МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

(ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова)

Кафедра «Программное обеспечение»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной квалификационной работе бакалавра на тему:

«Система управления учебным процессом. Серверная часть»

|  |  |
| --- | --- |
| Дипломник  студент гр. Б08-191-2 | М.Л. Поздеев |
| Руководитель | Р.О. Султанов |
| Нормоконтролер | В.П. Соболева |
| Зав. кафедрой ПО | А.В. Коробейников |

Ижевск 2020

РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе бакалавра размещена на 91 листах, содержит в себе 27 рисунков, 4 из которых являются схемами алгоритмов.

Ключевые слова: управление учебным процессом, серверная часть, серверное приложение, база данных, клиентское приложение, регистрация, авторизация, аутентификация, сервис авторизации, сервис ресурсов.

Цель работы – разработка системы, призванной повысить эффективность управления учебным процессом для студентов и преподавателей. В частности, требуется разработать серверную часть системы.

В процессе работы было проведено исследование уже существующих систем с похожей целью и выявлены их основные недостатки. Также были изучены инструменты, необходимые для выполнения поставленной задачи.

В результате работы были разработаны база данных системы и серверное приложение, предоставляющее доступ к данным, лежащим на сервере, клиентским приложениям.

Разработанная серверная часть позволяет реализовать в клиентских приложениях необходимые функции. Инструменты, которые использовались в разработке – дают возможность установить базу данных и серверное приложение на всех популярных серверных операционных системах, что позволит сократить затраты на содержание системы и на обучение администраторов.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc44000799)

[1. РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ УЧЕБНЫМ ПРОЦЕССОМ 6](#_Toc44000800)

[1.1. Обоснование целесообразности разработки системы 6](#_Toc44000801)

[1.1.1. Назначение системы 6](#_Toc44000802)

[1.1.2. Обоснование цели системы 6](#_Toc44000803)

[1.1.3. Обоснование состава автоматизируемых задач 6](#_Toc44000804)

[1.2. Аналитический обзор 7](#_Toc44000805)

[1.3. Основные требования к системе 9](#_Toc44000806)

[1.3.1. Основные цели создания системы и критерии эффективности ее функционирования 9](#_Toc44000807)

[1.3.2. Функциональное назначение системы 10](#_Toc44000808)

[1.3.3. Особенности системы, условия эксплуатации, определяющие основные требования к системе 12](#_Toc44000809)

[1.3.4. Требования к функциональной структуре системы 12](#_Toc44000810)

[1.3.5. Типовые проектные решения и (или) пакеты прикладных программ, применяемых в системе 13](#_Toc44000811)

[1.3.6. Требования к техническому обеспечению 13](#_Toc44000812)

[1.3.7. Требования к информационному обеспечению 13](#_Toc44000813)

[1.3.8. Требования к программному обеспечению 13](#_Toc44000814)

[1.3.9. Перспективность системы, возможности ее развития 14](#_Toc44000815)

[1.4. Основные технические решения проекта системы 14](#_Toc44000816)

[1.4.1. Описание организации информационной базы 14](#_Toc44000817)

[1.4.2. Описание системы программного обеспечения 17](#_Toc44000818)

[2. РАЗРАБОТКА ЗАДАЧ ПО СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ УПРАВЛЕНИЯ УЧЕБНЫМ ПРОЦЕССОМ 18](#_Toc44000819)

[2.1. Разработка сервиса авторизации 18](#_Toc44000820)

[2.1.1. Описание постановки задачи 18](#_Toc44000821)

[2.1.2. Описание алгоритма регистрации пользователей 19](#_Toc44000822)

[2.1.3. Описание алгоритма аутентификации и авторизации пользователей 21](#_Toc44000823)

[2.2. Разработка сервиса ресурсов 25](#_Toc44000824)

[2.2.1. Описание постановки задачи 25](#_Toc44000825)

[2.2.2. Описание алгоритма предоставления данных для приложений преподавателя и студента 27](#_Toc44000826)

[2.3. Описание контрольного примера 31](#_Toc44000827)

[2.3.1. Назначение 31](#_Toc44000828)

[2.3.2. Исходные данные 32](#_Toc44000829)

[2.3.4. Результаты расчета 32](#_Toc44000830)

[2.3.5. Результаты испытания программы 38](#_Toc44000831)

[ЗАКЛЮЧЕНИЕ 39](#_Toc44000832)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc44000833)

[ПРИЛОЖЕНИЕ 1 КОД ПРОГРАММЫ 41](#_Toc44000834)

[ПРИЛОЖЕНИЕ 2 РУКОВОДСТВО ОПЕРАТОРА 84](#_Toc44000837)

ВВЕДЕНИЕ

В настоящее время почти у каждого человека имеется в наличии мобильное устройство под управлением какой-либо операционной системы. Причина такой популярности мобильных устройств заключается в том, что благодаря им люди могут получать быстрый и удобный доступ к необходимым для них ресурсов в любом месте и в любое время.

В связи с этим разработчики программных продуктов адаптируют свои разработки под мобильные устройства, обеспечивая для своих пользователей удобный доступ к своим ресурсам, а кто-то вообще разрабатывает свои приложения только для мобильных устройств. Не должны оставаться позади и образовательные системы, так как использование в учебном процессе мобильных устройств позволит серьезно увеличить эффективность обучения, а также позволит поднять авторитет учебного заведения среди потенциальных студентов и преподавателей.

Для повышения эффективности обучения, система управления учебным процессом должна предоставлять студентам и преподавателям доступ к своим расписаниям, к своим личным данным, к данным обучаемых студентов или обучающих преподавателей, к новостям и объявлениям, связанным с учебой и к набору изучаемых или преподаваемых предметов с детальной информацией о них.

В некоторых учебных заведениях уже реализованы подобные системы, работающие на мобильных устройствах. Они не реализуют всех вышеперечисленных функций, но их главная проблема в том, что они привязаны к своему учебному заведению, поэтому круг пользователей сильно ограничен, что замедляет развитие таких систем.

Из всего вышесказанного следует, что цель всей работы заключается в разработке системы управления учебным процессом, которая предоставляет мобильный доступ преподавателям и студентам к важным данным в учебном процессе и не привязана к какому-либо одному учебному заведению.

Целью данной работы является разработка серверной части системы. Серверная часть системы должна иметь возможность установки на всех популярных операционных системах, чтобы любое учебное заведение могло без особых проблем развернуть ее на своей инфраструктуре, а также должна обеспечить безопасность хранимых и передаваемых на мобильные приложения данных.

1. РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ УЧЕБНЫМ ПРОЦЕССОМ

1.1. Обоснование целесообразности разработки системы

1.1.1. Назначение системы

Назначение системы заключается в увеличении эффективности ведения учебного процесса в высших учебных заведениях (ВУЗах). Эффективность учебного процесса повышается за счет предоставления пользователям системы (преподавателям и студентам) быстрого и удобного доступа к важной для учебного процесса информации через мобильные приложения.

Система предназначена для работы в большом количестве ВУЗов.

1.1.2. Обоснование цели системы

Так как в настоящее время мобильные устройства обладают огромной популярностью, открываются новые горизонты для увеличения эффективности учебного процесса в ВУЗах. Благодаря мобильным устройствам люди имеют возможность быстро и удобно получать необходимую для них информацию, но в образовательной сфере, качественных инструментов, реализующих полный потенциал мобильных устройств – просто недостаточно. Из этого и следует цель системы – предоставить участникам учебного процесса в ВУЗах универсальное программное решение, которое поможет облегчить многие аспекты в обучении.

От количества пользователей напрямую зависит качество системы и скорость ее улучшения, поэтому система предназначена для работы в разных учебных заведениях. При работе в нескольких учебных система охватит более разностороннюю аудиторию, что даст возможность для серьезного роста качества разработки. Большое количество пользователей позволит получить огромное количество отзывов и предложений, что поспособствует развитию системы, а также, увеличенный финансовый доход от системы позволит сделать куда более качественное решение, чем у аналогичных разработок, созданных и поддерживаемых на чистом энтузиазме.

1.1.3. Обоснование состава автоматизируемых задач

Текущая работа заключается в разработке серверной части системы. Задача серверной части состоит в том, чтобы предоставить мобильным приложениям единый способ доступа к данным, независимо от университета, в котором система развернута. Серверная часть должна обеспечить безопасное хранение данных, безопасность пользовательских данных (с помощью безопасного хранения паролей и безопасного способа аутентификации и авторизации) и предоставлять данные в удобном для обработки на мобильных приложениях виде.

Из всего вышесказанного следует список задач для разработки серверной части:

* разработка базы данных (БД);
* разработка сервиса авторизации;
* разработка сервиса ресурсов.

1.2. Аналитический обзор

Подобные системы, предоставляющие информацию для пользователей через мобильные приложения, существуют, например, у таких университетов, как УрФУ, ИТМО и БГУ. Однако их главная проблема заключается в том, что они привязаны только к одному учебному заведению. Отсутствие масштабируемости системы серьезно замедляет ее развитие. В ИжГТУ подобный функционал на данный момент выполняют веб-сайты университета istu.ru и ee.istu.ru, их проблема заключается в отсутствии удобного доступа через мобильные устройства и в разделении полезного функционала на несколько ресурсов. Также вышеописанные ресурсы имеют достаточное количество других проблем, но на данный момент разрабатываемая система их не решает.

Главное преимущество разрабатываемой системы в возможности масштабирования на любое количество учебных заведений. Такой возможности не имеет ни одна известная система для ВУЗов. И такую возможность можно реализовать благодаря наличию собственной серверной части системы. Детальное сравнение существующих решений с разрабатываемой системой представлено в таблице 1 (для сравнения выбраны ресурсы ИжГТУ и мобильное приложение ИТМО, так как они наиболее функциональны).

Таблица 1

Сравнение существующих систем с разрабатываемой

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ресурс istu.ru | Ресурс ee.istu.ru | Мобильное приложение ИТМО | Разрабатываемая система |
| Масштабируемость на любое количество учебных заведений | Отсутствует | Отсутствует | Отсутствует | Имеется |
| Расписание пар | Имеется (неудобно просматривать) | Отсутствует | Имеется | Имеется |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ресурс istu.ru | Ресурс ee.istu.ru | Мобильное приложение ИТМО | Разрабатываемая система |
| Список изучаемых предметов | Имеется | Имеется (студент должен сам записаться на каждый предмет) | Имеется | Имеется |
| Отдельные функции для преподавателей | Имеются | Имеются | Отсутствуют | Имеются |
| Мобильное приложение | Отсутствует | Отсутствует | Имеется | Имеется |
| Веб версия | Имеется | Имеется | Отсутствует | На данный момент отсутствует |
| Просмотр списка группы студентов | Отсутствует | Имеется (только записанных на курс) | Отсутствует | Имеется |
| Просмотр информации о преподавателях | Имеется | Имеется | Отсутствует | Имеется |
| Просмотр новостей ВУЗа | Имеется | Отсутствует | Отсутствует | Имеется |

В таблице выше перечислены не все возможности и недочеты системы, но перечисленного достаточно для понимания, в какую сторону будет развиваться система, ее архитектура позволяет с легкостью добавлять новый функционал, а установка системы в нескольких учебных заведениях потенциально даст большие финансовые возможности для развития.

В ИжГТУ уже существует серверная инфраструктура, позволяющая университетским ресурсам получать некоторые данные, которые могли бы пригодиться в разрабатываемой системе, но так как разрабатываемая система должна иметь возможность установки в разных ВУЗах, она должна иметь свою собственную полностью независимую серверную часть. Для обеспечения безопасности данных каждого ВУЗа, разрабатываемая серверная часть должна будет устанавливаться на инфраструктуре каждого учебного заведения, которое будет использовать систему. Чтобы серверную часть можно было развернуть без технических проблем в любом ВУЗе, она должна иметь возможность разворачивания на любой популярной операционной системе (ОС) и быть максимально дешевой в обслуживании. Поэтому для разработки серверной части были выбраны кроссплатформенные инструменты: СУБД PostgreSQL, платформа разработки общего назначения с открытым кодом .NET Core и язык программирования C#. Выбор пал на эти инструменты, так как они достаточно популярны (не трудно будет найти разработчиков), дешевы в обслуживании и достаточно просты в эксплуатации.

Для разработки API для взаимодействия мобильного приложения с сервером – была выбрана технология платформы .NET Core – ASP.NET Core Web API. Данная технология предоставляет удобные способы для обработки HTTP-запросов и технологию для взаимодействия с СУБД из программного кода – Entity Framework Core. Выбранные технологии позволят без особого труда в дальнейшем разрабатывать новые клиентские приложения, например, для браузеров.

1.3. Основные требования к системе

1.3.1. Основные цели создания системы и критерии эффективности ее функционирования

Основные цели создания системы:

* предоставление важной в учебном процессе информации для преподавателей и студентов через мобильное приложение;
* возможность установки системы на инфраструктуре любого ВУЗа.

В связи с вышеперечисленными целями требуется выполнять следующие требования:

* серверная часть должна иметь возможность простой установки на всех популярных ОС;
* приложение должно иметь раздельный функционал для студента и преподавателя;
* доступ к информации должен быть осуществлен через разрабатываемое программный интерфейс приложения (API);
* должна иметься возможность переноса в базу данных системы необходимых данных университета, с соблюдением требований информационной безопасности;
* архитектура системы должна быть построена таким образом, функционал системы можно было без труда увеличивать.

1.3.2. Функциональное назначение системы

На данный момент система предоставит для пользователей данные, описанные ниже и возможность регистрации и авторизации.

Приложение для студентов:

* расписание занятий группы студента;
* список изучаемых предметов;
* информация об изучаемом предмете;
* нумерованный список студентов группы;
* информация о преподавателях;
* информация о студентах группы;
* новости ВУЗа;
* личный профиль с возможностью редактирования контактной информации.

Диаграмма вариантов использования приложения для студентов представлена на рисунке 1.1.

Диаграмма вариантов использования (студент)

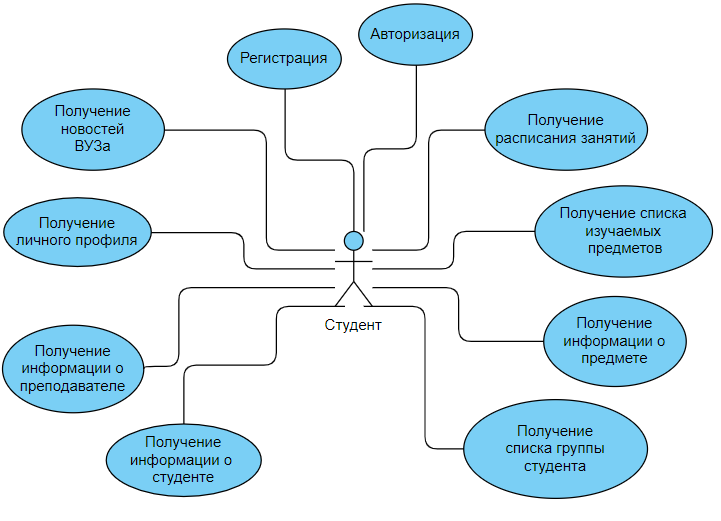


Рис.1.1

Приложение преподавателя:

* расписание занятий преподавателя;
* список обучаемых групп студентов;
* профиль обучаемого студента;
* список предметов, преподаваемых в группах;
* информация о преподаваемом предмете;
* новости ВУЗа;
* личный профиль с возможностью редактирования контактной информации.

Диаграммы вариантов использования для приложения преподавателей представлена на рисунке 1.2.

Диаграмма вариантов использования

(преподаватель)

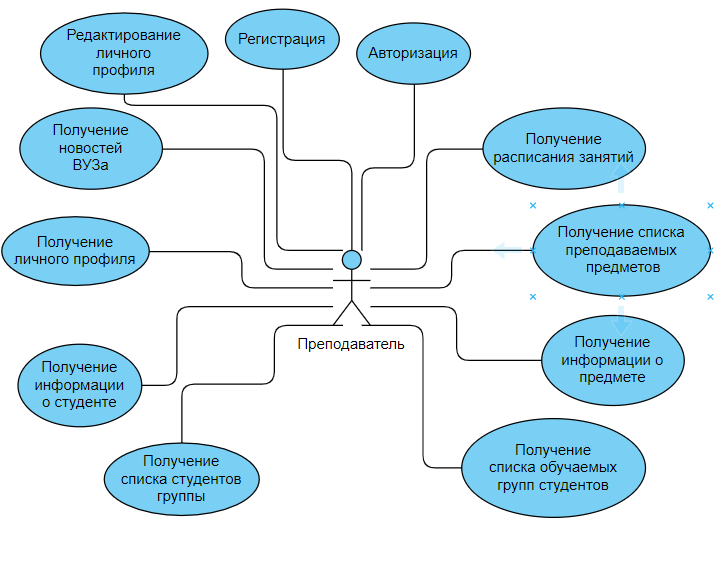


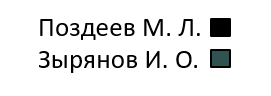
Рис.1.2

1.3.3. Особенности системы, условия эксплуатации, определяющие основные требования к системе

Серверная часть системы будет устанавливаться на инфраструктуре учебного заведения. Способы установки разные в зависимости от операционной системы, поэтому данный вопрос необходимо будет обсуждать дополнительно с каждым ВУЗом-клиентом. В зависимости от потенциального количества пользователей системы в одном ВУЗе – будут рассчитываться требуемые вычислительные мощности для сервера.

1.3.4. Требования к функциональной структуре системы

Структурная схема системы представлена на рисунке 1.3. Разделение частей между разработчиками выделено соответствующим цветом.

Структурная схема

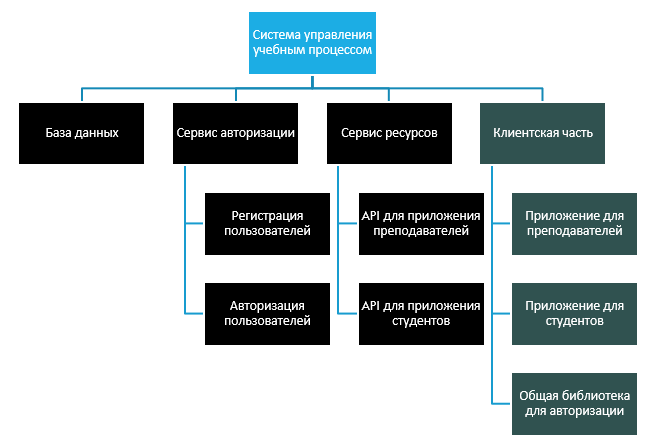


Рис.1.3

Часть системы, разрабатываемая в данной работе, состоит из 3 частей:

* база данных. Хранилище всех данных системы, развернутое на инфраструктуре ВУЗа;
* сервис авторизации. Сервис, отвечающий за предоставление доступа для пользователей;
* сервис ресурсов. Сервис, отвечающий за обеспечение взаимодействия мобильного приложения и базы данных.

Сервис авторизации состоит из следующих модулей:

* модуль регистрации пользователей. Модуль получает данные регистрируемых пользователей и с соблюдением правил информационной безопасности сохраняет их в БД;
* модуль авторизации. Модуль идентифицирует пользователей и предоставляет им доступ к ресурсам системы.

Сервис ресурсов состоит из следующих модулей:

* модуль API для приложения преподавателей. Модуль предоставляет данные, предназначенные для преподавательского приложения;
* модуль API для приложения студента. Модуль предоставляет данные, предназначенные для студенческого приложения.

1.3.5. Типовые проектные решения и (или) пакеты прикладных программ, применяемых в системе

Для разработки сервиса авторизации и сервиса ресурсов используется платформа разработки общего назначения с открытым кодом .NET Core и язык программирования C#, как удобное кроссплатформенное решение. Для базы данных используется СУБД PostgreSQL.

Применяемые средства разработки: среда разработки .NET Core – Visual Studio 2019, среда разработки СУБД PostgreSQL – dbForge for PostgreSQL Express.

1.3.6. Требования к техническому обеспечению

Минимальные требования к техническому обеспечению сервера:

* процессор: Intel Pentium 4, 2 ГГц;
* размер оперативной памяти: 1 Гб;
* объем жесткого диска: 1 Гб.

1.3.7. Требования к информационному обеспечению

Информационное обеспечение системы составляет СУБД PostgreSQL.

1.3.8. Требования к программному обеспечению

На сервере должно быть установлено следующее программное обеспечение:

* .NET Core версии 3.1;
* веб-сервер Nginx версии 1.17.10;
* PostgreSQL версии 12.

1.3.9. Перспективность системы, возможности ее развития

В перспективе система может дополняться различным функционалом, к примеру:

* чат с преподавателем;
* веб-версия приложения;
* iOS-версия приложения.

А также возможны серьезные улучшения текущего функционала, такие как:

* текущий статус преподавателя в его профиле (в каком кабинете, на больничном, в командировке и так далее);
* возможность создавать объявления;
* отображение зачетов или оценок по предметам;
* отображение заработанных баллов по предмету за семестр.

Система разработана таким образом, чтобы ее можно было без труда улучшать.

1.4. Основные технические решения проекта системы

1.4.1. Описание организации информационной базы

Ниже представлены описания всех таблиц, создаваемых в рамках текущей работы.

Таблица staff содержит информацию о сотрудниках ВУЗа:

* id – внутренний идентификатор;
* name – имя;
* surname – фамилия;
* patronymic – отчество.

Таблица staff\_requisites содержит личную информацию сотрудников ВУЗа:

* staff\_id – внешний идентификатор сотрудника;
* e\_mail – почтовый адрес;
* mobile\_phone – номер мобильного телефона;
* home\_phone – номер домашнего телефона.

Таблица institutes содержит информацию об институтах и факультетах ВУЗа:

* id – внутренний идентификатор;
* name – наименование;
* short\_name – краткое наименование;
* director\_id – внешний идентификатор сотрудника руководителя (директора или ректора);
* director\_deputy\_id – внешний идентификатор сотрудника заместителя руководителя.

Таблица departments содержит информацию о кафедрах:

* id – внутренний идентификатор;
* institute\_id – внешний идентификатор института или факультета;
* name – наименование;
* short\_name – краткое наименование;
* director\_id – внешний идентификатор сотрудника руководителя.

Таблица staff\_departments содержит связи между сотрудниками ВУЗа и кафедрами:

* department\_id – внешний идентификатор кафедры;
* staff\_id – внешний идентификатор сотрудника.

Таблица subject\_information содержит информацию о предметах:

* id – внутренний идентификатор;
* name – наименование;
* short\_name – краткое наименование;
* description – описание.

Таблица subjects содержит связи между сотрудниками (преподавателями) и информацией о предметах:

* id – внутренний идентификатор;
* lecturer\_id – внешний идентификатор сотрудника преподавателя;
* information\_id – внешний идентификатор информации о предмете.

Таблица sub\_subjects содержит информацию о частях основного предмета (лабораторные и т.д.):

* id – внутренний идентификатор;
* subject\_id – внешний идентификатор предмета (таблица subjects);
* name – наименование;
* short\_name – краткое наименование;
* sub\_lecturer\_id – внешний идентификатор сотрудника преподавателя.

Таблица students содержит данные о студентах:

* id – внутренний идентификатор;
* student\_number – номер зачетной книжки;
* name – имя;
* surname – фамилия;
* patronymic – отчество.

Таблица groups содержит информацию о студенческих группах:

* id – внутренний идентификатор;
* name – наименование;
* department\_id – внешний идентификатор кафедры;
* head\_id – внешний идентификатор студента старосты группы.

Таблица groups\_subjects содержит связи между студенческими группами и предметами:

* group\_id – внешний идентификатор группы;
* subject\_id – внешний идентификатор предмета;

Таблица student\_requisites содержит личную информацию студентов:

* student\_id – внешний идентификатор студента;
* group\_id – внешний идентификатор группы студента;
* e\_mail – почтовый адрес;
* mobile\_phone – номер мобильного телефона;
* home\_phone – номер домашнего телефона.

Таблица classes содержит информацию о времени проведения пар в ВУЗе:

* id – внутренний идентификатор;
* start\_time – время начала;
* end\_time – время окончания;
* number – порядковый номер (первая пара, вторая и т.д.).

Таблица schedule содержит информацию о расписании пар:

* id – внутренний идентификатор;
* class\_id – внешний идентификатор пары (таблица classes);
* group\_id – внешний идентификатор группы студентов;
* subject\_id – внешний идентификатор предмета;
* sub\_subject\_id – внешний идентификатор части предмета;
* date – день недели проведения пары;
* week\_line – индикатор недели под чертой/над чертой;
* location – место проведения пары.

Таблица news содержит новости:

* id – внутренний идентификатор;
* text – текст новости;
* title – заголовок новости;
* date – дата новости.

Таблица news\_receivers содержит информацию о том, какие группы пользователей будут получателями новостей:

* news\_id – внешний идентификатор новости;
* group\_id – внешний идентификатор студенческой группы;
* staff\_id – внешний идентификатор сотрудника университета;
* department\_id – внешний идентификатор кафедры ВУЗа;
* institute\_id – внешний идентификатор института или факультета.

Таблица student\_auth\_data содержит данные зарегистрированных студентов:

* student\_id – внешний идентификатор студента;
* username – имя пользователя в системе;
* password – пароль пользователя в системе;
* salt – добавочная соль для сохранения пароля;
* refresh\_token – токен восстановления доступа.

Таблица staff\_auth\_data содержит данные зарегистрированных сотрудников ВУЗа:

* staff\_id – внешний идентификатор сотрудника;
* username – имя пользователя в системе;
* password – пароль пользователя в системе;
* salt – добавочная соль для сохранения пароля;
* refresh\_token – токен восстановления доступа.

База данных спроектирована с учетом всевозможных ситуаций и в ее структуре сможет без труда разобраться любой администратор. Схема базы данных, описанной выше представлена на рисунке 1.4.

Схема базы данных



Рис. 1.4

1.4.2. Описание системы программного обеспечения

Серверная часть представляет собой приложение, созданное с помощью платформы .NET Core версии 3.1, с использованием технологии ASP.NET Core Web API. Хранилище данных реализовано с помощью СУБД PostgreSQL версии 12. Система устанавливается на следующих ОС:

* Alpine: 3.10+;
* Debian: 9+;
* Ubuntu 16.04+;
* Fedora: 29+;
* RHEL: 6+;
* openSUSE: 15+;
* SUSE Enterprise Linux (SLES): 12 SP2+;
* macOS: 10.13+;
* Windows Client: 7, 8.1, 10 (1607+);
* Windows Server: 2012 R2+.

2. РАЗРАБОТКА ЗАДАЧ ПО СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ УПРАВЛЕНИЯ УЧЕБНЫМ ПРОЦЕССОМ

2.1. Разработка сервиса авторизации

2.1.1. Описание постановки задачи

2.1.1.1. Характеристика задачи

Задача – разработать сервис авторизации.

Сервис авторизации разрабатывается с целью обеспечения безопасности данных, хранимых в системе. Для реализации возможности масштабирования системы в будущем на большое количество университетов – было решено разработать собственный сервис авторизации, который будет предоставлять доступ мобильным приложениям пользователей к API системы.

Разработка сервиса авторизации связана с разработкой сервиса ресурсов системы. С помощью проверки доступа к методам API сервиса ресурсов – будет проверяться работоспособность сервера авторизации.

Разработка задачи будет завершена при успешном прохождении тестирования сервера.

2.1.1.2. Входная информация

Перечень входных данных при регистрации пользователей:

* номер зачетной книжки (в случае, если пользователь – студент);
* идентификатор преподавателя в системе (в случае, если пользователь – преподаватель);
* уникальный логин пользователя;
* пароль.

Перечень входных данных при аутентификации и авторизации пользователей:

* логин пользователя, созданный при регистрации;
* пароль пользователя, созданный при регистрации.

Перечень входных данных при валидации токена доступа: токен доступа.

Перечень входных данных при обновлении токена доступа:

* токен восстановления доступа;
* старый токен доступа.

Вся вышеперечисленная информация передается на сервер с помощью HTTP-запросов к методам API.

2.1.1.3. Выходная информация

Перечень выходных данных при регистрации пользователей: сообщение об успешной или о неудачной регистрации.

Перечень выходных данных при аутентификации и авторизации пользователей:

* токен доступа;
* токен восстановления доступа;
* идентификатор пользователя в системе.

Перечень выходных данных при валидации токена доступа: сообщение об успешной или неудачной валидации.

Перечень выходных данных при обновлении токена доступа:

* новый токен доступа при успешном обновлении токена;
* новый токен восстановления доступа при успешном обновлении токена;
* идентификатор пользователя в системе при успешном обновлении токена;
* ошибка обновления токена при не правильных входных данных.

Вся вышеперечисленная информация выдается сервером в ответ на HTTP-запросы.

2.1.2. Описание алгоритма регистрации пользователей

2.1.2.1. Назначение и характеристика

Регистрация необходима для создания пользователя в системе и его сопоставления с сущностью студента или преподавателя в базе данных.

Алгоритм должен сохранять данные пользователей в базе данных, при этом проверив существование этого пользователя как студента или преподавателя в системе. Пароли пользователей должны записываться в базу в виде хеша таким образом, чтобы одинаковые пароли имели разное представление в базе данных. Полученный хеш должен быть устойчивым к атакам перебора паролей.

2.1.2.2. Используемая информация

Для алгоритма регистрации используются:

* номер зачетной книжки студента или идентификатор преподавателя в системе;
* уникальный логин пользователя;
* пароль пользователя.

2.1.2.3. Результаты решения

В результате выполнения алгоритма формируется следующая информация:

* оповещение клиентского приложения об успешной или неудачной регистрации;
* сохранение регистрационных данных пользователя (логин, пароль в виде хеша, идентификатор пользователя) в базе данных.

2.1.2.4. Алгоритм решения

Схема алгоритма представлена на рисунке 2.1.

Схема алгоритма регистрации



Рис. 2.1

Описание алгоритма:

1. сервер авторизации получает регистрационные данные от клиентского приложения;
2. сервер авторизации сопоставляет полученный номер зачетной книжки или идентификатор преподавателя с данными в базе;
3. если такого студента или преподавателя не существует, то сервер посылает на клиентское приложение ответ с сообщением об ошибке регистрации и алгоритм завершается;
4. если такой студент или преподаватель существует в университете, то алгоритм выполняется дальше;
5. сервер авторизации генерирует «соль» для хеширования пароля – случайный текст, который добавится к введенному паролю для обеспечения уникальности хешей при введении пользователями одинаковых паролей;
6. сервер генерирует хеш пароля и добавлением «соли»;
7. сервер сохраняет регистрационные данные в базу данных пользователей системы;
8. сервер посылает на клиент ответ с сообщением об успешной регистрации и алгоритм завершается.

2.1.3. Описание алгоритма аутентификации и авторизации пользователей

2.1.3.1. Назначение и характеристика

Аутентификация и авторизация нужны для обеспечения безопасности данных каждого пользователя системы. С помощью аутентификации и авторизации защищается вся личная информация пользователей. Аутентификация в системе необходима для подтверждения идентичности пользователя. Авторизация в системе необходима для предоставления пользователю прав доступа на необходимые ресурсы.

Алгоритм должен сверять данные пользователя (логин и пароль), пришедшие с клиентского приложения с данными в базе данных, и если все указано верно, то алгоритм должен предоставить доступ пользователю клиентского приложения с помощью токена доступа к необходимым для него ресурсам. Время действия токена должно быть ограничено, для обеспечения безопасности. Алгоритм также должен предоставить метод обновления токена доступа с помощью токена восстановления доступа, для того чтобы токен доступа можно было восстановить без введения логина и пароля пользователя. В добавок к токенам, алгоритм должен отправлять на клиентское приложение идентификатор пользователя в системе.

2.1.3.2. Используемая информация

При первой аутентификации используются:

* логин пользователя;
* пароль пользователя.

При наличии токена доступа для аутентификации и авторизации используется этот токен доступа.

При обновлении токена доступа используются:

* токен доступа;
* токен восстановления доступа.

2.1.3.3. Результаты решения

В результате выполнения алгоритма при первой аутентификации и при обновлении токена доступа формируется следующая информация:

* токен доступа;
* токен восстановления доступа;
* идентификатор пользователя в системе.

Токен восстановления доступа сохраняется в базе данных пользователей. Все сформированные данные отправляются на клиентское приложение.

При наличии актуального токена – пользователю предоставляется доступ к необходимым ресурсам.

2.1.3.4. Алгоритм решения

Схема алгоритма при наличии токена доступа не представлена, этот вариант рассмотрен в текстовом описании алгоритма. Схемы алгоритма при первой аутентификации и при обновлении токена доступа представлены на рисунках 2.2 и 2.3, соответственно.

|  |  |
| --- | --- |
| Схема алгоритма при  первой аутентификации | Схема алгоритма при  обновлении токена доступа |
|  |  |
| Рис. 2.2 | Рис. 2.3 |

Описание алгоритма при первой аутентификации пользователя:

1. сервер получает данные для аутентификации с клиентского приложения (логин и пароль);
2. сервер сверяет введенные данные с данными из БД, пароль сверяется с помощью вычисление хеша как в алгоритме регистрации пользователей;
3. если данные введены неверно, то на клиентское приложение возвращается ошибка;
4. если данные введены верно, то сервер генерирует токен доступа и токен восстановления доступа;
5. сервер сохраняет токен восстановления доступа в базе данных;
6. сервер отправляет на клиент токен доступа, токен восстановления доступа и идентификатор пользователя в системе.

Описание алгоритма при наличии токена доступа:

1. сервер получает токен доступа;
2. сервер проверяет валидность полученного токена доступа;
3. если токен не валиден, то на клиентское приложение отправляется ошибка авторизации с информацией, показывающей, почему токен не валиден (если время действия токена истекло);
4. если токен валиден, то сервер предоставляет пользователю доступ к запрашиваемому ресурсу.

Описание алгоритма при восстановлении токена доступа:

1. сервер получает токен доступа и токен восстановления доступа;
2. сервер проверяет валидность токена доступа (опуская проверку времени действия токена);
3. если токен доступа не валиден, то сервер отправляет на клиентское приложение ошибку с сообщением о неправильном токене доступа;
4. сервер проверяет валидность токена восстановления доступа;
5. если токен восстановления доступа не валиден, то сервер отправляет на клиентское приложение ошибку с сообщением о неправильном токене восстановления доступа;
6. если полученные токены валидны, то сервер генерирует токен доступа и токен восстановления доступа;
7. сервер сохраняет токен восстановления доступа в базе данных;
8. сервер отправляет на клиент токен доступа, токен восстановления доступа и идентификатор пользователя в системе.

2.2. Разработка сервиса ресурсов

2.2.1. Описание постановки задачи

2.2.1.1. Характеристика задачи

Задача – разработка сервиса ресурсов.

Сервис ресурсов разрабатывается с целью предоставления клиентским приложениям для студентов и преподавателей интерфейса взаимодействия с базой данных системы. Для реализации возможности масштабирования системы в будущем на большое количество университетов – было решено спроектировать собственную базу данных и разработать сервис ресурсов, который будет предоставлять доступ (API) мобильным приложениям пользователей к данным системы.

Разработка сервиса ресурсов связана с разработкой сервиса авторизации, так как через сервис авторизации пользователям будет предоставляться доступ к методам сервера ресурсов.

Разработка задачи будет завершена при успешном прохождении тестирования сервера.

2.2.1.2. Входная информация

Вся входная информация приходит в составе http запросов.

Практически для всех запросов студенческого приложения используется идентификатор студента, выданный приложению на этапе авторизации. Ниже представлены варианты запросов с данными кроме идентификатора студента.

Входная информация при запросе информации о предмете: идентификатор предмета.

Входная информация при запросе информации о преподавателе: идентификатор преподавателя.

Входная информация при редактировании личного профиля:

* идентификатор студента;
* редактируемая информация.

Практически для всех запросов преподавательского приложения используется идентификатор преподавателя, выданный на этапе авторизации. Ниже представлены варианты запросов с данными кроме идентификатора преподавателя:

Входная информация при запросе информации о предмете: идентификатор предмета.

Входная информация при запросе списка студентов одной из обучаемых групп: идентификатор группы студентов.

Входная информация при запросе списка преподаваемых предметов в группе:

* идентификатор преподавателя;
* идентификатор группы студентов.

Входная информация при запросе информации о студенте: идентификатор студента.

Входная информация при редактировании личного профиля:

* идентификатор преподавателя;
* редактируемая информация.

2.2.1.3. Выходная информация

Вся выходная информация отправляется в качестве ответа на http запрос. Для получения все данных – необходим токе доступа, который формируется сервисом авторизации.

Ниже представлена выходная информация студенческого приложения.

Выходная информация при запросе расписания: расписание на запрошенный день в виде списка предметов с датами.

Выходная информация при запросе информации о предмете: данные по предмету (описание, преподаватель).

Выходная информация при запросе списка студентов группы: список студентов группы.

Выходная информация при запросе списка изучаемых предметов группы: список изучаемых предметов группы.

Выходная информация при запросе информации о преподавателе: информация о преподавателе.

Выходная информация при запросе личного профиля: информация из личного профиля студента пользователя.

Выходная информация при редактировании личного профиля: сообщение об успешном редактировании.

Выходная информация при запросе новостей и объявлений: список новостей и объявлений.

Далее представлена выходная информация преподавательского приложения.

Выходная информация при запросе расписания: расписание в виде списка предметов с датами.

Выходная информация при запросе информации о предмете: данные по предмету (описание, преподаватель).

Выходная информация при запросе списка обучаемых групп: список обучаемых групп.

Выходная информация при запросе списка студентов одной из обучаемых групп: список студентов группы.

Выходная информация при запросе списка преподаваемых предметов в группе: список преподаваемых предметов в группе.

Выходная информация при запросе данных о студенте: данные о студенте.

Выходная информация при запросе личного профиля: информация из личного профиля преподавателя.

Выходная информация при редактировании личного профиля: сообщение об успешном редактировании.

Выходная информация при запросе новостей и объявлений: список новостей.

2.2.2. Описание алгоритма предоставления данных для приложений преподавателя и студента

2.2.2.1. Назначение и характеристика

Алгоритм предоставления данных для приложений преподавателя и студента необходим для сбора и формирования в удобный вид данных из БД, с последующей отправкой их на клиентское приложение.

Алгоритм должен уметь принимать запросы от клиентских приложений и маршрутизировать каждый запрос по назначению. Количество методов API должно быть достаточным, чтобы не загружать сеть большим количеством данных. Алгоритм должен вытаскивать данные для приложения преподавателя из БД, при этом формируя их в тот вид, в котором они необходимы в клиентском приложении, чтобы как можно больше уменьшить количество бизнес-логики на клиенте, что поспособствует увеличению производительности. Алгоритм должен предоставлять доступ к своим функциям только при наличии валидного токена доступа, который можно получить, используя сервис авторизации. Валидация токена доступа в алгоритме должна происходить с использованием сервиса авторизации.

2.2.2.2. Используемая информация

Для авторизации пользователя используется токен доступа.

Информация, которая используется только для приложения преподавателя или только для приложения студента – подписана соответствующим образом. Информация, используемая в обоих случаях – не подписана.

При выполнении запроса на выдачу расписания используется следующая информация:

* идентификатор преподавателя-пользователя (приложение преподавателя);
* данные таблицы расписания занятий из БД;
* данные таблицы студенческих групп из БД (приложение преподавателя);
* данные таблицы преподавателей из БД (приложение студента);
* данные таблицы предметов из БД.

При выполнении запроса на выдачу данных об определенном предмете используется следующая информация:

* идентификатор этого предмета;
* данные таблицы предметов из БД.

При выполнении запроса на выдачу списка обучаемых преподавателем студенческих групп используется следующая информация (приложение преподавателя):

* идентификатор преподавателя, выполняющего запрос;
* данные таблицы, связывающей студенческие группы и преподавателей из БД.

При выполнении запроса на выдачу списка студентов одной из групп используется следующая информация:

* идентификатор требуемой группы;
* данные таблицы студенческих групп из БД.

При выполнении запроса на выдачу списка преподаваемых предметов в определенной группе используется следующая информация (приложение преподавателя):

* идентификатор группы;
* идентификатор преподавателя, выполняющего запрос;
* данные таблицы предметов из БД;
* данные таблицы связывающей предметы и группы из БД.

При выполнении запроса на выдачу данных о студенте используется следующая информация:

* идентификатор требуемого студента;
* данные таблицы реквизитов студентов из БД.

При выполнении запроса на выдачу данных о преподавателе используется следующая информация:

* идентификатор требуемого преподавателя;
* данные таблицы реквизитов сотрудников университета из БД.

При выполнении запроса на редактирование данных из личного профиля используется следующая информация:

* идентификатор преподавателя-пользователя (приложение преподавателя);
* идентификатор студента-пользователя (приложение студента);
* редактируемые данные;
* данные таблицы реквизитов сотрудников университета из БД (приложение преподавателя);
* данные из таблицы реквизитов студентов (приложение студента);

При выполнении запроса на выдачу новостей используется следующая информация:

* идентификатор преподавателя-пользователя (приложение преподавателя);
* идентификатор студента-пользователя (приложение студента);
* данные таблицы новостей из БД;

При выполнении запроса на выдачу списка изучаемых студентом предметов используется следующая информация (приложение студента): идентификатор студента-пользователя.

2.2.2.3. Результаты решения

В результате валидации токена доступа в сервисе авторизации при неудаче передается сообщение, сформированное в сервисе авторизации и при успехе предоставляется доступ к запрашиваемому методу.

Информация, которая формируется только для приложения преподавателя или только для приложения студента – подписана соответствующим образом. Информация, формируемая в обоих случаях – не подписана.

В результате выполнения запроса на выдачу расписания формируется список предметов. Объект предмета для расписания включает в себя:

* идентификатор предмета;
* дата и время проведения пары;
* наименование студенческой группы, в которой будет проводиться пара (приложение преподавателя);
* ФИО преподавателя (приложение студента).

В результате выполнения запроса на выдачу данных об определенном предмете формируется объект предмета, в составе которого:

* идентификатор предмета;
* описание предмета;
* идентификатор преподавателя (приложение студента);
* ФИО преподавателя (приложение студента).

В результате выполнения запроса на выдачу списка обучаемых преподавателем студенческих групп формируется список групп (приложение преподавателя). Объект группы включает в себя:

* идентификатор группы;
* наименование группы.

В результате выполнения запроса на выдачу списка студентов одной из групп, формируется список студентов. Объект студента включает в себя:

* идентификатор студента;
* ФИО студента.

В результате выполнения запроса на выдачу данных о студенте формируется объект студента, в составе которого:

* идентификатор студента;
* ФИО студента;
* почтовый адрес студента;
* номер телефона студента;

В результате выполнения запроса на выдачу данных о преподавателе формируется объект преподавателя, в составе которого:

* идентификатор преподавателя;
* ФИО преподавателя;
* почтовый адрес преподавателя;
* номер телефона преподавателя;

В результате запроса на редактирование данных из личного профиля пользователя формируется сообщение об успешном редактировании и те же результаты, что и при выдаче данных о преподавателе или студенте.

2.2.2.4. Алгоритм решения

Схема алгоритма представлена на рисунке 2.4.

Схема алгоритма предоставления данных



Рис. 2.4

Описание алгоритма:

1. при поступлении запроса происходит определение метода, который его выполнит (маршрутизация запроса);
2. на данный момент ко всем методам сервиса ресурсов отсутствует анонимный доступ, поэтому перед выполнением запроса происходит валидация токена доступа пользователя с помощью сервиса авторизации;
3. если токен доступа пользователя не валиден, то сервис ресурсов отказывает пользователю в доступе;
4. если токен валиден, то сервис ресурсов выполняет необходимый запрос.

2.3. Описание контрольного примера

2.3.1. Назначение

Контрольным примером проверяются:

* регистрация пользователя в системе;
* занесение нового пользователя в базу;
* хеширование пароля.
* выдача токена доступа и токена восстановления доступа пользователю;
* выдача токенов при первой аутентификации;
* выдача токенов при восстановлении доступа;
* сохранение токена восстановления в базе.
* доступ к методам сервиса ресурсов;
* доступ при валидном токене доступа;
* доступ при не валидном токене доступа;
* работа метода по предоставлению данных о студенте.

Контрольный пример должен представить принцип работы всей разрабатываемой части системы и показать каким образом предоставленные данные используются в мобильном приложении. А также контрольный пример должен показать общую работоспособность системы.

2.3.2. Исходные данные

Исходные данные не являются чьими-то персональными данными и придуманы случайным образом с целью тестирования системы.

Перечень исходных данных:

* запись в базе о группе Б08-191-2 c идентификатором 1;
* запись в базе о студенте Поздееве Максиме Львовиче с номером зачетной книжки 00345678 и идентификатором группы 1, номером мобильного телефона +7(999)-123-49-41, номером домашнего телефона +7(3412)555351;
* логин, который будет использоваться для регистрации этого студента в системе PozdeevML;
* пароль, который будет использоваться для регистрации этого студента в системе qwerty.

2.3.4. Результаты расчета

Результат проверки HTTP-запроса на регистрацию пользователя представлен на рисунке 2.5.

Пример работы регистрации

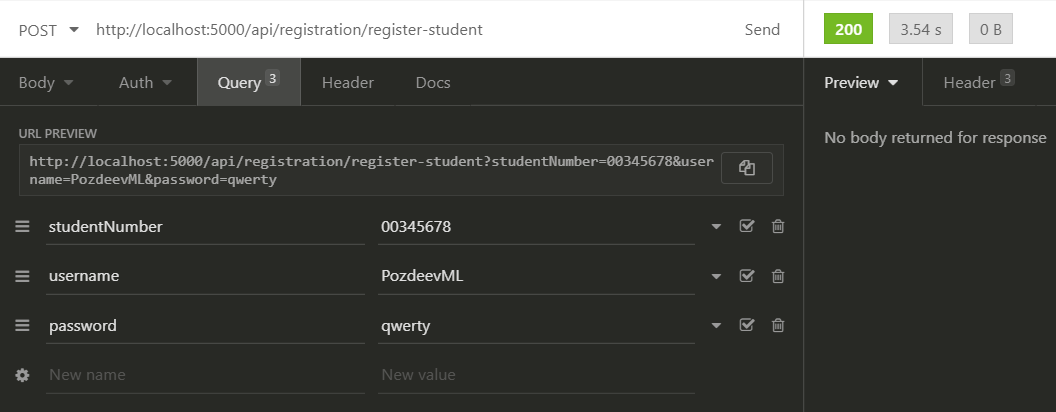


Рис. 2.5

Результат сохранения данных пользователя в базу данных представлен на рисунке 2.6.

Пример занесения зарегистрированного пользователя в базу



Рис. 2.6

Пример HTTP-запроса на авторизацию представлен на рисунке 2.7.

Пример HTTP-запроса для первой авторизации

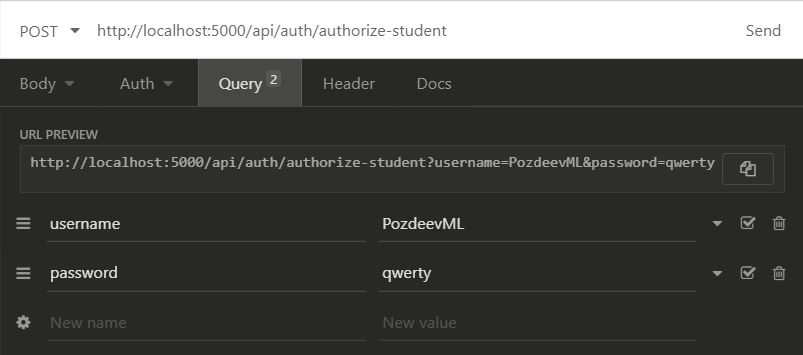


Рис. 2.7

Ответ сервера на правильный HTTP-запрос об авторизации представлен на рисунке 2.8.

Ответ сервера при успешной авторизации

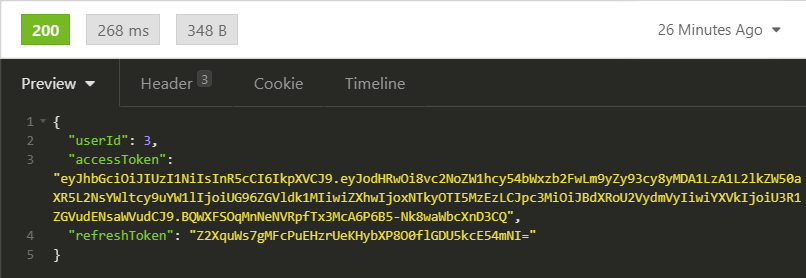


Рис. 2.8

Ответ сервера при передаче в HTTP-запросе на авторизацию неверных данных пользователя представлен на рисунке 2.9.

Ответ сервера при ошибке авторизации

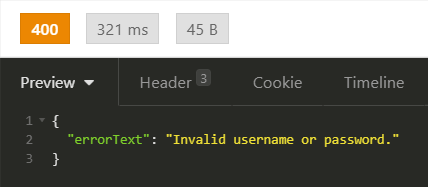


Рис. 2.9

Пример HTTP-запроса на обновление токена доступа представлен на рисунке 2.10.

Пример HTTP-запроса на обновление токена доступа

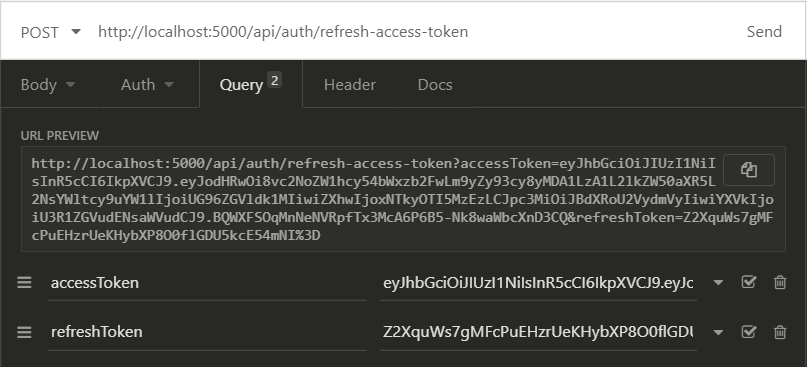


Рис. 2.10

Ответ сервера при успешном обновлении токена доступа представлен на рисунке 2.11.

Ответ сервера при успешном обновлении токена доступа

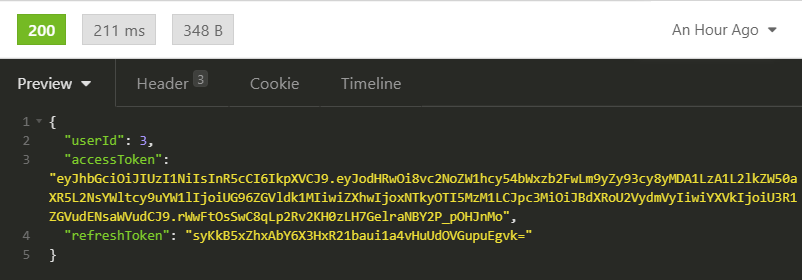


Рис. 2.11

Ответ сервера при передаче в HTTP-запросе на обновление токена неверного старого токена доступа представлен на рисунке 2.12.

Ответ сервера при неверном старом токене доступа

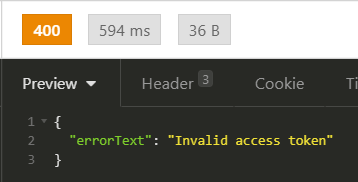


Рис. 2.12

Ответ сервера при передаче в HTTP-запросе неверного токена восстановления доступа представлен на рисунке 2.13.

Ответ сервера при неверном токене восстановления доступа

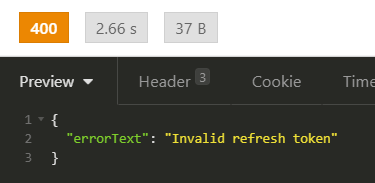


Рис. 2.13

Результат сохранения токена восстановления доступа в базе данных представлен на рисунке 2.14.

Пример сохранения токена восстановления в базе

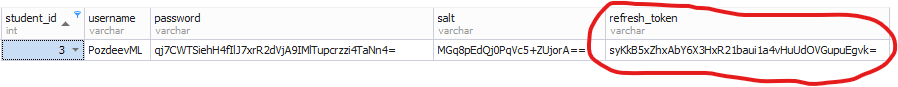


Рис. 2.14

Пример HTTP-запроса на предоставление данных профиля пользователя представлен на рисунке 2.15.

Пример HTTP-запроса на предоставление данных профиля пользователя

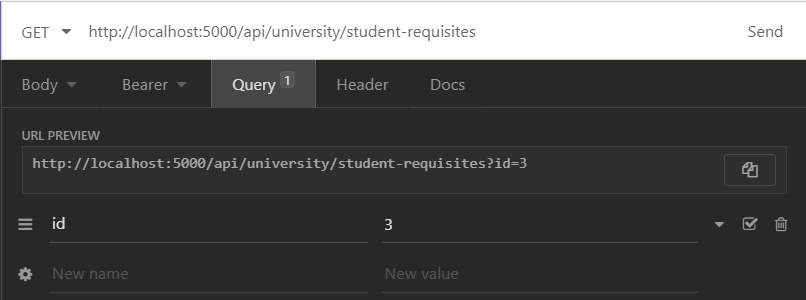


Рис. 2.15

Ответ сервера при правильно введенных данных в HTTP-запросе представлен на рисунке 2.16.

Ответ сервера при правильных данных в запросе

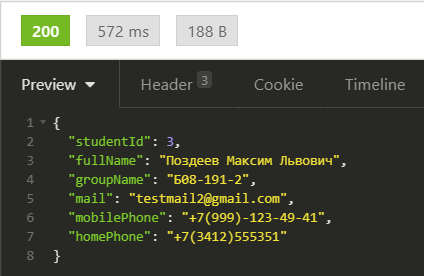


Рис. 2.16

Ответ сервера при неправильном токене доступа в HTTP-запросе представлен на рисунке 2.17.

Ответ сервера при неправильном токене доступа

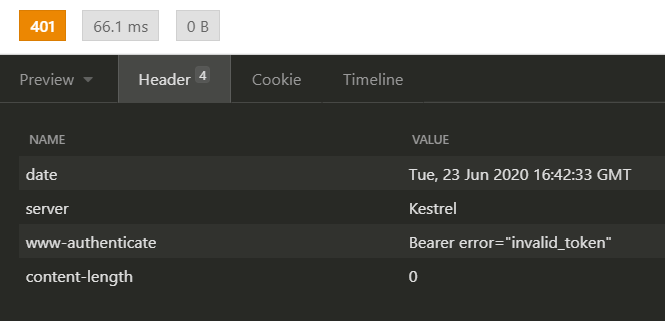


Рис. 2.17

Пример экранной формы профиля пользователя в мобильном приложении, получившим данные с сервера – представлен на рисунке 2.18.

Пример профиля пользователя в мобильном приложении

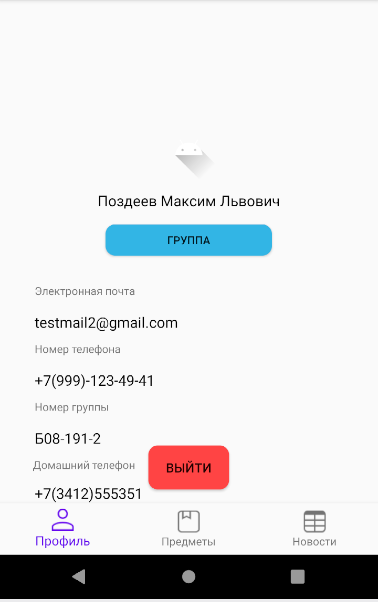


Рис. 2.18

2.3.5. Результаты испытания программы

Для проверки работы сервисов использовались: программа для тестирования API – Insomnia, разработанное в рамках системы мобильное приложение для устройств под управлением ОС Android, среда разработки СУБД PostgreSQL – dbForge for PostgreSQL, среда разработки платформы .NET Core – Visual Studio 2019.

По результатам проверки удалось выяснить, что разработанная часть системы функционирует безошибочно.

ЗАКЛЮЧЕНИЕ

В результате данной работы была разработана серверная часть системы управления учебным процессом. А именно, были созданы и протестированы: база данных системы, серверное приложение, включающее в себя сервис ресурсов и сервис авторизации.

На данный момент серверная часть имеет следующие возможности:

* хранение необходимых в учебном процессе данных;
* безопасное предоставление мобильному приложению необходимой в учебном процессе информации;
* регистрация пользователей;
* авторизация и аутентификация пользователей;
* защита паролей пользователей.

Использованные в разработке технологии являются широко используемыми среди программистов, поэтому для увеличения функционала и качества системы будет легко найти новых разработчиков. Реализованная архитектура дает возможность разработки разных типов клиентских приложений, что открывает большие возможности для развития и внедрения системы в учебные заведения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по .NET Core: [Электронный ресурс]. <https://docs.microsoft.com/ru-ru/dotnet/core/> (Дата обращения: 5.05.2020),
2. Документация по PostgreSQL: [Электронный ресурс] <https://postgrespro.ru/docs/postgresql> (Дата обращения 10.05.2020),
3. Обзор способов и протоколов аутентификации: [Электронный ресурс] <https://habr.com/ru/company/dataart/blog/262817/> (Дата обращения 14.05.2020),
4. Авторизация с помощью JWT-токенов: [Электронный ресурс] <https://metanit.com/sharp/aspnet5/23.7.php> (Дата обращения 15.05.2020),
5. Хранение паролей в БД: [Электронный ресурс] <https://habr.com/ru/company/acribia/blog/413157/> (Дата обращения 16.05.2020),
6. ГОСТ 19.701 – 90 (ИСО 5807 – 85) Схемы алгоритмов, программ, данных и систем. – М.: Издательство стандартов, 1991
7. ГОСТ 19.505 – 79 Единая система программной документации. Руководство оператора. Требования к содержанию и оформлению. – М.: Издательство стандартов, 1988
8. Соболева, В.П. Методические указания по оформлению курсовых и выпускных квалификационных работ для студентов, обучающихся на кафедре «Программное обеспечение» /В.П. Соболева. — Ижевск: ИжГТУ, 2018

ПРИЛОЖЕНИЕ 1

КОД ПРОГРАММЫ

П. 1.1. Код создания таблиц в базе данных

DROP TABLE IF EXISTS staff\_auth\_data;

DROP TABLE IF EXISTS students\_auth\_data;

DROP TABLE IF EXISTS schedule;

DROP TABLE IF EXISTS classes;

DROP TABLE IF EXISTS groups\_subjects;

DROP TABLE IF EXISTS sub\_subjects;

DROP TABLE IF EXISTS subjects;

DROP TABLE IF EXISTS subjects\_information;

DROP TABLE IF EXISTS news\_receivers;

DROP TABLE IF EXISTS news;

DROP TABLE IF EXISTS student\_requisites;

DROP TABLE IF EXISTS groups;

DROP TABLE IF EXISTS students;

DROP TABLE IF EXISTS staff\_departments;

DROP TABLE IF EXISTS departments;

DROP TABLE IF EXISTS staff\_requisites;

DROP TABLE IF EXISTS institutes;

DROP TABLE IF EXISTS staff;

CREATE TABLE staff (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL,

surname VARCHAR(50) NOT NULL,

patronymic VARCHAR(50)

);

CREATE TABLE institutes (

id SERIAL PRIMARY KEY,

name VARCHAR(50) UNIQUE NOT NULL,

short\_name VARCHAR(25),

director\_id INTEGER UNIQUE REFERENCES staff (id) UNIQUE NOT NULL,

director\_deputy\_id INTEGER UNIQUE REFERENCES staff (id) UNIQUE NOT NULL

);

CREATE TABLE staff\_requisites (

staff\_id INTEGER REFERENCES staff (id) PRIMARY KEY,

e\_mail VARCHAR(50),

mobile\_phone VARCHAR(50),

home\_phone VARCHAR(50)

);

CREATE TABLE departments (

id SERIAL PRIMARY KEY,

institute\_id INTEGER REFERENCES institutes (id) NOT NULL,

name VARCHAR(50) UNIQUE NOT NULL,

short\_name VARCHAR(25),

director\_id INTEGER UNIQUE REFERENCES staff (id)

);

CREATE TABLE staff\_departments (

department\_id INTEGER REFERENCES departments (id) NOT NULL,

staff\_id INTEGER REFERENCES staff (id) NOT NULL,

CONSTRAINT sd\_pk PRIMARY KEY (department\_id, staff\_id)

);

CREATE TABLE students (

id SERIAL PRIMARY KEY,

student\_number VARCHAR(50) UNIQUE NOT NULL,

name VARCHAR(50) NOT NULL,

surname VARCHAR(50) NOT NULL,

patronymic VARCHAR(50)

);

CREATE TABLE groups (

id SERIAL PRIMARY KEY,

name VARCHAR(25) UNIQUE NOT NULL,

department\_id INTEGER REFERENCES departments (id) NOT NULL,

head\_id INTEGER REFERENCES students (id) NOT NULL

);

CREATE TABLE student\_requisites (

student\_id INTEGER REFERENCES students (id) PRIMARY KEY,

group\_id INTEGER REFERENCES groups (id) NOT NULL,

e\_mail VARCHAR(50),

mobile\_phone\_number VARCHAR(50),

home\_phone\_number VARCHAR(50)

);

CREATE TABLE news (

id SERIAL PRIMARY KEY,

text TEXT NOT NULL,

title VARCHAR(300) NOT NULL,

author\_student\_id INTEGER REFERENCES students (id),

author\_staff\_id INTEGER REFERENCES staff (id),

date TIMESTAMP NOT NULL

);

CREATE TABLE news\_receivers (

id SERIAL PRIMARY KEY,

news\_id INTEGER REFERENCES news (id) NOT NULL,

group\_id INTEGER REFERENCES groups (id),

staff\_id INTEGER REFERENCES staff (id),

department\_id INTEGER REFERENCES departments (id),

institute\_id INTEGER REFERENCES institutes (id)

);

CREATE TABLE subjects\_information (

id SERIAL PRIMARY KEY,

name VARCHAR(50) UNIQUE NOT NULL,

short\_name VARCHAR(25),

description TEXT

);

CREATE TABLE subjects (

id SERIAL PRIMARY KEY,

lecturer\_id INTEGER REFERENCES staff (id) NOT NULL,

information\_id INTEGER REFERENCES subjects\_information (id) NOT NULL

);

CREATE TABLE sub\_subjects (

id SERIAL PRIMARY KEY,

subject\_id INTEGER REFERENCES subjects (id) NOT NULL,

name VARCHAR(100) UNIQUE NOT NULL,

shortname VARCHAR(25),

sub\_lecturer\_id INTEGER REFERENCES staff (id) NOT NULL

);

CREATE TABLE groups\_subjects (

group\_id INTEGER REFERENCES groups (id) NOT NULL,

subject\_id INTEGER REFERENCES subjects (id) NOT NULL,

semester INTEGER CHECK (semester > 0 AND semester < 11) DEFAULT 1 NOT NULL,

CONSTRAINT gs\_pk PRIMARY KEY (group\_id, subject\_id)

);

CREATE TABLE classes (

id SERIAL PRIMARY KEY,

start\_time VARCHAR(25) UNIQUE NOT NULL,

end\_time VARCHAR(25) UNIQUE NOT NULL,

number INTEGER UNIQUE NOT NULL

);

CREATE TABLE schedule (

id SERIAL PRIMARY KEY,

class\_id INTEGER REFERENCES classes (id) NOT NULL,

group\_id INTEGER REFERENCES groups (id),

subject\_id INTEGER REFERENCES subjects (id) NOT NULL,

sub\_subject\_id INTEGER REFERENCES sub\_subjects (id),

date TIMESTAMP NOT NULL,

week\_line BOOLEAN NOT NULL,

location VARCHAR(25) NOT NULL

);

CREATE TABLE students\_auth\_data (

student\_id INTEGER REFERENCES students (id) PRIMARY KEY,

username VARCHAR(25) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

salt VARCHAR(100) NOT NULL,

refresh\_token VARCHAR(100)

);

CREATE TABLE staff\_auth\_data (

staff\_id INTEGER REFERENCES staff (id) PRIMARY KEY,

username VARCHAR(25) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

salt VARCHAR(100) NOT NULL,

refresh\_token VARCHAR(100)

);

П. 1.2. Код серверного приложения

Startup.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.AspNetCore.Builder;  
using Microsoft.AspNetCore.Hosting;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.Extensions.Configuration;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;  
using UniversityData;  
  
[assembly: ApiController]  
  
namespace UniversityWebApi  
{  
 public class Startup  
 {  
 public Startup(IConfiguration configuration)  
 {  
 Configuration = configuration;  
 }  
  
 public IConfiguration Configuration { get; }  
public void ConfigureServices(IServiceCollection services)  
 {  
 services.AddAuthentication(JwtBearerDefaults.**AuthenticationScheme**)  
 .AddJwtBearer(options =>  
 {  
 options.RequireHttpsMetadata = false;  
 options.TokenValidationParameters = new TokenValidationParameters  
 {  
 ValidIssuer = Configuration["Jwt:Issuer"],  
 ValidAudience = Configuration["Jwt:Audience"],  
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))  
 };  
 });  
  
 services.AddControllers();  
  
 services.AddDbContext<UniversityDbContext>(builder =>  
 builder.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));  
 }  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
 {  
 if (env.IsDevelopment())  
 {  
 app.UseDeveloperExceptionPage();  
 }  
  
 app.UseHttpsRedirection();  
  
 app.UseRouting();  
  
 app.UseAuthentication();  
 app.UseAuthorization();  
  
 app.UseEndpoints(endpoints => { endpoints.MapControllers(); });  
 }  
 }  
}

Program.cs

using Microsoft.AspNetCore.Hosting;  
using Microsoft.Extensions.Hosting;  
  
namespace UniversityWebApi  
{  
 public class Program  
 {  
 public static void Main(string[] args)  
 {  
 CreateHostBuilder(args).Build().Run();  
 }  
  
 public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .ConfigureWebHostDefaults(webBuilder => { webBuilder.UseStartup<Startup>(); });  
 }  
}

UniversityDataController.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
using Microsoft.AspNetCore.Authorization;  
using Microsoft.AspNetCore.Mvc;  
using UniversityData;  
using UniversityData.DTO;  
using UniversityData.Entities;  
  
namespace UniversityWebApi.Controllers  
{  
 [Route("api/university")]  
 [Authorize]  
 public class UniversityDataController : ControllerBase  
 {  
 private readonly UniversityDbContext \_context;  
  
 public UniversityDataController(UniversityDbContext context)  
 {  
 \_context = context;  
 }  
  
 [HttpGet]  
 [Route("student-requisites")]  
 public IActionResult GetUserStudentRequisites(int id)  
 {  
 var studentRequisite = \_context.StudentRequisites.FirstOrDefault(sr => sr.StudentId == id);  
 if (studentRequisite == null)  
 {  
 return NotFound();  
 }  
  
 var group = \_context.Groups.First(g => g.Id == studentRequisite.GroupId);  
 var student = \_context.Students.First(s => s.Id == id);  
  
 return Ok(new StudentProfileData  
 {  
 StudentId = studentRequisite.StudentId,  
 FullName = GetFullName(student.Name, student.Surname, student.Patronymic),  
 GroupName = group.Name, Mail = studentRequisite.EMail,  
 MobilePhone = studentRequisite.MobilePhoneNumber, HomePhone = studentRequisite.HomePhoneNumber  
 });  
 }  
  
 [HttpGet]  
 [Route("students")]  
 public ActionResult<IEnumerable<StudentData>> GetStudents(int id)  
 {  
 var studentRequisite = \_context.StudentRequisites.FirstOrDefault(sr => sr.StudentId == id);  
 if (studentRequisite == null)  
 {  
 return NotFound();  
 }  
  
 var studentsRequisites = \_context.StudentRequisites.Where(sr => sr.GroupId == studentRequisite.GroupId);  
 var group = \_context.Groups.First(g => g.Id == studentRequisite.GroupId);  
 var studentsData = \_context.Students  
 .Where(s => studentsRequisites.FirstOrDefault(sr => sr.StudentId == s.Id) != null)  
 .Select(s => new StudentData  
 {  
 FullName = GetFullName(s.Name, s.Surname, s.Patronymic),  
 StudentId = s.Id,  
 IsHead = s.Id == group.HeadId  
 });  
  
 return Ok(studentsData);  
 }  
  
 [HttpGet]  
 [Route("subjects")]  
 public ActionResult<IEnumerable<SubjectData>> GetSubjects(int id)  
 {  
 var studentRequisite = \_context.StudentRequisites.FirstOrDefault(sr => sr.StudentId == id);  
 if (studentRequisite == null)  
 {  
 return NotFound();  
 }  
  
 var subjectsData = new List<SubjectData>();  
 var subjectsIdsSemesters = \_context.GroupsSubjects.Where(gs => gs.GroupId == studentRequisite.GroupId)  
 .Select(gs => new {gs.SubjectId, gs.Semester});  
 foreach (var subjectIdSemester in subjectsIdsSemesters)  
 {  
 var subSubjectsIds = \_context.SubSubjects.Where(ss => ss.SubjectId == subjectIdSemester.SubjectId).Select(ss => ss.Id);  
 var subSubjectsData = new List<SubSubjectData>();  
 foreach (var subSubjectId in subSubjectsIds)  
 {  
 var subSubject = \_context.SubSubjects.First(ss => ss.Id == subSubjectId);  
 subSubjectsData.Add(new SubSubjectData  
 {  
 Id = subSubject.Id,  
 LecturerId = subSubject.SubLecturerId,  
 Name = subSubject.Name,  
 ShortName = subSubject.Shortname  
 });  
 }  
 var subject = \_context.Subjects.First(s => s.Id == subjectIdSemester.SubjectId);  
 var subjectInfo = \_context.SubjectsInformation.First(si => si.Id == subject.InformationId);  
  
 subjectsData.Add(new SubjectData  
 {  
 Id = subjectIdSemester.SubjectId,  
 Name = subjectInfo.Name,  
 ShortName = subjectInfo.ShortName,  
 Description = subjectInfo.Description,  
 LecturerId = subject.LecturerId,  
 Semester = subjectIdSemester.Semester,  
 SubSubjectsData = subSubjectsData  
 });  
 }  
  
 return Ok(subjectsData);  
 }  
  
 private static string GetFullName(string name, string surname, string patronymic)  
 {  
 return (surname + " " + name + " " + patronymic).TrimEnd();  
 }  
  
 #region TestData  
  
 [HttpGet]  
 [Route("generate-test-data")]  
 [AllowAnonymous]  
 public IActionResult GenerateTestData()  
 {  
 try  
 {  
 GenerateStaff();  
 GenerateStaffRequisites();  
 GenerateInstitutes();  
 GenerateDepartments();  
 GenerateStaffDepartments();  
 GenerateStudents();  
 GenerateGroups();  
 GenerateStudentsRequisites();  
 GenerateClasses();  
 GenerateSubjectsInformation();  
 GenerateSubjects();  
 GenerateSubSubjects();  
 GenerateGroupsSubjects();  
 }  
 catch (Exception e)  
 {  
 BadRequest(new ErrorData {ErrorText = e.Message});  
 }  
  
 return Ok();  
 }  
  
private void GenerateStaff()  
 {  
 \_context.Staff.Add(new Staff  
 {  
 Name = "Игорь",  
 Surname = "Архипов",  
 Patronymic = "Олегович"  
 });  
 \_context.Staff.Add(new Staff  
 {  
 Name = "Валентина",  
 Surname = "Соболева",  
 Patronymic = "Павловна"  
 });  
 \_context.Staff.Add(new Staff  
 {  
 Name = "Александр",  
 Surname = "Коробейников",  
 Patronymic = "Васильевич"  
 });  
 \_context.Staff.Add(new Staff  
 {  
 Name = "Владимир",  
 Surname = "Тарасов",  
 Patronymic = "Георгиевич"  
 });  
 \_context.Staff.Add(new Staff  
 {  
 Name = "Артем",  
 Surname = "Юминов"  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateStaffRequisites()  
 {  
 \_context.StaffRequisites.Add(new StaffRequisite  
 {  
 StaffId = \_context.Staff.First(s => s.Surname == "Архипов").Id,  
 EMail = "a@istu.ru"  
 });  
 \_context.StaffRequisites.Add(new StaffRequisite  
 {  
 StaffId = \_context.Staff.First(s => s.Surname == "Тарасов").Id,  
 EMail = "t@istu.ru"  
 });  
 \_context.StaffRequisites.Add(new StaffRequisite  
 {  
 StaffId = \_context.Staff.First(s => s.Surname == "Соболева").Id,  
 EMail = "s@istu.ru"  
 });  
 \_context.StaffRequisites.Add(new StaffRequisite  
 {  
 StaffId = \_context.Staff.First(s => s.Surname == "Коробейников").Id,  
 EMail = "k@istu.ru"  
 });  
 \_context.StaffRequisites.Add(new StaffRequisite  
 {  
 StaffId = \_context.Staff.First(s => s.Surname == "Юминов").Id,  
 EMail = "u@istu.ru"  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateInstitutes()  
 {  
 \_context.Institutes.Add(new Institute  
 {  
 Name = "Институт \"Информатика и вычислительная техника\"",  
 ShortName = "ИВТ",  
 DirectorId = \_context.Staff.First(s => s.Surname == "Архипов").Id,  
 DirectorDeputyId = \_context.Staff.First(s => s.Surname == "Соболева").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateDepartments()  
 {  
 \_context.Departments.Add(new Department  
 {  
 InstituteId = \_context.Institutes.First(i => i.ShortName == "ИВТ").Id,  
 Name = "Кафедра \"Программное обеспечение\"",  
 ShortName = "Кафедра ПО",  
 DirectorId = \_context.Staff.First(s => s.Surname == "Коробейников").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateStaffDepartments()  
 {  
 \_context.StaffDepartments.Add(new StaffDepartment  
 {  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 StaffId = \_context.Staff.First(s => s.Surname == "Соболева").Id  
 });  
 \_context.StaffDepartments.Add(new StaffDepartment  
 {  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 StaffId = \_context.Staff.First(s => s.Surname == "Архипов").Id  
 });  
 \_context.StaffDepartments.Add(new StaffDepartment  
 {  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 StaffId = \_context.Staff.First(s => s.Surname == "Коробейников").Id  
 });  
 \_context.StaffDepartments.Add(new StaffDepartment  
 {  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 StaffId = \_context.Staff.First(s => s.Surname == "Тарасов").Id  
 });  
 \_context.StaffDepartments.Add(new StaffDepartment  
 {  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 StaffId = \_context.Staff.First(s => s.Surname == "Юминов").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateStudents()  
 {  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "12345678",  
 Name = "Никита",  
 Surname = "Вахрушев"  
 });  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "02345678",  
 Name = "Никита",  
 Surname = "Штек"  
 });  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "00345678",  
 Name = "Максим",  
 Surname = "Поздеев",  
 Patronymic = "Львович"  
 });  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "00045678",  
 Name = "Иван",  
 Surname = "Зырянов",  
 Patronymic = "Олегович"  
 });  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "00005678",  
 Name = "Михаил",  
 Surname = "Мокрушин"  
 });  
 \_context.Students.Add(new Student  
 {  
 StudentNumber = "00000678",  
 Name = "Михаил",  
 Surname = "Ивченко"  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateGroups()  
 {  
 \_context.Groups.Add(new Group  
 {  
 Name = "Б08-191-1",  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 HeadId = \_context.Students.First(s => s.Surname == "Штек").Id  
 });  
 \_context.Groups.Add(new Group  
 {  
 Name = "Б08-191-2",  
 DepartmentId = \_context.Departments.First(d => d.ShortName == "Кафедра ПО").Id,  
 HeadId = \_context.Students.First(s => s.Surname == "Вахрушев").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateStudentsRequisites()  
 {  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Штек").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id  
 });  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Ивченко").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id  
 });  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Мокрушин").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id  
 });  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Вахрушев").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id  
 });  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Поздеев").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id  
 });  
 \_context.StudentRequisites.Add(new StudentRequisites  
 {  
 StudentId = \_context.Students.First(s => s.Surname == "Зырянов").Id,  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateClasses()  
 {  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "8:30",  
 EndTime = "10:00",  
 Number = 1  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "10:10",  
 EndTime = "11:40",  
 Number = 2  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "12:20",  
 EndTime = "13:50",  
 Number = 3  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "14:00",  
 EndTime = "15:30",  
 Number = 4  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "15:40",  
 EndTime = "17:10",  
 Number = 5  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "17:20",  
 EndTime = "18:50",  
 Number = 6  
 });  
 \_context.Classes.Add(new Class  
 {  
 StartTime = "19:00",  
 EndTime = "20:30",  
 Number = 7  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateSubjectsInformation()  
 {  
 \_context.SubjectsInformation.Add(new SubjectInformation  
 {  
 Name = "Операционные системы",  
 ShortName = "ОС",  
 Description = "Учи ОС!"  
 });  
 \_context.SubjectsInformation.Add(new SubjectInformation  
 {  
 Name = "Математические основы искусственного интеллекта",  
 ShortName = "МОИИ",  
 Description = "Учи МОИИ!"  
 });  
 \_context.SubjectsInformation.Add(new SubjectInformation  
 {  
 Name = "Теория цифровой обработки сигналов",  
 ShortName = "ТЦОС",  
 Description = "Учи ТЦОС!"  
 });  
 \_context.SubjectsInformation.Add(new SubjectInformation  
 {  
 Name = "Информатика",  
 Description = "Учи Информатику!"  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateSubjects()  
 {  
 \_context.Subjects.Add(new Subject  
 {  
 LecturerId = \_context.Staff.First(s => s.Surname == "Архипов").Id,  
 InformationId = \_context.SubjectsInformation.First(si => si.ShortName == "ТЦОС").Id  
 });  
 \_context.Subjects.Add(new Subject  
 {  
 LecturerId = \_context.Staff.First(s => s.Surname == "Тарасов").Id,  
 InformationId = \_context.SubjectsInformation.First(si => si.ShortName == "ОС").Id  
 });  
 \_context.Subjects.Add(new Subject  
 {  
 LecturerId = \_context.Staff.First(s => s.Surname == "Коробейников").Id,  
 InformationId = \_context.SubjectsInformation.First(si => si.ShortName == "МОИИ").Id  
 });  
 \_context.Subjects.Add(new Subject  
 {  
 LecturerId = \_context.Staff.First(s => s.Surname == "Соболева").Id,  
 InformationId = \_context.SubjectsInformation.First(si => si.Name == "Информатика").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateSubSubjects()  
 {  
 \_context.SubSubjects.Add(new SubSubject  
 {  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ТЦОС").Id).Id,  
 Name = "Теория цифровой обработки сигналов. Практика",  
 Shortname = "ТЦОС (Практика)",  
 SubLecturerId = \_context.Staff.First(s => s.Surname == "Архипов").Id  
 });  
 \_context.SubSubjects.Add(new SubSubject  
 {  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ТЦОС").Id).Id,  
 Name = "Теория цифровой обработки сигналов. Лабораторная",  
 Shortname = "ТЦОС (Лаб)",  
 SubLecturerId = \_context.Staff.First(s => s.Surname == "Архипов").Id  
 });  
 \_context.SubSubjects.Add(new SubSubject  
 {  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ОС").Id).Id,  
 Name = "Операционные системы. Лабораторная",  
 Shortname = "ОС (Лаб)",  
 SubLecturerId = \_context.Staff.First(s => s.Surname == "Юминов").Id  
 });  
 \_context.SubSubjects.Add(new SubSubject  
 {  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.Name == "Информатика").Id).Id,  
 Name = "Информатика. Лабораторная",  
 Shortname = "Информатика (Лаб)",  
 SubLecturerId = \_context.Staff.First(s => s.Surname == "Соболева").Id  
 });  
 \_context.SaveChanges();  
 }  
  
 private void GenerateGroupsSubjects()  
 {  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ОС").Id).Id,  
 Semester = 6  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "МОИИ").Id).Id,  
 Semester = 6  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ТЦОС").Id).Id,  
 Semester = 5  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-1").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.Name == "Информатика").Id).Id,  
 Semester = 1  
 });  
  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ОС").Id).Id,  
 Semester = 6  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "МОИИ").Id).Id,  
 Semester = 6  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.ShortName == "ТЦОС").Id).Id,  
 Semester = 5  
 });  
 \_context.GroupsSubjects.Add(new GroupSubject  
 {  
 GroupId = \_context.Groups.First(g => g.Name == "Б08-191-2").Id,  
 SubjectId = \_context.Subjects.First(s =>  
 s.InformationId == \_context.SubjectsInformation.First(si => si.Name == "Информатика").Id).Id,  
 Semester = 1  
 });  
 \_context.SaveChanges();  
 }  
#endregion  
 }  
}

RegistrationController.cs

using System;  
using System.Linq;  
using System.Security.Cryptography;  
using Microsoft.AspNetCore.Cryptography.KeyDerivation;  
using Microsoft.AspNetCore.Mvc;  
using UniversityData;  
using UniversityData.Entities;  
  
namespace UniversityWebApi.Controllers  
{  
 [Route("api/[controller]")]  
 public class RegistrationController : ControllerBase  
 {  
 private readonly UniversityDbContext \_context;  
  
 public RegistrationController(UniversityDbContext context)  
 {  
 \_context = context;  
 }  
 [HttpPost]  
 [Route("register-student")]  
 public IActionResult RegisterStudent(string studentNumber, string username, string password)  
 {  
 var student = \_context.Students.FirstOrDefault(s => s.StudentNumber == studentNumber);  
 if (student == null) return BadRequest();  
 var salt = GenerateSalt();  
 var hashedPassword = GetHashedPassword(salt, password);  
 \_context.StudentsAuthData.Add(new StudentAuthData()  
 {  
 StudentId = student.Id,  
 Username = username,  
 Password = hashedPassword,  
 Salt = Convert.ToBase64String(salt),  
 Student = student  
 });  
 \_context.SaveChanges();  
   
 return Ok();  
 }  
  
 private byte[] GenerateSalt()  
 {  
 var salt = new byte[128 / 8];  
 using (var rng = RandomNumberGenerator.Create())  
 {  
 rng.GetBytes(salt);  
 }  
  
 return salt;  
 }  
 private string GetHashedPassword(byte[] salt, string password) =>  
 Convert.ToBase64String(KeyDerivation.Pbkdf2(  
 password: password,  
 salt: salt,  
 prf: KeyDerivationPrf.**HMACSHA1**,  
 iterationCount: 10000,  
 numBytesRequested: 256 / 8));  
 }  
}

AuthController.cs

using System;  
using System.Collections.Generic;  
using System.IdentityModel.Tokens.Jwt;  
using System.Linq;  
using System.Security.Claims;  
using System.Security.Cryptography;  
using System.Text;  
using Microsoft.AspNetCore.Cryptography.KeyDerivation;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Configuration;  
using Microsoft.IdentityModel.Tokens;  
using UniversityData;  
using UniversityData.DTO;  
using UniversityData.Entities;  
  
namespace UniversityWebApi.Controllers  
{  
 [Route("api/[controller]")]  
 public class AuthController : ControllerBase  
 {  
 private readonly UniversityDbContext \_context;  
 private readonly IConfiguration \_config;  
  
 public AuthController(UniversityDbContext context, IConfiguration config)  
 {  
 \_context = context;  
 \_config = config;  
 }  
  
 [HttpPost]  
 [Route("authorize-student")]  
 public IActionResult AuthorizeStudent(string username, string password)  
 {  
 var user = \_context.StudentsAuthData.FirstOrDefault(s => s.Username == username);  
 if (user == null || !IsCorrectPassword(user, password))  
 return BadRequest(new ErrorData {ErrorText = "Invalid username or password."});  
 var encodedJwt = GenerateAccessToken(user);  
  
 user.RefreshToken = GenerateRefreshToken();  
 \_context.SaveChanges();  
  
 return Ok(new AuthData  
 {AccessToken = encodedJwt, UserId = user.StudentId, RefreshToken = user.RefreshToken});  
 }  
  
 [HttpPost]  
 [Route("refresh-access-token")]  
 public IActionResult RefreshAccessToken(string accessToken, string refreshToken)  
 {  
 var tokenValidationParameters = new TokenValidationParameters  
 {  
 ValidateLifetime = false,  
 ValidIssuer = \_config["Jwt:Issuer"],  
 ValidAudience = \_config["Jwt:Audience"],  
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]))  
 };  
  
 var tokenHandler = new JwtSecurityTokenHandler();  
 var principal = tokenHandler.ValidateToken(accessToken, tokenValidationParameters, out var securityToken);  
 if (!(securityToken is JwtSecurityToken jwtSecurityToken) ||  
 !jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.**HmacSha256**, StringComparison.**InvariantCulture**))  
 {  
 return BadRequest(new ErrorData {ErrorText = "Invalid access token"});  
 }  
  
 var username = principal.FindFirst(ClaimTypes.**Name**)?.Value;  
 var user = \_context.StudentsAuthData.FirstOrDefault(u => u.Username == username);  
  
 if (user == null)  
 {  
 return BadRequest(new ErrorData {ErrorText = "Invalid access token"});  
 }  
  
 if (!user.RefreshToken.Equals(refreshToken, StringComparison.**InvariantCulture**))  
 {  
 return BadRequest(new ErrorData {ErrorText = "Invalid refresh token"});  
 }  
  
 var encodedJwt = GenerateAccessToken(user);  
  
 user.RefreshToken = GenerateRefreshToken();  
 \_context.SaveChanges();  
  
 return Ok(new AuthData  
 {AccessToken = encodedJwt, UserId = user.StudentId, RefreshToken = user.RefreshToken});  
 }  
  
 private static bool IsCorrectPassword(StudentAuthData user, string password)  
 {  
 var salt = Convert.FromBase64String(user.Salt);  
 var hashedPassword = GetHashedPassword(salt, password);  
 return hashedPassword == user.Password;  
 }  
  
 private static string GetHashedPassword(byte[] salt, string password) =>  
 Convert.ToBase64String(KeyDerivation.Pbkdf2(  
 password: password,  
 salt: salt,  
 prf: KeyDerivationPrf.**HMACSHA1**,  
 iterationCount: 10000,  
 numBytesRequested: 256 / 8));  
  
 private string GenerateAccessToken(StudentAuthData user)  
 {  
 var claims = new List<Claim>  
 {  
 new Claim(ClaimsIdentity.**DefaultNameClaimType**, user.Username)  
 };  
 var jwt = new JwtSecurityToken(  
 issuer: \_config["Jwt:Issuer"],  
 audience: \_config["Jwt:Audience"],  
 claims: claims,  
 expires: DateTime.Now.AddMinutes(60),  
 signingCredentials: new SigningCredentials(  
 new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"])),  
 SecurityAlgorithms.**HmacSha256**));  
 var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);  
  
 return encodedJwt;  
 }  
  
 private static string GenerateRefreshToken()  
 {  
 var randomNumber = new byte[32];  
 using var rng = RandomNumberGenerator.Create();  
 rng.GetBytes(randomNumber);  
  
 return Convert.ToBase64String(randomNumber);  
 }  
 }  
}

UniversityDbContext.cs

using System;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.EntityFrameworkCore.Metadata;  
using UniversityData.Entities;  
  
namespace UniversityData  
{  
 public partial class UniversityDbContext : DbContext  
 {  
 public UniversityDbContext()  
 {  
 }  
  
 public UniversityDbContext(DbContextOptions<UniversityDbContext> options)  
 : base(options)  
 {  
 }  
  
 public virtual DbSet<Class> Classes { get; set; }  
 public virtual DbSet<Department> Departments { get; set; }  
 public virtual DbSet<Group> Groups { get; set; }  
 public virtual DbSet<GroupSubject> GroupsSubjects { get; set; }  
 public virtual DbSet<Institute> Institutes { get; set; }  
 public virtual DbSet<News> News { get; set; }  
 public virtual DbSet<NewsReceiver> NewsReceivers { get; set; }  
 public virtual DbSet<Schedule> Schedule { get; set; }  
 public virtual DbSet<Staff> Staff { get; set; }  
 public virtual DbSet<StaffAuthData> StaffAuthData { get; set; }  
 public virtual DbSet<StaffDepartment> StaffDepartments { get; set; }  
 public virtual DbSet<StaffRequisite> StaffRequisites { get; set; }  
 public virtual DbSet<StudentRequisites> StudentRequisites { get; set; }  
 public virtual DbSet<Student> Students { get; set; }  
 public virtual DbSet<StudentAuthData> StudentsAuthData { get; set; }  
 public virtual DbSet<SubSubject> SubSubjects { get; set; }  
 public virtual DbSet<Subject> Subjects { get; set; }  
 public virtual DbSet<SubjectInformation> SubjectsInformation { get; set; }  
protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 modelBuilder.Entity<Class>(entity =>  
 {  
 entity.ToTable("classes");  
  
 entity.HasIndex(e => e.EndTime)  
 .HasName("classes\_end\_time\_key")  
 .IsUnique();  
  
 entity.HasIndex(e => e.Number)  
 .HasName("classes\_number\_key")  
 .IsUnique();  
  
 entity.HasIndex(e => e.StartTime)  
 .HasName("classes\_start\_time\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.EndTime)  
 .IsRequired()  
 .HasColumnName("end\_time")  
 .HasMaxLength(25);  
  
 entity.Property(e => e.Number).HasColumnName("number");  
  
 entity.Property(e => e.StartTime)  
 .IsRequired()  
 .HasColumnName("start\_time")  
 .HasMaxLength(25);  
 });  
  
 modelBuilder.Entity<Department>(entity =>  
 {  
 entity.ToTable("departments");  
  
 entity.HasIndex(e => e.DirectorId)  
 .HasName("departments\_director\_id\_key")  
 .IsUnique();  
  
 entity.HasIndex(e => e.Name)  
 .HasName("departments\_name\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.DirectorId).HasColumnName("director\_id");  
  
 entity.Property(e => e.InstituteId).HasColumnName("institute\_id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.ShortName)  
 .HasColumnName("short\_name")  
 .HasMaxLength(25);  
  
 entity.HasOne(d => d.Director)  
 .WithOne(p => p.Departments)  
 .HasForeignKey<Department>(d => d.DirectorId)  
 .HasConstraintName("departments\_director\_id\_fkey");  
  
 entity.HasOne(d => d.Institute)  
 .WithMany(p => p.Departments)  
 .HasForeignKey(d => d.InstituteId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("departments\_institute\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Group>(entity =>  
 {  
 entity.ToTable("groups");  
  
 entity.HasIndex(e => e.Name)  
 .HasName("groups\_name\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.DepartmentId).HasColumnName("department\_id");  
  
 entity.Property(e => e.HeadId).HasColumnName("head\_id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(25);  
  
 entity.HasOne(d => d.Department)  
 .WithMany(p => p.Groups)  
 .HasForeignKey(d => d.DepartmentId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("groups\_department\_id\_fkey");  
  
 entity.HasOne(d => d.Head)  
 .WithMany(p => p.Groups)  
 .HasForeignKey(d => d.HeadId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("groups\_head\_id\_fkey");  
 });  
  
 modelBuilder.Entity<GroupSubject>(entity =>  
 {  
 entity.HasKey(e => new { e.GroupId, e.SubjectId })  
 .HasName("gs\_pk");  
  
 entity.ToTable("groups\_subjects");  
  
 entity.Property(e => e.GroupId).HasColumnName("group\_id");  
  
 entity.Property(e => e.SubjectId).HasColumnName("subject\_id");  
  
 entity.Property(e => e.Semester)  
 .HasColumnName("semester")  
 .HasDefaultValueSql("1");  
  
 entity.HasOne(d => d.Group)  
 .WithMany(p => p.GroupsSubjects)  
 .HasForeignKey(d => d.GroupId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("groups\_subjects\_group\_id\_fkey");  
  
 entity.HasOne(d => d.Subject)  
 .WithMany(p => p.GroupsSubjects)  
 .HasForeignKey(d => d.SubjectId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("groups\_subjects\_subject\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Institute>(entity =>  
 {  
 entity.ToTable("institutes");  
  
 entity.HasIndex(e => e.DirectorDeputyId)  
 .HasName("institutes\_director\_deputy\_id\_key")  
 .IsUnique();  
  
 entity.HasIndex(e => e.DirectorId)  
 .HasName("institutes\_director\_id\_key")  
 .IsUnique();  
  
 entity.HasIndex(e => e.Name)  
 .HasName("institutes\_name\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.DirectorDeputyId).HasColumnName("director\_deputy\_id");  
  
 entity.Property(e => e.DirectorId).HasColumnName("director\_id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.ShortName)  
 .HasColumnName("short\_name")  
 .HasMaxLength(25);  
  
 entity.HasOne(d => d.DirectorDeputy)  
 .WithOne(p => p.InstitutesDirectorDeputy)  
 .HasForeignKey<Institute>(d => d.DirectorDeputyId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("institutes\_director\_deputy\_id\_fkey");  
  
 entity.HasOne(d => d.Director)  
 .WithOne(p => p.InstitutesDirector)  
 .HasForeignKey<Institute>(d => d.DirectorId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("institutes\_director\_id\_fkey");  
 });  
  
 modelBuilder.Entity<News>(entity =>  
 {  
 entity.ToTable("news");  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.AuthorStaffId).HasColumnName("author\_staff\_id");  
  
 entity.Property(e => e.AuthorStudentId).HasColumnName("author\_student\_id");  
  
 entity.Property(e => e.Date).HasColumnName("date");  
  
 entity.Property(e => e.Text)  
 .IsRequired()  
 .HasColumnName("text");  
  
 entity.Property(e => e.Title)  
 .IsRequired()  
 .HasColumnName("title")  
 .HasMaxLength(300);  
  
 entity.HasOne(d => d.AuthorStaff)  
 .WithMany(p => p.News)  
 .HasForeignKey(d => d.AuthorStaffId)  
 .HasConstraintName("news\_author\_staff\_id\_fkey");  
  
 entity.HasOne(d => d.AuthorStudent)  
 .WithMany(p => p.News)  
 .HasForeignKey(d => d.AuthorStudentId)  
 .HasConstraintName("news\_author\_student\_id\_fkey");  
 });  
  
 modelBuilder.Entity<NewsReceiver>(entity =>  
 {  
 entity.ToTable("news\_receivers");  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.DepartmentId).HasColumnName("department\_id");  
  
 entity.Property(e => e.GroupId).HasColumnName("group\_id");  
  
 entity.Property(e => e.InstituteId).HasColumnName("institute\_id");  
  
 entity.Property(e => e.NewsId).HasColumnName("news\_id");  
  
 entity.Property(e => e.StaffId).HasColumnName("staff\_id");  
  
 entity.HasOne(d => d.Department)  
 .WithMany(p => p.NewsReceivers)  
 .HasForeignKey(d => d.DepartmentId)  
 .HasConstraintName("news\_receivers\_department\_id\_fkey");  
  
 entity.HasOne(d => d.Group)  
 .WithMany(p => p.NewsReceivers)  
 .HasForeignKey(d => d.GroupId)  
 .HasConstraintName("news\_receivers\_group\_id\_fkey");  
  
 entity.HasOne(d => d.Institute)  
 .WithMany(p => p.NewsReceivers)  
 .HasForeignKey(d => d.InstituteId)  
 .HasConstraintName("news\_receivers\_institute\_id\_fkey");  
  
 entity.HasOne(d => d.News)  
 .WithMany(p => p.NewsReceivers)  
 .HasForeignKey(d => d.NewsId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("news\_receivers\_news\_id\_fkey");  
  
 entity.HasOne(d => d.Staff)  
 .WithMany(p => p.NewsReceivers)  
 .HasForeignKey(d => d.StaffId)  
 .HasConstraintName("news\_receivers\_staff\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Schedule>(entity =>  
 {  
 entity.ToTable("schedule");  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.ClassId).HasColumnName("class\_id");  
  
 entity.Property(e => e.Date).HasColumnName("date");  
  
 entity.Property(e => e.GroupId).HasColumnName("group\_id");  
  
 entity.Property(e => e.Location)  
 .IsRequired()  
 .HasColumnName("location")  
 .HasMaxLength(25);  
  
 entity.Property(e => e.SubSubjectId).HasColumnName("sub\_subject\_id");  
  
 entity.Property(e => e.SubjectId).HasColumnName("subject\_id");  
  
 entity.Property(e => e.WeekLine).HasColumnName("week\_line");  
  
 entity.HasOne(d => d.Class)  
 .WithMany(p => p.Schedule)  
 .HasForeignKey(d => d.ClassId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("schedule\_class\_id\_fkey");  
  
 entity.HasOne(d => d.Group)  
 .WithMany(p => p.Schedule)  
 .HasForeignKey(d => d.GroupId)  
 .HasConstraintName("schedule\_group\_id\_fkey");  
  
 entity.HasOne(d => d.SubSubject)  
 .WithMany(p => p.Schedule)  
 .HasForeignKey(d => d.SubSubjectId)  
 .HasConstraintName("schedule\_sub\_subject\_id\_fkey");  
  
 entity.HasOne(d => d.Subject)  
 .WithMany(p => p.Schedule)  
 .HasForeignKey(d => d.SubjectId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("schedule\_subject\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Staff>(entity =>  
 {  
 entity.ToTable("staff");  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.Patronymic)  
 .HasColumnName("patronymic")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.Surname)  
 .IsRequired()  
 .HasColumnName("surname")  
 .HasMaxLength(50);  
 });  
  
 modelBuilder.Entity<StaffAuthData>(entity =>  
 {  
 entity.HasKey(e => e.StaffId)  
 .HasName("staff\_auth\_data\_pkey");  
  
 entity.ToTable("staff\_auth\_data");  
  
 entity.HasIndex(e => e.Username)  
 .HasName("staff\_auth\_data\_username\_key")  
 .IsUnique();  
  
 entity.Property(e => e.StaffId)  
 .HasColumnName("staff\_id")  
 .ValueGeneratedNever();  
  
 entity.Property(e => e.Password)  
 .IsRequired()  
 .HasColumnName("password")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.RefreshToken)  
 .HasColumnName("refresh\_token")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.Salt)  
 .IsRequired()  
 .HasColumnName("salt")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.Username)  
 .IsRequired()  
 .HasColumnName("username")  
 .HasMaxLength(25);  
  
 entity.HasOne(d => d.Staff)  
 .WithOne(p => p.StaffAuthData)  
 .HasForeignKey<StaffAuthData>(d => d.StaffId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("staff\_auth\_data\_staff\_id\_fkey");  
 });  
  
 modelBuilder.Entity<StaffDepartment>(entity =>  
 {  
 entity.HasKey(e => new { e.DepartmentId, e.StaffId })  
 .HasName("sd\_pk");  
  
 entity.ToTable("staff\_departments");  
  
 entity.Property(e => e.DepartmentId).HasColumnName("department\_id");  
  
 entity.Property(e => e.StaffId).HasColumnName("staff\_id");  
  
 entity.HasOne(d => d.Department)  
 .WithMany(p => p.StaffDepartments)  
 .HasForeignKey(d => d.DepartmentId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("staff\_departments\_department\_id\_fkey");  
  
 entity.HasOne(d => d.Staff)  
 .WithMany(p => p.StaffDepartments)  
 .HasForeignKey(d => d.StaffId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("staff\_departments\_staff\_id\_fkey");  
 });  
  
 modelBuilder.Entity<StaffRequisite>(entity =>  
 {  
 entity.HasKey(e => e.StaffId)  
 .HasName("staff\_requisites\_pkey");  
  
 entity.ToTable("staff\_requisites");  
  
 entity.Property(e => e.StaffId)  
 .HasColumnName("staff\_id")  
 .ValueGeneratedNever();  
  
 entity.Property(e => e.EMail)  
 .HasColumnName("e\_mail")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.HomePhone)  
 .HasColumnName("home\_phone")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.MobilePhone)  
 .HasColumnName("mobile\_phone")  
 .HasMaxLength(50);  
  
 entity.HasOne(d => d.Staff)  
 .WithOne(p => p.StaffRequisites)  
 .HasForeignKey<StaffRequisite>(d => d.StaffId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("staff\_requisites\_staff\_id\_fkey");  
 });  
  
 modelBuilder.Entity<StudentRequisites>(entity =>  
 {  
 entity.HasKey(e => e.StudentId)  
 .HasName("student\_requisites\_pkey");  
  
 entity.ToTable("student\_requisites");  
  
 entity.Property(e => e.StudentId)  
 .HasColumnName("student\_id")  
 .ValueGeneratedNever();  
  
 entity.Property(e => e.EMail)  
 .HasColumnName("e\_mail")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.GroupId).HasColumnName("group\_id");  
  
 entity.Property(e => e.HomePhoneNumber)  
 .HasColumnName("home\_phone\_number")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.MobilePhoneNumber)  
 .HasColumnName("mobile\_phone\_number")  
 .HasMaxLength(50);  
  
 entity.HasOne(d => d.Group)  
 .WithMany(p => p.StudentRequisites)  
 .HasForeignKey(d => d.GroupId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("student\_requisites\_group\_id\_fkey");  
  
 entity.HasOne(d => d.Student)  
 .WithOne(p => p.StudentRequisites)  
 .HasForeignKey<StudentRequisites>(d => d.StudentId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("student\_requisites\_student\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Student>(entity =>  
 {  
 entity.ToTable("students");  
  
 entity.HasIndex(e => e.StudentNumber)  
 .HasName("students\_student\_number\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.Patronymic)  
 .HasColumnName("patronymic")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.StudentNumber)  
 .IsRequired()  
 .HasColumnName("student\_number")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.Surname)  
 .IsRequired()  
 .HasColumnName("surname")  
 .HasMaxLength(50);  
 });  
  
 modelBuilder.Entity<StudentAuthData>(entity =>  
 {  
 entity.HasKey(e => e.StudentId)  
 .HasName("students\_auth\_data\_pkey");  
  
 entity.ToTable("students\_auth\_data");  
  
 entity.HasIndex(e => e.Username)  
 .HasName("students\_auth\_data\_username\_key")  
 .IsUnique();  
  
 entity.Property(e => e.StudentId)  
 .HasColumnName("student\_id")  
 .ValueGeneratedNever();  
  
 entity.Property(e => e.Password)  
 .IsRequired()  
 .HasColumnName("password")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.RefreshToken)  
 .HasColumnName("refresh\_token")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.Salt)  
 .IsRequired()  
 .HasColumnName("salt")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.Username)  
 .IsRequired()  
 .HasColumnName("username")  
 .HasMaxLength(25);  
  
 entity.HasOne(d => d.Student)  
 .WithOne(p => p.StudentsAuthData)  
 .HasForeignKey<StudentAuthData>(d => d.StudentId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("students\_auth\_data\_student\_id\_fkey");  
 });  
  
 modelBuilder.Entity<SubSubject>(entity =>  
 {  
 entity.ToTable("sub\_subjects");  
  
 entity.HasIndex(e => e.Name)  
 .HasName("sub\_subjects\_name\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(100);  
  
 entity.Property(e => e.Shortname)  
 .HasColumnName("shortname")  
 .HasMaxLength(25);  
  
 entity.Property(e => e.SubLecturerId).HasColumnName("sub\_lecturer\_id");  
  
 entity.Property(e => e.SubjectId).HasColumnName("subject\_id");  
  
 entity.HasOne(d => d.SubLecturer)  
 .WithMany(p => p.SubSubjects)  
 .HasForeignKey(d => d.SubLecturerId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("sub\_subjects\_sub\_lecturer\_id\_fkey");  
  
 entity.HasOne(d => d.Subject)  
 .WithMany(p => p.SubSubjects)  
 .HasForeignKey(d => d.SubjectId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("sub\_subjects\_subject\_id\_fkey");  
 });  
  
 modelBuilder.Entity<Subject>(entity =>  
 {  
 entity.ToTable("subjects");  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.InformationId).HasColumnName("information\_id");  
  
 entity.Property(e => e.LecturerId).HasColumnName("lecturer\_id");  
  
 entity.HasOne(d => d.Information)  
 .WithMany(p => p.Subjects)  
 .HasForeignKey(d => d.InformationId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("subjects\_information\_id\_fkey");  
  
 entity.HasOne(d => d.Lecturer)  
 .WithMany(p => p.Subjects)  
 .HasForeignKey(d => d.LecturerId)  
 .OnDelete(DeleteBehavior.**ClientSetNull**)  
 .HasConstraintName("subjects\_lecturer\_id\_fkey");  
 });  
  
 modelBuilder.Entity<SubjectInformation>(entity =>  
 {  
 entity.ToTable("subjects\_information");  
  
 entity.HasIndex(e => e.Name)  
 .HasName("subjects\_information\_name\_key")  
 .IsUnique();  
  
 entity.Property(e => e.Id).HasColumnName("id");  
  
 entity.Property(e => e.Description).HasColumnName("description");  
  
 entity.Property(e => e.Name)  
 .IsRequired()  
 .HasColumnName("name")  
 .HasMaxLength(50);  
  
 entity.Property(e => e.ShortName)  
 .HasColumnName("short\_name")  
 .HasMaxLength(25);  
 });  
  
 OnModelCreatingPartial(modelBuilder);  
 }  
  
 partial void OnModelCreatingPartial(ModelBuilder modelBuilder);  
 }  
}

Class.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Class  
 {  
 public Class()  
 {  
 Schedule = new HashSet<Schedule>();  
 }  
  
 public int Id { get; set; }  
 public string StartTime { get; set; }  
 public string EndTime { get; set; }  
 public int Number { get; set; }  
  
 public virtual ICollection<Schedule> Schedule { get; set; }  
 }  
}

Department.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Department  
 {  
 public Department()  
 {  
 Groups = new HashSet<Group>();  
 NewsReceivers = new HashSet<NewsReceiver>();  
 StaffDepartments = new HashSet<StaffDepartment>();  
 }  
  
 public int Id { get; set; }  
 public int InstituteId { get; set; }  
 public string Name { get; set; }  
 public string ShortName { get; set; }  
 public int? DirectorId { get; set; }  
  
 public virtual Staff Director { get; set; }  
 public virtual Institute Institute { get; set; }  
 public virtual ICollection<Group> Groups { get; set; }  
 public virtual ICollection<NewsReceiver> NewsReceivers { get; set; }  
 public virtual ICollection<StaffDepartment> StaffDepartments { get; set; }  
 }  
}

Group.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Department  
 {  
 public Department()  
 {  
 Groups = new HashSet<Group>();  
 NewsReceivers = new HashSet<NewsReceiver>();  
 StaffDepartments = new HashSet<StaffDepartment>();  
 }  
  
 public int Id { get; set; }  
 public int InstituteId { get; set; }  
 public string Name { get; set; }  
 public string ShortName { get; set; }  
 public int? DirectorId { get; set; }  
  
 public virtual Staff Director { get; set; }  
 public virtual Institute Institute { get; set; }  
 public virtual ICollection<Group> Groups { get; set; }  
 public virtual ICollection<NewsReceiver> NewsReceivers { get; set; }  
 public virtual ICollection<StaffDepartment> StaffDepartments { get; set; }  
 }  
}

GroupSubject.cs

namespace UniversityData.Entities  
{  
 public partial class GroupSubject  
 {  
 public int GroupId { get; set; }  
 public int SubjectId { get; set; }  
 public int Semester { get; set; }  
  
 public virtual Group Group { get; set; }  
 public virtual Subject Subject { get; set; }  
 }  
}

Institute.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Institute  
 {  
 public Institute()  
 {  
 Departments = new HashSet<Department>();  
 NewsReceivers = new HashSet<NewsReceiver>();  
 }  
  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public string ShortName { get; set; }  
 public int DirectorId { get; set; }  
 public int DirectorDeputyId { get; set; }  
  
 public virtual Staff Director { get; set; }  
 public virtual Staff DirectorDeputy { get; set; }  
 public virtual ICollection<Department> Departments { get; set; }  
 public virtual ICollection<NewsReceiver> NewsReceivers { get; set; }  
 }  
}

News.cs

using System;  
using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class News  
 {  
 public News()  
 {  
 NewsReceivers = new HashSet<NewsReceiver>();  
 }  
  
 public int Id { get; set; }  
 public string Text { get; set; }  
 public string Title { get; set; }  
 public int? AuthorStudentId { get; set; }  
 public int? AuthorStaffId { get; set; }  
 public DateTime Date { get; set; }  
  
 public virtual Staff AuthorStaff { get; set; }  
 public virtual Student AuthorStudent { get; set; }  
 public virtual ICollection<NewsReceiver> NewsReceivers { get; set; }  
 }  
}

Schedule.cs

using System;  
  
namespace UniversityData.Entities  
{  
 public partial class Schedule  
 {  
 public int Id { get; set; }  
 public int ClassId { get; set; }  
 public int? GroupId { get; set; }  
 public int SubjectId { get; set; }  
 public int? SubSubjectId { get; set; }  
 public DateTime Date { get; set; }  
 public bool WeekLine { get; set; }  
 public string Location { get; set; }  
  
 public virtual Class Class { get; set; }  
 public virtual Group Group { get; set; }  
 public virtual SubSubject SubSubject { get; set; }  
 public virtual Subject Subject { get; set; }  
 }  
}

Staff.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Staff  
 {  
 public Staff()  
 {  
 News = new HashSet<News>();  
 NewsReceivers = new HashSet<NewsReceiver>();  
 StaffDepartments = new HashSet<StaffDepartment>();  
 SubSubjects = new HashSet<SubSubject>();  
 Subjects = new HashSet<Subject>();  
 }  
  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public string Surname { get; set; }  
 public string Patronymic { get; set; }  
  
 public virtual Department Departments { get; set; }  
 public virtual Institute InstitutesDirector { get; set; }  
 public virtual Institute InstitutesDirectorDeputy { get; set; }  
 public virtual StaffAuthData StaffAuthData { get; set; }  
 public virtual StaffRequisite StaffRequisites { get; set; }  
 public virtual ICollection<News> News { get; set; }  
 public virtual ICollection<NewsReceiver> NewsReceivers { get; set; }  
 public virtual ICollection<StaffDepartment> StaffDepartments { get; set; }  
 public virtual ICollection<SubSubject> SubSubjects { get; set; }  
 public virtual ICollection<Subject> Subjects { get; set; }  
 }  
}

StaffAuthData.cs

namespace UniversityData.Entities  
{  
 public partial class StaffAuthData  
 {  
 public int StaffId { get; set; }  
 public string Username { get; set; }  
 public string Password { get; set; }  
 public string Salt { get; set; }  
 public string RefreshToken { get; set; }  
  
 public virtual Staff Staff { get; set; }  
 }  
}

StaffDepartment.cs

namespace UniversityData.Entities  
{  
 public partial class StaffDepartment  
 {  
 public int DepartmentId { get; set; }  
 public int StaffId { get; set; }  
  
 public virtual Department Department { get; set; }  
 public virtual Staff Staff { get; set; }  
 }  
}

StaffRequsite.cs

namespace UniversityData.Entities  
{  
 public partial class StaffRequisite  
 {  
 public int StaffId { get; set; }  
 public string EMail { get; set; }  
 public string MobilePhone { get; set; }  
 public string HomePhone { get; set; }  
  
 public virtual Staff Staff { get; set; }  
 }  
}

Student.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Student  
 {  
 public Student()  
 {  
 Groups = new HashSet<Group>();  
 News = new HashSet<News>();  
 }  
  
 public int Id { get; set; }  
 public string StudentNumber { get; set; }  
 public string Name { get; set; }  
 public string Surname { get; set; }  
 public string Patronymic { get; set; }  
  
 public virtual StudentRequisites StudentRequisites { get; set; }  
 public virtual StudentAuthData StudentsAuthData { get; set; }  
 public virtual ICollection<Group> Groups { get; set; }  
 public virtual ICollection<News> News { get; set; }  
 }  
}

StudentAuthData.cs

namespace UniversityData.Entities  
{  
 public partial class StudentAuthData  
 {  
 public int StudentId { get; set; }  
 public string Username { get; set; }  
 public string Password { get; set; }  
 public string Salt { get; set; }  
 public string RefreshToken { get; set; }  
  
 public virtual Student Student { get; set; }  
 }  
}

StudentRequisite.cs

namespace UniversityData.Entities  
{  
 public partial class StudentRequisites  
 {  
 public int StudentId { get; set; }  
 public int GroupId { get; set; }  
 public string EMail { get; set; }  
 public string MobilePhoneNumber { get; set; }  
 public string HomePhoneNumber { get; set; }  
  
 public virtual Group Group { get; set; }  
 public virtual Student Student { get; set; }  
 }  
}

Subjects.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class Subject  
 {  
 public Subject()  
 {  
 GroupsSubjects = new HashSet<GroupSubject>();  
 Schedule = new HashSet<Schedule>();  
 SubSubjects = new HashSet<SubSubject>();  
 }  
  
 public int Id { get; set; }  
 public int LecturerId { get; set; }  
 public int InformationId { get; set; }  
  
 public virtual SubjectInformation Information { get; set; }  
 public virtual Staff Lecturer { get; set; }  
 public virtual ICollection<GroupSubject> GroupsSubjects { get; set; }  
 public virtual ICollection<Schedule> Schedule { get; set; }  
 public virtual ICollection<SubSubject> SubSubjects { get; set; }  
 }  
}

SubjectInformation.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class SubjectInformation  
 {  
 public SubjectInformation()  
 {  
 Subjects = new HashSet<Subject>();  
 }  
  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public string ShortName { get; set; }  
 public string Description { get; set; }  
  
 public virtual ICollection<Subject> Subjects { get; set; }  
 }  
}

SubSubject.cs

using System.Collections.Generic;  
  
namespace UniversityData.Entities  
{  
 public partial class SubSubject  
 {  
 public SubSubject()  
 {  
 Schedule = new HashSet<Schedule>();  
 }  
  
 public int Id { get; set; }  
 public int SubjectId { get; set; }  
 public string Name { get; set; }  
 public string Shortname { get; set; }  
 public int SubLecturerId { get; set; }  
  
 public virtual Staff SubLecturer { get; set; }  
 public virtual Subject Subject { get; set; }  
 public virtual ICollection<Schedule> Schedule { get; set; }  
 }  
}

ПРИЛОЖЕНИЕ 2

РУКОВОДСТВО ОПЕРАТОРА

П. 2.1. Назначение программы

Серверная часть системы предназначена для обеспечения механизма регистрации, авторизации пользователей в системе и для предоставления данных в удобном виде для мобильного приложения и планируемых в будущем клиентских программ. Предоставляемые для мобильного приложения данные делятся на данные для преподавателя и для студента. А также существуют общие методы, предназначенные для всех пользователей. Далее представлен полный список методов сервера, реализованных на данный момент.

Методы сервера для студенческого приложения:

* зарегистрировать студента;
* авторизовать студента;
* предоставить новости ВУЗа, актуальные для студента;
* предоставить список изучаемых студентом предметов;
* предоставить расписание пар студента;
* предоставить список группы студента;
* редактировать данные личного профиля студента.

Методы сервера для преподавательского приложения:

* предоставить новости ВУЗа, актуальные для преподавателя;
* зарегистрировать преподавателя;
* авторизовать преподавателя;
* предоставить список обучаемых преподавателем групп;
* предоставить список студентов в одной из обучаемых преподавателем групп;
* предоставить список преподаваемых предметов в одной из обучаемых преподавателем групп;
* редактировать данные личного профиля преподавателя;
* предоставить расписание пар преподавателя;

Общие методы сервера:

* предоставить данные профиля студента;
* предоставить данные по предмету;
* предоставить данные профиля преподавателя.

П. 2.2. Условия выполнения программы

Для обращения к методам сервера на мобильных устройствах с установленным мобильным приложением системы – необходим стабильный доступ к интернету. Язык программирования для разработки мобильных приложений должен позволять совершать HTTP-запросы на сервер.

П. 2.3. Выполнение программы

Для доступа к методам API (кроме регистрации и аутентификации) – необходимо в заголовке HTTP-запроса прислать токен доступа. Название заголовка должно быть: “Authorization”. Значение в заголовке должно иметь вид: “Bearer {токен доступа}” (без фигурных скобок). Если токен доступа не будет прислан или будет не валиден, то сервер пришлет в ответе код 401.

Если запрос на сервер подразумевает получение каких-либо данных от сервера, то данные передаются сервером в теле ответа в формате JSON. При успешной передаче данных код ответа сервера будет 200. Примерный вид передаваемого JSON файла представлен на рисунке П.2.1.

Примерный вид ответа сервера

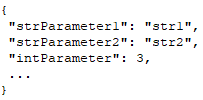


Рис. П.2.1

При какой-либо возникшей ошибке, связанной с неверными данными, полученными сервером от мобильного приложения (неправильный пароль пользователя, не существующий номер зачетной книжки и т.д.) – сервером формируется сообщение об ошибке и отправляется в мобильное приложение с кодом ответа 400. Пример сообщения об ошибке представлен на рисунке П.2.2.

Сообщение об ошибке регистрации



Рис. П.2.2

П. 2.3.1. Аутентификация и регистрация

Для регистрации необходимо послать запрос по адресу: “*http(s)://{адрес сервера}/api/registration/register-student*” – для студента, и “*http(s)://{адрес сервера}/api/registration/register-staff*” – для преподавателя, а также в параметрах запроса послать значения:

* studentNumber – номер зачетной книжки студента (приложение студента);
* staffId – идентификатор преподавателя (приложение преподавателя);
* username – имя пользователя;
* password – пароль пользователя.

В ответе на запрос при успешной регистрации сервер пришлет код ответа 200.

Для первой аутентификации студента необходимо послать запрос по адресу: “*http(s)://{адрес сервера}/api/auth/authorize-student”*, а также в параметрах запроса послать значения:

* username – имя пользователя;
* password – пароль пользователя.

В ответе на запрос при успешной аутентификации сервер пришлет код ответа 200 и данные в запросе. Среди присланных данных фигурируют параметры:

* userId – идентификатор пользователя в системе;
* accessToken – токен доступа;
* refreshToken – токен восстановления доступа.

П.2.3.2. Общие методы API для приложений

Для обращения к общим методам используется следующий адрес: “*http(s)://{адрес сервера}/api/university/common/{название метода}”*. Названия методов и их входные и выходные параметры:

* student-requisites – метод для получения данных профиля студента. Входной параметр: studentId – идентификатор студента. Выходные параметры:
* studentId – идентификатор студента;
* fullName – ФИО студента;
* groupName – имя группы студента;
* mail – электронный почтовый адрес студента;
* mobilePhone – номер мобильного телефона студента;
* homePhone – номер домашнего телефона студента.
* subject-information – метод для получения данных по предмету. Входной параметр: subjectId – идентификатор предмета. Выходные параметры:
* subjectId – идентификатор предмета;
* name – наименование предмета;
* lecturerId – идентификатор преподавателя;
* lecturerName – фамилия и инициалы преподавателя;
* description – описание предмета.
* staff-requisites – метод для получения данных профиля преподавателя. Входной параметр: staffId – идентификатор преподавателя.
* staffId – идентификатор преподавателя;
* fullName – ФИО преподавателя;
* departmentName – наименование кафедры преподавателя;
* mail – электронный почтовый адрес преподавателя;
* mobilePhone – номер мобильного телефона преподавателя;
* homePhone – номер домашнего телефона преподавателя.

П.2.3.3. Методы API для студенческого приложения

Для обращения к методам для студенческого приложения используется следующий адрес: “*http(s)://{адрес сервера}/api/university/student/{название метода}”*. Входным параметром всех методов кроме редактирования профиля является только userId (идентификатор студента пользователя). Названия методов и их входные и выходные параметры:

* news – метод для получения новостей ВУЗа, актуальных для студента. Выходные данные представлены набором новостей с параметрами вида:
* newsId – идентификатор новости;
* title – заголовок новости;
* date – дата новости;
* text – текст новости.
* subjects – метод для получения изучаемых предметов. Выходные данные представлены набором предметов с параметрами вида:
* subjectId – идентификатор предмета;
* name – наименование предмета;
* shortName – краткое наименование предмета.
* schedule – метод для получения расписания пар. Выходные данные представлены набором предметов-пар с параметрами вида:
* subjectId – идентификатор предмета;
* name – наименование предмета;
* location – место проведения пары;
* day – день недели проведения пары;
* weekLine – индикатор означающий под чертой/над чертой;
* staffName – фамилия и инициалы преподавателя.
* group – метод для получения списка группы студента. Выходные данные представлены параметром groupName (наименование группы) и набором студентов с параметрами вида:
* studentId – идентификатор студента;
* number – порядковый номер студента в группе;
* studentName – фамилия и инициалы студента;
* isHead – индикатор означающий является ли студент старостой.
* edit-profile – метод для редактирования личной информации пользователя. Входные и выходные данные метода идентичны выходным данным метода student-requisites в общих методах API.

П.2.3.4. Методы API для преподавательского приложения

Для обращения к методам для студенческого приложения используется следующий адрес: “*http(s)://{адрес сервера}/api/university/staff/{название метода}”*. Названия методов и их входные и выходные параметры:

* news – метод для получения новостей ВУЗа, актуальных для преподавателя. Выходные данные идентичны выходным данным метода news из API студенческого приложения. Входные данные: userId – идентификатор преподавателя пользователя;
* groups – метод для получения списка обучаемых преподавателем групп студентов. Входные данные: userId – идентификатор преподавателя пользователя. Выходные данные представлены набором групп с параметрами вида:
* groupId – идентификатор группы;
* groupName – наименование группы.
* group – метод для получения списка студентов группы. Входные параметры: groupId – идентификатор группы. Выходные данные идентичны методу group API студенческого приложения;
* subjects – метод для получения преподаваемых в группе предметов. Входные параметры: groupId – идентификатор группы. Выходные параметры идентичны методу subjects API студенческого приложения;
* schedule – метод для получения расписания преподавателя. Входные данные: userId – идентификатор преподавателя пользователя. Выходные данные представлены набором предметов-пар с параметрами вида:
* subjectId – идентификатор предмета;
* name – наименование предмета;
* location – место проведения пары;
* day – день недели проведения пары;
* weekLine – индикатор означающий под чертой/над чертой;
* groupName – наименование группы студентов.
* edit-profile – метод для редактирования личной информации пользователя. Входные и выходные данные метода идентичны выходным данным метода staff-requisites в общих методах API.

П.2.3.5. Разработанная часть студенческого приложения

Пример экранной формы приложения, показывающей профиль студента и использующей метод общего API – student-requisites представлен на рисунке П.2.3.

Профиль студента

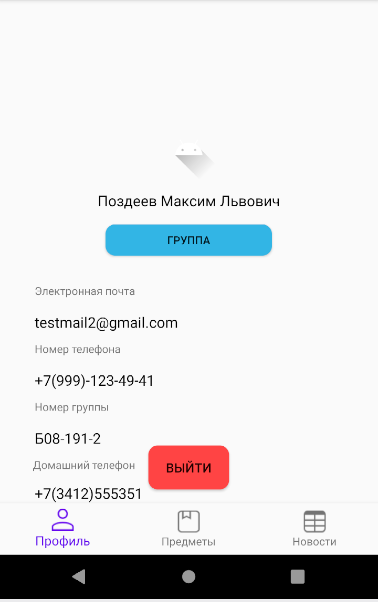


Рис. П.2.3

Пример экранной формы приложения, отображающей список изучаемых студентом предметов и использующей метод студенческого API – subjects представлен на рисунке П.2.4.

Список предметов

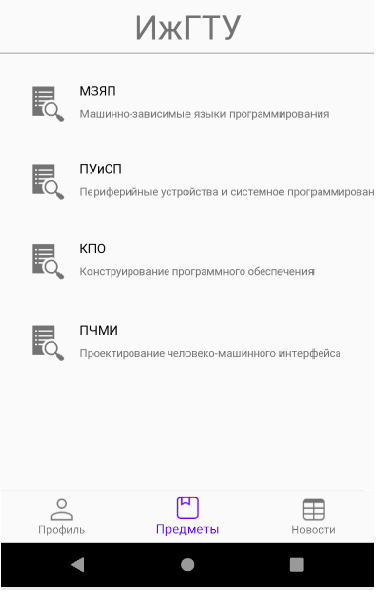


Рис. П.2.4

Пример экранной формы приложения, показывающей список новостей ВУЗа для студента и использующей метод news студенческого API представлен на рисунке П.2.5.

Список новостей

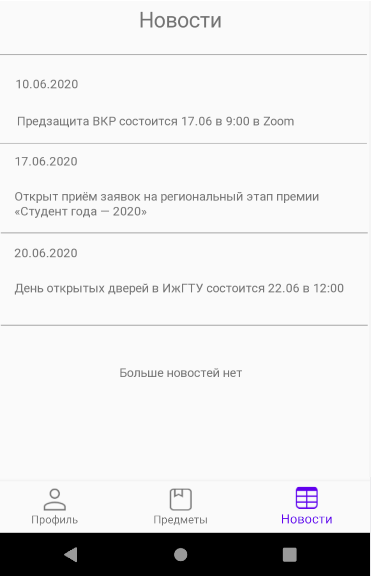


Рис. П.2.5