МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра Информационные системы

Дисциплина Проектирование и архитектура программных систем

**КУРСОВОЙ ПРОЕКТ**

Тема Проектирование системы для обучения. Web-модуль преподавателя.

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Д. А. Арутюнян /

подпись инициалы, фамилия

Курс третий Группа ПИбд-32

Направление/ специальность 09.03.04 Программная инженерия (профиль Программная инженерия)

Руководитель ст. преподаватель

должность, ученая степень, ученое звание

Корунова Надежда Владимировна

фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

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

Ульяновск

2023 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра Информационные системы

Дисциплина Проектирование и архитектура программных систем

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

студенту ПИбд-32 Арутюнян Д. А.

группа фамилия, инициалы

**Тема проекта** Проектирование системы для обучения. Web-модуль преподавателя.

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

**Срок сдачи законченного проекта** «26» мая 2023 г.

**Исходные данные к проекту** Описание задания по теме, утверждённой

(базовое предприятие, характер курсового проекта (работы):

распоряжением деканата ФИСТ. Курс лекций по дисциплине «Проектирование и

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

архитектура программных систем», методические указания к лабораторным работам по дисциплине «Проектирование и архитектура программных систем», интернет-источники.

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

Описание предметной области

Техническое задание на разработку ИС

Проектное решение ИС

Контрольный пример реализации проектного решения

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

Диаграмма классов

Диаграмма сущностей и связей

Диаграмма последовательности

Диаграмма вариантов использования

Диаграмма развертывания

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

должность подпись инициалы, фамилия

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Д. А. Арутюнян /

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_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 Арутюнян Дмитрия Аркадьевича

фамилия, имя и отчество

Факультет\_\_\_ИСТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_группа\_\_\_ПИбд-32\_\_\_курс\_\_3\_\_\_\_

Дисциплина\_\_ Проектирование и архитектура программных систем\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема проекта (работы) Проектирование системы для обучения. Web-модуль преподавателя.

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

**Содержание**

[Введение 5](#_Toc137731600)

[Описание предметной области 6](#_Toc137731601)

[1. Поиск аналогов 8](#_Toc137731602)

[Техническое задание на разработку ИС 10](#_Toc137731603)

[1. Назначение системы 10](#_Toc137731604)

[2. Цели создания системы 10](#_Toc137731605)

[3. Требования к системе 11](#_Toc137731606)

[4. Требования к видам обеспечения 13](#_Toc137731607)

[5. Атрибуты качества системы 14](#_Toc137731608)

[6. Сквозная функциональность системы 15](#_Toc137731609)

[Проектное решение ИС 17](#_Toc137731610)

[1. Выбор, оценка и описание архитектурного решения 17](#_Toc137731611)

[2. Диаграмма классов 24](#_Toc137731612)

[3. ER-диаграмма 29](#_Toc137731613)

[4. Диаграмма вариантов использования 30](#_Toc137731614)

[5. Диаграмма последовательности 31](#_Toc137731615)

[6. Диаграмма развёртывания 32](#_Toc137731616)

[Контрольный пример реализации проектного решения 33](#_Toc137731617)

[1. Описание технологий разработки 33](#_Toc137731618)

[2. Описание основных экранов web-модуля преподавателя 34](#_Toc137731619)

[3. Сгенерированные классы по коду 38](#_Toc137731620)

[4. Выводы по реализации приложения 39](#_Toc137731621)

[Заключение 40](#_Toc137731622)

[Список литературы 41](#_Toc137731623)

[Приложение 42](#_Toc137731624)

[1. Листинг кода 42](#_Toc137731625)

**Введение**

В современном мире информационных технологий образование переживает своеобразную революцию. Все больше внимания уделяется созданию электронных образовательных ресурсов, которые позволяют получать знания и умения на удалённой основе: в любое время суток и из любой точки мира. В данной работе будет рассмотрено проектирование системы для обучения, в которой будет использоваться web-модуль преподавателя для заполнения учебных планов, создания расписания занятий и других способов организации учебных процессов, а также для ведения учёта успеваемости и контроля прогресса студентов и обратной связи с модулем администратора (непосредственное начальство, кафедра, директорат и др. органы образовательного управления) посредством стандартизированной отчётности.

Цель курсовой работы: разработка системы для обучения, включающей в себя Web-модуль преподавателя.

Объект исследования: системы электронного обучения и процесс, связанный с разработкой программного обеспечения.

Предмет исследования: модуль преподавателя в системе электронного обучения.

Методы исследования: анализ требований пользователя, проектирование архитектуры программного обеспечения, использование современных технологий программирования, тестирование и отладка программы.

Результатом курсовой работы будет являться функционирующая система обучения, позволяющая работать web-модулю преподавателя и модулю администратора в связке. Разработанный web-модуль преподавателя позволит осуществлять тайм-менеджмент занятий по учебным планам, управлять успеваемостью студентов и взаимодействовать с модулем администратора через отчётность, передаваемую по электронной почте.

**Описание предметной области**

Система для обучения – это совокупность программных и аппаратных средств, которые позволяют преподавательскому составу и администрации образовательного учреждения получать обратную связь друг от друга с целью взаимоконтроля образовательного процесса и предоставления структурированных знаний студентам. Система состоит из нескольких модулей, каждый из которых выполняет свою функцию.

Web-модуль преподавателя является одним из модулей системы для обучения и представляет собой интерфейс для преподавателей, через который они могут распределять часы занятий требуемого учебного плана и вести учет результатов испытаний студентов.

В процессе функционирования объекта предоставления образования, в данном случае университета, значительную роль играет вопрос автоматизации учета успеваемости студента по различным дисциплинам с различными занятиями. В данной области необходима статистика и если вести ее вручную, то есть вероятность ошибок, а также на это уходит определенное время. Многие программы, работающие с базами данных, рассчитывают статистику с легкостью, что является полезной функцией.

Главной задачей web-модуля преподавателя является создание удобной и функциональной среды для управления занятиями студентов структурированных по дисциплинам, группам и учебным планам, ведения учёта успеваемости студентов, а также обеспечение возможности получения отчетов по результатам испытаний студентов в разрезе учебного плана и дисциплины в целом в виде текстовых документов, как и их последующей отправки на проверку и официального заверения администрацией используя средства электронной отправки почтовых сообщений.

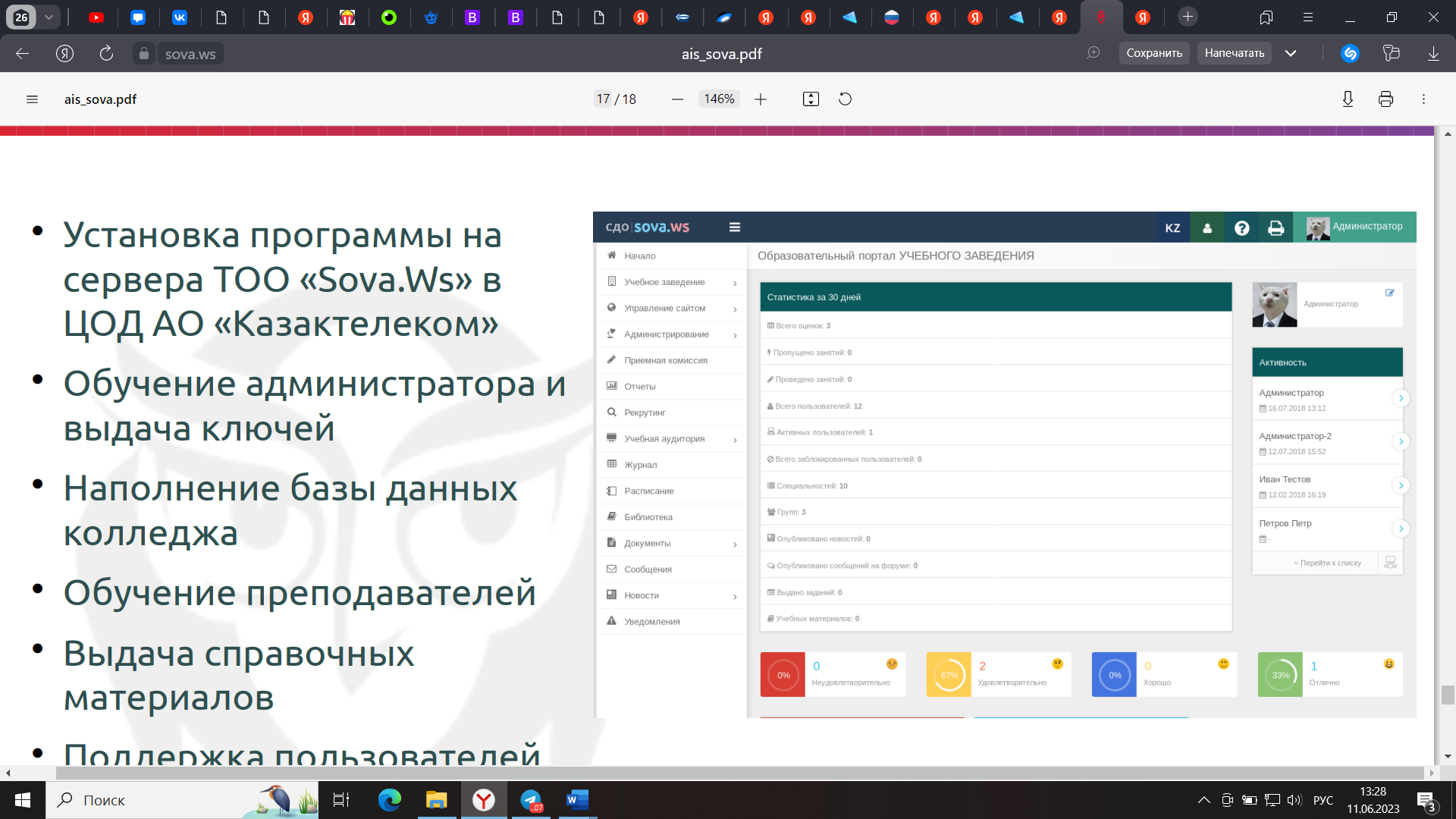
Web-модуль преподавателя обладает следующими функциональными возможностями:

* Управление графиком курса: преподаватель может определять даты заданий и тестов, управлять расписанием занятий и контролировать прогресс студентов.
* Онлайн-успеваемость и оценка: преподаватель может выставлять студентам оценки и/или посещаемость по конкретным занятиям используя удобные формы поиска и навигации по дисциплинам, группам и планам.
* Взаимодействие с администрацией: через модуль преподаватель может выполнять поручения своего непосредственного начальства по отправке отчётов и ведомостей об успеваемости студентов на проверку и официальное заверение.
* Самоконтроль и таим-менеджмент: посредством уведомлений преподаватель может отслеживать текущие занятия, требующие своевременной проверки и оценки успеваемости студентов, а также активные поручения от администрации.

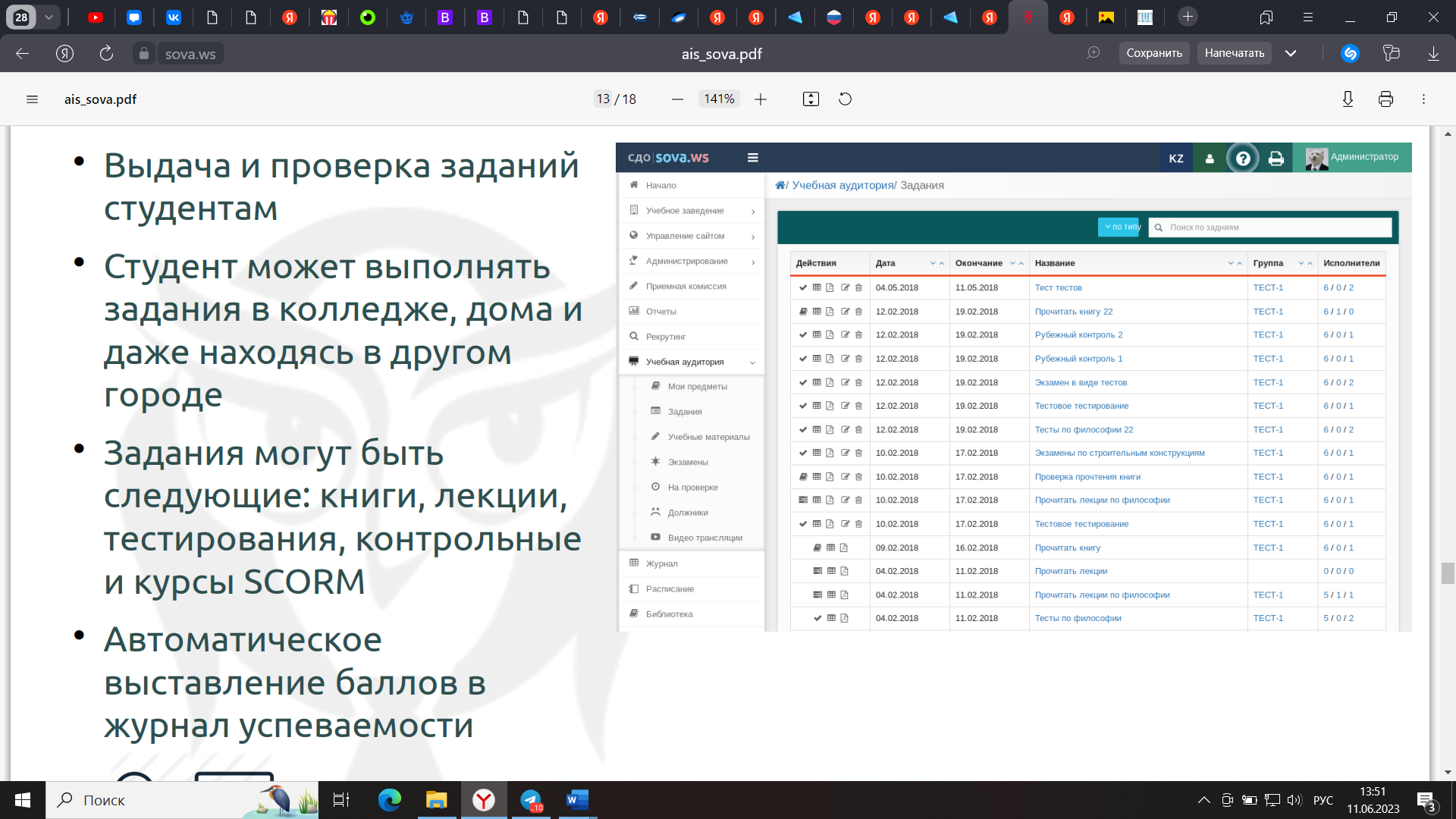
Таким образом, web-модуль преподавателя является важным компонентом системы для обучения, который позволяет организовать удобное и эффективное онлайн-управление образовательной деятельности.

1. **Поиск аналогов**

Система автоматизации учебного процесса «Сова» (Рис. 1-2.)



*Рис. 1. Главное окно модуля преподавателя системы «Sova».*



*Рис. 2. Окно ведения учёта успеваемости студентов модуля преподавателя системы «Sova».*

Основное назначение системы – это организация информационного взаимодействия между преподавателями, учащимися и их родителями. Система позволяет составлять и просматривать расписания уроков, выставлять отметки и писать замечания в классный журнал и дневник, выдавать общие и индивидуальные домашние задания, выполнять задания учащимися, проверять задания преподавателями.

Преимущества системы:

* + Простое внедрение и небольшая системная требовательность.
  + Широкий функционал, включающий в себя ведение онлайн-конференций.

Недостатки системы:

* + Проблемы с валидацией данных и отказоустойчивости системы.
  + Недоступность персонального обслуживания модулей системы.
  + Отсутствие распределение прав доступа как на модульном, так и на системном уровне.
  + Отсутствие связи модулей преподавателя и администрации (нежелательное упрощение системы до моно-модульности на данном узле).

**Техническое задание на разработку ИС**

1. **Назначение системы**

Система для обучения – это специальный программный продукт, который предназначен для решения одной из главных потребностей рынка в сфере образования на данное время: организации образовательной деятельности и управления ей удаленно через интернет.

Web-модуль преподавателя – это конкретный инструмент, который предназначен для управления датами и длительностью занятий, мониторинга процесса обучения студентов, а также оценки их успеваемости. Преподаватель может добавлять новые занятия, проверять выполнение заданий и оценивать полученные результаты студентов, а также коммуницировать с администрацией через стандартизированную отчётную документацию, формирующуюся автоматически.

1. **Цели создания системы**

Система для обучения разрабатывается с целью:

* Универсального хранения данных об административном органе, преподавателях, студентах образовательного учреждения.
* Обеспечения удалённой связи между отдельными участниками образовательного процесса (конкретно ведение учёта успеваемости студентов, формирование отчётов об успеваемости и посещаемости, формирование отчётов о поступивших студентов и их распределения по группам).
* Организации и управления учебным и преподавательским процессами.

Конкретно web-модуль преподавателя разрабатывается с целью:

* Обеспечения преподавателей возможностью создания и управления занятиями в разрезах дисциплин и учебных планов.
* Предоставления преподавателям возможности удалённого управления успеваемостью студентов.
* Автоматизации составления отчётов, требуемых от преподавателя административным органом образовательного учреждения.
* Обеспечения преподавателей обратной связью с их непосредственными органами управления через систему и электронную почту.
* Упрощения мониторинга обязанностей преподавателей средствами вывода необходимой актуальной информации на главную форму модуля.

1. **Требования к системе**

Система в целом должна представлять из себя составной программный продукт, включающий в себя серверную часть и модули администратора и преподавателя, с возможностью дальнейшего его интегрирования в учреждения образовательного назначения. Конкретно web-модуль преподавателя должен представлять собой программный продукт, предполагающий работу в браузерах стандартного и нового образца мобильных и десктопных версий, доступных пользователю.

Требования к функциям, выполняемым системой:

* Ввод данных. Входные данные: данные, вводимые пользователем (занятия, список оценок студентов). Выходные данные: вывод введенных данных на страницу и сохранение результатов в базе данных.
* Редактирование данных. Входные данные: данные справочников. Выходные данные: вывод измененных данных на страницу и сохранение результатов в базе данных
* Удаление данных. Входные данные: данные справочников. Выходные данные: удаление данных из таблиц базы данных
* Формирование занятия. Входные данные: данные справочников. Выходные данные: вывод сформированной формы на страницу, с распределением по дисциплине и учебному плану.
* Формирование результатов успеваемости. Входные данные: данные справочников. Выходные данные: вывод сформированной формы занятия со списком студентов и соответствующих им оценок успеваемости на страницу.
* Поиск занятия. Входные данные: дата занятия, тема занятия, группа, у которой проводится занятие и дисциплина, по которой проводится занятия. Выходные данные: вывод сформированной формы занятия со списком студентов и соответствующих им оценок успеваемости на страницу.
* Фильтрация поручений. Входные данные: тип поручения, тема поручения. Выходные данные: вывод сформированной формы поручения с указанием его статуса.
* Формирование отчёта по успеваемости студентов в разрезе учебного плана. Входные данные: данные справочников. Выходные данные: вывод сформированного в соответствии со стандартами образовательной документации отчёта на страницу.
* Формирование отчёта по успеваемости студентов в разрезе дисциплины. Входные данные: данные справочников. Выходные данные: вывод сформированного в соответствии со стандартами образовательной документации отчёта на страницу.
* Сохранение отчёта в файл формата «.docx». Входные данные: данные справочников Выходные данные: сохранение сформированного файла в формат «.docx».
* Отправка отчёта на электронную почту. Входные данные: данные справочников. Выходные данные: отправка сформированного файла в формате «.docx» на указанный e-mail с обозначением темы и текста письма.

1. **Требования к видам обеспечения**

Вид пользовательского управляющего и справочного интерфейса web-модуля преподавателя должен быть как интуитивно понятен пользователю, так и сочетать в себе спокойные тона во избежание утомления за работой и броские элементы графического дизайна с целью привлечения внимания аудитории. Пользователю предоставляется удобный графический интерфейс, в спокойной цветовой гамме (голубой – лиловый – серый – белый – чёрный) с авторскими рисунками на каждой странице в тех же тонах и подписями элементов управления.

К квалификации персонала, эксплуатирующего web-модуль преподавателя системы обучения, предъявляется перечень требований, таких как: обладание конечного пользователя знаниями соответствующей предметной области, навык конечного пользователя в средствах управления ПК и браузерными приложениями среднего уровня.

Спецификации по информации:

* Исходные данные вводятся пользователем вручную, с помощью клавиатуры
* Данные должны храниться в БД
* Данные должны обрабатываться в СУБД

Требования к программным средствам:

* Google Chrome 21.x – 32.x, Yandex, Mozila FireFox v 4.5 и выше
* Microsoft SQL Server 2019 Express

1. **Атрибуты качества системы**

На основе перечня пользовательских, функциональных и нефункциональных требований к системе были выявлены и описаны необходимые атрибуты качества для данного проектного решения, на основе которых будет проводиться формирование архитектурного решения системы.

*Безопасность*. Разрабатываемая система будет содержать персональные данные сотрудников и студентов учебного заведения. Соответственно, от архитектуры требуется безопасность (у не авторизованного пользователя не может быть доступа к данным приложения).

*Доступность*. Приложение будет устанавливаться на различные desktop-машины и открываться через различные браузеры, поэтому от архитектуры будет требоваться обеспечение доступности. Самой производительной частью архитектуры должен быть сервер, поскольку требования к клиентским машинам должны быть минимальны (Desktop: Windows 7,8,10 с минимальными требованиями системы, Web-клиент: Браузер Google Chrome не ниже 70, Yandex не ниже 15, FireFox не ниже 4.5, Microsoft Edge любой версии).

*Удобство сопровождения.* Приложение должно быть удобным для использования клиентом и не требовать регулярных обновлений, поэтому от архитектуры требуется возможность сопровождения приложения без изменения клиентской части. (обновление клиентской части приложения не чаще раза в год)

*Функциональность.* Также приложение взаимодействует со сторонними системами редактирования документов (могут быть MS Word, MS Excel, Libre Office и др.), поэтому желательно обеспечить данное взаимодействие без участие клиентской части архитектуры. (приложение должно выполнять требуемый функционал без установки данных дополнительных программ на клиентскую машину)

*Производительность.* Ответ от сервера клиент должен получать не более чем за 10 секунд.

*Удобство использования.* Пользователю должно быть удобно перемещаться по клиентскому приложению, тогда весь основной функционал должен располагаться на основном меню, а дополнительный на расстоянии не более 3х страниц-кликов.

*Переносимость.* Приложение редко будет переноситься с одной клиентской машины на другую, поэтому требования по переносимости минимальны. Приложение должно разворачиваться в одном определённом окружении (Клиент – Windows/Browser, Сервер – Windows).

*Масштабируемость.* Функционал приложения не будет кардинально становиться ресурс-затратнее, а также кол-во пользователей не будет кардинально увеличиваться, поэтому требования для масштабируемости минимальны (не более 1000 пользователей).

*Время реализации.* Время разработки средне-ограничено, поэтому на разработку приложения будет выделено 187 календарных дней.

*Стоимость.* Приложение не использует никакие платные дополнительные пакеты и подразумевает реализацию на архитектуре без сложных, новых решений. Поэтому ценовая категория приложения будет средняя, ниже средний.

1. **Сквозная функциональность системы**

Системы обучения оперируют с информацией различного типа, которая должна быть типизирована и стандартизирована в конечном итоге. Учитывая факт минимизации требований по опыту владения для пользователей и обширного объёма вводимой вручную информации, сквозной функциональностью системы будет являться валидация.

Валидация – это процесс проверки корректности данных, вводимых пользователем перед тем, как они будут обработаны и сохранены в базе данных. Это очень важный процесс в системе онлайн-обучения, потому что он гарантирует, что в систему вводятся только правильные и допустимые данные, что повышает уровень безопасности, минимизирует количество ошибок и улучшает качество работы всей системы в целом.

Виды валидации, используемые в данной системе:

* Проверка на отсутствие пустых полей – это гарантирует, что все поля, обязательные для заполнения, заполнены.
* Проверка данных на уникальность – это гарантирует, что каждый пользователь или материал курса имеет уникальный идентификатор.
* Проверка формата и правильности данных (например, валидация адресов электронной почты) – это гарантирует, что все данные корректны и в правильном формате.
* Проверка доступа к данным – это гарантирует безопасность данных студентов и преподавателей и предотвращает несанкционированный доступ к ним.

В данной системе валидация будет сквозь все уровни реализуемого проекта (уровни будут зависеть от выбранной архитектуры, ориентировочно: уровень базы данных, серверный уровень, клиентский уровень) и предотвращать возникновение ошибок на всех этапах системы.

**Проектное решение ИС**

1. Выбор, оценка и описание архитектурного решения

Согласно обозначенным атрибутам качества системы и анализа её составных частей в качестве основы архитектуры был выбран клиент-сервер.

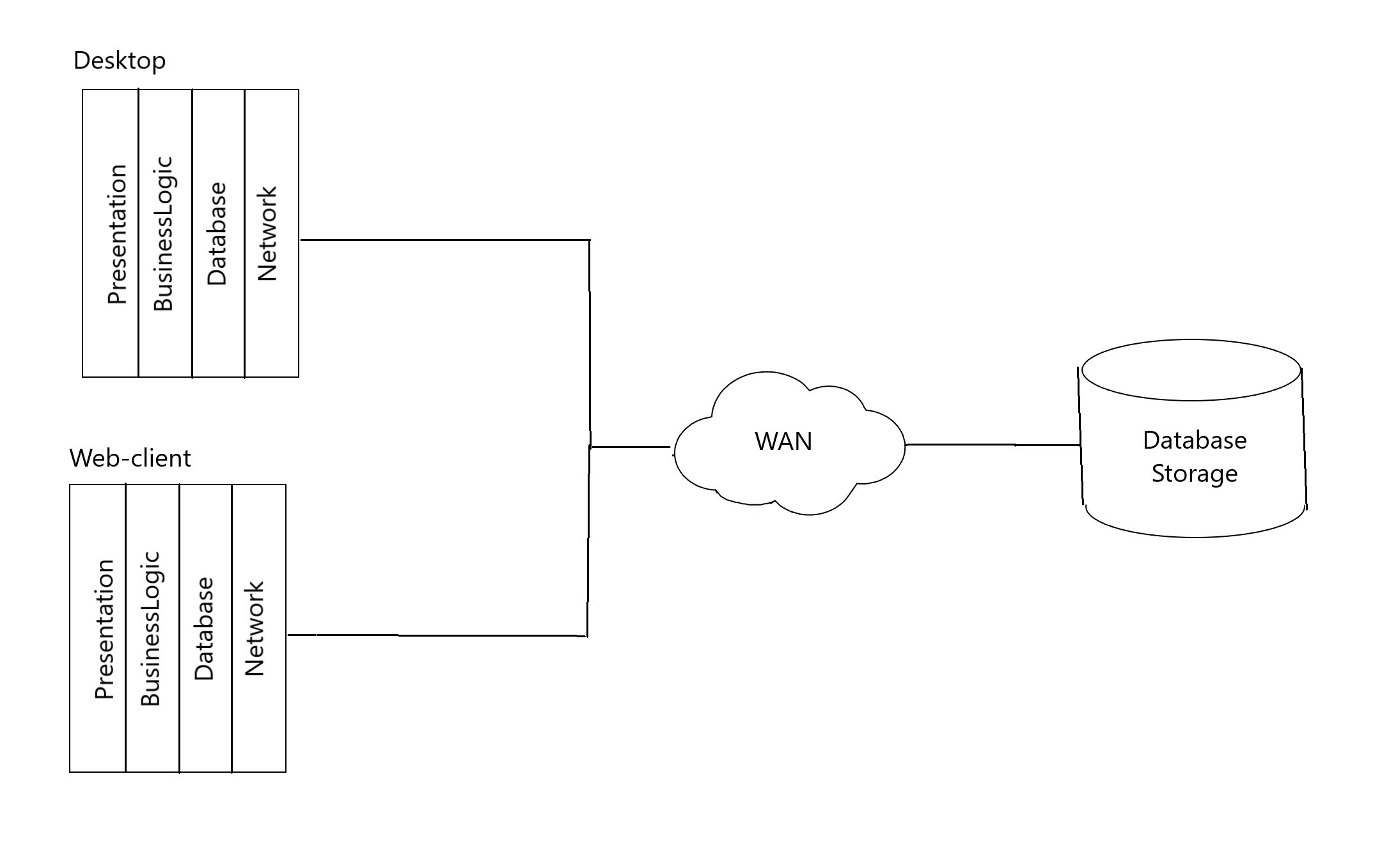
Клиент-сервер – это распределенная архитектура, которая используется для взаимодействия клиента (пользователя) и сервера. В данной архитектуре клиент и сервер могут находиться на разных физических машинах, в разных сетях или даже на разных континентах.

В проектируемой системе обучения в качестве клиентов будут выступать модули преподавателя и администратора. Они будут размещаться на отдельных настольных клиентских машинах, либо передаваться в управление через web-пространство. Серверная часть может находиться как на собственной машине образовательного учреждения при наличии для неё обслуживающего персонала, так и на предоставляемом сервисе.

В соответствии с этим, архитектурное решение клиент-сервера системы обучения может быть представлено различными способами. Были рассмотрены наиболее подходящие три из них и проведены их оценки в соответствии с требуемыми атрибутами качества системы.

Архитектура 1

Одноуровневый клиент-сервер

****

*Рис. 3. Одноуровневый клиент-сервер.*

Статическая структура:

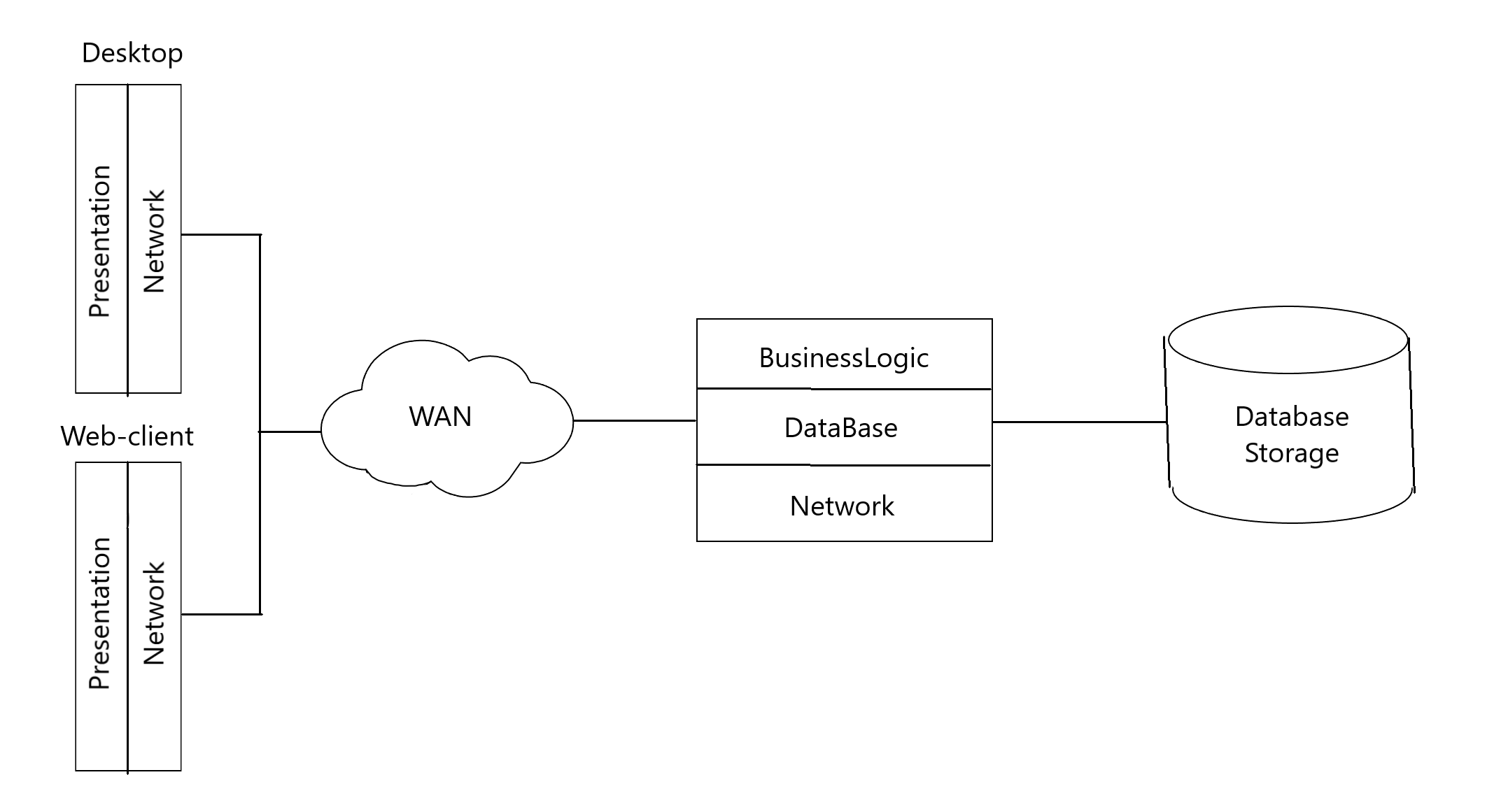
* Клиенты (Desktop и Web-client)
* Сервер
* Соединения

Динамическая структура:

* Одноуровневая модель «запрос/ответ»
* Запрос идёт от клиента к серверу через сеть
* Ответ возвращается от сервера к клиенту через сеть

Архитектура 2

Трёхуровневый клиент-сервер

****

*Рис. 4. Трёхуровневый клиент-сервер.*

Статическая структура:

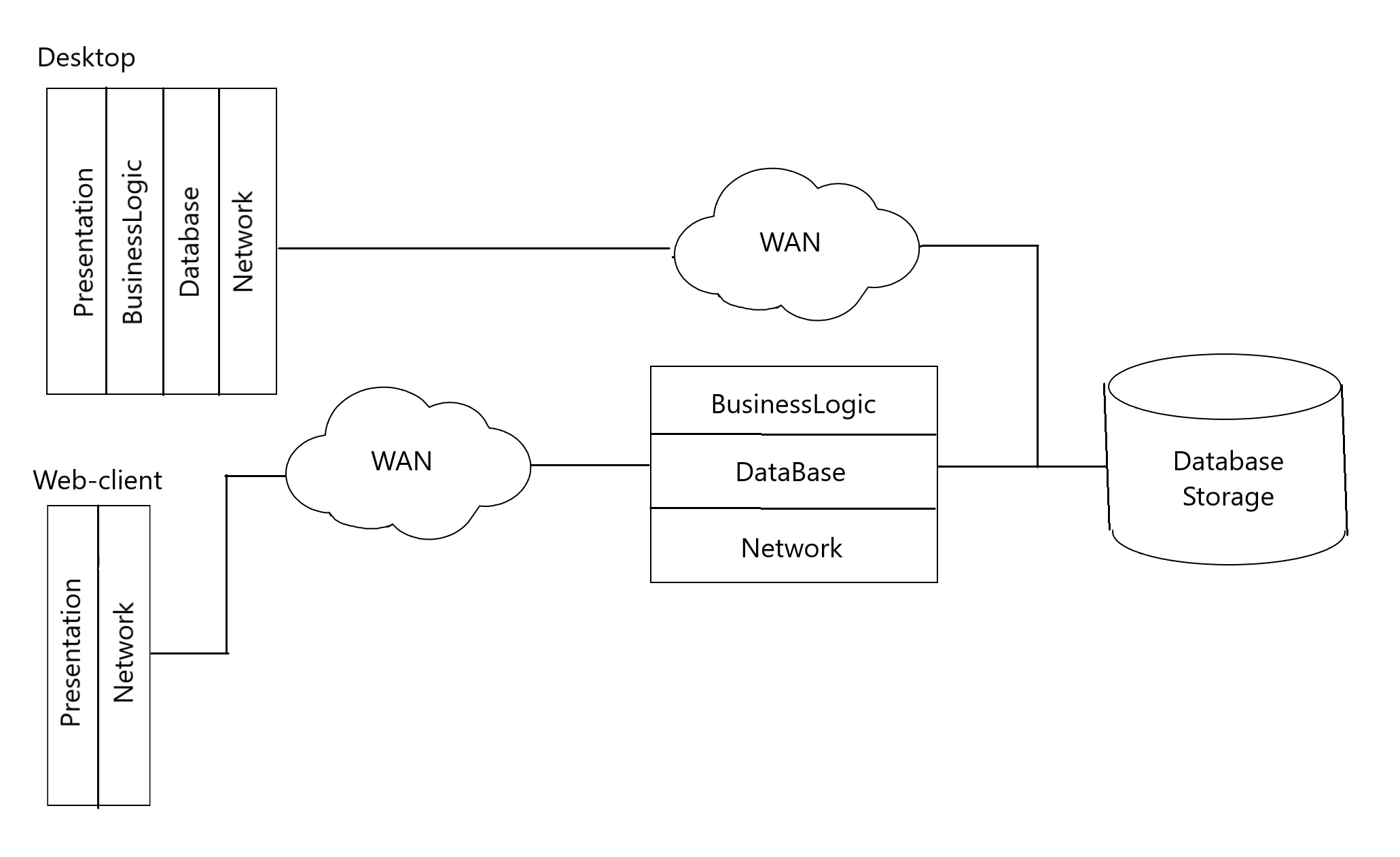
* Клиенты (Desktop и Web-client)
* Сервер приложения
* Сервер баз данных
* Соединения

Динамическая структура:

* Трёхуровневая модель «запрос/ответ»
* Запрос идёт от клиента идёт к серверу приложения через сеть, сервер приложения передаёт запрос серверу базы данных
* Ответ от сервера базы данных получает сервер приложения и, если необходимо, через сеть пересылает его клиенту

Архитектура 3

Клиент-сервер со смешанными уровнями (многослойный)

****

*Рис. 5. Клиент-сервер со смешанными уровнями (многослойный).*

Статическая структура:

* Клиенты (Desktop и Web-client)
* Сервер приложения
* Сервер баз данных
* Соединения

Динамическая структура:

* Многослойная модель «запрос/ответ» со смешанными уровнями
* (Слой Desktop)
* Запрос идёт от клиента к серверу базы данных через сеть
* Ответ возвращается от сервера базы данных к клиенту через сеть
* (Слой Web-client)
* Запрос идёт от клиента идёт к серверу приложения через сеть, сервер приложения передаёт запрос серверу базы данных
* Ответ от сервера базы данных получает сервер приложения и, если необходимо, через сеть пересылает его клиенту

Анализ архитектур проектного решения проводится методом составления ранговой матрицы, с учётом приоритета каждого атрибута качества, заданного в его описании на основе роли, играющей в итоговой системе и наличия доступного способа реализации данного атрибута в исследуемой архитектуре.

Выбор приоритетной архитектуры клиент-сервера для системы осуществляется по максимальной сумме значений ранговой матрицы каждого архитектурного решения (обозначены по степени градиента белый «плохо» – красный «хорошо» в итоговой строке ранговой матрицы):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут качества** | **Веса** | **Архитектура 1** | **Архитектура 2** | **Архитектура 3** |
| Функциональность | 0,292897 | 0,292897 | 1,171588 | 0,878691 |
| Производительность | 0.0845635 | 0,253691 | 0,253691 | 0,338254 |
| Удобство использования | 0,0645635 | 0,129127 | 0,258254 | 0,193691 |
| Удобство сопровождения | 0,192897 | 0,192897 | 0,771588 | 0,192897 |
| Переносимость | 0,0478968 | 0,095794 | 0,191587 | 0,095794 |
| Доступность | 0,142897 | 0,142897 | 0,571588 | 0,285794 |
| Масштабируемость | 0,03361 | 0 | 0,13444 | 0,06722 |
| Безопасность | 0,109563 | 0,219126 | 0,438252 | 0,328689 |
| Время реализации | 0,021 | 0,084 | 0,042 | 0,021 |
| Стоимость | 0,0101122 | 0,040449 | 0,020224 | 0 |
| Итого (общих) |  | 1,450877 | 3,853212 | 2,402029 |

*Таблица 1. Ранговая матрица оценки архитектур системы*.

Исходя из таблицы, архитектурой-победителем была выявлена «Трёхуровневый клиент-сервер». В таком случае, данная архитектура будет использоваться для создания системы, так как она показывает наибольшую компромиссность между всеми атрибутами качества и наилучшим образом охватывает более требуемые из них.

Опираясь на выбранную наиболее подходящую архитектуру системы обучения «Трёхуровневый клиент-сервер» было проведено распределение всех модулей проекта на соответствующие части архитектуры. Посредством определения модулей как составных частей архитектуры и выявления связей между ними были заданы соответствующие фреймворки для каждого уровня системы:

Уровень «клиент».

В системе обучения представлен desktop-модулем администратора и web-модулем преподавателя. Архитектурно данные решения представляют собой тонких клиентов, реализующих пользовательский интерфейс и обращающихся к логике сервера через интернет посредством HTTP/HTTPS запросов. Web-модуль преподавателя реализует фреймворк ASP .Net Core MVC, позволяющий использовать технологию отрисовки интерфейса совместимо с описанием контроллеров .Net Razor Pages. Модуль администратора объединяет в себе два фреймворка: .Net Framework для использования системы построения интерфейса WPF и ASP .Net Core MVC для передачи запросов. Использование MVC фреймворка позволяет обеспечивать обозначенную сквозную функциональность – валидацию – на данном уровне через задание типизированных классов объектов и последующей их серриализации в формат Json. Также благодаря возможностям описания интерфейсов .Net Framework WPF и .Net Razor Pages валидация входных данных происходит и на уровне пользовательского интерфейса.

Уровень «сервер приложения».

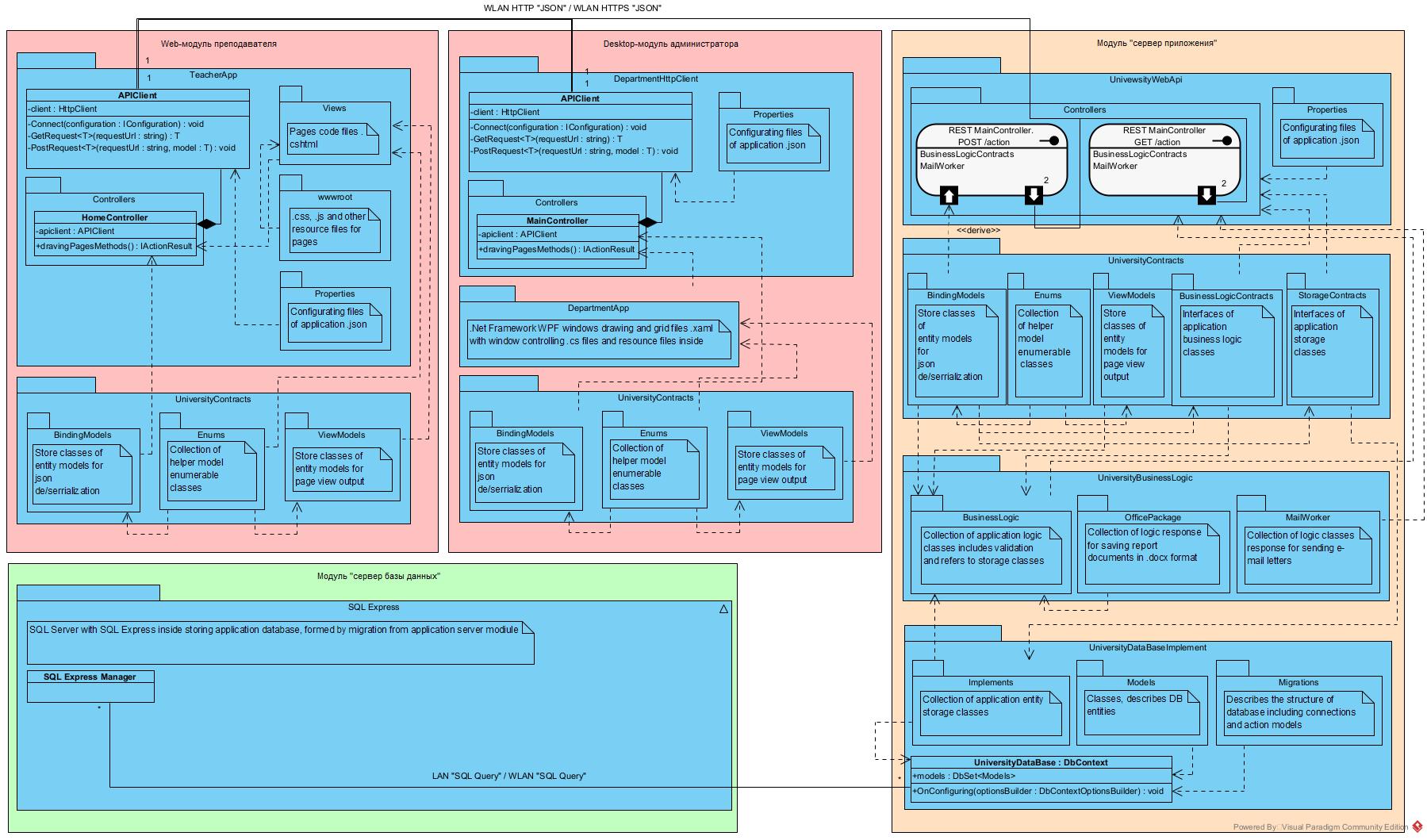
В системе обучения представлен модулем сервера приложения. Архитектурно данное решение представляет собой совокупность библиотек классов сущностей объектов, обработчиков и интерфейсов логики и хранения данных, и контроллеров обработки web-запросов и запросов к базе данных. Реализация контроллера передачи/получения пакетов данных через интернет реализована через фреймворк ASP .Net Core MVC средствами REST API. Десериализация данных проводится методами Json конвертера по типовым моделям, что позволяет обеспечить их валидацию на этапе получения/отправки. Контроллер обращения к базе данных реализован фреймворком .Net Entity Framework, что позволяет осуществлять запросы методом LINQ. В библиотеках классов логики и хранения данных также описаны методы валидации, что позволяет ограничить попадание несоответствующих моделей на уровни клиента и сервера базы данных и поддерживает сквозную функциональность приложения.

Уровень «сервер базы данных»

В системе обучения представлен сервером Microsoft SQL Server и реализован системой SQL Express. Обеспечение подключения базы данных к серверу приложения может осуществляться как по локальной сети, так и удалённо в зависимости от возможностей его расположения заказчиком. Обеспечивается это средствами управления сущностями фреймворка .Net Entity Framework, расположенными на уровне сервера приложения. Такое решение повышает доступность и удобство сопровождения системы.

1. **Диаграмма классов**

Для более явного отображения архитектурного решения проекта и его представления с использованием обозначенных технологий программирования, фреймворков и паттернов была составлена диаграмма классов.



*Диаграмма 1. Общая диаграмма классов (уровень классов/пакетов).*

На данной диаграмме чётко обозначены уровни архитектуры системы: уровень клиента обозначен красными блоками, уровень «сервер приложения» – оранжевым блоком и уровень «сервер базы данных» – зелёным блоком.

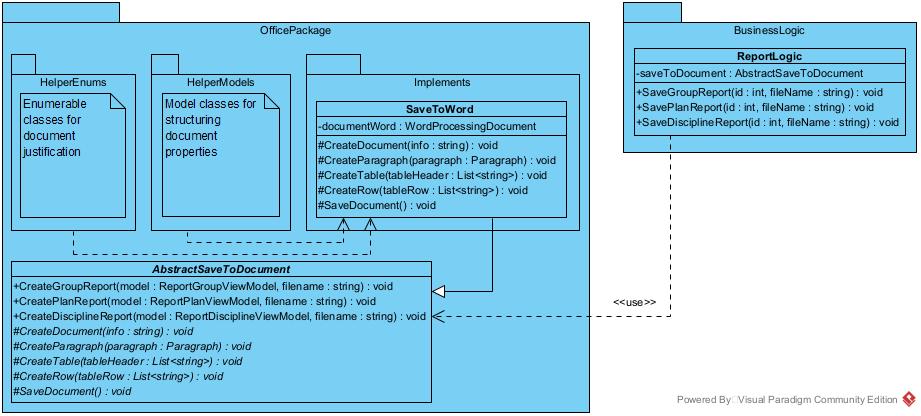
На модулях администратора и преподавателя приложениями Http клиентов реализован фреймворк ASP .Net Core MVC через класс APIClient и пакет контроллеров Controllers, в котором располагаются классы контроллеров. Данная композиция реализует web-клиенты модулей, благодаря которым происходит передача запросов серверу приложения по HTTP или HTTPS протоколам. Помимо клиентов, на данных модулях располагаются пакеты отрисовки окон или страниц и библиотека классов-моделей сущностей системы – UniversityContracts.

На модуле «сервер приложения» Web-API реализован через пакет контроллеров с классами REST контроллеров для GET и POST запросов, что также соответствует фреймворку ASP .Net Core MVC. Классы-модели сущностей системы, а также интерфейсы логики и хранилищ данных находятся в библиотеке классов UniversityContracts. Реализации интерфейсов данной библиотеки расположены в пакетах библиотек UniversityBusinessLogic и UniversityDataBaseImplement соответственно. Класс UniversityDataBase библиотеки классов UniversityDataBaseImplement реализует подключения сервера приложения к модулю «сервер базы данных» через .Net Entity Framework. Такая реализация позволяет использовать LINQ запросы к базе данных в классах хранилищ.

На модуле «сервер базы данных» располагается SQL Server с установленным менеджером базы данных SQL Express. Связь данного модуля с модулем «сервер приложения» осуществляется по локальной или глобальной сети с помощью SQL запросов.

Рассмотрение структурированных объектов диаграммы классов в частности приведено далее (Диаграмма 2–4).

Создание документов отчётности



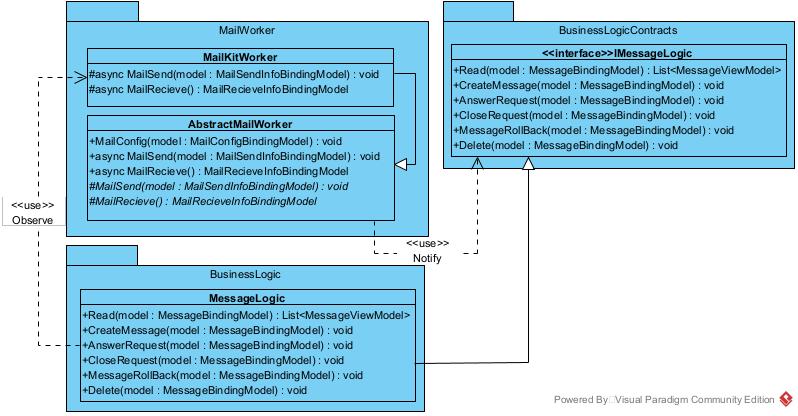
*Диаграмма 2. Диаграмма классов логики создания документов отчётности.*

Для ведения отчётности в системе используется формирование типизированной документации в файлы. Поскольку одним из главных атрибутов качества системы является доступность, было введена возможность использования различных обработчиков текстовой документации.

Для реализации этого был использован порождающий шаблон проектирования AbstractFactory. В пакете логики создания документации OfficePackage (библиотека классов UniversityBusinessLogic, модуль «сервер приложения») хранится абстрактный класс AbstractSaveToDocument. Данный класс позволяет создавать классу логики отчётов ReportLogic объект класса работы с документом, не указывая на его конкретный класс. Конкретизированным классом в данном случае выступает SaveToWord – компонент для создания файлов .docx.

Таким образом, при желании, компонент создания текстовых документов может быть заменён любым другим (компоненты Libre Office), либо новый компонент может быть добавлен к существующему для вариативности реализации.

Отправка сообщений на почту



*Диаграмма 3. Диаграмма классов управления e-mail сообщениями.*

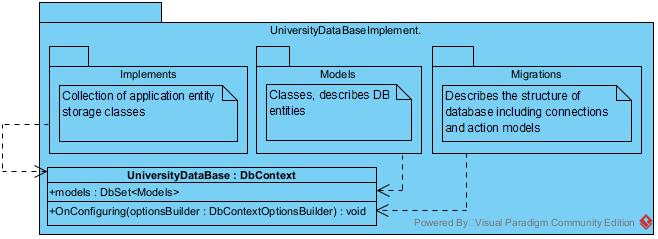
В системе одним из требований является наличие возможности отправки отчётности преподавателем администрации на проверку по электронной почте.

Данная функция реализована через поведенческий паттерн Observer. Класс логики сообщений MailKitWorker следит за сообщениями на почте приложения. При наличии нового сообщения, он оповещает класс логики сообщений через интерфейс последнего IMessageLogic и свою абстракцию AbstractMailWorker. Класс логики сообщений MessageLogic следит за оповещениями класса логики отправки сообщений и, при наличии таковых, создаёт, удаляет сообщение или изменяет его статус.

Благодаря такой технологии, при получении отчёта администрацией от преподавателя, статус данного отчёта меняется на «Проверяется», а при отправке письма-ответа, статус меняется на «Проверен».

Данная система обеспечивает пользовательскую связь между модулями преподавателя и администратора.

Управление базой данных



*Диаграмма 4. Диаграмма классов управления базой данных.*

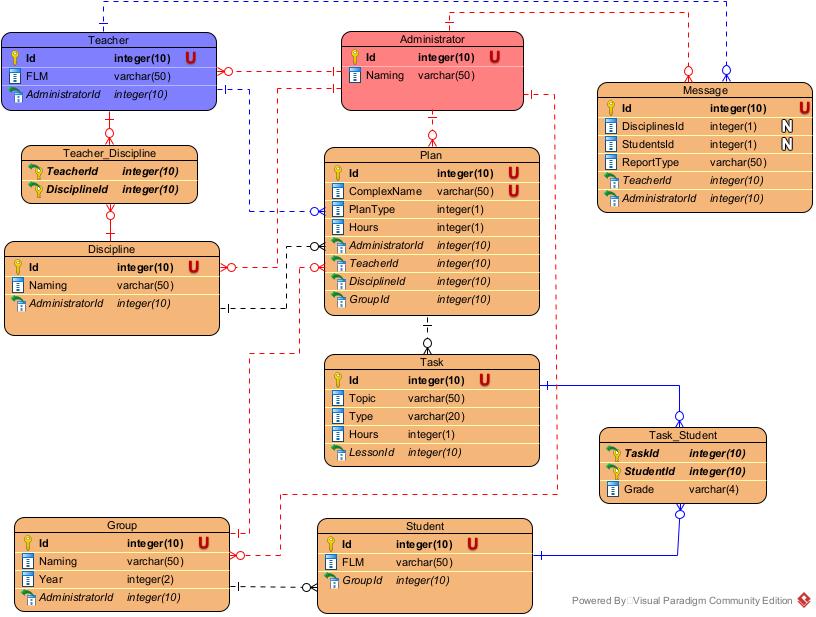
Хранение данных в системе происходит на отдельном уровне архитектурного решения, управление хранением данных, как и самой базой данных, осуществляется на сервере приложения.

За предоставление доступа к базе данных и передачу меж-серверных запросов отвечает класс UniversityDataBase, реализующий порождающий шаблон проектирования Singleton. Это позволяет избежать создания множественных экземпляров класса управления базой данных.

Сервер базы данных может находиться как в локальной, так и в глобальной сети вместе с сервером приложения. За сетевое подключение данных серверов друг к другу отвечает метод конфигураций класса UniversityDataBase. Создание, обновление и удаление базы данных происходит через пакет миграций, управляемый данным классом. Сущности базы данных назначаются в соответствии с классами-моделями, хранящимися в пакете Models.

Конкретизировано, атрибуты и сущности базы данных представлены на er-диаграмме (Диаграмма 5).

1. **ER-диаграмма**

****

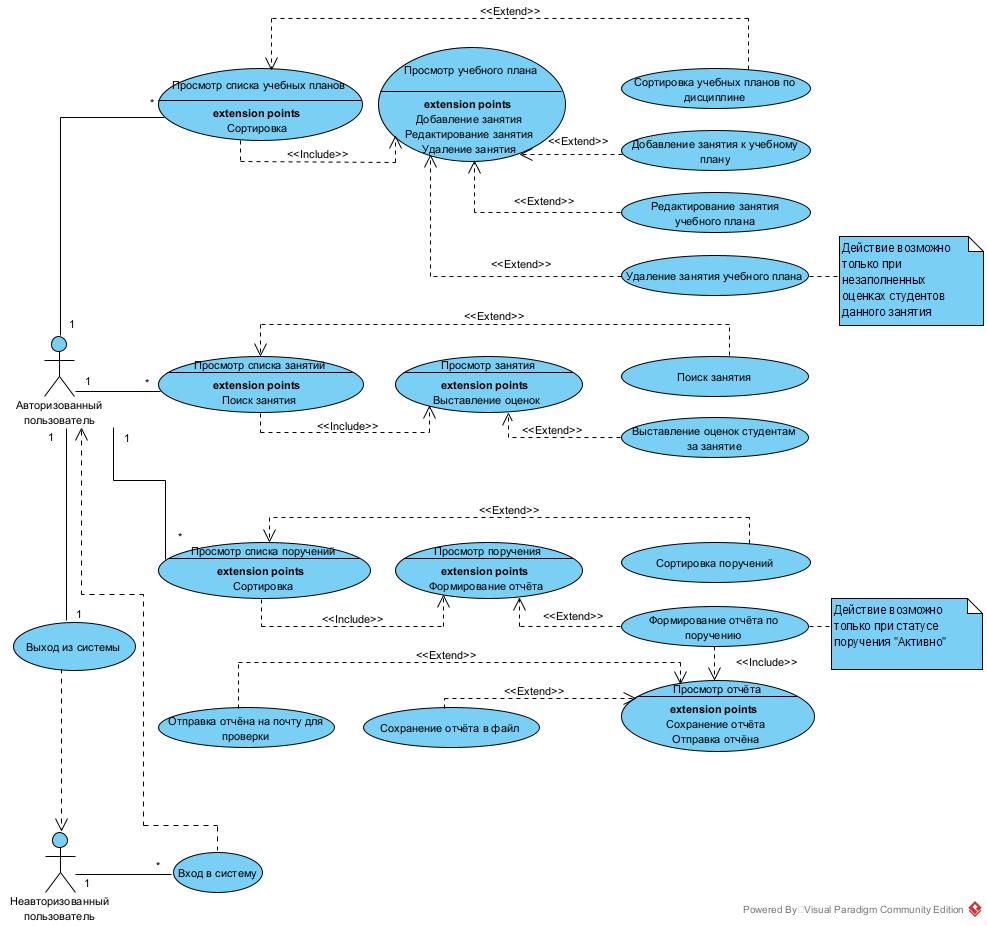
*Диаграмма 5. ER-диаграмма системы.*

На ER-диаграмме отображены сущности базы данных системы и их связи. Каждой сущности соответствует класс-модель на модуле «сервер приложения».

Система имеет два типа пользователей: преподаватель (сущность, отмеченная синим цветом) и администратор (сущность, отмеченная красным цветом). У каждого пользователя имеются свои права доступа и сценарии использования базы данных. Связи, используемые администратором отмечены на диаграмме красным, используемые учителем – синим. Связи, используемые обеими пользователями отмечены чёрным.

1. **Диаграмма вариантов использования**

Система включает в себя web-модуль преподавателя, являющийся пользовательским интерфейсом. Сценарии взаимодействия пользователя с данным модулем представлены на диаграмме (Диаграмма 6).

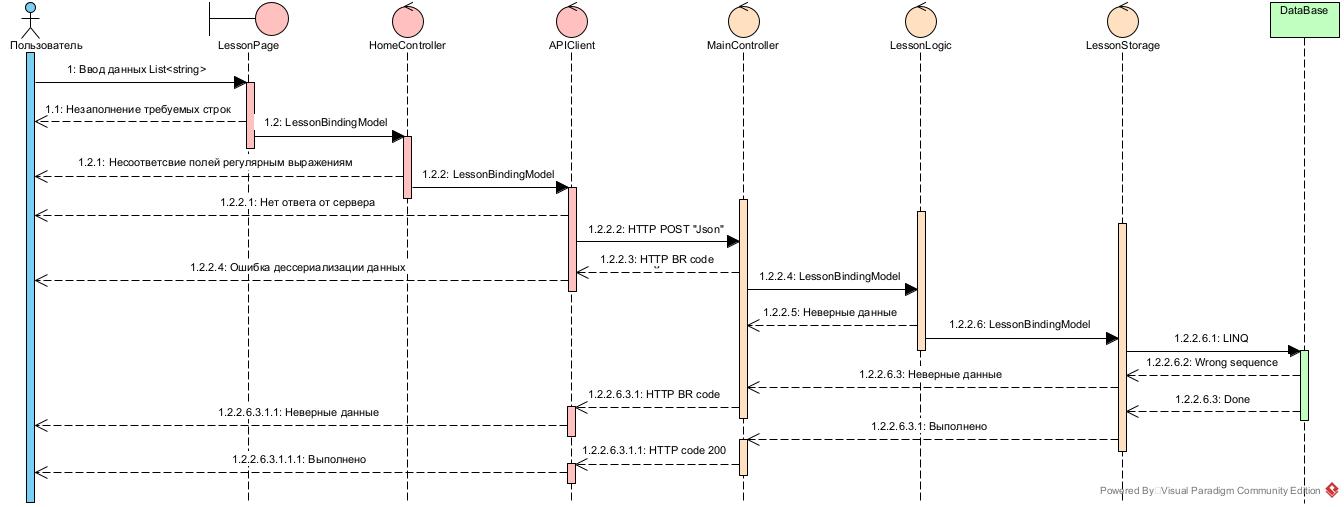


*Диаграмма 6. Диаграмма вариантов использования web-модуля преподавателя.*

Преподаватели, как пользователи системы, регистрируются администрацией образовательного учреждения. Для незарегистрированных пользователей доступ к web-модулю закрыт. У неавторизованного пользователя имеется возможность войти в систему, другие операции недоступны для данной роли. Для авторизованного преподавателя доступны все пользовательские функции модуля.

1. **Диаграмма последовательности**

Валидация данных обозначена как сквозная функциональность системы для обучения. Для конкретного рассмотрения данной функциональности приведена диаграмма последовательности запроса преподавателя (пользователь) на создание нового занятия (Диаграмма 7).



*Диаграмма 7. Диаграмма последовательности для создания нового занятия.*

На диаграмме каждый уровень системы обозначен отдельным цветом: объекты красного цвета – уровень «клиент», оранжевого цвета – уровень «сервер приложения», зелёного цвета – уровень «сервер базы данных».

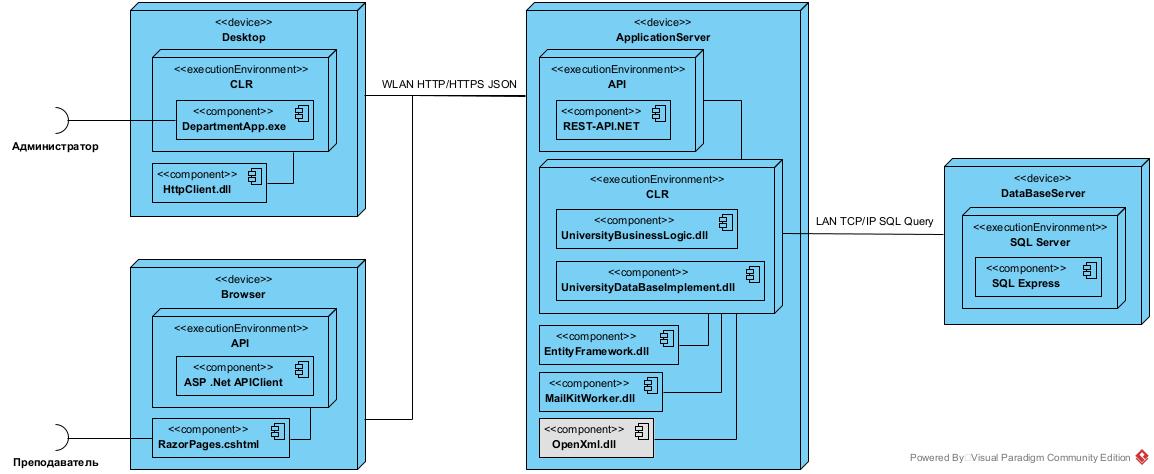
На первом уровне валидация данных происходит с помощью регулярных выражений и анализа визуальных компонент ввода данных. На этом уровне отсекаются незаполненные поля и неверные типы данных.

На втором уровне валидация данных происходит посредством десериализации из Json. На данном уровне отсекаются данные неверного типа и данные, не имеющие значений.

На третьем уровне валидация происходит путём проверки LINQ запроса и сравнение его возможного результата с архитектурой базы данных. На данном уровне отсекаются данные, не имеющие необходимой структуры или необходимой логической связи с другими данными.

Функциональность, описанная на каждом уровне и придерживающаяся одной и той же цели считается сквозной.

1. **Диаграмма развёртывания**



*Диаграмма 8. Диаграмма развёртывания системы.*

Система для обучения интегрируется непосредственно в образовательное учреждения. Существует несколько различных способов её интеграции.

В первом случае, сервер приложения и сервер базы данных располагается на доступном предлагаемом хосте, а организации передаётся только управление и доступ к клиентским приложениям.

Во втором способе, сервер приложения или сервер базы данных располагается на машине самой организации-заказчика. Для этого, между серверами базы данных и приложения также предусмотрено WLAN подключение.

Последним способом является развёртывание всей системы на машинах, предоставленных образовательной организацией.

Все устанавливаемые вместе с системой компоненты имеют гарантии долгосрочной поддержки. Компонент формирования отчётов в виде текстовых документов (в диаграмме отмечен серым цветом) может меняться в зависимости от доступности использования компонент компании Microsoft на территории Российской Федерации или по пожеланию заказчика.

**Контрольный пример реализации проектного решения**

1. **Описание технологий разработки**

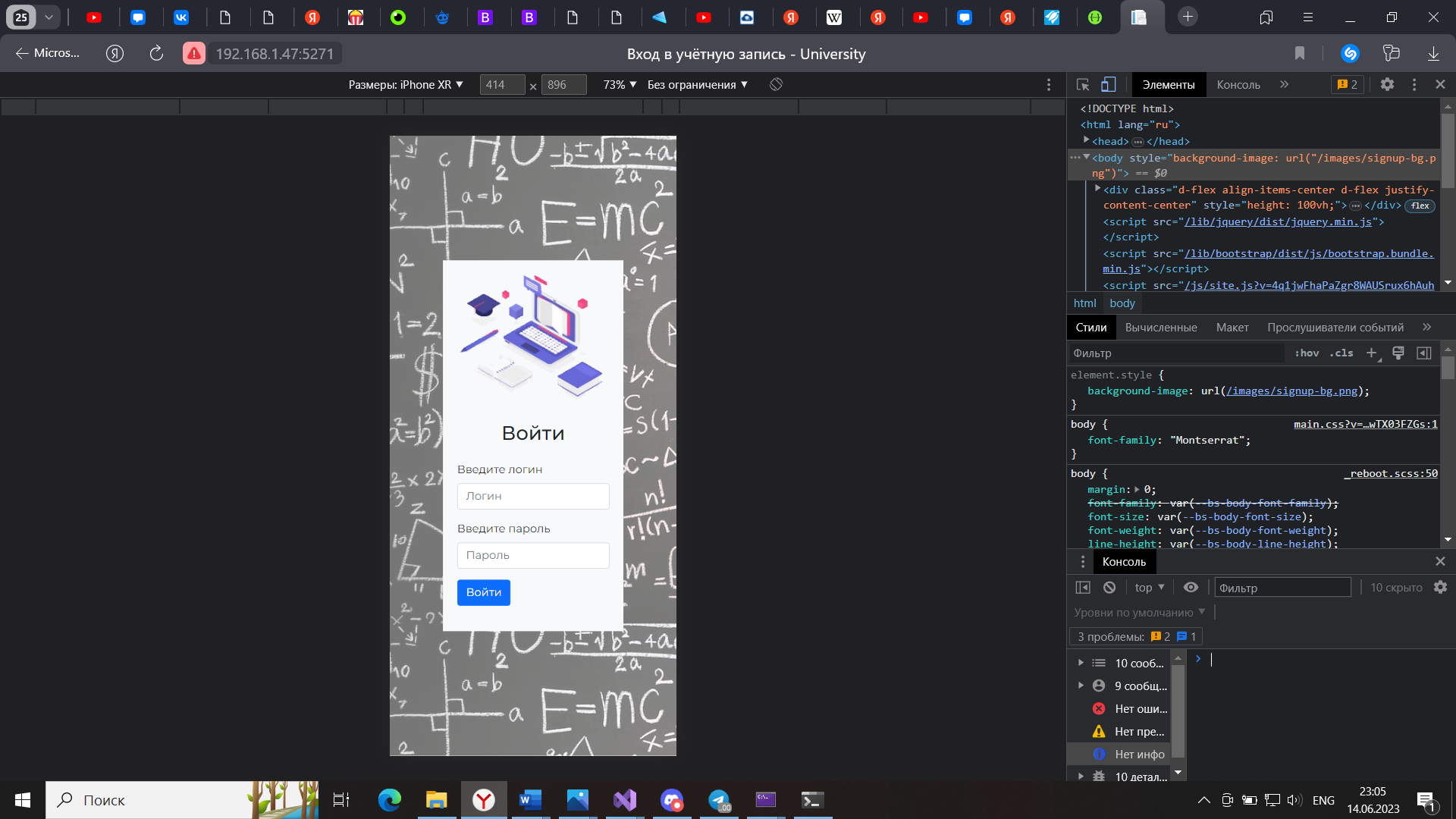
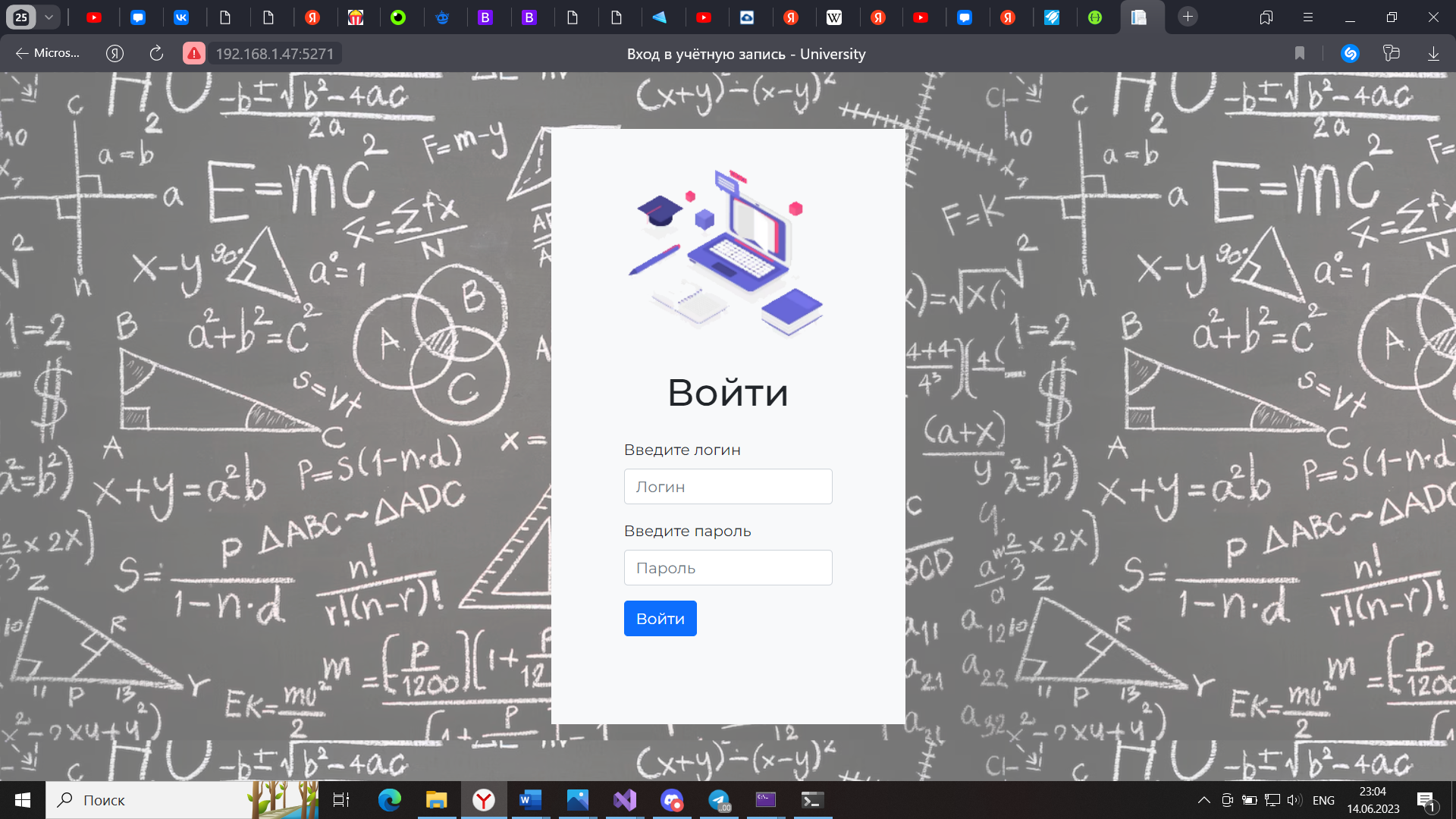
Система для обучения была разработана на языке C# платформы .NET Core версии 6.0. На данный момент эта версия определяется компанией Microsoft как версия с долгосрочной поддержкой.

Клиенты системы используют реализуют фреймворк ASP .NET Core MVC, который позволяет API приложениям связываться по HTTP/HTTPS протоколам и осуществлять передачу объектов в формате Json. При разработке Web-модуля преподавателя использовалась новая технология .NET Razor Pages, которая позволяет использовать код контроллера и отображение страницы в связке, что обеспечивает быстрый отклик пользовательского интерфейса и позволяет использовать динамический интерфейс страниц без применения сторонних языков, таких как JavaScript, jQuery или AJAX.

Сервер приложения реализует фреймворк ASP .NET Core MVC, конкретно технологию REST API. Использование данной технологии упрощает постобработку запросов, используя только GET и POST аннотации. Библиотеки классов логики и хранилищ реализованы технологиями объектно-ориентированного программирования с использованием промежуточных интерфейсов к каждому классу, что позволяет не обращаться к библиотекам напрямую, а использовать их функционал только после проверки методов и их аргументов. Подключение к серверу базы данных реализовано через .NET Entity Framework, позволяющего использовать LINQ запросы и динамические протоколы подключения. Таким образом, при переходе с локального сервера базы данных на удалённый, достаточно редактирования файла конфигураций.

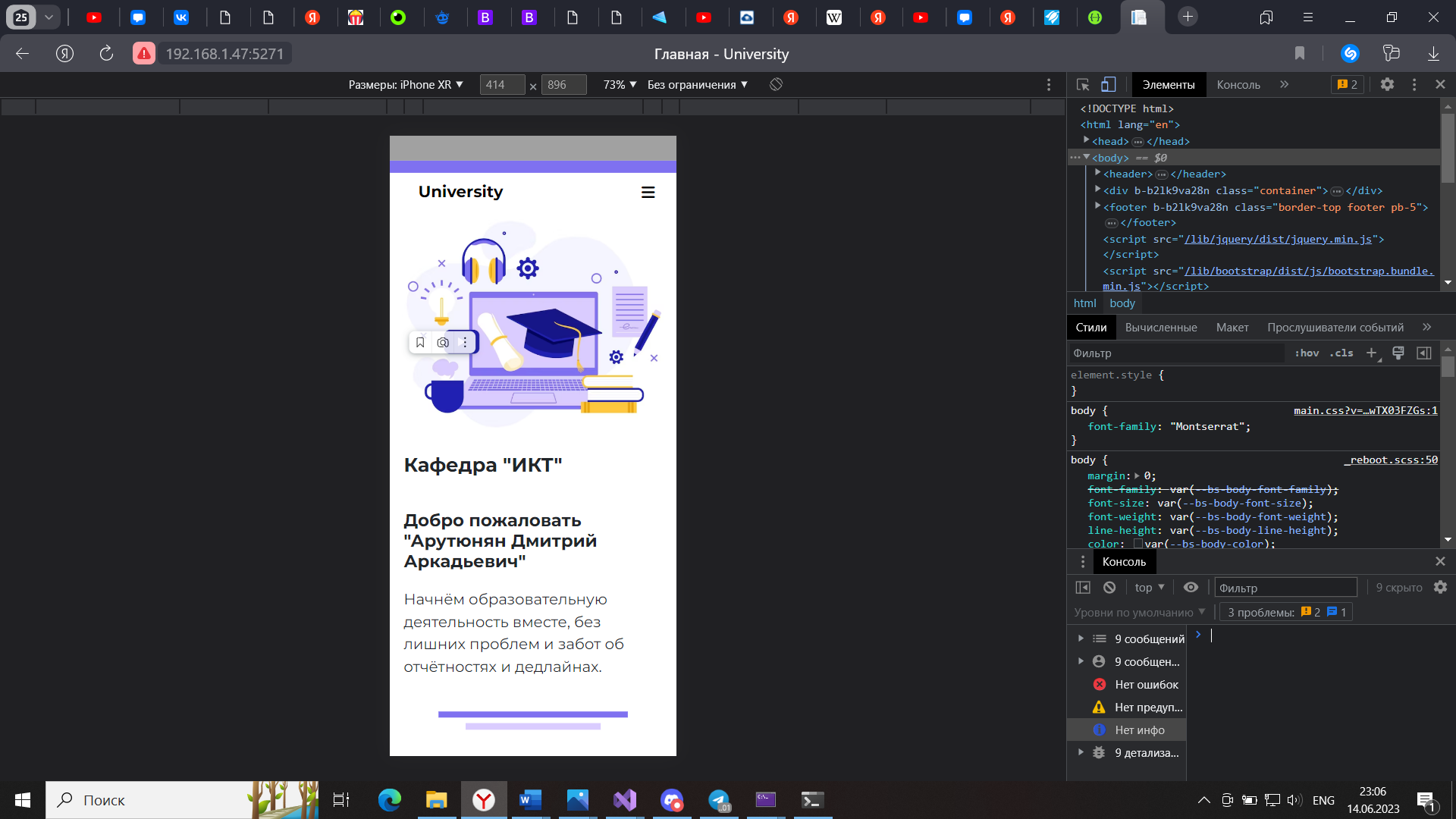
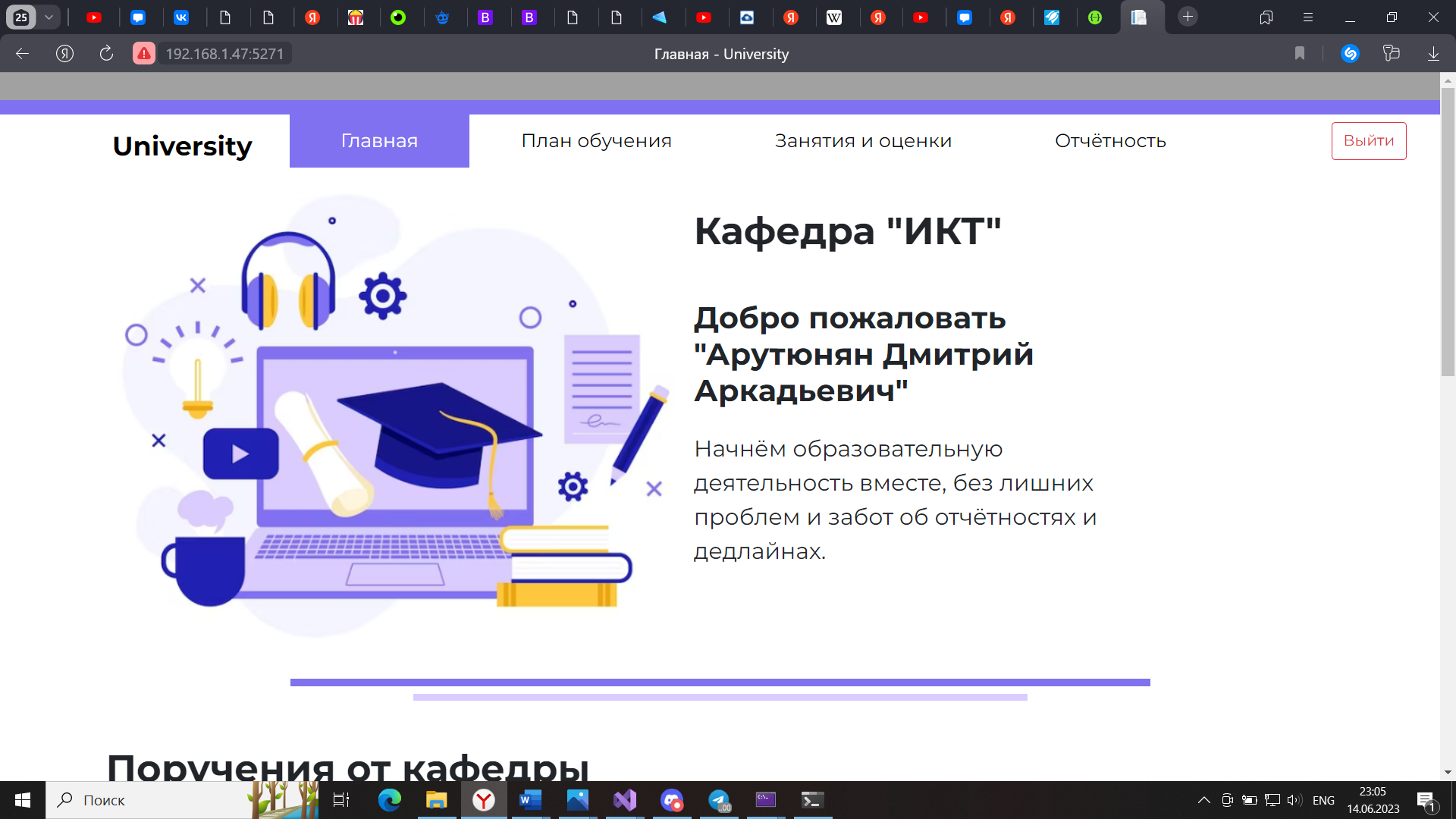
Сервер базы данных представляет собой Microsoft SQL Server 2019 с СУБД SQL Express 2019. Из линейки Microsoft SQL данная СУБД отличается возможностью масштабирования и бесплатности решения. Также SQL Express поддерживает функцию резервного онлайн копирования данных.

1. **Описание основных экранов web-модуля преподавателя**

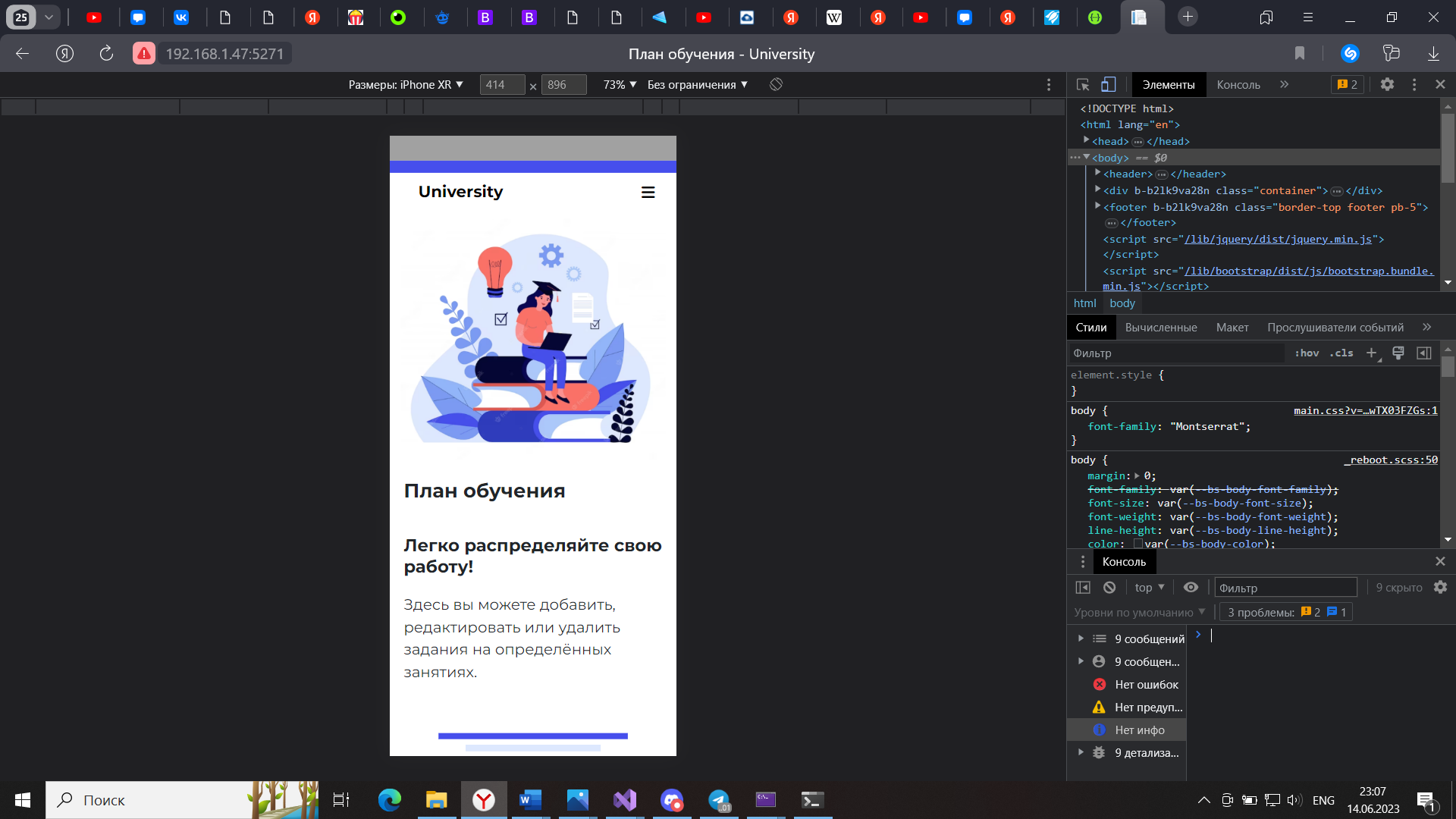
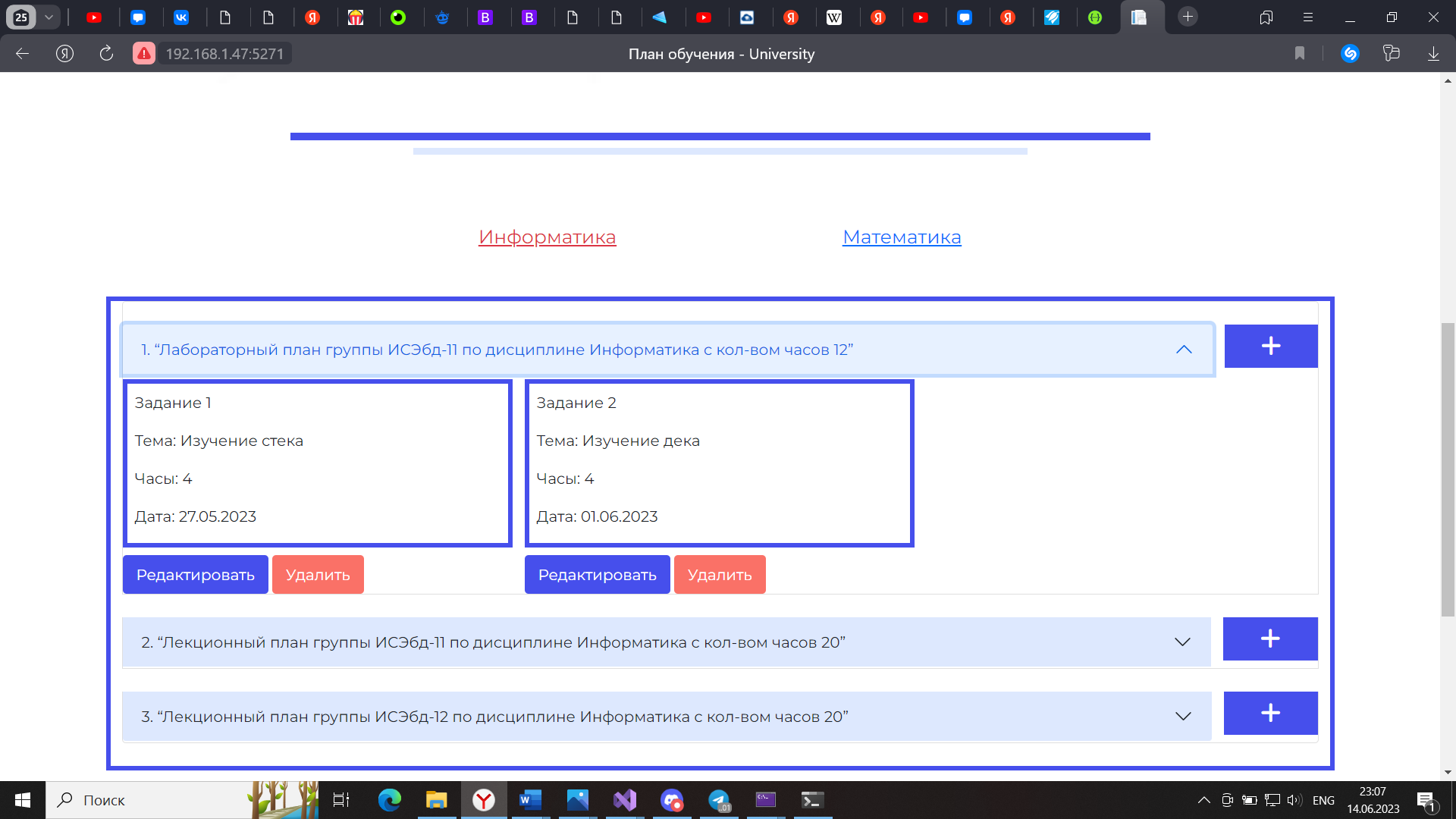
*Рис. 6-7. Страница авторизации web-модуля преподавателя (мобильная версия, ПК версия).*

На странице авторизации преподаватель осуществляет вход в систему по заданному администрацией логину и паролю. Ошибки валидации данных динамически выводятся на верхнюю часть формы. Ошибки подключения и передачи данных выводятся alert уведомлениями на странице.

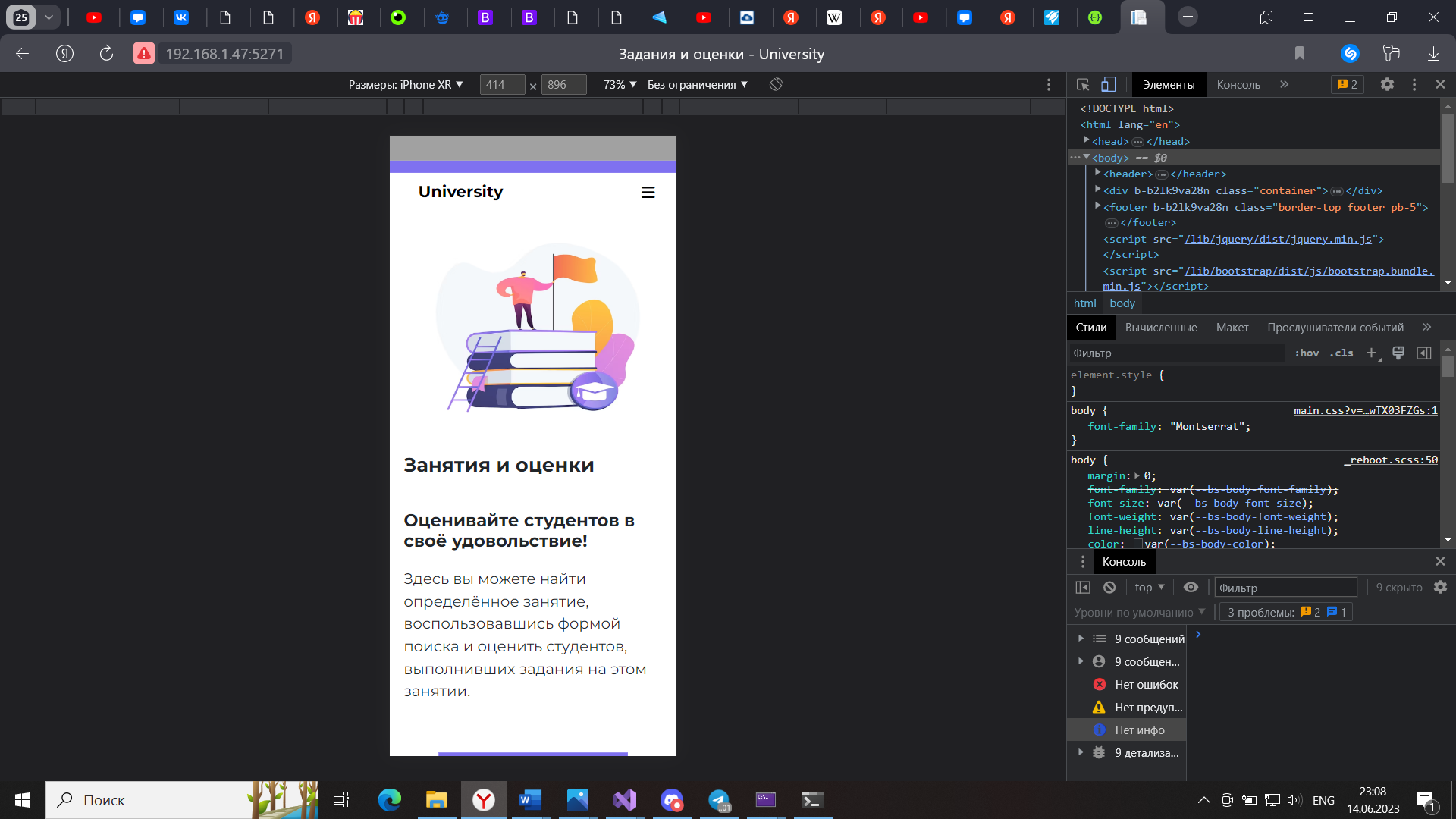
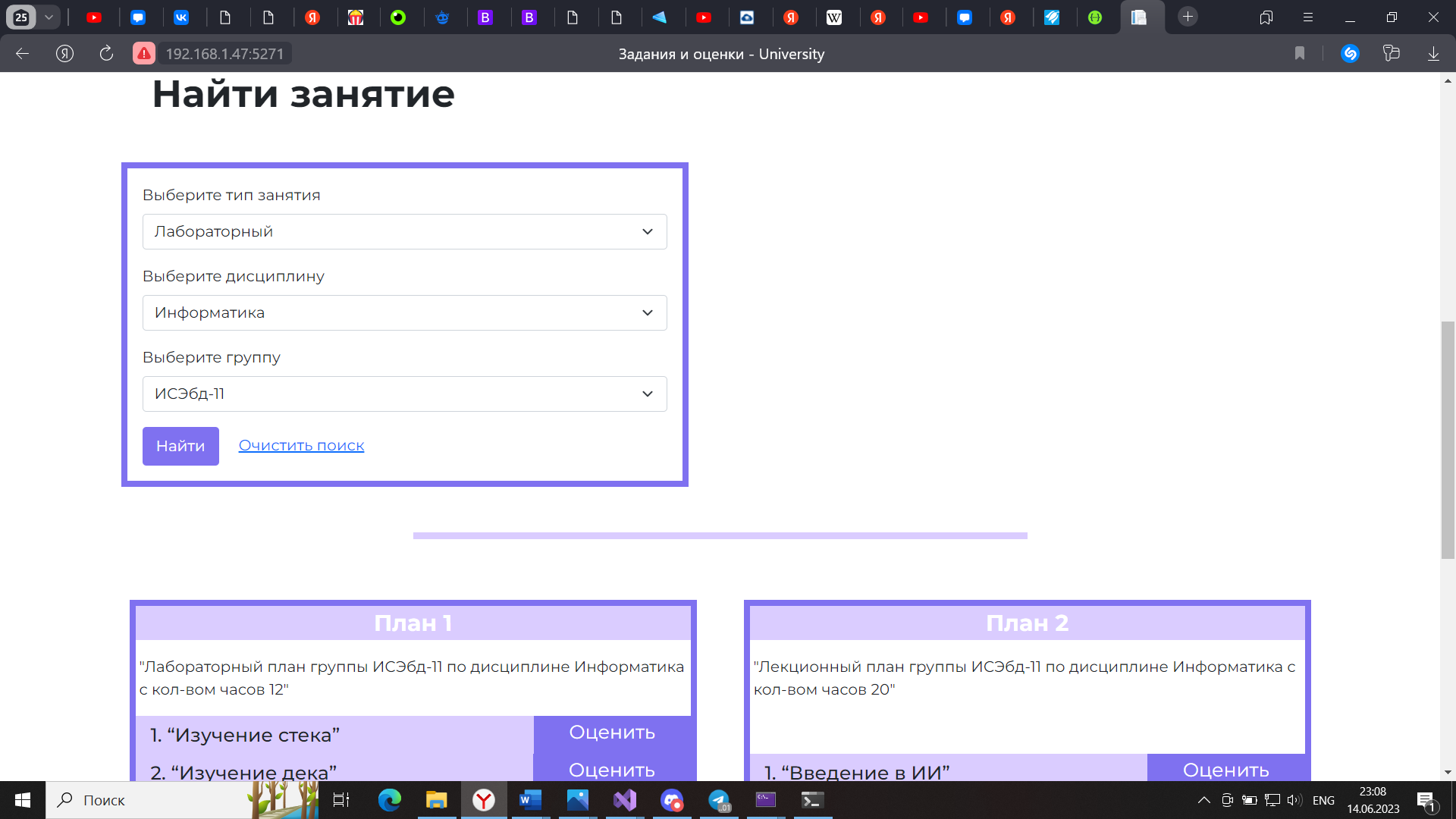
*Рис. 8-9. Главная страница web-модуля преподавателя (мобильная версия, ПК версия).*

На главной странице преподавателю доступно меню навигации по приложению. Также на главной странице отображается ФИО преподавателя, название кафедры, занятия, назначенные преподавателем на текущую дату и активные поручения преподавателю от модуля администратора.

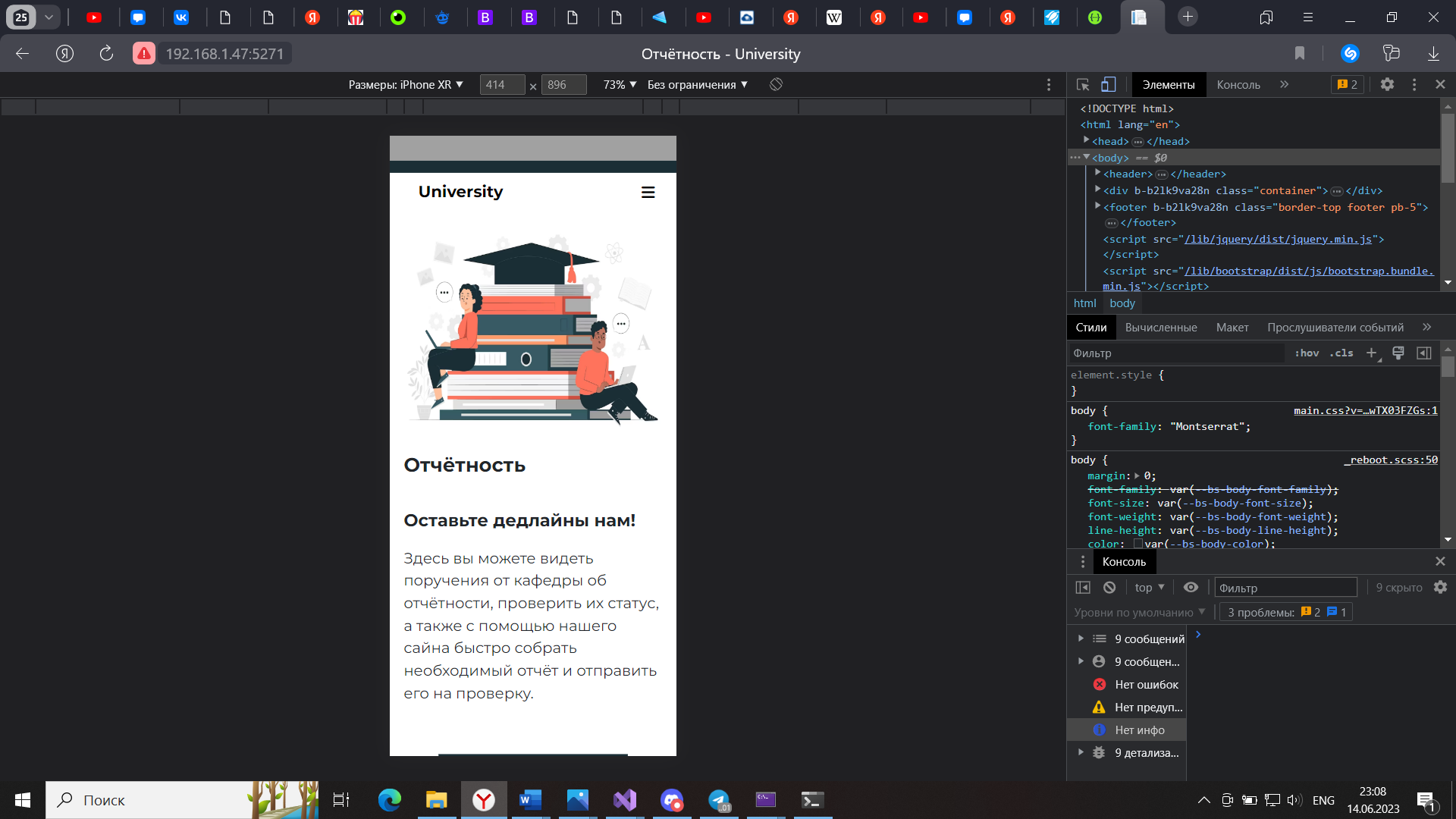
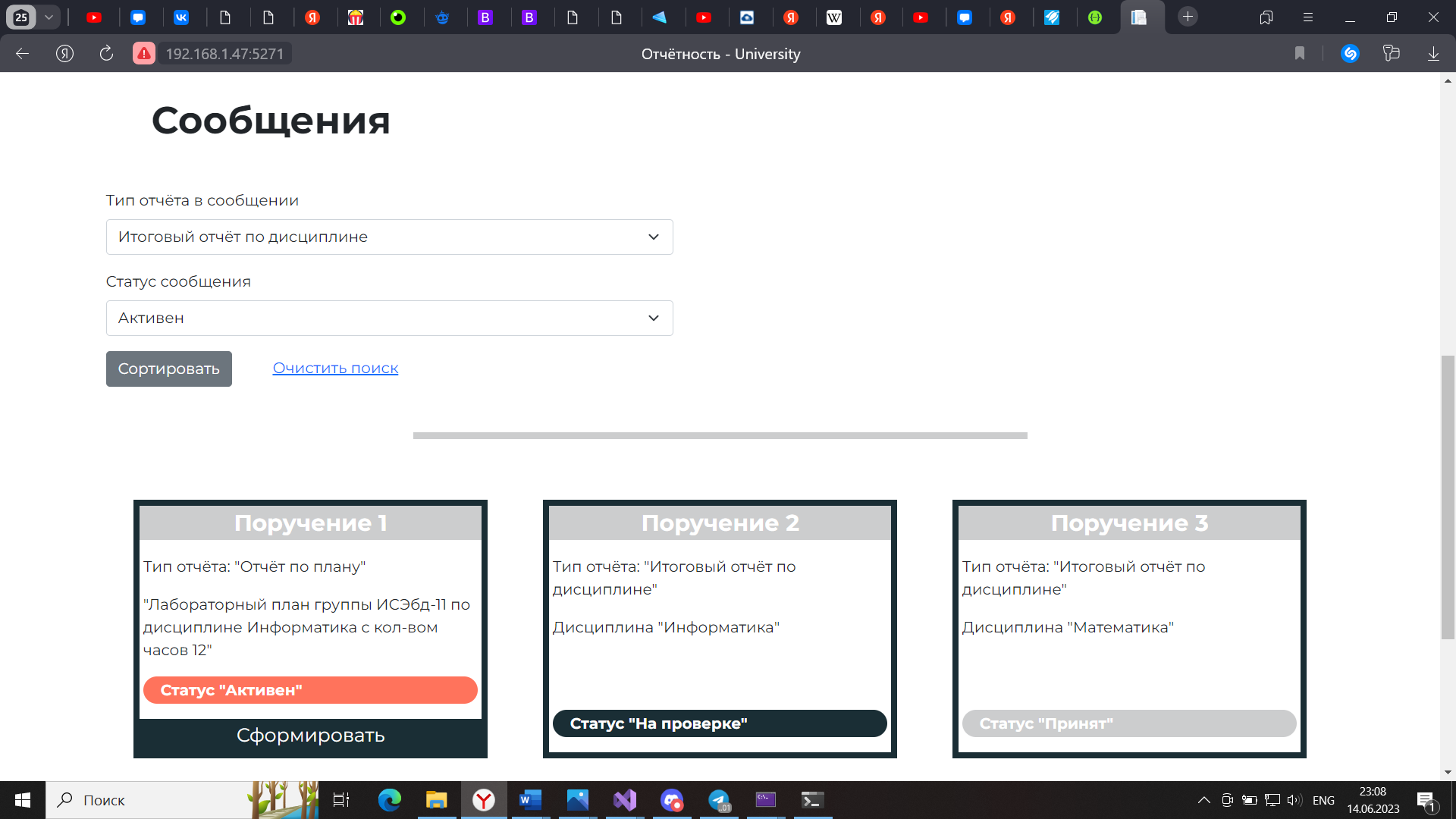
*Рис. 10-11. Страница планов обучения web-модуля преподавателя (мобильная версия, ПК версия).*

На странице планов обучения преподавателю доступен просмотр назначенных ему планов обучения по разным дисциплинам. Для каждого плана преподаватель может добавить занятия, назначив их дату, тему и количество часов. В дальнейшем, занятия могут быть редактированы или удалены из учебного плана.

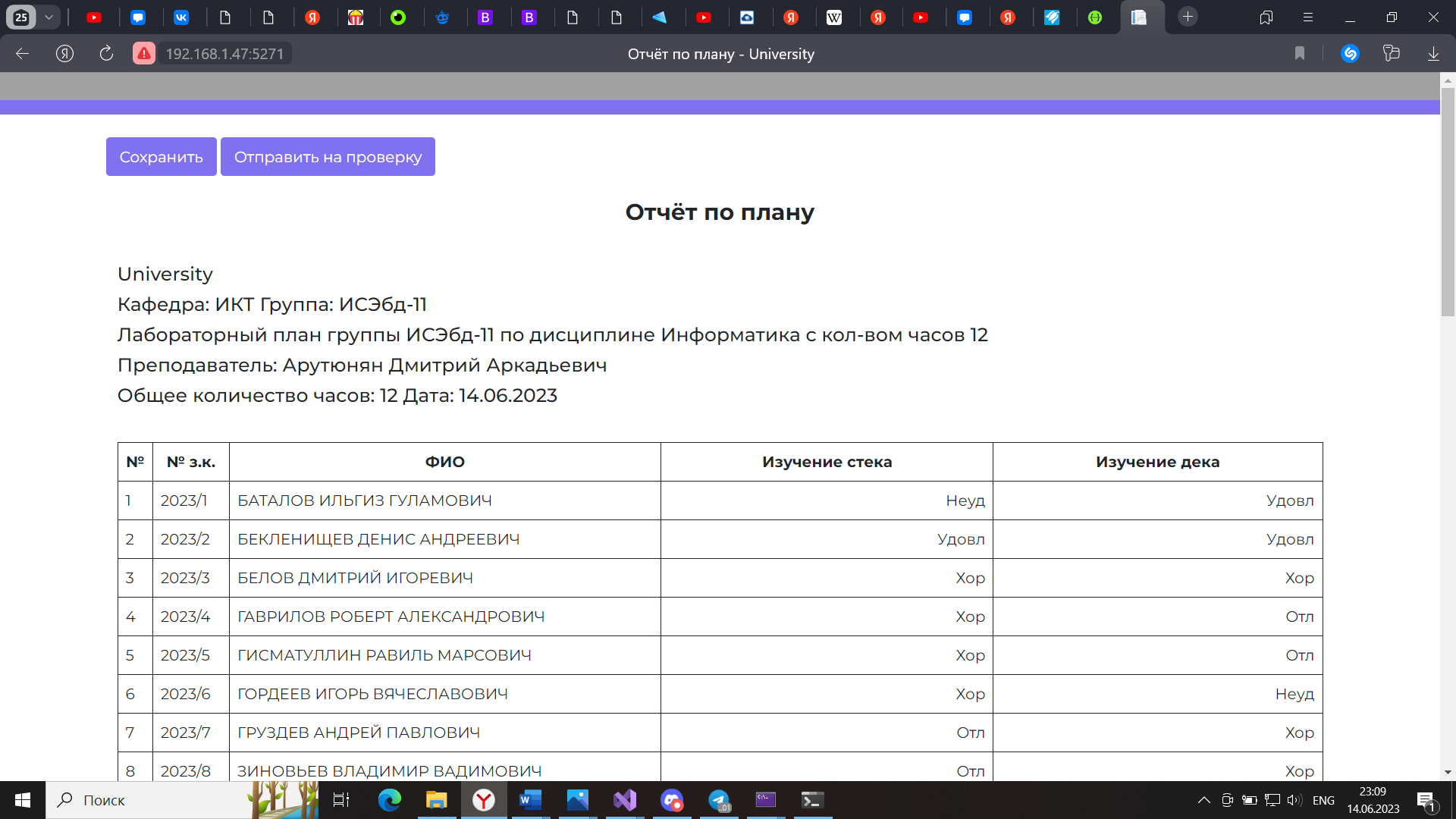
*Рис. 12-13. Страница занятий web-модуля преподавателя (мобильная версия, ПК версия).*

На странице занятий преподаватель может найти конкретное занятие используя форму поиска. Также для каждого занятия преподаватель имеет возможность выставления оценок студентам по кнопке «Оценить». Занятия отображаются на странице по 6 штук с использованием пагинации.

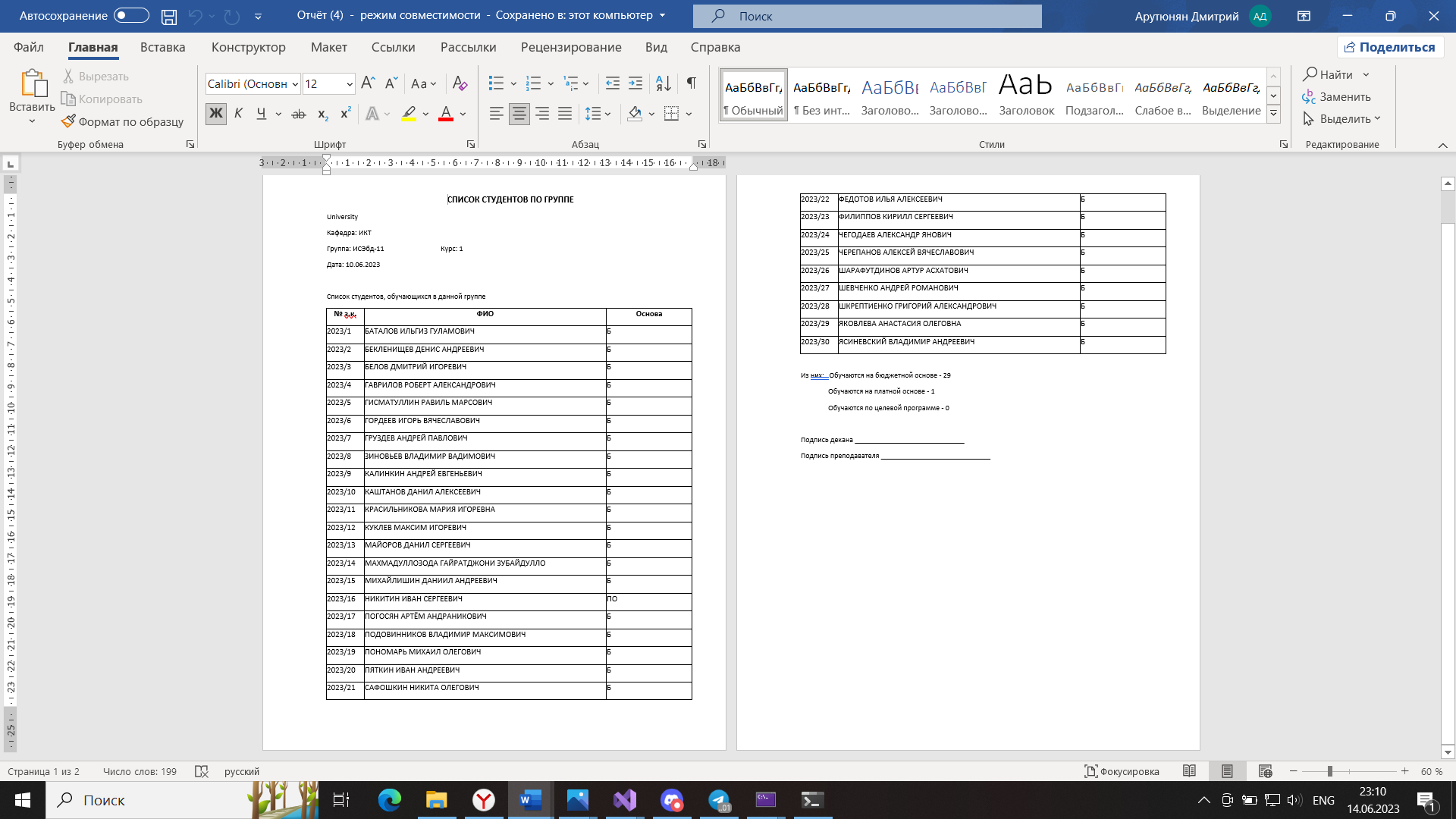
*Рис. 14-15. Страница отчётности web-модуля преподавателя (мобильная версия, ПК версия).*

На странице отчётности преподаватель может отсортировать поручения кафедры по типу отчёта и статусу поручения. Поручения выводятся на странице по 6 штук с использованием пагинации. У активных поручений преподаватель может сгенерировать отчёт по кнопке «Сформировать».

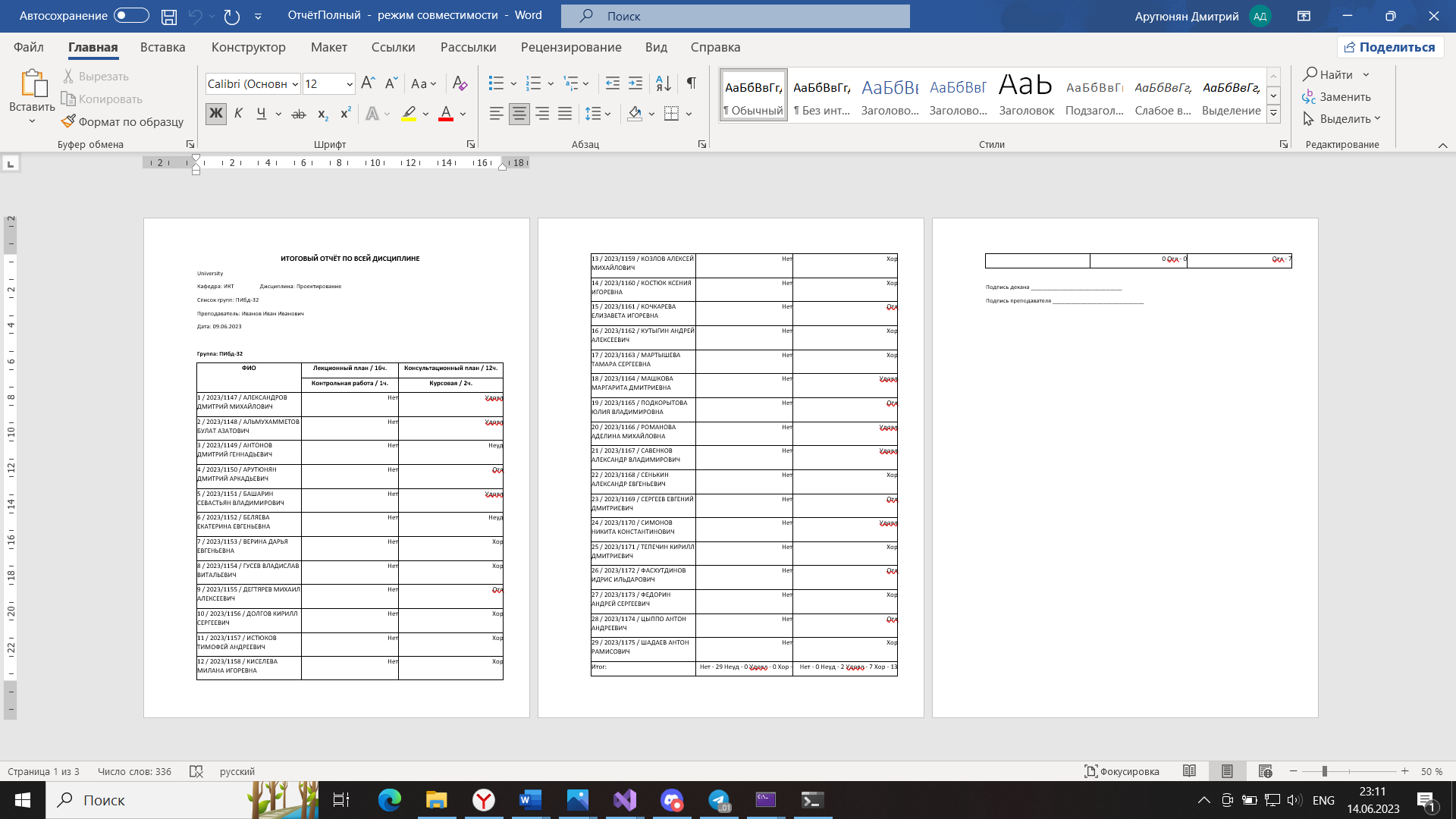


*Рис. 16. Страница сформированного отчёта web-модуля преподавателя.*

На странице конкретного отчёта преподаватель может просмотреть сгенерированный отчёт по выбранному поручению. Также по кнопке «Отправить на проверку» преподаватель отправляет отчёт на почту кафедры. По кнопке «Сохранить» преподаватель скачивает сгенерированный документ по сформированному отчёту. Сгенерированные документы отчётов формата .docx представлены на рисунках 17 и 18.



*Рис. 17. Отчёт по успеваемости студентов в разрезе конкретного плана обучения, сгенерированный в документ Microsoft Office Word.*



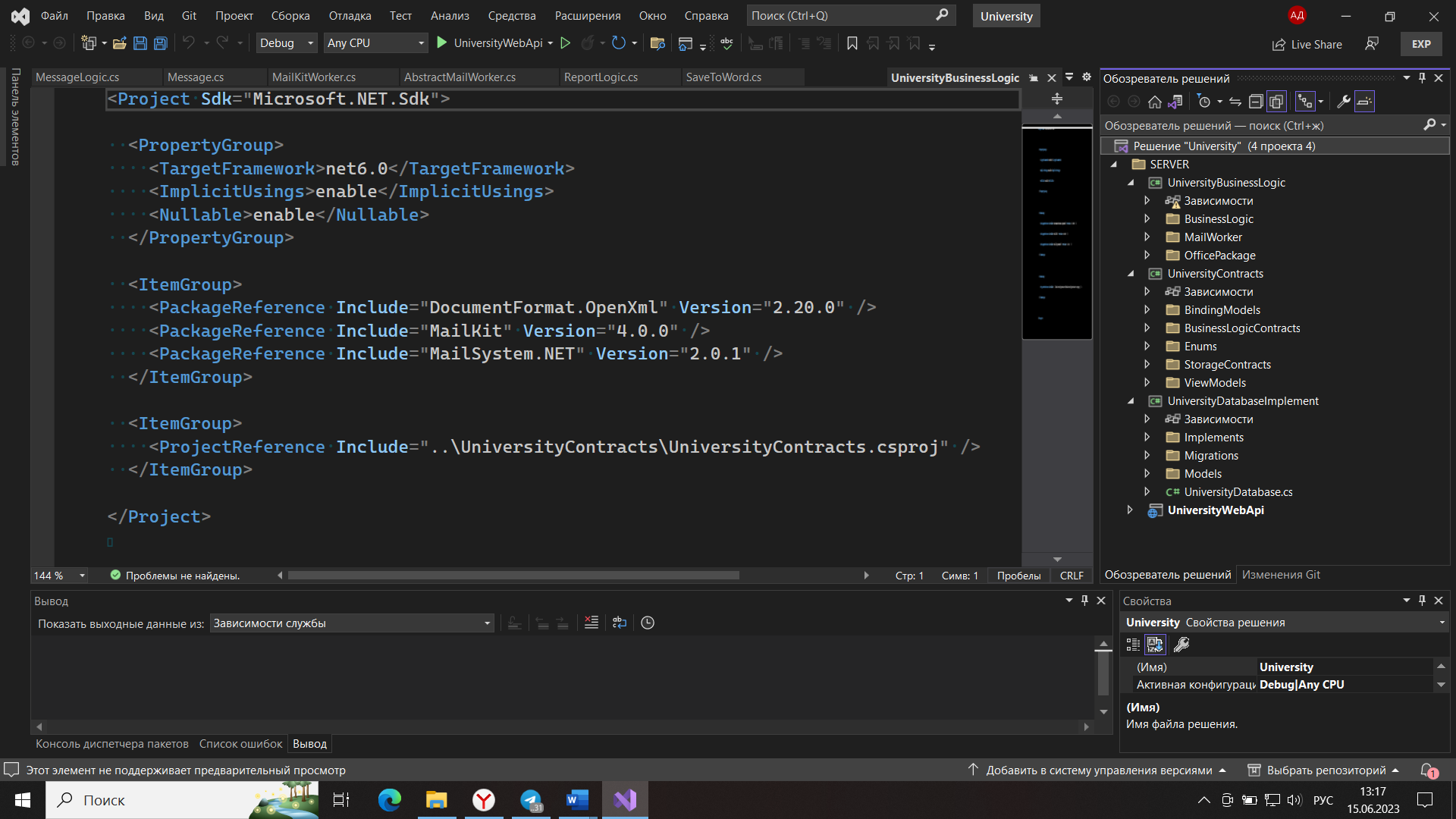
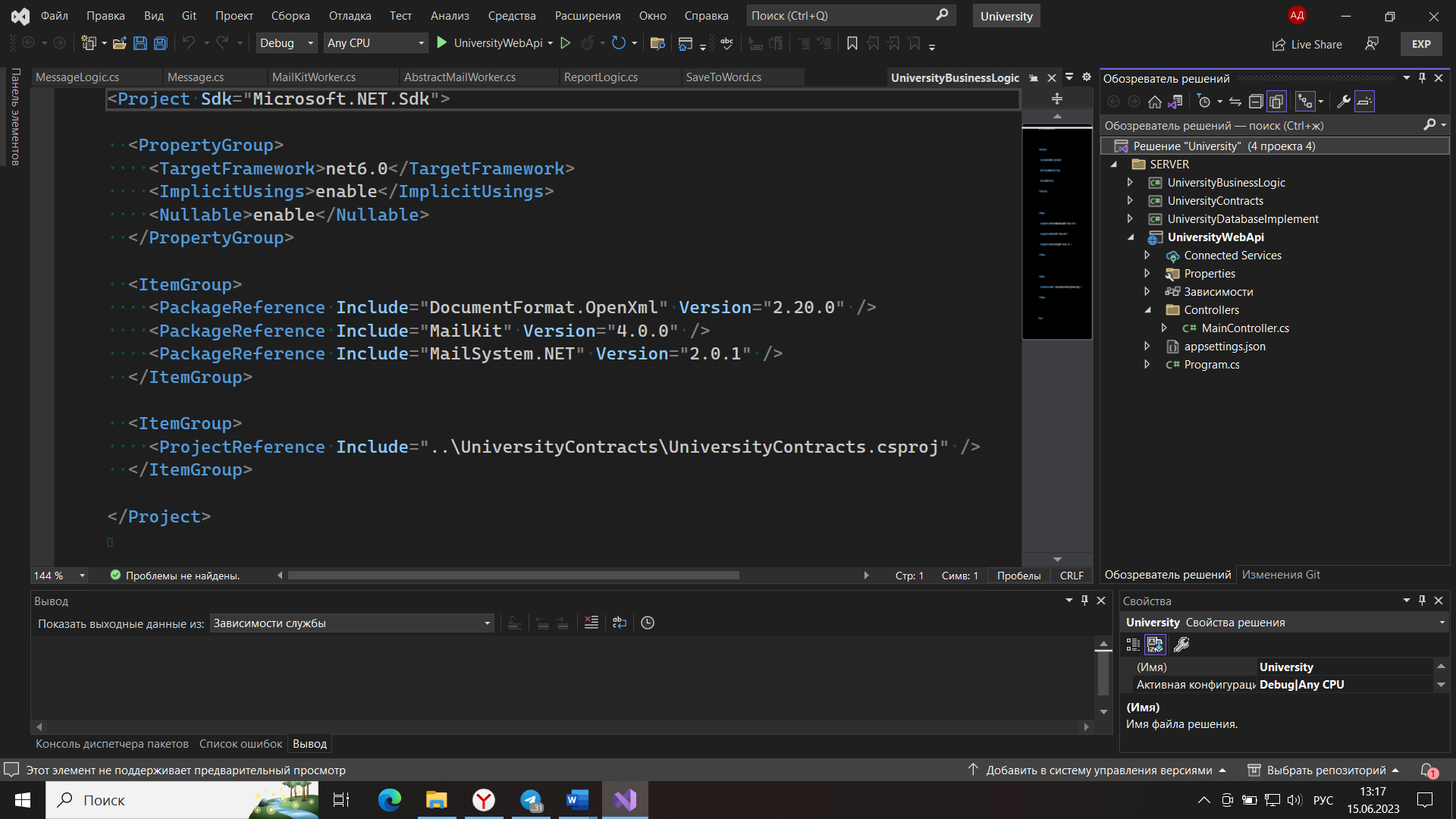
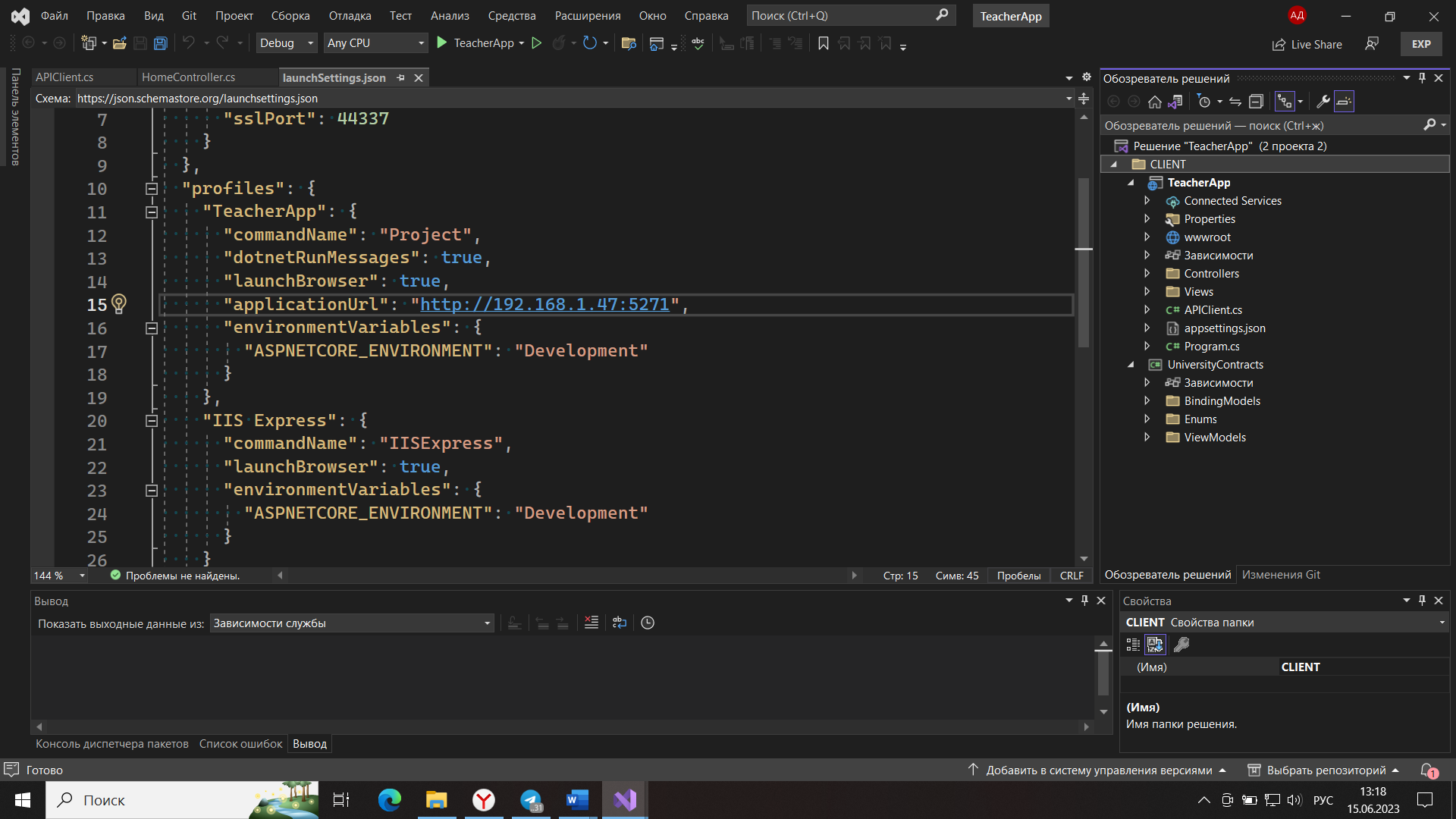
*Рис. 18. Отчёт по успеваемости студентов в разрезе дисциплины, сгенерированный в документ Microsoft Office Word.*

1. **Сгенерированные классы по коду**

Классы логики и хранилищ сервера приложения размещены в библиотеках классов UniversityBusinessLogic и UniversityDataBaseImplement. Управляющий взаимодействием с базой данных класс UniversityDataBase реализует паттерн Singleton и находится в библиотеке классов хранилищ. Связующие модели, вспомогательные классы перечислений, интерфейсы логики и хранилищ находятся в библиотеке классов UniversityContracts, в соответсвии со стандратиами ООП (Рисунок 19).

Приложение UniversityWebApi реализует REST API интерфейс с классом описания запросов MainController (Рисунок 20).

Web-модуль преподавателя TeacherApp содержит в себе класс управления запросами к серверу APIClient. Объекты страниц хранятся в пакете Views. Библиотека классов UniversityContracts содержит связующие модели сущностей, необходимые для сериализации / десериализации и валидации (Рисунок 21).

*Рис. 19-21. Сгенерированные классы системы обучения по коду.*

1. **Выводы по реализации приложения**

Реализованное приложение отвечает всем обозначенным атрибутом качества системы. В следствие использования платформ с долгосрочной поддержкой, приложение стабильно работает как на малоэффективных машинах с установленными предыдущими версиями браузеров, так и на современных системах с последними версиями браузеров.

Работа системы тестировалась на двух машинах различной мощности во всех возможных вариациях системы. Тестирование не выявило ошибок и исключений, связанных с неверной конфигурацией окружающей среды, а также показало сравнительно одинаковые значения по ресурс-затратности оперативной и системной памяти, процессора и сетевой карты. Данные характеристики на машинах среднего уровня показывают результаты с запасом ресурсов в 70% от общего их числа, что соответствует возможному приросту пользователей на 200-250 человек без потери производительности.

**Заключение**

Результатом данного курсового проекта является разработанная система для обучения с web-модулем преподавателя.

В ходе работы над курсовым проектом были использованы методы анализа требований пользователя, изучены методы выбора, оценки и проектирования архитектуры программного обеспечения, использованы современные технологии программирования, изучены методы тестирования и отладки программ.

Разработанная система для обучения с использованием web-модуля преподавателя имеет большой потенциал для применения в образовательных учреждениях, что позволяет значительно улучшить эффективность обучения и осуществлять контроль за успеваемостью студентов. В результате получается более интерактивный и удобный процесс обучения, который позволяет студентам получать знания в более удобном формате и доступе к дополнительным материалам.

**Список литературы**

1. 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.
2. Теория и практика UML. Диаграмма последовательности [Электронный ресурс] / Режим доступа: http://itgost.ru/articles/view\_articles/94
3. 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>.
4. Хамадеев Р.И. Проектирование интернет-ресурсов для дистанционного обучения. // Электронное обучение: реалии и перспективы. Материалы Всероссийской научно-практической конференции. - М.: ИГУМО-СПБ, 2015.
5. Богданов А.В., Кузнецов В.А. Проектирование систем электронного обучения на основе технологий облачных вычислений. // Информатика и программирование. - 2018. - Т. 45, № 6. - С. 53-64.
6. Береснева И.М. Проектирование Web-приложений на основе микро-сервисной архитектуры для обучения. // Цифровая образовательная среда. Сборник научных трудов. - М.: НИУ ВШЭ, 2020.
7. Лядов А.И. Web-ориентированное обучение: механизмы, технологии, методология. // Открытое образование. - 2018. - Т. 26, № 5. - С. 48-57.
8. ГОСТ 34.602-89. Информационная технология. Техническое задание на создание автоматизированной системы. – Москва: Издательство стандартов, 1989. – 11 с.
9. ГОСТ Р 6.30–2003. Требования к оформлению документов. – Взамен ГОСТ 6.30–97 : введ. 2003–07–01. – М. : Изд-во стандартов, 2003. – 16 с.
10. Мейер, Б. Основы объектно-ориентированного проектирования [Электронный ресурс] : учебник / Б. Мейер. — Электрон. дан. — Москва : , 2016. — 765 с. — Режим доступа: <https://e.lanbook.com/book/100305>. — Загл. с экрана.

**Приложение**

1. **Листинг кода**

Библиотека классов UniversityBusinessLogic

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** MessageLogic : IMessageLogic

{

**private** **readonly** IMessageStorage \_messageStorage;

**public** MessageLogic(IMessageStorage messageStorage)

{

\_messageStorage = messageStorage;

}

**public** List<MessageViewModel> Read(MessageBindingModel model)

{

**if** (model == **null**)

{

**return** \_messageStorage.GetFullList();

}

**if** (model.Id.HasValue)

{

**return** new List<MessageViewModel> { \_messageStorage.GetElement(model) };

}

**return** \_messageStorage.GetFilteredList(model);

}

**public** **void** CreateMessage(MessageBindingModel model)

{

model.Status = Status.Активен;

\_messageStorage.Insert(model);

}

**public** **void** AnswerRequest(MessageBindingModel model)

{

**var** message = \_messageStorage.GetElement(new MessageBindingModel { Id = model.Id });

**if** (message == **null**)

{

**throw** new Exception("Сообщение не найдено");

}

message.Status = Status.Проверяется;

\_messageStorage.Update(new MessageBindingModel

{

Id = message.Id,

DepartmentId = message.DepartmentId,

TeacherId = message.TeacherId,

PlanId = message.PlanId,

DisciplineId = message.DisciplineId,

ReportType = message.ReportType,

Status = message.Status

});

}

**public** **void** CloseRequest(MessageBindingModel model)

{

**var** message = \_messageStorage.GetElement(new MessageBindingModel { Id = model.Id });

**if** (message == **null**)

{

**throw** new Exception("Сообщение не найдено");

}

message.Status = Status.Проверен;

\_messageStorage.Update(new MessageBindingModel

{

Id = message.Id,

DepartmentId = message.DepartmentId,

TeacherId = message.TeacherId,

PlanId = message.PlanId,

DisciplineId = message.DisciplineId,

ReportType = message.ReportType,

Status = message.Status

});

}

**public** **void** MessageRollBack(MessageBindingModel model)

{

**var** message = \_messageStorage.GetElement(new MessageBindingModel { Id = model.Id });

**if** (message == **null**)

{

**throw** new Exception("Сообщение не найдено");

}

message.Status = Status.Активен;

\_messageStorage.Update(new MessageBindingModel

{

Id = message.Id,

DepartmentId = message.DepartmentId,

TeacherId = message.TeacherId,

PlanId = message.PlanId,

DisciplineId = message.DisciplineId,

ReportType = message.ReportType,

Status = message.Status

});

}

**public** **void** Delete(MessageBindingModel model)

{

**var** message = \_messageStorage.GetElement(new MessageBindingModel { Id = model.Id });

**if** (message == **null**)

{

**throw** new Exception("Сообщение не найдено");

}

**if** (message.Status == Status.Проверяется)

{

**throw** new Exception("Статус сообщения **\"**проверяется**\"**");

}

\_messageStorage.Delete(new MessageBindingModel { Id = model.Id });

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** TestingLogic : ITestingLogic

{

**private** **readonly** ITestingStorage \_testingStorage;

**private** **readonly** IStudentStorage \_studentStorage;

**private** **readonly** IPlanStorage \_planStorage;

**public** TestingLogic(ITestingStorage testingStorage, IStudentStorage studentStorage,

IPlanStorage planStorage)

{

\_testingStorage = testingStorage;

\_studentStorage = studentStorage;

\_planStorage = planStorage;

}

**public** List<TestingViewModel> Read(TestingBindingModel model)

{

**if** (model == **null**)

{

**return** \_testingStorage.GetFullList();

}

**if** (model.Id.HasValue || model.Topic != **null**)

{

**return** new List<TestingViewModel> { \_testingStorage.GetElement(model) };

}

**return** \_testingStorage.GetFilteredList(model);

}

**public** **void** CreateOrUpdate(TestingBindingModel model)

{

**var** element = \_testingStorage.GetElement(new TestingBindingModel

{

PlanId = model.PlanId,

Topic = model.Topic,

Type = model.Type,

Hours = model.Hours,

Date = model.Date

});

**var** plan = \_planStorage.GetElement(new PlanBindingModel

{

Id = model.PlanId

});

model.Type = plan.Type;

**if** ((element != **null** && element.Id != model.Id) ||

(element != **null** && element.Topic != model.Topic && element.PlanId == model.PlanId))

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Id.HasValue)

{

\_testingStorage.Update(model);

}

**else**

{

\_testingStorage.Insert(model);

}

}

**public** **void** Delete(TestingBindingModel model)

{

**var** element = \_testingStorage.GetElement(new TestingBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_testingStorage.Delete(model);

}

**public** **void** AddStudent(StudentTestingBindingModel model)

{

**var** testing = \_testingStorage.GetElement(new TestingBindingModel

{

Id = model.TestingId

});

**var** student = \_studentStorage.GetElement(new StudentBindingModel

{

Id = model.StudentId

});

**if** (testing == **null** || student == **null**)

{

**throw** new Exception("Элемент не найден");

}

Dictionary<**int**, MarkType> studentTestings = new Dictionary<**int**, MarkType>();

**foreach** (**var** item **in** testing.StudentTestings)

studentTestings.**Add**(item.Item1, item.Item3);

**if** (studentTestings.ContainsKey(model.StudentId))

{

studentTestings[model.StudentId] = model.Grade;

}

**else**

{

studentTestings.**Add**(model.StudentId, model.Grade);

}

\_testingStorage.Update(new TestingBindingModel

{

Id = testing.Id,

Topic = testing.Topic,

Hours = testing.Hours,

Date = testing.Date,

Type = testing.Type,

StudentTestings = studentTestings

});

}

**public** **void** RemoveStudent(StudentTestingBindingModel model)

{

**var** testing = \_testingStorage.GetElement(new TestingBindingModel

{

Id = model.TestingId

});

**var** student = \_studentStorage.GetElement(new StudentBindingModel

{

Id = model.StudentId

});

**if** (testing == **null** || student == **null**)

{

**throw** new Exception("Элемент не найден");

}

Dictionary<**int**, MarkType> studentTestings = new Dictionary<**int**, MarkType>();

**foreach** (**var** item **in** testing.StudentTestings)

studentTestings.**Add**(item.Item1, item.Item3);

**if** (studentTestings.ContainsKey(model.StudentId))

{

studentTestings.**Remove**(model.StudentId);

}

**else**

{

**throw** new Exception("Запись не найдена");

}

\_testingStorage.Update(new TestingBindingModel

{

Id = testing.Id,

Topic = testing.Topic,

Hours = testing.Hours,

Date = testing.Date,

Type = testing.Type,

StudentTestings = studentTestings

});

}

}

}

﻿**using** System.Text.RegularExpressions;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** DepartmentLogic : IDepartmentLogic

{

**private** **readonly** IDepartmentStorage \_departmentStorage;

**private** **readonly** **int** \_passwordMaxLength = 50;

**private** **readonly** **int** \_passwordMinLength = 10;

**public** DepartmentLogic(IDepartmentStorage departmentStorage)

{

\_departmentStorage = departmentStorage;

}

**public** List<DepartmentViewModel> Read(DepartmentBindingModel model)

{

**if** (model == **null**)

{

**return** \_departmentStorage.GetFullList();

}

**if** (model.Id.HasValue || (model.Login != **null** && model.Password != **null**))

{

**return** new List<DepartmentViewModel> { \_departmentStorage.GetElement(model) };

}

**throw** new Exception("Элемент не найден");

}

**public** DepartmentViewModel? Authorization(**string** login, **string** password)

{

**var** department = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Login = login,

});

**if** (department == **null** || !department.Password.Equals(password))

{

department = **null**;

}

**return** department;

}

**public** **void** CreateOrUpdate(DepartmentBindingModel model)

{

**var** element = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Login = model.Login

});

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (!Regex.IsMatch(model.Login, @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$"))

{

**throw** new Exception("В качестве логина должна быть указана почта");

}

**if** (model.Password.Length > \_passwordMaxLength || model.Password.Length < \_passwordMinLength ||

!Regex.IsMatch(model.Password, @"^((\w+\d+\W+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]\*$"))

{

**throw** new Exception($"Пароль должен быть длиной от {\_passwordMinLength} до " +

$"{\_passwordMaxLength} и состоять из цифр, букв и небуквенных символов");

}

**if** (model.Id.HasValue)

{

\_departmentStorage.Update(model);

}

**else**

{

\_departmentStorage.Insert(model);

}

}

**public** **void** Delete(DepartmentBindingModel model)

{

**var** element = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_departmentStorage.Delete(model);

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** StudentLogic : IStudentLogic

{

**private** **readonly** IStudentStorage \_studentStorage;

**public** StudentLogic(IStudentStorage studentStorage)

{

\_studentStorage = studentStorage;

}

**public** List<StudentViewModel> Read(StudentBindingModel model)

{

**if** (model == **null**)

{

**return** \_studentStorage.GetFullList();

}

**if** (model.Id.HasValue)

{

**return** new List<StudentViewModel> { \_studentStorage.GetElement(model) };

}

**return** \_studentStorage.GetFilteredList(model);

}

**public** **void** CreateOrUpdate(StudentBindingModel model)

{

**var** element = \_studentStorage.GetElement(new StudentBindingModel

{

GroupId = model.GroupId,

Flm = model.Flm

});

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Id.HasValue)

{

\_studentStorage.Update(model);

}

**else**

{

\_studentStorage.Insert(model);

}

}

**public** **void** Delete(StudentBindingModel model)

{

**var** element = \_studentStorage.GetElement(new StudentBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_studentStorage.Delete(model);

}

}

}

﻿**using** System.Text.RegularExpressions;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** TeacherLogic : ITeacherLogic

{

**private** **readonly** ITeacherStorage \_teacherStorage;

**private** **readonly** IDisciplineStorage \_disciplineStorage;

**private** **readonly** **int** \_passwordMaxLength = 50;

**private** **readonly** **int** \_passwordMinLength = 10;

**public** TeacherLogic(ITeacherStorage teacherStorage, IDisciplineStorage disciplineStorage)

{

\_teacherStorage = teacherStorage;

\_disciplineStorage = disciplineStorage;

}

**public** List<TeacherViewModel> Read(TeacherBindingModel model)

{

**if** (model == **null**)

{

**return** \_teacherStorage.GetFullList();

}

**if** (model.Id.HasValue || (model.Login != **null** && model.Password != **null**))

{

**return** new List<TeacherViewModel> { \_teacherStorage.GetElement(model) };

}

**return** \_teacherStorage.GetFilteredList(model);

}

**public** List<TeacherViewModel> GetTeacherByDiscipline(**int** id)

{

**return** \_teacherStorage.GetTeacherByDiscipline(id);

}

**public** **void** CreateOrUpdate(TeacherBindingModel model)

{

**var** element = \_teacherStorage.GetElement(new TeacherBindingModel

{

DepartmentId = model.DepartmentId,

Flm = model.Flm,

Login = model.Login,

Password = model.Password

});

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Password.Length > \_passwordMaxLength || model.Password.Length < \_passwordMinLength ||

!Regex.IsMatch(model.Password, @"^((\w+\d+\W+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]\*$"))

{

**throw** new Exception($"Пароль должен быть длиной от {\_passwordMinLength} до " +

$"{\_passwordMaxLength} и состоять из цифр, букв и небуквенных символов");

}

**if** (model.Id.HasValue)

{

\_teacherStorage.Update(model);

}

**else**

{

\_teacherStorage.Insert(model);

}

}

**public** **void** Delete(TeacherBindingModel model)

{

**var** element = \_teacherStorage.GetElement(new TeacherBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_teacherStorage.Delete(model);

}

**public** **void** AddDiscipline(TeacherDisciplineBindingModel model)

{

**var** teacher = \_teacherStorage.GetElement(new TeacherBindingModel

{

Id = model.TeacherId

});

**var** discipline = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

Id = model.DisciplineId

});

**if** (teacher == **null** || discipline == **null**)

{

**throw** new Exception("Элемент не найден");

}

List<**int**> teacherDisciplines = new List<**int**>();

**foreach** (**int** key **in** teacher.TeacherDisciplines.Keys)

teacherDisciplines.**Add**(key);

**if** (!teacherDisciplines.Contains(model.DisciplineId))

{

teacherDisciplines.**Add**(model.DisciplineId);

}

\_teacherStorage.Update(new TeacherBindingModel

{

Id = teacher.Id,

DepartmentId = teacher.DepartmentId,

Flm = teacher.Flm,

Login = teacher.Login,

Password = teacher.Password,

TeacherDisciplines = teacherDisciplines

});

}

**public** **void** RemoveDiscipline(TeacherDisciplineBindingModel model)

{

**var** teacher = \_teacherStorage.GetElement(new TeacherBindingModel

{

Id = model.TeacherId

});

**var** discipline = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

Id = model.DisciplineId

});

**if** (teacher == **null** || discipline == **null**)

{

**throw** new Exception("Элемент не найден");

}

List<**int**> teacherDisciplines = new List<**int**>();

**foreach** (**int** key **in** teacher.TeacherDisciplines.Keys)

teacherDisciplines.**Add**(key);

**if** (teacherDisciplines.Contains(model.DisciplineId))

{

teacherDisciplines.**Remove**(model.DisciplineId);

}

**else**

{

**throw** new Exception("Запись не найдена");

}

\_teacherStorage.Update(new TeacherBindingModel

{

Id = teacher.Id,

DepartmentId = teacher.DepartmentId,

Flm = teacher.Flm,

Login = teacher.Login,

Password = teacher.Password,

TeacherDisciplines = teacherDisciplines

});

}

}

}

﻿**using** UniversityBusinessLogic.MailWorker;

**using** UniversityBusinessLogic.OfficePackage;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** ReportLogic : IReportLogic

{

**public** IPlanStorage \_planStorage;

**public** ITestingStorage \_testingStorage;

**public** ITeacherStorage \_teacherStorage;

**public** IDisciplineStorage \_disciplineStorage;

**public** IStudentStorage \_studentStorage;

**public** IGroupStorage \_groupStorage;

**public** IDepartmentStorage \_departmentStorage;

**private** **readonly** AbstractSaveToWord \_saveToWord;

**private** **readonly** AbstractMailWorker \_mailWorker;

**public** ReportLogic(IPlanStorage planStorage, ITeacherStorage teacherStorage,

ITestingStorage testingStorage, IDisciplineStorage disciplineStorage,

IStudentStorage studentStorage, IGroupStorage groupStorage, IDepartmentStorage departmentStorage,

AbstractSaveToWord abstractSaveToWord, AbstractMailWorker mailWorker)

{

\_planStorage = planStorage;

\_teacherStorage = teacherStorage;

\_testingStorage = testingStorage;

\_disciplineStorage = disciplineStorage;

\_studentStorage = studentStorage;

\_groupStorage = groupStorage;

\_departmentStorage = departmentStorage;

\_saveToWord = abstractSaveToWord;

\_mailWorker = mailWorker;

}

**public** ReportGroupViewModel GetObjectsForGroupReport(**int** id)

{

List<Tuple<**string**, **string**, TypeEducationBasement>> items = new List<Tuple<**string**, **string**, TypeEducationBasement>>();

GroupViewModel **group** = \_groupStorage.GetElement(new GroupBindingModel

{

Id = id

});

DepartmentViewModel department = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Id = **group**.DepartmentId

});

List<StudentViewModel> students = \_studentStorage.GetFilteredList(new StudentBindingModel

{

GroupId = **group**.Id

});

**int** count0 = 0;

**int** count1 = 0;

**int** count2 = 0;

**foreach** (**var** student **in** students)

{

items.**Add**(new Tuple<**string**, **string**, TypeEducationBasement>(

student.NumFB, student.Flm, student.Basement));

**if** (student.Basement == TypeEducationBasement.Б)

count0++;

**if** (student.Basement == TypeEducationBasement.ПО)

count1++;

**if** (student.Basement == TypeEducationBasement.ЦП)

count2++;

}

List<**string**> footer = new List<**string**>

{

"Из них: Обучаются на бюджетной основе - " + count0,

" Обучаются на платной основе - " + count1,

" Обучаются по целевой программе - " + count2,

"",

"Подпись декана \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

};

List<**string**> upper = new List<**string**>

{

"University",

"Кафедра: " + department.Name,

"Группа: " + **group**.Name + " Курс: " + **group**.Course,

"Дата: " + DateTime.Now.Date.ToShortDateString(),

"",

"Список студентов, обучающихся в данной группе"

};

**return** new ReportGroupViewModel

{

Mail = department.Login,

Title = "Список студентов по группе",

Upper = upper,

Footer = footer,

Items = items

};

}

**public** ReportFullViewModel GetObjectsForSumReport(MessageBindingModel model)

{

List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>>> items = new List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>>>();

List<Tuple<List<**string**>>> Itog = new List<Tuple<List<**string**>>>();

TeacherViewModel teacher = \_teacherStorage.GetElement(new TeacherBindingModel

{

Id = model.TeacherId

});

DepartmentViewModel department = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Id = teacher.DepartmentId

});

DisciplineViewModel discipline = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

Id = model.DisciplineId

});

List<PlanViewModel> plans = \_planStorage.GetFilteredList(new PlanBindingModel

{

TeacherId = (**int**)model.TeacherId,

DisciplineId = (**int**)model.DisciplineId

});

**foreach** (**var** plan **in** plans)

{

**bool** flag = **true**;

**foreach** (**var** item **in** items)

{

**if** (item.Item1 == plan.GroupName)

{

flag = **false**;

}

}

**if** (flag)

{

items.**Add**(new Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>>

(plan.GroupName, new List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>()));

}

}

**foreach** (**var** subitem **in** items)

{

GroupViewModel **group** = \_groupStorage.GetElement(new GroupBindingModel

{

DepartmentId = (**int**)model.DepartmentId,

Name = subitem.Item1

});

List<StudentViewModel> students = \_studentStorage.GetFilteredList(new StudentBindingModel

{

GroupId = **group**.Id

});

**foreach** (**var** student **in** students)

{

subitem.Item2.**Add**(new Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>

(

(subitem.Item2.Count() + 1) + " / " + student.NumFB + " / " + student.Flm, new List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>()

));

**foreach** (**var** subItem2 **in** subitem.Item2)

{

List<PlanViewModel> plns = \_planStorage.GetFilteredList(new PlanBindingModel

{

TeacherId = (**int**)model.TeacherId,

GroupId = **group**.Id,

DisciplineId = (**int**)model.DisciplineId

});

**foreach** (**var** plan **in** plns)

{

**if** (subItem2.Item2.**Where**(rec => rec.Item1 == (plan.Type.ToString() + " план / " + plan.Hours + "ч.")).ToList().Count() == 0)

{

subItem2.Item2.**Add**(new Tuple<**string**, List<Tuple<**string**, MarkType>>>

(

plan.Type.ToString() + " план / " + plan.Hours + "ч.", new List<Tuple<**string**, MarkType>>()

));

**foreach** (**var** subItem3 **in** subItem2.Item2)

{

**if** (subItem3.Item2.Count() == 0)

{

List<TestingViewModel> testings = \_testingStorage.GetFilteredList(new TestingBindingModel

{

PlanId = plan.Id,

});

**for** (**int** i = 0; i < testings.Count; i++)

{

**foreach** (**var** grades **in** testings[i].StudentTestings)

{

**if** (student.Flm == grades.Item2)

{

subItem3.Item2.**Add**(new Tuple<**string**, MarkType>(testings[i].Topic + " / " + testings[i].Hours + "ч.", grades.Item3));

}

}

}

}

}

}

}

}

}

}

**for** (**int** i = 0; i < items.Count(); i++)

{

List<**string**> itog = new List<**string**> { "Итог:" };

**for** (**int** j = 0; j < items[i].Item2[0].Item2.Count(); j++)

{

**for** (**int** k = 0; k < items[i].Item2[0].Item2[j].Item2.Count(); k++)

{

**if** (items[i].Item2[0].Item2[j].Item2[k].Item2 == MarkType.П ||

items[i].Item2[0].Item2[j].Item2[k].Item2 == MarkType.НП ||

items[i].Item2[0].Item2[j].Item2[k].Item2 == MarkType.УП)

{

**int** upCount = 0;

**int** pCount = 0;

**int** npCount = 0;

**foreach** (**var** student **in** items[i].Item2)

{

**if** (student.Item2[j].Item2[k].Item2 == MarkType.УП)

upCount++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.П)

pCount++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.НП)

npCount++;

}

itog.**Add**("УП - " + upCount +

" П - " + pCount +

" НП - " + npCount);

}

**else**

{

**int** Count0 = 0;

**int** Count2 = 0;

**int** Count3 = 0;

**int** Count4 = 0;

**int** Count5 = 0;

**foreach** (**var** student **in** items[i].Item2)

{

**if** (student.Item2[j].Item2[k].Item2 == MarkType.Нет)

Count0++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.Неуд)

Count2++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.Удовл)

Count3++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.Хор)

Count4++;

**else** **if** (student.Item2[j].Item2[k].Item2 == MarkType.Отл)

Count5++;

}

itog.**Add**("Нет - " + Count0 +

" Неуд - " + Count2 +

" Удовл - " + Count3 +

" Хор - " + Count4 +

" Отл - " + Count5);

}

}

}

Itog.**Add**(new Tuple<List<**string**>>(itog));

}

**string** groupList = "";

**foreach** (**var** item **in** items)

{

**if** (items.IndexOf(item) != items.Count - 1)

groupList += item.Item1 + ", ";

**else**

groupList += item.Item1;

}

List<**string**> footer = new List<**string**>

{

"Подпись декана \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

};

List<**string**> disciplineName = new List<**string**>

{

"University",

"Кафедра: " + department.Name + " Дисциплина: " + discipline.Name,

"Список групп: " + groupList,

"Преподаватель: " + teacher.Flm,

"Дата: " + DateTime.Now.Date.ToShortDateString()

};

**return** new ReportFullViewModel

{

Title = "Итоговый отчёт по всей дисциплине",

Footer = footer,

DisciplineName = disciplineName,

Itog = Itog,

Items = items

};

}

**public** ReportPlanViewModel GetObjectsForPlanReport(MessageBindingModel model)

{

List<Tuple<**string**, **string**, List<Tuple<**string**, MarkType>>>> items = new List<Tuple<**string**, **string**, List<Tuple<**string**, MarkType>>>>();

TeacherViewModel teacher = \_teacherStorage.GetElement(new TeacherBindingModel

{

Id = model.TeacherId

});

DepartmentViewModel department = \_departmentStorage.GetElement(new DepartmentBindingModel

{

Id = teacher.DepartmentId

});

PlanViewModel plan = \_planStorage.GetElement(new PlanBindingModel

{

Id = model.PlanId

});

GroupViewModel **group** = \_groupStorage.GetElement(new GroupBindingModel

{

Id = plan.GroupId

});

List<TestingViewModel> testings = \_testingStorage.GetFilteredList(new TestingBindingModel

{

PlanId = (**int**)model.PlanId

});

**var** studentTestings = testings[0].StudentTestings;

**foreach** (**var** item **in** studentTestings)

{

StudentViewModel student = \_studentStorage.GetElement(new StudentBindingModel

{

Id = item.Item1

});

items.**Add**(new Tuple<**string**, **string**, List<Tuple<**string**, MarkType>>>

(

student.NumFB, item.Item2, new List<Tuple<**string**, MarkType>>()

));

}

**for** (**int** i = 0; i < testings.Count; i++)

{

**foreach** (**var** grades **in** testings[i].StudentTestings)

{

items.**Where**(rec => rec.Item2 == grades.Item2).ToList()?[0]

.Item3.**Add**(new Tuple<**string**, MarkType>(testings[i].Topic, grades.Item3));

}

}

List<**string**> itog = new List<**string**> { "", "", "Итог: " };

**for** (**int** i = 0; i < items[0].Item3?.Count; i++)

{

**if** (items[0].Item3[i].Item2 == MarkType.П ||

items[0].Item3[i].Item2 == MarkType.НП ||

items[0].Item3[i].Item2 == MarkType.УП)

{

**int** upCount = 0;

**int** pCount = 0;

**int** npCount = 0;

**foreach** (**var** item **in** items)

{

**if** (item.Item3[i].Item2 == MarkType.УП)

upCount++;

**else** **if** (item.Item3[i].Item2 == MarkType.П)

pCount++;

**else** **if** (item.Item3[i].Item2 == MarkType.НП)

npCount++;

}

itog.**Add**("УП - " + upCount +

" П - " + pCount +

" НП - " + npCount);

}

**else**

{

**int** Count0 = 0;

**int** Count2 = 0;

**int** Count3 = 0;

**int** Count4 = 0;

**int** Count5 = 0;

**foreach** (**var** item **in** items)

{

**if** (item.Item3[i].Item2 == MarkType.Нет)

Count0++;

**else** **if** (item.Item3[i].Item2 == MarkType.Неуд)

Count2++;

**else** **if** (item.Item3[i].Item2 == MarkType.Удовл)

Count3++;

**else** **if** (item.Item3[i].Item2 == MarkType.Хор)

Count4++;

**else** **if** (item.Item3[i].Item2 == MarkType.Отл)

Count5++;

}

itog.**Add**("Нет - " + Count0 +

" Неуд - " + Count2 +

" Удовл - " + Count3 +

" Хор - " + Count4 +

" Отл - " + Count5);

}

}

List<**string**> footer = new List<**string**>

{

"Подпись декана \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

};

List<**string**> planName = new List<**string**>

{

"University",

"Кафедра: " + department.Name + " Группа: " + **group**.Name,

plan.Name,

"Преподаватель: " + teacher.Flm,

"Общее количество часов: " + plan.Hours + " Дата: " + DateTime.Now.Date.ToShortDateString()

};

**return** new ReportPlanViewModel

{

Title = "Отчёт по плану",

Footer = footer,

PlanName = planName,

Itog = itog,

Items = items

};

}

**public** **void** SendGroupReport(**int** id)

{

**string** filePath = "D:**\\**ULSTU**\\**ПиАПС**\\**Курсовая**\\**PROJECT**\\**University**\\**reports**\\**Отчёт.docx";

**var** model = GetObjectsForGroupReport(id);

\_saveToWord.CrwateGroupDoc(model, filePath);

\_mailWorker.MailSendAsync(new MailSendInfoBindingModel

{

MailAddress = model.Mail,

Subject = model.Title,

Text = "@Приложение автоматического учёта студентов",

FilePath = filePath

});

}

**public** **void** SaveSumReport(MessageBindingModel model, **string** filename)

{

**var** result = GetObjectsForSumReport(model);

\_saveToWord.CreateFullDoc(result, filename);

}

**public** **void** SavePlanReport(MessageBindingModel model, **string** filename)

{

**var** result = GetObjectsForPlanReport(model);

\_saveToWord.CreatePlanDoc(result, filename);

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** DisciplineLogic : IDisciplineLogic

{

**private** **readonly** IDisciplineStorage \_disciplineStorage;

**public** DisciplineLogic(IDisciplineStorage disciplineStorage)

{

\_disciplineStorage = disciplineStorage;

}

**public** List<DisciplineViewModel> Read(DisciplineBindingModel model)

{

**if** (model == **null**)

{

**return** \_disciplineStorage.GetFullList();

}

**if** (model.Id.HasValue || (model.DepartmentId != **null** &&

model.Name != **null**))

{

**return** new List<DisciplineViewModel> { \_disciplineStorage.GetElement(model) };

}

**return** \_disciplineStorage.GetFilteredList(model);

}

**public** **void** CreateOrUpdate(DisciplineBindingModel model)

{

**var** element = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

DepartmentId = model.DepartmentId,

Name = model.Name

});

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Id.HasValue)

{

\_disciplineStorage.Update(model);

}

**else**

{

\_disciplineStorage.Insert(model);

}

}

**public** **void** Delete(DisciplineBindingModel model)

{

**var** element = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_disciplineStorage.Delete(model);

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** GroupLogic : IGroupLogic

{

**private** **readonly** IGroupStorage \_groupStorage;

**public** GroupLogic(IGroupStorage groupStorage)

{

\_groupStorage = groupStorage;

}

**public** List<GroupViewModel> Read(GroupBindingModel model)

{

**if** (model == **null**)

{

**return** \_groupStorage.GetFullList();

}

**if** (model.Id.HasValue || (model.DepartmentId != **null** &&

model.Name != **null**))

{

**return** new List<GroupViewModel> { \_groupStorage.GetElement(model) };

}

**return** \_groupStorage.GetFilteredList(model);

}

**public** **void** CreateOrUpdate(GroupBindingModel model)

{

**var** element = \_groupStorage.GetElement(new GroupBindingModel

{

DepartmentId = model.DepartmentId,

Name = model.Name,

Course = model.Course

});

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Id.HasValue)

{

\_groupStorage.Update(model);

}

**else**

{

\_groupStorage.Insert(model);

}

}

**public** **void** Delete(GroupBindingModel model)

{

**var** element = \_groupStorage.GetElement(new GroupBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_groupStorage.Delete(model);

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.BusinessLogic

{

**public** **class** PlanLogic : IPlanLogic

{

**private** **readonly** IPlanStorage \_planStorage;

**private** **readonly** IDisciplineStorage \_disciplineStorage;

**private** **readonly** IGroupStorage \_groupStorage;

**public** PlanLogic(IPlanStorage planStorage, IDisciplineStorage disciplineStorage,

IGroupStorage groupStorage)

{

\_planStorage = planStorage;

\_disciplineStorage = disciplineStorage;

\_groupStorage = groupStorage;

}

**public** List<PlanViewModel> Read(PlanBindingModel model)

{

**if** (model == **null**)

{

**return** \_planStorage.GetFullList();

}

**if** (model.Id.HasValue || (model.DepartmentId != **null** &&

model.Name != **null**))

{

**return** new List<PlanViewModel> { \_planStorage.GetElement(model) };

}

**return** \_planStorage.GetFilteredList(model);

}

**public** List<PlanViewModel> GetPlansByDiscipline(**int** id)

{

**return** \_planStorage.GetPlansByDiscipline(id);

}

**public** **void** CreateOrUpdate(PlanBindingModel model)

{

**var** element = \_planStorage.GetElement(new PlanBindingModel

{

DepartmentId = model.DepartmentId,

TeacherId = model.TeacherId,

DisciplineId = model.DisciplineId,

GroupId = model.GroupId,

Name = model.Name,

Hours = model.Hours,

Type = model.Type

});

DisciplineViewModel discipline = \_disciplineStorage.GetElement(new DisciplineBindingModel

{

Id = model.DisciplineId

});

GroupViewModel **group** = \_groupStorage.GetElement(new GroupBindingModel

{

Id = model.GroupId

});

model.Name = model.Type + " план группы " + **group**.Name + " по дисциплине " +

discipline.Name + " с кол-вом часов " + model.Hours;

**if** (element != **null** && element.Id != model.Id)

{

**throw** new Exception("Уже есть такой элемент");

}

**if** (model.Id.HasValue)

{

\_planStorage.Update(model);

}

**else**

{

\_planStorage.Insert(model);

}

}

**public** **void** Delete(PlanBindingModel model)

{

**var** element = \_planStorage.GetElement(new PlanBindingModel

{

Id = model.Id

});

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

\_planStorage.Delete(model);

}

}

}

﻿**using** UniversityContracts.BindingModels;

**namespace** UniversityBusinessLogic.MailWorker

{

**public** **abstract** **class** AbstractMailWorker

{

**protected** **string** \_mailLogin;

**protected** **string** \_mailPassword;

**protected** **string** \_smtpClientHost;

**protected** **int** \_smtpClientPort;

**protected** **string** \_popHost;

**protected** **int** \_popPort;

**public** AbstractMailWorker()

{

}

**public** **void** MailConfig(MailConfigBindingModel config)

{

\_mailLogin = config.MailLogin;

\_mailPassword = config.MailPassword;

\_smtpClientHost = config.SmtpClientHost;

\_smtpClientPort = config.SmtpClientPort;

\_popHost = config.PopHost;

\_popPort = config.PopPort;

}

**public** **async** **void** MailSendAsync(MailSendInfoBindingModel info)

{

**if** (**string**.IsNullOrEmpty(\_mailLogin) || **string**.IsNullOrEmpty(\_mailPassword))

{

**return**;

}

**if** (**string**.IsNullOrEmpty(\_smtpClientHost) || \_smtpClientPort == 0)

{

**return**;

}

**if** (**string**.IsNullOrEmpty(info.MailAddress) || **string**.IsNullOrEmpty(info.Subject) || **string**.IsNullOrEmpty(info.Text))

{

**return**;

}

**await** SendMailAsync(info);

}

**protected** **abstract** Task SendMailAsync(MailSendInfoBindingModel info);

}

}

﻿**using** System.Net.Mail;

**using** System.Net.Mime;

**using** System.Net;

**using** System.Text;

**using** UniversityContracts.BindingModels;

**namespace** UniversityBusinessLogic.MailWorker

{

**public** **class** MailKitWorker : AbstractMailWorker

{

**public** MailKitWorker()

{

}

**protected** **override** **async** Task SendMailAsync(MailSendInfoBindingModel info)

{

**using** **var** objMailMessage = new MailMessage();

**using** **var** objSmtpClient = new SmtpClient(\_smtpClientHost, \_smtpClientPort);

**try**

{

objMailMessage.**From** = new MailAddress(\_mailLogin);

objMailMessage.To.**Add**(new MailAddress(info.MailAddress));

objMailMessage.Subject = info.Subject;

objMailMessage.Body = info.Text;

objMailMessage.SubjectEncoding = Encoding.UTF8;

objMailMessage.BodyEncoding = Encoding.UTF8;

Attachment data = new Attachment(info.FilePath, MediaTypeNames.Application.Octet);

ContentDisposition disposition = data.ContentDisposition;

disposition.CreationDate = System.IO.File.GetCreationTime(info.FilePath);

disposition.ModificationDate = System.IO.File.GetLastWriteTime(info.FilePath);

disposition.ReadDate = System.IO.File.GetLastAccessTime(info.FilePath);

objMailMessage.Attachments.**Add**(data);

objSmtpClient.UseDefaultCredentials = **false**;

objSmtpClient.EnableSsl = **true**;

objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

objSmtpClient.Credentials = new NetworkCredential(\_mailLogin, \_mailPassword);

**await** Task.Run(() => objSmtpClient.Send(objMailMessage));

}

**catch** (Exception)

{

**throw**;

}

}

}

}

﻿**namespace** UniversityBusinessLogic.OfficePackage.HelperEnums

{

**public** **enum** WordJustificationType

{

Center,

Both,

End

}

}

﻿**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** UniversityBusinessLogic.OfficePackage.HelperModels

{

**public** **class** WordInfo

{

**public** **string** FileName { **get**; **set**; }

**public** **string** Title { **get**; **set**; }

**public** List<**int**> StudDisc { **get**; **set**; }

}

}

﻿**namespace** UniversityBusinessLogic.OfficePackage.HelperModels

{

**public** **class** WordParagraph

{

**public** List<(**string**, WordTextProperties)> Texts { **get**; **set**; }

**public** WordTextProperties TextProperties { **get**; **set**; }

}

}

﻿**using** UniversityBusinessLogic.OfficePackage.HelperEnums;

**namespace** UniversityBusinessLogic.OfficePackage.HelperModels

{

**public** **class** WordTextProperties

{

**public** **string** Size { **get**; **set**; }

**public** **bool** Bold { **get**; **set**; }

**public** WordJustificationType JustificationType { **get**; **set**; }

}

}

﻿**using** DocumentFormat.OpenXml.Packaging;

**using** DocumentFormat.OpenXml.Wordprocessing;

**using** DocumentFormat.OpenXml;

**using** UniversityBusinessLogic.OfficePackage.HelperEnums;

**using** UniversityBusinessLogic.OfficePackage.HelperModels;

**namespace** UniversityBusinessLogic.OfficePackage.Implements

{

**public** **class** SaveToWord : AbstractSaveToWord

{

**private** WordprocessingDocument \_wordDocument;

**private** Body \_docBody;

**private** Table \_table;

*/// <summary>*

*/// Получение типа выравнивания*

*/// </summary>*

*/// <param name="type"></param>*

*/// <returns></returns>*

**private** **static** JustificationValues GetJustificationValues(WordJustificationType type)

{

**return** type **switch**

{

WordJustificationType.End => JustificationValues.End,

WordJustificationType.Both => JustificationValues.Both,

WordJustificationType.Center => JustificationValues.Center,

\_ => JustificationValues.Left,

};

}

*/// <summary>*

*/// Настройки страницы*

*/// </summary>*

*/// <returns></returns>*

**private** **static** SectionProperties CreateSectionProperties()

{

**var** properties = new SectionProperties();

**var** pageSize = new PageSize

{

Orient = PageOrientationValues.Portrait

};

properties.AppendChild(pageSize);

**return** properties;

}

*/// <summary>*

*/// Задание форматирования для абзаца*

*/// </summary>*

*/// <param name="paragraphProperties"></param>*

*/// <returns></returns>*

**private** **static** ParagraphProperties CreateParagraphProperties(WordTextProperties paragraphProperties)

{

**if** (paragraphProperties != **null**)

{

**var** properties = new ParagraphProperties();

properties.AppendChild(new Justification()

{

Val = GetJustificationValues(paragraphProperties.JustificationType)

});

properties.AppendChild(new SpacingBetweenLines

{

LineRule = LineSpacingRuleValues.Auto

});

properties.AppendChild(new Indentation());

**var** paragraphMarkRunProperties = new ParagraphMarkRunProperties();

**if** (!**string**.IsNullOrEmpty(paragraphProperties.Size))

{

paragraphMarkRunProperties.AppendChild(new FontSize

{

Val = paragraphProperties.Size

});

}

properties.AppendChild(paragraphMarkRunProperties);

**return** properties;

}

**return** **null**;

}

**protected** **override** **void** CreateWord(**string** info)

{

\_wordDocument = WordprocessingDocument.Create(info, WordprocessingDocumentType.Document);

MainDocumentPart mainPart = \_wordDocument.AddMainDocumentPart();

mainPart.Document = new Document();

\_docBody = mainPart.Document.AppendChild(new Body());

}

**protected** **override** **void** CreateParagraph(WordParagraph paragraph)

{

**if** (paragraph != **null**)

{

**var** docParagraph = new Paragraph();

docParagraph.AppendChild(CreateParagraphProperties(paragraph.TextProperties));

**foreach** (**var** run **in** paragraph.Texts)

{

**var** docRun = new Run();

**var** properties = new RunProperties();

properties.AppendChild(new FontSize { Val = run.Item2.Size });

**if** (run.Item2.Bold)

{

properties.AppendChild(new Bold());

}

docRun.AppendChild(properties);

docRun.AppendChild(new Text

{

Text = run.Item1,

Space = SpaceProcessingModeValues.Preserve

});

docParagraph.AppendChild(docRun);

}

\_docBody.AppendChild(docParagraph);

}

}

**protected** **override** **void** CreateTable(List<**string**> tableHeader)

{

\_table = new Table();

TableProperties tblProp = new TableProperties(

new TableBorders(

new TopBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new BottomBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new LeftBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new RightBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new InsideHorizontalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new InsideVerticalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

}

)

);

\_table.AppendChild(tblProp);

TableRow tableRowHeader = new TableRow();

**int** j = 0;

**foreach** (**string** stringHeaderCell **in** tableHeader)

{

**if** (j == 0)

{

TableCell cellHeader = new TableCell();

cellHeader.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = "450" }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification() { Val = JustificationValues.Center }), new Run(new RunProperties(new Bold()), new Text(stringHeaderCell))));

tableRowHeader.Append(cellHeader);

}

**else** **if** (j == 1)

{

TableCell cellHeader = new TableCell();

cellHeader.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = "1000" }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification() { Val = JustificationValues.Center }), new Run(new RunProperties(new Bold()), new Text(stringHeaderCell))));

tableRowHeader.Append(cellHeader);

}

**else** **if** (j == 2)

{

TableCell cellHeader = new TableCell();

cellHeader.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = "5000" }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification() { Val = JustificationValues.Center }), new Run(new RunProperties(new Bold()), new Text(stringHeaderCell))));

tableRowHeader.Append(cellHeader);

}

**else**

{

TableCell cellHeader = new TableCell();

cellHeader.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = "2000" }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification() { Val = JustificationValues.Center }), new Run(new RunProperties(new Bold()), new Text(stringHeaderCell))));

tableRowHeader.Append(cellHeader);

}

j++;

}

\_table.Append(tableRowHeader);

\_docBody.AppendChild(\_table);

}

**protected** **override** **void** CreateComplexTable(**int**[] width, List<(**int**, **int**, **string**)> joined,

List<**string**> columns)

{

\_table = new Table();

TableProperties tblProp = new TableProperties(

new TableBorders(

new TopBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new BottomBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new LeftBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new RightBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new InsideHorizontalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

},

new InsideVerticalBorder

{

Val = new EnumValue<BorderValues>(BorderValues.Single),

Size = 8

}

)

);

\_table.AppendChild(tblProp);

TableRow tableRowHeaderOne = new TableRow();

**for** (**int** i = 0; i < width.Length; i++)

{

TableCell cellHeader = new TableCell();

**bool** flag = **true**;

**foreach** (**var** **join** **in** joined)

{

**if** (i == **join**.Item1)

{

cellHeader.Append(new TableCellProperties(new HorizontalMerge() { Val = MergedCellValues.Restart }));

cellHeader.Append(new TableCellProperties(new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = width[i].ToString() }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new

Justification()

{ Val = JustificationValues.Center }),

new Run(new RunProperties(new Bold()),

new Text(**join**.Item3))));

flag = **false**;

}

**else** **if** (i > **join**.Item1 && i <= **join**.Item2)

{

cellHeader.Append(new TableCellProperties(new HorizontalMerge() { Val = MergedCellValues.**Continue** }));

cellHeader.Append(new TableCellProperties(new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = width[i].ToString() }));

flag = **false**;

}

}

**if** (flag)

{

cellHeader.Append(new TableCellProperties(new VerticalMerge() { Val = MergedCellValues.Restart }));

cellHeader.Append(new TableCellProperties(new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = width[i].ToString() }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification()

{ Val = JustificationValues.Center }),

new Run(new RunProperties(new Bold()),

new Text(columns[i]))));

}

tableRowHeaderOne.Append(cellHeader);

}

TableRow tableRowHeaderTwo = new TableRow();

**for** (**int** i = 0; i < width.Length; i++)

{

TableCell cellHeader = new TableCell();

cellHeader.Append(new TableCellProperties(new VerticalMerge() { Val = MergedCellValues.**Continue** }));

cellHeader.Append(new TableCellProperties(new TableCellWidth() { Type = TableWidthUnitValues.Dxa, Width = width[i].ToString() }));

cellHeader.Append(new Paragraph(new ParagraphProperties(new Justification()

{ Val = JustificationValues.Center }),

new Run(new RunProperties(new Bold()),

new Text(columns[i]))));

tableRowHeaderTwo.Append(cellHeader);

}

\_table.Append(tableRowHeaderOne);

\_table.Append(tableRowHeaderTwo);

\_docBody.AppendChild(\_table);

}

**protected** **override** **void** CreateRow(List<**string**> tableRowInfo, **bool** complex)

{

TableRow tableRow = new TableRow();

**if** (!complex)

{

TableCell tableCell0 = new TableCell();

tableRow.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa }));

tableRow.Append(new Paragraph(new Run(new Text(" " + tableRowInfo[0]))));

tableRow.Append(tableCell0);

tableRowInfo.**Remove**(tableRowInfo[0]);

TableCell tableCell1 = new TableCell();

tableRow.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa }));

tableRow.Append(new Paragraph(new Run(new Text(" " + tableRowInfo[0]))));

tableRow.Append(tableCell1);

tableRowInfo.**Remove**(tableRowInfo[0]);

}

TableCell tableCell2 = new TableCell();

tableRow.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa }));

tableRow.Append(new Paragraph(new Run(new Text(" " + tableRowInfo[0]))));

tableRow.Append(tableCell2);

tableRowInfo.**Remove**(tableRowInfo[0]);

**foreach** (**string** celltext **in** tableRowInfo)

{

TableCell tableCellmain = new TableCell();

tableRow.Append(new TableCellProperties(

new TableCellWidth() { Type = TableWidthUnitValues.Dxa }));

tableRow.Append(new Paragraph(new ParagraphProperties(

new Justification() { Val = JustificationValues.End }), (new Run(new Text(celltext + " ")))));

tableRow.Append(tableCellmain);

}

\_table.Append(tableRow);

}

**protected** **override** **void** SaveWord()

{

\_docBody.AppendChild(CreateSectionProperties());

\_wordDocument.MainDocumentPart.Document.Save();

\_wordDocument.Close();

}

}

}

﻿**using** DocumentFormat.OpenXml.EMMA;

**using** UniversityBusinessLogic.OfficePackage.HelperEnums;

**using** UniversityBusinessLogic.OfficePackage.HelperModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityBusinessLogic.OfficePackage

{

**public** **abstract** **class** AbstractSaveToWord

{

**public** **void** CrwateGroupDoc(ReportGroupViewModel model, **string** filename)

{

CreateWord(filename);

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(model.Title.ToUpper(), new WordTextProperties { Bold = **true**, Size = "24", })},

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

**foreach** (**var** u **in** model.Upper)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(u, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

List<**string**> head = new List<**string**>

{

"№ з.к.", "ФИО", "Основа"

};

CreateTable(head);

**foreach** (**var** items **in** model.Items)

{

CreateRow(new List<**string**> { items.Item1, items.Item2, items.Item3.ToString() }, **false**);

}

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(" ", new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

**foreach** (**var** p **in** model.Footer)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(p, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

SaveWord();

}

**public** **void** CreatePlanDoc(ReportPlanViewModel model, **string** filename)

{

CreateWord(filename);

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(model.Title.ToUpper(), new WordTextProperties { Bold = **true**, Size = "24", })},

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

**foreach** (**var** pn **in** model.PlanName)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(pn, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(" ", new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

List<**string**> head = new List<**string**>

{

"№", "№ з.к.", "ФИО"

};

**foreach** (**var** testing **in** model.Items[0].Item3)

{

head.**Add**(testing.Item1);

}

CreateTable(head);

**int** j = 0;

**foreach** (**var** items **in** model.Items)

{

j++;

**var** list = new List<**string**>

{

j.ToString(), items.Item1, items.Item2

};

**foreach** (**var** subItem **in** items.Item3)

{

list.**Add**(subItem.Item2.ToString());

}

CreateRow(list, **false**);

}

CreateRow(model.Itog, **false**);

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(" ", new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

**foreach**(**var** p **in** model.Footer)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(p, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

SaveWord();

}

**public** **void** CreateFullDoc(ReportFullViewModel model, **string** filename)

{

**int** maxWidth = 13000;

CreateWord(filename);

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(model.Title.ToUpper(), new WordTextProperties { Bold = **true**, Size = "24", })},

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

**foreach** (**var** pn **in** model.DisciplineName)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(pn, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

*//-------------body------------------*

**foreach** (**var** **group** **in** model.Items)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(" ", new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

("Группа: " + **group**.Item1, new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

**var** **join** = new List<(**int**, **int**, **string**)>();

**int** count = 1;

**for** (**int** i = 0; i < **group**.Item2[0].Item2.Count(); i++)

{

**join**.**Add**((

count,

count + **group**.Item2[0].Item2[i].Item2.Count() - 1,

**group**.Item2[0].Item2[i].Item1));

count += **group**.Item2[0].Item2[i].Item2.Count();

}

**var** columns = new List<**string**> { "ФИО" };

**foreach** (**var** subitem **in** **group**.Item2[0].Item2)

{

**foreach** (**var** subitem2 **in** subitem.Item2)

{

columns.**Add**(subitem2.Item1);

}

}

**var** width = new **int**[columns.Count()];

**for** (**int** i = 0; i < width.Length; i++)

{

width[i] = maxWidth / width.Length;

}

CreateComplexTable(width, **join**, columns);

**foreach** (**var** subItem **in** **group**.Item2)

{

**var** row = new List<**string**>{ subItem.Item1 };

**foreach**(**var** subItem2 **in** subItem.Item2)

{

**foreach**(**var** subItem3 **in** subItem2.Item2)

{

row.**Add**(subItem3.Item2.ToString());

}

}

CreateRow(row, **true**);

}

CreateRow(model.Itog[model.Items.IndexOf(**group**)].Item1, **true**);

}

*//--------------------------------------------------*

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(" ", new WordTextProperties { Bold = **true**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

**foreach** (**var** p **in** model.Footer)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(**string**, WordTextProperties)> {

(p, new WordTextProperties { Bold = **false**, Size = "20", })},

TextProperties = new WordTextProperties

{

Size = "20",

JustificationType = WordJustificationType.Both

}

});

}

SaveWord();

}

*/// <summary>*

*/// Создание doc-файла*

*/// </summary>*

*/// <param name="info"></param>*

**protected** **abstract** **void** CreateWord(**string** info);

*/// <summary>*

*/// Создание абзаца с текстом*

*/// </summary>*

*/// <param name="paragraph"></param>*

*/// <returns></returns>*

**protected** **abstract** **void** CreateParagraph(WordParagraph paragraph);

*/// <summary>*

*/// Создание заголовка таблицы с текстом*

*/// </summary>*

*/// <param name="tableHeader"></param>*

**protected** **abstract** **void** CreateTable(List<**string**> tableHeader);

*/// <summary>*

*/// Создание сложного заголовка таблицы с текстом*

*/// </summary>*

*/// <param name="tableHeader"></param>*

**protected** **abstract** **void** CreateComplexTable(**int**[] width, List<(**int**, **int**, **string**)> joined,

List<**string**> columns);

*/// <summary>*

*/// Создание строки таблицы с текстом*

*/// </summary>*

*/// <param name="tableRow"></param>*

**protected** **abstract** **void** CreateRow(List<**string**> tableRow, **bool** complex);

*/// <summary>*

*/// Сохранение файла*

*/// </summary>*

**protected** **abstract** **void** SaveWord();

}

}

Библиотека классов UniversityContracts

**namespace** UniversityContracts.BindingModels

{

**public** **class** MailConfigBindingModel

{

**public** **string** MailLogin { **get**; **set**; }

**public** **string** MailPassword { **get**; **set**; }

**public** **string** SmtpClientHost { **get**; **set**; }

**public** **int** SmtpClientPort { **get**; **set**; }

**public** **string** PopHost { **get**; **set**; }

**public** **int** PopPort { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** MessageBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** ReportTypes? ReportType { **get**; **set**; }

**public** Status? Status { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int**? PlanId { **get**; **set**; }

**public** **int**? TeacherId { **get**; **set**; }

**public** **int**? DisciplineId { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** MailSendInfoBindingModel

{

**public** **string** MailAddress { **get**; **set**; }

**public** **string** Subject { **get**; **set**; }

**public** **string** Text { **get**; **set**; }

**public** **string** FilePath { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** PlanBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** PlanType? Type { **get**; **set**; }

**public** **int**? ToSkip { **get**; **set**; }

**public** **int**? ToTake { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** StudentBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int**? DepartmentId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string**? NumFB { **get**; **set**; }

**public** TypeEducationBasement Basement { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** StudentTestingBindingModel

{

**public** **int** TestingId { **get**; **set**; }

**public** **int** StudentId { **get**; **set**; }

**public** MarkType Grade { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** TeacherBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string**? Login { **get**; **set**; }

**public** **string**? Password { **get**; **set**; }

**public** List<**int**> TeacherDisciplines { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** TeacherDisciplineBindingModel

{

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** TestingBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** Topic { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** DateTime Date { **get**; **set**; }

**public** PlanType? Type { **get**; **set**; }

**public** Dictionary<**int**, MarkType>? StudentTestings { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** DepartmentBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **string**? Name { **get**; **set**; }

**public** **string**? Login { **get**; **set**; }

**public** **string**? Password { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** DisciplineBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** GroupBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Course { **get**; **set**; }

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IMessageLogic

{

List<MessageViewModel> Read(MessageBindingModel model);

**void** CreateMessage(MessageBindingModel model);

**void** AnswerRequest(MessageBindingModel model);

**void** CloseRequest(MessageBindingModel model);

**void** MessageRollBack(MessageBindingModel model);

**void** Delete(MessageBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IReportLogic

{

ReportGroupViewModel GetObjectsForGroupReport(**int** id);

ReportFullViewModel GetObjectsForSumReport(MessageBindingModel model);

ReportPlanViewModel GetObjectsForPlanReport(MessageBindingModel model);

**void** SendGroupReport(**int** id);

**void** SaveSumReport(MessageBindingModel model, **string** filename);

**void** SavePlanReport(MessageBindingModel model, **string** filename);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IStudentLogic

{

List<StudentViewModel> Read(StudentBindingModel model);

**void** CreateOrUpdate(StudentBindingModel model);

**void** Delete(StudentBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IPlanLogic

{

List<PlanViewModel> Read(PlanBindingModel model);

List<PlanViewModel> GetPlansByDiscipline(**int** id);

**void** CreateOrUpdate(PlanBindingModel model);

**void** Delete(PlanBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** ITeacherLogic

{

List<TeacherViewModel> Read(TeacherBindingModel model);

List<TeacherViewModel> GetTeacherByDiscipline(**int** id);

**void** CreateOrUpdate(TeacherBindingModel model);

**void** Delete(TeacherBindingModel model);

**void** AddDiscipline(TeacherDisciplineBindingModel model);

**void** RemoveDiscipline(TeacherDisciplineBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** ITestingLogic

{

List<TestingViewModel> Read(TestingBindingModel model);

**void** CreateOrUpdate(TestingBindingModel model);

**void** Delete(TestingBindingModel model);

**void** AddStudent(StudentTestingBindingModel model);

**void** RemoveStudent(StudentTestingBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IDepartmentLogic

{

List<DepartmentViewModel> Read(DepartmentBindingModel model);

DepartmentViewModel? Authorization(**string** login, **string** password);

**void** CreateOrUpdate(DepartmentBindingModel model);

**void** Delete(DepartmentBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IDisciplineLogic

{

List<DisciplineViewModel> Read(DisciplineBindingModel model);

**void** CreateOrUpdate(DisciplineBindingModel model);

**void** Delete(DisciplineBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.BusinessLogicContracts

{

**public** **interface** IGroupLogic

{

List<GroupViewModel> Read(GroupBindingModel model);

**void** CreateOrUpdate(GroupBindingModel model);

**void** Delete(GroupBindingModel model);

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** Status

{

Активен = 0,

Проверяется = 1,

Проверен = 2

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**namespace** UniversityContracts.Enums

{

**public** **enum** ReportTypes

{

[Display(Name = "Итоговый отчёт по дисциплине")]

SumReport = 0,

[Display(Name = "Отчёт по плану")]

PlanReport = 1

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** TypeEducationBasement

{

Б = 0,

ПО = 1,

ЦП = 3

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** MarkType

{

Нет = 0,

*// оценки*

Неуд = 2,

Удовл = 3,

Хор = 4,

Отл = 5,

*// посещаемость*

НП = 6,

УП = 7,

П = 8

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** PlanType

{

Лабораторный = 0,

Лекционный = 1,

Практический = 2,

Семинарный = 3,

Консультационный = 4,

Проверочный = 5

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IMessageStorage

{

List<MessageViewModel> GetFullList();

List<MessageViewModel> GetFilteredList(MessageBindingModel model);

MessageViewModel GetElement(MessageBindingModel model);

**void** Insert(MessageBindingModel model);

**void** Update(MessageBindingModel model);

**void** Delete(MessageBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IStudentStorage

{

List<StudentViewModel> GetFullList();

List<StudentViewModel> GetFilteredList(StudentBindingModel model);

StudentViewModel GetElement(StudentBindingModel model);

**void** Insert(StudentBindingModel model);

**void** Update(StudentBindingModel model);

**void** Delete(StudentBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IPlanStorage

{

List<PlanViewModel> GetFullList();

List<PlanViewModel> GetFilteredList(PlanBindingModel model);

List<PlanViewModel> GetPlansByDiscipline(**int** id);

PlanViewModel GetElement(PlanBindingModel model);

**void** Insert(PlanBindingModel model);

**void** Update(PlanBindingModel model);

**void** Delete(PlanBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** ITestingStorage

{

List<TestingViewModel> GetFullList();

List<TestingViewModel> GetFilteredList(TestingBindingModel model);

TestingViewModel GetElement(TestingBindingModel model);

**void** Insert(TestingBindingModel model);

**void** Update(TestingBindingModel model);

**void** Delete(TestingBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** ITeacherStorage

{

List<TeacherViewModel> GetFullList();

List<TeacherViewModel> GetFilteredList(TeacherBindingModel model);

List<TeacherViewModel> GetTeacherByDiscipline(**int** id);

TeacherViewModel GetElement(TeacherBindingModel model);

**void** Insert(TeacherBindingModel model);

**void** Update(TeacherBindingModel model);

**void** Delete(TeacherBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IDepartmentStorage

{

List<DepartmentViewModel> GetFullList();

DepartmentViewModel GetElement(DepartmentBindingModel model);

**void** Insert(DepartmentBindingModel model);

**void** Update(DepartmentBindingModel model);

**void** Delete(DepartmentBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IDisciplineStorage

{

List<DisciplineViewModel> GetFullList();

List<DisciplineViewModel> GetFilteredList(DisciplineBindingModel model);

DisciplineViewModel GetElement(DisciplineBindingModel model);

**void** Insert(DisciplineBindingModel model);

**void** Update(DisciplineBindingModel model);

**void** Delete(DisciplineBindingModel model);

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.ViewModels;

**namespace** UniversityContracts.StorageContracts

{

**public** **interface** IGroupStorage

{

List<GroupViewModel> GetFullList();

List<GroupViewModel> GetFilteredList(GroupBindingModel model);

GroupViewModel GetElement(GroupBindingModel model);

**void** Insert(GroupBindingModel model);

**void** Update(GroupBindingModel model);

**void** Delete(GroupBindingModel model);

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** PlanViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **string** DisciplineName { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** GroupName { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** PlanType Type { **get**; **set**; }

**public** List<Tuple<**int**, **string**, **int**, DateTime>> Testings { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** ReportFullViewModel

{

**public** **string** Title { **get**; **set**; }

**public** List<**string**> Footer { **get**; **set**; }

**public** List<**string**> DisciplineName { **get**; **set**; }

**public** List<Tuple<List<**string**>>> Itog { **get**; **set**; }

**public** List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>>> Items { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** ReportPlanViewModel

{

**public** **string** Title { **get**; **set**; }

**public** List<**string**> Footer { **get**; **set**; }

**public** List<**string**> PlanName { **get**; **set**; }

**public** List<**string**> Itog { **get**; **set**; }

**public** List<Tuple<**string**, **string**, List<Tuple<**string**, MarkType>>>> Items { **get**; **set**;}

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** ReportGroupViewModel

{

**public** **string** Mail { **get**; **set**; }

**public** **string** Title { **get**; **set**; }

**public** List<**string**> Upper { **get**; **set**; }

**public** List<**string**> Footer { **get**; **set**; }

**public** List<Tuple<**string**, **string**, TypeEducationBasement>> Items { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** StudentViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string** NumFB { **get**; **set**; }

**public** TypeEducationBasement Basement { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** TeacherViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string** Login { **get**; **set**; }

**public** **string** Password { **get**; **set**; }

**public** Dictionary<**int**, **string**> TeacherDisciplines { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** TestingViewModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** Topic { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** DateTime Date { **get**; **set**; }

**public** PlanType Type { **get**; **set**; }

**public** List<Tuple<**int**, **string**, MarkType>> StudentTestings { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** DepartmentViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **string** Login { **get**; **set**; }

**public** **string** Password { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** DisciplineViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** GroupViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Course { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** MessageViewModel

{

**public** **int** Id { **get**; **set**; }

**public** ReportTypes ReportType { **get**; **set**; }

**public** Status Status { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** PlanName { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **string** DisciplineName { **get**; **set**; }

}

}

Библиотека классов UniversityDataBaseImplement

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** PlanStorage : IPlanStorage

{

**public** List<PlanViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Plans

.**Select**(CreateModel)

.ToList();

}

**public** List<PlanViewModel> GetFilteredList(PlanBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Plans

.**Where**(rec => (rec.DepartmentId == model.DepartmentId) ||

(rec.TeacherId == model.TeacherId && rec.DisciplineId == model.DisciplineId &&

rec.GroupId == model.GroupId && rec.Type == model.Type) ||

(rec.TeacherId == model.TeacherId && rec.DisciplineId == model.DisciplineId &&

model.GroupId == 0) ||

(rec.TeacherId == model.TeacherId && rec.GroupId == model.GroupId &&

rec.DisciplineId == model.DisciplineId && model.Type == **null**) ||

(rec.TeacherId == model.TeacherId && rec.Type == model.Type &&

model.GroupId == 0 && model.DisciplineId == 0) ||

(rec.TeacherId == model.TeacherId && model.DisciplineId == 0 && model.GroupId == 0) ||

(rec.TeacherId == 0 && rec.GroupId == 0 && rec.DisciplineId == model.DisciplineId))

.Skip(model.ToSkip ?? 0)

.Take(model.ToTake ?? context.Plans.Count())

.**Select**(CreateModel)

.ToList();

}

**public** List<PlanViewModel> GetPlansByDiscipline(**int** id)

{

**using** **var** context = new UniversityDatabase();

**return** context.Plans

.**Where**(rec => rec.Discipline.Id == id)

.**Select**(CreateModel)

.ToList();

}

**public** PlanViewModel GetElement(PlanBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** lesson = context.Plans

.FirstOrDefault(rec => rec.Id == model.Id ||

(rec.DepartmentId == model.DepartmentId && rec.Name == model.Name));

**return** lesson != **null** ? CreateModel(lesson) : **null**;

}

**public** **void** Insert(PlanBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Plans.**Add**(CreateModel(model, new Plan()));

context.SaveChanges();

}

**public** **void** Update(PlanBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Plans.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

**public** **void** Delete(PlanBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Plan? element = context.Plans.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Plans.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Plan CreateModel(PlanBindingModel model, Plan plan)

{

plan.DepartmentId = model.DepartmentId;

plan.TeacherId = model.TeacherId;

plan.GroupId = model.GroupId;

plan.DisciplineId = model.DisciplineId;

plan.Name = model.Name;

plan.Hours = model.Hours;

plan.Type = (PlanType)model.Type;

**return** plan;

}

**private** **static** PlanViewModel CreateModel(Plan plan)

{

**using** **var** context = new UniversityDatabase();

List<Testing> testings = context.Testings

.**Where**(rec => rec.PlanId == plan.Id)

.ToList();

Discipline? discipline = context.Disciplines.FirstOrDefault(rec => rec.Id == plan.DisciplineId);

**Group**? **group** = context.Groups.FirstOrDefault(rec => rec.Id == plan.GroupId);

PlanViewModel model = new PlanViewModel()

{

Id = plan.Id,

TeacherId = plan.TeacherId,

GroupId = plan.GroupId,

DisciplineId = plan.DisciplineId,

DepartmentId = plan.DepartmentId,

Name = plan.Name,

Hours = plan.Hours,

Type = plan.Type,

Testings = new List<Tuple<**int**, **string**, **int**, DateTime>>()

};

**if**(discipline != **null**)

{

model.DisciplineName = discipline.Name;

}

**if**(**group** != **null**)

{

model.GroupName = **group**.Name;

}

**if** (testings != **null**)

{

**foreach** (**var** test **in** testings)

{

model.Testings.**Add**(new Tuple<**int**, **string**, **int**, DateTime>( test.Id, test.Topic, test.Hours, test.Date ));

}

}

**return** model;

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** StudentStorage : IStudentStorage

{

**public** List<StudentViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Students

.**Select**(CreateModel)

.ToList();

}

**public** List<StudentViewModel> GetFilteredList(StudentBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Students

.**Where**(rec => rec.DepartmentId == model.DepartmentId || rec.GroupId == model.GroupId)

.**Select**(CreateModel)

.ToList();

}

**public** StudentViewModel GetElement(StudentBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** student = context.Students

.FirstOrDefault(rec => rec.Id == model.Id);

**return** student != **null** ? CreateModel(student) : **null**;

}

**public** **void** Insert(StudentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Students.**Add**(CreateModel(model, new Student()));

context.SaveChanges();

}

**public** **void** Update(StudentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Students.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

**public** **void** Delete(StudentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Student? element = context.Students.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Students.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Student CreateModel(StudentBindingModel model, Student student)

{

**using** **var** context = new UniversityDatabase();

**int**? idLastStudent;

**if** (context.Students.ToList().Count() > 0)

{

idLastStudent = context.Students.OrderBy(x => x.Id).Last().Id;

}

**else**

{

idLastStudent = 0;

}

student.DepartmentId = model.DepartmentId.**Value**;

student.GroupId = model.GroupId;

student.Flm = model.Flm;

student.NumFB = DateTime.Now.Year.ToString() + '/' + (idLastStudent + 1).ToString();

student.Basement = model.Basement;

**return** student;

}

**private** **static** StudentViewModel CreateModel(Student student)

{

**return** new StudentViewModel

{

Id = student.Id,

DepartmentId = student.DepartmentId,

Basement = student.Basement,

NumFB = student.NumFB,

GroupId = student.GroupId,

Flm = student.Flm

};

}

}

}

﻿**using** Microsoft.EntityFrameworkCore;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** TeacherStorage : ITeacherStorage

{

**public** List<TeacherViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Teachers

.Include(rec => rec.TeacherDisciplines)

.ThenInclude(rec => rec.Discipline)

.ToList()

.**Select**(CreateModel)

.ToList();

}

**public** List<TeacherViewModel> GetFilteredList(TeacherBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Teachers

.Include(rec => rec.TeacherDisciplines)

.ThenInclude(rec => rec.Discipline)

.**Where**(rec => (rec.Login == model.Login && rec.Password == model.Password) ||

rec.DepartmentId == model.DepartmentId)

.**Select**(CreateModel)

.ToList();

}

**public** List<TeacherViewModel> GetTeacherByDiscipline(**int** id)

{

**using** **var** context = new UniversityDatabase();

**var** teacherDisciplines = context.TeacherDisciplines.**Where**(rec => rec.DisciplineId == id).ToList();

List<TeacherViewModel> teachers = context.Teachers.Include(rec => rec.TeacherDisciplines).ThenInclude(rec => rec.Discipline).ToList()

.**Select**(CreateModel)

.ToList();

List<TeacherViewModel> result = new List<TeacherViewModel>();

**if** (teacherDisciplines.Count() > 0)

{

**foreach** (**var** discipline **in** teacherDisciplines)

{

**foreach** (**var** teacher **in** teachers)

{

**if** (discipline.TeacherId == teacher.Id)

{

result.**Add**(teacher);

}

}

}

}

**return** result;

}

**public** TeacherViewModel GetElement(TeacherBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** teacher = context.Teachers

.Include(rec => rec.TeacherDisciplines)

.ThenInclude(rec => rec.Discipline)

.FirstOrDefault(rec => rec.Login == model.Login || rec.Id == model.Id);

**return** teacher != **null** ? CreateModel(teacher) : **null**;

}

**public** **void** Insert(TeacherBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**using** **var** transaction = context.Database.BeginTransaction();

**try**

{

Teacher teacher = new Teacher()

{

DepartmentId = model.DepartmentId,

Flm = model.Flm,

Login = model.Login,

Password = model.Password

};

context.Teachers.**Add**(teacher);

context.SaveChanges();

CreateModel(model, teacher, context);

transaction.Commit();

}

**catch**

{

transaction.Rollback();

**throw**;

}

}

**public** **void** Update(TeacherBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**using** **var** transaction = context.Database.BeginTransaction();

**try**

{

**var** element = context.Teachers.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element, context);

context.SaveChanges();

transaction.Commit();

}

**catch**

{

transaction.Rollback();

**throw**;

}

}

**public** **void** Delete(TeacherBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Teacher? element = context.Teachers.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Teachers.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Teacher CreateModel(TeacherBindingModel model, Teacher teacher, UniversityDatabase context)

{

teacher.DepartmentId = model.DepartmentId;

teacher.Flm = model.Flm;

teacher.Login = model.Login;

teacher.Password = model.Password;

**if** (model.TeacherDisciplines == **null**)

{

**return** teacher;

}

**if** (model.Id.HasValue)

{

**var** teacherDisciplines = context.TeacherDisciplines.**Where**(rec => rec.TeacherId == model.Id.**Value**).ToList();

context.TeacherDisciplines.RemoveRange(teacherDisciplines.**Where**(rec =>

!model.TeacherDisciplines.Contains(rec.DisciplineId)).ToList());

context.SaveChanges();

}

**foreach** (**var** td **in** model.TeacherDisciplines)

{

**if** (context.TeacherDisciplines.**Where**(rec => rec.TeacherId == teacher.Id &&

rec.DisciplineId == td).ToList().Count() == 0) {

context.TeacherDisciplines.**Add**(new TeacherDiscipline

{

TeacherId = teacher.Id,

DisciplineId = td,

});

}

context.SaveChanges();

}

**return** teacher;

}

**private** **static** TeacherViewModel CreateModel(Teacher teacher)

{

**return** new TeacherViewModel

{

Id = teacher.Id,

DepartmentId = teacher.DepartmentId,

Flm = teacher.Flm,

Login = teacher.Login,

Password = teacher.Password,

TeacherDisciplines = teacher.TeacherDisciplines

.ToDictionary(recTD => recTD.DisciplineId,

recTD => (recTD.Discipline?.Name))

};

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** DisciplineStorage : IDisciplineStorage

{

**public** List<DisciplineViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Disciplines

.**Select**(CreateModel)

.ToList();

}

**public** List<DisciplineViewModel> GetFilteredList(DisciplineBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Disciplines

.**Where**(rec => rec.DepartmentId == model.DepartmentId)

.**Select**(CreateModel)

.ToList();

}

**public** DisciplineViewModel GetElement(DisciplineBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** discipline = context.Disciplines

.FirstOrDefault(rec => (rec.Name == model.Name &&

rec.DepartmentId == model.DepartmentId)

|| rec.Id == model.Id);

**return** discipline != **null** ? CreateModel(discipline) : **null**;

}

**public** **void** Insert(DisciplineBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Disciplines.**Add**(CreateModel(model, new Discipline()));

context.SaveChanges();

}

**public** **void** Update(DisciplineBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Disciplines.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

**public** **void** Delete(DisciplineBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Discipline? element = context.Disciplines.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Disciplines.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Discipline CreateModel(DisciplineBindingModel model, Discipline discipline)

{

discipline.DepartmentId = model.DepartmentId;

discipline.Name = model.Name;

**return** discipline;

}

**private** **static** DisciplineViewModel CreateModel(Discipline discipline)

{

**return** new DisciplineViewModel

{

Id = discipline.Id,

DepartmentId = discipline.DepartmentId,

Name = discipline.Name,

};

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** GroupStorage : IGroupStorage

{

**public** List<GroupViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Groups

.**Select**(CreateModel)

.ToList();

}

**public** List<GroupViewModel> GetFilteredList(GroupBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Groups

.**Where**(rec => rec.DepartmentId == model.DepartmentId)

.**Select**(CreateModel)

.ToList();

}

**public** GroupViewModel GetElement(GroupBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** **group** = context.Groups

.FirstOrDefault(rec => (rec.Name == model.Name &&

rec.DepartmentId == model.DepartmentId)

|| rec.Id == model.Id);

**return** **group** != **null** ? CreateModel(**group**) : **null**;

}

**public** **void** Insert(GroupBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Groups.**Add**(CreateModel(model, new **Group**()));

context.SaveChanges();

}

**public** **void** Update(GroupBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Groups.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

**public** **void** Delete(GroupBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**Group**? element = context.Groups.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Groups.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** **Group** CreateModel(GroupBindingModel model, **Group** **group**)

{

**group**.DepartmentId = model.DepartmentId;

**group**.Name = model.Name;

**group**.Course = model.Course;

**return** **group**;

}

**private** **static** GroupViewModel CreateModel(**Group** **group**)

{

**return** new GroupViewModel

{

Id = **group**.Id,

DepartmentId = **group**.DepartmentId,

Name = **group**.Name,

Course = **group**.Course

};

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** MessageStorage : IMessageStorage

{

**public** List<MessageViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Messages

.**Select**(CreateModel)

.ToList();

}

**public** List<MessageViewModel> GetFilteredList(MessageBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Messages

.**Where**(rec => (rec.DepartmentId == model.DepartmentId) ||

(rec.TeacherId == model.TeacherId && model.Status == **null**) ||

(rec.TeacherId == model.TeacherId && rec.Status == model.Status && model.ReportType == **null**) ||

(rec.TeacherId == model.TeacherId && rec.Status == model.Status && rec.ReportType == model.ReportType))

.**Select**(CreateModel)

.ToList();

}

**public** MessageViewModel GetElement(MessageBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** message = context.Messages

.FirstOrDefault(rec => rec.Id == model.Id);

**return** message != **null** ? CreateModel(message) : **null**;

}

**public** **void** Insert(MessageBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Messages.**Add**(CreateModel(model, new Message(), context));

context.SaveChanges();

}

**public** **void** Update(MessageBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Messages.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element, context);

context.SaveChanges();

}

**public** **void** Delete(MessageBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Message? element = context.Messages.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Messages.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Message CreateModel(MessageBindingModel model, Message message, UniversityDatabase context)

{

**if** (model.Status == Status.Активен && !model.ReportType.HasValue)

{

**if** (model.PlanId != **null**)

{

message.PlanId = (**int**)model.PlanId;

**var** lesson = context.Plans.FirstOrDefault(rec => rec.Id == model.PlanId);

message.TeacherId = lesson.TeacherId;

message.ReportType = ReportTypes.PlanReport;

}

**if** (model.TeacherId != **null** && model.DisciplineId != **null**)

{

message.TeacherId = (**int**)model.TeacherId;

message.DisciplineId = (**int**)model.DisciplineId;

message.ReportType = ReportTypes.SumReport;

}

}

message.DepartmentId = model.DepartmentId;

message.Status = (Status)model.Status;

**return** message;

}

**private** **static** MessageViewModel CreateModel(Message message)

{

**using** **var** context = new UniversityDatabase();

Discipline? discipline = context.Disciplines.FirstOrDefault(rec => rec.Id == message.DisciplineId);

Plan? plan = context.Plans.FirstOrDefault(rec => rec.Id == message.PlanId);

MessageViewModel model = new MessageViewModel

{

Id = message.Id,

DepartmentId = message.DepartmentId,

TeacherId = message.TeacherId,

PlanId = message.PlanId,

DisciplineId = message.DisciplineId,

ReportType = message.ReportType,

Status = message.Status

};

**if** (discipline != **null**)

{

model.DisciplineName = discipline.Name;

}

**if** (plan != **null**)

{

model.PlanName = plan.Name;

}

**return** model;

}

}

}

﻿**using** UniversityContracts.BindingModels;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** DepartmentStorage : IDepartmentStorage

{

**public** List<DepartmentViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Departments

.**Select**(CreateModel)

.ToList();

}

**public** DepartmentViewModel GetElement(DepartmentBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** department = context.Departments

.FirstOrDefault(rec => rec.Login == model.Login || rec.Id == model.Id);

**return** department != **null** ? CreateModel(department) : **null**;

}

**public** **void** Insert(DepartmentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

context.Departments.**Add**(CreateModel(model, new Department()));

context.SaveChanges();

}

**public** **void** Update(DepartmentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**var** element = context.Departments.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

**public** **void** Delete(DepartmentBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Department? element = context.Departments.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Departments.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Department CreateModel(DepartmentBindingModel model, Department department)

{

department.Name = model.Name;

department.Login = model.Login;

department.Password = model.Password;

**return** department;

}

**private** **static** DepartmentViewModel CreateModel(Department department)

{

**return** new DepartmentViewModel

{

Id = department.Id,

Name = department.Name,

Login = department.Login,

Password = department.Password

};

}

}

}

﻿**using** Microsoft.EntityFrameworkCore;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.Enums;

**using** UniversityContracts.StorageContracts;

**using** UniversityContracts.ViewModels;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement.Implements

{

**public** **class** TestingStorage : ITestingStorage

{

**public** List<TestingViewModel> GetFullList()

{

**using** **var** context = new UniversityDatabase();

**return** context.Testings

.Include(rec => rec.StudentTestings)

.ThenInclude(rec => rec.Student)

.ToList()

.**Select**(CreateModel)

.ToList();

}

**public** List<TestingViewModel> GetFilteredList(TestingBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**return** context.Testings

.Include(rec => rec.StudentTestings)

.ThenInclude(rec => rec.Student)

.**Where**(rec => rec.PlanId == model.PlanId)

.ToList()

.**Select**(CreateModel)

.ToList();

}

**public** TestingViewModel GetElement(TestingBindingModel model)

{

**if** (model == **null**)

{

**return** **null**;

}

**using** **var** context = new UniversityDatabase();

**var** testing = context.Testings

.Include(rec => rec.StudentTestings)

.ThenInclude(rec => rec.Student)

.FirstOrDefault(rec => rec.Id == model.Id || (rec.Topic == model.Topic && rec.PlanId == model.PlanId));

**return** testing != **null** ? CreateModel(testing) : **null**;

}

**public** **void** Insert(TestingBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**using** **var** transaction = context.Database.BeginTransaction();

**try**

{

Testing testing = new Testing()

{

PlanId = model.PlanId,

Topic = model.Topic,

Type = (PlanType)model.Type,

Hours = model.Hours,

Date = model.Date

};

context.Testings.**Add**(testing);

context.SaveChanges();

CreateModel(model, testing, context);

transaction.Commit();

}

**catch**

{

transaction.Rollback();

**throw**;

}

}

**public** **void** Update(TestingBindingModel model)

{

**using** **var** context = new UniversityDatabase();

**using** **var** transaction = context.Database.BeginTransaction();

**try**

{

**var** element = context.Testings.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element == **null**)

{

**throw** new Exception("Элемент не найден");

}

CreateModel(model, element, context);

context.SaveChanges();

transaction.Commit();

}

**catch**

{

transaction.Rollback();

**throw**;

}

}

**public** **void** Delete(TestingBindingModel model)

{

**using** **var** context = new UniversityDatabase();

Testing? element = context.Testings.FirstOrDefault(rec => rec.Id == model.Id);

**if** (element != **null**)

{

context.Testings.**Remove**(element);

context.SaveChanges();

}

**else**

{

**throw** new Exception("Элемент не найден");

}

}

**private** **static** Testing CreateModel(TestingBindingModel model, Testing testing, UniversityDatabase context)

{

**if** (model.Topic != **null** && model.Type != **null** &&

model.Hours != **null** && model.Date != **null**)

{

testing.Topic = model.Topic;

testing.Hours = model.Hours;

testing.Type = (PlanType)model.Type;

testing.Date = model.Date;

}

**if** (model.StudentTestings == **null**)

{

**return** testing;

}

**if** (model.Id.HasValue)

{

**var** studentTestings = context.StudentTestings.**Where**(rec => rec.TestingId == model.Id.**Value**).ToList();

context.StudentTestings.RemoveRange(studentTestings.**Where**(rec =>

!model.StudentTestings.ContainsKey(rec.StudentId)).ToList());

context.SaveChanges();

**foreach** (**var** updateTesting **in** studentTestings)

{

**try**

{

updateTesting.Grade = model.StudentTestings[updateTesting.StudentId];

}

**catch** { }

model.StudentTestings.**Remove**(updateTesting.StudentId);

}

context.SaveChanges();

}

**foreach** (**var** st **in** model.StudentTestings)

{

context.StudentTestings.**Add**(new StudentTesting

{

TestingId = testing.Id,

StudentId = st.Key,

Grade = st.**Value**

});

context.SaveChanges();

}

**return** testing;

}

**private** **static** TestingViewModel CreateModel(Testing testing)

{

TestingViewModel model = new TestingViewModel();

model.Id = testing.Id;

model.PlanId = testing.PlanId;

model.Topic = testing.Topic;

model.Type = testing.Type;

model.Hours = testing.Hours;

model.Date = testing.Date;

model.StudentTestings = new List<Tuple<**int**, **string**, MarkType>>();

**foreach**(**var** item **in** testing.StudentTestings)

{

model.StudentTestings.**Add**(new Tuple<**int**, **string**, MarkType>(item.StudentId, item.Student.Flm, item.Grade));

}

**return** model;

}

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**using** UniversityContracts.Enums;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Student

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** NumFB { **get**; **set**; }

[Required]

**public** **string** Flm { **get**; **set**; }

**public** TypeEducationBasement Basement { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **virtual** **Group** **Group** { **get**; **set**; }

[ForeignKey("StudentId")]

**public** **virtual** List<StudentTesting> StudentTestings { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Teacher

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Flm { **get**; **set**; }

[Required]

**public** **string** Login { **get**; **set**; }

[Required]

**public** **string** Password { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **virtual** Department Department { **get**; **set**; }

[ForeignKey("TeacherId")]

**public** **virtual** List<TeacherDiscipline> TeacherDisciplines { **get**; **set**; }

[ForeignKey("TeacherId")]

**public** **virtual** List<Plan> Plans { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** UniversityContracts.Enums;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** StudentTesting

{

**public** **int** Id { **get**; **set**; }

**public** **int** TestingId { **get**; **set**; }

**public** **int** StudentId { **get**; **set**; }

[Required]

**public** MarkType Grade { **get**; **set**; }

**public** **virtual** Testing Testing { **get**; **set**; }

**public** **virtual** Student Student { **get**; **set**; }

}

}

﻿**namespace** UniversityDatabaseImplement.Models

{

**public** **class** TeacherDiscipline

{

**public** **int** Id { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **virtual** Teacher Teacher { **get**; **set**; }

**public** **virtual** Discipline Discipline { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**using** UniversityContracts.Enums;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Testing

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Topic { **get**; **set**; }

[Required]

**public** **int** Hours { **get**; **set**; }

[Required]

**public** DateTime Date { **get**; **set**; }

[Required]

**public** PlanType Type { **get**; **set**; }

[Required]

**public** **int** PlanId { **get**; **set**; }

**public** **virtual** Plan Plan { **get**; **set**; }

[ForeignKey("TestingId")]

**public** **virtual** List<StudentTesting> StudentTestings { **get**; **set**;}

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Department

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Name { **get**; **set**; }

[Required]

**public** **string** Login { **get**; **set**; }

[Required]

**public** **string** Password { **get**; **set**; }

[ForeignKey("DepartmentId")]

**public** **virtual** List<Teacher> Teachers { **get**; **set**; }

[ForeignKey("DepartmentId")]

**public** **virtual** List<Discipline> Disciplines { **get**; **set**; }

[ForeignKey("DepartmentId")]

**public** **virtual** List<Plan> Plans { **get**; **set**; }

[ForeignKey("DepartmentId")]

**public** **virtual** List<**Group**> Groups { **get**; **set**; }

[ForeignKey("DepartmentId")]

**public** **virtual** List<Message> Messages { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Discipline

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Name { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **virtual** Department Department { **get**; **set**; }

[ForeignKey("DisciplineId")]

**public** **virtual** List<TeacherDiscipline> TeacherDisciplines { **get**; **set**; }

[ForeignKey("DisciplineId")]

**public** **virtual** List<Plan> Plans { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** **Group**

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Name { **get**; **set**; }

[Required]

**public** **int** Course { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **virtual** Department Department { **get**; **set**; }

[ForeignKey("GroupId")]

**public** **virtual** List<Student> Students { **get**; **set**; }

[ForeignKey("GroupId")]

**public** **virtual** List<Plan> Plans { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** UniversityContracts.Enums;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Message

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** ReportTypes ReportType { **get**; **set**; }

[Required]

**public** Status Status { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **virtual** Department Department { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**using** UniversityContracts.Enums;

**namespace** UniversityDatabaseImplement.Models

{

**public** **class** Plan

{

**public** **int** Id { **get**; **set**; }

[Required]

**public** **string** Name { **get**; **set**; }

[Required]

**public** **int** Hours { **get**; **set**; }

[Required]

**public** PlanType Type { **get**; **set**; }

[Required]

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **virtual** Department Department { **get**; **set**; }

**public** **virtual** Teacher Teacher { **get**; **set**; }

**public** **virtual** Discipline Discipline { **get**; **set**; }

**public** **virtual** **Group** **Group** { **get**; **set**; }

[ForeignKey("PlanId")]

**public** **virtual** List<Testing> Testings { **get**; **set**; }

}

}

﻿**using** Microsoft.EntityFrameworkCore;

**using** UniversityDatabaseImplement.Models;

**namespace** UniversityDatabaseImplement

{

**public** **class** UniversityDatabase : DbContext

{

**protected** **override** **void** OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

**if** (optionsBuilder.IsConfigured == **false**)

{

optionsBuilder.UseSqlServer(@"Data Source=ArutunyanWin64\SQLEXPRESS;Initial Catalog=UniversityDatabase;Integrated Security=True;MultipleActiveResultSets=True;");

}

**base**.OnConfiguring(optionsBuilder);

}

**public** **virtual** DbSet<Department> Departments { **set**; **get**; }

**public** **virtual** DbSet<Teacher> Teachers { **get**; **set**; }

**public** **virtual** DbSet<TeacherDiscipline> TeacherDisciplines { **get**; **set**; }

**public** **virtual** DbSet<Discipline> Disciplines { **set**; **get**; }

**public** **virtual** DbSet<Plan> Plans { **set**; **get**; }

**public** **virtual** DbSet<**Group**> Groups { **set**; **get**; }

**public** **virtual** DbSet<Student> Students { **get**; **set**; }

**public** **virtual** DbSet<StudentTesting> StudentTestings { **get**; **set**; }

**public** **virtual** DbSet<Testing> Testings { **get**; **set**; }

**public** **virtual** DbSet<Message> Messages { **get**; **set**; }

}

}

Приложение ASP .NET Core Rest API UniversityWebApi

**using** DocumentFormat.OpenXml.Office2010.Excel;

**using** Microsoft.AspNetCore.Mvc;

**using** UniversityBusinessLogic.MailWorker;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.Enums;

**using** UniversityContracts.ViewModels;

**namespace** UniversityWebApi.Controllers

{

[ApiController]

[Route("api/[controller]/[action]")]

**public** **class** MainController : ControllerBase

{

**private** **readonly** IDepartmentLogic \_department;

**private** **readonly** IDisciplineLogic \_discipline;

**private** **readonly** IGroupLogic \_group;

**private** **readonly** IPlanLogic \_plan;

**private** **readonly** IStudentLogic \_student;

**private** **readonly** ITeacherLogic \_teacher;

**private** **readonly** ITestingLogic \_testing;

**private** **readonly** IMessageLogic \_message;

**private** **readonly** IReportLogic \_report;

**private** **readonly** AbstractMailWorker \_mailWorker;

**private** **readonly** **int** objectsOnPage = 4;

**private** **int** NumOfPages;

**public** MainController(IDepartmentLogic department, IDisciplineLogic discipline,

IGroupLogic **group**, IPlanLogic plan,

IStudentLogic student, ITeacherLogic teacher,

ITestingLogic testing, IMessageLogic message,

IReportLogic report, AbstractMailWorker mailWorker)

{

\_department = department;

\_discipline = discipline;

\_group = **group**;

\_plan = plan;

\_student = student;

\_teacher = teacher;

\_testing = testing;

\_message = message;

\_report = report;

\_mailWorker = mailWorker;

**if** (objectsOnPage < 1) { objectsOnPage = 5; }

\_report = report;

}

#region COMMON METHODS

*//------------------------------Filtered Lists----------------------------*

[HttpGet]

**public** List<DisciplineViewModel> GetDisciplines(**int** departmentId) => \_discipline.Read(new DisciplineBindingModel

{ DepartmentId = departmentId });

[HttpGet]

**public** List<GroupViewModel> GetGroups(**int** departmentId) => \_group.Read(new GroupBindingModel

{ DepartmentId = departmentId });

[HttpGet]

**public** List<StudentViewModel> GetStudents(**int** groupId) => \_student.Read(new StudentBindingModel

{ GroupId = groupId });

*//------------------------------Filtered Lists----------------------------*

*//------------------------------ Elements ----------------------------*

[HttpGet]

**public** DepartmentViewModel? GetDepartment(**int** id, **string**? login, **string**? password) => \_department.Read(new DepartmentBindingModel

{ Id = id, Login = login, Password = password })?[0];

[HttpGet]

**public** TeacherViewModel? GetTeacher(**int** id, **string**? login, **string**? password) => \_teacher.Read(new TeacherBindingModel

{ Id = id, Login = login, Password = password })?[0];

[HttpGet]

**public** DisciplineViewModel? GetDiscipline(**int** id) => \_discipline.Read(new DisciplineBindingModel

{ Id = id })?[0];

[HttpGet]

**public** GroupViewModel? GetGroup(**int** id) => \_group.Read(new GroupBindingModel

{ Id = id })?[0];

[HttpGet]

**public** StudentViewModel? GetStudent(**int** id) => \_student.Read(new StudentBindingModel

{ Id = id })?[0];

[HttpGet]

**public** PlanViewModel? GetPlan(**int** id) => \_plan.Read(new PlanBindingModel

{ Id = id })?[0];

[HttpGet]

**public** MessageViewModel? GetMessage(**int** id) => \_message.Read(new MessageBindingModel

{ Id = id })?[0];

*//------------------------------ Elements ----------------------------*

#endregion

#region METHODS USED AS TEACHER CONTROLLER METHODS ONLY

*//------------------------------Filtered Lists----------------------------*

[HttpGet]

**public** List<PlanViewModel> GetTeachertPlans(**int** teacherId, **int** disciplineId) => \_plan.Read(new PlanBindingModel

{ TeacherId = teacherId, DisciplineId = disciplineId });

[HttpGet]

**public** List<PlanViewModel> GetFilteredPlans(**int** teacherId, **int** disciplineId, **int** groupId, PlanType type) => \_plan.Read(new PlanBindingModel

{

TeacherId = teacherId,

DisciplineId = disciplineId,

GroupId = groupId,

Type = type

});

[HttpGet]

**public** List<Tuple<**int**, **string**, **int**, DateTime>> GetTestingsToday(**int** teacherId)

{

**var** plans = \_plan.Read(new PlanBindingModel { TeacherId = teacherId });

List<Tuple<**int**, **string**, **int**, DateTime>> result = new List<Tuple<**int**, **string**, **int**, DateTime>>();

**foreach** (**var** plan **in** plans)

{

**foreach** (**var** testing **in** plan.Testings)

{

**if** (testing.Item4.Date == DateTime.Now.Date)

{

result.**Add**(testing);

}

}

}

**return** result;

}

[HttpGet]

**public** Tuple<List<PlanViewModel>, **int**> GetPaginatedPlans(**int** teacherId, **int** page)

{

**var** fullList = \_plan.Read(**null**);

NumOfPages = fullList.Count / objectsOnPage;

**if** (fullList.Count % objectsOnPage != 0) { NumOfPages++; }

**var** list = \_plan.Read(new PlanBindingModel { TeacherId = teacherId, ToSkip = (page - 1) \* objectsOnPage, ToTake = objectsOnPage }).ToList();

**var** result = new Tuple<List<PlanViewModel>, **int**>(list.Take(objectsOnPage).ToList(), NumOfPages);

**return** result;

}

[HttpGet]

**public** List<TestingViewModel> GetTestings(**int** planId) => \_testing.Read(new TestingBindingModel

{ PlanId = planId });

[HttpGet]

**public** List<MessageViewModel> GetTeacherMessages(**int** id) => \_message.Read(new MessageBindingModel

{ TeacherId = id });

[HttpGet]

**public** List<MessageViewModel> GetMessagesByStatus(**int** id, Status status) => \_message.Read(new MessageBindingModel

{ TeacherId = id, Status = status });

[HttpGet]

**public** List<MessageViewModel> GetMessagesByStatusAndType(**int** id, ReportTypes typeReport, Status status) => \_message.Read(new MessageBindingModel

{ TeacherId = id, Status = status, ReportType = typeReport });

*//------------------------------Filtered Lists----------------------------*

*//------------------------------ Elements ----------------------------*

[HttpGet]

**public** TestingViewModel? GetTesting(**int** id) => \_testing.Read(new TestingBindingModel

{ Id = id })?[0];

[HttpGet]

**public** TestingViewModel? GetTestingByPlanAndName(**int** planId, **string** topic) => \_testing.Read(new TestingBindingModel

{ Topic = topic, PlanId = planId })?[0];

*//------------------------------ Elements ----------------------------*

*//------------------------------ C-U ----------------------------*

[HttpPost]

**public** **void** CreateOrUpdateTesting(TestingBindingModel model) => \_testing.CreateOrUpdate(model);

*//------------------------------ C-U ----------------------------*

*//------------------------------ Put ----------------------------*

[HttpPost]

**public** **void** AnswerRequest(MessageBindingModel model) => \_message.AnswerRequest(model);

[HttpGet]

**public** **bool** AddGroupToTesting(**int** testingId, **int** groupId)

{

**var** students = \_student.Read(new StudentBindingModel { GroupId = groupId });

**foreach** (**var** student **in** students)

{

\_testing.AddStudent(new StudentTestingBindingModel

{

TestingId = testingId,

StudentId = student.Id,

Grade = MarkType.Нет

});

}

**return** **true**;

}

[HttpPost]

**public** **void** PutStudentMarks(List<StudentTestingBindingModel> studentGrades)

{

**foreach** (**var** studentGrade **in** studentGrades)

{

\_testing.AddStudent(studentGrade);

}

}

*//------------------------------ Put ----------------------------*

*//------------------------------ D ----------------------------*

[HttpGet]

**public** **bool** DeleteTesting(**int** id)

{

**var** testing = \_testing.Read(new TestingBindingModel() { Id = id })?[0];

\_testing.CreateOrUpdate(new TestingBindingModel()

{

Id = id,

PlanId = testing.PlanId,

StudentTestings = new Dictionary<**int**, MarkType>()

});

\_testing.Delete(new TestingBindingModel() { Id = id });

**return** **true**;

}

*//------------------------------ D ----------------------------*

*//------------------------------ Reports ----------------------------*

[HttpGet]

**public** ReportPlanViewModel GetObjectsForPlanReport(**int** id)

{

**var** tmp = \_message.Read(new MessageBindingModel { Id = id })[0];

MessageBindingModel model = new MessageBindingModel

{

Id = tmp.Id,

ReportType = tmp.ReportType,

Status = tmp.Status,

DepartmentId = tmp.DepartmentId,

PlanId = tmp.PlanId,

TeacherId = tmp.TeacherId,

DisciplineId = tmp.DisciplineId

};

**return** \_report.GetObjectsForPlanReport(model);

}

[HttpGet]

**public** ReportFullViewModel GetObjectsForSumReport(**int** id)

{

**var** tmp = \_message.Read(new MessageBindingModel { Id = id })[0];

MessageBindingModel model = new MessageBindingModel

{

Id = tmp.Id,

ReportType = tmp.ReportType,

Status = tmp.Status,

DepartmentId = tmp.DepartmentId,

PlanId = tmp.PlanId,

TeacherId = tmp.TeacherId,

DisciplineId = tmp.DisciplineId

};

**return** \_report.GetObjectsForSumReport(model);

}

[HttpPost]

**public** **void** SavePlanReport(Tuple<MessageBindingModel, **string**> model) => \_report.SavePlanReport(model.Item1, model.Item2);

[HttpPost]

**public** **void** SaveSumReport(Tuple<MessageBindingModel, **string**> model) => \_report.SaveSumReport(model.Item1, model.Item2);

*//------------------------------ Reports ----------------------------*

*//------------------------------ Mail ----------------------------*

[HttpPost]

**public** **void** SendMessage(MailSendInfoBindingModel model) => \_mailWorker.MailSendAsync(model);

*//------------------------------ Mail ----------------------------*

#endregion

#region METHODS USED AS DEPARTMENT CONTROLLER METHODS ONLY

*//------------------------------Filtered Lists----------------------------*

[HttpGet]

**public** List<DepartmentViewModel> GetAllDepartments() => \_department.Read(**null**);

[HttpGet]

**public** List<TeacherViewModel> GetTeachers(**int** departmentId) => \_teacher.Read(new TeacherBindingModel

{ DepartmentId = departmentId });

[HttpGet]

**public** List<PlanViewModel> GetDepartmentPlans(**int** departmentId) => \_plan.Read(new PlanBindingModel

{ DepartmentId = departmentId });

[HttpGet]

**public** List<MessageViewModel> GetDepartmentMessages(**int** id) => \_message.Read(new MessageBindingModel

{ DepartmentId = id });

[HttpGet]

**public** List<StudentViewModel> GetStudentsByDepartment(**int** departmentId) => \_student.Read(new StudentBindingModel

{ DepartmentId = departmentId });

[HttpGet]

**public** List<PlanViewModel> GetPlansByDiscipline(**int** disciplineId)

{

**return** \_plan.GetPlansByDiscipline(disciplineId);

}

[HttpGet]

**public** List<TeacherViewModel> GetTeacherByDiscipline(**int** disciplineId)

{

**return** \_teacher.GetTeacherByDiscipline(disciplineId);

}

*//------------------------------Filtered Lists----------------------------*

*//------------------------------ Element ----------------------------*

[HttpGet]

**public** DepartmentViewModel AuthorizationDepartment(**string** login, **string** password) => \_department.Authorization(login, password);

[HttpGet]

**public** List<GroupViewModel> GetGroupsByName(**string** groupName, **int** departmentId) => \_group.Read(new GroupBindingModel

{

DepartmentId = departmentId,

Name = groupName

});

[HttpGet]

**public** List<DisciplineViewModel> GetDisciplinesByName(**string** disciplineName, **int** departmentId) => \_discipline.Read(new DisciplineBindingModel

{

DepartmentId = departmentId,

Name = disciplineName

});

*//------------------------------ Element ----------------------------*

*//------------------------------ C-U ----------------------------*

[HttpPost]

**public** **void** CreateOrUpdateDepartment(DepartmentBindingModel model) => \_department.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateOrUpdateTeacher(TeacherBindingModel model) => \_teacher.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateOrUpdateDiscipline(DisciplineBindingModel model) => \_discipline.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateOrUpdateGroup(GroupBindingModel model) => \_group.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateOrUpdateStudent(StudentBindingModel model) => \_student.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateOrUpdatePlan(PlanBindingModel model) => \_plan.CreateOrUpdate(model);

[HttpPost]

**public** **void** CreateMessage(MessageBindingModel model) => \_message.CreateMessage(model);

[HttpPost]

**public** **void** SendGroupReport(Tuple<**int**> id) => \_report.SendGroupReport(id.Item1);

*//------------------------------ C-U ----------------------------*

*//------------------------------ Put ----------------------------*

[HttpGet]

**public** **bool** AddTeachersToDiscipline(**int** disciplineId, **int** teacherId)

{

\_teacher.AddDiscipline(new TeacherDisciplineBindingModel

{

TeacherId = teacherId,

DisciplineId = disciplineId

});

**return** **true**;

}

[HttpPut]

**public** **void** RemoveTeacerFromDiscipline(TeacherDisciplineBindingModel model) => \_teacher.RemoveDiscipline(model);

[HttpPost]

**public** **void** CloseRequest(MessageBindingModel model) => \_message.CloseRequest(model);

[HttpPost]

**public** **void** MessageRollBack(MessageBindingModel model) => \_message.MessageRollBack(model);

*//------------------------------ Put ----------------------------*

*//------------------------------ D ----------------------------*

[HttpGet]

**public** **bool** DeletePlan(**int** id)

{

\_plan.Delete(new PlanBindingModel() { Id = id });

**return** **true**;

}

[HttpGet]

**public** **bool** DeleteStudent(**int** id)

{

\_student.Delete(new StudentBindingModel

{

Id = id

});

**return** **true**;

}

[HttpGet]

**public** **bool** DeleteTeacher(**int** id)

{

TeacherViewModel teacher = \_teacher.Read(new TeacherBindingModel

{

Id = id

})[0];

TeacherBindingModel model = new TeacherBindingModel

{

Id = teacher.Id,

DepartmentId = teacher.DepartmentId,

Flm = teacher.Flm,

Login = teacher.Login,

Password = teacher.Password,

TeacherDisciplines = teacher.TeacherDisciplines.Keys.ToList()

};

**if** (model.TeacherDisciplines.Count() > 0)

{

**foreach** (**int** teacherDiscipline **in** model.TeacherDisciplines)

{

\_teacher.RemoveDiscipline(new TeacherDisciplineBindingModel

{

TeacherId = model.Id.**Value**,

DisciplineId = teacherDiscipline

});

}

}

List<PlanViewModel> listPlans = \_plan.Read(new PlanBindingModel

{

DepartmentId = model.DepartmentId,

});

**if** (listPlans.Count() > 0)

{

**foreach** (PlanViewModel plan **in** listPlans)

{

**if** (plan.TeacherId == model.Id)

{

\_plan.Delete(new PlanBindingModel

{

Id = plan.Id

});

}

}

}

\_teacher.Delete(model);

**return** **true**;

}

[HttpGet]

**public** **bool** DeleteDiscipline(**int** id)

{

DisciplineViewModel discipline = \_discipline.Read(new DisciplineBindingModel

{

Id = id

})[0];

DisciplineBindingModel model = new DisciplineBindingModel

{

Id = discipline.Id,

DepartmentId = discipline.DepartmentId,

Name = discipline.Name

};

List<PlanViewModel> listPlans = \_plan.Read(new PlanBindingModel

{

DepartmentId = model.DepartmentId,

});

**if** (listPlans.Count() > 0)

{

**foreach** (PlanViewModel plan **in** listPlans)

{

**if** (plan.DisciplineId == model.Id)

{

\_plan.Delete(new PlanBindingModel

{

Id = plan.Id

});

}

}

}

List<TeacherViewModel> teachersDiscipline = \_teacher.GetTeacherByDiscipline(model.Id.**Value**);

**if** (teachersDiscipline.Count() > 0)

{

**foreach** (TeacherViewModel teacherDiscipline **in** teachersDiscipline)

{

\_teacher.RemoveDiscipline(new TeacherDisciplineBindingModel

{

TeacherId = teacherDiscipline.Id,

DisciplineId = model.Id.**Value**

});

}

}

\_discipline.Delete(new DisciplineBindingModel

{

Id = id

});

**return** **true**;

}

[HttpGet]

**public** **bool** DeleteGroup(**int** id)

{

GroupViewModel **group** = \_group.Read(new GroupBindingModel

{

Id = id

})[0];

GroupBindingModel model = new GroupBindingModel

{

Id = **group**.Id,

DepartmentId = **group**.DepartmentId,

Name = **group**.Name,

Course = **group**.Course

};

List<PlanViewModel> listPlans = \_plan.Read(new PlanBindingModel

{

DepartmentId = model.DepartmentId,

});

**if** (listPlans.Count() > 0)

{

**foreach** (PlanViewModel plan **in** listPlans)

{

**if** (plan.GroupId == model.Id)

{

\_plan.Delete(new PlanBindingModel

{

Id = plan.Id

});

}

}

}

\_group.Delete(model);

**return** **true**;

}

[HttpGet]

**public** **bool** DeleteMessage(**int** id)

{

\_message.Delete(new MessageBindingModel

{

Id = id

});

**return** **true**;

}

*//------------------------------ D ----------------------------*

#endregion

}

}

**using** DocumentFormat.OpenXml.Office2016.Drawing.ChartDrawing;

**using** M6T.Core.TupleModelBinder;

**using** Microsoft.Extensions.Configuration;

**using** Microsoft.OpenApi.Models;

**using** UniversityBusinessLogic.BusinessLogic;

**using** UniversityBusinessLogic.MailWorker;

**using** UniversityBusinessLogic.OfficePackage;

**using** UniversityBusinessLogic.OfficePackage.Implements;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.BusinessLogicContracts;

**using** UniversityContracts.StorageContracts;

**using** UniversityDatabaseImplement.Implements;

**var** builder = WebApplication.CreateBuilder(args);

*// Add services to the container.*

*// Services connection*

builder.Services.AddTransient<IDepartmentStorage, DepartmentStorage>();

builder.Services.AddTransient<IDisciplineStorage, DisciplineStorage>();

builder.Services.AddTransient<IGroupStorage, GroupStorage>();

builder.Services.AddTransient<IPlanStorage, PlanStorage>();

builder.Services.AddTransient<IStudentStorage, StudentStorage>();

builder.Services.AddTransient<ITeacherStorage, TeacherStorage>();

builder.Services.AddTransient<ITestingStorage, TestingStorage>();

builder.Services.AddTransient<IMessageStorage, MessageStorage>();

builder.Services.AddTransient<IDepartmentLogic, DepartmentLogic>();

builder.Services.AddTransient<IDisciplineLogic, DisciplineLogic>();

builder.Services.AddTransient<IGroupLogic, GroupLogic>();

builder.Services.AddTransient<IPlanLogic, PlanLogic>();

builder.Services.AddTransient<IStudentLogic, StudentLogic>();

builder.Services.AddTransient<ITeacherLogic, TeacherLogic>();

builder.Services.AddTransient<ITestingLogic, TestingLogic>();

builder.Services.AddTransient<IMessageLogic, MessageLogic>();

builder.Services.AddTransient<IReportLogic, ReportLogic>();

builder.Services.AddTransient<AbstractSaveToWord, SaveToWord>();

builder.Services.AddSingleton<AbstractMailWorker, MailKitWorker>();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

*// App builder options*

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "UniversityWebApi", Version = "v1" });

});

builder.Services.AddMvc(options =>

{

options.ModelBinderProviders.Insert(0, new TupleModelBinderProvider());

});

**var** app = builder.Build();

*// Configure the HTTP request pipeline.*

**if** (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "UniversityWebApi v1"));

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.MapControllers();

**var** mailSender = app.Services.GetService<AbstractMailWorker>();

**var** config = new ConfigurationBuilder().AddJsonFile("appsettings.json").Build();

mailSender.MailConfig(new MailConfigBindingModel

{

MailLogin = config?.GetSection("MailLogin")?.**Value**.ToString(),

MailPassword = config?.GetSection("MailPassword")?.**Value**.ToString(),

SmtpClientHost = config?.GetSection("SmtpClientHost")?.**Value**.ToString(),

SmtpClientPort = Convert.ToInt32(config?.GetSection("SmtpClientPort")?.**Value**.ToString()),

PopHost = config?.GetSection("PopHost")?.**Value**.ToString(),

PopPort = Convert.ToInt32(config?.GetSection("PopPort")?.**Value**.ToString())

});

app.Run();

Библиотека классов UniversityContracts

**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** StudentBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int**? DepartmentId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string**? NumFB { **get**; **set**; }

**public** TypeEducationBasement Basement { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** TestingBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** Topic { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** DateTime Date { **get**; **set**; }

**public** PlanType? Type { **get**; **set**; }

**public** Dictionary<**int**, MarkType>? StudentTestings { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** StudentTestingBindingModel

{

**public** **int** TestingId { **get**; **set**; }

**public** **int** StudentId { **get**; **set**; }

**public** MarkType Grade { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** DepartmentBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **string**? Name { **get**; **set**; }

**public** **string**? Login { **get**; **set**; }

**public** **string**? Password { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** TeacherBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string**? Login { **get**; **set**; }

**public** **string**? Password { **get**; **set**; }

**public** List<**int**> TeacherDisciplines { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** TeacherDisciplineBindingModel

{

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** DisciplineBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** GroupBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Course { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** MailConfigBindingModel

{

**public** **string** MailLogin { **get**; **set**; }

**public** **string** MailPassword { **get**; **set**; }

**public** **string** SmtpClientHost { **get**; **set**; }

**public** **int** SmtpClientPort { **get**; **set**; }

**public** **string** PopHost { **get**; **set**; }

**public** **int** PopPort { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.BindingModels

{

**public** **class** MailSendInfoBindingModel

{

**public** **string** MailAddress { **get**; **set**; }

**public** **string** Subject { **get**; **set**; }

**public** **string** Text { **get**; **set**; }

**public** **string** FilePath { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** MessageBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** ReportTypes? ReportType { **get**; **set**; }

**public** Status? Status { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int**? PlanId { **get**; **set**; }

**public** **int**? TeacherId { **get**; **set**; }

**public** **int**? DisciplineId { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.BindingModels

{

**public** **class** PlanBindingModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** PlanType? Type { **get**; **set**; }

**public** **int**? ToSkip { **get**; **set**; }

**public** **int**? ToTake { **get**; **set**; }

}

}

﻿**using** System.ComponentModel.DataAnnotations;

**namespace** UniversityContracts.Enums

{

**public** **enum** ReportTypes

{

[Display(Name = "Итоговый отчёт по дисциплине")]

SumReport = 0,

[Display(Name = "Отчёт по плану")]

PlanReport = 1

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** Status

{

Активен = 0,

Проверяется = 1,

Проверен = 2

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** MarkType

{

Нет = 0,

*// оценки*

Неуд = 2,

Удовл = 3,

Хор = 4,

Отл = 5,

*// посещаемость*

НП = 6,

УП = 7,

П = 8

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** TypeEducationBasement

{

Б = 0,

ПО = 1,

ЦП = 3

}

}

﻿**namespace** UniversityContracts.Enums

{

**public** **enum** PlanType

{

Лабораторный = 0,

Лекционный = 1,

Практический = 2,

Семинарный = 3,

Консультационный = 4,

Проверочный = 5

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** ReportFullViewModel

{

**public** **string** Title { **get**; **set**; }

**public** List<**string**> Footer { **get**; **set**; }

**public** List<**string**> DisciplineName { **get**; **set**; }

**public** List<Tuple<List<**string**>>> Itog { **get**; **set**; }

**public** List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, List<Tuple<**string**, MarkType>>>>>>>> Items { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** ReportPlanViewModel

{

**public** **string** Title { **get**; **set**; }

**public** List<**string**> Footer { **get**; **set**; }

**public** List<**string**> PlanName { **get**; **set**; }

**public** List<**string**> Itog { **get**; **set**; }

**public** List<Tuple<**string**, **string**, List<Tuple<**string**, MarkType>>>> Items { **get**; **set**;}

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** StudentViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string** NumFB { **get**; **set**; }

**public** TypeEducationBasement Basement { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** TestingViewModel

{

**public** **int**? Id { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** Topic { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** DateTime Date { **get**; **set**; }

**public** PlanType Type { **get**; **set**; }

**public** List<Tuple<**int**, **string**, MarkType>> StudentTestings { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** TeacherViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Flm { **get**; **set**; }

**public** **string** Login { **get**; **set**; }

**public** **string** Password { **get**; **set**; }

**public** Dictionary<**int**, **string**> TeacherDisciplines { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** DepartmentViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **string** Login { **get**; **set**; }

**public** **string** Password { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** DisciplineViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

}

}

﻿**namespace** UniversityContracts.ViewModels

{

**public** **class** GroupViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Course { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** MessageViewModel

{

**public** **int** Id { **get**; **set**; }

**public** ReportTypes ReportType { **get**; **set**; }

**public** Status Status { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** PlanId { **get**; **set**; }

**public** **string** PlanName { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **string** DisciplineName { **get**; **set**; }

}

}

﻿**using** UniversityContracts.Enums;

**namespace** UniversityContracts.ViewModels

{

**public** **class** PlanViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **int** DepartmentId { **get**; **set**; }

**public** **int** TeacherId { **get**; **set**; }

**public** **int** DisciplineId { **get**; **set**; }

**public** **string** DisciplineName { **get**; **set**; }

**public** **int** GroupId { **get**; **set**; }

**public** **string** GroupName { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** PlanType Type { **get**; **set**; }

**public** List<Tuple<**int**, **string**, **int**, DateTime>> Testings { **get**; **set**; }

}

}

Приложение ASP .NET Core Web-Api TeacherApp

**using** Microsoft.AspNetCore.Mvc;

**using** Microsoft.AspNetCore.Mvc.RazorPages;

**using** Newtonsoft.Json;

**using** System.Text;

**using** System.Text.RegularExpressions;

**using** UniversityContracts.BindingModels;

**using** UniversityContracts.Enums;

**using** UniversityContracts.ViewModels;

**namespace** TeacherApp.Controllers

{

**public** **class** HomeController : Controller

{

**private** **readonly** **string** rootPath = "D:**\\**ULSTU**\\**ПиАПС**\\**Курсовая**\\**PROJECT**\\**TeacherApp";

**private** **readonly** ILogger<HomeController> \_logger;

**protected** **readonly** IWebHostEnvironment \_hostEnvironment;

**protected** **static** **string** error = "";

**protected** **static** **string** unxpError = "";

**protected** **static** **string** success = "";

**public** HomeController(ILogger<HomeController> logger, IWebHostEnvironment webHostEnvironment)

{

\_logger = logger;

\_hostEnvironment = webHostEnvironment;

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

}

*//------------------------------ SignUp -------------------------------*

**public** IActionResult SignUp()

{

**if** (error != "")

{

ViewData["Error"] = error;

error = "";

}

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**return** View();

}

[HttpPost]

**public** **void** TeacherSignUp(**string** Login, **string** Password)

{

**try**

{

**if** (!**string**.IsNullOrEmpty(Login) && !**string**.IsNullOrEmpty(Password))

{

**if** (Login.Any(ch => ! **char**.IsLetterOrDigit(ch)))

{

error = "В логине не может содержаться спецсимволов";

Response.Redirect("SignUp");

**return**;

}

**if** (!Regex.IsMatch(Password, @"^((\w+\d+\W+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]\*$"))

{

error = "Пароль должен содержать в себе цифры, буквы и небуквенные символы";

Response.Redirect("SignUp");

**return**;

}

APIClient.teacher = APIClient.GetRequest<TeacherViewModel>($"api/Main/GetTeacher?login={Login}&password={Password}");

Response.Redirect("Index");

**return**;

}

**else**

{

Response.Redirect("SignUp");

**return**;

}

}

**catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

Response.Redirect("SignUp");

**return**;

}

}

**public** IActionResult Exit()

{

APIClient.teacher = **null**;

**return** Redirect("SignUp");

}

*//------------------------------ SignUp -------------------------------*

*//----------------------------- HomePage ------------------------------*

[HttpGet]

**public** IActionResult Index()

{

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

DepartmentViewModel d = APIClient.GetRequest<DepartmentViewModel>

($"api/Main/GetDepartment?id={APIClient.teacher.DepartmentId}&login={null}&password={null}");

**var** messages = APIClient.GetRequest<List<MessageViewModel>>(

$"api/Main/GetMessagesByStatus?id={APIClient.teacher.Id}&status={Status.Активен}");

**var** testings = APIClient.GetRequest<List<Tuple<**int**, **string**, **int**, DateTime>>>

($"api/Main/GetTestingsToday?teacherId={APIClient.teacher.Id}");

**var** model = (d.Name, APIClient.teacher.Flm, messages, testings);

**return** View(model);

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

*//----------------------------- HomePage ------------------------------*

*//------------------------------ EdPlan -------------------------------*

[HttpGet]

**public** IActionResult Plan(**int**? disciplineId)

{

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

List<PlanViewModel> plans = new List<PlanViewModel>();

**if** (disciplineId == **null**)

{

**if** (APIClient.teacher.TeacherDisciplines.Count() != 0)

{

plans = APIClient.GetRequest<List<PlanViewModel>>

($"api/Main/GetTeachertPlans?teacherId={APIClient.teacher.Id}&disciplineId={APIClient.teacher.TeacherDisciplines.Keys.ToList()[0]}");

}

}

**else**

{

plans = APIClient.GetRequest<List<PlanViewModel>>

($"api/Main/GetTeachertPlans?teacherId={APIClient.teacher.Id}&disciplineId={disciplineId}");

}

**var** model = (plans, APIClient.teacher.TeacherDisciplines);

**return** View(model);

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

[HttpGet]

**public** IActionResult Testing(**int** groupId, **int** planId, **int**? testingId)

{

**if** (error != "")

{

ViewData["Error"] = error;

error = "";

}

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

TestingViewModel testing = new TestingViewModel();

**if** (testingId != **null**)

{

testing = APIClient.GetRequest<TestingViewModel>(

$"api/Main/GetTesting?id={testingId}");

}

**var** model = (groupId, planId, testing);

**return** View(model);

} **catch**(Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

[HttpPost]

**public** **void** UpdateTesting(**int**? Id, **int** GroupId, **int** PlanId, **string** Topic, [Bind("PlanId, Topic, Hours, Date")] TestingBindingModel model)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

Response.Redirect("SignUp");

**return**;

}

**try**

{

**if** (**string**.IsNullOrEmpty(model.Topic))

{

error = "Название темы не может быть пустым";

**if** (Id == **null**)

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}");

} **else**

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}&testingId={Id}");

}

**return**;

}

**if** (model.Hours == 0)

{

error = "Укажите корректную продолжительность занятия";

**if** (Id == **null**)

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}");

}

**else**

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}&testingId={Id}");

}

**return**;

}

PlanViewModel plan = APIClient.GetRequest<PlanViewModel>(

$"api/Main/GetPlan?id={PlanId}");

**int** sum = 0;

**foreach** (**var** item **in** plan.Testings)

{

**if**(Id == **null**)

{

sum += item.Item3;

} **else**

{

**if**(item.Item1 != Id)

{

sum += item.Item3;

}

}

}

**if** ((model.Hours + sum) <= plan.Hours)

{

**if** (Id == **null**)

{

APIClient.PostRequest($"api/Main/CreateOrUpdateTesting", model);

**var** testing = APIClient.GetRequest<TestingViewModel>(

$"api/Main/GetTestingByPlanAndName?planId={PlanId}&topic={Topic}");

APIClient.GetRequest<**bool**>(

$"api/Main/AddGroupToTesting?testingId={testing.Id}&groupId={GroupId}");

}

**else**

{

model.Id = Id;

APIClient.PostRequest($"api/Main/CreateOrUpdateTesting", model);

}

} **else**

{

error = "Продолжительность занятия не может превышать продолжительность курса";

**if** (Id == **null**)

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}");

}

**else**

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}&testingId={Id}");

}

**return**;

}

Response.Redirect("Plan");

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**if** (Id == **null**)

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}");

}

**else**

{

Response.Redirect($"Testing?groupId={GroupId}&planId={PlanId}&testingId={Id}");

}

**return**;

}

}

[HttpGet]

**public** **void** DeleteTesting(**int** id, **int** planId)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

Response.Redirect("SignUp");

**return**;

}

**try**

{

APIClient.GetRequest<**bool**>($"api/Main/DeleteTesting?id={id}");

Response.Clear();

Response.Redirect($"/Home/Plan");

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

Response.Redirect("Plan");

**return**;

}

}

*//------------------------------ EdPlan -------------------------------*

*//---------------------------- Plan&Marks -----------------------------*

[HttpGet]

**public** IActionResult Plans(**string**? serializedList, **int** page = 1)

{

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

ViewBag.Disciplines = APIClient.GetRequest<List<DisciplineViewModel>>($"api/Main/GetDisciplines?departmentId={APIClient.teacher.DepartmentId}");

ViewBag.Groups = APIClient.GetRequest<List<GroupViewModel>>($"api/Main/GetGroups?departmentId={APIClient.teacher.DepartmentId}");

**if** (serializedList != **null**)

{

List<PlanViewModel>? list = JsonConvert.DeserializeObject<List<PlanViewModel>>(serializedList);

**return** View((list, 1, 1));

}

**var** tmp = APIClient.GetRequest<Tuple<List<PlanViewModel>, **int**>>

($"api/Main/GetPaginatedPlans?teacherId={APIClient.teacher.Id}&page={page}");

**var** model = (tmp.Item1, tmp.Item2, page);

**return** View(model);

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

[HttpGet]

**public** IActionResult FindPlan(PlanType Type, **int** DisciplineId, **int** GroupId)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

List<PlanViewModel> list = APIClient.GetRequest<List<PlanViewModel>>(

$"api/Main/GetFilteredPlans?teacherId={APIClient.teacher.Id}&disciplineId={DisciplineId}&groupId={GroupId}&type={Type}");

**return** RedirectToAction("Plans", "Home", new { serializedList = JsonConvert.SerializeObject(list) });

}

[HttpGet]

**public** IActionResult Student(**int** id)

{

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**try**

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

TestingViewModel testing = APIClient.GetRequest<TestingViewModel>($"api/Main/GetTesting?id={id}");

**return** View((testing.StudentTestings, id, testing.Type));

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

[HttpPost]

**public** **void** PutStudentsGrades(**int** testingId, List<**int**> studentId, List<MarkType> Grade)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

Response.Redirect("SignUp");

**return**;

}

**try**

{

List<StudentTestingBindingModel> list = new List<StudentTestingBindingModel>();

**for** (**int** i = 0; i < studentId.Count; i++)

{

list.**Add**(new StudentTestingBindingModel()

{

TestingId = testingId,

StudentId = studentId[i],

Grade = Grade[i]

});

}

APIClient.PostRequest($"api/Main/PutStudentMarks", list);

Response.Redirect("Plans");

**return**;

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

Response.Redirect("Plans");

**return**;

}

}

*//---------------------------- Plan&Marks -----------------------------*

*//------------------------- Messages&Reports --------------------------*

[HttpGet]

**public** IActionResult Reports(**string**? serializedList)

{

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

**if** (serializedList != **null**)

{

List<MessageViewModel>? list = JsonConvert.DeserializeObject<List<MessageViewModel>>(serializedList);

**return** View(list);

}

**var** messages = APIClient.GetRequest<List<MessageViewModel>>(

$"api/Main/GetTeacherMessages?id={APIClient.teacher.Id}");

**return** View(messages);

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

**public** IActionResult filterMessages(ReportTypes typeReport, Status status)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

List<MessageViewModel> list = APIClient.GetRequest<List<MessageViewModel>>(

$"api/Main/GetMessagesByStatusAndType?id={APIClient.teacher.Id}&status={status}&typeReport={typeReport}");

**return** RedirectToAction("Reports", "Home", new { serializedList = JsonConvert.SerializeObject(list) });

}

*//------------------------- Messages&Reports --------------------------*

*//----------------------------- Reports -------------------------------*

[HttpGet]

**public** IActionResult PlanReport(**int** id)

{

**if** (success != "")

{

ViewData["Success"] = success;

success = "";

}

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

**var** model = APIClient.GetRequest<ReportPlanViewModel>($"api/Main/GetObjectsForPlanReport?id={id}");

**return** View((model, id));

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

**public** IActionResult SavePlanReport(**int** messageId)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

**var** model = APIClient.GetRequest<MessageViewModel>($"api/Main/GetMessage?id={messageId}");

MessageBindingModel resModel = new MessageBindingModel

{

Id = model.Id,

ReportType = model.ReportType,

Status = model.Status,

DepartmentId = model.DepartmentId,

PlanId = model.PlanId,

DisciplineId = model.DisciplineId,

TeacherId = model.TeacherId

};

APIClient.PostRequest($"api/Main/SavePlanReport", new Tuple<MessageBindingModel, **string**>(resModel, rootPath + "**\\**TeacherApp**\\**wwwroot**\\**reports**\\**ОтчётПоЗанятию.docx"));

**string** fileName = "ОтчётПоЗанятию.docx";

**byte**[] fileBytes = System.IO.File.ReadAllBytes(\_hostEnvironment.WebRootPath + @"\reports\" + fileName);

**return** File(fileBytes, "application/force-download", fileName);

} **catch** (Exception ex)

{

unxpError = "Файл не был сохранён **\n**";

unxpError += GetCyrillic(ex.Message);

**return** Redirect($"/Home/PlanReport?id={messageId}");

}

}

[HttpGet]

**public** IActionResult FullReport(**int** id)

{

**if** (success != "")

{

ViewData["Success"] = success;

success = "";

}

**if** (unxpError != "")

{

ViewData["UnxpError"] = unxpError;

unxpError = "";

}

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

**var** model = APIClient.GetRequest<ReportFullViewModel>($"api/Main/GetObjectsForSumReport?id={id}");

**return** View((model, id));

} **catch** (Exception ex)

{

unxpError = GetCyrillic(ex.Message);

**return** View();

}

}

**public** IActionResult SaveFullReport(**int** messageId)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**try**

{

**var** model = APIClient.GetRequest<MessageViewModel>($"api/Main/GetMessage?id={messageId}");

MessageBindingModel resModel = new MessageBindingModel

{

Id = model.Id,

ReportType = model.ReportType,

Status = model.Status,

DepartmentId = model.DepartmentId,

PlanId = model.PlanId,

DisciplineId = model.DisciplineId,

TeacherId = model.TeacherId

};

APIClient.PostRequest($"api/Main/SaveSumReport", new Tuple<MessageBindingModel, **string**>(resModel, rootPath + "**\\**TeacherApp**\\**wwwroot**\\**reports**\\**ОтчётПолный.docx"));

**string** fileName = "ОтчётПолный.docx";

**byte**[] fileBytes = System.IO.File.ReadAllBytes(\_hostEnvironment.WebRootPath + @"\reports\" + fileName);

success = "Файл успешно сохранён";

**return** File(fileBytes, "application/force-download", fileName);

} **catch** (Exception ex)

{

unxpError = "Файл не был сохранён **\n**";

unxpError += GetCyrillic(ex.Message);

**return** Redirect($"/Home/FullReport?id={messageId}");

}

}

*//----------------------------- Reports -------------------------------*

*//------------------------------- Mail --------------------------------*

**public** IActionResult SendReportByMail(**int** messageId)

{

**if** (APIClient.teacher == **null**)

{

error = "Похожке, что вы вышли из системы";

**return** Redirect("SignUp");

}

**var** department = APIClient.GetRequest<DepartmentViewModel>($"api/Main/GetDepartment?id={APIClient.teacher.DepartmentId}&login={null}&password={null}");

**var** model = APIClient.GetRequest<MessageViewModel>($"api/Main/GetMessage?id={messageId}");

MessageBindingModel resModel = new MessageBindingModel

{

Id = model.Id,

ReportType = model.ReportType,

Status = model.Status,

DepartmentId = model.DepartmentId,

PlanId = model.PlanId,

DisciplineId = model.DisciplineId,

TeacherId = model.TeacherId

};

**if** (resModel.ReportType == ReportTypes.SumReport)

{

**try**

{

APIClient.PostRequest($"api/Main/SaveSumReport", new Tuple<MessageBindingModel, **string**>(resModel, rootPath + "**\\**TeacherApp**\\**wwwroot**\\**reports**\\**ОтчётПолный.docx"));

**string** fileName = "ОтчётПолный.docx";

**string** filePath = \_hostEnvironment.WebRootPath + @"\reports\" + fileName;

APIClient.PostRequest($"api/Main/SendMessage", new MailSendInfoBindingModel

{

MailAddress = department.Login,

Subject = "Итоговый отчёт по дисциплине " + model.DisciplineName + ", преподаватель: " + APIClient.teacher.Flm,

Text = "Отчёт находится в прикреплённом файле **\n** С уважением, **\n**" + APIClient.teacher.Flm,

FilePath = filePath

});

success = "Письмо успешно отправлено";

APIClient.PostRequest($"api/Main/AnswerRequest", new MessageBindingModel

{

Id = model.Id,

DepartmentId = model.DepartmentId

});

**return** Redirect($"/Home/FullReport?id={messageId}");

} **catch** (Exception ex)

{

unxpError = "Письмо не отправлено **\n**";

unxpError += GetCyrillic(ex.Message);

**return** Redirect($"/Home/FullReport?id={messageId}");

}

}

**else**

{

**try**

{

APIClient.PostRequest($"api/Main/SavePlanReport", new Tuple<MessageBindingModel, **string**>(resModel, rootPath + "**\\**TeacherApp**\\**wwwroot**\\**reports**\\**ОтчётПоЗанятию.docx"));

**string** fileName = "ОтчётПоЗанятию.docx";

**string** filePath = \_hostEnvironment.WebRootPath + @"\reports\" + fileName;

APIClient.PostRequest($"api/Main/SendMessage", new MailSendInfoBindingModel

{

MailAddress = department.Login,

Subject = "Отчёт по учебному плану " + model.PlanName + ", преподаватель: " + APIClient.teacher.Flm,

Text = model.PlanName + "**\n**Отчёт находится в прикреплённом файле **\n** С уважением, **\n**" + APIClient.teacher.Flm,

FilePath = filePath

});

success = "Письмо успешно отправлено";

APIClient.PostRequest($"api/Main/AnswerRequest", new MessageBindingModel

{

Id = model.Id,

DepartmentId = model.DepartmentId

});

**return** Redirect($"/Home/PlanReport?id={messageId}");

} **catch**(Exception ex)

{

unxpError = "Письмо не отправлено **\n**";

unxpError += GetCyrillic(ex.Message);

**return** Redirect($"/Home/PlanReport?id={messageId}");

}

}

}

*//------------------------------- Mail --------------------------------*

*//----------------------------- Encoding ------------------------------*

**public** **string** GetCyrillic(**string** input)

{

**string** result = "";

**string** pattern = @"[А-Яа-я]+[,.:;!?]\*(\s+)|[А-Яа-я]+[,.:;!?]";

**foreach** (Match m **in** Regex.Matches(input, pattern))

{

result += m.**Value**;

}

**return** result;

}

}

}

﻿@**using** UniversityContracts.Enums;

@**using** UniversityContracts.ViewModels;

@model List<MessageViewModel>

@{

ViewData["Title"] = "Отчётность";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

}

<div>

<img src="/images/reports-left-img.png" **class**="img-thumbnail" />

<div **class**="d-inline-flex ml-5 p-2" style="max-width: 65vh; vertical-align: top;">

<div>

<h1 **class**="fw-bold mb-5 mt-3">Отчётность</h1>

<h2 **class**="fw-bold mb-4">Оставьте дедлайны нам!</h2>

<p **class**="text-break fs-4">

Здесь вы можете видеть поручения

от кафедры об отчётности, проверить их статус,

а также с помощью нашего сайна быстро собрать

необходимый отчёт и отправить его на проверку.

</p>

</div>

</div>

</div>

<div **class**="d-flex justify-content-center mt-5">

<div id="solidFooterLine" **class**="mid-solid-line"></div>

</div>

<div **class**="d-flex justify-content-center mt-2 mb-5">

<div id="normalFooterLine" **class**="mid-normal-line"></div>

</div>

<h1 **class**="fw-bold m-5">Сообщения</h1>

<form asp-action="filterMessages">

<div **class**="mb-3 w-80">

<label **class**="form-label">Тип отчёта в сообщении</label>

<**select** name="typeReport" **class**="form-select" asp-items="@Html.GetEnumSelectList<ReportTypes>()">

</**select**>

</div>

<div **class**="mb-3 w-80">

<label **class**="form-label">Статус сообщения</label>

<**select** name="status" **class**="form-select" asp-items="@Html.GetEnumSelectList<Status>()">

</**select**>

</div>

<button **class**="btn btn-secondary" type="submit">Сортировать</button>

<a style="margin-left: 3%" asp-action="Reports">Очистить поиск</a>

</form>

<!-- Loading next page -->

<div **class**="modal fade" id="loadPage" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content">

</div>

</div>

</div>

<div **class**="d-flex justify-content-center mt-5 mb-5">

<div id="normalFooterLine" **class**="mid-normal-line"></div>

</div>

<div **class**="d-flex justify-content-around flex-wrap">

@{

**if** (Model == **null**)

{

<p>Сообщений от кафедры нет</p>

}

**else**

{

**if** (Model.Count() == 0)

{

<div>

<p>Сообщения с данным статусом не найдены</p>

<a style="margin-left: 3%" asp-action="Reports">Вернуться к полному списку</a>

</div>

}

**for** (**int** i = 0; i < Model.Count(); i++)

{

<div **class**="m-3 border-asphalt vw-60 d-flex flex-column">

<p **class**="fw-bold fs-4 w-100 light-grey text-center">Поручение @(i + 1)</p>

@**if** (Model[i].ReportType == ReportTypes.SumReport)

{

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Тип отчёта: "Итоговый отчёт по дисциплине"

</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Дисциплина "@Model[i].DisciplineName"

</p>

}

@**if** (Model[i].ReportType == ReportTypes.PlanReport)

{

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Тип отчёта: "Отчёт по плану"

</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

"@Model[i].PlanName"

</p>

}

@**if** (Model[i].Status == Status.Активен)

{

<p **class**="msg-active" style="margin-top:auto; margin-left: 0.25rem; margin-right: 0.25rem;">

Статус "Активен"

</p>

@**if** (Model[i].ReportType == ReportTypes.SumReport)

{

<div **class**="d-flex">

<a onclick="SetPageAsLoading()" asp-action="FullReport" asp-route-id="@Model[i].Id" **class**="fs-5 a-msg text-center flex-grow-1">

Сформировать

</a>

</div>

} **else**

{

<div **class**="d-flex">

<a onclick="SetPageAsLoading()" asp-action="PlanReport" asp-route-id="@Model[i].Id" **class**="fs-5 a-msg text-center flex-grow-1">

Сформировать

</a>

</div>

}

}

@**if** (Model[i].Status == Status.Проверяется)

{

<p **class**="msg-checking" style="margin-top:auto; margin-left: 0.25rem; margin-right: 0.25rem;">

Статус "На проверке"

</p>

}

@**if** (Model[i].Status == Status.Проверен)

{

<p **class**="msg-done" style="margin-top:auto; margin-left: 0.25rem; margin-right: 0.25rem;">

Статус "Принят"

</p>

}

</div>

}

<!--------------->

}

}

</div>

@section scripts {

<script src="~/js/reports-page.js"></script>

}

﻿@**using** UniversityContracts.ViewModels;

@model (ReportFullViewModel model, **int** id)

@{

ViewData["Title"] = "Отчёт по дисциплине";

Layout = "\_Blank\_Layout";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

**if** (ViewData["Success"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop1").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop1" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #b5ffb0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Выполнено</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["Success"]

</div>

</div>

</div>

</div>

}

}

<!-- Loading page -->

<div **class**="modal fade" id="loadPage" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content">

</div>

</div>

</div>

<a **class**="btn btn-primary bth-find mt-4" role="button" asp-action="SaveFullReport" asp-route-messageId="@Model.id">Сохранить</a>

<a onclick="SetPageAsLoading()" **class**="btn btn-primary bth-find mt-4" role="button" asp-action="SendReportByMail" asp-route-messageId="@Model.id">Отправить на проверку</a>

<div **class**="container-sm" style="min-height: 100vh;">

<p>@ViewData["test"]</p>

<h1 **class**="text-center m-4 fs-4 fw-bold">@Model.model.Title</h1>

<div style="margin-bottom: 3%; margin-top: 3%">

@**foreach** (**var** pn **in** Model.model.DisciplineName)

{

<h1 **class**="fs-5">@pn</h1>

}

</div>

@{

**foreach** (**var** **group** **in** Model.model.Items)

{

<h1 **class**="mt-4 fs-5">Группа: @**group**.Item1</h1>

<table **class**="table table-bordered border-dark border-3">

<tr>

@{

<th rowspan="2" **class**="text-center">ФИО</th>

**for** (**int** i = 0; i < **group**.Item2[0].Item2.Count(); i++)

{

<th **class**="text-center" colspan="@group.Item2[0].Item2[i].Item2.Count()">@**group**.Item2[0].Item2[i].Item1</th>

}

}

</tr>

<tr>

@{

**foreach**(**var** subitem **in** **group**.Item2[0].Item2)

{

**foreach** (**var** subitem2 **in** subitem.Item2)

{

<th **class**="text-end">@subitem2.Item1</th>

}

}

}

</tr>

@{

**foreach**(**var** subItem **in** **group**.Item2)

{

<tr>

<td> @subItem.Item1</td>

@**foreach**(**var** subItem2 **in** subItem.Item2)

{

@**foreach** (**var** subItem3 **in** subItem2.Item2)

{

<td **class**="text-end">@subItem3.Item2</td>

}

}

</tr>

}

<tr>

@**foreach**(**var** itog **in** Model.model.Itog[Model.model.Items.IndexOf(**group**)].Item1)

{

<td **class**="text-end">@itog</td>

}

</tr>

}

</table>

}

}

<div style="margin-bottom: 1%; margin-top: 5%">

@**foreach** (**var** p **in** Model.model.Footer)

{

<p **class**="fs-5">@p</p>

}

</div>

<a asp-action="Reports">Вернуться к сообщениям</a>

</div>

@section scripts {

<script src="~/js/report.js"></script>

}

﻿@**using** UniversityContracts.Enums;

@**using** UniversityContracts.ViewModels;

@model (**string** naming, **string** flm, List<MessageViewModel> message, List<Tuple<**int**, **string**, **int**, DateTime>> testing)

@{

ViewData["Title"] = "Главная";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

}

<div>

<img src="/images/top-left-img.png" **class**="img-thumbnail" />

<div **class**="d-inline-flex p-2" style="max-width: 65vh; vertical-align: top;">

<div>

<h1 **class**="fw-bold mb-5 mt-3">Кафедра "@Model.naming"</h1>

<h2 **class**="fw-bold mb-4">Добро пожаловать "@Model.flm"</h2>

<p **class**="text-break fs-4">

Начнём образовательную деятельность

вместе, без лишних проблем и забот

об отчётностях и дедлайнах.

</p>

</div>

</div>

</div>

<div **class**="d-flex justify-content-center mt-4">

<div **class**="mid-solid-line"></div>

</div>

<div **class**="d-flex justify-content-center mt-2 mb-4">

<div **class**="mid-normal-line"></div>

</div>

<h1 **class**="fw-bold mb-2 mt-5">Поручения от кафедры</h1>

<div **class**="d-flex justify-content-lg-start mt-2 mb-5">

<div **class**="short-solid-line"></div>

</div>

<div **class**="align-content-end mt-5 mb-5">

<div **class**="d-flex flex-row justify-content-end">

<div **class**="msg-border-violet d-flex flex-wrap justify-content-center">

@{

**if** (Model.message.Count() == 0)

{

<h1 **class**="text-center fs-5 mb-3 mt-2">У вас пока что нет поручений</h1>

}

**else**

{

**var** length = 2;

**if** (Model.message.Count() < length)

{

length = Model.message.Count();

}

**for** (**int** i = 0; i < length; i++)

{

<div **class**="m-3 border-violet d-flex flex-column" style="width: 60vh;">

<p **class**="fw-bold fs-4 w-100 light-violet text-center">Поручение @(i+1)</p>

@**if** (Model.message[i].ReportType == ReportTypes.SumReport)

{

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Тип отчёта: "Итоговый отчёт по дисциплине"

</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Дисциплина "@Model.message[i].DisciplineName"

</p>

}

@**if** (Model.message[i].ReportType == ReportTypes.PlanReport)

{

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Тип отчёта: "Отчёт по занятию"

</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Занятие "@Model.message[i].PlanName"

</p>

}

<p **class**="msg-active" style="margin-left: 0.25rem; margin-right: 0.25rem; margin-top:auto; background-color: #FDC83E;">

Статус "Активен"

</p>

</div>

}

<div style="margin-bottom: 3%" **class**="text-center w-100">

<a **class**="fs-5" asp-action="Reports">Больше</a>

</div>

}

}

</div>

</div>

</div>

<h1 **class**="fw-bold mb-2 mt-5">Занятия сегодня</h1>

<div **class**="d-flex justify-content-lg-start mt-2 mb-5">

<div **class**="short-solid-line"></div>

</div>

<div **class**="align-content-end mt-5 mb-5">

<div **class**="d-flex flex-row justify-content-end">

<div **class**="msg-border-violet d-flex flex-wrap justify-content-center">

@{

**if** (Model.testing.Count() == 0)

{

<h1 **class**="text-center fs-5 mb-3 mt-2">Сегодня нет никаких занятий!</h1>

}

**else**

{

**var** length = 2;

**if** (Model.testing.Count() < length)

{

length = Model.testing.Count();

}

**for** (**int** i = 0; i < length; i++)

{

<div **class**="m-3 border-violet" style="width: 60vh;">

<p **class**="fw-bold fs-4 w-100 light-violet text-center">Занятие @(i + 1)</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Тема "@Model.testing[i].Item2"

</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

Часы "@Model.testing[i].Item3"

</p>

</div>

}

<div style="margin-bottom: 3%" **class**="text-center w-100">

<a **class**="fs-5" asp-action="Plans">Больше</a>

</div>

}

}

</div>

</div>

</div>

@section scripts {

<script src="~/js/index-page.js"></script>

}

﻿@{

Layout = "\_SignUp\_Layout";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

**else**

{

**if** (ViewData["Error"] != **null**)

{

<div **class**="alert alert-danger alert-dismissible fade show" role="alert" style="max-width: 50vh;">

@ViewData["Error"]

<button type="button" **class**="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>

</div>

}

}

}

<div **class**="mb-4">

<img src="/images/signup-img.png" **class**="rounded mx-auto d-block">

</div>

<div **class**="mb-4 text-center">

<h1>Войти</h1>

</div>

<form asp-action="TeacherSignUp" method="post">

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Введите логин</label>

<input **class**="form-control" type="text" name="Login" placeholder="Логин" required />

</div>

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Введите пароль</label>

<input **class**="form-control" type="password" name="Password" placeholder="Пароль" required />

</div>

<input type="submit" **class**="btn btn-primary" **value**="Войти" />

</form>

﻿@**using** UniversityContracts.Enums;

@model (List<Tuple<**int**, **string**, MarkType>> students, **int** testingId, PlanType testingType)

@{

Layout = "\_Simp\_2\_Layout";

ViewData["Title"] = "Оценивание студентов";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

}

<div **class**="mb-4 text-center">

<h1>Оценить студентов</h1>

</div>

<form asp-action="PutStudentsGrades" method="post">

<input type="hidden" name="testingId" **value**="@Model.testingId" />

@{

**for** (**int** i = 0; i < Model.students.Count(); i++)

{

<input type="hidden" name="studentId" **value**="@Model.students[i].Item1" />

<div **class**="mb-3 d-flex frex-wrap">

<p style="margin-right: 2%; margin-bottom: 0;" **class**="fs-5">@(i + 1).</p>

<p **class**="flex-grow-1 fs-5" style=" margin-bottom: 0;">@Model.students[i].Item2</p>

@{

**var** selectList = Html.GetEnumSelectList<MarkType>();

**if** (Model.testingType == PlanType.Лекционный)

{

selectList = selectList.**Where**(rec => rec.Text != MarkType.Неуд.ToString() &&

rec.Text != MarkType.Удовл.ToString() &&

rec.Text != MarkType.Хор.ToString() &&

rec.Text != MarkType.Отл.ToString()).ToList();

} **else**

{

selectList = selectList.**Where**(rec => rec.Text != MarkType.П.ToString() &&

rec.Text != MarkType.УП.ToString() &&

rec.Text != MarkType.НП.ToString()).ToList();

}

**foreach** (**var** item **in** selectList)

{

**if** (item.Text == Model.students[i].Item3.ToString())

item.Selected = **true**;

}

}

<**select** **class**="form-select" style="width: 13vh;" asp-items="selectList" name="Grade"></**select**>

</div>

}

}

<input type="submit" **class**="btn btn-primary" **value**="Оценить" />

<a style="margin-left: 3%" asp-action="Plans">Назад</a>

</form>

﻿@**using** UniversityContracts.Enums;

@**using** UniversityContracts.ViewModels;

@model (**int** groupId, **int** planId, TestingViewModel testing)

@{

Layout = "\_Simp\_Layout";

ViewData["Title"] = "Добавление и редактирование задания";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

**else**

{

**if** (ViewData["Error"] != **null**)

{

<div **class**="alert alert-danger alert-dismissible fade show" role="alert" style="max-width: 50vh;">

@ViewData["Error"]

<button type="button" **class**="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>

</div>

}

}

}

<div **class**="mb-4 text-center">

@{

**if** (Model.testing.Id != **null**)

{

<h1>Редактировать задание</h1>

}

**else**

{

<h1>Добавить задание</h1>

}

}

</div>

<form asp-action="UpdateTesting" method="post">

@{

<input type="hidden" name="PlanId" **value**="@Model.planId" />

<input type="hidden" name="GroupId" **value**="@Model.groupId" />

**if** (Model.testing.Id != **null**)

{

<input type="hidden" name="Id" **value**="@Model.testing.Id" />

}

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Введите тему задания</label>

<input **class**="form-control" type="text" name="Topic" placeholder="Тема" **value**="@Model.testing.Topic" required />

</div>

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Выберите дату задания</label>

@**if** (Model.testing.Id == **null**)

{

<input **class**="form-control" type="date" name="Date" required />

} **else**

{

<input **class**="form-control" type="date" name="Date" **value**="@(Model.testing.Date.Date.ToString("yyyy-MM-dd"))" required />

}

</div>

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Выберите продолжительность задания</label>

<input **class**="form-control" type="number" name="Hours" **value**="@Model.testing.Hours" required />

</div>

@**if** (Model.testing.Id != **null**)

{

<input type="submit" **class**="btn btn-primary" **value**="Редактировать" />

}

**else**

{

<input type="submit" **class**="btn btn-primary" **value**="Добавить" />

}

<a style="margin-left: 3%" asp-action="Plan">Назад</a>

}

</form>

@section scripts {

<script src="~/js/Testing-page.js"></script>

}

﻿@**using** UniversityContracts.ViewModels;

@model (List<PlanViewModel> plans, Dictionary<**int**, **string**> disciplines)

@{

ViewData["Title"] = "План обучения";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

}

<div>

<img src="/images/plan-left-img.png" **class**="img-thumbnail" />

<div **class**="d-inline-flex ml-5 p-2" style="max-width: 65vh; vertical-align: top;">

<div>

<h1 **class**="fw-bold mb-5 mt-3">План обучения</h1>

<h2 **class**="fw-bold mb-4">Легко распределяйте свою работу!</h2>

<p **class**="text-break fs-4">

Здесь вы можете добавить, редактировать

или удалить задания на определённых

занятиях.

</p>

</div>

</div>

</div>

<div **class**="d-flex justify-content-center mt-5">

<div id="solidFooterLine" **class**="mid-solid-line"></div>

</div>

<div **class**="d-flex justify-content-center mt-2 mb-4">

<div id="normalFooterLine" **class**="mid-normal-line"></div>

</div>

<div **class**="d-flex justify-content-center">

<div **class**="discipline-links d-flex justify-content-around flex-wrap mt-5">

@{

**if** (Model.disciplines.Count() == 0)

{

<p **class**="text-center">

Хм, похоже мы не нашли никакой информации о том,

что вы ведёте какие либо предметы. Обратитесь на кафедру, если вы уверены

что ведёте хоть что-то

</p>

}

**else**

{

**foreach** (**var** item **in** Model.disciplines)

{

**if** (Model.plans.Count != 0)

{

**if** (Model.plans[0].DisciplineId == item.Key)

{

<a asp-action="Plan" asp-route-disciplineId="@item.Key" **class**="link-danger fs-5">@item.**Value**</a>

}

**else**

{

<a asp-action="Plan" asp-route-disciplineId="@item.Key" **class**="link-primary fs-5">@item.**Value**</a>

}

}

**else**

{

<a asp-action="Plan" asp-route-disciplineId="@item.Key" **class**="link-primary fs-5">@item.**Value**</a>

}

}

}

}

</div>

</div>

<div **class**="container-sm lessons-border mt-5 mb-5 pb-4">

<div>

<!--Dates with lessons-->

@{

**if** (Model.plans.Count() == 0)

{

<p **class**="text-center mt-4">

У вас пока что нет учебных планов, дождитесь, поа их назначат вам на кафедре

</p>

}

**else**

{

<div **class**="accordion" id="accordionPanelsStayOpenExample">

@**for** (**int** i = 0; i < Model.plans.Count(); i++)

{

<!--Plans-->

<div **class**="accordion-item">

<div **class**="d-flex justify-content-between flex-wrap mt-4">

<h1 **class**="fs-5 flex-grow-5 accordion-header">

<button **class**="accordion-button collapsed h-lesson" type="button" data-bs-toggle="collapse" data-bs-target="#panelsStayOpen-collapse@(i)" aria-expanded="true" aria-controls="panelsStayOpen-collapse@(i)">

@(i + 1). “@Model.plans[i].Name”

</button>

</h1>

<a asp-action="Testing" asp-route-groupId="@Model.plans[i].GroupId" asp-route-planId="@Model.plans[i].Id" **class**="fs-4 text-center flex-grow-1 a-lesson">

<i **class**="fa-sharp fa-solid fa-plus"></i>

</a>

</div>

@**if** (@Model.plans[i].Testings != **null**)

{

<div id="panelsStayOpen-collapse@(i)" **class**="accordion-collapse collapse">

<div **class**="d-flex flex-wrap">

@{

**int** j = 0;

}

@**foreach** (**var** item **in** @Model.plans[i].Testings)

{

j++;

<!--Tasks of particular plan-->

<div **class**="d-flex justify-content-between flex-wrap" style="max-width: 55vh; flex-grow: 1; margin-right: 1%;">

<div style="flex-grow: 1;">

<div **class**="testing mt-1">

<p **class**="testing-item mt-2">Задание @j</p>

<p **class**="testing-item">Тема: @item.Item2</p>

<p **class**="testing-item">Часы: @item.Item3</p>

<p **class**="testing-item">Дата: @item.Item4.ToShortDateString()</p>

</div>

<a asp-action="Testing" asp-route-groupId="@Model.plans[i].GroupId" asp-route-planId="@Model.plans[i].Id" asp-route-testingId="@item.Item1" **class**="btn btn-primary bth-update mt-2" role="button">Редактировать</a>

<!-- Deleting -->

<div **class**="modal fade" id="Modal-@item.Item1" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content">

<div **class**="modal-header">

<h1 **class**="modal-title fs-5" id="exampleModalLabel">Удалить задание</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

Вы действительно хотите удалить это задание?

</div>

<div **class**="modal-footer">

<button type="button" **class**="btn btn-secondary" data-bs-dismiss="modal">Отмена</button>

<a asp-action="DeleteTesting" asp-route-id="@item.Item1" type="button" **class**="btn btn-primary">Удалить</a>

</div>

</div>

</div>

</div>

<a data-bs-toggle="modal" data-bs-target="#Modal-@item.Item1" **class**="btn btn-primary bth-delete mt-2" role="button">Удалить</a>

</div>

</div>

}

</div>

</div>

}

</div>

}

</div>

}

}

</div>

</div>

@section scripts {

<script src="~/js/Plan-page.js"></script>

}

﻿@**using** UniversityContracts.ViewModels;

@model (ReportPlanViewModel model, **int** id)

@{

ViewData["Title"] = "Отчёт по плану";

Layout = "\_Blank\_Layout";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

**if** (ViewData["Success"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop1").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop1" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #b5ffb0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Выполнено</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["Success"]

</div>

</div>

</div>

</div>

}

}

<!-- Loading page -->

<div **class**="modal fade" id="loadPage" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content">

</div>

</div>

</div>

<a **class**="btn btn-primary bth-find mt-4" role="button" asp-action="SavePlanReport" asp-route-messageId="@Model.id">Сохранить</a>

<a onclick="SetPageAsLoading()" **class**="btn btn-primary bth-find mt-4" role="button" asp-action="SendReportByMail" asp-route-messageId="@Model.id">Отправить на проверку</a>

<div **class**="container-sm" style="min-height: 100vh;">

<h1 **class**="text-center m-4 fs-4 fw-bold">@Model.model.Title</h1>

<div style="margin-bottom: 3%; margin-top: 3%">

@**foreach** (**var** pn **in** Model.model.PlanName)

{

<h1 **class**="fs-5">@pn</h1>

}

</div>

<table **class**="table table table-bordered border-dark border-3">

<tr>

@{

<th **class**="text-center">№</th>

<th **class**="text-center">№ з.к.</th>

<th **class**="text-center">ФИО</th>

**for** (**int** i = 0; i < Model.model.Items[0].Item3.Count(); i++)

{

<th **class**="text-center">@Model.model.Items[0].Item3[i].Item1</th>

}

}

</tr>

@{

**int** j = 0;

**foreach** (**var** item **in** Model.model.Items)

{

j++;

<tr>

<td>@j</td>

<td>@item.Item1</td>

<td>@item.Item2</td>

@**foreach**(**var** grade **in** item.Item3)

{

<td **class**="text-end">@grade.Item2</td>

}

</tr>

}

<tr>

@**foreach**(**var** itog **in** Model.model.Itog)

{

<td>@itog</td>

}

</tr>

}

</table>

<div style="margin-bottom: 1%; margin-top: 5%">

@**foreach** (**var** p **in** Model.model.Footer)

{

<p **class**="fs-5">@p</p>

}

</div>

<a asp-action="Reports">Вернуться к сообщениям</a>

</div>

@section scripts {

<script src="~/js/report.js"></script>

}

﻿@**using** UniversityContracts.Enums;

@**using** UniversityContracts.ViewModels;

@model (List<PlanViewModel> plans, **int** numOfPage, **int** page)

@{

ViewData["Title"] = "Задания и оценки";

**if** (ViewData["UnxpError"] != **null**)

{

<script type="text/javascript">

window.onload = function () {

OpenBootstrapPopup();

};

function OpenBootstrapPopup() {

$("#staticBackdrop").modal('show');

}

</script>

<div **class**="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div **class**="modal-dialog">

<div **class**="modal-content" style="background: #ffb0b0;">

<div **class**="modal-header">

<h1 **class**="modal-title fs-3" id="staticBackdropLabel">Ошибка</h1>

<button type="button" **class**="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div **class**="modal-body">

@ViewData["UnxpError"]

</div>

</div>

</div>

</div>

}

}

<div>

<img src="/images/lesson-left-img.png" **class**="img-thumbnail" />

<div **class**="d-inline-flex ml-5 p-2" style="max-width: 65vh; vertical-align: top;">

<div>

<h1 **class**="fw-bold mb-5 mt-3">Занятия и оценки</h1>

<h2 **class**="fw-bold mb-4">Оценивайте студентов в своё удовольствие!</h2>

<p **class**="text-break fs-4">

Здесь вы можете найти определённое занятие,

воспользовавшись формой поиска и

оценить студентов, выполнивших задания

на этом занятии.

</p>

</div>

</div>

</div>

<div **class**="d-flex justify-content-center mt-5">

<div id="solidFooterLine" **class**="mid-solid-line"></div>

</div>

<div **class**="d-flex justify-content-center mt-2 mb-5">

<div id="normalFooterLine" **class**="mid-normal-line"></div>

</div>

<h1 **class**="fw-bold m-5">Найти занятие</h1>

<div **class**="m-3 p-3 border-violet w-80">

<form asp-action="FindPlan" method="get">

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Выберите тип занятия</label>

<**select** **class**="form-select" asp-items="@Html.GetEnumSelectList<PlanType>()" name="Type"></**select**>

</div>

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Выберите дисциплину</label>

<**select** **class**="form-select" name="DisciplineId" asp-items="@(new SelectList(ViewBag.Disciplines, "Id", "Name"))">

</**select**>

</div>

<div **class**="mb-3">

<label **for**="basic-url" **class**="form-label">Выберите группу</label>

<**select** **class**="form-select" name="GroupId" asp-items="@(new SelectList(ViewBag.Groups, "Id", "Name"))">

</**select**>

</div>

<input type="submit" **class**="btn btn-primary bth-find" **value**="Найти" />

<a style="margin-left: 3%" asp-action="Plans">Очистить поиск</a>

</form>

</div>

<div **class**="d-flex justify-content-center mt-5 mb-5">

<div id="normalFooterLine" **class**="mid-normal-line"></div>

</div>

<div **class**="d-flex justify-content-around flex-wrap">

@{

**if** (Model.plans == **null**)

{

<p **class**="text-center">Занятий не назначено</p>

}

**else**

{

**if** (Model.plans.Count == 0)

{

<div>

<p **class**="text-center">По вашему запросу занятий не найдено</p>

<a style="margin-left: 3%" asp-action="Plans">Вернуться к полному списку</a>

</div>

}

**for** (**int** i = 0; i < Model.plans.Count(); i++)

{

<!--Lessons here-->

<div **class**="m-3 border-violet vw-80 d-flex flex-column">

<p **class**="fw-bold fs-4 w-100 light-violet text-center">План @(i + 1)</p>

<p style="margin-left: 0.25rem; margin-right: 0.25rem;">

"@Model.plans[i].Name"

</p>

@**if** (Model.plans[i].Testings != **null**)

{

**for** (**int** j = 0; j < Model.plans[i].Testings.Count(); j++)

{

<!--Testings here-->

<div **class**="d-flex justify-content-between flex-wrap" style="margin-top:auto;">

<h1 **class**="fs-5 pt-2 pb-2 flex-grow-5 h-testing">

@(j + 1). “@Model.plans[i].Testings[j].Item2”

</h1>

<a asp-action="Student" asp-route-id="@Model.plans[i].Testings[j].Item1" **class**="fs-5 text-center flex-grow-1 a-testing">

Оценить

</a>

</div>

}

}

</div>

}

}

}

</div>

<div **class**="text-center mt-5">

@{

**if** ((Model.page - 1) >= 1)

{

<a asp-action="Plans" asp-route-page="@(Model.page-1)">Назад</a>

}

**for** (**int** i = 1; i <= Model.numOfPage; i++)

{

<a asp-action="Plans" asp-route-page="@(i)">@(i)</a>

}

**if** (Model.page < Model.numOfPage)

{

<a asp-action="Plans" asp-route-page="@(Model.page+1)">Далее</a>

}

}

</div>

@section scripts {

<script src="~/js/lessons-page.js"></script>

}

﻿<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - University</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/main.css" asp-append-version="true" />

<link rel="stylesheet" href="~/css/navbar.css" asp-append-version="true" />

<script src="https://kit.fontawesome.com/0550389aeb.js" crossorigin="anonymous"></script>

</head>

<body>

<header>

<div **class**="upper-line"></div>

<div id="navLine" **class**="line"></div>

</header>

<div **class**="d-flex align-items-center d-flex justify-content-center" style="height: 100vh;">

<div **class**="bg-light" style="padding: 2% 5% 5% 5%;">

<main role="main" **class**="pb-3">

@RenderBody()

</main>

</div>

</div>

<footer **class**="border-top footer pb-5">

<div **class**="container-sm">

<h1 **class**="text-center fs-4 mb-3 mt-4">2023 - TeacherApp</h1>

<div **class**="d-flex justify-content-center mt-2 mb-4">

<div **class**="mid-solid-line"></div>

</div>

<p **class**="mb-1 mt-1 text-center">

Образование – это незаменимая составляющая нашей жизни.

Оно даёт возможность расширять свои знания, развивать личность и

достигать новых вершин. На сайте мы собрали для вас полезную

информацию о различных образовательных программах и курсах,

которые помогут вам реализовать свой потенциал. Мы верим, что

образование – лучший способ инвестировать в себя и своё будущее

</p>

<div **class**="d-flex justify-content-center mt-4 mb-4">

<div **class**="mid-normal-line"></div>

</div>

<p **class**="mb-0">Поддержите нас</p>

<a href="https://www.youtube.com/watch?v=rXbPAT62hSs">Поддержать тут</a>

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@**await** RenderSectionAsync("Scripts", required: **false**)

</body>

</html>

﻿@page

@model ErrorModel

@{

ViewData["Title"] = "Error";

}

<h1 **class**="text-danger">Error.</h1>

<h2 **class**="text-danger">An error occurred **while** processing your request.</h2>

@**if** (Model.ShowRequestId)

{

<p>

<strong>Request ID:</strong> <code>@Model.RequestId</code>

</p>

}

<h3>Development Mode</h3>

<p>

Swapping to the <strong>Development</strong> environment displays detailed information about the error that occurred.

</p>

<p>

<strong>The Development environment shouldn't be enabled for deployed applications.</strong>

It can result in displaying sensitive information from exceptions to end users.

For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE\_ENVIRONMENT</strong> environment variable to <strong>Development</strong>

and restarting the app.

</p>

﻿<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using System.Diagnostics;

namespace TeacherApp.Pages

{

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

[IgnoreAntiforgeryToken]

public class ErrorModel : PageModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

private readonly ILogger<ErrorModel> \_logger;

public ErrorModel(ILogger<ErrorModel> logger)

{

\_logger = logger;

}

public void OnGet()

{

RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;

}

}

}

﻿<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - University</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/main.css" asp-append-version="true" />

<link rel="stylesheet" href="~/css/navbar.css" asp-append-version="true" />

<script src="https://kit.fontawesome.com/0550389aeb.js" crossorigin="anonymous"></script>

</head>

<body>

<header>

<div class="upper-line"></div>

<div id="navLine" class="line"></div>

</header>

<div class="container">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

<footer class="border-top footer pb-5">

<div class="container-sm">

<h1 class="text-center fs-4 mb-3 mt-4">2023 - TeacherApp</h1>

<div class="d-flex justify-content-center mt-2 mb-4">

<div class="mid-solid-line"></div>

</div>

<p class="mb-1 mt-1 text-center">

Образование – это незаменимая составляющая нашей жизни.

Оно даёт возможность расширять свои знания, развивать личность и

достигать новых вершин. На сайте мы собрали для вас полезную

информацию о различных образовательных программах и курсах,

которые помогут вам реализовать свой потенциал. Мы верим, что

образование – лучший способ инвестировать в себя и своё будущее

</p>

<div class="d-flex justify-content-center mt-4 mb-4">

<div class="mid-normal-line"></div>

</div>

<p class="mb-0">Поддержите нас</p>

<a href="https://www.youtube.com/watch?v=rXbPAT62hSs">Поддержать тут</a>

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

﻿<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - University</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/main.css" asp-append-version="true" />

<link rel="stylesheet" href="~/css/navbar.css" asp-append-version="true" />

<script src="https://kit.fontawesome.com/0550389aeb.js" crossorigin="anonymous"></script>

</head>

<body>

<header>

<div class="upper-line"></div>

<div id="navLine" class="line"></div>

<div style="margin-left: 7%; margin-right: 0%;">

<nav class="navbar navbar-expand-lg bg-body-tertiary">

<div class="container-fluid">

<a class="navbar-brand text-black fs-3 fw-bold" href="#">University</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon fa-solid fa-bars fs-4"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNavAltMarkup">

<div class="container-sm">

<div class="navbar-nav">

<a id="index" class="nav-link fs-5" asp-action="Index">Главная</a>

<a id="plan" class=" nav-link fs-5" asp-action="Plan">План обучения</a>

<a id="lessons" class="nav-link fs-5" asp-action="Plans">Занятия и оценки</a>

<a id="reports" class="nav-link fs-5" asp-action="Reports">Отчётность</a>

<div class="navbar-text d-flex flex-row-reverse" style="margin-left: 10%;">

<a class="btn btn-outline-danger" asp-action="Exit">Выйти</a>

</div>

</div>

</div>

</div>

</div>

</nav>

</div>

</header>

<div class="container">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

<footer class="border-top footer pb-5">

<div class="container-sm">

<h1 class="text-center fs-4 mb-3 mt-4">2023 - TeacherApp</h1>

<div class="d-flex justify-content-center mt-2 mb-4">

<div class="mid-solid-line"></div>

</div>

<p class="mb-1 mt-1 text-center">Образование – это незаменимая составляющая нашей жизни.

Оно даёт возможность расширять свои знания, развивать личность и

достигать новых вершин. На сайте мы собрали для вас полезную

информацию о различных образовательных программах и курсах,

которые помогут вам реализовать свой потенциал. Мы верим, что

образование – лучший способ инвестировать в себя и своё будущее</p>

<div class="d-flex justify-content-center mt-4 mb-4">

<div class="mid-normal-line"></div>

</div>

<p class="mb-0">Поддержите нас</p>

<a href="https://www.youtube.com/watch?v=rXbPAT62hSs">Поддержать тут</a>

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

﻿/\* Please see documentation at https://docs.microsoft.com/aspnet/core/client-side/bundling-and-minification

for details on configuring this project to bundle and minify static web assets. \*/

a.navbar-brand {

white-space: normal;

text-align: center;

word-break: break-all;

}

a {

color: #0077cc;

}

.btn-primary {

color: #fff;

background-color: #1b6ec2;

border-color: #1861ac;

}

.nav-pills .nav-link.active, .nav-pills .show > .nav-link {

color: #fff;

background-color: #1b6ec2;

border-color: #1861ac;

}

.border-top {

border-top: 1px solid #e5e5e5;

}

.border-bottom {

border-bottom: 1px solid #e5e5e5;

}

.box-shadow {

box-shadow: 0 .25rem .75rem rgba(0, 0, 0, .05);

}

button.accept-policy {

font-size: 1rem;

line-height: inherit;

}

.footer {

position: absolute;

bottom: 0;

width: 100%;

white-space: nowrap;

line-height: 60px;

}

﻿<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Вход в учётную запись - University</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/main.css" asp-append-version="true" />

<link rel="stylesheet" href="~/TeacherApp.styles.css" asp-append-version="true" />

</head>

<body style='background-image: url("/images/signup-bg.png")'>

<div class="d-flex align-items-center d-flex justify-content-center" style="height: 100vh;">

<div class="bg-light" style="padding: 2% 5% 5% 5%;">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

</div>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

﻿<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - University</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/main.css" asp-append-version="true" />

<link rel="stylesheet" href="~/css/navbar.css" asp-append-version="true" />

<script src="https://kit.fontawesome.com/0550389aeb.js" crossorigin="anonymous"></script>

</head>

<body>

<header>

<div class="upper-line"></div>

<div id="navLine" class="line"></div>

</header>

<div class="d-flex align-items-center d-flex justify-content-center" style="min-height: 100vh;">

<div class="bg-light" style="padding: 2% 5% 5% 5%; width: 120vh;">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

</div>

<footer class="border-top footer pb-5">

<div class="container-sm">

<h1 class="text-center fs-4 mb-3 mt-4">2023 - TeacherApp</h1>

<div class="d-flex justify-content-center mt-2 mb-4">

<div class="mid-solid-line"></div>

</div>

<p class="mb-1 mt-1 text-center">

Образование – это незаменимая составляющая нашей жизни.

Оно даёт возможность расширять свои знания, развивать личность и

достигать новых вершин. На сайте мы собрали для вас полезную

информацию о различных образовательных программах и курсах,

которые помогут вам реализовать свой потенциал. Мы верим, что

образование – лучший способ инвестировать в себя и своё будущее

</p>

<div class="d-flex justify-content-center mt-4 mb-4">

<div class="mid-normal-line"></div>

</div>

<p class="mb-0">Поддержите нас</p>

<a href="https://www.youtube.com/watch?v=rXbPAT62hSs">Поддержать тут</a>

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

﻿using Newtonsoft.Json;

using System.Net.Http.Headers;

using System.Text;

using UniversityContracts.ViewModels;

namespace TeacherApp

{

public class APIClient

{

public static TeacherViewModel teacher { get; set; }

private static readonly HttpClient client = new HttpClient();

public static void Connect(IConfiguration configuration)

{

client.BaseAddress = new Uri(configuration["IPAddress"]);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new

MediaTypeWithQualityHeaderValue("application/json"));

}

public static T GetRequest<T>(string requestUrl)

{

var response = client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

}

}

using TeacherApp;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddControllersWithViews();

builder.Services.AddHttpClient<APIClient>();

var app = builder.Build();

APIClient.Connect(app.Configuration);

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

else

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=SignUp}/{id?}");

});

app.Run();