\*\*validated\_data, theme=self.context.get('theme'))

def update(self, instance, validated\_data):

"""Обновить урок"""

instance.title = validated\_data.get('title', instance.title)

new\_image = validated\_data.get('image\_url', -1)

if new\_image != -1:

update\_image = Util.get\_image(old=instance.image\_url, new=new\_image,

default=Util.DEFAULT\_IMAGES.get('lesson'))

instance.image\_url = Util.get\_update\_image(old=instance.image\_url, new=update\_image)

instance.save()

return instance

# PAGE STEPS

class GetStepSerializer(serializers.ModelSerializer):

"""SERIALIZER. Шаг"""

is\_active = serializers.SerializerMethodField(default=False)

is\_complete = serializers.SerializerMethodField(default=False)

class Meta:

model = Step

fields = ('path', 'is\_active', 'is\_complete')

def get\_is\_active(self, step):

"""Текущий шаг"""

if step == self.context.get('current\_step'):

return True

return False

def get\_is\_complete(self, step):

"""Завешен ли шаг"""

profile\_step\_list = ProfileStep.objects.filter(step=step, profile=self.context.get('profile'))

if len(profile\_step\_list) == 0:

return False

if profile\_step\_list[0].status.name == Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME:

return True

return False

def update(self, instance, validated\_data):

"""Обновить страницу с шагом"""

new\_status = self.context.get('status', None)

new\_progress = self.context.get('progress', None)

if (new\_status is not None) and (instance.status != new\_status):

instance.status = new\_status

if new\_status.name == Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME:

new\_progress = instance.step.max\_progress

if (new\_progress is not None) and (new\_progress != instance.progress):

ProgressUpdater.update\_progress(old=instance.progress, new=new\_progress, profile\_step=instance)

instance.save()

return instance

class ProfileStepSerializer(serializers.ModelSerializer):

"""SERIALIZER. Профиль к шагу"""

path\_step = serializers.SerializerMethodField()

path\_profile = serializers.SerializerMethodField()

class Meta:

model = ProfileStep

fields = ('path\_step', 'path\_profile', 'status', 'progress')

@staticmethod

def get\_path\_step(profile\_step):

"""Вернуть путь к шагу"""

return profile\_step.step.path

@staticmethod

def get\_path\_profile(profile\_step):

"""Вернуть путь к профилю"""

return profile\_step.profile.path

@staticmethod

def get\_status(profile\_step):

"""Вернуть статус"""

return profile\_step.status.name

def update(self, instance, validated\_data):

"""Обновить шаг к профилю"""

new\_status = self.context.get('status', None)

new\_progress = self.context.get('progress', None)

if (new\_status is not None) and (new\_status != instance.status):

instance.status = new\_status

if new\_status.name == Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME:

new\_progress = instance.step.max\_progress

if (new\_progress is not None) and (new\_progress != instance.progress):

ProgressUpdater.update\_progress(old=instance.progress, new=new\_progress, profile\_step=instance)

instance.save()

return instance

class StepSerializer(serializers.ModelSerializer):

"""SERIALIZER. Шаг"""

is\_complete = serializers.SerializerMethodField(default=False)

prev = serializers.SerializerMethodField()

next = serializers.SerializerMethodField()

class Meta:

model = Step

fields = ('path', 'title', 'content', 'is\_complete', 'prev', 'next')

def get\_is\_complete(self, step):

"""Заверешен ли курс"""

profile\_step\_list = ProfileStep.objects.filter(step=step, profile=self.context.get('profile'))

if len(profile\_step\_list) == 0:

return False

if profile\_step\_list[0].status.name == Util.PROFILE\_COURSE\_STATUS\_STUDIED\_NAME:

return True

return False

def get\_link(self):

"""Вернуть ссылку"""

link\_old = self.context.get('request').build\_absolute\_uri()

return "/".join(link\_old.split('/')[:-2])

def get\_prev(self, step):

"""Вернуть ссылку на предыдущий шаг"""

if step.number == 1:

return None

step\_prev = Step.objects.get(number=step.number - 1)

return f"{self.get\_link()}/{step\_prev.path}"

def get\_next(self, step):

"""Вернуть ссылку на следующий шаг"""

if step.number == len(Step.objects.filter(lesson=step.lesson)):

return None

step\_next = Step.objects.get(number=step.number + 1)

return f"{self.get\_link()}/{step\_next.path}"

class ActionStepSerializer(serializers.ModelSerializer):

"""SERIALIZER. Действия над шагом"""

class Meta:

model = Step

fields = ('title', 'content', 'max\_progress', 'path')

@staticmethod

def update\_numbers(step\_list):

"""Обновить номер шага"""

for i in range(len(step\_list)):

step\_list[i].number = i + 1

step\_list[i].save()

def create(self, validated\_data):

"""Создание шага"""

lesson = self.context.get('lesson')

number = self.context.get('number')

step = Step.objects.create(\*\*validated\_data, lesson=lesson, number=number)

MaxProgressUpdater.update\_max\_progress(old=0, new=1, step=step)

return step

def update(self, instance, validated\_data):

"""Обновить шаг"""

instance.title = validated\_data.get('title', instance.title)

instance.content = validated\_data.get('content', instance.content)

new\_max\_progress = validated\_data.get('max\_progress', None)

if (new\_max\_progress is not None) and (new\_max\_progress != instance.max\_progress):

MaxProgressUpdater.update\_max\_progress(old=instance.max\_progress, new=new\_max\_progress, step=instance)

instance.save()

return instance

# #########################################

# ######## GRADE ########

# #########################################

class GradeCourseSerializer(serializers.ModelSerializer):

"""SERIALIZER. Оценки курса"""

class Meta:

model = ProfileCourse

fields = ('grade',)

@staticmethod

def add\_course\_star(course, grade):

"""Добавить оценку курсу"""

stars = CourseStars.objects.get(course=course)

if grade == 1:

stars.one\_stars\_count += 1

elif grade == 2:

stars.two\_stars\_count += 1

elif grade == 3:

stars.three\_stars\_count += 1

elif grade == 4:

stars.four\_stars\_count += 1

elif grade == 5:

stars.five\_stars\_count += 1

stars.save()

@staticmethod

def difference\_course\_star(course, grade):

"""Убрать оценку у курса"""

stars = CourseStars.objects.get(course=course)

if grade == 1:

stars.one\_stars\_count -= 1

elif grade == 2:

stars.two\_stars\_count -= 1

elif grade == 3:

stars.three\_stars\_count -= 1

elif grade == 4:

stars.four\_stars\_count -= 1

elif grade == 5:

stars.five\_stars\_count -= 1

stars.save()

@staticmethod

def update\_rating\_course(course):

"""Обновить рейтинг курса"""

stars = CourseStars.objects.get(course=course)

sum\_grade = stars.one\_stars\_count + stars.two\_stars\_count \* 2 + stars.three\_stars\_count \* 3 \

+ stars.four\_stars\_count \* 4 + stars.five\_stars\_count \* 5

count = stars.one\_stars\_count + stars.two\_stars\_count + stars.three\_stars\_count + stars.four\_stars\_count \

+ stars.five\_stars\_count

try:

rating = sum\_grade / count

except ZeroDivisionError:

rating = 0

course.rating = rating

course.save()

def create(self, validated\_data):

"""Создание оценки"""

profile\_course = ProfileCourse.objects.get(profile=self.context.get('profile'),

course=self.context.get('course'))

if profile\_course.grade is not None:

raise serializers.ValidationError({'grade': 'Вы уже оценивали курс'})

profile\_course.grade = validated\_data['grade']

self.add\_course\_star(course=profile\_course.course, grade=profile\_course.grade)

self.update\_rating\_course(course=profile\_course.course)

profile\_course.save()

return profile\_course

def update(self, instance, validated\_data):

"""Обновление оценки"""

grade\_list = (1, 2, 3, 4, 5)

new\_grade = validated\_data.get('grade')

if (type(new\_grade) == int) and (new\_grade in grade\_list) and (instance.grade != new\_grade):

self.difference\_course\_star(course=instance.course, grade=instance.grade)

instance.grade = validated\_data.get('grade')

instance.save()

self.add\_course\_star(course=instance.course, grade=instance.grade)

self.update\_rating\_course(course=instance.course)

return instance

def delete\_grade(self):

"""Удаление оценки"""

profile\_course = self.context.get('profile\_course')

if profile\_course.grade is not None:

self.difference\_course\_star(course=profile\_course.course, grade=profile\_course.grade)

self.update\_rating\_course(course=profile\_course.course)

profile\_course.grade = None

profile\_course.save()

return profile\_course