МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационны систем и технологий

Кафедра Информационные системы

Дисциплина Проектирование и архитектура програмных систем

**КУРСОВОЙ ПРОЕКТ**

Тема\_\_\_\_\_Проектирование и разработка модуля повышения квалификации для университета\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_Салин О.А.\_\_\_\_\_\_\_\_\_\_\_\_/

подпись инициалы, фамилия

Курс\_\_\_\_\_\_третий\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа\_\_\_ПИбд-33\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление/ специальность 09.03.04 Программная инженерия

Руководитель \_\_\_\_\_\_\_ст.преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность, ученая степень, ученое звание

\_\_\_\_\_Корунова Надежда Владимировна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2025 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационны систем и технологий

Кафедра Информационные системы

Дисциплина Проектирование и архитектура програмных систем

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

студенту \_\_ПИбд-33\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_Салину О.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

группа фамилия, инициалы

**Тема проекта** \_\_\_\_Проектирование и разработка модуля повышения квалификации для университета\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Срок сдачи законченного проекта** «30» мая 2025 г.

**Исходные данные к проекту** веб-приложение на flask, tg-bot на python, desktop-приложение windows forms, курс лекций по дисциплине «Проектирование и архитектура программных систем», методические указания к лабораторным работам по дисциплине «Проектирование и архитектура программных систем», \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(базовое предприятие, характер курсового проекта (работы):

задание кафедры, инициативная НИР, рекомендуемая литература, материалы практики)

**Содержание пояснительной записки** (перечень подлежащих разработке вопросов)

Описание предметной области

Требования к проекту

Оценка архитектурного решения

Проектное решение

Реализация приложения (контрольный пример)

**Перечень графического материала** (с точным указанием обязательных чертежей)

Диаграмма IDEF0

Диаграмма вариантов использования

Диаграмма классов

Диаграмма последовательности

Диаграмма компонентов

Диаграмма развертывания

Руководитель \_\_\_\_ст.преп.\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_Н.В. Корунова\_\_\_\_\_\_/

должность подпись инициалы, фамилия

« 25 »\_февраля\_\_\_\_\_\_\_\_\_\_2025 г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **/\_\_\_**О.А. Салин**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовой проект**

студента \_\_\_\_\_\_Салина Олега Алексеевича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя и отчество

Факультет\_\_\_ИСТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_группа\_ПИбд-33\_\_\_\_\_курс\_\_3\_\_\_\_

Дисциплина\_\_Проектирование и архитектура програмных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема проекта (работы) \_\_\_\_\_ Проектирование и разработка модуля повышения квалификации для университета \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Структура пояснительной записки курсового проекта соответствует заданию и теме. В ходе работы над курсовым проектом сформировано проектное решение, предоставлены обоснование и реализация данного проектного решения в виде контрольного примера. Студент выполнил курсовой проект самостоятельно.

Программный продукт по проектному решению реализован в виде клиентского приложения с запросами к БД.

Программный продукт по проектному решению реализован в виде клиент-серверного приложения с REST API.

Программный продукт по проектному решению реализован в виде трех крлиентов и сервера для обращения к БД.

Работа выполнена и сдана по графику в установленный срок.

Руководитель\_\_ ст.преп.\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ /Н.В. Корунова /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

**Оглавление**

[Введение 5](#_Toc200492022)

[Описание предметной области 6](#_Toc200492023)

[Требования к проекту 9](#_Toc200492024)

[Оценка архитектурного решения 22](#_Toc200492025)

[Проектное решение 27](#_Toc200492026)

[Реализация приложения 41](#_Toc200492027)

[Выводы по реализации приложения на основании проекта 57](#_Toc200492028)

[Заключение 58](#_Toc200492029)

[Библиографический список 59](#_Toc200492030)

[ПРИЛОЖЕНИЕ. Листинг кода 60](#_Toc200492031)

# Введение

Целью курсового проекта является создание системы, позволяющей хранить данные о повышениях квалификаций преподавателей университета (сертификат, дата прохождения, номер документа), а также отслеживание пропущенных квалификаций преподавателей и запись их на курсы.

Пользователями разрабатываемой системы являются преподаватели и сотрудники университета.

Объектом исследования является университет

Предметом исследования в данной работе является повышение квалификаций

В качестве методов исследования используется анализ предметной области, исследование и моделирование процессов, разработка проекта с использованием UML-диаграмм, а также реализация системы на языке программирования высокого уровня.

Для моделирования информационной системы был использован структурный подход (нотация IDEF0 - инструмент Ramus) и объектно-ориентированный подход (нотация UML – интструмент Visual Paradigm). При реализации использовался язык программирования python с использование сторонних библиотек и фреймворков. В качестве системы управления базами данных использовались postgresql, sqlite и redis.

# Описание предметной области

Сегодня преподавателям университетов приходится проходить различные курсы различных типов. Курсы каждого типа должны быть пройдены не позже, чем каждые 3 года. Чтобы упростить процесс отслеживания просроченных курсов и хранения данных о них, разрабатываются автоматизированные системы.

Чтобы реализовать автоматизированную систему, решающую описанную выше задачу необходимо разработать приложение, реализующее функционал работы с факультетами, кафедрами, курсами, типами курсов, преподавателями, сотрудниками и сертификатами.

Приложение условно можно разбить на 3 модуля: авторизация, модуль сотрудника, модуль преподавателя.

Преподаватель должен иметь возможность загружать сертификаты курсов и получать уведомления от сотрудников.

Сотрудник должен иметь возможность получать все необходимые данные и проставлять отметки о прохождении курсов преподавателями.

Некоторый функционал будет доступен только в определенных клиентах.

**Аналоги приложений для учета квалификаций**

**1. LinkedIn Learning + LinkedIn Profiles**

**Преимущества:**

* Интеграция с профессиональными профилями.
* Возможность добавления сертификатов и курсов.
* Автоматическая верификация некоторых программ (например, Coursera, Udemy).
* Социальная сеть для поиска работы и нетворкинга.

**Недостатки:**

* Нет глубокой интеграции с университетскими системами.
* Ограниченная аналитика для администраторов вузов.
* Зависимость от платформы LinkedIn.

**2. Degreed**

**Преимущества:**

* Платформа для управления навыками и обучением.
* Интеграция с LMS (Moodle, Blackboard).
* Персонализированные рекомендации по обучению.
* Аналитика для HR и учебных заведений.

**Недостатки:**

* Высокая стоимость для небольших вузов.
* Требуется адаптация под локальные стандарты учета.

**3. Badgr (сейчас часть Instructure/Canvas)**

**Преимущества:**

* Система цифровых бейджей (микрокредиты, сертификаты).
* Поддержка открытых стандартов (Open Badges).
* Интеграция с Canvas и другими LMS.

**Недостатки:**

* Ориентирован на микро-квалификации, а не на полный учет.
* Ограниченные возможности для сложных отчетов.

**4. MyCred** (для WordPress)

**Преимущества:**

* Гибкость (можно настроить под нужды вуза).
* Поддержка бейджей и балльных систем.
* Низкая стоимость.

**Недостатки:**

* Требует технической поддержки.
* Нет глубокой интеграции с профессиональными стандартами.

**5. СКИФ (Система квалификаций и сертификации, РФ)**

**Преимущества:**

* Официальная российская система учета квалификаций.
* Интеграция с государственными реестрами.

**Недостатки:**

* Ограниченная гибкость для вузов.

# Требования к проекту

**1. Назначение и цели создания системы**

Система «Университет. Повышение квалификации» предназначена для хранения/получения сертификатов прохождения курсов преподавателями, а также для получения списков прохождений квалификаций.

Целью системы является автоматизация процессов отслеживания прохождений курсов преподавателями.

**2. Требования к системе**

Требования к пользователям системы:

Пользователями системы должны являться преподаватели и сотрудники университета.

* 1. Требование к функциям (задачам), выполняемым системой

| Функция | Исходные данные | Результат |
| --- | --- | --- |
| Клиентская часть (неавторизованный пользователь) | | |
| Авторизация | Запрос к серверу приложений для авторизации пользователя в системе  Запрос содержит следующие данные:  Имя пользователя, пароль | В случае корректных данных:  Ответ от сервера приложений, переход на главную страницу приложения.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице авторизации. |
| Клиентская часть (пользователь авторизованный, сотрудник) | | |
| Создание, редактирование факультета | Запрос к серверу приложений для создания факультета. Запрос содержит следующие данные: название, id (для редактирования) | В случае корректных данных:  Ответ от сервера приложений, переход на страницу факультетов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания факультета. |
| Удаление факультета | Запрос к серверу приложений для удаления факультета. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений, переход на страницу факультетов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице факультетов. |
| Формирование отчета по факультету | Запрос к серверу приложения для получения страницы с отчетом. Запрос содержит следующие данные: дата начала отчета, дата конца отчета, id факультета. | В случае корректных данных:  Ответ от сервера приложений, получение страницы с сформированным отчетом.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице отчета. |
| Скачивание отчета по факультету | Запрос к серверу приложения для получения файла с отчетом. Запрос содержит следующие данные: дата начала отчета, дата конца отчета, id факультета. | В случае корректных данных:  Ответ от сервера приложений, получение файла отчета.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице отчета. |
| Создание, редактирование кафедры | Запрос к серверу приложений для создания факультета. Запрос содержит следующие данные: название, id (для редактирования), id факультета | В случае корректных данных:  Ответ от сервера приложений, переход на страницу кафедр.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания кафедры. |
| Удаление кафедры | Запрос к серверу приложений для удаления кафедры. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений, переход на страницу кафедра.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице кафедр. |
| Создание, редактирование типа курсов | Запрос к серверу приложений для создания типа курсов. Запрос содержит следующие данные: название, id (для редактирования) | В случае корректных данных:  Ответ от сервера приложений, переход на страницу типов курсов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания типа курсов. |
| Удаление типа курсов | Запрос к серверу приложений для удаления типа курсов. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений, переход на страницу типов курсов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице типов курсов. |
| Формирование отчета по типу курсов | Запрос к серверу приложения для получения страницы с отчетом. Запрос содержит следующие данные: дата начала отчета, дата конца отчета, id типа курсов. | В случае корректных данных:  Ответ от сервера приложений, получение страницы с сформированным отчетом.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице отчета. |
| Скачивание отчета по типу курсов | Запрос к серверу приложения для получения файла с отчетом. Запрос содержит следующие данные: дата начала отчета, дата конца отчета, id типа курсов. | В случае корректных данных:  Ответ от сервера приложений, получение файла отчета.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице отчета. |
| Создание, редактирование курсов | Запрос к серверу приложений для создания факультета. Запрос содержит следующие данные: название, id (для редактирования), id типа курсов | В случае корректных данных:  Ответ от сервера приложений, переход на страницу курсов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания курса. |
| Удаление курса | Запрос к серверу приложений для удаления кафедры. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений, переход на страницу курсов.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице курсов. |
| Создание, редактирование пользователей | Запрос к серверу приложений для создания пользователей. Запрос содержит следующие данные: полное имя, имя пользователя, пароль, роль (для создания), id(для редактирования) | В случае корректных данных:  Ответ от сервера приложений, переход на страницу пользователей.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания пользователя. |
| Удаление пользователя | Запрос к серверу приложений для удаления пользователя. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений, переход на страницу пользователей.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице пользователей. |
| Блокировка/разблокировка пользователя | Запрос к серверу приложений для блокировки/разблокировки пользователя. Запрос содержит следующие данные: id | В случае корректных данных:  Ответ от сервера приложений. Смена иконки.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице пользователей. |
| Создание квалификации для пользователя | Запрос к серверу приложений для составления квалификации пользователя. Запрос содержит следующие данные: id, дата начала, дата конца, id курсов, название места прохождения квалификации | В случае корректных данных:  Скачивание файла  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице создания квалификации. |
| Отправка сообщения пользователю | Запрос к серверу приложений для отправки сообщения пользователю. Запрос содержит следующие данные: id, текст сообщения. | В случае корректных данных:  Ответ от сервера приложений.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице пользователей. |
| Редактирование сертификата преподавателя | Запрос к серверу приложений для отправки редактирования сертификата. Запрос содержит следующие данные: id преподавателя, дата прохождения, номер подтверждающего документа. | В случае корректных данных:  Ответ от сервера приложений, переход на страницу сертификатов пользователей.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице сертификатов пользователей. |
| Скачивание сертификата преподавателя. | Запрос к серверу приложения для получения файла сертификата. Запрос содержит следующие данные: id преподавателя, id курса. | В случае корректных данных:  Ответ от сервера приложений, получение файла сертификата.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице сертификатов преподавателей или в чате с тг ботом. |
| Составление списка прохождения квалификаций. | Запрос к серверу приложения для получения списка. | В случае корректных данных:  Ответ от сервера приложений, получение списка.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице сертификатов преподавателей. |
| Отзыв токена пользователя. | Запрос к серверу приложения для получения файла сертификата. Запрос содержит следующие данные: id пользователя. | В случае корректных данных:  Ответ от сервера приложений.  Ответ от сервера приложений, с ошибкой и отображении ее в десктоп приложении. |
| Закрытие сессий пользователя. | Запрос к серверу приложения для получения закрытия сессий пользователя. Запрос содержит следующие данные: id пользователя. | В случае корректных данных:  Ответ от сервера приложений.  Ответ от сервера приложений, с ошибкой и отображении ее в десктоп приложении. |
| Закрытие сессии пользователя. | Запрос к серверу приложения для получения закрытия сессий пользователя. Запрос содержит следующие данные: id пользователя, id сессии. | В случае корректных данных:  Ответ от сервера приложений.  Ответ от сервера приложений, с ошибкой и отображении ее в десктоп приложении. |
| Клиентская часть (пользователь авторизованный, преподаватель) | | |
| Загрузка сертификата о прохождении курса | Запрос к серверу приложения для загрузки файла сертификата. Запрос содержит следующие данные: id преподавателя, id курса, файл сертификата | В случае корректных данных:  Ответ от сервера приложений.  Ответ от сервера приложений, с ошибкой и отображении ее на странице курсов. |
| Скачивание сертификата преподавателя. | Запрос к серверу приложения для получения файла сертификата. Запрос содержит следующие данные: id преподавателя, id курса. | В случае корректных данных:  Ответ от сервера приложений, получение файла сертификата.  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице курсов. |
| Клиентская часть (пользователь авторизованный) | | |
| Выход. | Запрос к серверу приложения для выхода из аккаунта. | В случае корректных данных:  Ответ от сервера приложений, переход на страницу входа  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице курсов. |
| Чтение уведомления. | Запрос к серверу приложения для получения файла сертификата. Запрос содержит следующие данные: id уведомления. | В случае корректных данных:  Ответ от сервера приложений  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице уведомленийй или в чате с тг ботом. |
| Удаление уведомления. | Запрос к серверу приложения для удаления файла сертификата. Запрос содержит следующие данные: id уведомления. | В случае корректных данных:  Ответ от сервера приложений  В случае некорректных данных:  Ответ от сервера приложений, с ошибкой и отображении ее на странице уведомленийй или в чате с тг ботом. |
| Серверная часть (сервер-приложение) | | |
| Запросы к серверу бд для соответствующих клиентских запросов. | Запрос к серверу бд. Запрос содержит все необходимые данные. | В случае корректных данных:  Ответ от сервера бд  В случае некорректных данных:  Исключение, которое необходимо обработать. |

1. **Требования к атрибутам качества**

3.1 Требования к производительности

К количеству одновременно работающих пользователей системы на начальном этапе не должно превышать 10 человек

Количество одновременно выполняемых запросов к серверу не должно превышать 10 запросов

Время отклика сервера на входящие запросы не должно превышать 20 с при количестве одновременно работающих пользователей, не превышающем 10 человек

Время отклика сервера на входящие запросы не должно превышать 30 с при количестве одновременно работающих пользователей, превышающем 10

3.2 Требования к безопасности

Предъявляются требования к разграничению ролей (соответствующие маршруты доступны только соответствующим ролям) и к защите от csrf атак.

3.3 Требования к эргономике и технической эстетике

Дизайн пользовательского интерфейса клиентского приложения должен удовлетворять человека без зрения.

3.4 Требования к защите информации от несанкционированного доступа

Для получения полного доступа к функциям Системы необходимо являться администратором (быть авторизованным).

Для надежной передачи данных используется протокол https.

3.5 Требования к доступности

* 1. Непрерывная работа системы. Система должна быть доступна 99% времени
  2. Требования к функциональности

Вся заявленная функциональность должна работать.

* 1. Удобство сопровождения

В случае несоответствия проекта требованиям заказчика (преподавателя) должна быть возможность удобно вносить правки.

1. **Требования к сквозной функциональности**
   1. Логирование (логирование ошибок и прочей информации)
   2. Модульное тестирование (тестирование бизнес логики)
   3. Сессия (для авторизации и аутентификации в основном модуле используется сессия)
   4. Bearer токен (для авторизации и аутентификации в модуле API используется Bearer)
   5. Обработка ошибок (пользователь получает ошибки не в «сыром» виде)
   6. Валидация (валидация данных на клиентах)

# Оценка архитектурного решения

Предоставим несколько потенциальных архитектурных решений и оценим их с помощью облегчённого метода оценки архитектур-альтернатив («Lightweight Architecture Alternative Assessment Method»).

**1. Построение дерева атрибутов качества.**

Для разрабатываемой системы можно выделить следующие категории и подкатегории атрибутов качества:

* *Доступность: непрерывная работа системы*
* *Производительность: Масштабируемость системы при высокой загрузке, быстрый отклик на запрос пользователя*
* *Расширяемость: простота добавления нового функционала*
* *Функциональность: реализация и простота реализации всего необходимого функционала*

**2. Ранжирование узлов дерева.**

На данном шаге необходимо определить приоритет каждой категории и подкатегории атрибутов качества и на основе полученных рангов распределить веса узлов дерева. Чем больше вес узла, тем приоритетнее данный атрибут качества для разрабатываемой системы.

Расставляются приоритеты и присваиваются значения весов согласно таблице весов ранга:



Результат ранжирования представлен на рисунке 1.

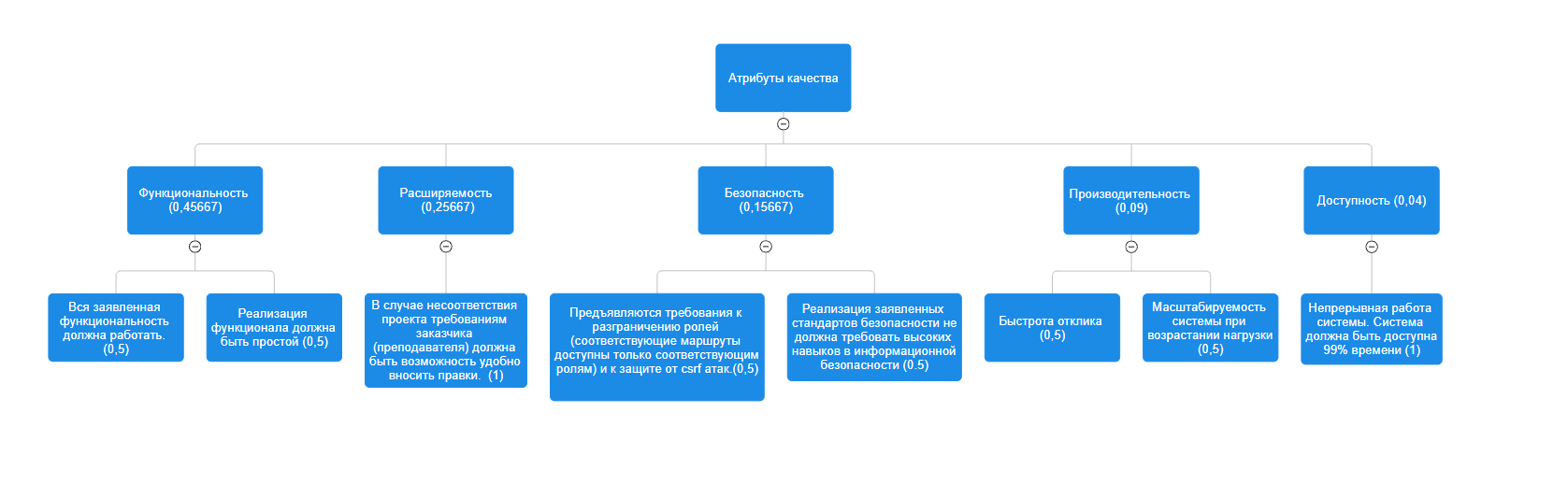


Рисунок 1. Атрибуты качества

**3. Предоставление архитектур-кандидатов.**

В качестве альтернативных решений будут рассматриваться следующие варианты архитектур:

а) Микросервисная архитектура

Микросервисная архитектура - это стиль проектирования программного обеспечения, где приложение разбито на маленькие независимые и взаимодействующие между собой компоненты - микросервисы. Каждый микросервис выполняет свою собственную задачу и может быть развернут, масштабирован, и заменен независимо от других микросервисов.

Преимущества:

* Гибкость: каждый микросервис может быть разработан независимо от других, что позволяет быстро адаптироваться к изменениям среды.
* Масштабируемость: каждый микросервис можно масштабировать отдельно, что позволяет увеличить производительность только тех компонентов, которые это требуют.
* Сопровождаемость: разные микросервисы можно сопровождать независимо друг от друга, что позволяет снизить затраты на обслуживание всей системы.
* Использование разных технологий: различные микросервисы могут быть разработаны на разных технологиях, что позволяет использовать наиболее эффективные инструменты для каждого компонента.

Недостатки:

* Сложность: микросервисная архитектура может быть достаточно сложной для понимания и управления, особенно если вы работаете над большой системой.
* Затраты на инфраструктуру: каждый микросервис требует инфраструктуры и ресурсов для работы, что может привести к увеличению затрат.
* Необходимость управления взаимодействием: необходимо управлять взаимодействием между различными микросервисами, чтобы обеспечить эффективную работу всей системы.
* Необходимость грамотного планирования: микросервисная архитектура требует грамотного планирования и дизайна, чтобы минимизировать потенциальные проблемы.

б) Монолитная архитектура

Монолитная архитектура - это классический подход к проектированию приложений, когда весь код и логика хранятся в одном приложении, которое запускается на одном сервере.

Преимущества:

* Простота: монолит приложение относительно проще в разработке и управлении, чем система, состоящая из микросервисов.
* Легко масштабируемость: монолит можно развернуть на нескольких серверах и настроить балансировку нагрузки, что позволяет успешно масштабировать приложение.
* Удобное тестирование: для монолита тестирование происходит внутри единого процесса и необходимые связи между различными компонентами существуют уже на уровне программного кода.
* Простота развертывания: монолит легко развернуть на любом сервере, таким образом, минимизируя вероятность возникновения проблем.

Недостатки:

* Сложность масштабирования: монолит сложнее масштабировать, чем систему, разбитую на микросервисы.
* Высокий уровень связности: в монолите между компонентов может существовать тесная связь, что делает проект менее гибким и менее устойчивым к изменениям.
* Сложность внесения изменений: внесение изменений в одно компоненты может затронуть другие компоненты, что может привести к проблемам в работе всего приложения.
* Необходимость внесения изменений: монолит может стать слишком громоздким и сложно крупным для поддержки со временем, что может привести к большим затратам на разработку и поддержку.

Сравним два решения согласно выделенным сценариям атрибутов качества системы.

**4. Оценка решений**

|  |  |  |  |
| --- | --- | --- | --- |
| **Сценарий** | **Вес** | **Архитектура 1** | **Архитектура 2** |
| 1.1 Вся заявленная функциональность должна работать. | 0,228335 | 5 | 5 |
| 1.2 Реализация функционала должна быть простой | 0,228335 | 2 | 4 |
| 2 Расширяемость | 0,25667 | 4 | 4 |
| 3.1 Предъявляются требования к разграничению ролей (соответствующие маршруты доступны только соответствующим ролям) и к защите от csrf атак. | 0,078335 | 3 | 4 |
| 3.2 Реализация заявленных стандартов безопасности не должна требовать высоких навыков в информационной безопасности | 0,078335 | 2 | 4 |
| 4.1 Быстрота отклика | 0,045 | 3 | 5 |
| 4.2 Масштабируемость системы при возрастании нагрузки | 0,045 | 3 | 4 |
| 5. Доступность | 0,04 | 4 | 3 |
| Итого |  | 3,4467 | 4,233375 |

**5. Анализ результатов**

По полученным результатам можно сделать вывод о том, что монолитная архитектура является наиболее подходящим под имеющиеся нужды решением среди представленных вариантов.

Далее рассмотрим проектное решение на основе монолитной архитектуры.

# Проектное решение

**Архитектурная модель**, используемая в данном приложении, представляет собой смесь слоистой архитектуры и архитектуры с использованием flask blueprints. Условно приложение поделено на несколько компонент: api, main, auth, errors. Общие части (к которым обращаются все компоненты) вынесены в общую зону видимости (бизнес логика, шаблоны), а остальные (контроллеры, формы, декораторы) сгруппированы по компонентам.

**Архитектурный стиль**, выбранный для данного приложения (сервер и клиенты), является монолитной архитектурой.

**Паттерны и фреймворки**

MVC – предполагает, что у нас есть model и view, которые взаимодействуют через controller. То есть присутствует четкое разделение кода.

Singleton **–** используется для базы данных (один объект). Предполагает, что у нас присутствует ровно один экземпляр базы данных.

Decorator – используется для реализации контроллеров и разграничения прав доступа. Является функцией, добавляющей новый функционал к уже готовой функции.

Controller – отвечает за получение и координацию выполнения операций, поступающих на уровне интерфейса пользователя.

Стратегия – поведенческий паттерн, позволяющий подменять алгоритм.

Фабрика приложений — это паттерн проектирования, часто используемый в веб-фреймворках для создания экземпляров приложения с гибкой конфигурацией.

**IDE и языки программирования**.

Язык программирования Python

Visual Studio Code - интегрированная среда разработки, стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений.

**Структурная модель проекта**

На рисунках 1-3 предоставлена контекстная диаграмма и декомпозиция в нотации IDEF0.

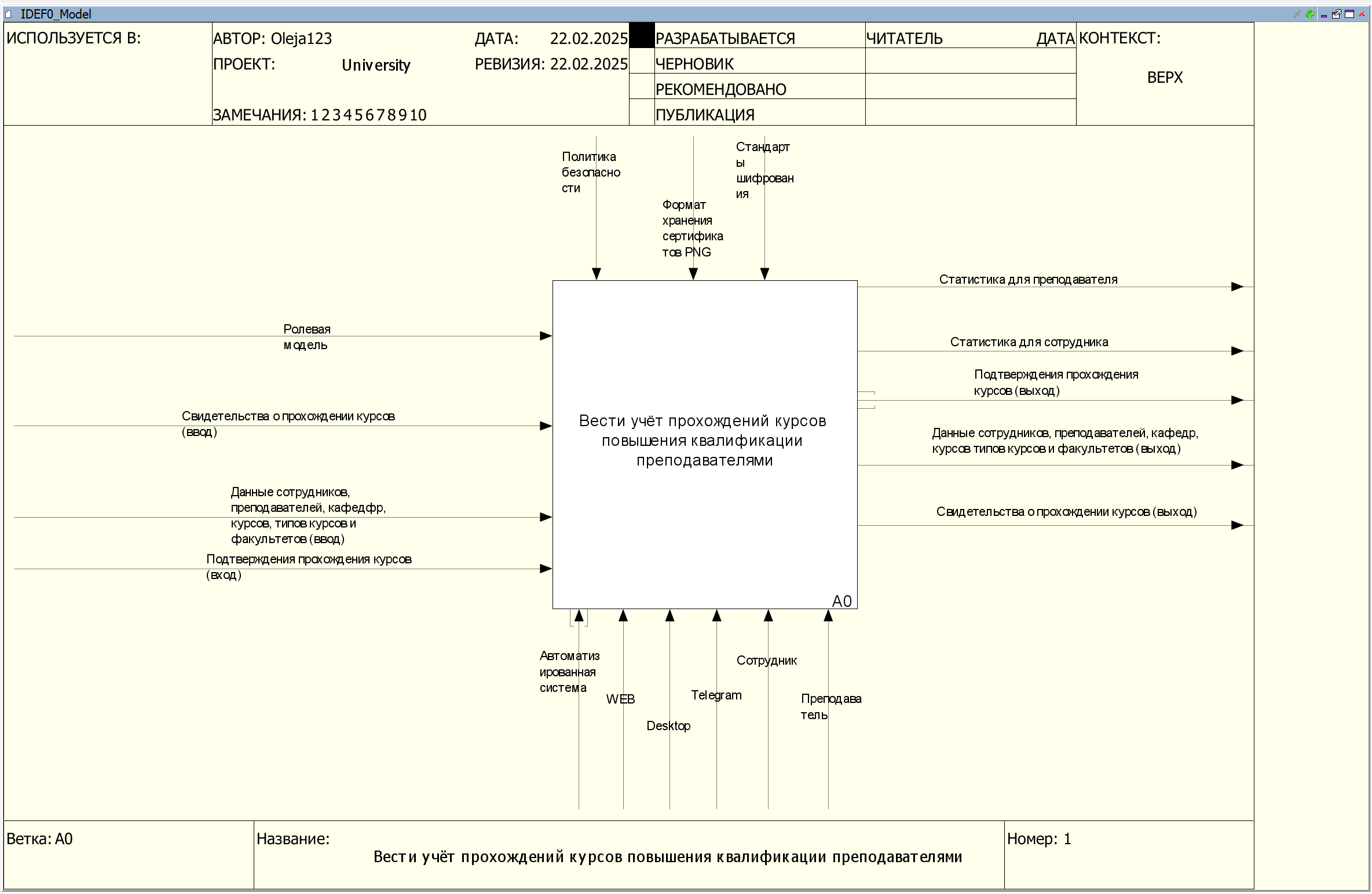


Рисунок 1. Контекстная диаграмма

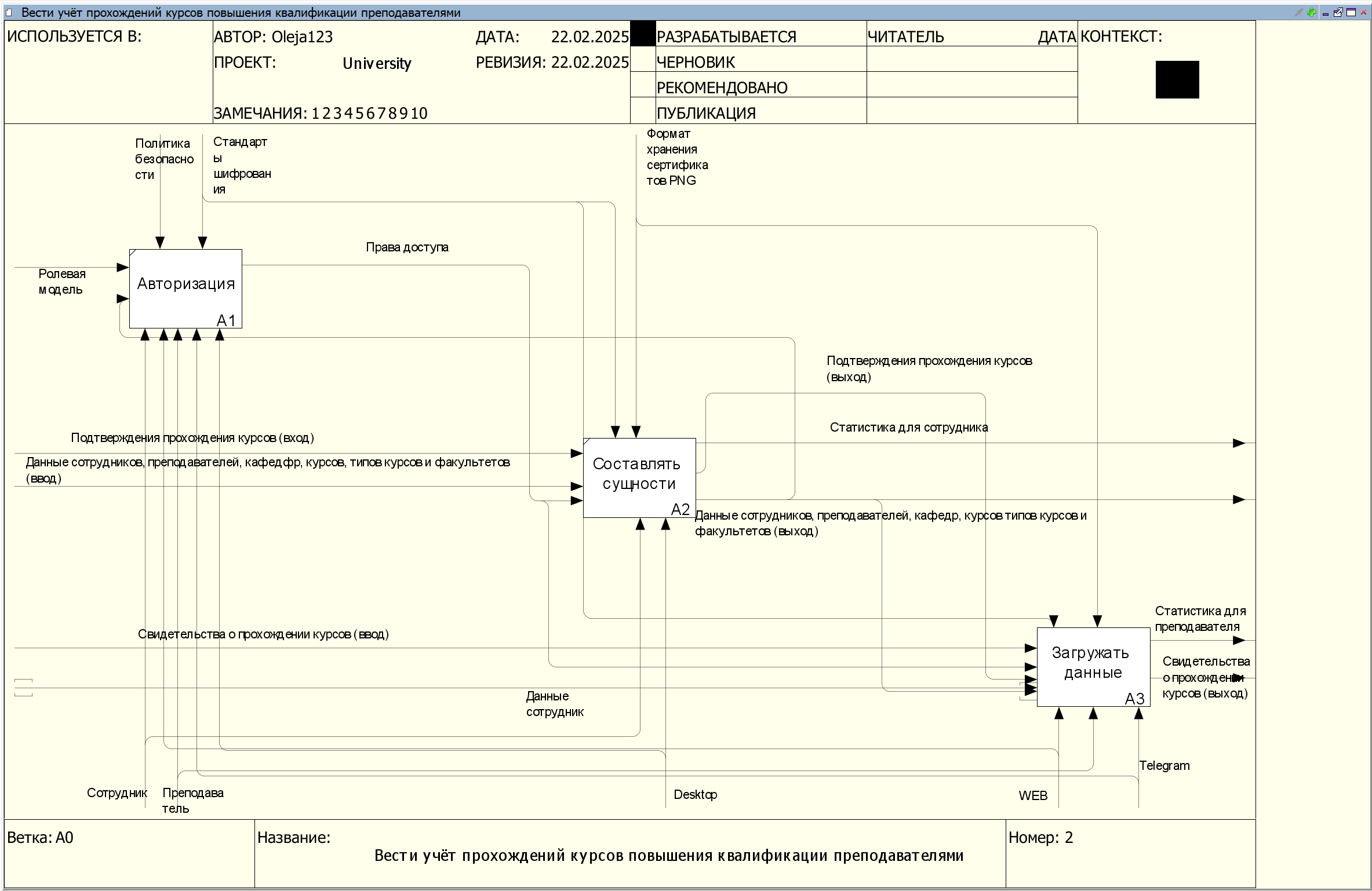


Рисунок 2. Декомпозиция 1-го уровня диаграммы



Рисунок 3. Декомпозиция 2-го уровня диаграммы

**Модель вариантов использования** **(Use Case Diagram)**

Данная диаграмма отражает отношения между актерами и прецедентами. Диаграмма является составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

На рисунке 1 изображена диаграмма вариантов использования для неавторизированного пользователя. Актером на данной диаграмме является неавторизованный пользователь.

Задачи пользователя заключаются в авторизации либо в аккаунт сотрудника, либо в аккаунт преподавателя.

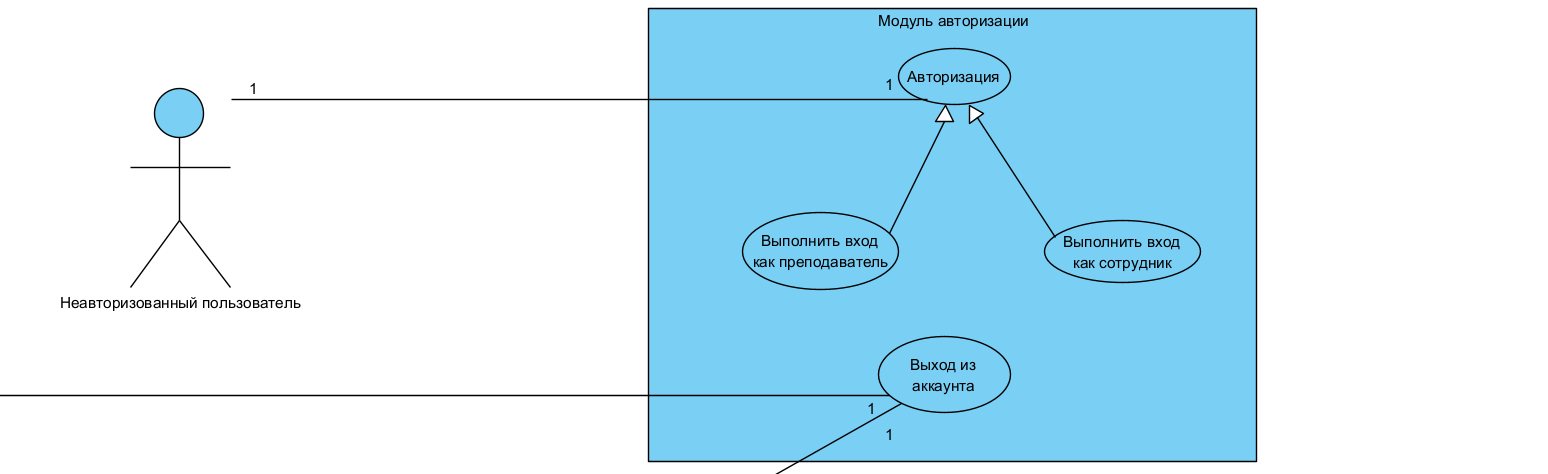


Рисунок 1. Диаграмма вариантов использования для неавторизированного пользователя.

На рисунке 2 изображена диаграмма, которая демонстрирует возможности сотрудника.

На данной диаграмме продемонстрированы следующие действия:

* Выход из аккаунта
* Работа со справочниками (работа с сущностями: преподаватель, факультет, сотрудник, кафедра, тип курсов, курс)
* Подтвердить прохождение курсов для преподавателя (выбрать курс преподавателя, просмотреть загруженный сертификат и принять/не принять его)
* Формировать статистику по преподавателям (отчет по факультетам, отчет по кафедрам, отчет по типу курсов)
* Получение списков прохождения квалификаций

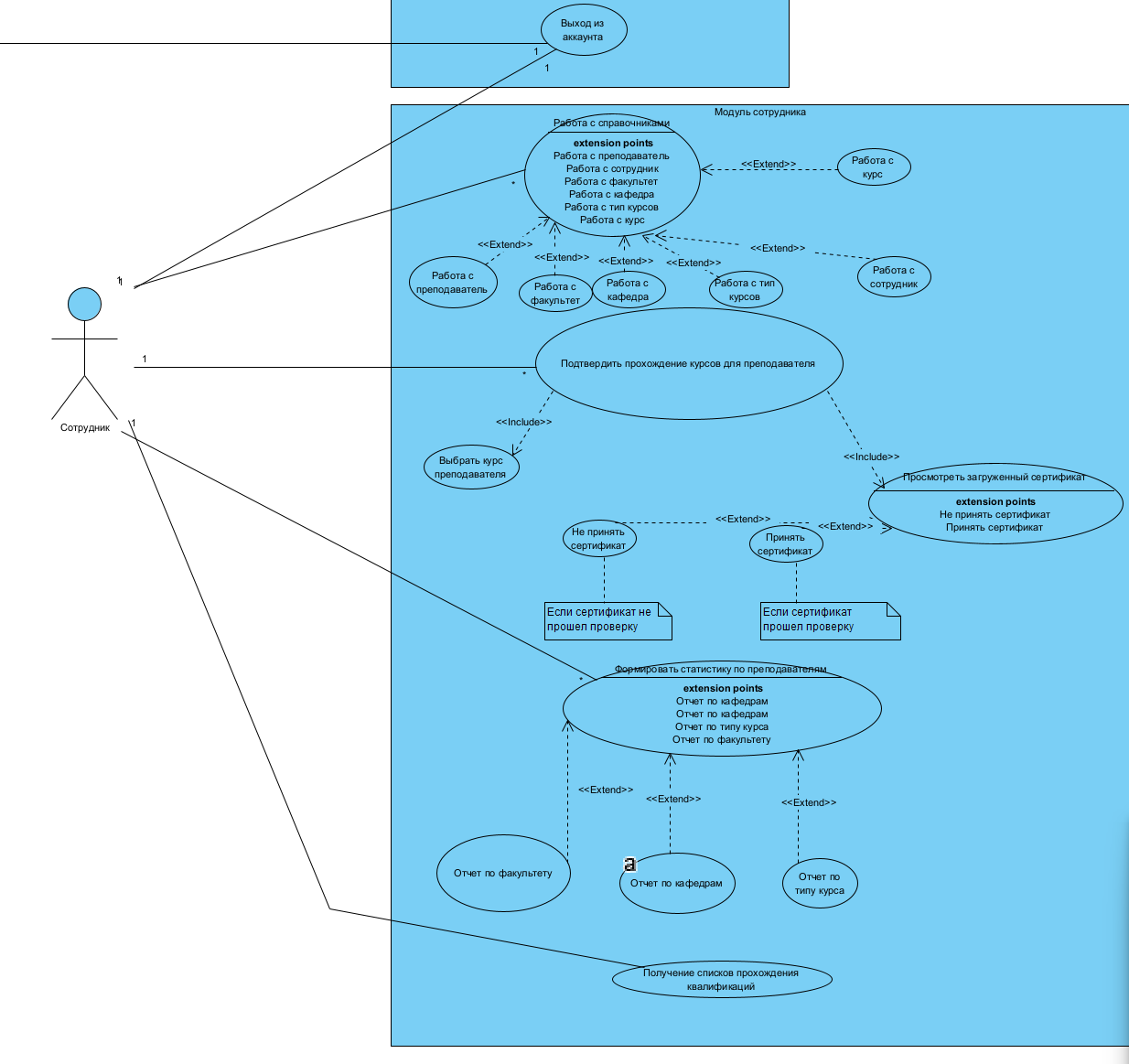


Рисунок 2. Диаграмма вариантов использования для сотрудника.

На рисунке 3 изображена диаграмма, которая демонстрирует возможности преподавателя.

На данной диаграмме продемонстрированы следующие действия:

* Выход из аккаунта
* Просмотр списка курсов (выгрузить сертификат, загрузить сертификат)
* Получить сертификат (выбрать курс)

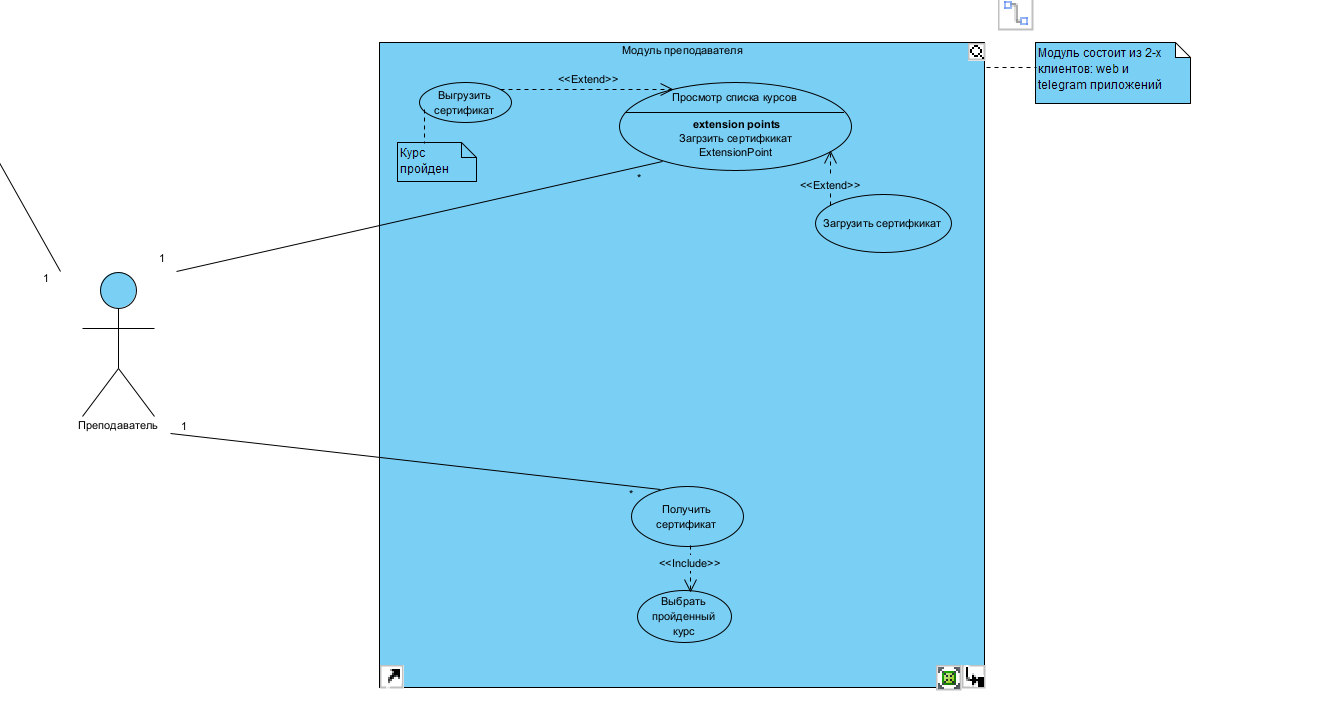


Рисунок 3. Диаграмма вариантов использования для преподавателя.

**Модель классов (Class Diagram)**

Данная диаграмма демонстрирует общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними.

Для начала рассмотрим пакет, содержащий логику клиентского приложения (десктоп-приложение).

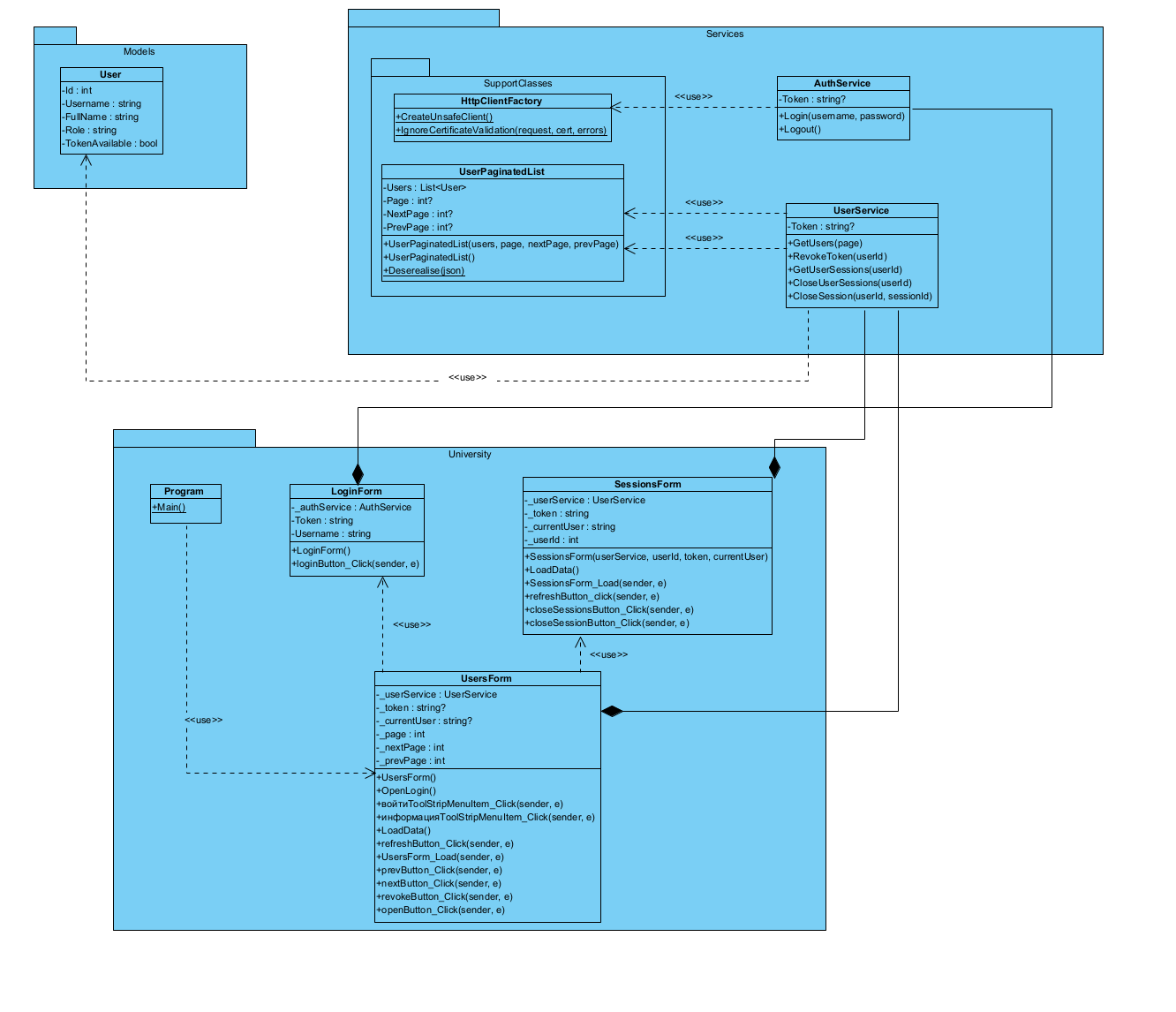


Рисунок 4. Диаграмма классов. Десктоп-приложение.

На диаграмме представлены 3 пакета. Models – модели, с которыми работает клиентское приложение, Services – бизнес-логика, работающая с моделями, University – пакет, содержащий логику форм.

Рассмотрим пакет Models.

В данном пакете находится модель User, которая составляется из данных получаемых с апи бэкенда.

Рассмотрим пакет Services

В данном пакете находятся следующие объекты:

* Пакет SupportClasses: классы HttpClientFactory – класс для создания http клиента, UserPaginatedList – класс, представляющий класс списка с пагинацией
* AuthService – логика работы авторизации
* UserService – логика работы пользователя, включая работу с сессиями и токенами

Рассмотрим диаграмму классов для телеграм-бота.

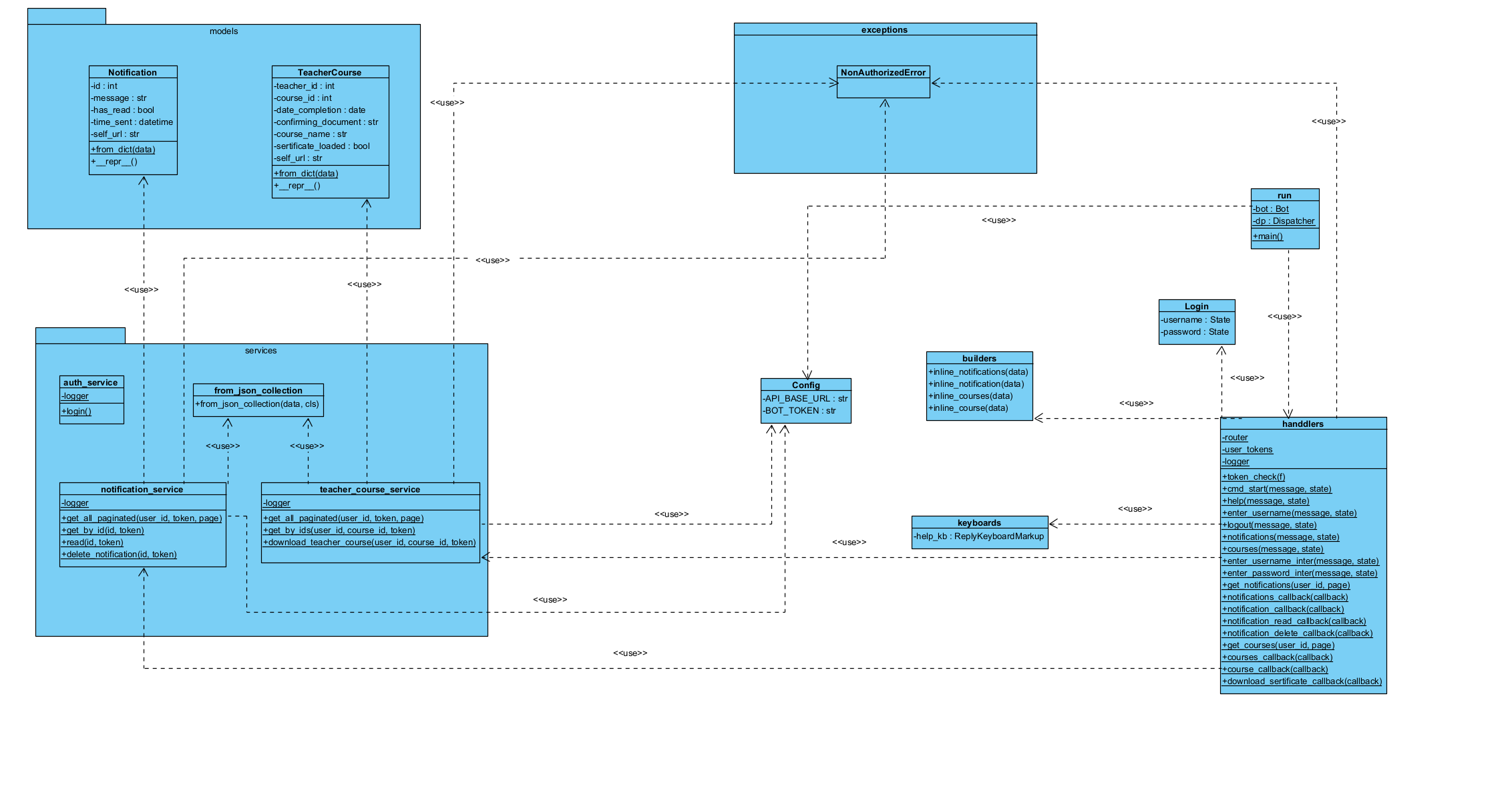


Рисунок 5. Диаграмма классов. Телеграм-бот.

На диаграмме представлены 3 пакета. models – модели, с которыми работает клиентское приложение, services – бизнес-логика, работающая с моделями, exceptions – собственные исключения.

Рассмотрим пакет models.

В данном пакете находятся следующие классы:

* Notification – модель уведомления
* TeacherCourse – модель пройденного преподавателем курса

Рассмотрим пакет services.

В данном пакете находятся следующие классы:

* auth\_service – логика авторизации
* from\_json\_collection – вспомогательный класс для получения коллекции из json
* notification\_service – логика уведомлений
* teacher\_course\_service – логика пройденных пользователем курсов

Рассмотрим пакет exceptions.

В данном пакете находятся следующие классы:

* NonAuthorizedError – ошибка авторизации

Рассмотрим остальные классы приложения:

* Config – конфигурационный класс приложения
* builders – класс для построения разметок
* keyboards – класс для клавиатур
* Login – класс для построения последовательности авторизации
* handlers – класс с обработчиками сообщений и callback’ов
* run – точка входа

Рассмотрим диаграмму классов для серверного приложения.

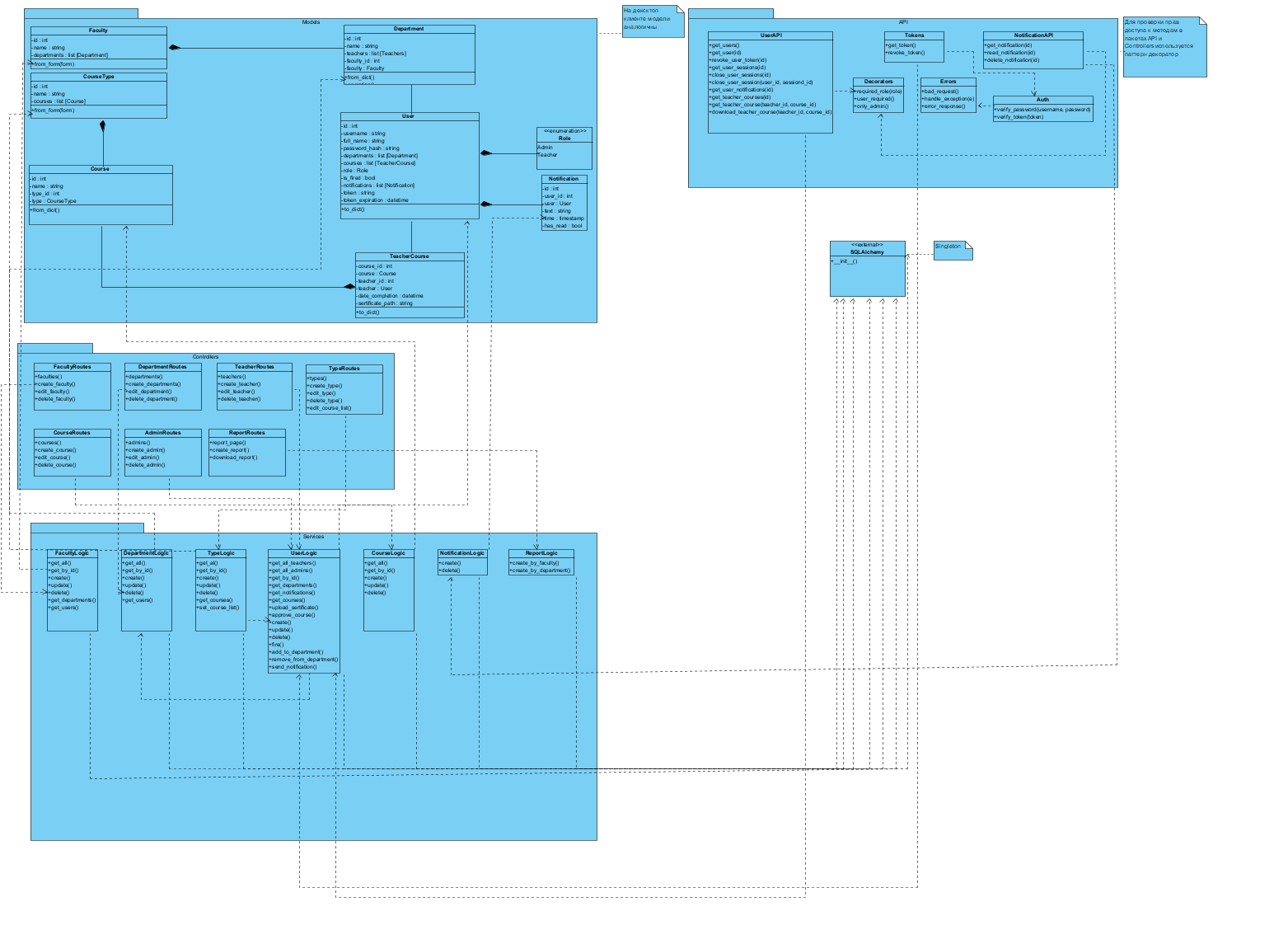


Рисунок 6. Диаграмма классов. Серверное приложение.

На диаграмме представлены 4 пакета: models, controllers, services, api.

Рассмотрим пакет Models.

В данном пакете находятся следующие классы:

* Faculty – модель факультета
* CourseType – модель типа курсов
* Course – модель курса
* Department – модель кафедры
* User – модель пользователя
* TeacherCourse – модель курса преподавателя
* Notification – модель уведомления

Рассмотрим пакет Controllers.

В данном пакете находятся следующие классы:

* FacultyRoutes – контроллер для факультетов
* DepartmentRoutes – контроллер для кафедр
* TeacherRoutes – контроллер для преподвателей
* CourseTypeRoutes – контроллер для типов курсов
* CourseRoutes – контроллер для курсов
* AdminRoutes – контроллер для сотрудников
* ReportRoutes – контроллер для отчетов

Рассмотрим пакет Services.

В данном пакете находятся следующие классы:

* FacultyLogic – логика факультетов
* DepartmentLogic – логика кафедры
* TypeLogic – логика типов курсов
* UserLogic – логика пользователя
* CourseLogic – логика курсов
* NotificationLogic – логика уведомлений
* ReportLogic – логика отчетов

Рассмотрим пакет API

В данном пакете находятся следующие классы:

* UserAPI – контроллер для запросов к пользователю
* NotificationAPI – контроллер для запросов к уведомлениям
* Tokens – класс для работы с токенами
* Decorators – класс, содержащий декораторы
* Errors – класс, содержащий ошибки

**Модель последовательности действий (Sequence Diagram)**

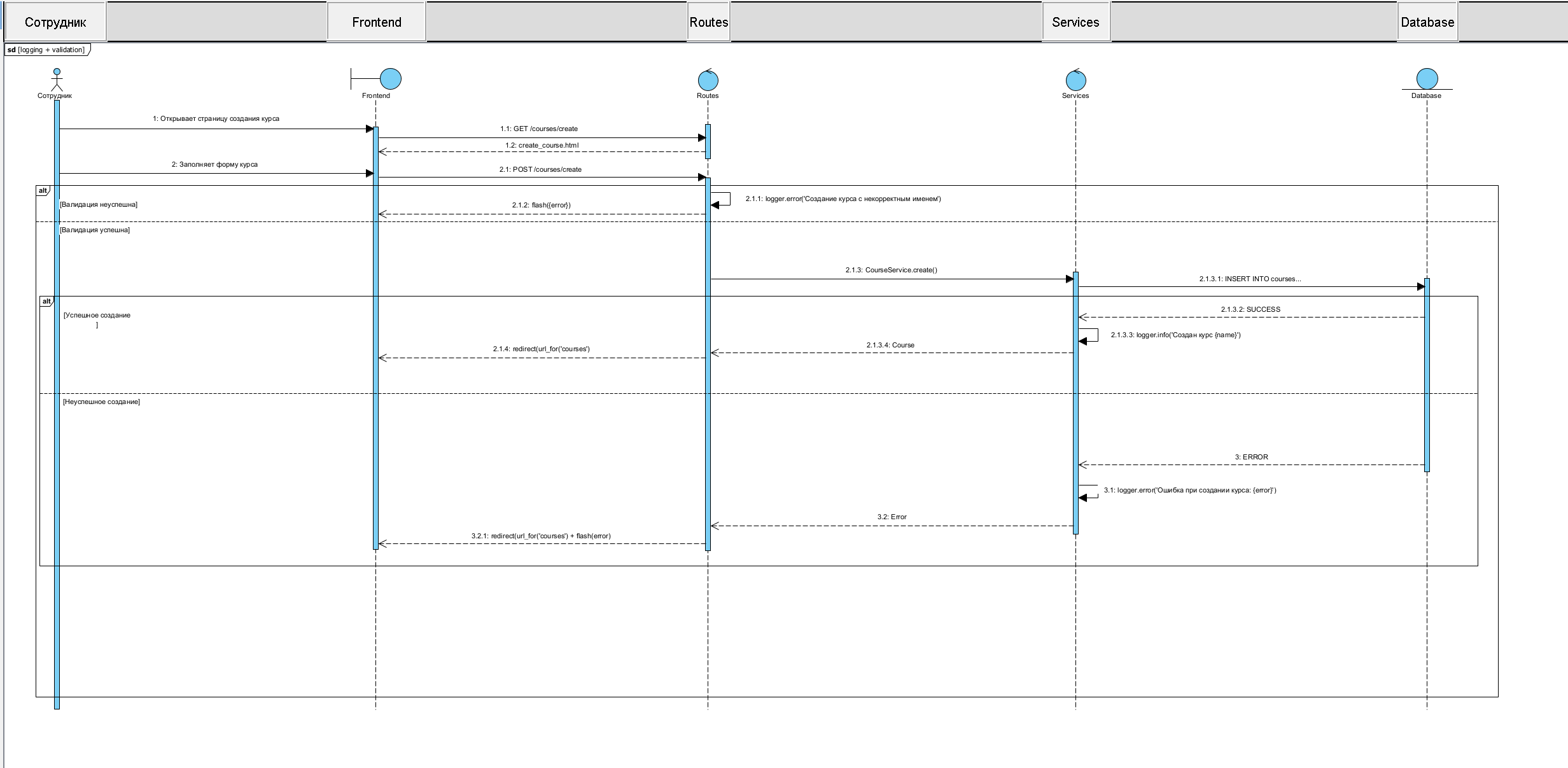


Рисунок 7. Диаграмма последовательности 1.

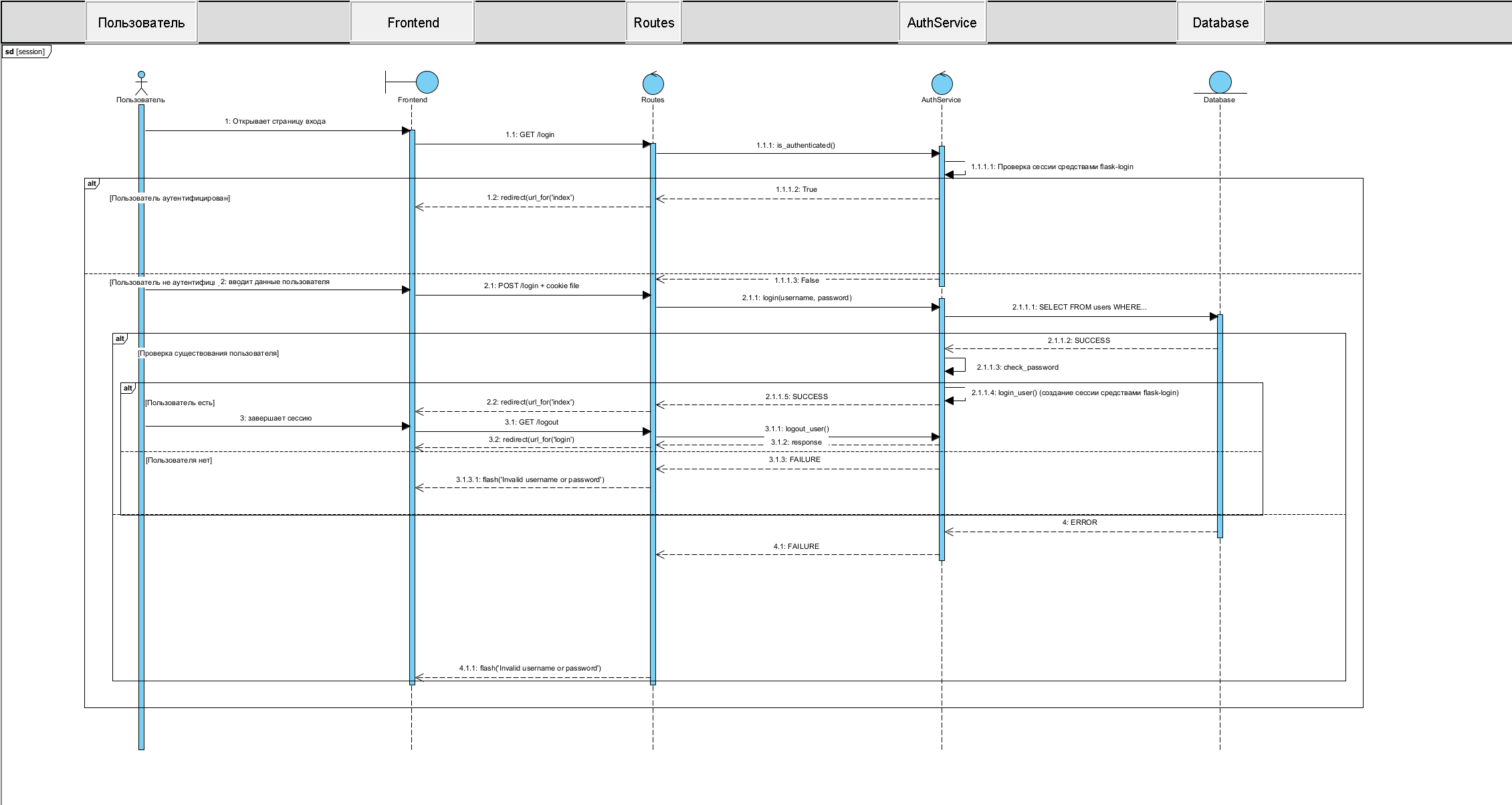


Рисунок 8. Диаграмма последовательности 2.

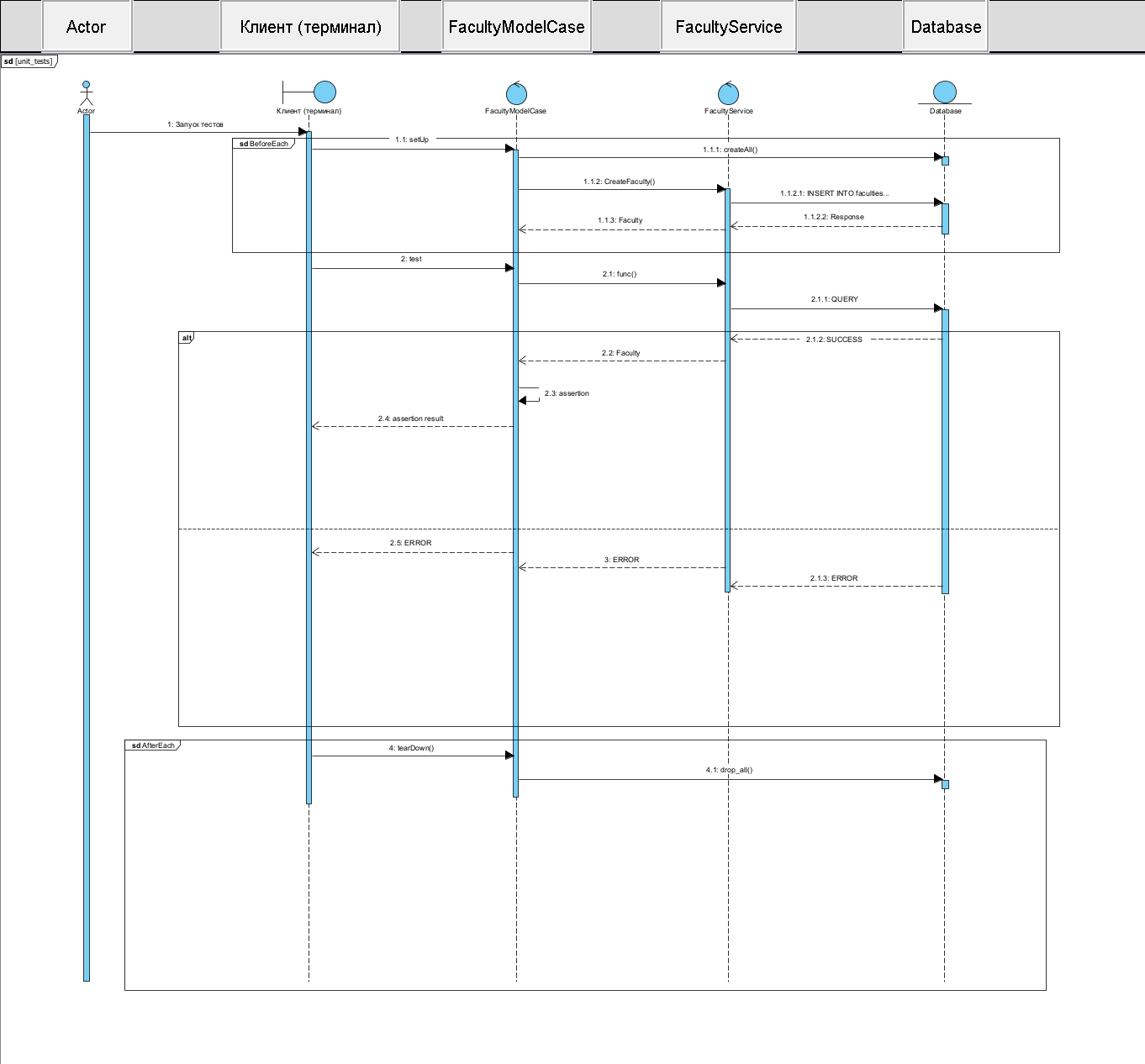


Рисунок 9. Диаграмма последовательности 3.

Рассмотрим на единой временной оси жизненный цикл заявленной сквозной функциональности.

На рисунке 7 отображено логирование, валидация и обработка ошибок.

На рисунке 8 отображена работа сессий.

На рисунке 9 отображено модульное тестирование.

**Модель компонентов (Component Diagram)**

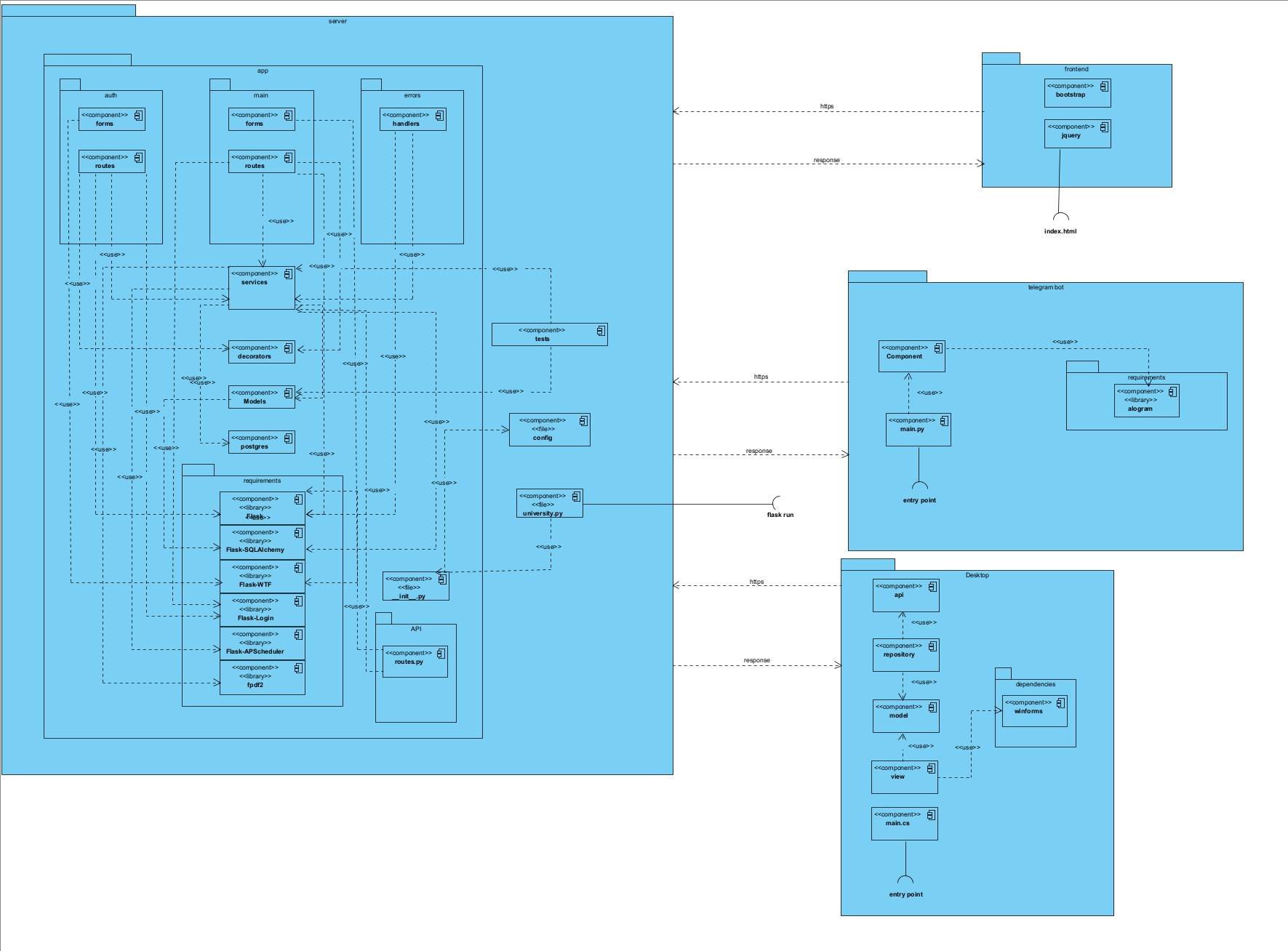
****

Рисунок 10. Диаграмма компонент

**Модель развёртывания (Deployment Diagram)**

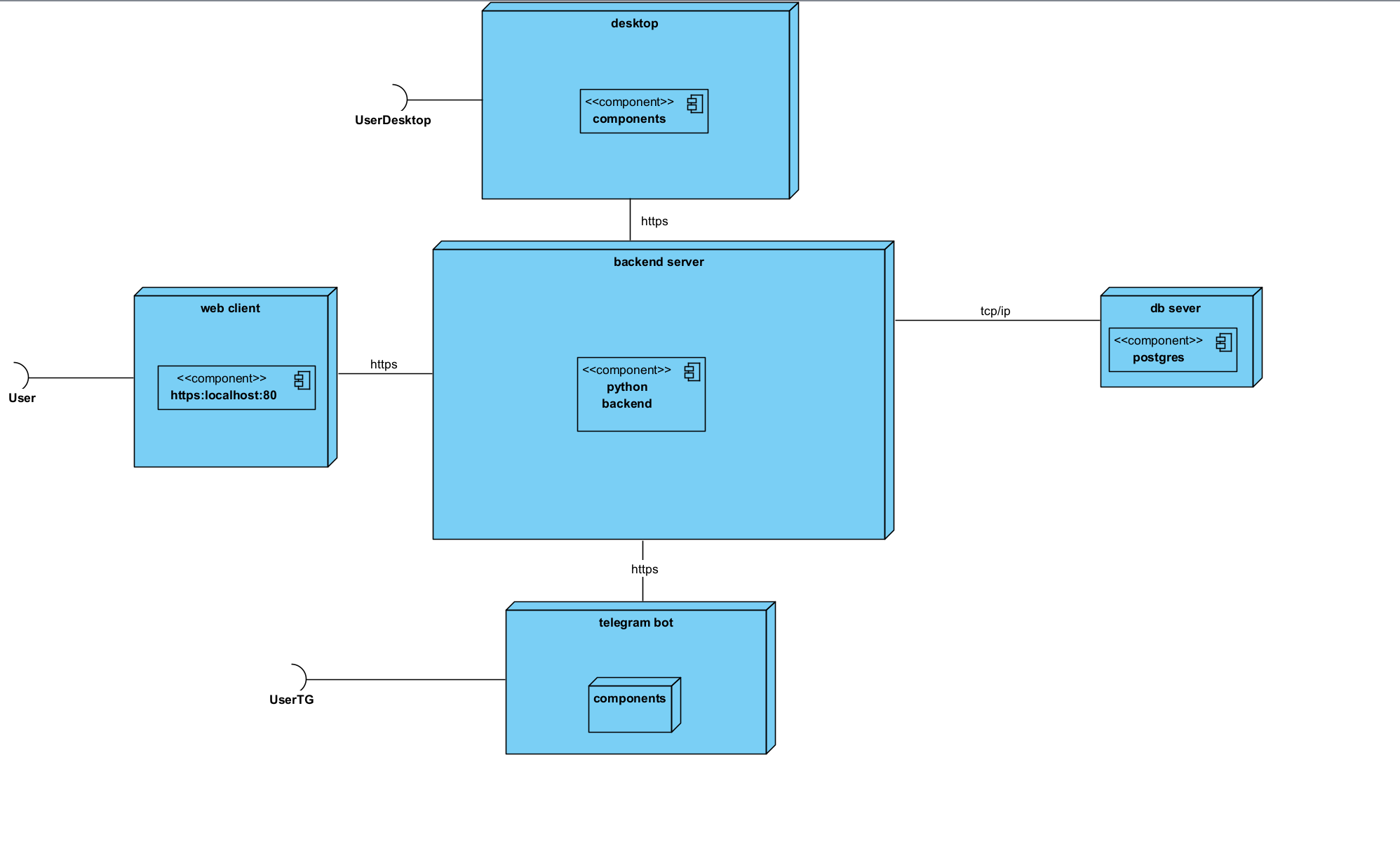
****

Рисунок 11. Диаграмма развертывания

**Генерация кода по проекту**

Генерация кода была выполнена для диаграммы классов серверного приложения. Пример генерации кода:

package Services;  
  
public class ReportLogic {  
  
 public void create\_by\_faculty() {  
 // *TODO - implement ReportLogic.create\_by\_faculty* throw new UnsupportedOperationException();  
 }  
  
 public void create\_by\_department() {  
 // *TODO - implement ReportLogic.create\_by\_department* throw new UnsupportedOperationException();  
 }  
  
}

Отличие сгенерированного кода по диаграмме классов и кодом в программе при реализации заключается в:

* Неполной сигнатуре функций
* Язык генерации – Java. Язык реализации python (была осуществлена смена стека технологий).

# Реализация приложения

**Описание технологий разработки приложения**

Языки программирования – python, c#

Используемые СУБД – postgresql, redis, sqlite Для взаимодействия с СУБД средствами языка программирования используются библиотеки sqlalchemy и redis Для автоматического управления SQL-миграциями используется менеджер миграций из библиотеки alembic

Для логгирования средствами языка программирования используется стандартная библиотека logging для python

Для генерации отчётов в формате PDF используется библиотека fpdf2

Для реализации маршрутизации используется фреймворк flask

Для реализации шаблонизации html страниц на сервере используется библиотека jinja

Для отправки данных с форм используется библиотека Flask-WTF

Для работы с формами в десктоп приложении используются windows forms

**Демонстрация приложения**

В качестве демонстрации работы приложения приводится основной сценарий работы по контрольному примеру.

Описание основных экранов программы.

На рис. 1 изображена страница входа для неавторизованного пользователя.

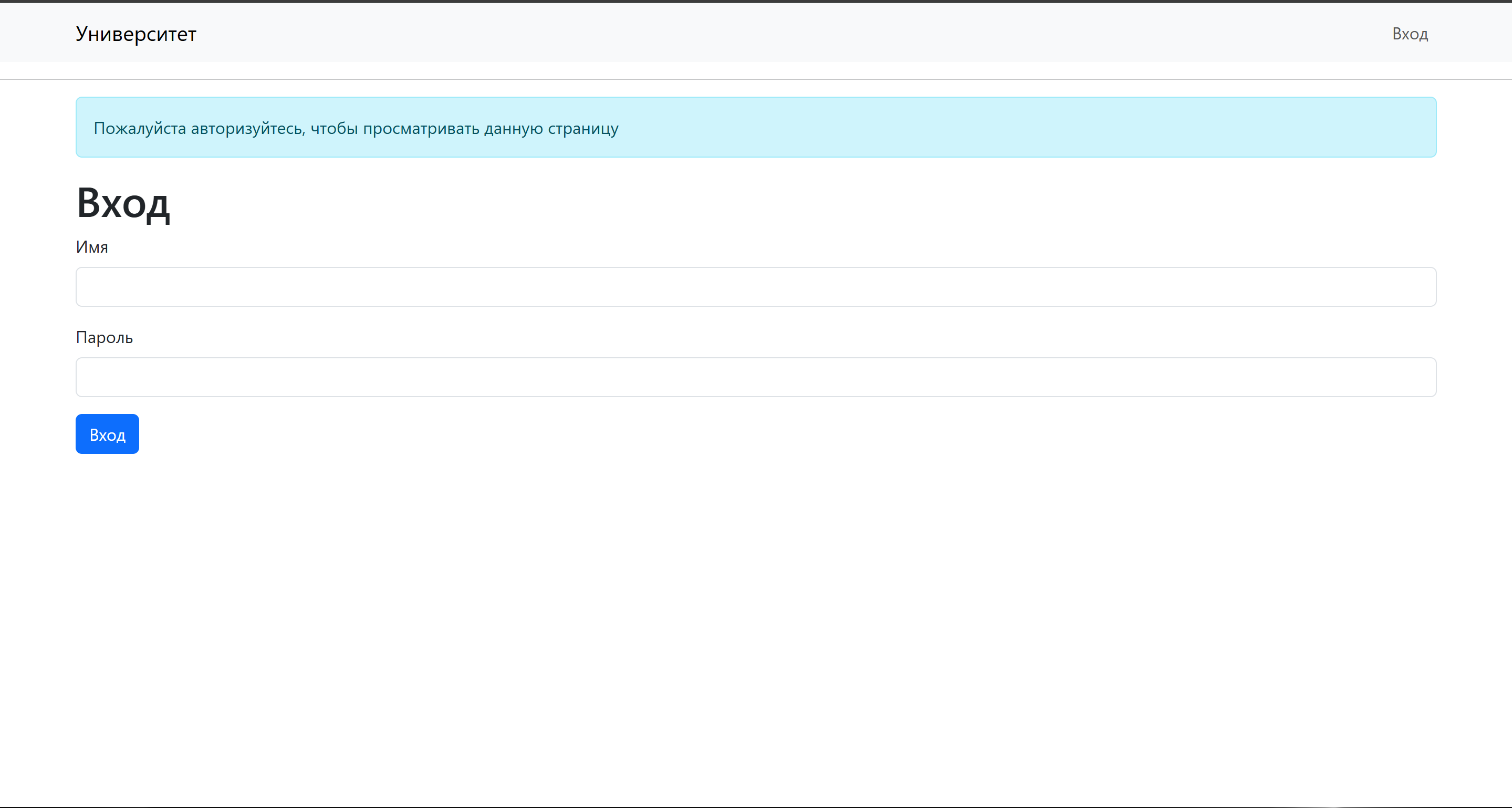
****

Рисунок 1. Страница авторизации

На рис. 2 изображена главная страница для сотрудника

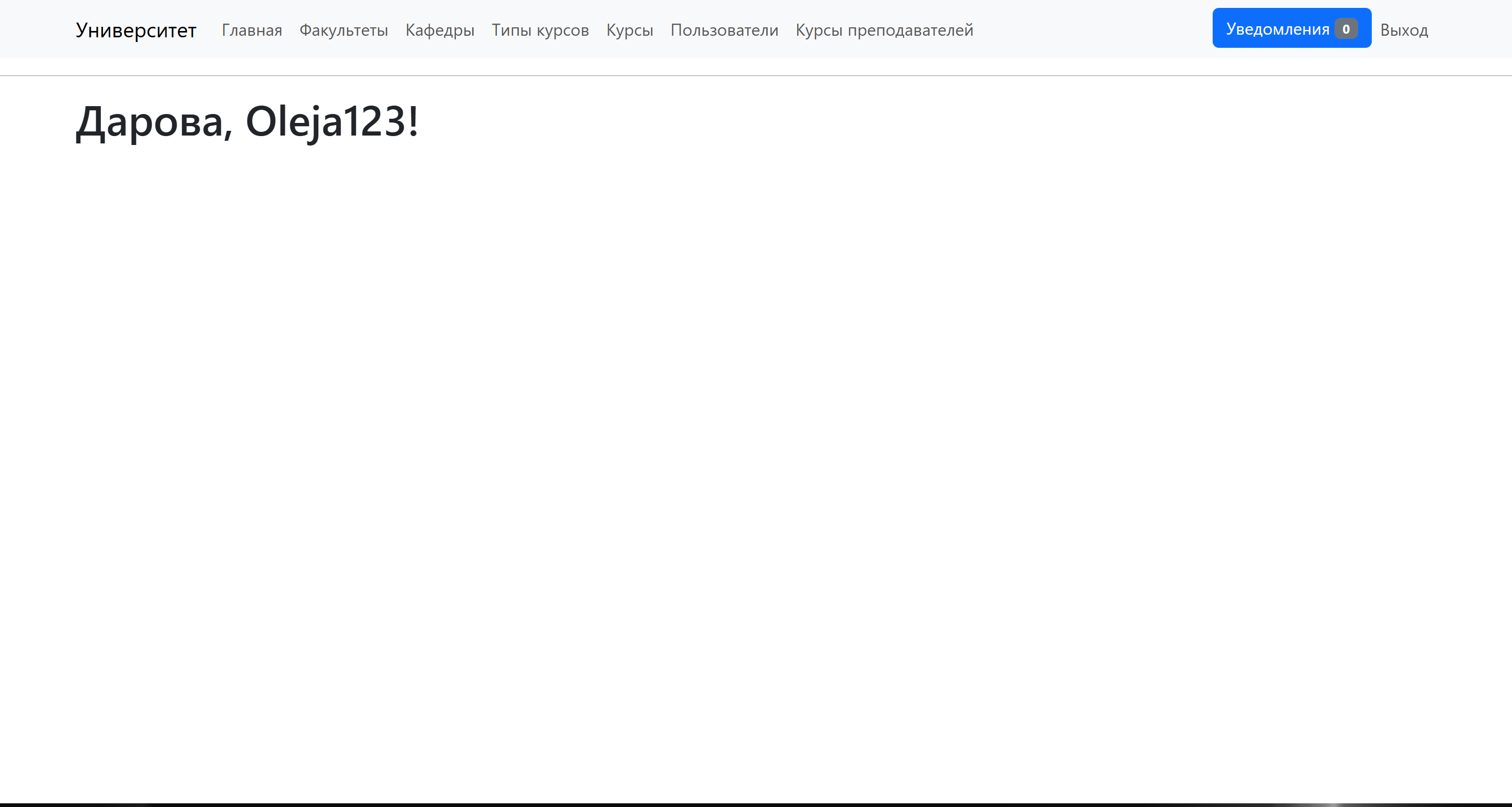
****

Рисунок 2. Главная страница для авторизованного сотрудника

На рис. 3 изображена страница факультетов

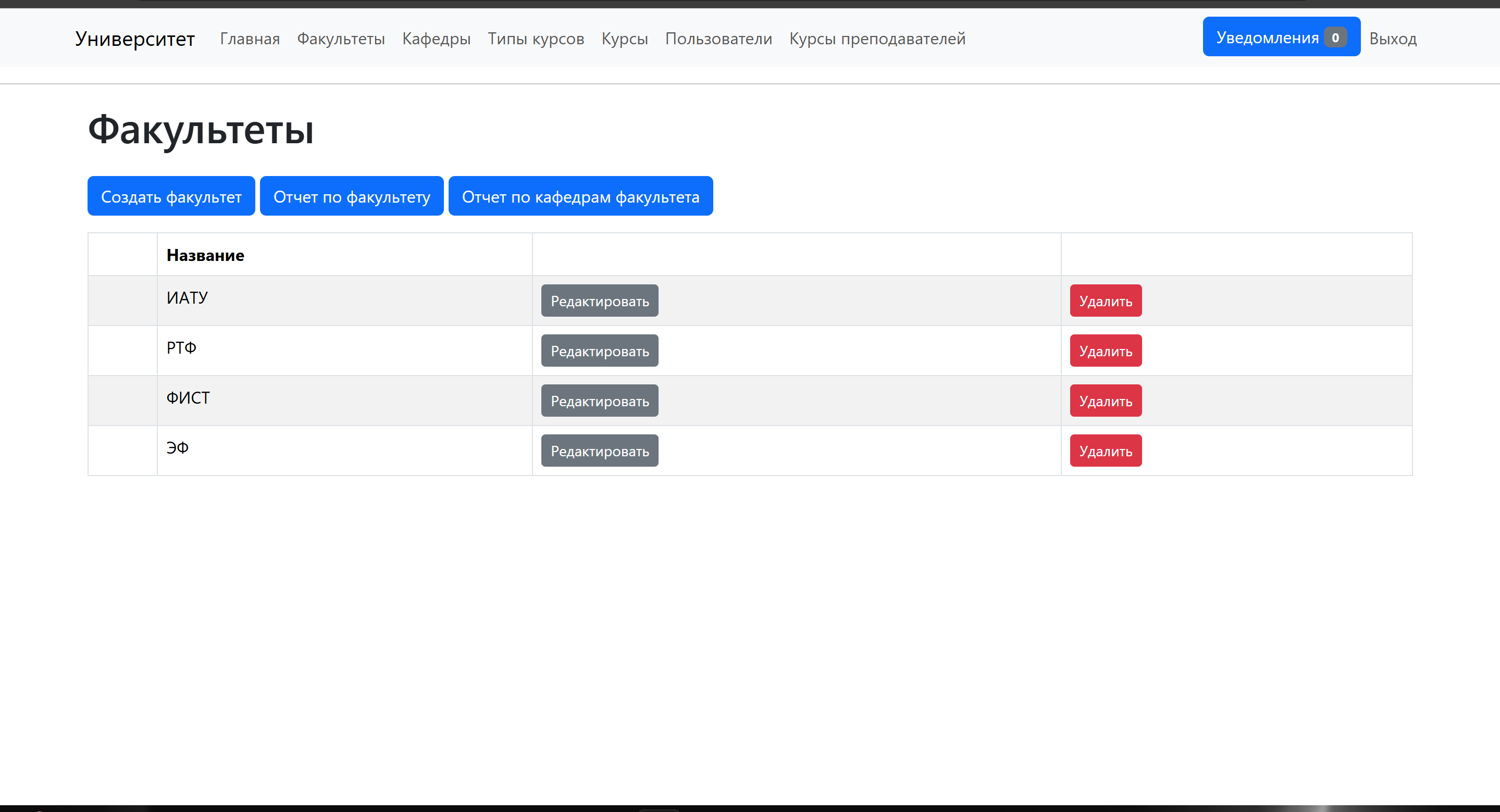
****

Рисунок 3. Страница факультетов

На рис. 4 изображена страница создания и редактирования факультета

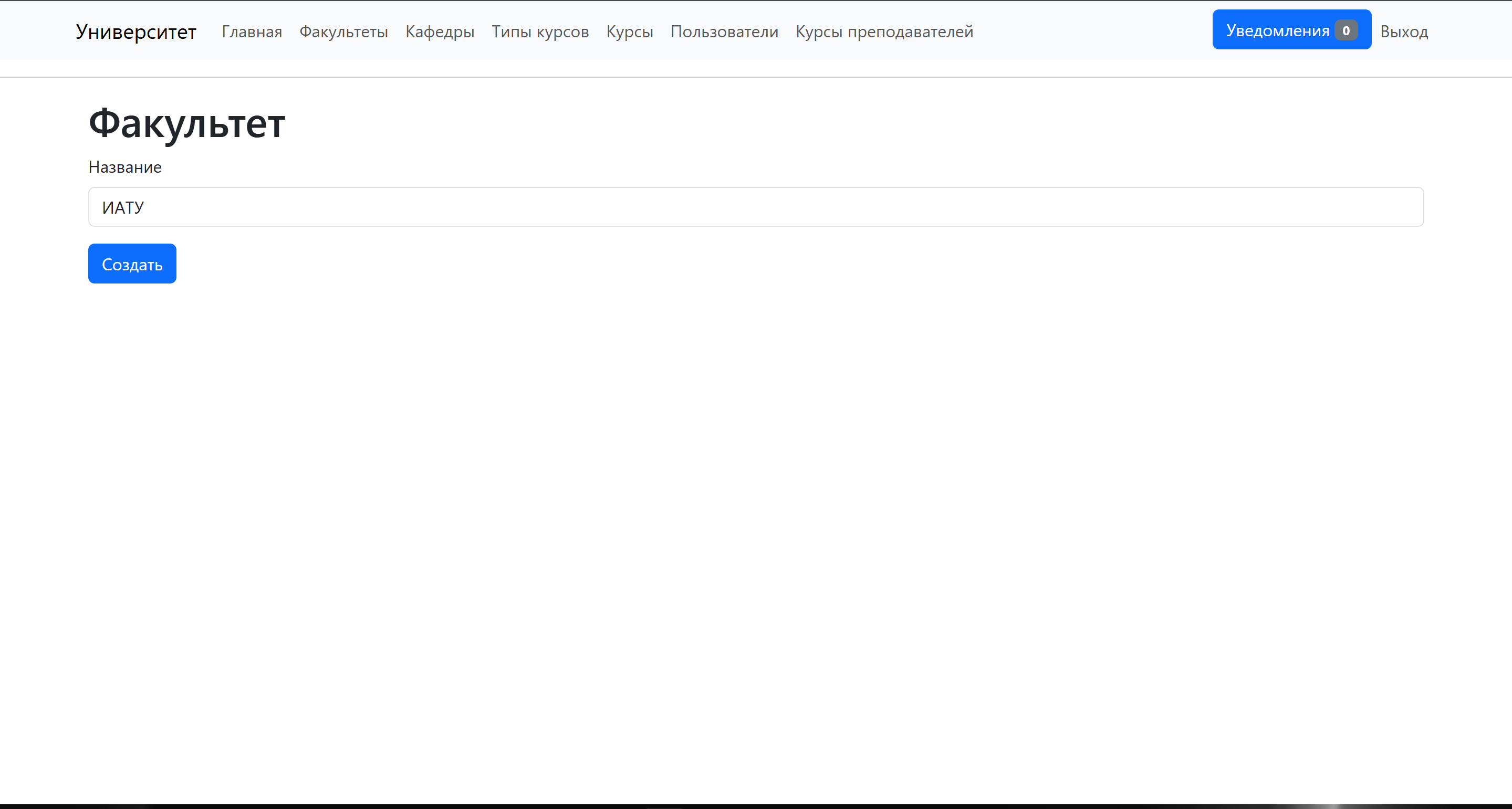
****

Рисунок 4. Страница создания и редактирования факультета

На рис. 5 изображена страница отчета по факультетам

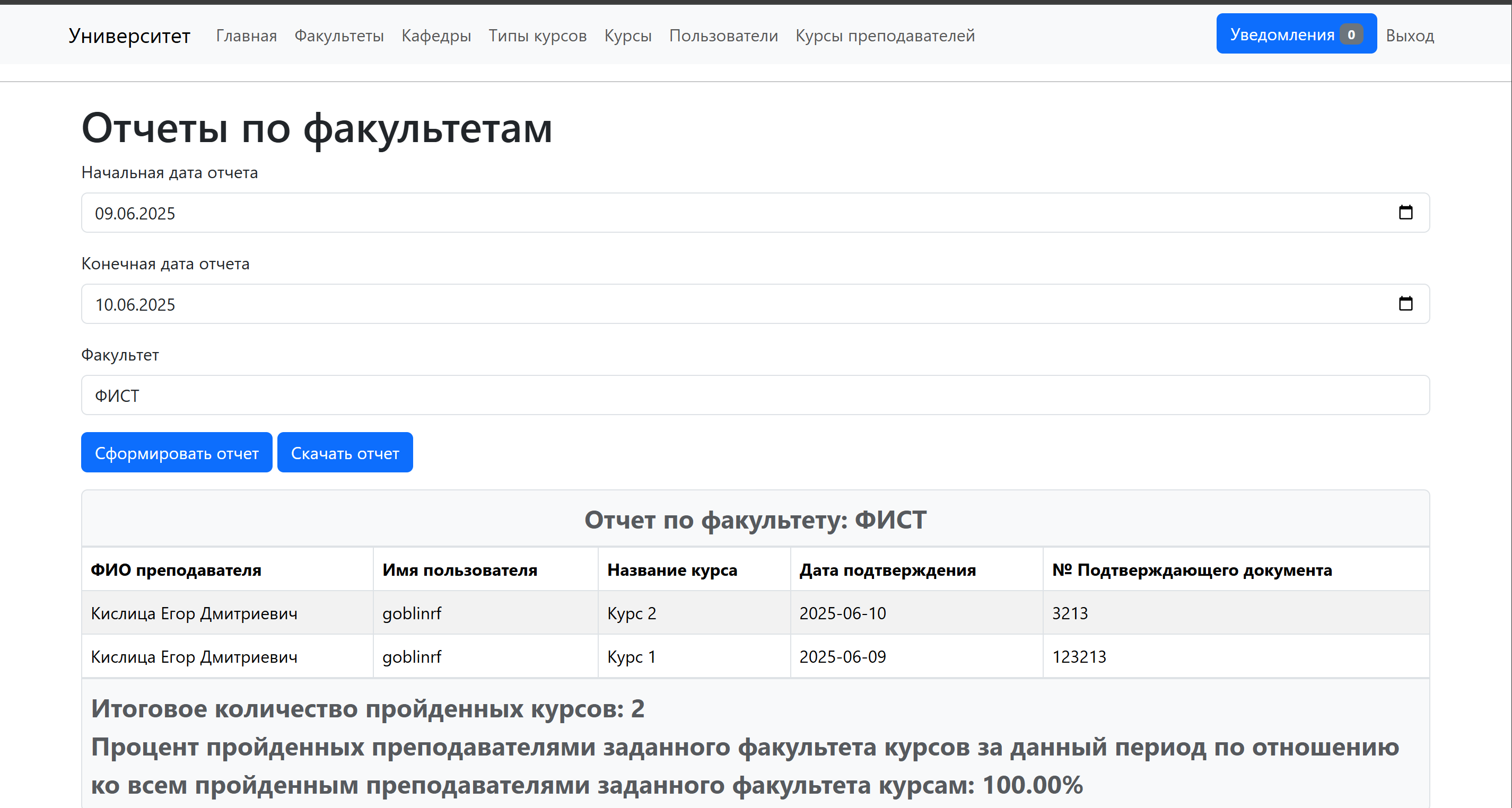
****

Рисунок 5. Страница отчета по факультетам

На рис. 6 изображена страница отчета по кафедрам факультета

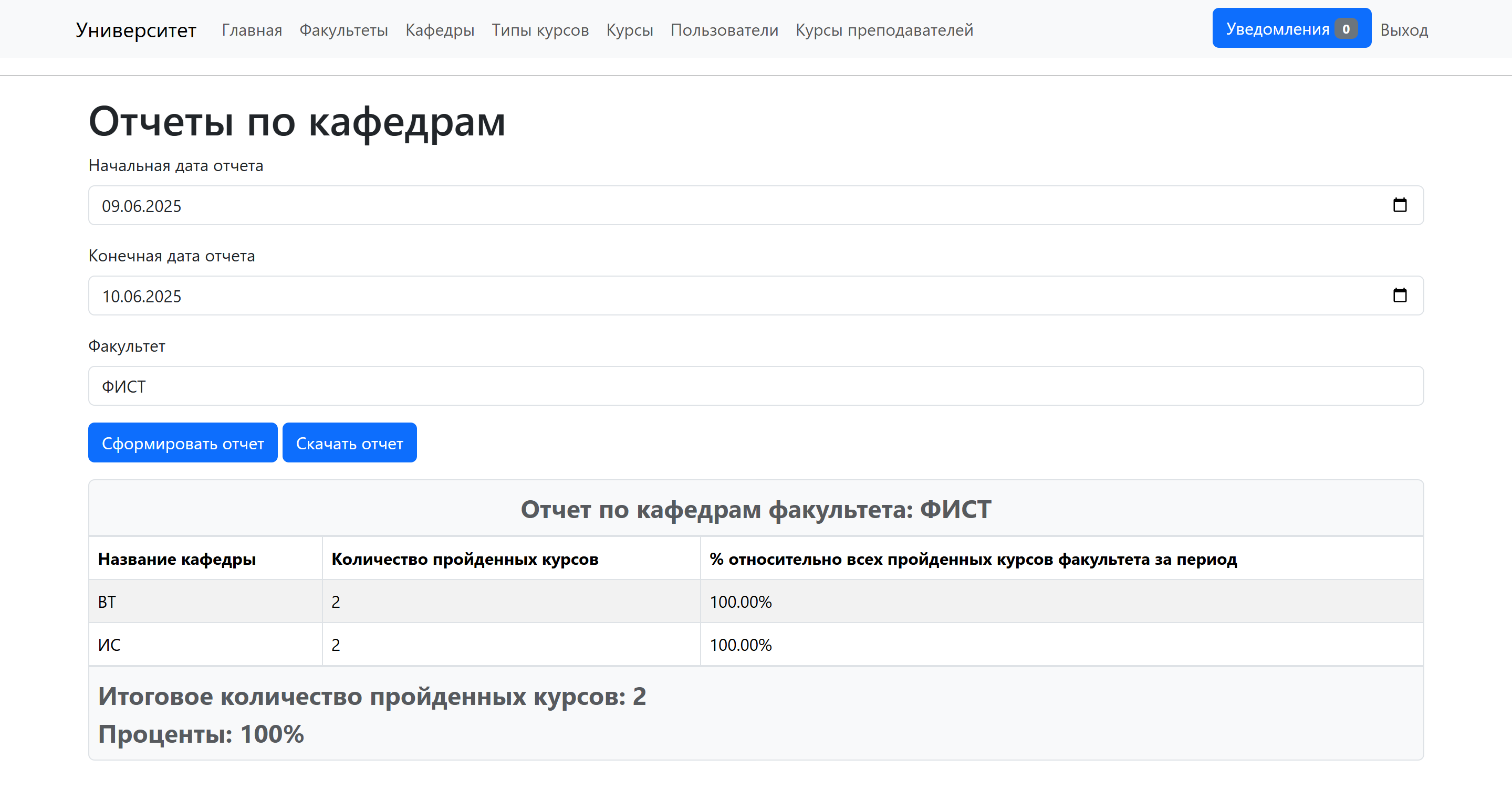
****

Рисунок 6. Страница отчета по кафедрам факультета

На рис. 7 изображена страница кафедр

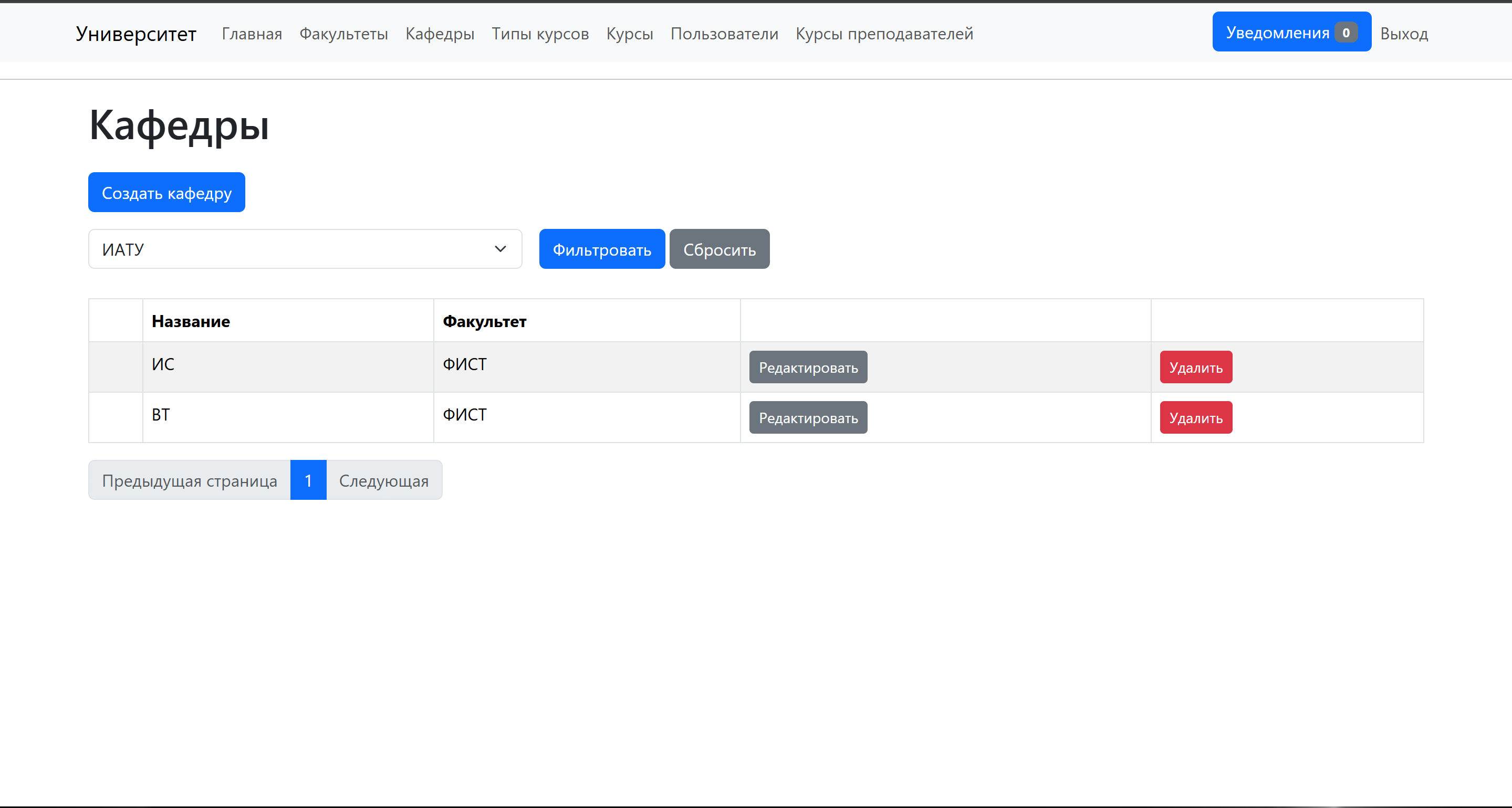
****

Рисунок 7. Страница кафедр

На рис. 8 изображена страница создания и редактирования кафедры

****

Рисунок 8. Страница создания и редактирования кафедры

На рис. 9 изображена страница типов курсов

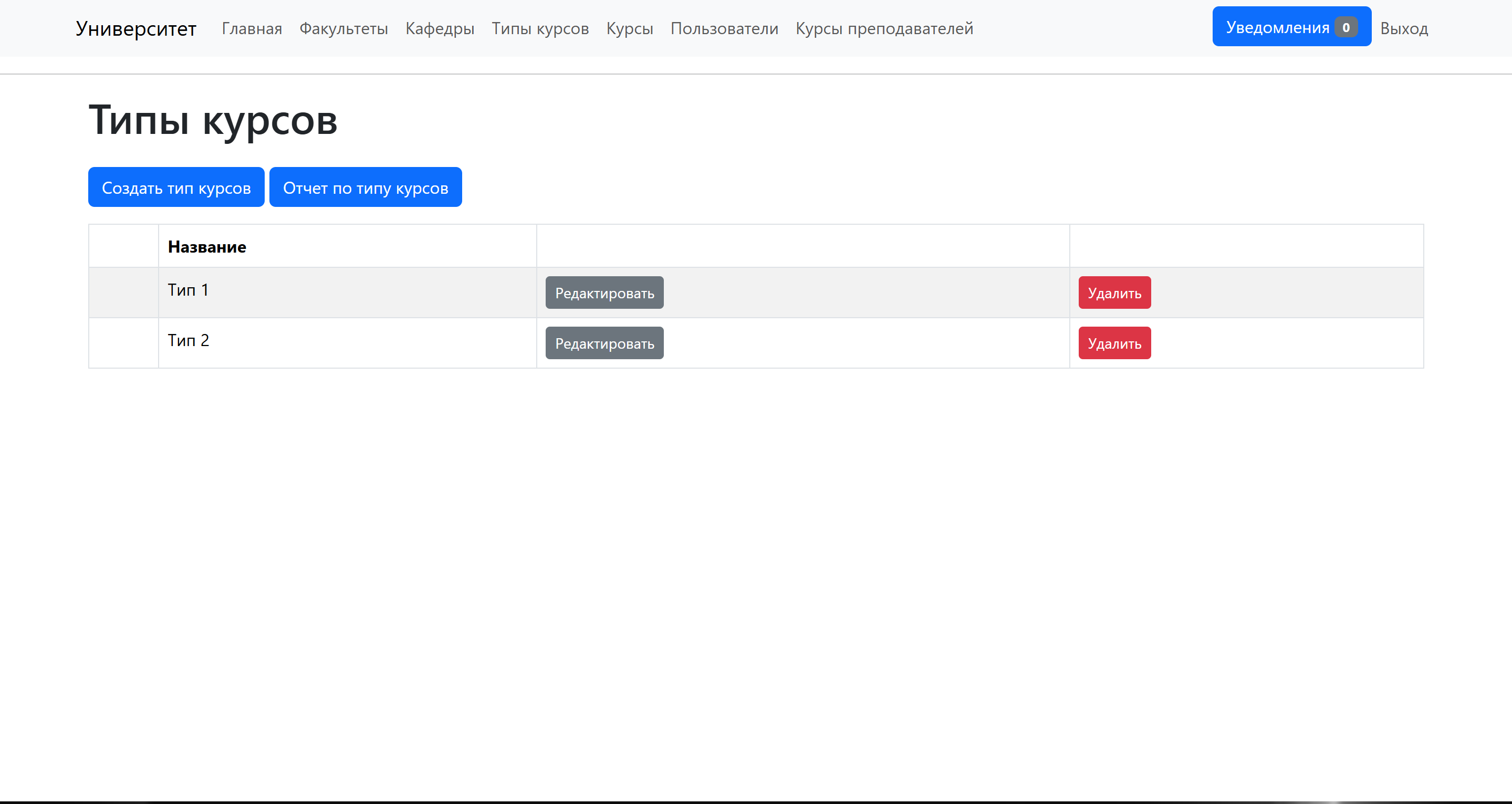
****

Рисунок 9. Страница типов курсов

На рис. 10 изображена страница создания и редактирования типа курсов

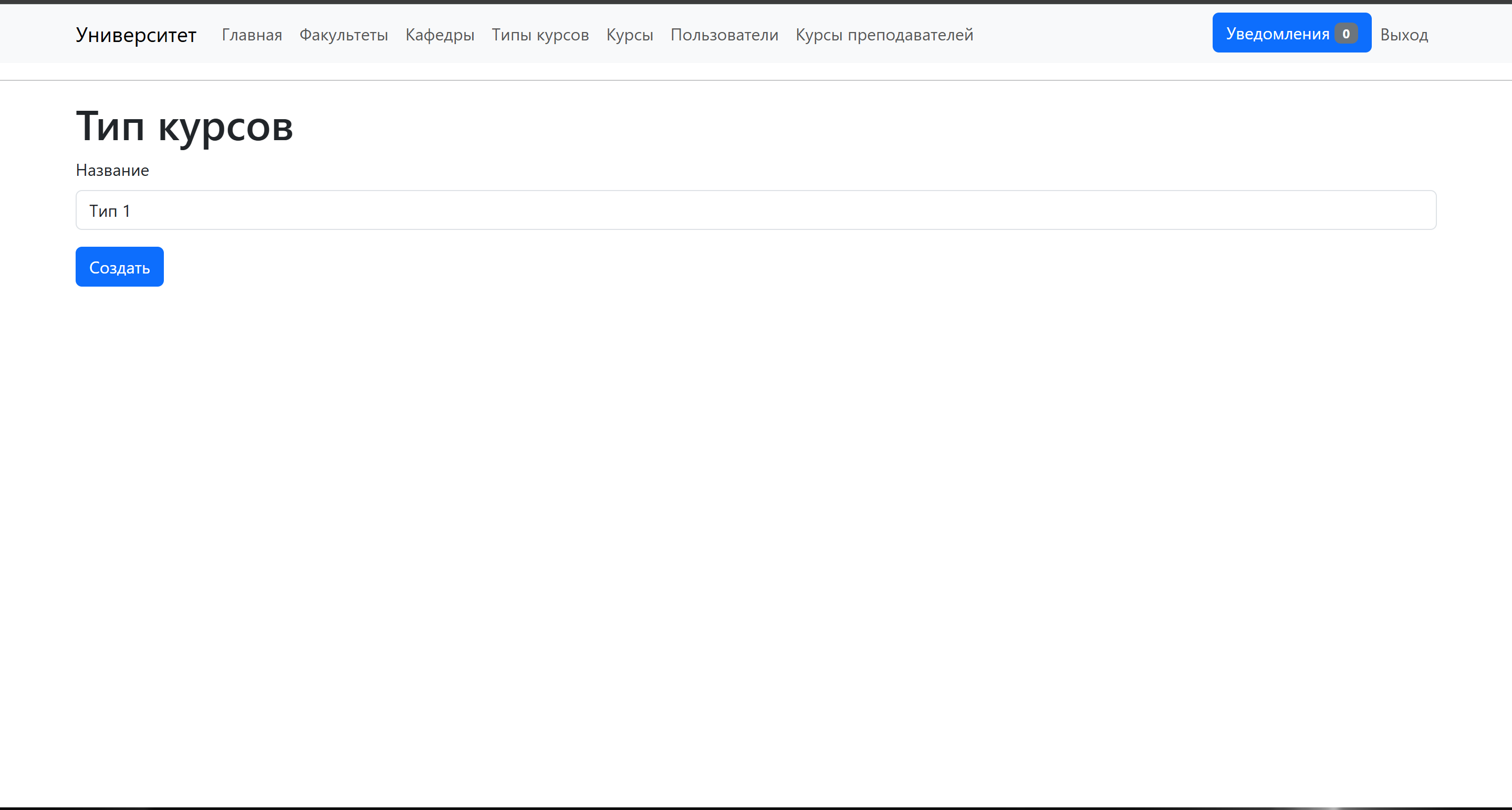
****

Рисунок 10. Страница создания и редактирования типа курсов

На рис. 11 изображена страница отчета по типам курсов

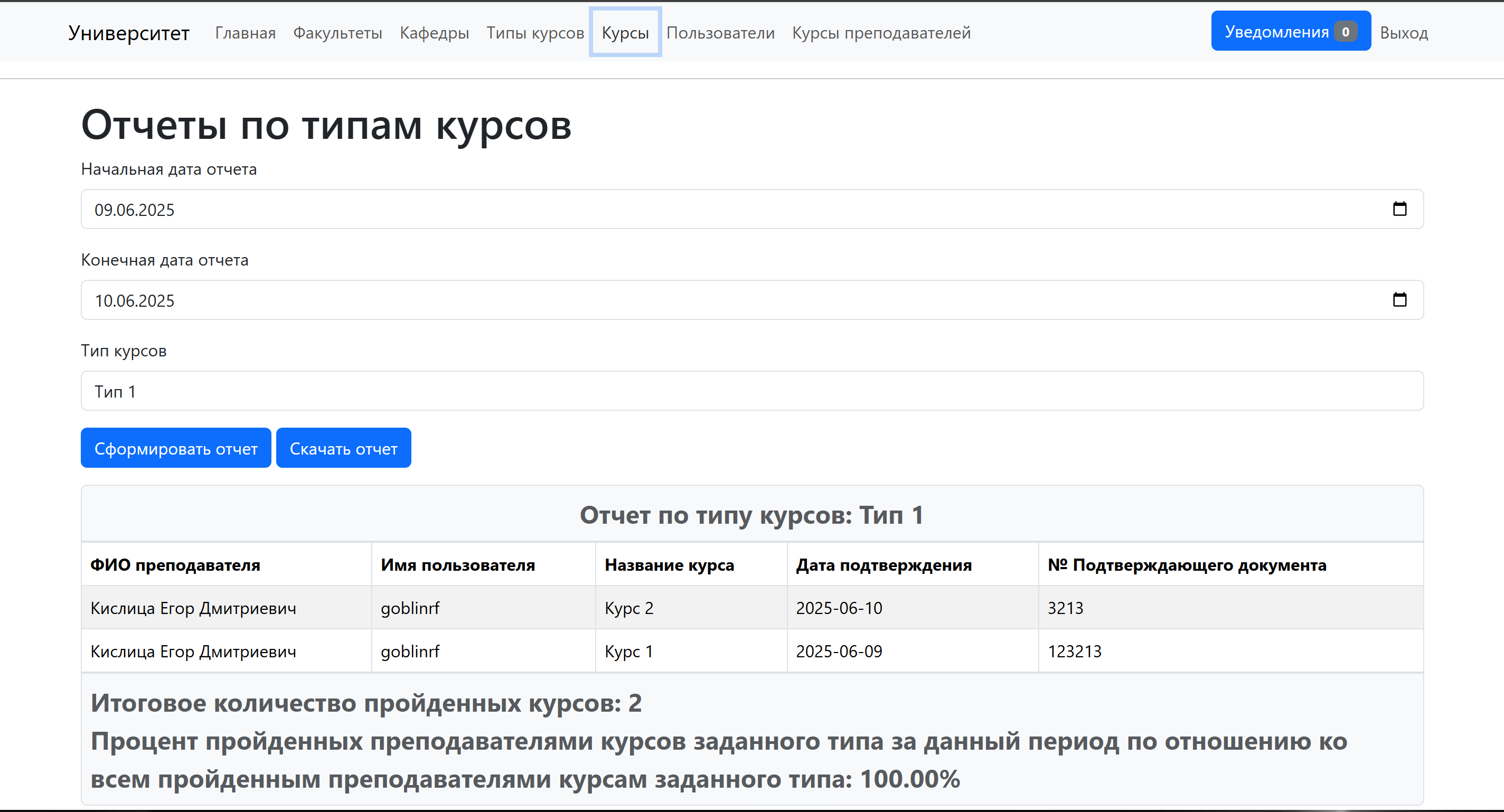
****

Рисунок 11. Страница отчета по типам курсов

На рис. 12 изображена страница курсов

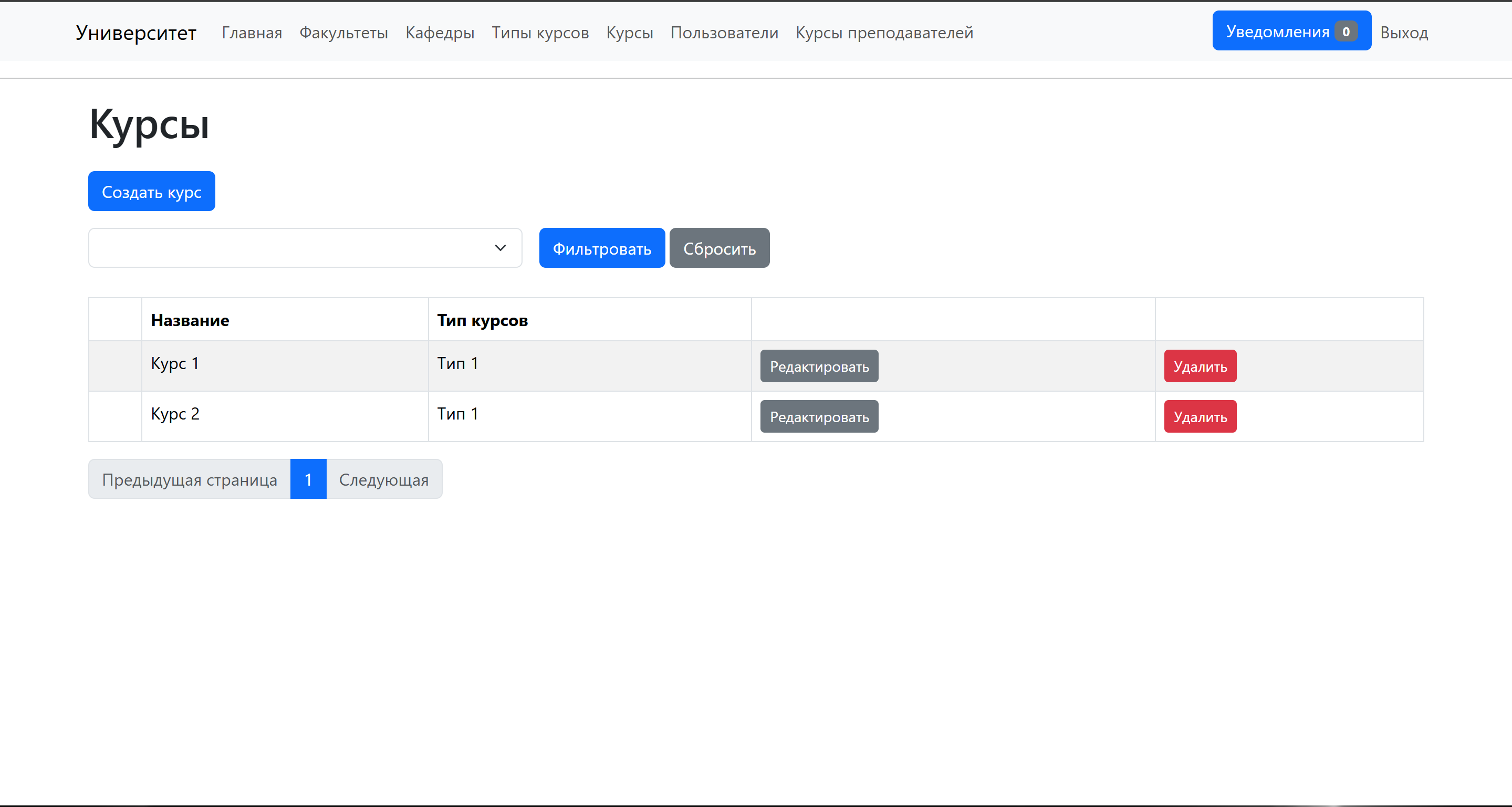
****

Рисунок 12. Страница курсов

На рис. 13 изображена страница создания и редактирования курса

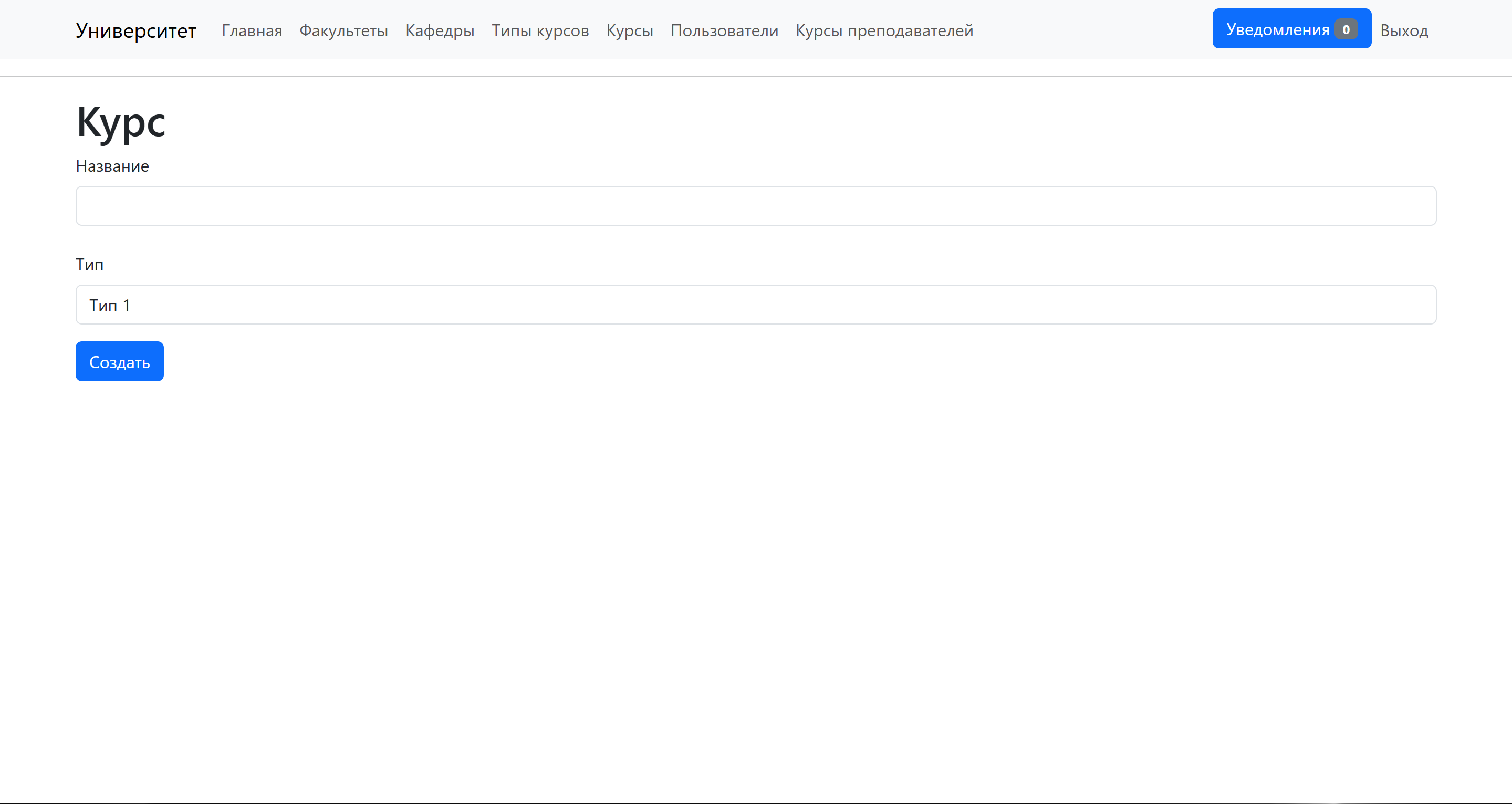
****

Рисунок 13. Страница создания и редактирования курсов

На рис. 14 изображена страница пользователей

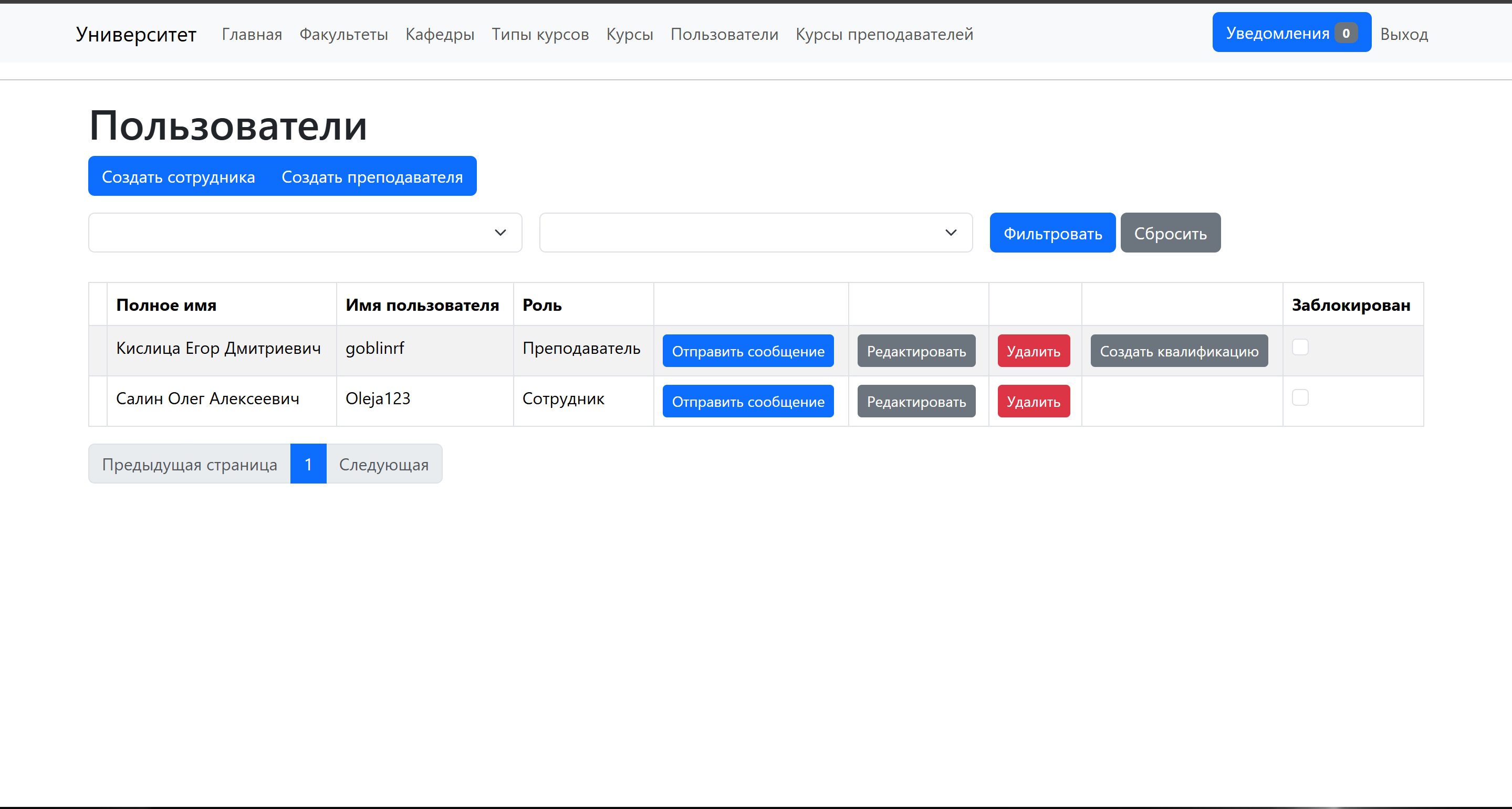
****

Рисунок 14. Страница пользователей

На рис. 15 изображена страница создания пользователя

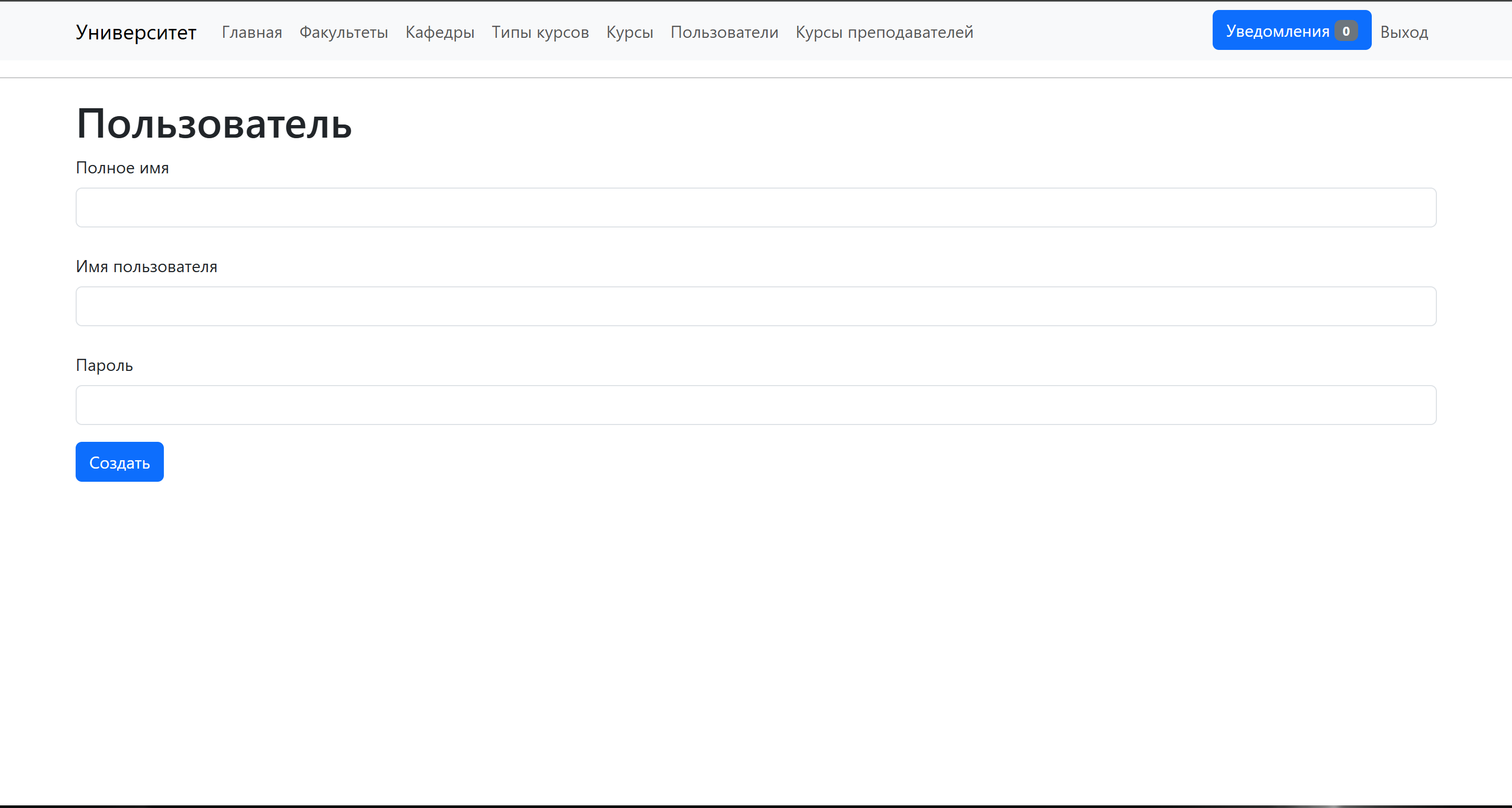
****

Рисунок 15. Страница создания пользователя

На рис. 16 изображена страница создания квалификации для пользователя

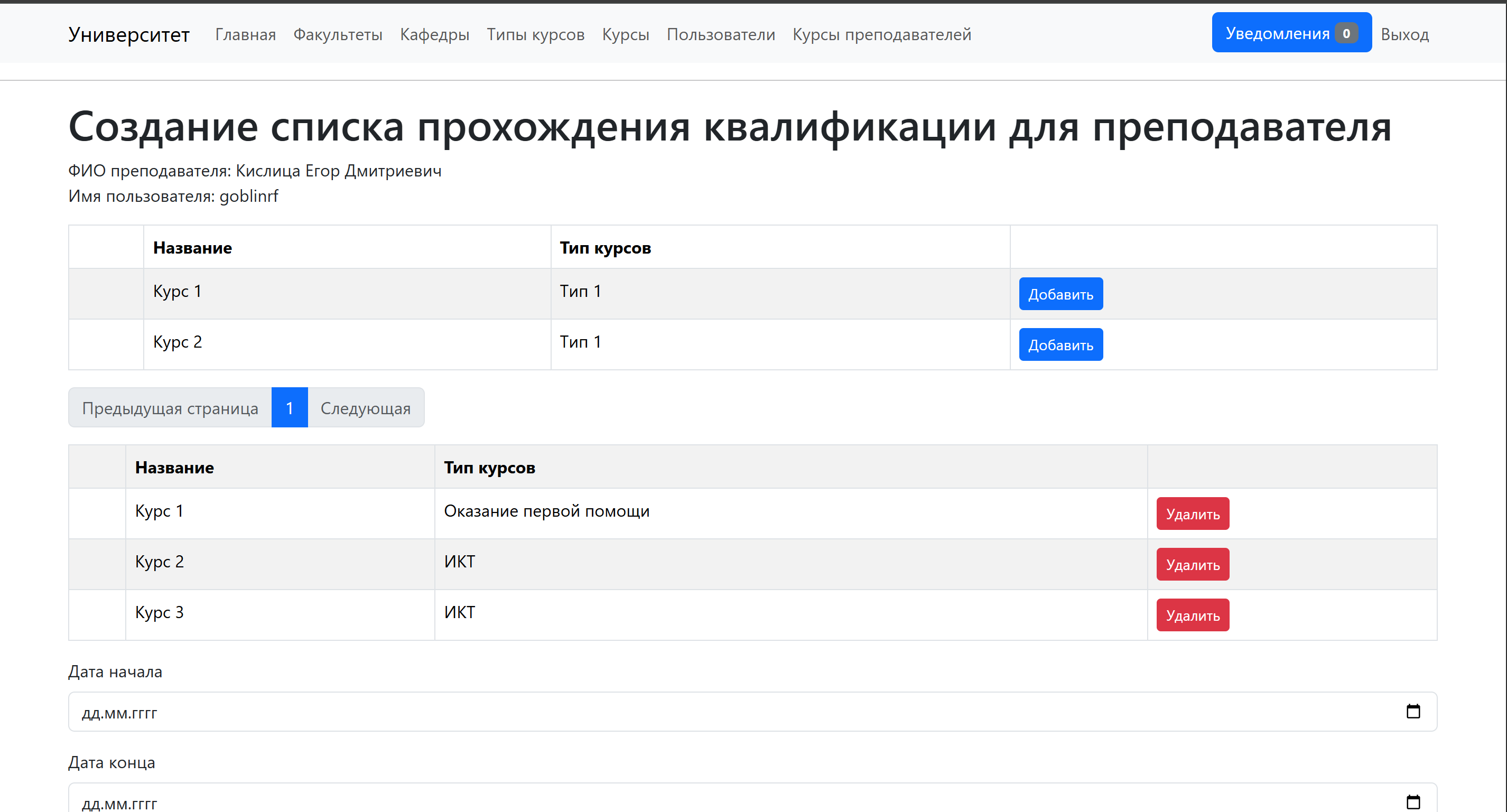
****

Рисунок 16. Страница создания квалификации для пользователя

На рис. 17 изображена страница курсов пользователей

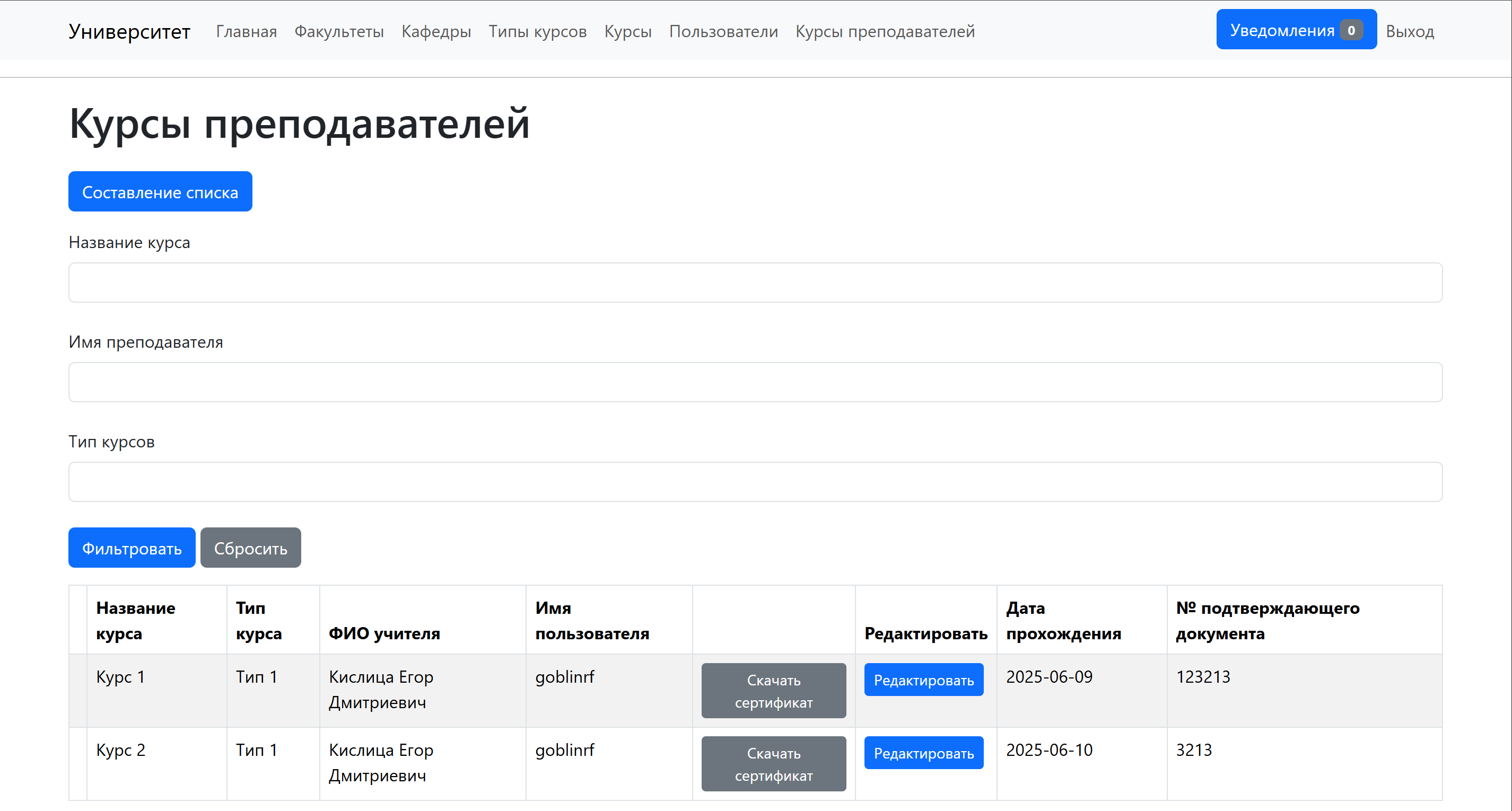
****

Рисунок 17. Страница курсов пользователей

На рис. 18 изображена страница редактирования курса пользователя

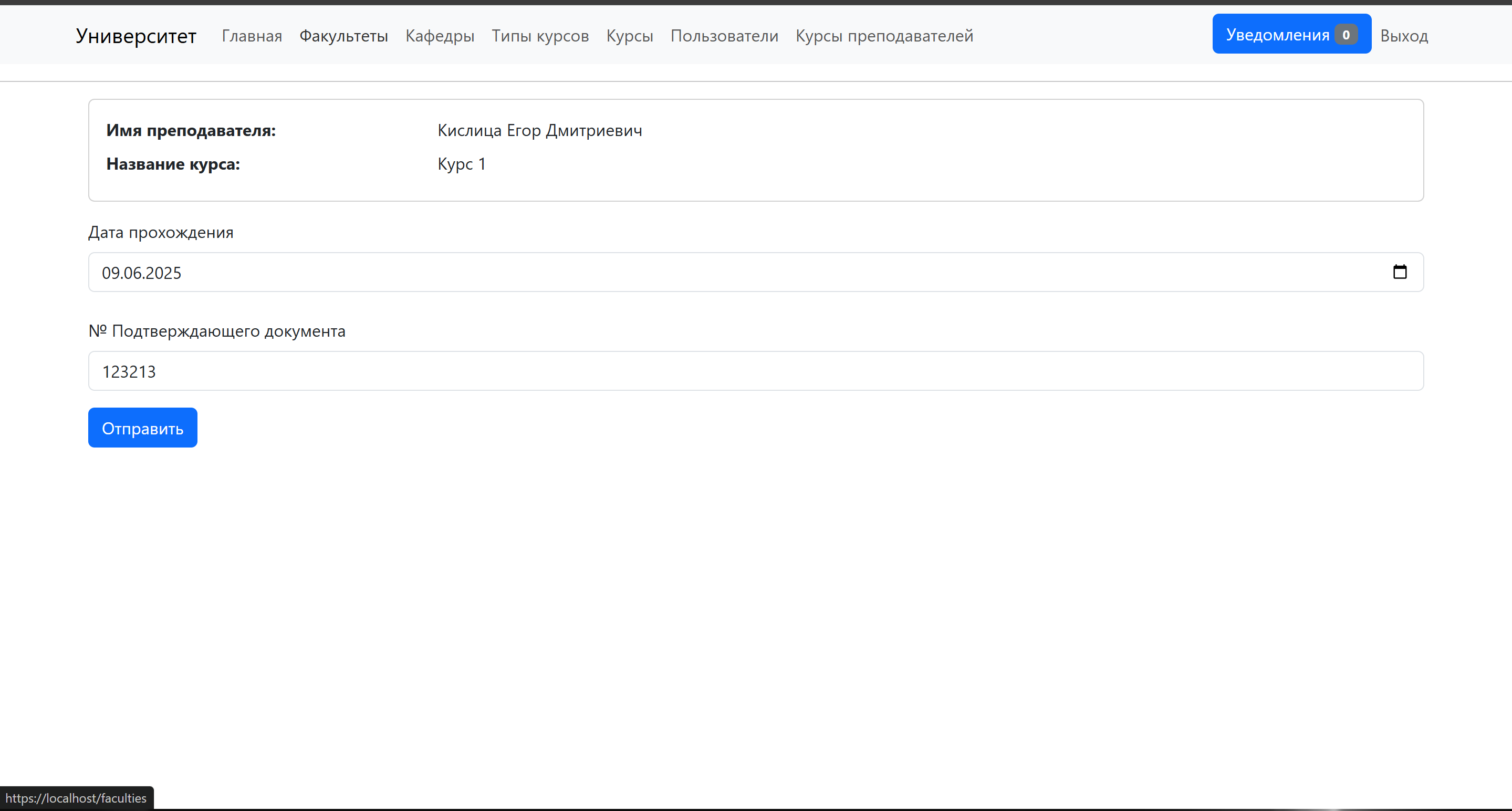
****

Рисунок 18. Страница редактирования курса пользователя

На рис. 19 изображена страница списка прохождения квалификаций

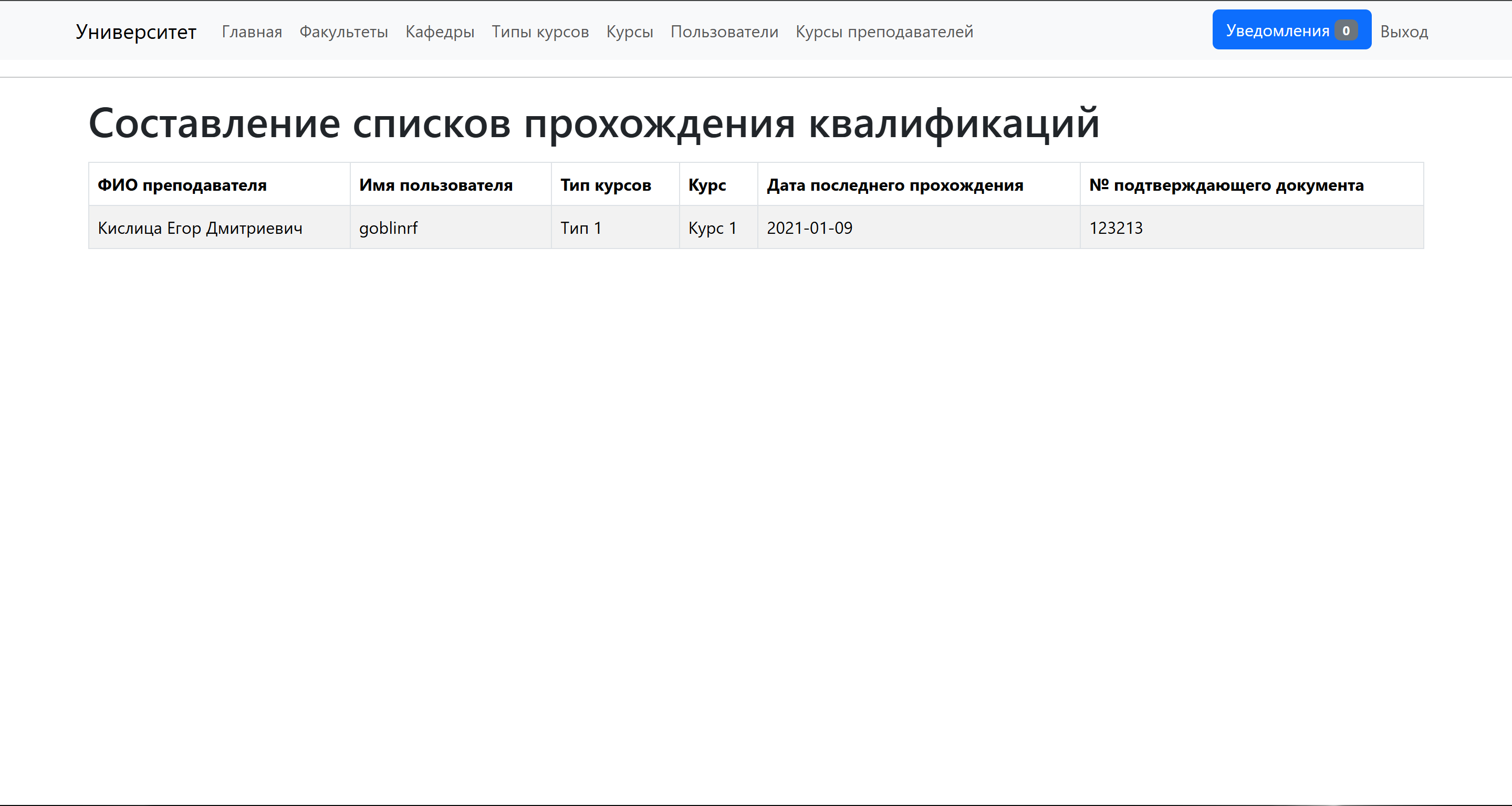
****

Рисунок 19. Страница списка прохождения квалификаций

На рис. 20 изображена страница уведомлений

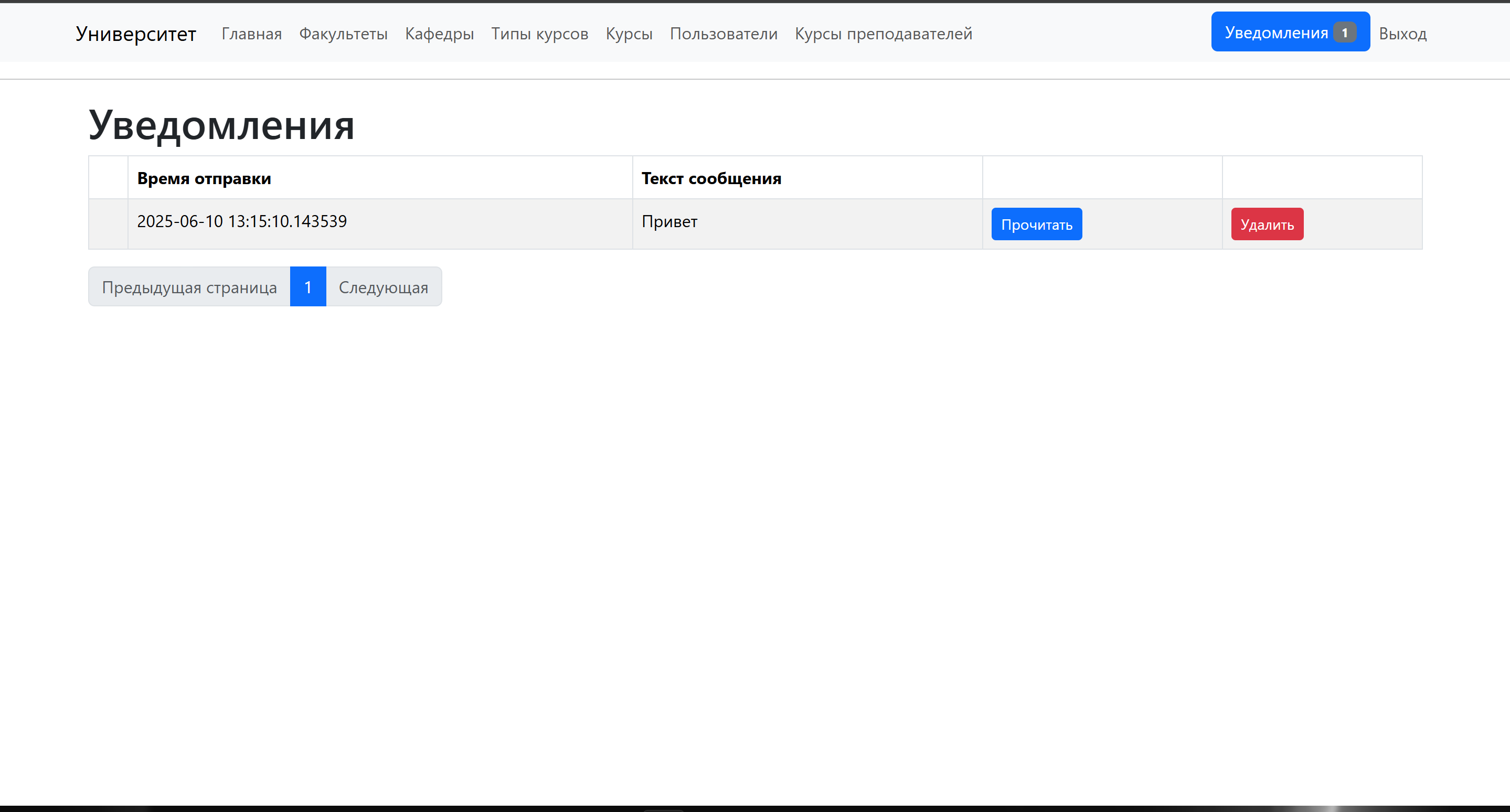
****

Рисунок 20. Страница уведомлений

На рис. 21 изображена главная страница преподавателя

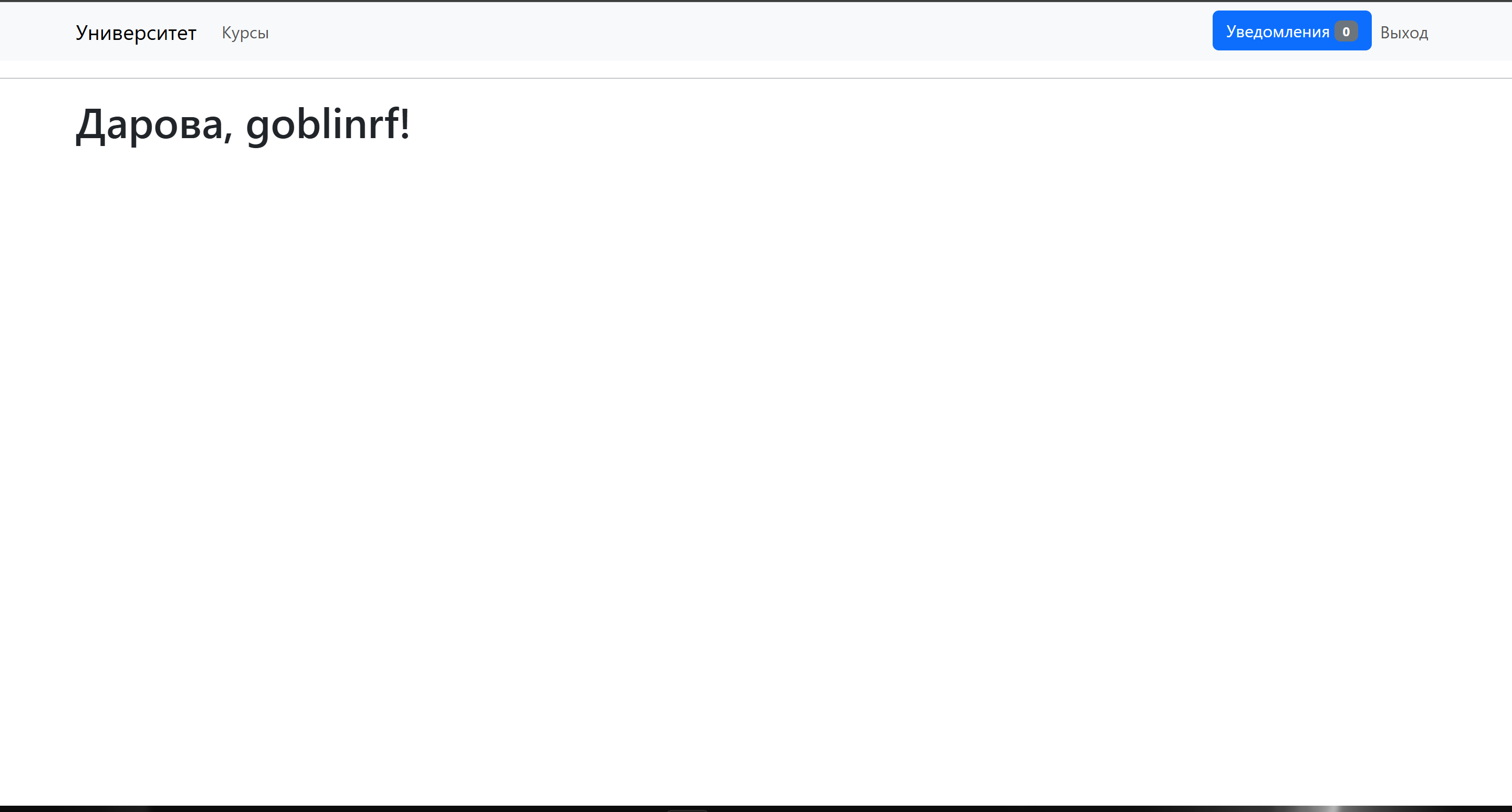
****

Рисунок 21. Главная страница преподавателя

На рис. 22 изображена страница курсов преподавателя

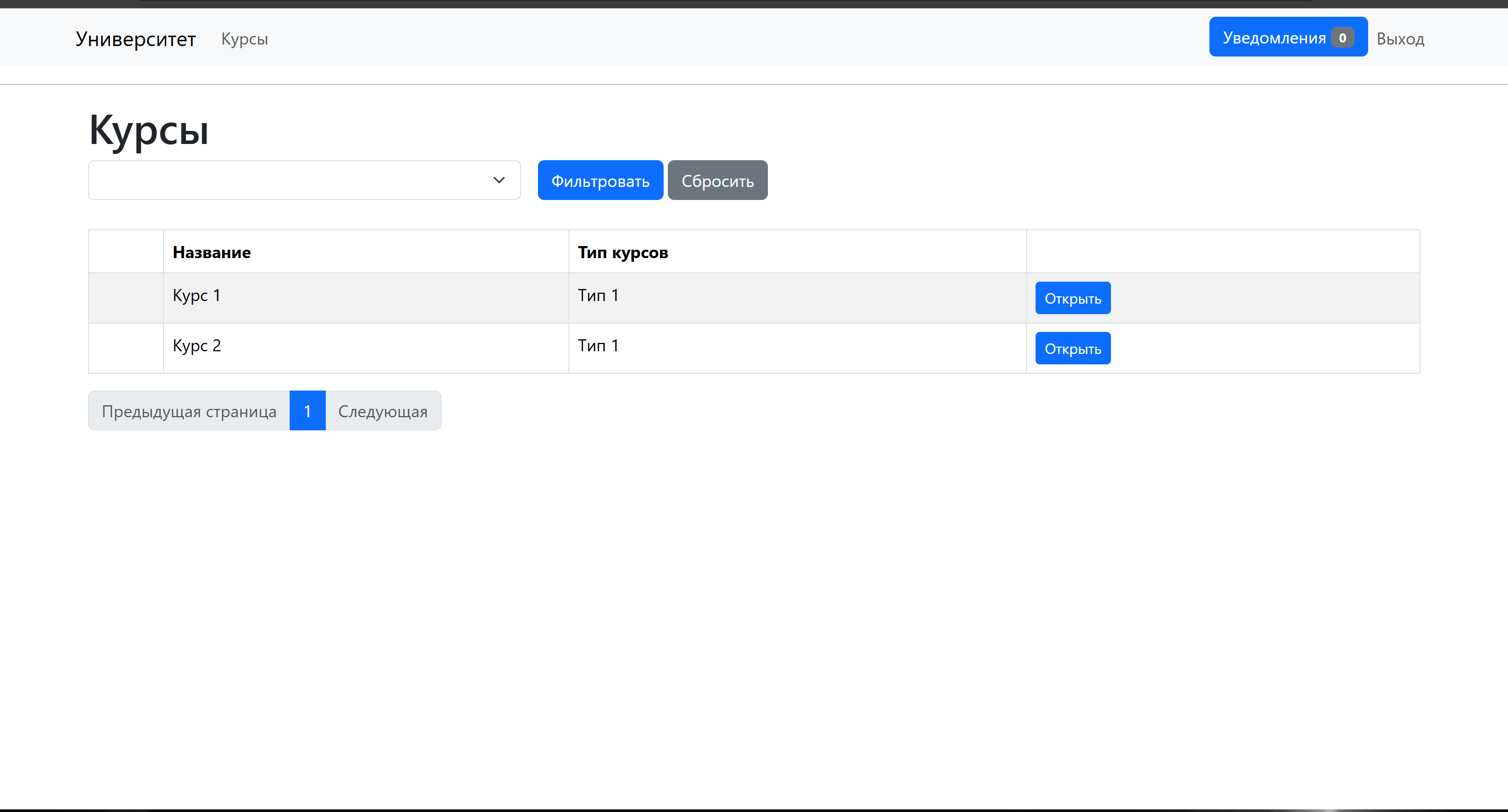
****

Рисунок 22. Страница курсов преподавателя

На рис. 23 изображена страница курса преподавателя

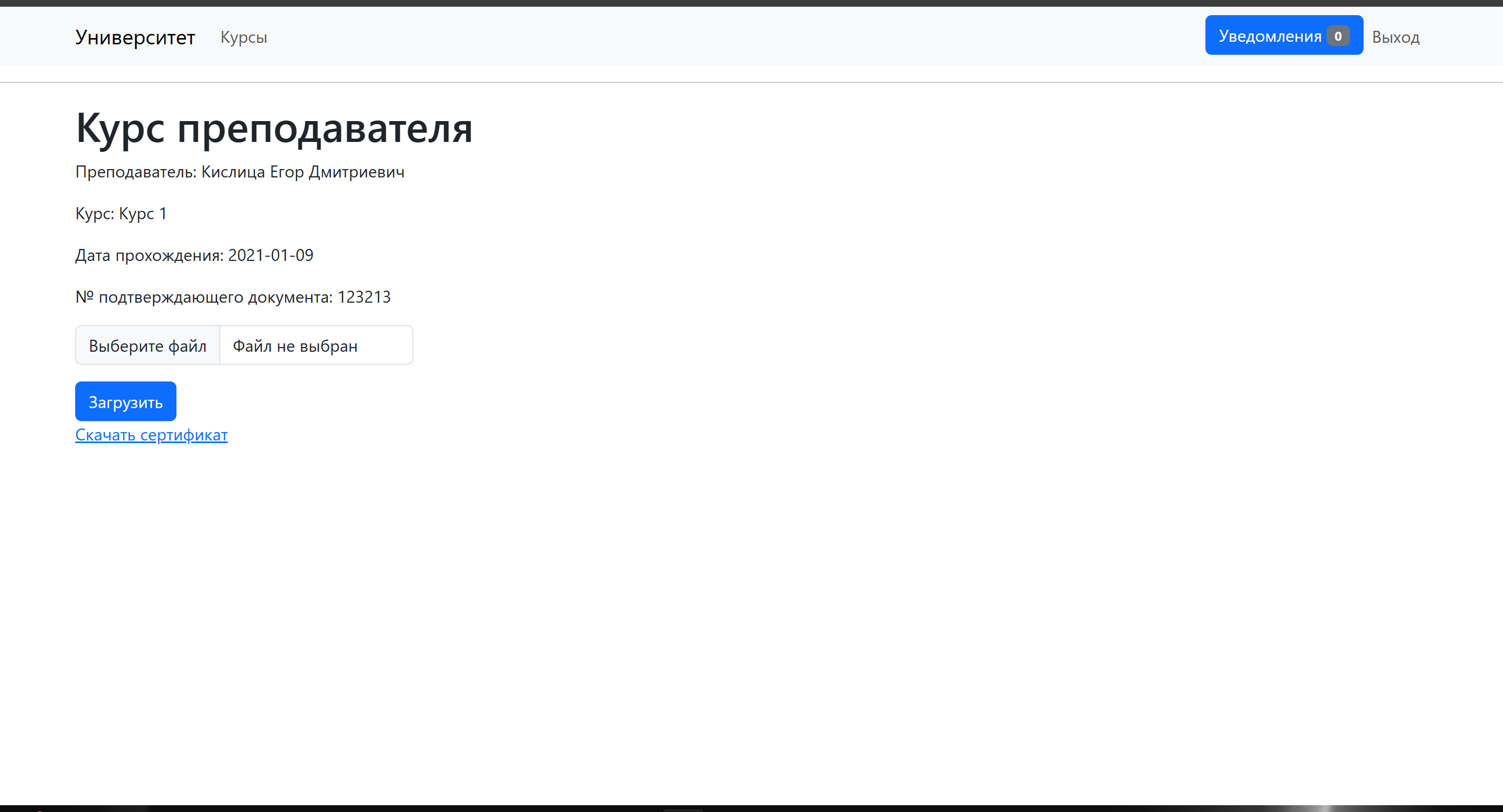
****

Рисунок 23. Страница курса преподавателя

На рис. 24 изображена форма входа в десктоп приложение

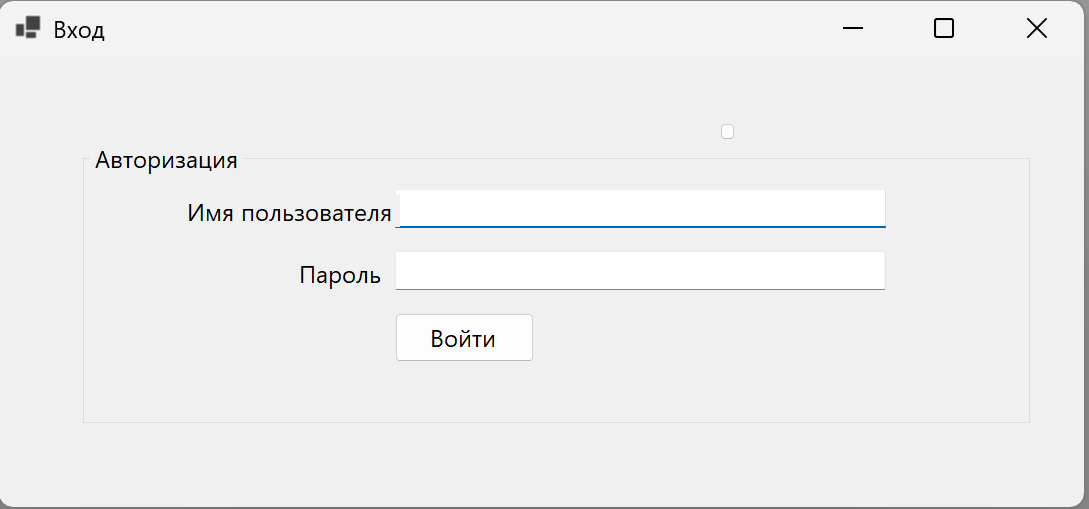
****

Рисунок 24. Форма входа в десктоп приложение

На рис. 25 изображена форма работы с пользователями

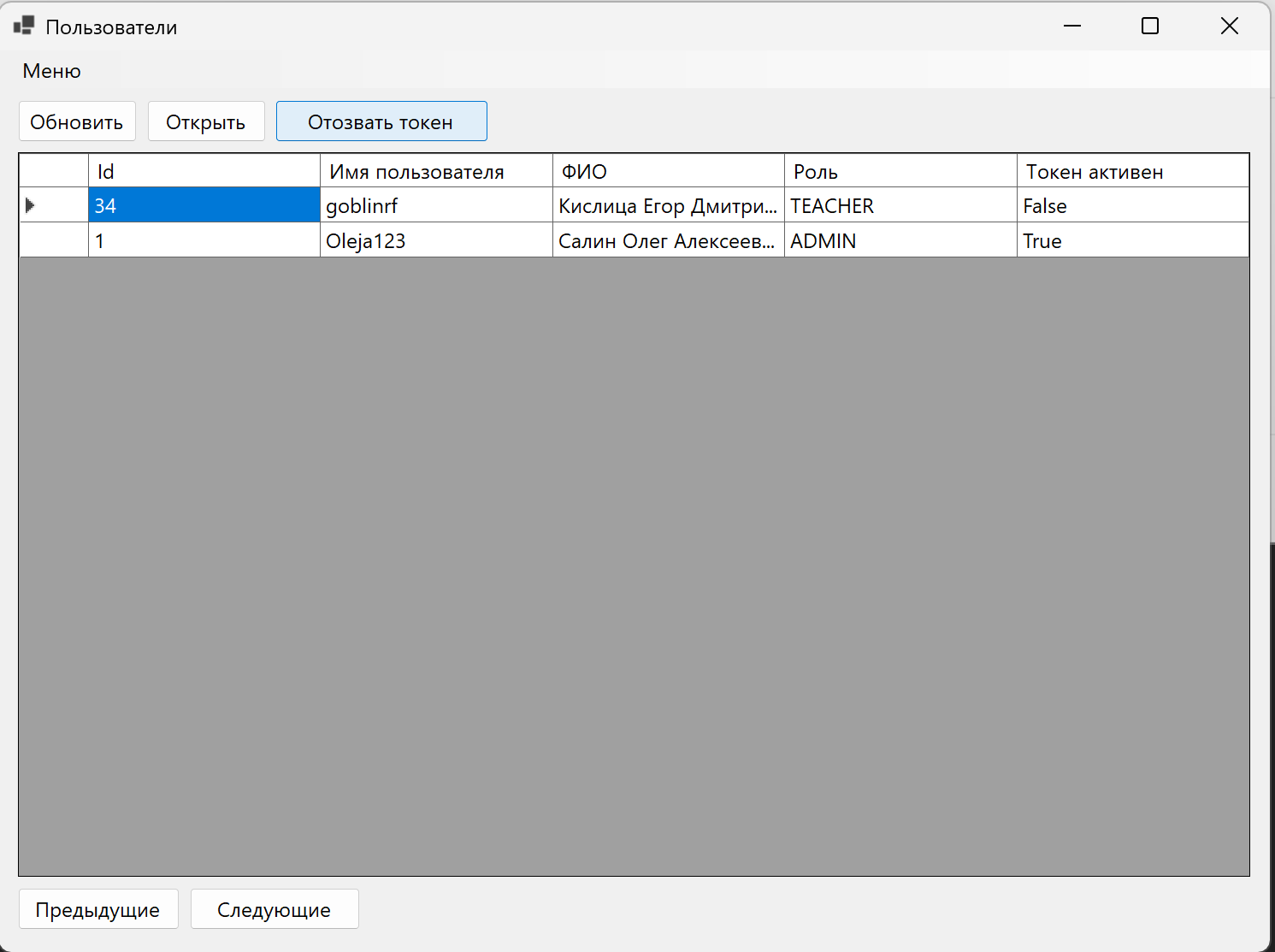
****

Рисунок 25. Форма работы с пользователями

На рис. 26 изображена форма работы с сессиями пользователя

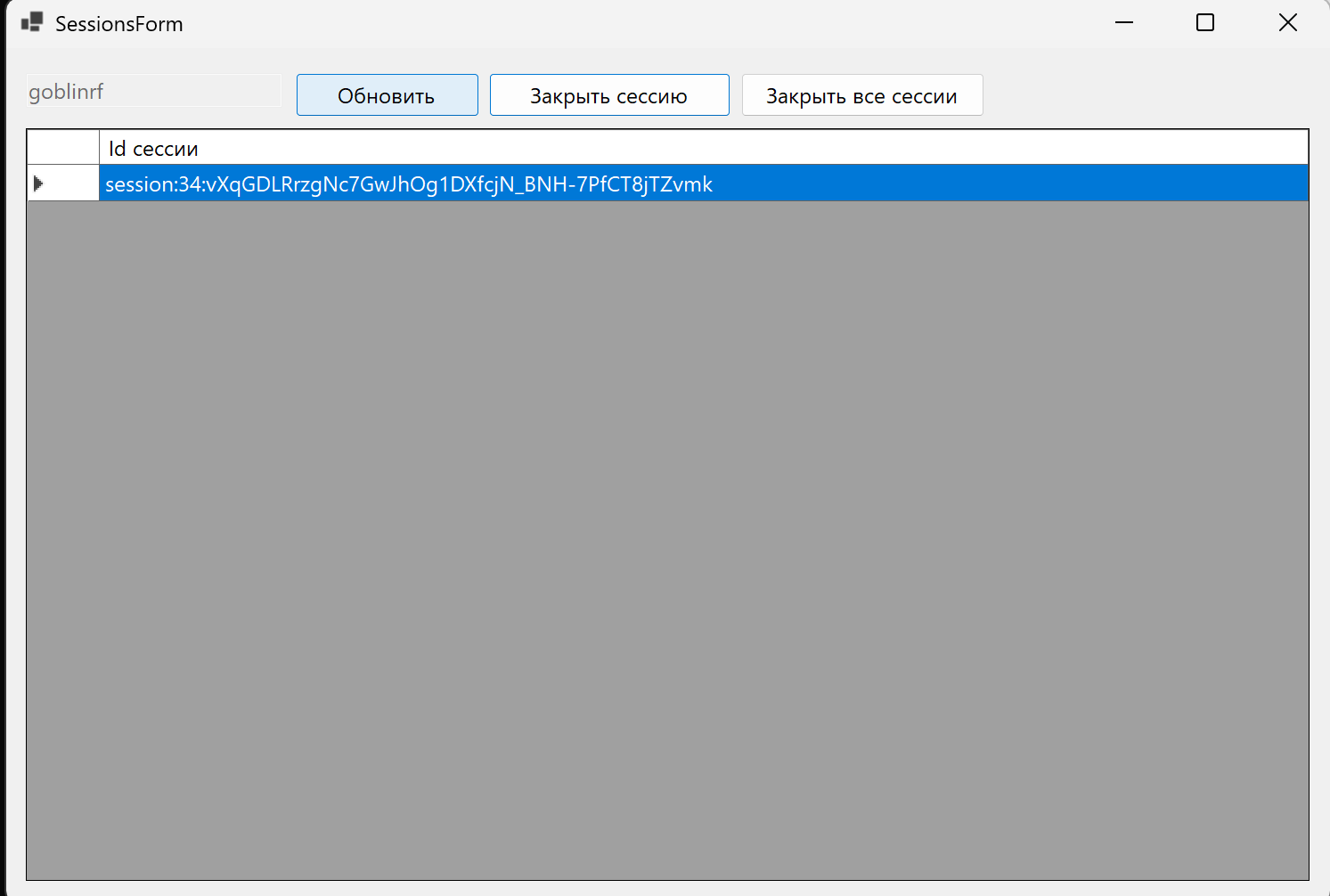
****

Рисунок 26. Форма работы с сессиями пользователя

На рис. 27 изображен пример работы с чат-ботом



Рисунок 27. Пример работы с чат-ботом

**Диаграмма классов, сгенерированная по коду**

Язык не предоставляет такой возможности.

# Выводы по реализации приложения на основании проекта

Во время реализации проекта конечный вид проекта претерпел изменения, т.к. исполнитель неверно трактовал ТЗ. Изменились технологии реализации решено было применить язык python, заместо Java.

В связи с этим произошли серьезные изменения в функционале – это изменение отчетов и изменение работы с статистическими данными.

Изменения, которые произошли с проектом в ходе разработки можно назвать существенными, так как оказали некоторое влияние на общую архитектуру системы. Но при всем этом, итоговая версия приложения отвечает всем основным требованиям, указанным в требованиях к проекту, и способна выполнять ключевые функции.

# Заключение

В результате был получен программный продукт, в котором были реализованы все поставленные цели и задачи –

* Авторизация для преподавателя и сотрудника
* Ведение списка факультетов, кафедр, преподавателей по кафедре
* Ведение курсов повышения квалификации (CRUD).
* Контроль и составление списков для прохождения квалификации по типам курсов (каждый тип курсов должен быть пройден не позже, чем каждые три года).
* Формирование курсов (запись преподавателя на курс, отметка о прохождении)
* Выдача свидетельств. Открытие прикреплённого файла свидетельства в системе.
* Формирование и печать статистики за период.
* Настройка и работа уведомлений

В ходе разработки системы получен опыт работы проектирования и разработки программных систем. Так же были получены знания по направлению архитектуры программных систем.

Были получены навыки проектирования программных систем, что является неотъемлемым этапом в ходе разработки программных продуктов. Проектирование позволяет согласовать работу разработчиков, разделить зоны ответственности, продумать особенности реализации приложения на этапе проектного решения, предложить обоснованные варианты реализации, а также сложить общую картину ожидаемого результата.

# Библиографический список

1. Крючкова, Е. Н. Объектно-ориентированное программирование: Архитектурное проектирование и паттерны программирования : учебно-методическое пособие / Е. Н. Крючкова, С. М. Старолетов. — Барнаул : АлтГТУ, 2020. — 180 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/292790>
2. Флегонтов, А. В. Моделирование информационных систем. Unified Modeling Language : учебное пособие / А. В. Флегонтов, И. Ю. Матюшичев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2022. — 112 с. — ISBN 978-5-8114-2907-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/206051
3. Корунова Н.В. Проектирование программного обеспечения: применение на практике / практикум по дисциплине «Проектирование информационных систем» / Н. В. Корунова. –Ульяновск : УлГТУ, 2019. – 69 с. - URL: […….](http://is.ulstu.ru/disc/pris). — Режим доступа: для авториз. пользователей.
4. Выполнение и оформление курсовых проектов (работ) [Текст]: методические указания / сост. Н. В. Корунова. - Ульяновск: УлГТУ, 2014. - 40 с. - Доступен также в Интернете <http://venec.ulstu.ru/lib/disk/2014/129.pdf>

# ПРИЛОЖЕНИЕ. Листинг кода

from datetime import timedelta

import os

from dotenv import load\_dotenv

import redis

basedir = os.path.abspath(os.path.dirname('\_\_file\_\_'))

load\_dotenv()

class Config:

SECRET\_KEY = os.getenv('SECRET\_KEY') or 'pososamba'

SQLALCHEMY\_DATABASE\_URI = os.getenv(

'DATABASE\_URL') or 'sqlite:///' + os.path.join(basedir, 'app.db')

DEPARTMENTS\_PER\_PAGE = 3

NOTIFICATIONS\_PER\_PAGE = 3

USERS\_PER\_PAGE = 3

COURSES\_PER\_PAGE = 3

UPLOAD\_FOLDER = 'sertificates'

ALLOWED\_EXTENSIONS = {'pdf'}

LOG\_TO\_STDOUT = os.getenv('LOG\_TO\_STDOUT')

SESSION\_TYPE = 'redis'

SESSION\_USE\_SIGNER = True

SESSION\_COOKIE\_HTTPONLY=True

SESSION\_REDIS = redis.Redis(

host=os.getenv('REDIS\_HOSTNAME'),

port=os.getenv('REDIS\_PORT'),

password=os.getenv('REDIS\_PASSWORD'),

db=0

)

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import create\_app, db

from app.models.course import Course

from app.models.department import Department

from app.models.faculty import Faculty

from app.models.teacher\_course import TeacherCourse

from app.models.user import User

app = create\_app()

@app.shell\_context\_processor

def make\_shell\_context():

return {'sa': sa, 'so': so, 'db': db, 'User': User,

'Course': Course, 'Faculty': Faculty,

'Department': Department, 'TeacherCourse': TeacherCourse,

'r': app.config['SESSION\_REDIS']}

import logging

from logging.handlers import RotatingFileHandler

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

from flask\_session import Session

from flask\_login import LoginManager

from flask\_wtf.csrf import CSRFProtect

from sqlalchemy.engine import Engine

from sqlalchemy import event

from config import Config

db = SQLAlchemy()

migrate = Migrate()

login = LoginManager()

csrf = CSRFProtect()

login.login\_view = 'auth.login'

login.login\_message = ('РџРѕР¶Р°Р»СѓР№СЃС‚Р° Р°РІС‚РѕСЂРёР·СѓР№С‚РµСЃСЊ, С‡С‚РѕР±С‹ РїСЂРѕСЃРјР°С‚СЂРёРІР°С‚СЊ РґР°РЅРЅСѓСЋ СЃС‚СЂР°РЅРёС†Сѓ')

def create\_app(config\_class=Config):

app = Flask(\_\_name\_\_)

app.config.from\_object(config\_class)

db.init\_app(app)

migrate.init\_app(app, db)

login.init\_app(app)

csrf.init\_app(app)

Session(app)

from app.errors import bp as errors\_bp

app.register\_blueprint(errors\_bp)

from app.auth import bp as auth\_bp

app.register\_blueprint(auth\_bp)

from app.main import bp as main\_bp

app.register\_blueprint(main\_bp)

from app.api import bp as api\_bp

app.register\_blueprint(api\_bp, url\_prefix='/api')

if 'postgresql' not in app.config['SQLALCHEMY\_DATABASE\_URI']:

@event.listens\_for(Engine, "connect")

def set\_sqlite\_pragma(dbapi\_connection, connection\_record):

cursor = dbapi\_connection.cursor()

cursor.execute("PRAGMA foreign\_keys=ON")

cursor.close()

if not app.debug:

if app.config['LOG\_TO\_STDOUT'] == True:

stream\_handler = logging.StreamHandler()

stream\_handler.setLevel(logging.INFO)

app.logger.addHandler(stream\_handler)

else:

if not os.path.exists('logs'):

os.mkdir('logs')

file\_handler = RotatingFileHandler('logs/app.log',

maxBytes=10240, backupCount=10)

file\_handler.setFormatter(logging.Formatter(

'%(asctime)s %(levelname)s: %(message)s '

'[in %(pathname)s:%(lineno)d]'))

file\_handler.setLevel(logging.INFO)

app.logger.addHandler(file\_handler)

app.logger.setLevel(logging.INFO)

app.logger.info('University startup')

return app

from app import models, services

from flask import abort, current\_app

from flask\_httpauth import HTTPBasicAuth, HTTPTokenAuth

from app.api.errors import error\_response

from app.exceptions.fired\_error import FiredError

from app.exceptions.wrong\_password\_error import WrongPasswordError

from app.services import user\_service

basic\_auth = HTTPBasicAuth()

token\_auth = HTTPTokenAuth()

@basic\_auth.verify\_password

def verify\_password(username, password):

try:

user = user\_service.check\_password(username, password)

return user

except ValueError as e:

abort(400)

except WrongPasswordError as e:

abort(401)

except FiredError as e:

abort(403)

except Exception as e:

abort(500)

@basic\_auth.error\_handler

def basic\_auth\_error(status):

current\_app.logger.info(status)

return error\_response(status)

@token\_auth.verify\_token

def verify\_token(token):

try:

return user\_service.check\_token(token) if token else None

except ValueError as e:

abort(401)

except Exception as e:

abort(500)

@token\_auth.error\_handler

def token\_auth\_error(status):

return error\_response(status)

from functools import wraps

from flask import abort

from app.api.auth import token\_auth

from app.models.user import ADMIN

def required\_role(role):

def decorator(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not token\_auth.current\_user().role == role:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

return decorator

def user\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

user\_id = kwargs.get('user\_id')

if not token\_auth.current\_user().role == ADMIN and \

(user\_id is None or str(user\_id) != str(str(token\_auth.current\_user().id))):

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

def only\_admin(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not token\_auth.current\_user().role == ADMIN:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

from werkzeug.http import HTTP\_STATUS\_CODES

from werkzeug.exceptions import HTTPException

from app.api import bp

def error\_response(status\_code, message=None):

payload = {'error': HTTP\_STATUS\_CODES.get(status\_code, 'Unknown error')}

if message:

payload['message'] = message

return payload, status\_code

def bad\_request(message):

return error\_response(400, message)

@bp.app\_errorhandler(HTTPException)

def handle\_exception(e):

return error\_response(e.code)

import json

from flask import Response, abort

from app.api import bp

from app import csrf

from app.models.user import ADMIN

from app.services import notification\_service

from app.api.auth import token\_auth

@bp.route('/notifications/<int:id>', methods=['GET'])

@token\_auth.login\_required

def get\_notification(id):

try:

res = notification\_service.get\_by\_id(id)

if token\_auth.current\_user().role != ADMIN and \

token\_auth.current\_user().id != res.user\_id:

abort(403)

return Response(

json.dumps(res.to\_dict(), ensure\_ascii=False),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

abort(404)

except Exception as e:

abort(500)

@bp.route('/notifications/<int:id>/read', methods=['PUT'])

@csrf.exempt

@token\_auth.login\_required

def read\_notification(id):

try:

res = notification\_service.get\_by\_id(id)

if token\_auth.current\_user().role != ADMIN and \

token\_auth.current\_user().id != res.user\_id:

abort(403)

notification\_service.read\_message(id)

return Response(

json.dumps(notification\_service.get\_by\_id(id).to\_dict(), ensure\_ascii=False),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

abort(404)

except Exception as e:

abort(500)

@bp.route('/notifications/<int:id>/delete', methods=['DELETE'])

@csrf.exempt

@token\_auth.login\_required

def delete\_notification(id):

try:

res = notification\_service.get\_by\_id(id)

if token\_auth.current\_user().role != ADMIN and \

token\_auth.current\_user().id != res.user\_id:

abort(403)

notification\_service.delete(id)

return '', 204

except ValueError as e:

abort(404)

except Exception as e:

abort(500)

from flask import abort

from app.api import bp

from app.api.auth import basic\_auth

from app.services import user\_service

from app.api.auth import token\_auth

from app import csrf

@bp.route('/tokens', methods=['POST'])

@csrf.exempt

@basic\_auth.login\_required

def get\_token():

try:

token = user\_service.get\_token(basic\_auth.current\_user().id)

return {'token': token,

'id': basic\_auth.current\_user().id,

'role': 'TEACHER' if basic\_auth.current\_user().role else 'ADMIN'}

except ValueError as e:

abort(404)

except Exception as e :

abort(500)

@bp.route('/tokens', methods=['DELETE'])

@csrf.exempt

@token\_auth.login\_required

def revoke\_token():

user\_service.revoke\_token(token\_auth.current\_user().id)

return '', 204

import json

import os

from flask import Response, abort, current\_app, request, send\_from\_directory, url\_for

from app.api import bp

from app.exceptions.role\_error import RoleError

from app.services import course\_service, sertificate\_service, user\_service

from app.services.to\_json\_collecion import to\_json\_collection

from app.models.user import TEACHER

from app.api.auth import token\_auth

from app.api.decorators import user\_required, only\_admin

from app.models import user

from app.dto.user\_dto import UserDTO

from app import csrf

@bp.route('/users/<int:user\_id>', methods=['GET'])

@token\_auth.login\_required

@user\_required

def get\_user(user\_id):

try:

return Response(

json.dumps(user\_service.get\_by\_id(

user\_id).to\_dict(), ensure\_ascii=False),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

current\_app.logger.error(e)

abort(404)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users', methods=['GET'])

@token\_auth.login\_required

@only\_admin

def get\_users():

try:

page = request.args.get('page', 1, type=int)

username = request.args.get('username', None, type=str)

full\_name = request.args.get('full\_name', None, type=str)

res = to\_json\_collection(user\_service.get\_all\_paginated(page=page,

userDTO=UserDTO(username=username,

full\_name=full\_name)),

'api.get\_users')

return Response(

json.dumps(

res,

ensure\_ascii=False

),

mimetype='application/json; charset=utf-8'

)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/revoke\_token', methods=['PUT'])

@csrf.exempt

@token\_auth.login\_required

@only\_admin

def revoke\_user\_token(user\_id):

try:

res = user\_service.revoke\_token(user\_id)

return Response(

json.dumps(res.to\_dict(), ensure\_ascii=False),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

abort(404)

except Exception as e:

abort(500)

@bp.route('/users/<int:user\_id>/sessions', methods=['GET'])

@token\_auth.login\_required

@only\_admin

def get\_user\_sessions(user\_id):

try:

res = [i.decode() for i in user\_service.get\_users\_sessions(user\_id)]

data = {

'items': [item for item in res],

'\_meta': {

'total\_items': len(res)

},

'\_links': {

'self': url\_for('api.get\_user\_sessions', user\_id=user\_id),

'close': url\_for('api.get\_user\_sessions', user\_id=user\_id) + '/close',

}

}

return Response(

json.dumps(

data,

ensure\_ascii=False

),

mimetype='application/json; charset=utf-8'

)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/sessions/close', methods=['DELETE'])

@csrf.exempt

@token\_auth.login\_required

@only\_admin

def close\_user\_sessions(user\_id):

try:

user\_service.close\_user\_sessions(user\_id)

return '', 204

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/sessions/close/<session\_id>', methods=['DELETE'])

@csrf.exempt

@token\_auth.login\_required

@only\_admin

def close\_user\_session(user\_id, session\_id):

try:

user\_service.close\_user\_session(session\_id)

return '', 204

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/notifications', methods=['GET'])

@token\_auth.login\_required

@user\_required

def get\_user\_notifications(user\_id):

try:

user = user\_service.get\_by\_id(user\_id)

page = request.args.get('page', 1, type=int)

res = to\_json\_collection(user\_service.get\_notifications(page=page, only\_new=False, user=user),

'api.get\_user\_notifications', user\_id=user\_id)

return Response(

json.dumps(

res,

ensure\_ascii=False

),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

current\_app.logger.error(e)

abort(404)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/courses', methods=['GET'])

@token\_auth.login\_required

@user\_required

def get\_teacher\_courses(user\_id):

try:

user = user\_service.get\_by\_id(user\_id)

page = request.args.get('page', 1, type=int)

res = to\_json\_collection(sertificate\_service.get\_user\_courses(page=page, user\_id=user\_id),

'api.get\_teacher\_courses', user\_id=user\_id)

return Response(

json.dumps(

res,

ensure\_ascii=False

),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

current\_app.logger.error(e)

abort(404)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/courses/<int:course\_id>', methods=['GET'])

@token\_auth.login\_required

@user\_required

def get\_teacher\_course(user\_id, course\_id):

try:

user = user\_service.get\_by\_id(user\_id)

if user.role != TEACHER:

raise RoleError('РЈ СЃРѕС‚СЂСѓРґРЅРёРєР° РЅРµ РјРѕР¶РµС‚ Р±С‹С‚СЊ РєСѓСЂСЃРѕРІ')

course = course\_service.get\_by\_id(course\_id)

return Response(

json.dumps(sertificate\_service.get(

user\_id, course\_id).to\_dict(), ensure\_ascii=False),

mimetype='application/json; charset=utf-8'

)

except ValueError as e:

current\_app.logger.error(e)

abort(404)

except Exception as e:

current\_app.logger.error(e)

abort(500)

@bp.route('/users/<int:user\_id>/courses/<int:course\_id>/download', methods=['GET'])

@token\_auth.login\_required

@user\_required

def download\_teacher\_course(user\_id, course\_id):

try:

user\_path = sertificate\_service.make\_path(str(user\_id), str(course\_id))

if not os.path.exists(user\_path):

raise ValueError('РЎРµСЂС‚РёС„РёРєР°С‚ РЅРµ Р·Р°РіСЂСѓР¶РµРЅ')

current\_app.logger.info(user\_path)

file = os.listdir(user\_path)[0]

current\_app.logger.info(file)

return send\_from\_directory(user\_path,

file,

as\_attachment=True,

mimetype='application/pdf')

except ValueError as e:

current\_app.logger.error(e)

abort(404)

except Exception as e:

current\_app.logger.error(e)

abort(500)

from flask import Blueprint

bp = Blueprint('api', \_\_name\_\_)

from app.api import errors, users, notifications, auth, tokens, decorators

from flask\_wtf import FlaskForm

from wtforms import BooleanField, PasswordField, StringField, SubmitField

from wtforms.validators import DataRequired

class LoginForm(FlaskForm):

username = StringField('РРјСЏ', validators=[DataRequired()])

password = PasswordField('РџР°СЂРѕР»СЊ', validators=[DataRequired()])

submit = SubmitField('Р’С…РѕРґ')

from urllib.parse import urlsplit

from flask import current\_app, flash, redirect, render\_template, request, session, url\_for

from flask\_login import current\_user, login\_required, login\_user, logout\_user

from app.auth.forms import LoginForm

from app.services import user\_service

from app import login

from app.auth import bp

@login.user\_loader

def load\_user(id):

try:

return user\_service.get\_by\_id(id)

except Exception as e:

current\_app.logger.info(e)

@bp.route('/login', methods=['GET', 'POST'])

def login():

if current\_user.is\_authenticated:

return redirect(url\_for('main.index'))

form = LoginForm()

if form.validate\_on\_submit():

try:

user = user\_service.get\_by\_username(form.username.data)

user\_service.check\_password(user.username, form.password.data)

login\_user(user)

session['user\_role'] = user.role

session['user\_id'] = user.id

r = current\_app.config['SESSION\_REDIS']

r.delete('session:' + session.sid)

session.sid = f'{user.id}:' + session.sid

next\_page = request.args.get('next')

if not next\_page or urlsplit(next\_page).netloc != '':

next\_page = url\_for('main.index')

return redirect(next\_page)

except Exception as e:

flash(str(e))

return redirect(url\_for('auth.login'))

return render\_template('auth/login.html', title='Р’С…РѕРґ', form=form)

@bp.route('/logout')

@login\_required

def logout():

logout\_user()

session.clear()

return redirect(url\_for('main.index'))

from flask import Blueprint

bp = Blueprint('auth', \_\_name\_\_)

from app.auth import routes

from functools import wraps

from flask import abort, make\_response, render\_template, session

from flask\_login import current\_user

def required\_role(role):

def decorator(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not current\_user.role == role:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

return decorator

from functools import wraps

from flask import abort

from flask\_login import current\_user

def user\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

user\_id = kwargs.get('user\_id')

if user\_id and str(current\_user.id) != str(user\_id):

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_function

from dataclasses import dataclass

from typing import Optional

@dataclass

class CourseDTO:

id: Optional[int] = None

name: Optional[str] = None

course\_type\_id: Optional[int] = None

@staticmethod

def from\_form(form, id=None):

if id is not None:

return CourseDTO(id=id, name=form.name.data, course\_type\_id=form.course\_type.data)

return CourseDTO(name=form.name.data, course\_type\_id=form.course\_type.data)

from dataclasses import dataclass

from typing import Optional

@dataclass

class CourseTypeDTO:

id: Optional[int] = None

name: Optional[str] = None

@staticmethod

def from\_form(form, id=None):

if id is not None:

return CourseTypeDTO(id=id, name=form.name.data)

return CourseTypeDTO(name=form.name.data)

from dataclasses import dataclass

from typing import Optional

@dataclass

class DepartmentDTO:

id: Optional[int] = None

faculty\_id: Optional[int] = None

name: Optional[str] = None

@staticmethod

def from\_form(form, id=None):

if id is not None:

return DepartmentDTO(id=id, name=form.name.data, faculty\_id=form.faculty.data)

return DepartmentDTO(name=form.name.data, faculty\_id=form.faculty.data)

from dataclasses import dataclass

from typing import Optional

@dataclass

class FacultyDTO:

id: Optional[int] = None

name: Optional[str] = None

@staticmethod

def from\_form(form, id=None):

if id is not None:

return FacultyDTO(id=id, name=form.name.data)

return FacultyDTO(name=form.name.data)

from dataclasses import dataclass

from typing import Optional

@dataclass

class NotificationDTO:

id: Optional[int] = None

user\_id: Optional[int] = None

message: Optional[str] = None

from dataclasses import dataclass

from typing import Optional

@dataclass

class UserDTO:

id: Optional[int] = None

username: Optional[str] = None

full\_name: Optional[str] = None

password: Optional[str] = None

is\_fired: Optional[bool] = None

role: Optional[int] = None

@staticmethod

def from\_form(form, role=None, id=None):

if id is not None:

return UserDTO(id=id, full\_name=form.full\_name.data, username=form.username.data, password=form.password.data, role=role)

return UserDTO(full\_name=form.full\_name.data, username=form.username.data, password=form.password.data, role=role)

from flask import current\_app, render\_template, request

from app.errors import bp

from app.api.errors import error\_response as api\_error\_response

def wants\_json\_response():

return request.accept\_mimetypes['application/json'] >= \

request.accept\_mimetypes['text/html']

@bp.app\_errorhandler(404)

def not\_found\_error(error):

if wants\_json\_response():

return api\_error\_response(404)

return render\_template('errors/404.html'), 404

@bp.app\_errorhandler(403)

def forbidden\_error(error):

if wants\_json\_response():

return api\_error\_response(403)

return render\_template('errors/403.html'), 403

@bp.app\_errorhandler(500)

def internal\_error(error):

if wants\_json\_response():

return api\_error\_response(500)

return render\_template('errors/500.html'), 500

@bp.app\_errorhandler(413)

def internal\_error(error):

if wants\_json\_response():

return api\_error\_response(413)

return render\_template('errors/413.html'), 500

from flask import Blueprint

bp = Blueprint('errors', \_\_name\_\_)

from app.errors import handlers

class FiredError(Exception):

pass

class RoleError(Exception):

pass

class WrongPasswordError(Exception):

pass

from flask\_wtf import FlaskForm

from wtforms import DateField, FileField, StringField, SubmitField, SelectField

from wtforms.validators import DataRequired, Length, Optional

from app.services import course\_type\_service, faculty\_service

class EditFacultyForm(FlaskForm):

name = StringField('РќР°Р·РІР°РЅРёРµ', validators=[DataRequired()])

submit = SubmitField('РЎРѕР·РґР°С‚СЊ')

def from\_model(self, model):

self.name.data = model.name

class EditCourseTypeForm(FlaskForm):

name = StringField('РќР°Р·РІР°РЅРёРµ', validators=[DataRequired()])

submit = SubmitField('РЎРѕР·РґР°С‚СЊ')

def from\_model(self, model):

self.name.data = model.name

class EditDepartmentForm(FlaskForm):

name = StringField('РќР°Р·РІР°РЅРёРµ', validators=[DataRequired()])

faculty = SelectField('Р¤Р°РєСѓР»СЊС‚РµС‚', validators=[DataRequired()], coerce=int)

submit = SubmitField('РЎРѕР·РґР°С‚СЊ')

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(EditDepartmentForm, self).\_\_init\_\_(\*args, \*\*kwargs)

faculties = faculty\_service.get\_all()

self.faculty.choices = [(f.id, f.name) for f in faculties]

def from\_model(self, model):

self.name.data = model.name

self.faculty.data = model.faculty\_id

class EditCourseForm(FlaskForm):

name = StringField('РќР°Р·РІР°РЅРёРµ', validators=[DataRequired()])

course\_type = SelectField('РўРёРї', validators=[DataRequired()], coerce=int)

submit = SubmitField('РЎРѕР·РґР°С‚СЊ')

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(EditCourseForm, self).\_\_init\_\_(\*args, \*\*kwargs)

course\_types = course\_type\_service.get\_all()

self.course\_type.choices = [(t.id, t.name) for t in course\_types]

def from\_model(self, model):

self.name.data = model.name

self.course\_type.data = model.course\_type\_id

class EditUserForm(FlaskForm):

full\_name = StringField('РџРѕР»РЅРѕРµ РёРјСЏ', validators=[DataRequired()])

username = StringField('РРјСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ', validators=[DataRequired(),

Length(min=6, message='РРјСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ РґРѕР»Р¶РЅРѕ СЃРѕРґРµСЂР¶Р°С‚СЊ РјРёРЅРёРјСѓРј 6 СЃРёРјРІРѕР»РѕРІ')])

password = StringField('РџР°СЂРѕР»СЊ', validators=[Optional(), Length(min=6, message='РџР°СЂРѕР»СЊ РґРѕР»Р¶РµРЅ СЃРѕРґРµСЂР¶Р°С‚СЊ РјРёРЅРёРјСѓРј 6 СЃРёРјРІРѕР»РѕРІ')])

def from\_model(self, model):

self.full\_name.data = model.full\_name

self.username.data = model.username

submit = SubmitField('РЎРѕР·РґР°С‚СЊ')

class UploadForm(FlaskForm):

file = FileField('Р’С‹Р±РµСЂРёС‚Рµ С„Р°Р№Р»', validators=[DataRequired()])

submit = SubmitField('Р—Р°РіСЂСѓР·РёС‚СЊ')

class TeachersCoursesForm(FlaskForm):

course\_name = StringField('РќР°Р·РІР°РЅРёРµ РєСѓСЂСЃР°', validators=[Optional()])

user\_full\_name = StringField('РРјСЏ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ', validators=[Optional()])

course\_type\_id = SelectField('РўРёРї РєСѓСЂСЃРѕРІ')

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(TeachersCoursesForm, self).\_\_init\_\_(\*args, \*\*kwargs)

course\_types = course\_type\_service.get\_all()

self.course\_type\_id.choices = [(t.id, t.name) for t in course\_types]

self.course\_type\_id.choices.insert(0, ('', ''))

class TeacherCourseForm(FlaskForm):

date\_completion = DateField('Р”Р°С‚Р° РїСЂРѕС…РѕР¶РґРµРЅРёСЏ', format='%Y-%m-%d', validators=[DataRequired()])

confirming\_document = StringField('в„– РџРѕРґС‚РІРµСЂР¶РґР°СЋС‰РµРіРѕ РґРѕРєСѓРјРµРЅС‚Р°', validators=[Optional()])

submit = SubmitField('РћС‚РїСЂР°РІРёС‚СЊ')

def from\_model(self, model):

self.date\_completion.data = model.date\_completion

self.confirming\_document.data = model.confirming\_document

class ReportForm(FlaskForm):

date\_from = DateField('РќР°С‡Р°Р»СЊРЅР°СЏ РґР°С‚Р° РѕС‚С‡РµС‚Р°')

date\_to = DateField('РљРѕРЅРµС‡РЅР°СЏ РґР°С‚Р° РѕС‚С‡РµС‚Р°')

filter\_id = SelectField(validators=[DataRequired()], coerce=int)

generate = SubmitField('РЎС„РѕСЂРјРёСЂРѕРІР°С‚СЊ РѕС‚С‡РµС‚')

download = SubmitField('РЎРєР°С‡Р°С‚СЊ РѕС‚С‡РµС‚')

def \_\_init\_\_(self, filter=None, filter\_name=None, \*args, \*\*kwargs):

super(ReportForm, self).\_\_init\_\_(\*args, \*\*kwargs)

self.filter\_id.choices = [(f.id, f.name) for f in filter]

if filter\_name:

self.filter\_id.label.text = filter\_name

from flask import Blueprint

bp = Blueprint('main', \_\_name\_\_)

from app.main import routes

from flask import flash, jsonify, redirect, render\_template, request, url\_for

from flask\_login import login\_required

from app.decorators.role\_decorator import required\_role

from app.dto.course\_dto import CourseDTO

from app.main.forms import EditCourseForm

from app.models import user

from app.services import course\_service, course\_type\_service

from app.main import bp

@bp.route('/courses')

@login\_required

def courses():

page = request.args.get('page', 1, type=int)

course\_type = request.args.get('course\_type\_id', None, type=int)

courses = course\_service.get\_all\_paginated(page, course\_type)

course\_types = course\_type\_service.get\_all()

return render\_template('courses/courses.html',

title='РљСѓСЂСЃС‹',

courses=courses,

course\_types=course\_types)

@bp.route('/courses/create', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_course():

form = EditCourseForm()

if form.validate\_on\_submit():

try:

course\_service.create(CourseDTO.from\_form(form))

return redirect(url\_for('main.courses'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_course'))

return render\_template('courses/edit\_course.html',

title='РЎРѕР·РґР°С‚СЊ РєСѓСЂСЃ',

form=form)

@bp.route('/courses/edit/<course\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def edit\_course(course\_id):

form = EditCourseForm()

if form.validate\_on\_submit():

try:

course\_service.update(CourseDTO.from\_form(form, course\_id))

return redirect(url\_for('main.courses'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.edit\_course', course\_id=course\_id))

try:

course = course\_service.get\_by\_id(course\_id)

form.from\_model(course)

except ValueError as e:

flash(e)

return redirect(request.referrer or url\_for('main.courses'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё РєСѓСЂСЃР°')

return redirect(request.referrer or url\_for('main.courses'))

return render\_template('courses/edit\_course.html',

title='Р РµРґР°РєС‚РёСЂРѕРІР°С‚СЊ РєСѓСЂСЃ',

form=form)

@bp.route('/courses/delete/<course\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def delete\_course(course\_id):

try:

course\_service.delete(course\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

from flask import flash, jsonify, redirect, render\_template, url\_for

from flask\_login import login\_required

from app.decorators.role\_decorator import required\_role

from app.dto.course\_type\_dto import CourseTypeDTO

from app.main.forms import EditCourseTypeForm

from app.models import user

from app.services import course\_type\_service

from app.main import bp

@bp.route('/course\_types')

@login\_required

@required\_role(role=user.ADMIN)

def course\_types():

return render\_template('course\_types/course\_types.html',

title='РўРёРїС‹ РєСѓСЂСЃРѕРІ',

course\_types=course\_type\_service.get\_all())

@bp.route('/course\_types/create', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_course\_type():

form = EditCourseTypeForm()

if form.validate\_on\_submit():

try:

course\_type\_service.create(CourseTypeDTO.from\_form(form))

return redirect(url\_for('main.course\_types'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_course\_type'))

return render\_template('course\_types/edit\_course\_type.html',

title='РЎРѕР·РґР°С‚СЊ С‚РёРї РєСѓСЂСЃРѕРІ',

form=form)

@bp.route('/course\_types/edit/<course\_type\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def edit\_course\_type(course\_type\_id):

form = EditCourseTypeForm()

if form.validate\_on\_submit():

try:

course\_type = course\_type\_service.update(

CourseTypeDTO.from\_form(form, course\_type\_id))

return redirect(url\_for('main.course\_types'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.edit\_course\_type', course\_type\_id=course\_type\_id))

try:

course\_type = course\_type\_service.get\_by\_id(course\_type\_id)

form.from\_model(course\_type)

except ValueError as e:

flash(e)

return redirect(url\_for('main.course\_types'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё С‚РёРїР° РєСѓСЂСЃР°')

return redirect(url\_for('main.course\_types'))

return render\_template('course\_types/edit\_course\_type.html',

title='Р РµРґР°РєС‚РёСЂРѕРІР°С‚СЊ С‚РёРї РєСѓСЂСЃРѕРІ',

form=form)

@bp.route('/course\_types/delete/<course\_type\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def delete\_course\_type(course\_type\_id):

try:

course\_type\_service.delete(course\_type\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

from flask import flash, jsonify, redirect, render\_template, request, url\_for

from flask\_login import login\_required

from app.decorators.role\_decorator import required\_role

from app.dto.department\_dto import DepartmentDTO

from app.main.forms import EditDepartmentForm

from app.models import user

from app.services import department\_service, faculty\_service

from app.main import bp

@bp.route('/departments')

@login\_required

@required\_role(role=user.ADMIN)

def departments():

page = request.args.get('page', 1, type=int)

faculty = request.args.get('faculty\_id', None, type=int)

if faculty is not None:

faculty = [faculty]

departments = department\_service.get\_all\_paginated(page, faculty)

faculties = faculty\_service.get\_all()

return render\_template('departments/departments.html',

title='РљР°С„РµРґСЂС‹',

departments=departments,

faculties=faculties)

@bp.route('/departments/create', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_department():

form = EditDepartmentForm()

if form.validate\_on\_submit():

try:

department\_service.create(DepartmentDTO.from\_form(form))

return redirect(url\_for('main.departments'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_department'))

return render\_template('departments/edit\_department.html',

title='РЎРѕР·РґР°С‚СЊ РєР°С„РµРґСЂСѓ',

form=form)

@bp.route('/departments/edit/<department\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def edit\_department(department\_id):

form = EditDepartmentForm()

if form.validate\_on\_submit():

try:

department = department\_service.update(

DepartmentDTO.from\_form(form, department\_id))

return redirect(url\_for('main.departments'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.edit\_department', department\_id=department\_id))

try:

department = department\_service.get\_by\_id(department\_id)

form.from\_model(department)

except ValueError as e:

flash(e)

return redirect(request.referrer or url\_for('main.departments'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё РєР°С„РµРґСЂС‹')

return redirect(request.referrer or url\_for('main.departments'))

return render\_template('departments/edit\_department.html',

title='Р РµРґР°РєС‚РёСЂРѕРІР°С‚СЊ РєР°С„РµРґСЂСѓ',

form=form)

@bp.route('/departments/delete/<department\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def delete\_department(department\_id):

try:

department\_service.delete(department\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

from flask import flash, jsonify, redirect, render\_template, request, url\_for

from flask\_login import login\_required

from app.decorators.role\_decorator import required\_role

from app.dto.faculty\_dto import FacultyDTO

from app.main.forms import EditFacultyForm

from app.models import user

from app.services import faculty\_service

from app.main import bp

@bp.route('/faculties')

@login\_required

@required\_role(role=user.ADMIN)

def faculties():

return render\_template('faculties/faculties.html',

title='Р¤Р°РєСѓР»СЊС‚РµС‚С‹',

faculties=faculty\_service.get\_all())

@bp.route('/faculties/create', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_faculty():

form = EditFacultyForm()

if form.validate\_on\_submit():

try:

faculty\_service.create(FacultyDTO.from\_form(form))

return redirect(url\_for('main.faculties'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_faculty'))

return render\_template('faculties/edit\_faculty.html',

title='РЎРѕР·РґР°С‚СЊ С„Р°РєСѓР»СЊС‚РµС‚',

form=form)

@bp.route('/faculties/edit/<faculty\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def edit\_faculty(faculty\_id):

form = EditFacultyForm()

if form.validate\_on\_submit():

try:

faculty\_service.update(FacultyDTO.from\_form(form, faculty\_id))

return redirect(url\_for('main.faculties'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.edit\_faculty', faculty\_id=faculty\_id))

try:

form.from\_model(faculty\_service.get\_by\_id(faculty\_id))

except ValueError as e:

flash(e)

return redirect(request.referrer or url\_for('main.faculties'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р°')

return redirect(request.referrer or url\_for('main.faculties'))

return render\_template('faculties/edit\_faculty.html',

title='Р РµРґР°РєС‚РёСЂРѕРІР°С‚СЊ С„Р°РєСѓР»СЊС‚РµС‚',

form=form)

@bp.route('/faculties/delete/<faculty\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def delete\_faculty(faculty\_id):

try:

faculty\_service.delete(faculty\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

from flask import jsonify, render\_template, request

from flask\_login import current\_user, login\_required

from app.decorators.role\_decorator import required\_role

from app.decorators.user\_decorator import user\_required

from app.services import notification\_service, user\_service

from app.main import bp

@bp.route('/get\_messages\_count/<user\_id>')

@login\_required

@user\_required

def get\_messages\_count(user\_id):

try:

res = notification\_service.get\_user\_notifications\_count(user\_id)

return jsonify({'cnt': res})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/notifications/<user\_id>')

@login\_required

@user\_required

def notifications(user\_id):

page = request.args.get('page', 1, type=int)

notifications = user\_service.get\_notifications(

page, only\_new=False, user=current\_user)

return render\_template('notifications/notifications.html',

title='РЈРІРµРґРѕРјР»РµРЅРёСЏ',

notifications=notifications)

@bp.route('/notifications/<user\_id>/delete/<notification\_id>', methods=['DELETE'])

@login\_required

@user\_required

def delete\_notification(user\_id, notification\_id):

try:

notification\_service.delete(notification\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/notifications/<user\_id>/read/<notification\_id>', methods=['POST'])

@login\_required

@user\_required

def read\_notification(user\_id, notification\_id):

try:

notification\_service.read\_message(notification\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

import os

from flask import abort, current\_app, flash, jsonify, redirect, render\_template, request, send\_from\_directory, url\_for

from flask\_login import current\_user, login\_required

from app.decorators.role\_decorator import required\_role

from app.decorators.user\_decorator import user\_required

from app.main.forms import TeacherCourseForm, TeachersCoursesForm, UploadForm

from app.models import user

from app.services import sertificate\_service

from app.main import bp

@bp.route('/teacher\_course/<user\_id>/<course\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.TEACHER)

@user\_required

def teacher\_course(user\_id, course\_id):

teacher\_course = sertificate\_service.get(user\_id, course\_id)

form = UploadForm()

if form.validate\_on\_submit():

file = form.file.data

try:

if file:

sertificate\_service.upload\_file(user\_id, course\_id, file)

flash('Р¤Р°Р№Р» СѓСЃРїРµС€РЅРѕ Р·Р°РіСЂСѓР¶РµРЅ')

except ValueError as e:

flash(str(e))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё Р·Р°РіСЂСѓР·РєРµ С„Р°Р№Р»Р°')

finally:

return redirect(url\_for('main.teacher\_course', user\_id=user\_id, course\_id=course\_id))

return render\_template('teachers\_courses/teacher\_course.html',

title='РљСѓСЂСЃС‹ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ',

teacher\_course=teacher\_course, form=form)

@bp.route('/teacher\_course/<user\_id>/<course\_id>/completion', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def teacher\_course\_completion(user\_id, course\_id):

try:

teacher\_course = sertificate\_service.get(user\_id, course\_id)

except ValueError as e:

abort(403)

except Exception as e:

abort(500)

form = TeacherCourseForm()

if not teacher\_course.sertificate\_path:

abort(403)

if form.validate\_on\_submit():

date = form.date\_completion

confirming\_document = form.confirming\_document

try:

sertificate\_service.update\_teacher\_course(user\_id, course\_id, date.data, confirming\_document.data)

flash('Р”Р°С‚Р° РїСЂРѕС…РѕР¶РґРµРЅРёСЏ СѓСЃРїРµС€РЅРѕ РѕР±РЅРѕРІР»РµРЅР°')

return redirect(url\_for('main.teachers\_courses'))

except ValueError as e:

flash(str(e))

current\_app.logger.error(e)

return redirect(url\_for('main.teacher\_course\_completion', user\_id=user\_id, course\_id=course\_id))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё РєСѓСЂСЃР° РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ')

current\_app.logger.error(e)

return redirect(url\_for('main.teacher\_course\_completion', user\_id=user\_id, course\_id=course\_id))

try:

teacher\_course = sertificate\_service.get(user\_id, course\_id)

form.from\_model(teacher\_course)

except ValueError as e:

flash(e)

return redirect(request.referrer or url\_for('main.teachers\_courses'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё РєСѓСЂСЃР° РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ')

return redirect(request.referrer or url\_for('main.teachers\_courses'))

return render\_template('teachers\_courses/teacher\_course\_approve.html',

title='РљСѓСЂСЃ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ',

teacher\_course=teacher\_course, form=form)

@bp.route('/download\_file/<user\_id>/<course\_id>', methods=['GET', 'POST'])

@login\_required

def download\_file(user\_id, course\_id):

try:

if current\_user.role != user.ADMIN and str(current\_user.id) != str(user\_id):

abort(403)

user\_path = sertificate\_service.make\_path(user\_id, course\_id)

current\_app.logger.info(user\_path)

file = os.listdir(user\_path)[0]

current\_app.logger.info(file)

return send\_from\_directory(user\_path, file, as\_attachment=True)

except Exception as e:

current\_app.logger.info(e)

return jsonify({"error": str(e)}), 500

@bp.route('/teachers\_courses', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def teachers\_courses():

form = TeachersCoursesForm(request.args)

page = request.args.get('page', 1, type=int)

course\_name = request.args.get('course\_name', None, type=str)

user\_full\_name = request.args.get('user\_full\_name', None, type=str)

course\_type\_id = request.args.get('course\_type\_id', None, type=int)

return render\_template('teachers\_courses/teachers\_courses.html',

title='РљСѓСЂСЃС‹ РїСЂРµРїРѕРґР°РІР°С‚РµР»РµР№',

form=form,

teachers\_courses=sertificate\_service.get\_all\_paginated(page, course\_name=course\_name,

user\_full\_name=user\_full\_name,

course\_type\_id=course\_type\_id))

@bp.route('/qualification\_list', methods=['GET'])

@login\_required

@required\_role(role=user.ADMIN)

def qualification\_list():

try:

list = sertificate\_service.get\_qualification\_list()

return render\_template('teachers\_courses/qualification\_list.html',

list=list,

title='РЎРѕСЃС‚Р°РІР»РµРЅРёРµ СЃРїРёСЃРєРѕРІ')

except Exception as e:

current\_app.logger.error(e)

flash('РћС€РёР±РєР° РїСЂРё СЃРѕСЃС‚Р°РІР»РµРЅРёРё СЃРїРёСЃРєРѕРІ РїСЂРѕС…РѕР¶РґРµРЅРёСЏ РєРІР°Р»РёС„РёРєР°С†РёРё')

return redirect(request.referrer or '/')

import io

from flask import current\_app, flash, redirect, render\_template, send\_file, url\_for

from flask\_login import login\_required

from app.decorators.role\_decorator import required\_role

from app.main.forms import ReportForm

from app.models import user

from app.services import course\_type\_service, faculty\_service

from app.services.reports.course\_type\_report import CourseTypeReport

from app.services.reports.create\_report import ReportCreator

from app.services.reports.departments\_report import DepartmentsReport

from app.services.reports.faculty\_report import FacultyReport

from app.main import bp

@bp.route('/reports/faculty', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def faculty\_report():

form = ReportForm(filter=faculty\_service.get\_all(),

filter\_name='Р¤Р°РєСѓР»СЊС‚РµС‚')

if form.validate\_on\_submit():

try:

report = FacultyReport(form.filter\_id.data,

form.date\_from.data, form.date\_to.data)

if form.generate.data:

return render\_template('reports/report.html', report=report, form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Р°Рј',

title='РћС‚С‡РµС‚ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ',

report\_title='РћС‚С‡РµС‚ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ')

if form.download.data:

pdf = ReportCreator()

pdf.create\_table(

report, [36, 36, 36, 36, 36], 'РћС‚С‡РµС‚ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ')

pdf\_output = pdf.output(dest='S')

return send\_file(

io.BytesIO(pdf\_output),

mimetype='application/pdf',

as\_attachment=True,

download\_name='report.pdf'

)

except Exception as e:

current\_app.logger.error(e)

flash('РћС€РёР±РєР° РїСЂРё СЂР°Р±РѕС‚Рµ СЃ РѕС‚С‡РµС‚РѕРј')

return redirect(url\_for('main.faculty\_report'))

return render\_template('reports/report.html', form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Р°Рј',

title='РћС‚С‡РµС‚ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ',

report\_title='РћС‚С‡РµС‚ РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ')

@bp.route('/reports/course\_type', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def course\_type\_report():

form = ReportForm(filter=course\_type\_service.get\_all(),

filter\_name='РўРёРї РєСѓСЂСЃРѕРІ')

if form.validate\_on\_submit():

try:

report = CourseTypeReport(

form.filter\_id.data, form.date\_from.data, form.date\_to.data)

if form.generate.data:

return render\_template('reports/report.html', report=report, form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ С‚РёРїР°Рј РєСѓСЂСЃРѕРІ',

title='РћС‚С‡РµС‚ РїРѕ С‚РёРїСѓ РєСѓСЂСЃРѕРІ',

report\_title='РћС‚С‡РµС‚ РїРѕ С‚РёРїСѓ РєСѓСЂСЃРѕРІ')

if form.download.data:

pdf = ReportCreator()

pdf.create\_table(

report, [36, 36, 36, 36, 36], 'РћС‚С‡РµС‚ РїРѕ С‚РёРїСѓ РєСѓСЂСЃРѕРІ')

pdf\_output = pdf.output(dest='S')

return send\_file(

io.BytesIO(pdf\_output),

mimetype='application/pdf',

as\_attachment=True,

download\_name='report.pdf'

)

except Exception as e:

current\_app.logger.error(e)

flash('РћС€РёР±РєР° РїСЂРё СЂР°Р±РѕС‚Рµ СЃ РѕС‚С‡РµС‚РѕРј')

return redirect(url\_for('main.course\_type\_report'))

return render\_template('reports/report.html', form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ С‚РёРїР°Рј РєСѓСЂСЃРѕРІ',

title='РћС‚С‡РµС‚ РїРѕ С‚РёРїСѓ РєСѓСЂСЃРѕРІ',

report\_title='РћС‚С‡РµС‚ РїРѕ С‚РёРїСѓ РєСѓСЂСЃРѕРІ')

@bp.route('/reports/departments', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def departments\_report():

form = ReportForm(filter=faculty\_service.get\_all(),

filter\_name='Р¤Р°РєСѓР»СЊС‚РµС‚')

if form.validate\_on\_submit():

try:

report = DepartmentsReport(

form.filter\_id.data, form.date\_from.data, form.date\_to.data)

if form.generate.data:

return render\_template('reports/report.html', report=report, form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ РєР°С„РµРґСЂР°Рј',

title='РћС‚С‡РµС‚ РїРѕ РєР°С„РµРґСЂР°Рј С„Р°РєСѓР»СЊС‚РµС‚Р°',

report\_title='РћС‚С‡РµС‚ РїРѕ РєР°С„РµРґСЂР°Рј С„Р°РєСѓР»СЊС‚РµС‚Р°')

if form.download.data:

pdf = ReportCreator()

pdf.create\_table(

report, [60, 60, 60], 'РћС‚С‡РµС‚ РїРѕ РєР°С„РµРґСЂР°Рј С„Р°РєСѓР»СЊС‚РµС‚Р°')

pdf\_output = pdf.output(dest='S')

return send\_file(

io.BytesIO(pdf\_output),

mimetype='application/pdf',

as\_attachment=True,

download\_name='report.pdf'

)

except Exception as e:

current\_app.logger.error(e)

flash('РћС€РёР±РєР° РїСЂРё СЂР°Р±РѕС‚Рµ СЃ РѕС‚С‡РµС‚РѕРј')

return redirect(url\_for('main.departments\_report'))

return render\_template('reports/report.html', form=form,

page\_title='РћС‚С‡РµС‚С‹ РїРѕ РєР°С„РµРґСЂР°Рј',

title='РћС‚С‡РµС‚ РїРѕ РєР°С„РµРґСЂР°Рј С„Р°РєСѓР»СЊС‚РµС‚Р°',

report\_title='РћС‚С‡РµС‚ РїРѕ РєР°С„РµРґСЂР°Рј С„Р°РєСѓР»СЊС‚РµС‚Р°')

import io

from flask import abort, current\_app, flash, jsonify, make\_response, redirect, render\_template, request, send\_file, url\_for

from flask\_login import login\_required, current\_user

from app.decorators.role\_decorator import required\_role

from app.dto.notification\_dto import NotificationDTO

from app.dto.user\_dto import UserDTO

from app.main.forms import EditUserForm

from app.models import user

from app.services import course\_service, notification\_service, user\_service, department\_service

from app.main import bp

from app.services.reports.create\_direction import DirectionCreator

@bp.route('/users')

@login\_required

@required\_role(role=user.ADMIN)

def users():

page = request.args.get('page', 1, type=int)

role = request.args.get('role', None, type=int)

is\_fired = request.args.get('is\_fired', None, type=str)

if is\_fired == '':

is\_fired = None

if is\_fired is not None and is\_fired == 'True':

is\_fired = True

if is\_fired is not None and is\_fired == 'False':

is\_fired = False

users = user\_service.get\_all\_paginated(

page, UserDTO(is\_fired=is\_fired, role=role))

return render\_template('users/users.html',

title='РџРѕР»СЊР·РѕРІР°С‚РµР»Рё',

users=users)

@bp.route('/users/<user\_id>/qualifications')

@login\_required

@required\_role(role=user.ADMIN)

def create\_user\_qualifications(user\_id):

try:

page = request.args.get('page', 1, type=int)

teacher = user\_service.get\_by\_id(user\_id)

if teacher.role != user.TEACHER:

raise ValueError('РЎРѕР·РґР°С‚СЊ РєРІР°Р»РёС„РёРєР°С†РёСЋ РјРѕР¶РЅРѕ С‚РѕР»СЊРєРѕ РґР»СЏ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ')

courses = course\_service.get\_all\_paginated(page)

return render\_template('users/create\_user\_qualification.html',

title='РљРІР°Р»РёС„РёРєР°С†РёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ',

user=teacher,

courses=courses)

except ValueError as e:

abort(404)

except Exception as e:

abort(500)

@bp.route("/users/<user\_id>/qualifications/download", methods=["POST"])

@login\_required

@required\_role(role=user.ADMIN)

def download\_user\_qualifications(user\_id):

try:

teacher = user\_service.get\_by\_id(user\_id)

courses = request.form.getlist("selected")

date\_from = request.form.get("date\_from")

date\_to = request.form.get("date\_to")

place = request.form.get("place")

if not courses:

raise ValueError("РќРµ РІС‹Р±СЂР°РЅС‹ РєСѓСЂСЃС‹")

if not date\_from or not date\_to:

raise ValueError("Р”Р°С‚С‹ РЅР°С‡Р°Р»Р° Рё РѕРєРѕРЅС‡Р°РЅРёСЏ РѕР±СЏР·Р°С‚РµР»СЊРЅС‹")

pdf = DirectionCreator(teacher, courses, date\_from, date\_to, place, current\_user)

pdf.create\_table()

pdf\_output = pdf.output(dest="S")

current\_app.logger.info('Р’РљРР”')

return send\_file(

io.BytesIO(pdf\_output),

mimetype="application/pdf",

as\_attachment=True,

download\_name="direction.pdf"

)

except ValueError as e:

current\_app.logger.warning(f"Validation error: {e}")

if request.headers.get("X-Requested-With") == "XMLHttpRequest":

return jsonify({"success": False, "error": str(e)}), 400

flash(str(e))

return redirect(request.referrer or '/')

except Exception as e:

current\_app.logger.exception("РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё РЅР°РїСЂР°РІР»РµРЅРёСЏ")

if request.headers.get("X-Requested-With") == "XMLHttpRequest":

return jsonify({"success": False, "error": "Р’РЅСѓС‚СЂРµРЅРЅСЏСЏ РѕС€РёР±РєР° СЃРµСЂРІРµСЂР°"}), 500

flash("РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё РЅР°РїСЂР°РІР»РµРЅРёСЏ")

return redirect(request.referrer or '/')

@bp.route('/users/create\_admin', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_admin():

form = EditUserForm()

if form.validate\_on\_submit():

try:

user\_service.create(UserDTO.from\_form(form, 0))

return redirect(url\_for('main.users'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_admin'))

return render\_template('users/edit\_user.html',

title='РЎРѕР·РґР°С‚СЊ СЃРѕС‚СЂСѓРґРЅРёРєР°',

form=form)

@bp.route('/users/create\_teacher', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def create\_teacher():

form = EditUserForm()

if form.validate\_on\_submit():

try:

user\_service.create(UserDTO.from\_form(form, 1))

return redirect(url\_for('main.users'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.create\_teacher'))

return render\_template('users/edit\_user.html',

title='РЎРѕР·РґР°С‚СЊ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ',

form=form)

@bp.route('/users/edit/<user\_id>', methods=['GET', 'POST'])

@login\_required

@required\_role(role=user.ADMIN)

def edit\_user(user\_id):

form = EditUserForm()

if form.validate\_on\_submit():

try:

user\_service.update(UserDTO.from\_form(form, id=user\_id))

return redirect(url\_for('main.users'))

except Exception as e:

flash(str(e))

return redirect(url\_for('main.edit\_user', user\_id=user\_id))

try:

page = request.args.get('page', 1, type=int)

search\_request = request.args.get('department\_name', '', type=int)

user = user\_service.get\_by\_id(user\_id)

form.from\_model(user)

except ValueError as e:

flash(e)

return redirect(request.referrer or url\_for('main.faculties'))

except Exception as e:

flash('РћС€РёР±РєР° РїСЂРё СЂРµРґР°РєС‚РёСЂРѕРІР°РЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

return redirect(request.referrer or url\_for('main.faculties'))

return render\_template('users/edit\_user.html', form=form, user=user,

user\_departments=user\_service.get\_departments(

UserDTO(user\_id)),

title='Р РµРґР°РєС‚РёСЂРѕРІР°С‚СЊ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ',

departments=department\_service.get\_all\_paginated(page=page, search\_request=search\_request))

@bp.route('/users/delete/<user\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def delete\_user(user\_id):

try:

if current\_user.get\_id() == user\_id:

return jsonify({"error": 'РќРµР»СЊР·СЏ СѓРґР°Р»РёС‚СЊ СЃРµР±СЏ'}), 500

user\_service.delete(user\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/users/fire/<user\_id>', methods=['POST'])

@login\_required

@required\_role(role=user.ADMIN)

def fire\_user(user\_id):

try:

if current\_user.get\_id() == user\_id:

return jsonify({"error": 'РќРµР»СЊР·СЏ Р·Р°Р±Р»РѕРєРёСЂРѕРІР°С‚СЊ СЃРµР±СЏ'}), 500

user\_service.fire(user\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/users/<user\_id>/remove\_from\_department/<department\_id>', methods=['DELETE'])

@login\_required

@required\_role(role=user.ADMIN)

def remove\_from\_department(user\_id, department\_id):

try:

user\_service.remove\_from\_department(user\_id, department\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/users/<user\_id>/add\_to\_department/<department\_id>', methods=['POST'])

@login\_required

@required\_role(role=user.ADMIN)

def add\_to\_department(user\_id, department\_id):

try:

user\_service.add\_to\_department(user\_id, department\_id)

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

@bp.route('/users/send\_notification/<user\_id>', methods=['POST'])

@login\_required

@required\_role(role=user.ADMIN)

def send\_notification(user\_id):

try:

data = request.get\_json()

msg = data.get('message')

notification\_service.send\_message(

NotificationDTO(user\_id=user\_id, message=msg))

return jsonify({'success': True})

except Exception as e:

return jsonify({"error": str(e)}), 500

from flask import render\_template

from flask\_login import login\_required

from app.main import bp

@bp.route('/')

@bp.route('/index')

@login\_required

def index():

return render\_template('index.html')

import app.main.routes.faculty\_routes, app.main.routes.department\_routes, app.main.routes.course\_type\_routes

import app.main.routes.course\_routes, app.main.routes.user\_routes, app.main.routes.qualification\_routes

import app.main.routes.notification\_routes, app.main.routes.reports\_routes

from datetime import datetime

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

class Course(db.Model):

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

name: so.Mapped[str] = so.mapped\_column(

sa.String(128), index=True, unique=True)

course\_type\_id: so.Mapped[int] = so.mapped\_column(

sa.ForeignKey('course\_type.id', ondelete='CASCADE'), index=True)

course\_type: so.Mapped['CourseType'] = so.relationship(

back\_populates='courses')

teachers\_courses: so.WriteOnlyMapped['TeacherCourse'] = so.relationship(

back\_populates='course', passive\_deletes=True)

def \_\_repr\_\_(self):

return f'Course {self.name}'

@classmethod

def from\_form(cls, form, course\_type):

return cls(name=form.name.data, course\_type=course\_type)

def to\_dict(self):

data = {

'id': self.id,

'name': self.name,

'course\_type\_id': self.course\_type\_id

}

return data

from typing import Optional

from datetime import datetime, timezone

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

class CourseType(db.Model):

\_\_tablename\_\_ = 'course\_type'

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

name: so.Mapped[str] = so.mapped\_column(

sa.String(128), index=True, unique=True)

courses: so.WriteOnlyMapped['Course'] = so.relationship(

back\_populates='course\_type', passive\_deletes=True)

def \_\_repr\_\_(self):

return f'Course Type {self.name}'

@classmethod

def from\_form(cls, form, faculty):

return cls(name=form.name.data)

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

from app.models.tables.teachers\_departments import teachers\_departments

class Department(db.Model):

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

name: so.Mapped[str] = so.mapped\_column(

sa.String(128), index=True, unique=True)

faculty\_id: so.Mapped[int] = so.mapped\_column(

sa.ForeignKey('faculty.id', ondelete='CASCADE'), index=True)

faculty: so.Mapped['Faculty'] = so.relationship(

back\_populates='departments')

teachers: so.WriteOnlyMapped['User'] = so.relationship(

secondary=teachers\_departments, primaryjoin=(teachers\_departments.c.department\_id == id), back\_populates='departments', passive\_deletes=True)

def \_\_repr\_\_(self):

return f'Department {self.name}'

@classmethod

def from\_form(cls, form, faculty):

return cls(name=form.name.data, faculty=faculty)

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

class Faculty(db.Model):

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

name: so.Mapped[str] = so.mapped\_column(

sa.String(128), index=True, unique=True)

departments: so.WriteOnlyMapped['Department'] = so.relationship(

back\_populates='faculty', passive\_deletes=True)

def \_\_repr\_\_(self):

return f'Faculty {self.name}'

@classmethod

def from\_form(cls, form):

return cls(name=form.name.data)

from datetime import datetime, timezone

from flask import url\_for

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

class Notification(db.Model):

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

message: so.Mapped[str] = so.mapped\_column(

sa.String(256), index=True)

user\_id: so.Mapped[int] = so.mapped\_column(

sa.ForeignKey('user.id', ondelete='CASCADE'), index=True)

user: so.Mapped['User'] = so.relationship(

back\_populates='notifications')

has\_read: so.Mapped[bool] = so.mapped\_column(sa.Boolean, default=False)

time\_sent: so.Mapped[datetime] = so.mapped\_column(

index=True, default=lambda: datetime.now())

def \_\_repr\_\_(self):

return f'Notification {self.message}'

@classmethod

def from\_form(cls, form, user):

return cls(name=form.message.data, user=user)

def to\_dict(self):

data = {

'id': self.id,

'message': self.message,

'user\_id': self.user\_id,

'has\_read': self.has\_read,

'time\_sent': self.time\_sent.isoformat(),

'\_links': {

'self': url\_for('api.get\_notification', id=self.id),

}

}

return data

from datetime import datetime

from typing import Optional

from flask import url\_for

import sqlalchemy as sa

import sqlalchemy.orm as so

from app import db

class TeacherCourse(db.Model):

teacher\_id: so.Mapped[int] = so.mapped\_column(

sa.ForeignKey('user.id', ondelete='CASCADE'), primary\_key=True, index=True)

teacher: so.Mapped['User'] = so.relationship(

back\_populates='courses')

course\_id: so.Mapped[int] = so.mapped\_column(

sa.ForeignKey('course.id', ondelete='CASCADE'), primary\_key=True, index=True)

course: so.Mapped['Course'] = so.relationship(

back\_populates='teachers\_courses')

sertificate\_path: so.Mapped[Optional[str]] = so.mapped\_column(

sa.String(260), unique=True, nullable=True)

date\_completion: so.Mapped[Optional[datetime]

] = so.mapped\_column(sa.Date, nullable=True)

confirming\_document: so.Mapped[Optional[str]] = so.mapped\_column(

sa.String(64), index=True, nullable=True

)

def \_\_repr\_\_(self):

return f'Teacher Course {self.teacher.full\_name}, {self.course.name}'

def to\_dict(self):

data = {

'teacher\_id': self.teacher\_id,

'course\_id': self.course\_id,

'date\_completion': (self.date\_completion.isoformat() if self.date\_completion else None),

'course\_name': self.course.name,

'sertificate\_loaded': (self.sertificate\_path is not None),

'confirming\_document': self.confirming\_document,

'\_links': {

'self': url\_for('api.get\_teacher\_course', user\_id=self.teacher\_id, course\_id=self.course\_id),

}

}

return data

from datetime import datetime, timedelta

from typing import Optional

from flask import url\_for

import sqlalchemy as sa

import sqlalchemy.orm as so

from flask\_login import UserMixin

from app import db

from app.models.tables.teachers\_departments import teachers\_departments

ADMIN = 0

TEACHER = 1

class User(UserMixin, db.Model):

id: so.Mapped[int] = so.mapped\_column(primary\_key=True)

username: so.Mapped[str] = so.mapped\_column(

sa.String(64), index=True, unique=True)

full\_name: so.Mapped[str] = so.mapped\_column(

sa.String(128))

password\_hash: so.Mapped[Optional[str]] = so.mapped\_column(sa.String(256))

role: so.Mapped[int] = so.mapped\_column(sa.Integer, index=True)

is\_fired: so.Mapped[bool] = so.mapped\_column(

sa.Boolean, index=True, default=False)

notifications: so.WriteOnlyMapped['Notification'] = so.relationship(

back\_populates='user', passive\_deletes=True)

departments: so.WriteOnlyMapped['Department'] = so.relationship(

secondary=teachers\_departments, primaryjoin=(teachers\_departments.c.teacher\_id == id), back\_populates='teachers', passive\_deletes=True)

courses: so.WriteOnlyMapped['TeacherCourse'] = so.relationship(

back\_populates='teacher', passive\_deletes=True)

token: so.Mapped[Optional[str]] = so.mapped\_column(

sa.String(32), index=True, unique=True)

token\_expiration: so.Mapped[Optional[datetime]]

def \_\_repr\_\_(self):

return f'User {self.full\_name}'

def to\_dict(self):

data = {

'id': self.id,

'username': self.username,

'full\_name': self.full\_name,

'role': ('ADMIN' if self.role == 0 else 'TEACHER'),

'token\_available': (self.token\_expiration is not None and self.token\_expiration > datetime.now() + timedelta(seconds=60)),

'\_links': {

'self': url\_for('api.get\_user', user\_id=self.id),

'notifications': url\_for('api.get\_user\_notifications', user\_id=self.id),

'teacher\_courses': url\_for('api.get\_teacher\_courses', user\_id=self.id),

'revoke\_token': url\_for('api.revoke\_user\_token', user\_id=self.id),

}

}

return data

from app.models.user import User

from app.models.faculty import Faculty

from app.models.department import Department

from app.models.course\_type import CourseType

from app.models.course import Course

from app.models.teacher\_course import TeacherCourse

from app.models.notification import Notification

import sqlalchemy as sa

from app import db

teachers\_departments = sa.Table('teachers\_departments',

db.metadata,

sa.Column('teacher\_id', sa.Integer, sa.ForeignKey(

'user.id'), primary\_key=True),

sa.Column('department\_id', sa.Integer, sa.ForeignKey('department.id', ondelete='CASCADE'), primary\_key=True),)

from app.models.department import Department

from flask import current\_app

import sqlalchemy as sa

from app import db

from app.dto.course\_dto import CourseDTO

from app.models.course import Course

from app.services import course\_type\_service

def get\_all\_paginated(page: int, course\_types=None):

try:

if course\_types is not None and not isinstance(course\_types, list):

course\_types = [course\_types]

query = sa.select(Course)

conditions = []

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {course\_types}")

if course\_types is not None:

conditions.append(Course.course\_type\_id.in\_(course\_types))

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {conditions}")

if conditions:

query = query.where(sa.and\_(\*conditions))

query = query.order\_by(Course.name)

return db.paginate(query, page=page, per\_page=current\_app.config['COURSES\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃРѕРІ')

def get\_by\_id(id: int):

try:

res = db.session.get(Course, id)

if res is None:

raise ValueError(f'РљСѓСЂСЃ СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃР° РїРѕ id')

def get\_by\_name(name: str):

try:

res = db.session.execute(sa.select(Course).where(

Course.name == name)).scalar\_one\_or\_none()

if res is None:

raise ValueError(f'РљСѓСЂСЃ СЃ РёРјРµРЅРµРј {name} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃР° РїРѕ РЅР°Р·РІР°РЅРёСЋ')

def create(courseDTO: CourseDTO):

try:

if courseDTO.course\_type\_id is None:

raise ValueError('РќРµ СѓРєР°Р·Р°РЅ id С‚РёРїР° РєСѓСЂСЃР°')

course\_type = course\_type\_service.get\_by\_id(courseDTO.course\_type\_id)

course = Course(name=courseDTO.name, course\_type=course\_type)

db.session.add(course)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(f'РљСѓСЂСЃ СЃ РёРјРµРЅРµРј {course.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё РєСѓСЂСЃР°')

def update(courseDTO: CourseDTO):

try:

record = get\_by\_id(courseDTO.id)

if courseDTO.name is not None:

record.name = courseDTO.name

if courseDTO.course\_type\_id is not None:

course\_type = course\_type\_service.get\_by\_id(

courseDTO.course\_type\_id)

record.course\_type = course\_type

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё РєСѓСЂСЃР°')

def delete(id: int) -> bool:

try:

course = get\_by\_id(id)

db.session.delete(course)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё РєСѓСЂСЃР°')

import sqlalchemy as sa

from flask import current\_app

from app import db

from app.dto.course\_type\_dto import CourseTypeDTO

from app.models.course\_type import CourseType

def get\_all():

try:

return db.session.execute(sa.select(CourseType).order\_by(CourseType.name)).scalars().all()

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё С‚РёРїРѕРІ РєСѓСЂСЃРѕРІ')

def get\_by\_id(id: int):

try:

res = db.session.get(CourseType, id)

if res is None:

raise ValueError(f'РўРёРї РєСѓСЂСЃРѕРІ СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё С‚РёРїР° РєСѓСЂСЃРѕРІ РїРѕ id')

def get\_by\_name(name: str):

try:

res = db.session.execute(sa.select(CourseType).where(

CourseType.name == name)).scalar\_one\_or\_none()

if res is None:

raise ValueError(f'РўРёРї РєСѓСЂСЃРѕРІ СЃ РёРјРµРЅРµРј {name} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃР° РїРѕ РЅР°Р·РІР°РЅРёСЋ')

def create(courseTypeDTO: CourseTypeDTO):

try:

courseTypeDTO = CourseType(name=courseTypeDTO.name)

db.session.add(courseTypeDTO)

db.session.commit()

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'РўРёРї РєСѓСЂСЃРѕРІ СЃ РёРјРµРЅРµРј {courseTypeDTO.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё С‚РїР° РєСѓСЂСЃРѕРІ')

def update(courseTypeDTO: CourseTypeDTO):

try:

record = get\_by\_id(courseTypeDTO.id)

record.name = courseTypeDTO.name

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'РўРёРї РєСѓСЂСЃРѕРІ СЃ РёРјРµРЅРµРј {courseTypeDTO.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё С‚РёРїР° РєСѓСЂСЃРѕРІ')

def delete(id: int) -> bool:

try:

course\_type = get\_by\_id(id)

db.session.delete(course\_type)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё С‚РёРїР° РєСѓСЂСЃРѕРІ')

from flask import current\_app

import sqlalchemy as sa

from app import db

from app.dto.department\_dto import DepartmentDTO

from app.exceptions.role\_error import RoleError

from app.models.department import Department

from app.models.user import User

from app.services import faculty\_service

def get\_all(faculties=None):

try:

query = sa.select(Department)

if faculties is not None:

query = query.where(Department.faculty\_id.in\_(faculties))

return db.session.execute(query).scalars().all()

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂ')

def get\_all\_paginated(page: int, faculties=None, search\_request=None):

try:

query = sa.select(Department)

conditions = []

if faculties is not None:

conditions.append(Department.faculty\_id.in\_(faculties))

if search\_request is not None:

conditions.append(Department.name.ilike(f'%{search\_request}%'))

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {conditions}")

if conditions:

query = query.where(sa.and\_(\*conditions))

query = query.order\_by(Department.faculty\_id)

return db.paginate(query, page=page, per\_page=current\_app.config['DEPARTMENTS\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂ')

def get\_by\_id(id: int):

try:

res = db.session.get(Department, id)

if res is None:

raise ValueError(f'РљР°С„РµРґСЂР° СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂС‹ РїРѕ id')

def get\_by\_name(name: str):

try:

res = db.session.execute(sa.select(Department).where(

Department.name == name)).scalar\_one\_or\_none()

if res is None:

raise ValueError(f'РљР°С„РµРґСЂР° СЃ РёРјРµРЅРµРј {name} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂС‹ РїРѕ РЅР°Р·РІР°РЅРёСЋ')

def create(departmentDTO: DepartmentDTO):

try:

if departmentDTO.faculty\_id is None:

raise ValueError('РќРµ СѓРєР°Р·Р°РЅ id С„Р°РєСѓР»СЊС‚РµС‚Р°')

faculty = faculty\_service.get\_by\_id(departmentDTO.faculty\_id)

department = Department(name=departmentDTO.name, faculty=faculty)

db.session.add(department)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'РљР°С„РµРґСЂР° СЃ РёРјРµРЅРµРј {departmentDTO.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё РєР°С„РµРґСЂС‹')

def get\_teachers(page: int, department: Department):

try:

query = department.teachers.select()

return db.paginate(query, page=page, per\_page=current\_app.config['TEACHERS\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїСЂРµРїРѕРґР°РІР°С‚РµР»РµР№ РєР°С„РµРґСЂС‹')

def update(departmentDTO: DepartmentDTO):

try:

record = get\_by\_id(departmentDTO.id)

if departmentDTO.name is not None:

record.name = departmentDTO.name

if departmentDTO.faculty\_id is not None:

faculty = faculty\_service.get\_by\_id(departmentDTO.faculty\_id)

record.faculty = faculty

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё РєР°С„РµРґСЂС‹')

def add\_teacher(id: int, user: User):

try:

record = get\_by\_id(id)

if (user.role != user.TEACHER):

raise RoleError('РќР° РєР°С„РµРґСЂСѓ РґРѕР±Р°РІР»СЏРµС‚СЃСЏ РЅРµ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЊ')

record.teachers.add(user)

db.session.commit()

except RoleError as e:

current\_app.logger.error(e)

raise

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РґРѕР±Р°РІР»РµРЅРёРё РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ РЅР° РєР°С„РµРґСЂСѓ')

def delete(id: int) -> bool:

try:

department = get\_by\_id(id)

db.session.delete(department)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё РєР°С„РµРґСЂС‹')

from flask import current\_app

import sqlalchemy as sa

from app import db

from app.models.faculty import Faculty

from app.dto.faculty\_dto import FacultyDTO

def get\_all():

try:

return db.session.execute(sa.select(Faculty).order\_by(Faculty.name)).scalars().all()

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚РѕРІ')

def get\_by\_id(id: int):

try:

res = db.session.get(Faculty, id)

if res is None:

raise ValueError(f'Р¤Р°РєСѓР»СЊС‚РµС‚ СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р° РїРѕ id')

def get\_by\_name(name: str):

try:

res = db.session.execute(sa.select(Faculty).where(

Faculty.name == name)).scalar\_one\_or\_none()

if res is None:

raise ValueError(f'Р¤Р°РєСѓР»СЊС‚РµС‚ СЃ РёРјРµРЅРµРј {name} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р° РїРѕ РЅР°Р·РІР°РЅРёСЋ')

def create(facultyDTO: FacultyDTO):

try:

faculty = Faculty(name=facultyDTO.name)

db.session.add(faculty)

db.session.commit()

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'Р¤Р°РєСѓР»СЊС‚РµС‚ СЃ РёРјРµРЅРµРј {facultyDTO.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р°')

def update(facultyDTO: FacultyDTO):

try:

record = get\_by\_id(facultyDTO.id)

record.name = facultyDTO.name

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'Р¤Р°РєСѓР»СЊС‚РµС‚ СЃ РёРјРµРЅРµРј {facultyDTO.name} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р°')

def delete(id: int) -> bool:

try:

faculty = get\_by\_id(id)

db.session.delete(faculty)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р°')

def get\_departments(facultyDTO: FacultyDTO):

try:

faculty = None

if facultyDTO.id is not None:

faculty = get\_by\_id(facultyDTO.id)

else:

faculty = get\_by\_name(facultyDTO.name)

return db.session.scalars(faculty.departments.select()).all()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂ С„Р°РєСѓР»СЊС‚РµС‚Р°')

from flask import current\_app

import sqlalchemy as sa

from app import db

from app.models.notification import Notification

from app.dto.notification\_dto import NotificationDTO

from app.services import user\_service

def notifications\_key(user\_id):

return f"notifications:{user\_id}"

def get\_by\_id(id: int):

try:

res = db.session.get(Notification, id)

if res is None:

raise ValueError(f'РЈРІРµРґРѕРјР»РµРЅРёРµ СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё СѓРІРµРґРѕРјР»РµРЅРёСЏ РїРѕ id')

def send\_message(notificationDTO: NotificationDTO):

try:

user = user\_service.get\_by\_id(notificationDTO.user\_id)

notification = Notification(

message=notificationDTO.message, user=user, has\_read=False)

db.session.add(notification)

db.session.commit()

r = current\_app.config['SESSION\_REDIS']

r.delete(notifications\_key(user.id))

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РѕС‚РїСЂР°РІРєРµ СЃРѕРѕР±С‰РµРЅРёСЏ')

def read\_message(id):

try:

notification = get\_by\_id(id)

notification.has\_read = True

db.session.commit()

r = current\_app.config['SESSION\_REDIS']

r.delete(notifications\_key(notification.user\_id))

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїСЂРѕС‡С‚РµРЅРёРё СЃРѕРѕР±С‰РµРЅРёСЏ')

def get\_user\_notifications\_count(user\_id):

try:

r = current\_app.config['SESSION\_REDIS']

cur\_key = notifications\_key(user\_id)

if r.exists(cur\_key):

return int(r.get(cur\_key))

query = sa.select(sa.func.count()).where(

sa.and\_(Notification.has\_read == False, Notification.user\_id == user\_id))

res = db.session.scalar(query)

r.setex(cur\_key, 86400, res)

return res

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєРѕР»РёС‡РµСЃС‚РІР° СЃРѕРѕР±С‰РµРЅРёР№')

def delete(id):

try:

notification = get\_by\_id(id)

db.session.delete(notification)

db.session.commit()

r = current\_app.config['SESSION\_REDIS']

r.delete(notifications\_key(notification.user\_id))

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё СЃРѕРѕР±С‰РµРЅРёСЏ')

from datetime import date, datetime

import shutil

import os

from flask import current\_app

import sqlalchemy as sa

from werkzeug.utils import secure\_filename

from app import db

from app.models import user

from app.models.user import User

from app.models.course import Course

from app.services import user\_service, course\_service

from app.models.teacher\_course import TeacherCourse

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in current\_app.config['ALLOWED\_EXTENSIONS']

def make\_path(user\_id: int, course\_id: int):

user\_path = os.path.join(

current\_app.config['UPLOAD\_FOLDER'], user\_id, course\_id)

project\_root = os.path.abspath(os.path.join(

os.path.join(os.path.dirname(\_\_file\_\_), '..'), '..'))

user\_path = os.path.abspath(os.path.join(project\_root, user\_path))

return user\_path

def upload\_file(user\_id: int, course\_id: int, file):

try:

if file.filename == '':

raise ValueError('РќРµС‚ РІС‹Р±СЂР°РЅРЅРѕРіРѕ С„Р°Р№Р»Р°')

if not allowed\_file(file.filename):

raise ValueError('РќРµРєРѕСЂСЂРµРєС‚РЅРѕРµ СЂР°СЃС€РёСЂРµРЅРёРµ С„Р°Р№Р»Р°')

filename = secure\_filename(file.filename)

user\_path = make\_path(user\_id, course\_id)

if os.path.exists(user\_path):

shutil.rmtree(user\_path)

os.makedirs(user\_path, exist\_ok=True)

filepath = os.path.join(user\_path, filename)

file.save(filepath)

res = get(user\_id, course\_id)

res.sertificate\_path = filepath

res.date\_completion = None

res.confirming\_document = None

db.session.commit()

except Exception as e:

current\_app.logger.error(e)

raise

def get(user\_id: int, course\_id: int):

try:

res = db.session.get(TeacherCourse, (user\_id, course\_id))

if res is None:

teacher = user\_service.get\_by\_id(user\_id)

if teacher.role != user.TEACHER:

raise ValueError('РљСѓСЂСЃС‹ РµСЃС‚СЊ С‚РѕР»СЊРєРѕ Сѓ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ')

course = course\_service.get\_by\_id(course\_id)

db.session.add(TeacherCourse(teacher=teacher, course=course))

db.session.commit()

res = db.session.get(TeacherCourse, (user\_id, course\_id))

current\_app.logger.info(res)

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃР° РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def get\_user\_courses(user\_id: int, page: int):

try:

query = sa.select(TeacherCourse)

conditions = []

conditions.append(TeacherCourse.teacher\_id == user\_id)

if conditions:

query = query.where(\*conditions)

return db.paginate(query, page=page, per\_page=current\_app.config['COURSES\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃРѕРІ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def get\_all\_paginated(page: int, course\_name=None, user\_full\_name=None, course\_type\_id=None):

try:

if course\_name is not None and len(course\_name) == 0:

course\_name = None

if user\_full\_name is not None and len(user\_full\_name) == 0:

user\_full\_name = None

current\_app.logger.info(

f"page = {page}, course = {course\_name}, user = {user\_full\_name}, course\_type = {course\_type\_id}")

query = sa.select(TeacherCourse, Course, User).join(Course).join(User)

conditions = []

if course\_name is not None:

conditions.append(sa.func.lower(

Course.name).like(f'%{course\_name.lower()}%'))

if user\_full\_name is not None:

conditions.append(sa.func.lower(User.full\_name).like(

f'%{user\_full\_name.lower()}%'))

if course\_type\_id is not None:

conditions.append(Course.course\_type\_id == course\_type\_id)

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {conditions}")

if conditions:

query = query.where(sa.and\_(\*conditions))

query = query.order\_by(Course.name, User.full\_name, User.username)

return db.paginate(query, page=page, per\_page=current\_app.config['COURSES\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃРѕРІ')

def update\_teacher\_course(user\_id: int, course\_id: int, date\_completion: date, confirming\_document: str = None):

try:

if confirming\_document is not None:

confirming\_document = confirming\_document.strip()

if len(confirming\_document) == 0:

confirming\_document = None

res = get(user\_id, course\_id)

if res.sertificate\_path is None:

raise ValueError('РЎРµСЂС‚РёС„РёРєР°С‚ РЅРµ РїСЂРёРєСЂРµРїР»РµРЅ')

res.date\_completion = date\_completion

res.confirming\_document = confirming\_document

db.session.commit()

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def get\_qualification\_list():

cur\_date = datetime.now().strftime("%Y-%m-%d")

try:

query = sa.text("SELECT public.user.full\_name, public.user.username, course.name,"

" course\_type.name, teacher\_course.date\_completion, teacher\_course.confirming\_document "

"FROM public.user CROSS JOIN course JOIN course\_type ON course\_type.id = "

"course.course\_type\_id LEFT JOIN teacher\_course ON "

"teacher\_course.teacher\_id = public.user.id AND teacher\_course.course\_id = "

"course.id WHERE public.user.role = :role AND "

"(teacher\_course.date\_completion IS NULL OR "

"EXTRACT(YEAR FROM AGE(:cur\_date, teacher\_course.date\_completion )) >= 3) "

"ORDER BY public.user.full\_name, public.user.username;")

qualification\_list = db.session.execute(

query, {'cur\_date': cur\_date, 'role': user.TEACHER})

return qualification\_list

except Exception as e:

current\_app.logger.info(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё СЃРїРёСЃРєР° РїСЂРѕС…РѕР¶РґРµРЅРёСЏ РєРІР°Р»РёС„РёРєР°С†РёР№')

from flask import current\_app, url\_for

def to\_json\_collection(resources, endpoint, \*\*kwargs):

data = {

'items': [item.to\_dict() for item in resources.items],

'\_meta': {

'page': resources.page,

'per\_page': resources.per\_page,

'total\_pages': resources.pages,

'total\_items': resources.total

},

'\_links': {

'self': url\_for(endpoint, page=resources.page, \*\*kwargs),

'next': url\_for(endpoint, page=resources.page + 1,

\*\*kwargs) if resources.has\_next else None,

'prev': url\_for(endpoint, page=resources.page - 1,

\*\*kwargs) if resources.has\_prev else None

}

}

current\_app.logger.info(f'data: {data}')

return data

from datetime import datetime, timedelta

import secrets

from flask import current\_app

import sqlalchemy as sa

from app.exceptions.fired\_error import FiredError

from app.exceptions.wrong\_password\_error import WrongPasswordError

from app import db

from app.models.user import User

from app.models.teacher\_course import TeacherCourse

from app.models.course import Course

from app.dto.user\_dto import UserDTO

from app.services import department\_service

from app.models.user import TEACHER

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from app.models.notification import Notification

def get\_all(userDTO: UserDTO):

try:

query = sa.select(User)

conditions = []

if userDTO.role is not None:

conditions.append(User.role == userDTO.role)

if userDTO.is\_fired is not None:

conditions.append(User.is\_fired == userDTO.is\_fired)

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {conditions}")

if conditions:

query = query.where(sa.and\_(\*conditions))

query = query.order\_by(User.full\_name)

return db.session.execute(query).scalars().all()

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»РµР№')

def get\_all\_paginated(page: int, userDTO: UserDTO):

try:

query = sa.select(User)

conditions = []

if userDTO.role is not None:

conditions.append(User.role == userDTO.role)

if userDTO.is\_fired is not None:

conditions.append(User.is\_fired == userDTO.is\_fired)

if userDTO.full\_name is not None and len(userDTO.full\_name) == 0:

userDTO.full\_name = None

if userDTO.username is not None and len(userDTO.username) == 0:

userDTO.username = None

if userDTO.full\_name is not None:

conditions.append(sa.func.lower(User.full\_name).like(

f'%{userDTO.full\_name.lower()}%'))

if userDTO.username is not None:

conditions.append(sa.func.lower(User.username).like(

f'%{userDTO.username.lower()}%'))

current\_app.logger.info(f"СѓСЃР»РѕРІРёСЏ {conditions}")

if conditions:

query = query.where(sa.and\_(\*conditions))

query = query.order\_by(User.full\_name)

return db.paginate(query, page=page, per\_page=current\_app.config['USERS\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»РµР№')

def get\_by\_id(id: int):

try:

res = db.session.get(User, id)

if res is None:

raise ValueError(f'РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ СЃ id = {id} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ РїРѕ id')

def get\_by\_username(username: str):

try:

res = db.session.execute(sa.select(User).where(

User.username == username)).scalar\_one\_or\_none()

if res is None:

raise ValueError(f'РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ СЃ РёРјРµРЅРµРј {username} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception(

f'РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ РїРѕ РёРјРµРЅРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def create(userDTO: UserDTO):

try:

user = User(

username=userDTO.username,

full\_name=userDTO.full\_name,

password\_hash=generate\_password\_hash(userDTO.password),

role=userDTO.role,

)

db.session.add(user)

db.session.commit()

except sa.exc.IntegrityError as e:

db.session.rollback()

current\_app.logger.error(e)

raise ValueError(

f'РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ СЃ РёРјРµРЅРµРј {userDTO.username} СѓР¶Рµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception(f'РћС€РёР±РєР° РїСЂРё СЃРѕР·РґР°РЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def check\_password(username: str, password: str):

try:

user = get\_by\_username(username)

res = check\_password\_hash(user.password\_hash, password)

if not res:

raise WrongPasswordError('РќРµРІРµСЂРЅС‹Р№ РїР°СЂРѕР»СЊ')

if user.is\_fired:

raise FiredError('РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ Р·Р°Р±Р»РѕРєРёСЂРѕРІР°РЅ')

return user

except ValueError as e:

current\_app.logger.error(e)

raise

except WrongPasswordError as e:

current\_app.logger.error(e)

raise

except FiredError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїСЂРѕРІРµСЂРєРµ РїР°СЂРѕР»СЏ')

def update(userDTO: UserDTO):

try:

record = get\_by\_id(userDTO.id)

record.username = userDTO.username

record.full\_name = userDTO.full\_name

if userDTO.password:

record.password\_hash = generate\_password\_hash(userDTO.password)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РѕР±РЅРѕРІР»РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def delete(id: int) -> bool:

try:

user = get\_by\_id(id)

db.session.delete(user)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def fire(id: int):

try:

record = get\_by\_id(id)

record.is\_fired = not record.is\_fired

if record.is\_fired:

close\_user\_sessions(id)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРІРѕР»СЊРЅРµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def add\_to\_department(user\_id: int, department\_id: int):

try:

user = get\_by\_id(user\_id)

if user.role != TEACHER:

raise ValueError('РќРµР»СЊР·СЏ РЅР°Р·РЅР°С‡РёС‚СЊ СЃРѕС‚СЂСѓРґРЅРёРєР° РЅР° РєР°С„РµРґСЂСѓ')

department = department\_service.get\_by\_id(department\_id)

user.departments.add(department)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РґРѕР±Р°РІР»РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ РЅР° РєР°С„РµРґСЂСѓ')

def remove\_from\_department(user\_id: int, department\_id: int):

try:

current\_app.logger.info('РЈРґР°Р»РµРЅРёРµ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ СЃ РєР°С„РµРґСЂС‹')

user = get\_by\_id(user\_id)

if user.role != TEACHER:

raise ValueError('РќРµР»СЊР·СЏ СѓР±СЂР°С‚СЊ СЃРѕС‚СЂСѓРґРЅРёРєР° СЃ РєР°С„РµРґСЂС‹')

department = department\_service.get\_by\_id(department\_id)

user.departments.remove(department)

db.session.commit()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё СѓРґР°Р»РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ СЃ РєР°С„РµРґСЂС‹')

def get\_departments(userDTO: UserDTO):

try:

user = None

if userDTO.id is not None:

user = get\_by\_id(userDTO.id)

else:

user = get\_by\_username(userDTO.name)

return db.session.scalars(user.departments.select()).all()

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєР°С„РµРґСЂ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def get\_notifications(page: int, only\_new, user: User):

try:

query = user.notifications.select()

if only\_new:

query = query.where(Notification.has\_read == False)

query = query.order\_by(sa.asc(Notification.has\_read),

sa.desc(Notification.time\_sent))

return db.paginate(query, page=page, per\_page=current\_app.config['NOTIFICATIONS\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё СЃРѕРѕР±С‰РµРЅРёР№ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def get\_courses(page: int, approved: bool, user: User):

try:

query = user.courses.select().where(TeacherCourse.date\_approved.is\_(None))

if approved:

query = query.where(TeacherCourse.date\_approved is not None)

query = query.order\_by(Course.name)

return db.paginate(query, page=page, per\_page=current\_app.config['COURSES\_PER\_PAGE'], error\_out=False)

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РєСѓСЂСЃРѕРІ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

def get\_users\_sessions(user\_id):

try:

pattern = f'session:{user\_id}:\*'

r = current\_app.config['SESSION\_REDIS']

keys = r.keys(pattern)

return keys

except Exception as e:

current\_app.logger.info(e)

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def close\_user\_session(session\_id):

try:

r = current\_app.config['SESSION\_REDIS']

r.delete(session\_id)

return True

except Exception as e:

current\_app.logger.info(e)

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def close\_user\_sessions(user\_id):

try:

r = current\_app.config['SESSION\_REDIS']

keys = get\_users\_sessions(user\_id)

if keys:

r.delete(\*keys)

return len(keys)

except Exception as e:

current\_app.logger.info(e)

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def get\_token(user\_id, expires\_in=3600):

try:

now = datetime.now()

user = get\_by\_id(user\_id)

if user.token and user.token\_expiration > now + timedelta(seconds=60):

return user.token

user.token = secrets.token\_hex(16)

user.token\_expiration = now + timedelta(seconds=expires\_in)

db.session.commit()

return user.token

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def revoke\_token(user\_id):

try:

user = get\_by\_id(user\_id)

user.token\_expiration = datetime.now() - timedelta(seconds=1)

db.session.commit()

return user

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

db.session.rollback()

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def check\_token(token):

try:

user = db.session.scalar(sa.select(User).where(User.token == token))

if user is None or user.token\_expiration < datetime.now():

raise ValueError(

f'РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ СЃ С‚РѕРєРµРЅРѕРј = {token} РЅРµ СЃСѓС‰РµСЃС‚РІСѓРµС‚')

return user

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ РїРѕ С‚РѕРєРµРЅСѓ')

def get\_latest\_course(user\_id):

try:

user = get\_by\_id(user\_id)

res = sa.select(TeacherCourse).where(TeacherCourse.teacher\_id == user\_id)\

.order\_by(TeacherCourse.date\_completion.desc())

res = db.session.scalars(res).first()

if res is None or res.date\_completion is None:

return None

return res

except ValueError as e:

current\_app.logger.error(e)

raise

except Exception as e:

current\_app.logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё РїРѕР»СѓС‡РµРЅРёРё РїРѕСЃР»РµРґРЅРµРіРѕ РєСѓСЂСЃР° РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

from app.services import course\_service, course\_type\_service, department\_service

from app.services import faculty\_service, notification\_service, sertificate\_service, user\_service

from app.services import reports

from datetime import datetime

import sqlalchemy as sa

from app.models.course import Course

from app.models.teacher\_course import TeacherCourse

from app.models.user import User

from app import db

from app.services import course\_type\_service

class CourseTypeReport:

def \_\_init\_\_(self, course\_type\_id, date\_from: datetime = None, date\_to: datetime = None):

self.date\_from = f"РћС‚: {date\_from.isoformat()}"

self.date\_to = f"Р”Рѕ: {date\_to.isoformat()}"

self.course\_type = course\_type\_service.get\_by\_id(course\_type\_id)

self.filter\_item\_name = self.course\_type.name

self.table\_header = ['Р¤РРћ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ', 'РРјСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ',

'РќР°Р·РІР°РЅРёРµ РєСѓСЂСЃР°', 'Р”Р°С‚Р° РїРѕРґС‚РІРµСЂР¶РґРµРЅРёСЏ', 'в„– РџРѕРґС‚РІРµСЂР¶РґР°СЋС‰РµРіРѕ РґРѕРєСѓРјРµРЅС‚Р°']

query = (sa.select(

User.full\_name,

User.username,

Course.name,

TeacherCourse.date\_completion,

TeacherCourse.confirming\_document,

)

.join(User.courses)

.outerjoin(User.departments)

.join(TeacherCourse.course)

.where(sa.and\_(

Course.course\_type\_id == course\_type\_id,

TeacherCourse.date\_completion.is\_not(None),

sa.or\_(

date\_from is None,

TeacherCourse.date\_completion >= date\_from

),

sa.or\_(

date\_to is None,

TeacherCourse.date\_completion <= date\_to,

),

))

.distinct()

.order\_by(sa.desc(TeacherCourse.date\_completion)))

self.rows = db.session.execute(query).all()

self.result = len(self.rows)

subquery = (sa.select(

User.full\_name,

User.username,

Course.name,

TeacherCourse.date\_completion,

TeacherCourse.confirming\_document,

)

.join(User.courses)

.outerjoin(User.departments)

.join(TeacherCourse.course)

.where(sa.and\_(

Course.course\_type\_id == course\_type\_id,

TeacherCourse.date\_completion.is\_not(None)

)

).distinct().subquery())

query = sa.select(sa.func.count()).select\_from(subquery)

delim = db.session.scalar(query)

if delim == 0:

self.percent = ''

else:

self.percent = f"{len(self.rows) / db.session.scalar(query) \* 100:.2f}%"

self.percent\_target = 'РџСЂРѕС†РµРЅС‚ РїСЂРѕР№РґРµРЅРЅС‹С… РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏРјРё РєСѓСЂСЃРѕРІ Р·Р°РґР°РЅРЅРѕРіРѕ С‚РёРїР° Р·Р° РґР°РЅРЅС‹Р№ РїРµСЂРёРѕРґ ' \

'РїРѕ РѕС‚РЅРѕС€РµРЅРёСЋ РєРѕ РІСЃРµРј РїСЂРѕР№РґРµРЅРЅС‹Рј РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏРјРё РєСѓСЂСЃР°Рј Р·Р°РґР°РЅРЅРѕРіРѕ С‚РёРїР°'

from datetime import date

import sqlalchemy as sa

from app.dto.user\_dto import UserDTO

from app.models.course import Course

from app.services import user\_service

from app import db

from app.services.reports.pdf\_draw\_row import PdfDrawRow

class DirectionCreator(PdfDrawRow):

def \_\_init\_\_(self, teacher, courses, date\_from, date\_to, place, admin):

super().\_\_init\_\_()

if date\_from > date\_to:

raise ValueError('Р”Р°С‚Р° РЅР°С‡Р°Р»Р° РЅРµ РґРѕР»Р¶РЅР° Р±С‹С‚СЊ РїРѕР·Р¶Рµ РґР°С‚С‹ РєРѕРЅС†Р°')

self.teacher = teacher

self.courses = db.session.scalars(

sa.select(Course).where(Course.id.in\_(courses))).all()

self.period = f"РЎ {date\_from} РїРѕ {date\_to}"

self.departments = user\_service.get\_departments(UserDTO(teacher.id))

self.admin = admin

self.place = place

def create\_latest\_course\_info(self, latest\_course):

return f"РўРёРї РєСѓСЂСЃР°: {latest\_course.course.course\_type.name}\n" +\

f"РќР°Р·РІР°РЅРёРµ РєСѓСЂСЃР°: {latest\_course.course.name}\n" +\

f"Р”Р°С‚Р° РїСЂРѕС…РѕР¶РґРµРЅРёСЏ РєСѓСЂСЃР°: {latest\_course.date\_completion.isoformat()}\n" +\

f"в„– РїРѕРґС‚РІРµСЂР¶РґР°СЋС‰РµРіРѕ РґРѕРєСѓРјРµРЅС‚Р°: {latest\_course.confirming\_document}\n"

def create\_courses\_info(self):

res = ""

for course in self.courses:

res += f"{course.course\_type.name}: {course.name}\n"

return res

def create\_departments\_info(self):

res = ""

for department in self.departments:

res += f"{department.faculty.name}: {department.name}\n"

return res

def create\_table(self):

self.add\_page()

self.set\_font("DejaVu", size=16)

self.set\_font(style="B")

self.multi\_cell(

text=f"РЎРїСЂР°РІРєР° РїСЂРµРґСЃС‚Р°РІР»РµРЅРёРµ", w=120, border=0, align='C')

self.ln()

self.set\_font("DejaVu", size=10)

self.set\_font(style="B")

self.draw\_row(['РРЅС„РѕСЂРјР°С†РёСЏ Рѕ СЃР»СѓС€Р°С‚РµР»Рµ'], [180])

self.set\_font(style="")

self.draw\_row(['Р¤РРћ', self.teacher.full\_name], [90, 90])

self.draw\_row(['РљР°С„РµРґСЂС‹', self.create\_departments\_info()],

[90, 90])

latest\_course = user\_service.get\_latest\_course(self.teacher.id)

if latest\_course is None:

self.draw\_row(['РџРѕСЃР»РµРґРЅРµРµ РїРѕРІС‹С€РµРЅРёРµ РєРІР°Р»РёС„РёРєР°С†РёРё', 'РќРµ РїСЂРѕС…РѕРґРёР»РѕСЃСЊ'],

[90, 90])

else:

self.draw\_row(['РџРѕСЃР»РµРґРЅРµРµ РїРѕРІС‹С€РµРЅРёРµ РєРІР°Р»РёС„РёРєР°С†РёРё', self.create\_latest\_course\_info(latest\_course)],

[90, 90])

self.set\_font(style="B")

self.draw\_row(['РРЅС„РѕСЂРјР°С†РёСЏ Рѕ РїСЂРѕРіСЂР°РјРјРµ РїРѕРІС‹С€РµРЅРёСЏ РєРІР°Р»РёС„РёРєР°С†РёРё'], [180])

self.set\_font(style="")

self.draw\_row(['РќР°РёРјРµРЅРѕРІР°РЅРёСЏ РїСЂРѕРіСЂР°РјРј',

self.create\_courses\_info()], [90, 90])

self.draw\_row(['РњРµСЃС‚Рѕ РїРѕРІС‹С€РµРЅРёСЏ РєРІР°Р»РёС„РёРєР°С†РёРё', self.place],

[90, 90])

self.draw\_row(['РџРµСЂРёРѕРґ РїСЂРѕС…РѕР¶РґРµРЅРёСЏ РїРѕРІС‹С€РµРЅРёСЏ РєРІР°Р»РёС„РёРєР°С†РёРё', self.period],

[90, 90])

self.draw\_row(['РџРѕРґРїРёСЃСЊ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ', ''],

[90, 90])

self.draw\_row(['РџРѕРґРїРёСЃСЊ СЃРѕС‚СЂСѓРґРЅРёРєР°', ''],

[90, 90])

self.draw\_row(['Р”Р°С‚Р°', date.today().isoformat()], [90, 90])

self.draw\_row(['Р¤РРћ СЃРѕС‚СЂСѓРґРЅРёРєР°', self.admin.full\_name], [90, 90])

from datetime import date

from flask import current\_app

from app.services.reports.pdf\_draw\_row import PdfDrawRow

class ReportCreator(PdfDrawRow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def \_to\_str(self, text):

if not text:

return 'РћС‚СЃСѓС‚СЃС‚РІСѓРµС‚'

elif isinstance(text, date):

return text.isoformat()

else:

return str(text)

def create\_table(self, report, col\_widths, report\_name):

self.add\_page()

self.set\_font("DejaVu", size=16)

self.set\_font(style="B")

self.multi\_cell(

text=f"{report\_name}: {report.filter\_item\_name}", w=120, border=0, align='C')

self.ln()

self.set\_font("DejaVu", size=10)

self.set\_font(style="B")

self.multi\_cell(text=report.date\_from, w=120, border=0, align='C')

self.ln()

self.multi\_cell(text=report.date\_to, w=120, border=0, align='C')

self.ln()

self.draw\_row(report.table\_header, col\_widths)

self.set\_font(style="")

for row in report.rows:

self.draw\_row(row, col\_widths)

self.set\_font(style="B")

lst = ['' for i in range(len(col\_widths))]

lst[0] = 'РС‚РѕРіРѕРІРѕРµ РєРѕР»РёС‡РµСЃС‚РІРѕ РїСЂРѕР№РґРµРЅРЅС‹С… РєСѓСЂСЃРѕРІ'

lst[-1] = report.result

self.draw\_row(lst, col\_widths)

lst[0] = report.percent\_target

lst[-1] = report.percent

self.draw\_row(lst, col\_widths)

from flask import current\_app

import sqlalchemy as sa

from app.models.department import Department

from app.models.faculty import Faculty

from app.models.teacher\_course import TeacherCourse

from app.models.user import User

from app.services import faculty\_service

from app import db

class DepartmentsReport:

def \_\_init\_\_(self, faculty\_id, date\_from, date\_to):

self.faculty = faculty\_service.get\_by\_id(faculty\_id)

self.date\_from = f"РћС‚: {date\_from}"

self.date\_to = f"Р”Рѕ: {date\_to}"

self.table\_header = ['РќР°Р·РІР°РЅРёРµ РєР°С„РµРґСЂС‹', 'РљРѕР»РёС‡РµСЃС‚РІРѕ РїСЂРѕР№РґРµРЅРЅС‹С… РєСѓСЂСЃРѕРІ', '% РѕС‚РЅРѕСЃРёС‚РµР»СЊРЅРѕ РІСЃРµС… РїСЂРѕР№РґРµРЅРЅС‹С… РєСѓСЂСЃРѕРІ С„Р°РєСѓР»СЊС‚РµС‚Р° Р·Р° РїРµСЂРёРѕРґ']

self.filter\_item\_name = self.faculty.name

current\_app.logger.info(f"Р”Р°С‚Р° РЅР°С‡Р°Р»Р°: {date\_from}")

current\_app.logger.info(f"Р”Р°С‚Р° РєРѕРЅС†Р°: {date\_to}")

subquery1 = sa.select(TeacherCourse.teacher\_id, TeacherCourse.course\_id)\

.join(User)\

.join(User.departments)\

.where(

sa.and\_(

Department.faculty\_id == faculty\_id,

TeacherCourse.date\_completion >= date\_from,

TeacherCourse.date\_completion <= date\_to

)

)\

.distinct()\

.subquery()

query = sa.select(sa.func.count().label('faculty\_total'))\

.select\_from(subquery1)

cnt = db.session.scalar(query)

current\_app.logger.info(f"РљРѕР»РёС‡РµСЃС‚РІРѕ: {cnt}")

count\_expr = sa.func.count().label('on\_department')

query = sa.select(Department.name,

count\_expr,

sa.func.round(count\_expr \* 100.0 / cnt, 2).label('on\_department\_percent')

).select\_from(Faculty)\

.join(Faculty.departments)\

.join(Department.teachers)\

.join(User.courses)\

.where(sa.and\_(TeacherCourse.date\_completion >= date\_from,

TeacherCourse.date\_completion <= date\_to,

Department.faculty\_id == faculty\_id,

))\

.group\_by(Department.id, Department.name)\

.order\_by(Department.name)

self.result = cnt

self.percent\_target = 'РџСЂРѕС†РµРЅС‚С‹'

self.percent = f"{100}%"

self.rows = db.session.execute(query).all()

self.rows = list(map(lambda x: [x[0], x[1], f"{x[2]}%"], self.rows))

current\_app.logger.info(f"Р РµР·СѓР»СЊС‚Р°С‚: {self.rows}")

from datetime import datetime

import sqlalchemy as sa

from app.models.course import Course

from app.models.department import Department

from app.models.teacher\_course import TeacherCourse

from app.models.user import User

from app.services import faculty\_service

from app import db

class FacultyReport:

def \_\_init\_\_(self, faculty\_id, date\_from: datetime = None, date\_to: datetime = None):

self.date\_from = f"РћС‚: {date\_from}"

self.date\_to = f"Р”Рѕ: {date\_to}"

self.faculty = faculty\_service.get\_by\_id(faculty\_id)

self.filter\_item\_name = self.faculty.name

self.table\_header = ['Р¤РРћ РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏ', 'РРјСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ',

'РќР°Р·РІР°РЅРёРµ РєСѓСЂСЃР°', 'Р”Р°С‚Р° РїРѕРґС‚РІРµСЂР¶РґРµРЅРёСЏ', 'в„– РџРѕРґС‚РІРµСЂР¶РґР°СЋС‰РµРіРѕ РґРѕРєСѓРјРµРЅС‚Р°']

query = (sa.select(

User.full\_name,

User.username,

Course.name,

TeacherCourse.date\_completion,

TeacherCourse.confirming\_document,

)

.join(User.courses)

.join(User.departments)

.join(TeacherCourse.course)

.where(sa.and\_(

Department.faculty\_id == faculty\_id,

TeacherCourse.date\_completion.is\_not(None),

sa.or\_(

date\_from is None,

TeacherCourse.date\_completion >= date\_from

),

sa.or\_(

date\_to is None,

TeacherCourse.date\_completion <= date\_to,

),

))

.distinct()

.order\_by(sa.desc(TeacherCourse.date\_completion)))

self.rows = db.session.execute(query).all()

self.result = len(self.rows)

subquery = (sa.select(

User.full\_name,

User.username,

Course.name,

TeacherCourse.date\_completion,

TeacherCourse.confirming\_document,

)

.join(User.courses)

.join(User.departments)

.join(TeacherCourse.course)

.where(sa.and\_(

Department.faculty\_id == faculty\_id,

TeacherCourse.date\_completion.is\_not(None)

)

)

.distinct().subquery())

query = sa.select(sa.func.count()).select\_from(subquery)

delim = db.session.scalar(query)

if delim == 0:

self.percent = ''

else:

self.percent = f"{len(self.rows) / db.session.scalar(query) \* 100:.2f}%"

self.percent\_target = 'РџСЂРѕС†РµРЅС‚ РїСЂРѕР№РґРµРЅРЅС‹С… РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏРјРё Р·Р°РґР°РЅРЅРѕРіРѕ С„Р°РєСѓР»СЊС‚РµС‚Р° РєСѓСЂСЃРѕРІ Р·Р° РґР°РЅРЅС‹Р№ РїРµСЂРёРѕРґ ' \

'РїРѕ РѕС‚РЅРѕС€РµРЅРёСЋ РєРѕ РІСЃРµРј РїСЂРѕР№РґРµРЅРЅС‹Рј РїСЂРµРїРѕРґР°РІР°С‚РµР»СЏРјРё Р·Р°РґР°РЅРЅРѕРіРѕ С„Р°РєСѓР»СЊС‚РµС‚Р° РєСѓСЂСЃР°Рј'

from datetime import date

from flask import current\_app

from fpdf import FPDF

class PdfDrawRow(FPDF):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.line\_height = self.font\_size + 2

self.add\_font(

'DejaVu', '', 'app/static/fonts/DejaVuSans.ttf', uni=True)

self.add\_font(

'DejaVu', 'B', 'app/static/fonts/DejaVuSans-Bold.ttf', uni=True)

def \_to\_str(self, text):

if text is None:

return 'РћС‚СЃСѓС‚СЃС‚РІСѓРµС‚'

elif isinstance(text, date):

return text.isoformat()

else:

return str(text)

def draw\_row(self, row, col\_widths):

cell\_heights = []

for i, text in enumerate(row):

text = self.\_to\_str(text)

nb\_lines = self.multi\_cell(

col\_widths[i],

self.line\_height,

text,

border=0,

split\_only=True,

)

cell\_heights.append(self.line\_height \* len(nb\_lines))

max\_height = max(cell\_heights)

if self.get\_y() + max\_height > self.page\_break\_trigger:

self.add\_page()

x\_start = self.get\_x()

y\_start = self.get\_y()

for i, text in enumerate(row):

text = self.\_to\_str(text)

x = self.get\_x()

self.rect(x, y\_start, col\_widths[i], max\_height)

self.multi\_cell(

col\_widths[i],

self.line\_height,

text,

border=0,

align='L'

)

self.set\_xy(x + col\_widths[i], y\_start)

self.set\_y(y\_start + max\_height)

from app.services.reports import course\_type\_report, create\_report, faculty\_report

import logging

from logging.config import fileConfig

from flask import current\_app

from alembic import context

# this is the Alembic Config object, which provides

# access to the values within the .ini file in use.

config = context.config

# Interpret the config file for Python logging.

# This line sets up loggers basically.

fileConfig(config.config\_file\_name)

logger = logging.getLogger('alembic.env')

def get\_engine():

try:

# this works with Flask-SQLAlchemy<3 and Alchemical

return current\_app.extensions['migrate'].db.get\_engine()

except (TypeError, AttributeError):

# this works with Flask-SQLAlchemy>=3

return current\_app.extensions['migrate'].db.engine

def get\_engine\_url():

try:

return get\_engine().url.render\_as\_string(hide\_password=False).replace(

'%', '%%')

except AttributeError:

return str(get\_engine().url).replace('%', '%%')

# add your model's MetaData object here

# for 'autogenerate' support

# from myapp import mymodel

# target\_metadata = mymodel.Base.metadata

config.set\_main\_option('sqlalchemy.url', get\_engine\_url())

target\_db = current\_app.extensions['migrate'].db

# other values from the config, defined by the needs of env.py,

# can be acquired:

# my\_important\_option = config.get\_main\_option("my\_important\_option")

# ... etc.

def get\_metadata():

if hasattr(target\_db, 'metadatas'):

return target\_db.metadatas[None]

return target\_db.metadata

def run\_migrations\_offline():

"""Run migrations in 'offline' mode.

This configures the context with just a URL

and not an Engine, though an Engine is acceptable

here as well. By skipping the Engine creation

we don't even need a DBAPI to be available.

Calls to context.execute() here emit the given string to the

script output.

"""

url = config.get\_main\_option("sqlalchemy.url")

context.configure(

url=url, target\_metadata=get\_metadata(), literal\_binds=True

)

with context.begin\_transaction():

context.run\_migrations()

def run\_migrations\_online():

"""Run migrations in 'online' mode.

In this scenario we need to create an Engine

and associate a connection with the context.

"""

# this callback is used to prevent an auto-migration from being generated

# when there are no changes to the schema

# reference: http://alembic.zzzcomputing.com/en/latest/cookbook.html

def process\_revision\_directives(context, revision, directives):

if getattr(config.cmd\_opts, 'autogenerate', False):

script = directives[0]

if script.upgrade\_ops.is\_empty():

directives[:] = []

logger.info('No changes in schema detected.')

conf\_args = current\_app.extensions['migrate'].configure\_args

if conf\_args.get("process\_revision\_directives") is None:

conf\_args["process\_revision\_directives"] = process\_revision\_directives

connectable = get\_engine()

with connectable.connect() as connection:

context.configure(

connection=connection,

target\_metadata=get\_metadata(),

\*\*conf\_args

)

with context.begin\_transaction():

context.run\_migrations()

if context.is\_offline\_mode():

run\_migrations\_offline()

else:

run\_migrations\_online()

"""empty message

Revision ID: 1e7bc8a5d33a

Revises: fcfc0e623b7a

Create Date: 2025-05-31 07:49:13.742862

"""

from alembic import op

import sqlalchemy as sa

from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.

revision = '1e7bc8a5d33a'

down\_revision = 'fcfc0e623b7a'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('date\_completion', sa.DateTime(), nullable=True))

batch\_op.drop\_column('date\_approved')

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('date\_approved', postgresql.TIMESTAMP(), autoincrement=False, nullable=True))

batch\_op.drop\_column('date\_completion')

# ### end Alembic commands ###

"""empty message

Revision ID: 28cc14c7bebd

Revises:

Create Date: 2025-05-13 12:15:49.369144

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = '28cc14c7bebd'

down\_revision = None

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

op.create\_table('course\_type',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('name', sa.String(length=128), nullable=False),

sa.Column('deadline', sa.DateTime(), nullable=True),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('course\_type', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_course\_type\_deadline'), ['deadline'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_course\_type\_name'), ['name'], unique=True)

op.create\_table('faculty',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('name', sa.String(length=128), nullable=False),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('faculty', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_faculty\_name'), ['name'], unique=True)

op.create\_table('user',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('username', sa.String(length=64), nullable=False),

sa.Column('full\_name', sa.String(length=128), nullable=False),

sa.Column('password\_hash', sa.String(length=256), nullable=True),

sa.Column('role', sa.Integer(), nullable=False),

sa.Column('is\_fired', sa.Boolean(), nullable=False),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('user', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_user\_is\_fired'), ['is\_fired'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_user\_role'), ['role'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_user\_username'), ['username'], unique=True)

op.create\_table('course',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('name', sa.String(length=128), nullable=False),

sa.Column('course\_type\_id', sa.Integer(), nullable=False),

sa.Column('is\_included', sa.Boolean(), nullable=False),

sa.ForeignKeyConstraint(['course\_type\_id'], ['course\_type.id'], ondelete='CASCADE'),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('course', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_course\_course\_type\_id'), ['course\_type\_id'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_course\_is\_included'), ['is\_included'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_course\_name'), ['name'], unique=True)

op.create\_table('department',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('name', sa.String(length=128), nullable=False),

sa.Column('faculty\_id', sa.Integer(), nullable=False),

sa.ForeignKeyConstraint(['faculty\_id'], ['faculty.id'], ondelete='CASCADE'),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('department', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_department\_faculty\_id'), ['faculty\_id'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_department\_name'), ['name'], unique=True)

op.create\_table('notification',

sa.Column('id', sa.Integer(), nullable=False),

sa.Column('message', sa.String(length=256), nullable=False),

sa.Column('user\_id', sa.Integer(), nullable=False),

sa.Column('has\_read', sa.Boolean(), nullable=False),

sa.Column('time\_sent', sa.DateTime(), nullable=False),

sa.ForeignKeyConstraint(['user\_id'], ['user.id'], ondelete='CASCADE'),

sa.PrimaryKeyConstraint('id')

)

with op.batch\_alter\_table('notification', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_notification\_message'), ['message'], unique=True)

batch\_op.create\_index(batch\_op.f('ix\_notification\_time\_sent'), ['time\_sent'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_notification\_user\_id'), ['user\_id'], unique=False)

op.create\_table('teacher\_course',

sa.Column('teacher\_id', sa.Integer(), nullable=False),

sa.Column('course\_id', sa.Integer(), nullable=False),

sa.Column('sertificate\_path', sa.String(length=260), nullable=True),

sa.Column('date\_approved', sa.DateTime(), nullable=True),

sa.ForeignKeyConstraint(['course\_id'], ['course.id'], ondelete='CASCADE'),

sa.ForeignKeyConstraint(['teacher\_id'], ['user.id'], ondelete='CASCADE'),

sa.PrimaryKeyConstraint('teacher\_id', 'course\_id'),

sa.UniqueConstraint('sertificate\_path')

)

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.create\_index(batch\_op.f('ix\_teacher\_course\_course\_id'), ['course\_id'], unique=False)

batch\_op.create\_index(batch\_op.f('ix\_teacher\_course\_teacher\_id'), ['teacher\_id'], unique=False)

op.create\_table('teachers\_departments',

sa.Column('teacher\_id', sa.Integer(), nullable=False),

sa.Column('department\_id', sa.Integer(), nullable=False),

sa.ForeignKeyConstraint(['department\_id'], ['department.id'], ),

sa.ForeignKeyConstraint(['teacher\_id'], ['user.id'], ),

sa.PrimaryKeyConstraint('teacher\_id', 'department\_id')

)

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

op.drop\_table('teachers\_departments')

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_teacher\_course\_teacher\_id'))

batch\_op.drop\_index(batch\_op.f('ix\_teacher\_course\_course\_id'))

op.drop\_table('teacher\_course')

with op.batch\_alter\_table('notification', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_notification\_user\_id'))

batch\_op.drop\_index(batch\_op.f('ix\_notification\_time\_sent'))

batch\_op.drop\_index(batch\_op.f('ix\_notification\_message'))

op.drop\_table('notification')

with op.batch\_alter\_table('department', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_department\_name'))

batch\_op.drop\_index(batch\_op.f('ix\_department\_faculty\_id'))

op.drop\_table('department')

with op.batch\_alter\_table('course', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_course\_name'))

batch\_op.drop\_index(batch\_op.f('ix\_course\_is\_included'))

batch\_op.drop\_index(batch\_op.f('ix\_course\_course\_type\_id'))

op.drop\_table('course')

with op.batch\_alter\_table('user', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_user\_username'))

batch\_op.drop\_index(batch\_op.f('ix\_user\_role'))

batch\_op.drop\_index(batch\_op.f('ix\_user\_is\_fired'))

op.drop\_table('user')

with op.batch\_alter\_table('faculty', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_faculty\_name'))

op.drop\_table('faculty')

with op.batch\_alter\_table('course\_type', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_course\_type\_name'))

batch\_op.drop\_index(batch\_op.f('ix\_course\_type\_deadline'))

op.drop\_table('course\_type')

# ### end Alembic commands ###

"""empty message

Revision ID: 31225a820f7e

Revises: 39873caa3d62

Create Date: 2025-05-31 08:55:55.235052

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = '31225a820f7e'

down\_revision = '39873caa3d62'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teachers\_departments', schema=None) as batch\_op:

batch\_op.drop\_constraint('teachers\_departments\_department\_id\_fkey', type\_='foreignkey')

batch\_op.create\_foreign\_key(None, 'department', ['department\_id'], ['id'], ondelete='CASCADE')

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teachers\_departments', schema=None) as batch\_op:

batch\_op.drop\_constraint(None, type\_='foreignkey')

batch\_op.create\_foreign\_key('teachers\_departments\_department\_id\_fkey', 'department', ['department\_id'], ['id'])

# ### end Alembic commands ###

"""empty message

Revision ID: 39873caa3d62

Revises: 9c4785df694e

Create Date: 2025-05-31 08:51:58.865412

"""

from alembic import op

import sqlalchemy as sa

from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.

revision = '39873caa3d62'

down\_revision = '9c4785df694e'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('course\_type', schema=None) as batch\_op:

batch\_op.drop\_index('ix\_course\_type\_deadline')

batch\_op.drop\_column('deadline')

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('course\_type', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('deadline', postgresql.TIMESTAMP(), autoincrement=False, nullable=True))

batch\_op.create\_index('ix\_course\_type\_deadline', ['deadline'], unique=False)

# ### end Alembic commands ###

"""empty message

Revision ID: 4f9e17df04bf

Revises: b88b1151e45c

Create Date: 2025-06-01 09:03:40.707591

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = '4f9e17df04bf'

down\_revision = 'b88b1151e45c'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('confirming\_document', sa.String(length=64), nullable=True))

batch\_op.create\_index(batch\_op.f('ix\_teacher\_course\_confirming\_document'), ['confirming\_document'], unique=False)

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_teacher\_course\_confirming\_document'))

batch\_op.drop\_column('confirming\_document')

# ### end Alembic commands ###

"""empty message

Revision ID: 9c4785df694e

Revises: 1e7bc8a5d33a

Create Date: 2025-05-31 08:49:30.581784

"""

from alembic import op

import sqlalchemy as sa

from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.

revision = '9c4785df694e'

down\_revision = '1e7bc8a5d33a'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.alter\_column('date\_completion',

existing\_type=postgresql.TIMESTAMP(),

type\_=sa.Date(),

existing\_nullable=True)

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.alter\_column('date\_completion',

existing\_type=sa.Date(),

type\_=postgresql.TIMESTAMP(),

existing\_nullable=True)

# ### end Alembic commands ###

"""empty message

Revision ID: b88b1151e45c

Revises: ea70de0222c6

Create Date: 2025-05-31 21:28:04.480408

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = 'b88b1151e45c'

down\_revision = 'ea70de0222c6'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('document\_number', sa.String(length=260), nullable=True))

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('teacher\_course', schema=None) as batch\_op:

batch\_op.drop\_column('document\_number')

# ### end Alembic commands ###

"""empty message

Revision ID: ea70de0222c6

Revises: 31225a820f7e

Create Date: 2025-05-31 21:16:54.637959

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = 'ea70de0222c6'

down\_revision = '31225a820f7e'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('course', schema=None) as batch\_op:

batch\_op.drop\_index('ix\_course\_is\_included')

batch\_op.drop\_column('is\_included')

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('course', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('is\_included', sa.BOOLEAN(), autoincrement=False, nullable=False))

batch\_op.create\_index('ix\_course\_is\_included', ['is\_included'], unique=False)

# ### end Alembic commands ###

"""empty message

Revision ID: f20cce325fca

Revises: 4f9e17df04bf

Create Date: 2025-06-02 08:59:04.137160

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = 'f20cce325fca'

down\_revision = '4f9e17df04bf'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('notification', schema=None) as batch\_op:

batch\_op.drop\_index('ix\_notification\_message')

batch\_op.create\_index(batch\_op.f('ix\_notification\_message'), ['message'], unique=False)

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('notification', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_notification\_message'))

batch\_op.create\_index('ix\_notification\_message', ['message'], unique=True)

# ### end Alembic commands ###

"""empty message

Revision ID: fcfc0e623b7a

Revises: 28cc14c7bebd

Create Date: 2025-05-19 20:32:58.378001

"""

from alembic import op

import sqlalchemy as sa

# revision identifiers, used by Alembic.

revision = 'fcfc0e623b7a'

down\_revision = '28cc14c7bebd'

branch\_labels = None

depends\_on = None

def upgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('user', schema=None) as batch\_op:

batch\_op.add\_column(sa.Column('token', sa.String(length=32), nullable=True))

batch\_op.add\_column(sa.Column('token\_expiration', sa.DateTime(), nullable=True))

batch\_op.create\_index(batch\_op.f('ix\_user\_token'), ['token'], unique=True)

# ### end Alembic commands ###

def downgrade():

# ### commands auto generated by Alembic - please adjust! ###

with op.batch\_alter\_table('user', schema=None) as batch\_op:

batch\_op.drop\_index(batch\_op.f('ix\_user\_token'))

batch\_op.drop\_column('token\_expiration')

batch\_op.drop\_column('token')

# ### end Alembic commands ###

from config import Config

class TestConfig(Config):

TESTING = True

SQLALCHEMY\_DATABASE\_URI = 'sqlite://'

import os

import unittest

from tests.test\_config import TestConfig

from app import create\_app, db

from app.services import course\_service, course\_type\_service

from app.dto.course\_dto import CourseDTO

from app.dto.course\_type\_dto import CourseTypeDTO

os.environ['DATABASE\_URL'] = 'sqlite://'

class CourseServiceCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

course\_type\_service.create(

CourseTypeDTO(name='Programming'))

self.course\_type = course\_type\_service.get\_by\_name('Programming')

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def create\_courses(self):

test\_courses = [

CourseDTO(name='Python Basics',

course\_type\_id=self.course\_type.id),

CourseDTO(name='Web Development',

course\_type\_id=self.course\_type.id),

CourseDTO(name='Data Science', course\_type\_id=self.course\_type.id)

]

for course in test\_courses:

course\_service.create(course)

def test\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РєСѓСЂСЃР°')

courseDTO = CourseDTO(

name='Algorithms', course\_type\_id=self.course\_type.id)

course\_service.create(courseDTO)

created\_course = course\_service.get\_by\_name(courseDTO.name)

self.assertTrue(

created\_course is not None and

courseDTO.name == created\_course.name and

courseDTO.course\_type\_id == created\_course.course\_type\_id

)

def test\_update(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕР±РЅРѕРІР»РµРЅРёСЏ РєСѓСЂСЃР°')

self.create\_courses()

course = course\_service.get\_by\_name('Web Development')

new\_name = 'Advanced Web Development'

courseDTO = CourseDTO(id=course.id, name=new\_name,

course\_type\_id=self.course\_type.id)

course\_service.update(courseDTO)

updated\_course = course\_service.get\_by\_id(course.id)

self.assertTrue(

updated\_course is not None and updated\_course.name == new\_name)

def test\_delete(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ РєСѓСЂСЃР°')

self.create\_courses()

course = course\_service.get\_by\_name('Data Science')

course\_service.delete(course.id)

with self.assertRaises(ValueError):

course\_service.get\_by\_id(course.id)

def test\_get\_all\_paginated(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РїР°РіРёРЅР°С†РёРё РєСѓСЂСЃРѕРІ')

self.create\_courses()

page = course\_service.get\_all\_paginated(page=1)

self.assertEqual(page.total, 3)

def test\_same\_name\_create(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РєСѓСЂСЃР° СЃ РѕРґРёРЅР°РєРѕРІС‹Рј РёРјРµРЅРµРј')

self.create\_courses()

with self.assertRaises(ValueError):

courseDTO = CourseDTO(name='Python Basics',

course\_type\_id=self.course\_type.id)

course\_service.create(courseDTO)

def test\_get\_by\_id\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµРіРѕ РєСѓСЂСЃР° РїРѕ ID')

with self.assertRaises(ValueError):

course\_service.get\_by\_id(999)

def test\_get\_by\_name\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµРіРѕ РєСѓСЂСЃР° РїРѕ РёРјРµРЅРё')

with self.assertRaises(ValueError):

course\_service.get\_by\_name('Non\_Existent\_Course')

def test\_create\_without\_course\_type(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РєСѓСЂСЃР° Р±РµР· С‚РёРїР°')

with self.assertRaises(ValueError):

courseDTO = CourseDTO(name='Invalid Course')

course\_service.create(courseDTO)

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

unittest.main(verbosity=2)

import os

import unittest

from tests.test\_config import TestConfig

from app import create\_app, db

from app.services import course\_type\_service

from app.dto.course\_type\_dto import CourseTypeDTO

os.environ['DATABASE\_URL'] = 'sqlite://'

class CourseTypeServiceCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def create\_course\_types(self):

test\_course\_types = [

CourseTypeDTO(name='Bebra'),

CourseTypeDTO(name='aboba'),

CourseTypeDTO(name='boba')

]

for course\_type in test\_course\_types:

course\_type\_service.create(course\_type)

def test\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° СЃРѕР·РґР°РЅРёСЏ С‚РёРїР° РєСѓСЂСЃРѕРІ')

course\_typeDTO = CourseTypeDTO(name='Test Course Type')

course\_type\_service.create(course\_typeDTO)

created = course\_type\_service.get\_by\_name(course\_typeDTO.name)

self.assertEqual(created.name, 'Test Course Type')

def test\_update(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕР±РЅРѕРІР»РµРЅРёСЏ С‚РёРїРѕРІ РєСѓСЂСЃРѕРІ')

course\_typeDTO = CourseTypeDTO(name='Test Course Type')

course\_type\_service.create(course\_typeDTO)

created = course\_type\_service.get\_by\_name(course\_typeDTO.name)

course\_type\_service.update(

CourseTypeDTO(id=created.id, name='New name'))

course\_type = course\_type\_service.get\_by\_id(created.id)

self.assertEqual(course\_type.name, 'New name')

def test\_delete(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ С‚РёРїР° РєСѓСЂСЃРѕРІ')

course\_typeDTO = CourseTypeDTO(name='Test Course Type')

course\_type\_service.create(course\_typeDTO)

created = course\_type\_service.get\_by\_name(course\_typeDTO.name)

course\_type\_service.delete(created.id)

with self.assertRaises(ValueError):

course\_type = course\_type\_service.get\_by\_id(created.id)

def test\_get\_all(self):

self.app.logger.info('Р—Р°РїСѓСЃРє РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… С‚РёРїРѕРІ РєСѓСЂСЃРѕРІ')

self.create\_course\_types()

self.assertTrue(len(course\_type\_service.get\_all()) == 3)

def test\_get\_all\_empty(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… С‚РёРїРѕРІ РєСѓСЂСЃРѕРІ РёР· РїСѓСЃС‚РѕР№ Р‘Р”')

self.assertEqual(len(course\_type\_service.get\_all()), 0)

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

unittest.main(verbosity=2)

import os

import unittest

from tests.test\_config import TestConfig

from app.dto.department\_dto import DepartmentDTO

from app.dto.faculty\_dto import FacultyDTO

from app.services import department\_service, faculty\_service

from app import create\_app, db

os.environ['DATABASE\_URL'] = 'sqlite://'

class DepartmentServiceCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

faculty\_service.create(FacultyDTO(name='Test Faculty'))

self.faculty = faculty\_service.get\_by\_id(1)

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def create\_departments(self):

test\_departments = [

DepartmentDTO(name='IS', faculty\_id=self.faculty.id),

DepartmentDTO(name='CT', faculty\_id=self.faculty.id),

DepartmentDTO(name='AB', faculty\_id=self.faculty.id),

]

for department in test\_departments:

department\_service.create(department)

def test\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° СЃРѕР·РґР°РЅРёСЏ РєР°С„РµРґСЂС‹')

departmentDTO = DepartmentDTO(

name='Test Department', faculty\_id=self.faculty.id)

department\_service.create(departmentDTO)

created = department\_service.get\_by\_name(departmentDTO.name)

self.assertEqual(created.name, 'Test Department')

self.assertEqual(created.faculty\_id, self.faculty.id)

def test\_update(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕР±РЅРѕРІР»РµРЅРёСЏ РєР°С„РµРґСЂС‹')

faculty\_service.create(FacultyDTO(name='FIST'))

new\_faculty = faculty\_service.get\_by\_name('FIST')

new\_name = 'SF'

department\_service.create(DepartmentDTO(

name='test', faculty\_id=self.faculty.id))

department = department\_service.get\_by\_name('test')

departmentDTO = DepartmentDTO(

id=department.id, name=new\_name, faculty\_id=new\_faculty.id)

department\_service.update(departmentDTO)

department = department\_service.get\_by\_id(department.id)

self.assertTrue(department is not None and department.name ==

new\_name and department.faculty.id == new\_faculty.id)

def test\_delete(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ РєР°С„РµРґСЂС‹')

department\_service.create(DepartmentDTO(

name='test', faculty\_id=self.faculty.id))

department = department\_service.get\_by\_name('test')

department\_service.delete(department.id)

with self.assertRaises(ValueError):

department = department\_service.get\_by\_id(department.id)

def test\_department\_faculty\_relationship(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° СЃРІСЏР·Рё РєР°С„РµРґСЂС‹ СЃ С„Р°РєСѓР»СЊС‚РµС‚РѕРј')

department\_service.create(DepartmentDTO(

name='CS', faculty\_id=self.faculty.id))

department = department\_service.get\_by\_name('CS')

self.assertEqual(department.faculty.name, 'Test Faculty')

self.assertIn(department, faculty\_service.get\_departments(

FacultyDTO(department.faculty.id)))

def test\_cascade\_on\_faculty\_delete(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РєР°СЃРєР°РґРЅРѕРіРѕ СѓРґР°Р»РµРЅРёСЏ РїСЂРё СѓРґР°Р»РµРЅРёРё С„Р°РєСѓР»СЊС‚РµС‚Р°')

department\_service.create(DepartmentDTO(

name='Math', faculty\_id=self.faculty.id))

faculty\_service.delete(self.faculty.id)

with self.assertRaises(ValueError):

deleted = department\_service.get\_by\_name('Math')

def test\_same\_name\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° СѓРЅРёРєР°Р»СЊРЅРѕСЃС‚Рё РёРјРµРЅРё РєР°С„РµРґСЂС‹')

department\_service.create(DepartmentDTO(

name='Chemistry', faculty\_id=self.faculty.id))

with self.assertRaises(ValueError):

department\_service.create(DepartmentDTO(

name='Chemistry', faculty\_id=self.faculty.id))

def test\_get\_by\_id\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµР№ РєР°С„РµРґСЂС‹ РїРѕ ID')

with self.assertRaises(ValueError):

department = department\_service.get\_by\_id(999)

def test\_get\_by\_name\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµР№ РєР°С„РµРґСЂС‹ РїРѕ РёРјРµРЅРё')

with self.assertRaises(ValueError):

department = department\_service.get\_by\_name('NON\_EXISTENT')

def test\_get\_all(self):

self.app.logger.info('Р—Р°РїСѓСЃРє РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… РєР°С„РµРґСЂ')

self.create\_departments()

self.assertTrue(len(department\_service.get\_all()) == 3)

def test\_get\_all\_empty(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… РєР°С„РµРґСЂ РёР· РїСѓСЃС‚РѕР№ Р‘Р”')

self.assertEqual(len(department\_service.get\_all()), 0)

def test\_get\_all\_paginated(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїР°РіРёРЅР°С†РёРё РєР°С„РµРґСЂ')

self.create\_departments()

page1 = department\_service.get\_all\_paginated(page=1)

self.assertEqual(len(page1.items), 3)

self.assertEqual(page1.page, 1)

self.assertEqual(

page1.per\_page, self.app.config['DEPARTMENTS\_PER\_PAGE'])

self.assertFalse(page1.has\_prev)

self.assertFalse(page1.has\_next)

original\_per\_page = self.app.config['DEPARTMENTS\_PER\_PAGE']

self.app.config['DEPARTMENTS\_PER\_PAGE'] = 2

page1 = department\_service.get\_all\_paginated(page=1)

self.assertEqual(len(page1.items), 2)

self.assertTrue(page1.has\_next)

self.assertFalse(page1.has\_prev)

page2 = department\_service.get\_all\_paginated(page=2)

self.app.logger.info(page2.items)

self.assertEqual(len(page2.items), 1)

self.assertFalse(page2.has\_next)

self.assertTrue(page2.has\_prev)

self.app.config['DEPARTMENTS\_PER\_PAGE'] = original\_per\_page

def test\_get\_all\_paginated\_empty(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїР°РіРёРЅР°С†РёРё СЃ РїСѓСЃС‚С‹Рј СЃРїРёСЃРєРѕРј РєР°С„РµРґСЂ')

page = department\_service.get\_all\_paginated(page=1)

self.assertEqual(len(page.items), 0)

self.assertEqual(page.total, 0)

self.assertFalse(page.has\_prev)

self.assertFalse(page.has\_next)

def test\_get\_all\_paginated\_empty(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїР°РіРёРЅР°С†РёРё СЃ РїСѓСЃС‚С‹Рј СЃРїРёСЃРєРѕРј РєР°С„РµРґСЂ')

page = department\_service.get\_all\_paginated(page=1)

self.assertEqual(len(page.items), 0)

self.assertEqual(page.total, 0)

self.assertFalse(page.has\_prev)

self.assertFalse(page.has\_next)

def test\_get\_all\_paginated\_with\_faculty\_filter(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїР°РіРёРЅР°С†РёРё СЃ С„РёР»СЊС‚СЂРѕРј РїРѕ С„Р°РєСѓР»СЊС‚РµС‚Сѓ')

faculty\_service.create(FacultyDTO(name='Another Faculty'))

faculty2 = faculty\_service.get\_by\_name('Another Faculty')

department\_service.create(DepartmentDTO(

name='Dep1', faculty\_id=faculty2.id))

department\_service.create(DepartmentDTO(

name='Dep2', faculty\_id=faculty2.id))

original\_per\_page = self.app.config['DEPARTMENTS\_PER\_PAGE']

self.app.config['DEPARTMENTS\_PER\_PAGE'] = 1

page1 = department\_service.get\_all\_paginated(

page=1, faculties=[faculty2.id])

self.assertEqual(len(page1.items), 1)

self.assertEqual(page1.items[0].name, 'Dep1')

self.assertTrue(page1.has\_next)

page2 = department\_service.get\_all\_paginated(

page=2, faculties=[faculty2.id])

self.assertEqual(len(page2.items), 1)

self.assertEqual(page2.items[0].name, 'Dep2')

self.assertFalse(page2.has\_next)

self.app.config['DEPARTMENTS\_PER\_PAGE'] = original\_per\_page

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

unittest.main(verbosity=2)

import os

import unittest

from tests.test\_config import TestConfig

from app.dto.faculty\_dto import FacultyDTO

from app.services import faculty\_service

from app import create\_app, db

os.environ['DATABASE\_URL'] = 'sqlite://'

class FacultyServiceCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def create\_faculties(self):

test\_faculties = [

FacultyDTO(name='FIST'),

FacultyDTO(name='EF'),

FacultyDTO(name='RTF')

]

for faculty in test\_faculties:

faculty\_service.create(faculty)

def test\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ С„Р°РєСѓР»СЊС‚РµС‚Р°')

facultyDTO = FacultyDTO(name='GF')

faculty\_service.create(facultyDTO)

created\_faculty = faculty\_service.get\_by\_name(facultyDTO.name)

self.assertTrue(

created\_faculty is not None and facultyDTO.name == created\_faculty.name)

def test\_update(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕР±РЅРѕРІР»РµРЅРёСЏ С„Р°РєСѓР»СЊС‚РµС‚РѕРІ')

self.create\_faculties()

faculty = faculty\_service.get\_by\_name('EF')

new\_name = 'SF'

facultyDTO = FacultyDTO(id=faculty.id, name=new\_name)

faculty\_service.update(facultyDTO)

faculty = faculty\_service.get\_by\_id(faculty.id)

self.assertTrue(faculty is not None and faculty.name == new\_name)

def test\_delete(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ С„Р°РєСѓР»СЊС‚РµС‚Р°')

self.create\_faculties()

faculty = faculty\_service.get\_by\_id(1)

faculty\_service.delete(faculty.id)

with self.assertRaises(ValueError):

faculty = faculty\_service.get\_by\_id(faculty.id)

def test\_get\_all(self):

self.app.logger.info('Р—Р°РїСѓСЃРє РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… С„Р°РєСѓР»СЊС‚РµС‚РѕРІ')

self.create\_faculties()

self.assertTrue(len(faculty\_service.get\_all()) == 3)

def test\_same\_name\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє РїСЂРѕРІРµСЂРєРё РЅР° РѕРґРёРЅР°РєРѕРІС‹Рµ РёРјРµРЅР°')

self.create\_faculties()

with self.assertRaises(ValueError):

facultyDTO = FacultyDTO(name='FIST')

faculty\_service.create(facultyDTO)

def test\_get\_by\_id\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµРіРѕ С„Р°РєСѓР»СЊС‚РµС‚Р° РїРѕ ID')

with self.assertRaises(ValueError):

faculty = faculty\_service.get\_by\_id(999)

def test\_get\_by\_name\_nonexistent(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РЅРµСЃСѓС‰РµСЃС‚РІСѓСЋС‰РµРіРѕ С„Р°РєСѓР»СЊС‚РµС‚Р° РїРѕ РёРјРµРЅРё')

with self.assertRaises(ValueError):

faculty = faculty\_service.get\_by\_name('NON\_EXISTENT')

def test\_get\_all\_empty(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ РІСЃРµС… С„Р°РєСѓР»СЊС‚РµС‚РѕРІ РёР· РїСѓСЃС‚РѕР№ Р‘Р”')

self.assertEqual(len(faculty\_service.get\_all()), 0)

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

unittest.main(verbosity=2)

import os

import unittest

from datetime import datetime

from tests.test\_config import TestConfig

from app.dto.notification\_dto import NotificationDTO

from app.services import user\_service, notification\_service

from app.dto.user\_dto import UserDTO

from app.models import user

from app import create\_app, db

os.environ['DATABASE\_URL'] = 'sqlite://'

class FacultyServiceCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

user\_service.create(userDTO=UserDTO(

username='Test User', full\_name='Tester tester test', password='test', role=user.TEACHER))

self.user = user\_service.get\_by\_id(1)

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def create\_notifications(self):

test\_notifications = [

NotificationDTO(message='message1', user\_id=self.user.id),

NotificationDTO(message='message2', user\_id=self.user.id),

NotificationDTO(message='message3', user\_id=self.user.id)

]

for notification in test\_notifications:

notification\_service.send\_message(notification)

def test\_send(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕС‚РїСЂР°РІРєРё СѓРІРµРґРѕРјР»РµРЅРёСЏ')

notificationDTO = NotificationDTO(message='test', user\_id=self.user.id)

notification\_service.send\_message(notificationDTO)

created\_notification = notification\_service.get\_by\_id(1)

res = user\_service.get\_notifications(1, True, self.user).items

self.app.logger.info(res)

self.assertTrue(

len(res) == 1 and res[0].message == notificationDTO.message)

self.assertTrue(

created\_notification is not None and created\_notification.message == notificationDTO.message)

self.assertTrue(created\_notification.time\_sent.date()

== datetime.now().date())

def test\_notification\_count(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РєРѕР»РёС‡РµСЃС‚РІР° СѓРІРµРґРѕРјР»РµРЅРёР№')

self.create\_notifications()

self.assertEqual(

notification\_service.get\_user\_notifications\_count(self.user.id), 3)

notification\_service.read\_message(1)

self.assertEqual(

notification\_service.get\_user\_notifications\_count(self.user.id), 2)

def test\_read(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ С‡С‚РµРЅРёСЏ СѓРІРµРґРѕРјР»РµРЅРёСЏ')

self.create\_notifications()

notification\_service.read\_message(1)

created\_notification = notification\_service.get\_by\_id(1)

self.assertTrue(created\_notification.has\_read == True)

self.assertTrue(len(user\_service.get\_notifications(

1, True, user=self.user).items) == 2)

self.assertTrue(len(user\_service.get\_notifications(

1, False, user=self.user).items) == 3)

def test\_delete(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ СѓРІРµРґРѕРјР»РµРЅРёСЏ')

self.create\_notifications()

notification = notification\_service.get\_by\_id(1)

notification\_service.delete(notification.id)

with self.assertRaises(ValueError):

notification = notification\_service.get\_by\_id(1)

self.assertTrue(len(user\_service.get\_notifications(

1, False, self.user).items) == 2)

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

unittest.main(verbosity=2)

import os

import unittest

from datetime import datetime

from tests.test\_config import TestConfig

from app import create\_app, db

from app.services import course\_service, course\_type\_service, sertificate\_service, user\_service

from app.models import user

from app.dto.user\_dto import UserDTO

from app.dto.course\_type\_dto import CourseTypeDTO

from app.dto.course\_dto import CourseDTO

os.environ['DATABASE\_URL'] = 'sqlite://'

class SertificateServiceTestCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

user\_service.create(UserDTO(

username='test\_user', password='test\_password', role=user.TEACHER, full\_name='Tester'))

self.user = user\_service.get\_by\_username('test\_user')

course\_type\_service.create(CourseTypeDTO(name='test\_course\_type'))

self.course\_type = course\_type\_service.get\_by\_name('test\_course\_type')

def tearDown(self):

db.session.remove()

db.drop\_all()

db.engine.dispose()

self.app\_context.pop()

def create\_courses(self):

test\_courses = [

CourseDTO(name='test\_course1', course\_type\_id=self.course\_type.id),

CourseDTO(name='test\_course2', course\_type\_id=self.course\_type.id),

CourseDTO(name='test\_course3', course\_type\_id=self.course\_type.id)

]

for course in test\_courses:

course\_service.create(course)

def test\_several\_users(self):

self.create\_courses()

user\_service.create(UserDTO(

username='test\_user2', password='test\_password', role=user.TEACHER, full\_name='Tester2'))

user2 = user\_service.get\_by\_username('test\_user2')

sertificate\_service.get(self.user.id, 1)

sertificate\_service.get(self.user.id, 2)

sertificate\_service.get(self.user.id, 3)

self.assertEqual(

len(sertificate\_service.get\_user\_courses(self.user.id, 1).items), 3)

self.assertEqual(

len(sertificate\_service.get\_user\_courses(user2.id, 1).items), 0)

def test\_update\_teacher\_course(self):

self.create\_courses()

teacher\_course = sertificate\_service.get(

user\_id=self.user.id, course\_id=1)

timing = datetime.now().date()

with self.assertRaises(ValueError):

sertificate\_service.update\_teacher\_course(self.user.id, 1, timing)

teacher\_course.sertificate\_path = 'aboba'

sertificate\_service.update\_teacher\_course(self.user.id, 1, timing)

teacher\_course = sertificate\_service.get(

user\_id=self.user.id, course\_id=1)

self.assertEqual(timing, teacher\_course.date\_completion)

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

unittest.main(verbosity=2)

import os

import unittest

from tests.test\_config import TestConfig

from app import create\_app, db

from app.dto.department\_dto import DepartmentDTO

from app.dto.faculty\_dto import FacultyDTO

from app.dto.notification\_dto import NotificationDTO

from app.dto.user\_dto import UserDTO

from app.exceptions.wrong\_password\_error import WrongPasswordError

from app.services import department\_service, faculty\_service, user\_service, notification\_service

from app.models.user import TEACHER, ADMIN

os.environ['DATABASE\_URL'] = 'sqlite://'

class UserServiceTestCase(unittest.TestCase):

def setUp(self):

self.app = create\_app(TestConfig)

self.app\_context = self.app.app\_context()

self.app\_context.push()

db.create\_all()

faculty\_service.create(FacultyDTO(name='test faculty'))

self.faculty = faculty\_service.get\_by\_name('test faculty')

department\_service.create(DepartmentDTO(

faculty\_id=self.faculty.id, name='test department'))

self.department = department\_service.get\_by\_name('test department')

self.user\_data = {

'username': 'testuser',

'full\_name': 'Test User',

'password': 'secure123',

'role': 'student'

}

def tearDown(self):

db.session.remove()

db.drop\_all()

self.app\_context.pop()

def test\_create(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

dto = UserDTO(\*\*self.user\_data)

user\_service.create(dto)

user = user\_service.get\_by\_username('testuser')

self.assertEqual(user.full\_name, 'Test User')

self.assertTrue(user\_service.check\_password\_hash(

user.password\_hash, 'secure123'))

def test\_create\_duplicate\_username(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ СЃ СЃСѓС‰РµСЃС‚РІСѓСЋС‰РёРј РёРјРµРЅРµРј')

user\_service.create(UserDTO(\*\*self.user\_data))

with self.assertRaises(ValueError):

user\_service.create(UserDTO(\*\*self.user\_data))

def test\_check\_password(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СЃРѕР·РґР°РЅРёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ СЃ СЃСѓС‰РµСЃС‚РІСѓСЋС‰РёРј РёРјРµРЅРµРј')

user\_service.create(UserDTO(\*\*self.user\_data))

with self.assertRaises(WrongPasswordError):

user\_service.check\_password('testuser', 'wrongpass')

def test\_update\_user(self):

self.app.logger.info(

'Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ РѕР±РЅРѕРІР»РµРЅРёСЏ РґР°РЅРЅС‹С… РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

user\_service.create(UserDTO(\*\*self.user\_data))

user = user\_service.get\_by\_username('testuser')

update\_dto = UserDTO(

id=user.id,

username='updateduser',

full\_name='Updated Name',

password='newpassword',

role='teacher'

)

user\_service.update(update\_dto)

updated = user\_service.get\_by\_id(user.id)

self.assertEqual(updated.username, 'updateduser')

self.assertTrue(user\_service.check\_password(

'updateduser', 'newpassword'))

def test\_delete\_user(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚РёСЂРѕРІР°РЅРёСЏ СѓРґР°Р»РµРЅРёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

user\_service.create(UserDTO(\*\*self.user\_data))

user = user\_service.get\_by\_username('testuser')

user\_service.delete(user.id)

with self.assertRaises(ValueError):

user\_service.get\_by\_id(user.id)

def test\_get\_notifications(self):

self.app.logger.info('Р—Р°РїСѓСЃРє С‚РµСЃС‚Р° РїРѕР»СѓС‡РµРЅРёСЏ СѓРІРµРґРѕРјР»РµРЅРёР№ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

user\_service.create(UserDTO(\*\*self.user\_data))

user = user\_service.get\_by\_username('testuser')

notification\_service.send\_message(

NotificationDTO(user\_id=user.id, message='Hello'))

notifications = user\_service.get\_notifications(1, False, user)

self.assertEqual(notifications.total, 1)

self.assertEqual(notifications.items[0].message, 'Hello')

def test\_add\_to\_department(self):

teacher\_data = {

'username': 'teacher1',

'full\_name': 'Teacher One',

'password': 'teacher123',

'role': TEACHER

}

user\_service.create(UserDTO(\*\*teacher\_data))

teacher = user\_service.get\_by\_username('teacher1')

user\_service.add\_to\_department(teacher.id, self.department.id)

teacher = user\_service.get\_by\_id(teacher.id)

self.assertIn(self.department, user\_service.get\_departments(

UserDTO(id=teacher.id)))

def test\_add\_to\_department\_non\_teacher(self):

admin\_data = {

'username': 'admin',

'full\_name': 'Admin One',

'password': 'admin123',

'role': ADMIN

}

user\_service.create(UserDTO(\*\*admin\_data))

admin = user\_service.get\_by\_username('admin')

with self.assertRaises(ValueError) as context:

user\_service.add\_to\_department(admin.id, self.department.id)

def test\_add\_to\_department\_invalid\_department(self):

teacher\_data = {

'username': 'teacher2',

'full\_name': 'Teacher Two',

'password': 'teacher123',

'role': TEACHER

}

teacher = user\_service.create(UserDTO(\*\*teacher\_data))

with self.assertRaises(Exception) as context:

user\_service.add\_to\_department(teacher.id, 999)

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

unittest.main(verbosity=2)

import os

from dotenv import load\_dotenv

load\_dotenv()

class Config:

API\_BASE\_URL=os.getenv('API\_BASE\_URL') or 'pososamba'

BOT\_TOKEN=os.getenv('BOT\_TOKEN') or 'pososamba'

import urllib3

import asyncio

import logging

from aiogram import Bot, Dispatcher

from config import Config

from app.handlers import router

urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)

bot = Bot(token=Config.BOT\_TOKEN)

dp = Dispatcher()

async def main():

dp.include\_router(router)

await dp.start\_polling(bot)

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

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

filename='app.log'

)

logging.getLogger(\_\_name\_\_)

asyncio.run(main())

from aiogram.utils.keyboard import InlineKeyboardBuilder

from aiogram.types import InlineKeyboardButton

async def inline\_notifications(data):

keyboard = InlineKeyboardBuilder()

for notification in data['items']:

keyboard.add(InlineKeyboardButton(

text=f"{notification.message} {'вњ…' if notification.has\_read else ''}",

callback\_data=f"notification:{notification.id}",

))

page = data['\_meta']['page']

total\_pages = data['\_meta']['total\_pages']

total\_items = data['\_meta']['total\_items']

if total\_items == 0:

return None

left = page-1 if page != 1 else total\_pages

right = page+1 if page != total\_pages else 1

left\_button = InlineKeyboardButton(

text="в†ђ", callback\_data=f"notifications:{left}")

page\_button = InlineKeyboardButton(

text=f"{data['\_meta']['page']}/{data['\_meta']['total\_pages']}", callback\_data="None")

right\_button = InlineKeyboardButton(

text="в†’", callback\_data=f"notifications:{right}")

keyboard.adjust(1)

keyboard.row(left\_button, page\_button, right\_button)

return keyboard.as\_markup()

async def inline\_notification(data):

keyboard = InlineKeyboardBuilder()

if not data.has\_read:

read\_button = InlineKeyboardButton(

text="РџСЂРѕС‡РёС‚Р°С‚СЊ", callback\_data=f"read\_notification:{data.id}")

keyboard.add(read\_button)

delete\_button = InlineKeyboardButton(

text="РЈРґР°Р»РёС‚СЊ", callback\_data=f"delete\_notification:{data.id}")

keyboard.add(delete\_button)

keyboard.adjust(1)

return keyboard.as\_markup()

async def inline\_courses(data):

keyboard = InlineKeyboardBuilder()

for course in data['items']:

keyboard.add(InlineKeyboardButton(

text=f"{course.course\_name} {'вњ…' if course.date\_completion else ''}",

callback\_data=f"course:{course.teacher\_id}:{course.course\_id}",

))

page = data['\_meta']['page']

total\_pages = data['\_meta']['total\_pages']

total\_items = data['\_meta']['total\_items']

if total\_items == 0:

return None

left = page-1 if page != 1 else total\_pages

right = page+1 if page != total\_pages else 1

left\_button = InlineKeyboardButton(

text="в†ђ", callback\_data=f"courses:{left}")

page\_button = InlineKeyboardButton(

text=f"{data['\_meta']['page']}/{data['\_meta']['total\_pages']}", callback\_data="None")

right\_button = InlineKeyboardButton(

text="в†’", callback\_data=f"courses:{right}")

keyboard.adjust(1)

keyboard.row(left\_button, page\_button, right\_button)

return keyboard.as\_markup()

async def inline\_course(data):

keyboard = InlineKeyboardBuilder()

if data.sertificate\_loaded:

download\_button = InlineKeyboardButton(

text="РЎРєР°С‡Р°С‚СЊ СЃРµСЂС‚РёС„РёРєР°С‚", callback\_data=f"download\_sertificate:{data.teacher\_id}:{data.course\_id}")

keyboard.add(download\_button)

return keyboard.as\_markup()

from functools import wraps

import logging

from aiogram.types import Message, CallbackQuery, FSInputFile

from aiogram import Router

from aiogram.filters import Command, CommandStart

from aiogram.fsm.state import StatesGroup, State

from aiogram.fsm.context import FSMContext

from app.builders import inline\_course, inline\_courses, inline\_notification, inline\_notifications

from app.exceptions.non\_authorized\_error import NonAuthorizedError

from app.keyboards import help\_kb

from app.services import auth\_service, notification\_service, teacher\_course\_service

router = Router()

user\_tokens = {}

logger = logging.getLogger(\_\_name\_\_)

class Login(StatesGroup):

username = State()

password = State()

def token\_check(f):

@wraps(f)

async def decorated\_function(\*args, \*\*kwargs):

try:

if args[0].from\_user.id not in user\_tokens:

raise NonAuthorizedError

return await f(\*args, \*\*kwargs)

except NonAuthorizedError as e:

logger.error(e)

return await args[0].answer('РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё: РІРІРµРґРёС‚Рµ РєРѕРјР°РЅРґСѓ /login, С‡С‚РѕР±С‹ Р°РІС‚РѕСЂРёР·РѕРІР°С‚СЊСЃСЏ')

except Exception as e:

logger.error(e)

return await args[0].answer(str(e))

return decorated\_function

@router.message(CommandStart())

async def cmd\_start(message: Message, state: FSMContext):

await state.clear()

await message.reply(f"РџСЂРёРІРµС‚\nРћС‚РїСЂР°РІСЊ РєРѕРјР°РЅРґСѓ /help, С‡С‚РѕР±С‹ РїРѕР»СѓС‡РёС‚СЊ СЃРїРёСЃРѕРє РєРѕРјР°РЅРґ\n",

reply\_markup=help\_kb)

@router.message(Command('help'))

async def help(message: Message, state: FSMContext):

await state.clear()

await state.set\_state(Login.username)

await message.answer('РЎРїРёСЃРѕРє РєРѕРјР°РЅРґ\n' + \

'/help - РїРѕРјРѕС‰СЊ\n' + \

'/notifications - СЃРїРёСЃРѕРє СѓРІРµРґРѕРјР»РµРЅРёР№\n' + \

'/courses - СЃРїРёСЃРѕРє РєСѓСЂСЃРѕРІ\n' + \

'/login - РІС…РѕРґ РІ Р°РєРєР°СѓРЅС‚\n' + \

'/logout - РІС‹С…РѕРґ РёР· Р°РєРєР°СѓРЅС‚Р°\n')

@router.message(Command('login'))

async def enter\_username(message: Message, state: FSMContext):

await state.clear()

await state.set\_state(Login.username)

await message.answer('Р’РІРµРґРёС‚Рµ РёРјСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ')

@router.message(Command('logout'))

async def logout(message: Message, state: FSMContext):

await state.clear()

user\_id = message.from\_user.id

if user\_id not in user\_tokens:

await message.answer('Р’С‹ РЅРµ Р°РІС‚РѕСЂРёР·РѕРІР°РЅС‹')

else:

del user\_tokens[message.from\_user.id]

await message.answer('Р’С‹ СѓСЃРїРµС€РЅРѕ РІС‹С€Р»Рё РёР· Р°РєРєР°СѓРЅС‚Р°')

@router.message(Command('notifications'))

@token\_check

async def notifications(message: Message, state: FSMContext):

await state.clear()

user\_id = message.from\_user.id

res = await get\_notifications(user\_id)

if res:

await message.reply(text='РЎРїРёСЃРѕРє РІР°С€РёС… СѓРІРµРґРѕРјР»РµРЅРёР№:', reply\_markup=res)

else:

await message.reply(text='РЈРІРµРґРѕРјР»РµРЅРёР№ РЅРµС‚', reply\_markup=None)

@router.message(Command('courses'))

@token\_check

async def courses(message: Message, state: FSMContext):

await state.clear()

user\_id = message.from\_user.id

res = await get\_courses(user\_id)

if res:

await message.reply(text='РЎРїРёСЃРѕРє РІР°С€РёС… РєСѓСЂСЃРѕРІ:', reply\_markup=res)

else:

await message.reply(text='РЈ РІР°СЃ РЅРµС‚ РєСѓСЂСЃРѕРІ', reply\_markup=None)

@router.message(Login.username)

async def enter\_username\_inter(message: Message, state: FSMContext):

await state.update\_data(username=message.text)

await state.set\_state(Login.password)

await message.answer('Р’РІРµРґРёС‚Рµ РїР°СЂРѕР»СЊ')

@router.message(Login.password)

async def enter\_password\_inter(message: Message, state: FSMContext):

await state.update\_data(password=message.text)

try:

data = await state.get\_data()

res = auth\_service.login(data['username'], data['password'])

user\_tokens[message.from\_user.id] = res

await state.update\_data(token=res)

await message.answer(f"Р’Р°С€ С‚РѕРєРµРЅ: {res['token']}")

except ValueError as e:

logger.error(e)

await message.answer(f"РћС€РёР±РєР° РїСЂРё Р°РІС‚РѕСЂРёР·Р°С†РёРё: {e}")

except Exception as e:

logger.error(e)

await message.answer(f"РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°")

await state.clear()

async def get\_notifications(user\_id, page=1):

notifications = notification\_service.get\_all\_paginated(

int(user\_tokens[user\_id]['id']),

user\_tokens[user\_id]['token'],

page=page

)

logger.info(notifications)

return await inline\_notifications(notifications)

@router.callback\_query(lambda c: c.data and c.data.startswith('notifications:'))

@token\_check

async def notifiactions\_callback(callback: CallbackQuery):

page = int(callback.data.split(':')[-1])

user\_id = callback.from\_user.id

await callback.answer()

res = await get\_notifications(user\_id, page)

if res:

await callback.message.edit\_text(text='РЎРїРёСЃРѕРє РІР°С€РёС… СѓРІРµРґРѕРјР»РµРЅРёР№:', reply\_markup=res)

else:

await callback.message.edit\_text(text='РЈРІРµРґРѕРјР»РµРЅРёР№ РЅРµС‚', reply\_markup=None)

@router.callback\_query(lambda c: c.data and c.data.startswith('notification:'))

@token\_check

async def notification\_callback(callback: CallbackQuery):

user\_id = callback.from\_user.id

notification = notification\_service.get\_by\_id(int(callback.data.split(':')[-1]),

user\_tokens[user\_id]['token'])

await callback.answer()

await callback.message.edit\_text(text=notification.\_\_repr\_\_() + " \nР’СЃРµ СѓРІРµРґРѕРјР»РµРЅРёСЏ /notifications",

reply\_markup=await inline\_notification(notification))

@router.callback\_query(lambda c: c.data and c.data.startswith('read\_notification:'))

@token\_check

async def notification\_read\_callback(callback: CallbackQuery):

user\_id = callback.from\_user.id

notification = notification\_service.read(int(callback.data.split(':')[-1]),

user\_tokens[user\_id]['token'])

await callback.answer()

await callback.message.edit\_text(text=notification.\_\_repr\_\_() + " \nР’СЃРµ СѓРІРµРґРѕРјР»РµРЅРёСЏ /notifications",

reply\_markup=await inline\_notification(notification))

@router.callback\_query(lambda c: c.data and c.data.startswith('delete\_notification:'))

@token\_check

async def notification\_delete\_callback(callback: CallbackQuery):

user\_id = callback.from\_user.id

notification\_service.delete\_notification(int(callback.data.split(':')[-1]),

user\_tokens[user\_id]['token'])

await callback.answer()

await callback.message.edit\_text(text='РЈРІРµРґРѕРјР»РµРЅРёРµ СѓСЃРїРµС€РЅРѕ СѓРґР°Р»РµРЅРѕ\nР’СЃРµ СѓРІРµРґРѕРјР»РµРЅРёСЏ /notifications',

reply\_markup=None)

async def get\_courses(user\_id, page=1):

courses = teacher\_course\_service.get\_all\_paginated(

int(user\_tokens[user\_id]['id']),

user\_tokens[user\_id]['token'],

page=page

)

logger.info(courses)

return await inline\_courses(courses)

@router.callback\_query(lambda c: c.data and c.data.startswith('courses:'))

@token\_check

async def courses\_callback(callback: CallbackQuery):

page = int(callback.data.split(':')[-1])

user\_id = callback.from\_user.id

await callback.answer()

res = await get\_courses(user\_id, page)

if res:

await callback.message.edit\_text(text='РЎРїРёСЃРѕРє РІР°С€РёС… РєСѓСЂСЃРѕРІ:', reply\_markup=res)

else:

await callback.message.edit\_text(text='РЈ РІР°СЃ РЅРµС‚ РєСѓСЂСЃРѕРІ', reply\_markup=None)

@router.callback\_query(lambda c: c.data and c.data.startswith('course:'))

@token\_check

async def course\_callback(callback: CallbackQuery):

user\_id = callback.from\_user.id

course = teacher\_course\_service.get\_by\_ids(int(callback.data.split(':')[-2]),

int(callback.data.split(

':')[-1]),

user\_tokens[user\_id]['token'])

await callback.answer()

await callback.message.edit\_text(text=course.\_\_repr\_\_() + " \nР’СЃРµ РєСѓСЂСЃС‹ /courses",

reply\_markup=await inline\_course(course))

@router.callback\_query(lambda c: c.data and c.data.startswith('download\_sertificate:'))

@token\_check

async def download\_sertificate\_callback(callback: CallbackQuery):

user\_id = int(callback.data.split(':')[-2])

course\_id = int(callback.data.split(':')[-1])

f = teacher\_course\_service.download\_teacher\_course(

user\_id, course\_id, user\_tokens[callback.from\_user.id]['token'])

await callback.answer()

await callback.message.answer\_document(

document=FSInputFile(f.name, filename=f'РЎРµСЂС‚РёС„РёРєР°С‚ {user\_id}:{course\_id}'),

caption='РЎРµСЂС‚РёС„РёРєР°С‚ СѓСЃРїРµС€РЅРѕ РѕС‚РїСЂР°РІР»РµРЅ'

)

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

help\_kb = ReplyKeyboardMarkup(

keyboard=[

[KeyboardButton(text='/help')],

[KeyboardButton(text='/notifications')],

[KeyboardButton(text='/courses')],

[KeyboardButton(text='/login')],

[KeyboardButton(text='/logout')]

],

resize\_keyboard=True,

input\_field\_placeholder='РќР°Р¶РјРёС‚Рµ РєРЅРѕРїРєСѓ, С‡С‚РѕР±С‹ РїРѕР»СѓС‡РёС‚СЊ СЃРїРёСЃРѕРє РєРѕРјР°РЅРґ',

)

class NonAuthorizedError(Exception):

pass

from dataclasses import dataclass

from datetime import datetime

from typing import Optional

@dataclass

class Notifification:

id: int

message: str

has\_read: bool

time\_sent: datetime

self\_url: Optional[str]

def \_\_init\_\_(self):

pass

@staticmethod

def from\_dict(data):

if data is None:

return None

notification = \_\_class\_\_()

notification.id = data['id']

notification.message = data['message']

notification.has\_read = data['has\_read']

notification.time\_sent = datetime.fromisoformat(data['time\_sent'])

notification.self\_url = data['\_links']['self']

return notification

def \_\_repr\_\_(self):

return f"РўРµРєСЃС‚ СЃРѕРѕР±С‰РµРЅРёСЏ:\n{self.message}\n" +\

f"РЎРѕРѕР±С‰РµРЅРёРµ РїСЂРѕС‡РёС‚Р°РЅРѕ: {'вњ…' if self.has\_read else 'вќЊ'}\n" +\

f"Р’СЂРµРјСЏ РѕС‚РїСЂР°РІРєРё: {self.time\_sent}\n"

from dataclasses import dataclass

from datetime import datetime

from typing import Optional

@dataclass

class TeacherCourse:

teacher\_id: int

course\_id: int

date\_completion: datetime

course\_name: str

confirming\_document: str

sertificate\_loaded: bool

self\_url: Optional[str]

def \_\_init\_\_(self):

pass

@staticmethod

def from\_dict(data):

if data is None:

return None

teacher\_course = \_\_class\_\_()

teacher\_course.teacher\_id = data['teacher\_id']

teacher\_course.course\_id = data['course\_id']

teacher\_course.course\_name = data['course\_name']

teacher\_course.date\_completion = (datetime.fromisoformat(

data['date\_completion']).date() if data['date\_completion'] else None)

teacher\_course.confirming\_document = data['confirming\_document']

teacher\_course.sertificate\_loaded = data['sertificate\_loaded']

teacher\_course.self\_url = data['\_links']['self']

return teacher\_course

def \_\_repr\_\_(self):

return f"РќР°Р·РІР°РЅРёРµ РєСѓСЂСЃР°:\n{self.course\_name}\n" +\

f"РЎРµСЂС‚РёС„РёРєР°С‚ Р·Р°РіСЂСѓР¶РµРЅ: {'вњ…' if self.sertificate\_loaded else 'вќЊ'}\n" +\

(f"Р”Р°С‚Р° РїСЂРѕС…РѕР¶РґРµРЅРёСЏ: {self.date\_completion.isoformat()}\n" if self.date\_completion else 'РљСѓСЂСЃ РїРѕРєР° РЅРµ РїСЂРѕР№РґРµРЅ\n') +\

f"в„– РџРѕРґС‚РІРµСЂР¶РґР°СЋС‰РµРіРѕ РґРѕРєСѓРјРµРЅС‚Р°: " + (f"{self.confirming\_document}" if self.confirming\_document else 'РќРµ РїСЂРѕСЃС‚Р°РІР»РµРЅ')

import logging

import requests

from config import Config

logger = logging.getLogger(\_\_name\_\_)

def login(username, password):

logger.info('РЇ РўРЈРў')

query = Config.API\_BASE\_URL + f"/tokens"

response = requests.post(query, auth=(username,password), verify=False)

try:

response.raise\_for\_status()

return response.json()

except requests.HTTPError as e:

if response.status\_code == 400:

raise ValueError("РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ РЅРµ РЅР°Р№РґРµРЅ")

elif response.status\_code == 401:

raise ValueError("РќРµРІРµСЂРЅС‹Р№ РїР°СЂРѕР»СЊ")

elif response.status\_code == 403:

raise ValueError("РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ Р·Р°Р±Р»РѕРєРёСЂРѕРІР°РЅ")

else:

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РќРµРёР·РІРµСЃС‚РЅР°СЏ РѕС€РёР±РєР°')

def from\_json\_collection(data: dict, cls):

res = []

for i in data['items']:

res.append(cls.from\_dict(i))

return {'items': res, '\_meta': data['\_meta']}

import logging

import requests

from app.exceptions.non\_authorized\_error import NonAuthorizedError

from config import Config

from app.models.notification import Notifification

from app.services.from\_json\_collection import from\_json\_collection

logger = logging.getLogger(\_\_name\_\_)

def get\_all\_paginated(user\_id: int, token: str, page: int = None) -> list[Notifification]:

if page is None:

page = 1

query = Config.API\_BASE\_URL + f"/users/{user\_id}/notifications?page={page}"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.get(query, headers=headers, verify=False)

logger.info(response)

try:

response.raise\_for\_status()

lst = from\_json\_collection(response.json(), Notifification)

logger.info(f"РџРѕР»СѓС‡РµРЅРЅС‹Рµ СѓРІРµРґРѕРјР»РµРЅРёСЏ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {lst['items']}")

return lst

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise ValueError("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

def get\_by\_id(id: int, token: str):

query = Config.API\_BASE\_URL + f"/notifications/{id}"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.get(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

notifification = Notifification.from\_dict(response.json())

logger.info(f"РџРѕР»СѓС‡РµРЅРЅРѕРµ СѓРІРµРґРѕРјР»РµРЅРёРµ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {notifification}")

return notifification

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise ValueError("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

def read(id: int, token: str):

query = Config.API\_BASE\_URL + f"/notifications/{id}/read"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.put(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

notifification = Notifification.from\_dict(response.json())

logger.info(f"РџСЂРѕС‡РёС‚Р°РЅРѕ СѓРІРµРґРѕРјР»РµРЅРёРµ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {notifification}")

return notifification

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise ValueError("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

def delete\_notification(id: int, token: str):

query = Config.API\_BASE\_URL + f"/notifications/{id}/delete"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.delete(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

logger.info(f"РЈРІРµРґРѕРјР»РµРЅРёРµ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {id} СѓРґР°Р»РµРЅРѕ")

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise ValueError("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

import logging

import requests

from app.exceptions.non\_authorized\_error import NonAuthorizedError

from config import Config

from app.models.teacher\_course import TeacherCourse

from app.services.from\_json\_collection import from\_json\_collection

logger = logging.getLogger(\_\_name\_\_)

def get\_all\_paginated(user\_id: int, token: str, page: int = None) -> list[TeacherCourse]:

if page is None:

page = 1

query = Config.API\_BASE\_URL + f"/users/{user\_id}/courses?page={page}"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.get(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

list = from\_json\_collection(response.json(), TeacherCourse)

logger.info(f"РџРѕР»СѓС‡РµРЅРЅС‹Рµ РєСѓСЂСЃС‹ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {list['items']}")

return list

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise Exception("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

def get\_by\_ids(user\_id: int, course\_id: int, token: str):

query = Config.API\_BASE\_URL + f"/users/{user\_id}/courses/{course\_id}"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.get(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

teaher\_course = TeacherCourse.from\_dict(response.json())

logger.info(f"РџРѕР»СѓС‡РµРЅРЅС‹Р№ РєСѓСЂСЃ РїРѕР»СЊР·РѕРІР°С‚РµР»СЏ {teaher\_course}")

return teaher\_course

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise Exception("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

def download\_teacher\_course(user\_id: int, course\_id: int, token: str):

query = Config.API\_BASE\_URL + f"/users/{user\_id}/courses/{course\_id}/download"

headers = {

"Authorization": f"Bearer {token}"

}

response = requests.get(query, headers=headers, verify=False)

try:

response.raise\_for\_status()

with open("temp.pdf", "wb") as f:

f.write(response.content)

return f

except requests.HTTPError as e:

logger.error(e)

if response.status\_code == 401:

raise NonAuthorizedError("РћС€РёР±РєР° Р°РІС‚РѕСЂРёР·Р°С†РёРё")

if response.status\_code == 404:

raise Exception("Р РµСЃСѓСЂСЃ РЅРµ РЅР°Р№РґРµРЅ")

raise Exception(f"РћС€РёР±РєР° api")

except Exception as e:

logger.error(e)

raise Exception('РћС€РёР±РєР° РїСЂРё Р·Р°РїСЂРѕСЃРµ Рє API')

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Threading.Tasks;

namespace Models

{

public class User

{

[JsonPropertyName("id")]

[DisplayName("Id")]

public int Id { get; set; }

[JsonPropertyName("username")]

[DisplayName("Имя пользователя")]

public string Username { get; set; }

[JsonPropertyName("full\_name")]

[DisplayName("ФИО")]

public string FullName { get; set; }

[JsonPropertyName("role")]

[DisplayName("Роль")]

public string Role { get; set; }

[JsonPropertyName("token\_available")]

[DisplayName("Токен активен")]

public bool TokenAvailable { get; set; }

public User(int id, string username, string fullName, string role, bool tokenAvailable)

{

Id = id;

Username = username;

FullName = fullName;

Role = role;

TokenAvailable = tokenAvailable;

}

}

}

using System.Net.Http;

using System.Net.Security;

using System.Security.Cryptography.X509Certificates;

public static class HttpClientFactory

{

public static HttpClient CreateUnsafeClient()

{

var handler = new HttpClientHandler

{

ServerCertificateCustomValidationCallback = IgnoreCertificateValidation

};

return new HttpClient(handler);

}

private static bool IgnoreCertificateValidation(HttpRequestMessage request, X509Certificate2 cert, X509Chain chain, SslPolicyErrors errors)

{

return true;

}

}

using Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

namespace Services.SupportClasses

{

public class UserPaginatedList

{

public List<User>? Users { get; set; }

public int? Page { get; set; }

public int? NextPage { get; set; }

public int? PrevPage { get; set; }

public UserPaginatedList(List<User> users, int page, int nextPage, int prevPage)

{

Users = users;

Page = page;

NextPage = nextPage;

PrevPage = prevPage;

}

public UserPaginatedList()

{

}

public static UserPaginatedList Deserealise(String json)

{

using JsonDocument doc = JsonDocument.Parse(json);

JsonElement root = doc.RootElement;

UserPaginatedList cur = new UserPaginatedList();

cur.Users = JsonSerializer.Deserialize<List<User>>(root.GetProperty("items"));

cur.Page = JsonSerializer.Deserialize<int>(root.GetProperty("\_meta").GetProperty("page").ToString()!);

int totalPages = JsonSerializer.Deserialize<int>(root.GetProperty("\_meta").GetProperty("total\_pages").ToString()!);

if (cur.Page != totalPages)

{

cur.NextPage = cur.Page + 1;

}

else

{

cur.NextPage = 1;

}

if (cur.Page != 1)

{

cur.PrevPage = cur.Page - 1;

}

else

{

cur.PrevPage = totalPages;

}

return cur;

}

}

}

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Text.Json;

namespace Services

{

public class AuthService

{

public string? Token { get; set; }

public async Task<string> Login(string username, string password)

{

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + "/tokens";

var authToken = Convert.ToBase64String(Encoding.ASCII.GetBytes($"{username}:{password}"));

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Basic", authToken);

var response = await client.PostAsync(request, new StringContent(""));

if (!response.IsSuccessStatusCode)

{

if(response.StatusCode == HttpStatusCode.BadRequest)

{

throw new Exception("Пользователь не найден");

}

if (response.StatusCode == HttpStatusCode.Unauthorized)

{

throw new Exception("Неверный пароль");

}

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Пользователь заблокирован");

}

throw new Exception("Ошибка авторизации");

}

string jsonString = await response.Content.ReadAsStringAsync();

using JsonDocument doc = JsonDocument.Parse(jsonString);

JsonElement root = doc.RootElement;

string token = root.GetProperty("token").GetString()!;

string role = root.GetProperty("role").GetString()!;

if(role != "ADMIN")

{

throw new Exception("Данный клиент предназначен для сотрудников");

}

Token = token;

return token;

}

catch (Exception e)

{

Console.WriteLine(e);

throw ;

}

}

public void Logout()

{

Token = null;

}

}

}

using Models;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Net.Http.Headers;

using System.Net;

using System.Text;

using System.Text.Json;

using System.Threading.Tasks;

using System.Reflection.Metadata.Ecma335;

using Services.SupportClasses;

namespace Services

{

public class UserService

{

public string? Token { get; set; }

public async Task<UserPaginatedList> GetUsers(int page)

{

if (Token == null)

{

throw new Exception("Необходимо получить токен для авторизации");

}

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + $"/users?page={page}";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);

var response = await client.GetAsync(request);

if (!response.IsSuccessStatusCode)

{

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Недостаточно прав");

}

if (response.StatusCode == HttpStatusCode.Unauthorized)

{

throw new Exception("Необходимо авторизоваться");

}

throw new Exception("Ошибка доступа к API");

}

string jsonString = await response.Content.ReadAsStringAsync();

return UserPaginatedList.Deserealise(jsonString);

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

public async Task<User> RevokeToken(int userId)

{

if (Token == null)

{

throw new Exception("Необходимо получить токен для авторизации");

}

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + $"/users/{userId}/revoke\_token";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);

var response = await client.PutAsync(request, new StringContent(""));

if (!response.IsSuccessStatusCode)

{

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Недостаточно прав");

}

throw new Exception("Ошибка доступа к API");

}

string jsonString = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<User>(jsonString)!;

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

public async Task<List<string>> GetUserSessions(int userId)

{

if (Token == null)

{

throw new Exception("Необходимо получить токен для авторизации");

}

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + $"/users/{userId}/sessions";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);

var response = await client.GetAsync(request);

if (!response.IsSuccessStatusCode)

{

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Недостаточно прав");

}

if (response.StatusCode == HttpStatusCode.Unauthorized)

{

throw new Exception("Необходимо авторизоваться");

}

throw new Exception("Ошибка доступа к API");

}

string jsonString = await response.Content.ReadAsStringAsync();

using JsonDocument doc = JsonDocument.Parse(jsonString);

JsonElement root = doc.RootElement;

return JsonSerializer.Deserialize<List<string>>(root.GetProperty("items"))!;

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

public async Task CloseSessions(int userId)

{

if (Token == null)

{

throw new Exception("Необходимо получить токен для авторизации");

}

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + $"/users/{userId}/sessions/close";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);

var response = await client.DeleteAsync(request);

if (!response.IsSuccessStatusCode)

{

if (response.StatusCode == HttpStatusCode.Unauthorized)

{

throw new Exception("Необходимо авторизоваться");

}

if (response.StatusCode == HttpStatusCode.NotFound)

{

throw new Exception("Не найдено");

}

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Недостаточно прав");

}

throw new Exception("Ошибка доступа к API");

}

string jsonString = await response.Content.ReadAsStringAsync();

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

public async Task CloseSession(int userId, string sessionId)

{

if (Token == null)

{

throw new Exception("Необходимо получить токен для авторизации");

}

using var client = HttpClientFactory.CreateUnsafeClient();

string apiUrl = ConfigurationManager.AppSettings["ApiUrl"]!;

try

{

string request = apiUrl + $"/users/{userId}/sessions/close/{sessionId}";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Token);

var response = await client.DeleteAsync(request);

if (!response.IsSuccessStatusCode)

{

if (response.StatusCode == HttpStatusCode.Unauthorized)

{

throw new Exception("Необходимо авторизоваться");

}

if (response.StatusCode == HttpStatusCode.NotFound)

{

throw new Exception("Не найдено");

}

if (response.StatusCode == HttpStatusCode.Forbidden)

{

throw new Exception("Недостаточно прав");

}

throw new Exception("Ошибка доступа к API");

}

string jsonString = await response.Content.ReadAsStringAsync();

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

}

}

}

namespace University

{

partial class LoginForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

usernameTextBox = new TextBox();

passwordTextBox = new MaskedTextBox();

loginGroupBox = new GroupBox();

loginButton = new Button();

passwordLabel = new Label();

usernameLabel = new Label();

button1 = new Button();

loginGroupBox.SuspendLayout();

SuspendLayout();

//

// usernameTextBox

//

usernameTextBox.Location = new Point(168, 22);

usernameTextBox.Name = "usernameTextBox";

usernameTextBox.Size = new Size(266, 23);

usernameTextBox.TabIndex = 0;

//

// passwordTextBox

//

passwordTextBox.Location = new Point(168, 51);

passwordTextBox.Name = "passwordTextBox";

passwordTextBox.PasswordChar = '\*';

passwordTextBox.Size = new Size(266, 23);

passwordTextBox.TabIndex = 1;

//

// loginGroupBox

//

loginGroupBox.Anchor = AnchorStyles.None;

loginGroupBox.Controls.Add(loginButton);

loginGroupBox.Controls.Add(passwordLabel);

loginGroupBox.Controls.Add(usernameLabel);

loginGroupBox.Controls.Add(usernameTextBox);

loginGroupBox.Controls.Add(passwordTextBox);

loginGroupBox.Location = new Point(45, 40);

loginGroupBox.Name = "loginGroupBox";

loginGroupBox.Size = new Size(510, 132);

loginGroupBox.TabIndex = 2;

loginGroupBox.TabStop = false;

loginGroupBox.Text = "Авторизация";

//

// loginButton

//

loginButton.Location = new Point(168, 80);

loginButton.Name = "loginButton";

loginButton.Size = new Size(75, 23);

loginButton.TabIndex = 4;

loginButton.Text = "Войти";

loginButton.UseVisualStyleBackColor = true;

loginButton.Click += loginButton\_Click;

//

// passwordLabel

//

passwordLabel.AutoSize = true;

passwordLabel.Location = new Point(113, 54);

passwordLabel.Name = "passwordLabel";

passwordLabel.Size = new Size(49, 15);

passwordLabel.TabIndex = 3;

passwordLabel.Text = "Пароль";

//

// usernameLabel

//

usernameLabel.AutoSize = true;

usernameLabel.Location = new Point(53, 25);

usernameLabel.Name = "usernameLabel";

usernameLabel.Size = new Size(109, 15);

usernameLabel.TabIndex = 2;

usernameLabel.Text = "Имя пользователя";

//

// button1

//

button1.Location = new Point(388, 31);

button1.Name = "button1";

button1.Size = new Size(8, 8);

button1.TabIndex = 3;

button1.Text = "button1";

button1.UseVisualStyleBackColor = true;

//

// LoginForm

//

AutoScaleDimensions = new SizeF(7F, 15F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(584, 211);

Controls.Add(button1);

Controls.Add(loginGroupBox);

FormBorderStyle = FormBorderStyle.FixedSingle;

Name = "LoginForm";

StartPosition = FormStartPosition.CenterScreen;

Text = "Вход";

loginGroupBox.ResumeLayout(false);

loginGroupBox.PerformLayout();

ResumeLayout(false);

}

#endregion

private TextBox usernameTextBox;

private MaskedTextBox passwordTextBox;

private GroupBox loginGroupBox;

private Label passwordLabel;

private Label usernameLabel;

private Button button1;

private Button loginButton;

}

}

using Services;

namespace University

{

public partial class LoginForm : Form

{

private readonly AuthService \_authService;

public string Token { get; private set; }

public string Username { get; private set; }

public LoginForm()

{

InitializeComponent();

\_authService = new AuthService();

}

private async void loginButton\_Click(object sender, EventArgs e)

{

try

{

var username = usernameTextBox.Text;

var password = passwordTextBox.Text;

var token = await \_authService.Login(username, password);

MessageBox.Show("Вы авторизованы");

Token = token;

Username = username;

DialogResult = DialogResult.OK;

Close();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

using Services;

using Services.SupportClasses;

using System.Net;

using System.Net.Security;

using System.Security.Cryptography.X509Certificates;

namespace University

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static async Task Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

var form = new LoginForm();

Application.Run(new UsersForm());

}

}

}

using Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace University

{

public partial class SessionsForm : Form

{

private UserService \_userService { get; set; }

private string \_token { get; set; }

private string \_currentUser { get; set; }

private int \_userId { get; set; }

public SessionsForm(UserService userService, int userId, string token, string currentUser)

{

InitializeComponent();

\_userService = userService;

\_token = token;

\_currentUser = currentUser;

\_userId = userId;

usernameTextBox.Text = \_currentUser;

}

private async Task LoadData()

{

try

{

var list = await \_userService.GetUserSessions(\_userId);

if (list != null)

{

DataTable table = new DataTable();

table.Columns.Add("Id сессии", typeof(string));

foreach (var str in list)

{

table.Rows.Add(str);

}

dataGridView.DataSource = table;

dataGridView.AllowUserToAddRows = false;

dataGridView.Columns["Id сессии"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK);

}

}

private async void SessionsForm\_Load(object sender, EventArgs e)

{

await LoadData();

}

private async void refreshButton\_Click(object sender, EventArgs e)

{

await LoadData();

}

private async void closeSessionsButton\_Click(object sender, EventArgs e)

{

try

{

await \_userService.CloseSessions(\_userId);

await LoadData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK);

}

}

private async void closeSessionButton\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count != 0)

{

try

{

var selectedRow = dataGridView.SelectedRows[0];

var value = selectedRow.Cells["Id сессии"].Value!.ToString();

await \_userService.CloseSession(\_userId, value!);

await LoadData();

}

catch (Exception ex)

{

MessageBox.Show("Неизвестая ошибка");

}

}

}

}

}

namespace University

{

partial class SessionsForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

closeSessionButton = new Button();

usernameTextBox = new TextBox();

refreshButton = new Button();

closeSessionsButton = new Button();

dataGridView = new DataGridView();

((System.ComponentModel.ISupportInitialize)dataGridView).BeginInit();

SuspendLayout();

//

// closeSessionButton

//

closeSessionButton.Location = new Point(292, 13);

closeSessionButton.Name = "closeSessionButton";

closeSessionButton.Size = new Size(146, 23);

closeSessionButton.TabIndex = 1;

closeSessionButton.Text = "Закрыть сессию";

closeSessionButton.UseVisualStyleBackColor = true;

closeSessionButton.Click += closeSessionButton\_Click;

//

// usernameTextBox

//

usernameTextBox.Enabled = false;

usernameTextBox.Location = new Point(12, 13);

usernameTextBox.Name = "usernameTextBox";

usernameTextBox.Size = new Size(157, 23);

usernameTextBox.TabIndex = 2;

//

// refreshButton

//

refreshButton.Location = new Point(175, 13);

refreshButton.Name = "refreshButton";

refreshButton.Size = new Size(111, 23);

refreshButton.TabIndex = 3;

refreshButton.Text = "Обновить";

refreshButton.UseVisualStyleBackColor = true;

refreshButton.Click += refreshButton\_Click;

//

// closeSessionsButton

//

closeSessionsButton.Location = new Point(444, 13);

closeSessionsButton.Name = "closeSessionsButton";

closeSessionsButton.Size = new Size(147, 23);

closeSessionsButton.TabIndex = 4;

closeSessionsButton.Text = "Закрыть все сессии";

closeSessionsButton.UseVisualStyleBackColor = true;

closeSessionsButton.Click += closeSessionsButton\_Click;

//

// dataGridView

//

dataGridView.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize;

dataGridView.Location = new Point(12, 42);

dataGridView.Name = "dataGridView";

dataGridView.Size = new Size(776, 396);

dataGridView.TabIndex = 5;

//

// SessionsForm

//

AutoScaleDimensions = new SizeF(7F, 15F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(800, 450);

Controls.Add(dataGridView);

Controls.Add(closeSessionsButton);

Controls.Add(refreshButton);

Controls.Add(usernameTextBox);

Controls.Add(closeSessionButton);

FormBorderStyle = FormBorderStyle.FixedSingle;

Name = "SessionsForm";

Text = "SessionsForm";

Load += SessionsForm\_Load;

((System.ComponentModel.ISupportInitialize)dataGridView).EndInit();

ResumeLayout(false);

PerformLayout();

}

#endregion

private Button closeSessionButton;

private TextBox usernameTextBox;

private Button refreshButton;

private Button closeSessionsButton;

private DataGridView dataGridView;

}

}

using Microsoft.VisualBasic;

using Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace University

{

public partial class UsersForm : Form

{

private UserService \_userService { get; set; }

private string? \_token { get; set; }

private string? \_currentUser { get; set; }

private int \_page { get; set; }

private int \_nextPage { get; set; }

private int \_prevPage { get; set; }

public UsersForm()

{

InitializeComponent();

\_userService = new UserService();

}

private async Task OpenLogin()

{

using (var dialogForm = new LoginForm())

{

Enabled = false;

if (dialogForm.ShowDialog() == DialogResult.OK)

{

\_token = dialogForm.Token;

\_currentUser = dialogForm.Username;

\_userService.Token = \_token;

await LoadData();

}

Console.WriteLine(\_token);

Enabled = true;

}

}

private async void войтиToolStripMenuItem\_Click(object sender, EventArgs e)

{

await OpenLogin();

}

private void информацияToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show($"Текущий пользователь: {\_currentUser}\n" +

$"Текущий токен: {\_token}");

}

private async Task LoadData()

{

try

{

dataGridView.Columns["Id"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

dataGridView.Columns["Username"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

dataGridView.Columns["FullName"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

dataGridView.Columns["Role"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

dataGridView.Columns["TokenAvailable"].AutoSizeMode =

DataGridViewAutoSizeColumnMode.Fill;

var list = await \_userService.GetUsers(\_page);

if (list != null)

{

if (list.Users!.Count == 0)

{

\_page = 1;

list = await \_userService.GetUsers(\_page);

}

dataGridView.DataSource = list.Users;

\_page = list.Page!.Value!;

\_nextPage = list.NextPage!.Value;

\_prevPage = list.PrevPage!.Value!;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK);

}

}

private async void refreshButton\_Click(object sender, EventArgs e)

{

await LoadData();

}

private async void UsersForm\_Load(object sender, EventArgs e)

{

await LoadData();

}

private async void prevButton\_Click(object sender, EventArgs e)

{

\_page = \_prevPage;

await LoadData();

}

private async void nextButton\_Click(object sender, EventArgs e)

{

\_page = \_nextPage;

await LoadData();

}

private async void revokeButton\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count != 0)

{

var selectedRow = dataGridView.SelectedRows[0];

var value = selectedRow.Cells["Id"].Value!.ToString();

await \_userService.RevokeToken(int.Parse(value!));

await LoadData();

}

}

private void openButton\_Click(object sender, EventArgs e)

{

if (dataGridView.SelectedRows.Count != 0)

{

try

{

var selectedRow = dataGridView.SelectedRows[0];

var value = selectedRow.Cells["Id"].Value!.ToString();

var username = selectedRow.Cells["Username"].Value!.ToString();

int userId = int.Parse(value!);

Enabled = false;

SessionsForm sessionsForm = new SessionsForm(\_userService, userId, \_token!, username!);

sessionsForm.Show();

Enabled = true;

}

catch (Exception ex)

{

MessageBox.Show("Неизвестая ошибка");

}

}

}

}

}

namespace University

{

partial class UsersForm

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

dataGridView = new DataGridView();

Id = new DataGridViewTextBoxColumn();

Username = new DataGridViewTextBoxColumn();

FullName = new DataGridViewTextBoxColumn();

Role = new DataGridViewTextBoxColumn();

TokenAvailable = new DataGridViewTextBoxColumn();

openButton = new Button();

revokeButton = new Button();

nextButton = new Button();

prevButton = new Button();

menuStrip = new MenuStrip();

менюToolStripMenuItem = new ToolStripMenuItem();

войтиToolStripMenuItem = new ToolStripMenuItem();

информацияToolStripMenuItem = new ToolStripMenuItem();

refreshButton = new Button();

((System.ComponentModel.ISupportInitialize)dataGridView).BeginInit();

menuStrip.SuspendLayout();

SuspendLayout();

//

// dataGridView

//

dataGridView.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize;

dataGridView.Columns.AddRange(new DataGridViewColumn[] { Id, Username, FullName, Role, TokenAvailable });

dataGridView.Location = new Point(12, 56);

dataGridView.Name = "dataGridView";

dataGridView.Size = new Size(776, 397);

dataGridView.TabIndex = 0;

//

// Id

//

Id.DataPropertyName = "Id";

Id.HeaderText = "Id";

Id.Name = "Id";

//

// Username

//

Username.DataPropertyName = "Username";

Username.HeaderText = "Имя пользователя";

Username.Name = "Username";

//

// FullName

//

FullName.DataPropertyName = "FullName";

FullName.HeaderText = "ФИО";

FullName.Name = "FullName";

//

// Role

//

Role.DataPropertyName = "Role";

Role.HeaderText = "Роль";

Role.Name = "Role";

//

// TokenAvailable

//

TokenAvailable.DataPropertyName = "TokenAvailable";

TokenAvailable.HeaderText = "Токен активен";

TokenAvailable.Name = "TokenAvailable";

//

// openButton

//

openButton.Location = new Point(93, 27);

openButton.Name = "openButton";

openButton.Size = new Size(75, 23);

openButton.TabIndex = 1;

openButton.Text = "Открыть";

openButton.UseVisualStyleBackColor = true;

openButton.Click += openButton\_Click;

//

// revokeButton

//

revokeButton.Location = new Point(174, 27);

revokeButton.Name = "revokeButton";

revokeButton.Size = new Size(134, 23);

revokeButton.TabIndex = 2;

revokeButton.Text = "Отозвать токен";

revokeButton.UseVisualStyleBackColor = true;

revokeButton.Click += revokeButton\_Click;

//

// nextButton

//

nextButton.Location = new Point(120, 459);

nextButton.Name = "nextButton";

nextButton.Size = new Size(107, 23);

nextButton.TabIndex = 3;

nextButton.Text = "Следующие";

nextButton.UseVisualStyleBackColor = true;

nextButton.Click += nextButton\_Click;

//

// prevButton

//

prevButton.Location = new Point(12, 459);

prevButton.Name = "prevButton";

prevButton.Size = new Size(102, 23);

prevButton.TabIndex = 4;

prevButton.Text = "Предыдущие";

prevButton.UseVisualStyleBackColor = true;

prevButton.Click += prevButton\_Click;

//

// menuStrip

//

menuStrip.Items.AddRange(new ToolStripItem[] { менюToolStripMenuItem });

menuStrip.Location = new Point(0, 0);

menuStrip.Name = "menuStrip";

menuStrip.Size = new Size(800, 24);

menuStrip.TabIndex = 5;

menuStrip.Text = "Меню";

//

// менюToolStripMenuItem

//

менюToolStripMenuItem.DropDownItems.AddRange(new ToolStripItem[] { войтиToolStripMenuItem, информацияToolStripMenuItem });

менюToolStripMenuItem.Name = "менюToolStripMenuItem";

менюToolStripMenuItem.Size = new Size(53, 20);

менюToolStripMenuItem.Text = "Меню";

//

// войтиToolStripMenuItem

//

войтиToolStripMenuItem.Name = "войтиToolStripMenuItem";

войтиToolStripMenuItem.Size = new Size(148, 22);

войтиToolStripMenuItem.Text = "Войти";

войтиToolStripMenuItem.Click += войтиToolStripMenuItem\_Click;

//

// информацияToolStripMenuItem

//

информацияToolStripMenuItem.Name = "информацияToolStripMenuItem";

информацияToolStripMenuItem.Size = new Size(148, 22);

информацияToolStripMenuItem.Text = "Информация";

информацияToolStripMenuItem.Click += информацияToolStripMenuItem\_Click;

//

// refreshButton

//

refreshButton.Location = new Point(12, 27);

refreshButton.Name = "refreshButton";

refreshButton.Size = new Size(75, 23);

refreshButton.TabIndex = 6;

refreshButton.Text = "Обновить";

refreshButton.UseVisualStyleBackColor = true;

refreshButton.Click += refreshButton\_Click;

//

// UsersForm

//

AutoScaleDimensions = new SizeF(7F, 15F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(800, 494);

Controls.Add(refreshButton);

Controls.Add(prevButton);

Controls.Add(nextButton);

Controls.Add(revokeButton);

Controls.Add(openButton);

Controls.Add(dataGridView);

Controls.Add(menuStrip);

FormBorderStyle = FormBorderStyle.FixedSingle;

MainMenuStrip = menuStrip;

Name = "UsersForm";

StartPosition = FormStartPosition.CenterScreen;

Text = "Пользователи";

Load += UsersForm\_Load;

((System.ComponentModel.ISupportInitialize)dataGridView).EndInit();

menuStrip.ResumeLayout(false);

menuStrip.PerformLayout();

ResumeLayout(false);

PerformLayout();

}

#endregion

private DataGridView dataGridView;

private Button openButton;

private Button revokeButton;

private Button nextButton;

private Button prevButton;

private MenuStrip menuStrip;

private ToolStripMenuItem менюToolStripMenuItem;

private Button refreshButton;

private ToolStripMenuItem войтиToolStripMenuItem;

private ToolStripMenuItem информацияToolStripMenuItem;

private DataGridViewTextBoxColumn Id;

private DataGridViewTextBoxColumn Username;

private DataGridViewTextBoxColumn FullName;

private DataGridViewTextBoxColumn Role;

private DataGridViewTextBoxColumn TokenAvailable;

}

}