СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc133413839)

[1 ОБЩИЕ СВЕДЕНИЯ О ПРЕДПРИЯТИИ 4](#_Toc133413840)

[1.1 История организации 4](#_Toc133413841)

[1.2 Охрана труда и техника безопасности на рабочем месте 7](#_Toc133413842)

[2 АНАЛИТИЧЕСКИЙ ОБЗОР 9](#_Toc133413843)

[2.1 Обзор существующих средств автоматизации документооборота 9](#_Toc133413844)

[2.2 Постановка задачи 14](#_Toc133413845)

[2.3 Выбор средств реализации для сервера приложения 14](#_Toc133413846)

[2.4 Выбор хранилища данных 17](#_Toc133413847)

[2.5 Выбор средств реализации графического интерфейса приложения 19](#_Toc133413848)

[2.6 Выбор средст доставки и развёртывания 20](#_Toc133413849)

[3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ 22](#_Toc133413850)

[3.1 Функциональная схема 22](#_Toc133413851)

[3.2 Структурная схема 24](#_Toc133413852)

[3.3 Моделирование архитектуры приложения 24](#_Toc133413853)

[3.4 Моделирование базы данных 26](#_Toc133413854)

[ЗАКЛЮЧЕНИЕ 35](#_Toc133413855)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc133413856)

# **ВВЕДЕНИЕ**

Информация – это один из самых важных ресурсов для любой компании. Организация того или иного предприятия требует сбора информации, её последующего анализа и принятия на основе анализа решений, структуризации процессов. Чем сложнее предприятие, тем больше информации требуется.

Работа учреждений образования, особенно университетов, является процессом с весьма сложной структурой. Как следствие, работники данной сферы вынуждены обрабатывать и анализировать большие объёмы информации, а это в свою очередь приводит к необходимости увеличения документа оборота. В результате этого преподавательскому и руководящему составу приходится всё больше времени тратить на заполнение и проверку документации, больше усилий прилагать для передачи информации и предотвращения несанкционированного доступа. Это всё не может не сказываться на их здоровье и результативности труда.

Одним из способов упрощения ведения документации является автоматизация, то есть применение различных технических средств, которые берут на себя частичное или полное выполнение тех или иных процессов.

Современные компьютерные системы позволяют автоматизировать процессы проверки, составления и передачи документации.

Одним из документов, составление которого можно автоматизировать является учебная нагрузка по преподавателям.

Целью дипломной работы является проектирование и разработка программного комплекса для автоматизации распределения нагрузки по преподавателям.

# **1 АНАЛИТИЧЕСКИЙ ОБЗОР**

## **1.1 Обзор существующих средств автоматизации документооборота**

К разработке нового продукта нельзя приступать без анализа уже созданных решений, так как это позволит избежать многих недостатков, которые есть у конкурентов. Так как проблема автоматизации документооборота существует очень давно и очень важна, то существует множество программных решений для упрощения данного процесса.

Самым распространённым и простым способом является замена ручного заполнения документов и их передачи на использование продукции *Microsoft Office*. Данный подход позволяет упростить передачу документов, благородя возможности их синхронизации через систему *OneDrive*. Также нет сложности с ограничением доступа. Интерфейс программ из пакета *Microsoft Office* приведен на рисунке 1.1.

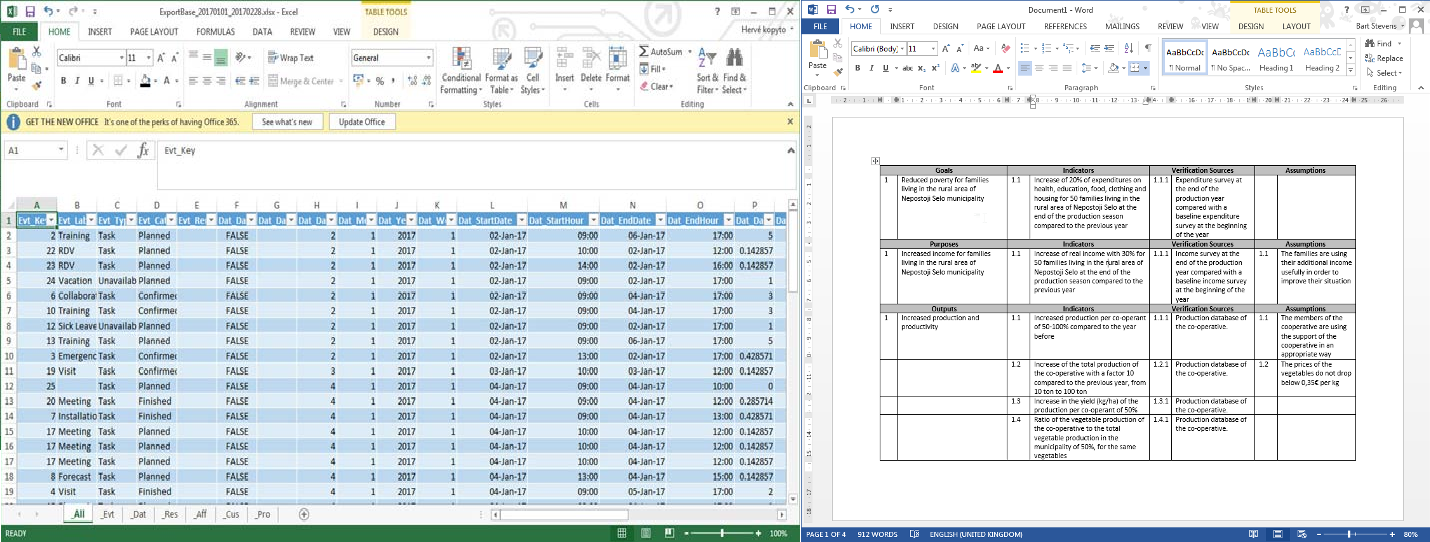


Рисунок 1.1 – Интерфейс программ из пакета *Microsoft Office*

При составлении документов можно использовать автоматическую систему расчёта тех или иных данных по формулам, заданным пользователями, однако для этого может понадобиться знание основ программирования на языке *Visual Basic* и математических функций в системе, что повышает входной порог для пользователей.

Стоит отметить, что так как система *Microsoft Office* разработана для работы с различными документами, то в интерфейсе программ из данного пакета можно обнаружить большое количество функций, которые не используются, но при этом уменьшают площадь рабочей поверхности и усложняют поиск нужных функций.

Также cреди минусов можно выделить:

* высокая стоимость;
* возможность установки только на компьютеры, работающие на операционной системе *Microsoft Windows* и *macOS.*

Популярным продуктом для автоматизации создания отчётов и документооборота является использование систем на базе 1C. Интерфейс системы представлен на рисунке 1.2.

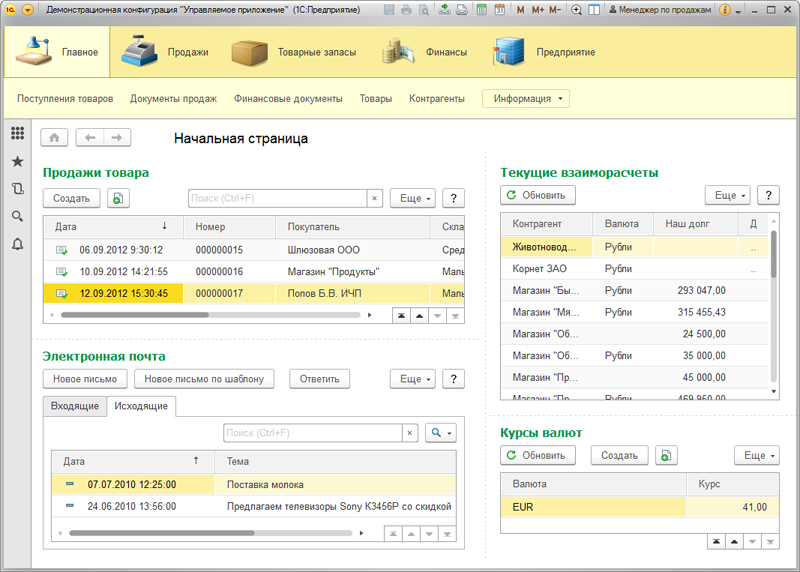


Рисунок 1.2 – Интерфейс системы на основе 1C

Благодаря модулю 1С Документооборот Проф можно с лёгкостью ограничить доступ к конкретным файлам работникам, которым не разрешается просматривать определённые документы [1].

Составление отчётов в данной системе упрощается за счёт использования справочников – объектов, имеющих одинаковую структуру, списочный характер и хранятся во внутренней базе данных [2]. Это может быть, например, список преподавателей, перечень предметов и так далее. Благодаря этому при составлении документов есть возможность ссылаться на тот или иной объект что упрощает проверку, так как отсутствует необходимость каждый раз заполнять одинаковые данные.

Стоит отметь также что в программе можно ограничить не только доступ к документам, но и к различному функционалу для каждого пользователя, за счёт этого упрощается взаимодействие с системой.

Главными недостатками данной системы является то, что её необходимо индивидуально настраивать под каждую организацию, для чего требуются соответствующие специалисты и необходимость оплаты различного дополнительного функционала, например, облачной синхронизации документов.

*ECM* *ELMA365* – российская система электронного документооборота. Интерфейс *ECM ELMA365* представлен на рисунке 1.3.

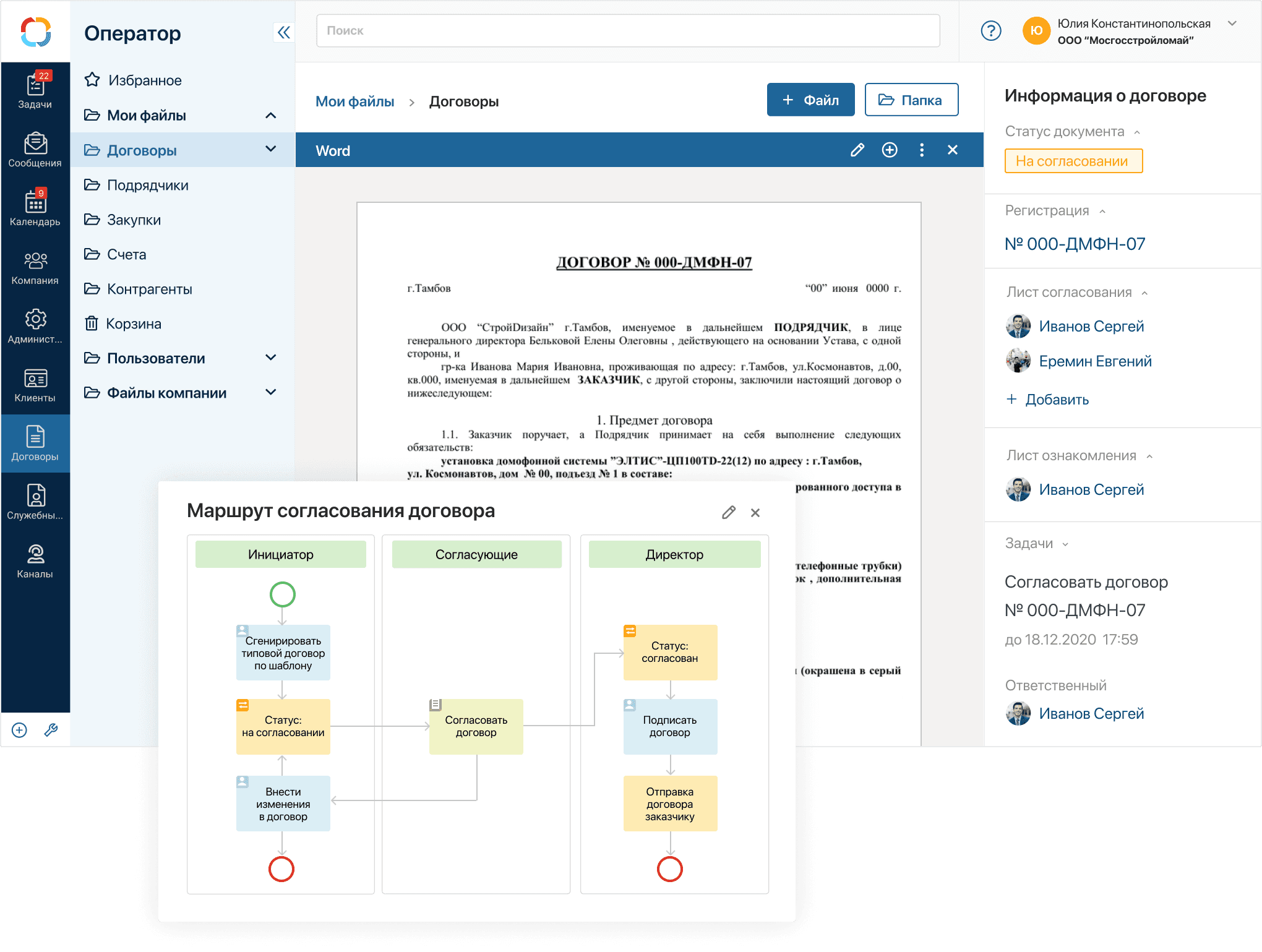


Рисунок 1.3 – Интерфейс *ECM* *ELMA365*

Данная система имеет как *web,* так и *desktop* приложения, что позволяет использовать её на различных устройствах в независимости от операционных систем. Позволяет создавать шаблоны разнообразных документов, а это уменьшает время работы пользователя.

Имеется возможность настройки уведомлений об истечении сроков использования различных документов, а также напоминания о их необходимости заполнения отчётов.

Среди недостатков можно выделить то, что данная система позволяет лишь хранить документы, а не объекты как системы, построенные на базе 1C, что приводит к необходимости контроля за заполняемыми данными.

Также большой популярностью пользуется программа *Directum*. Интерфейс программы представлен на рисунке 1.4.

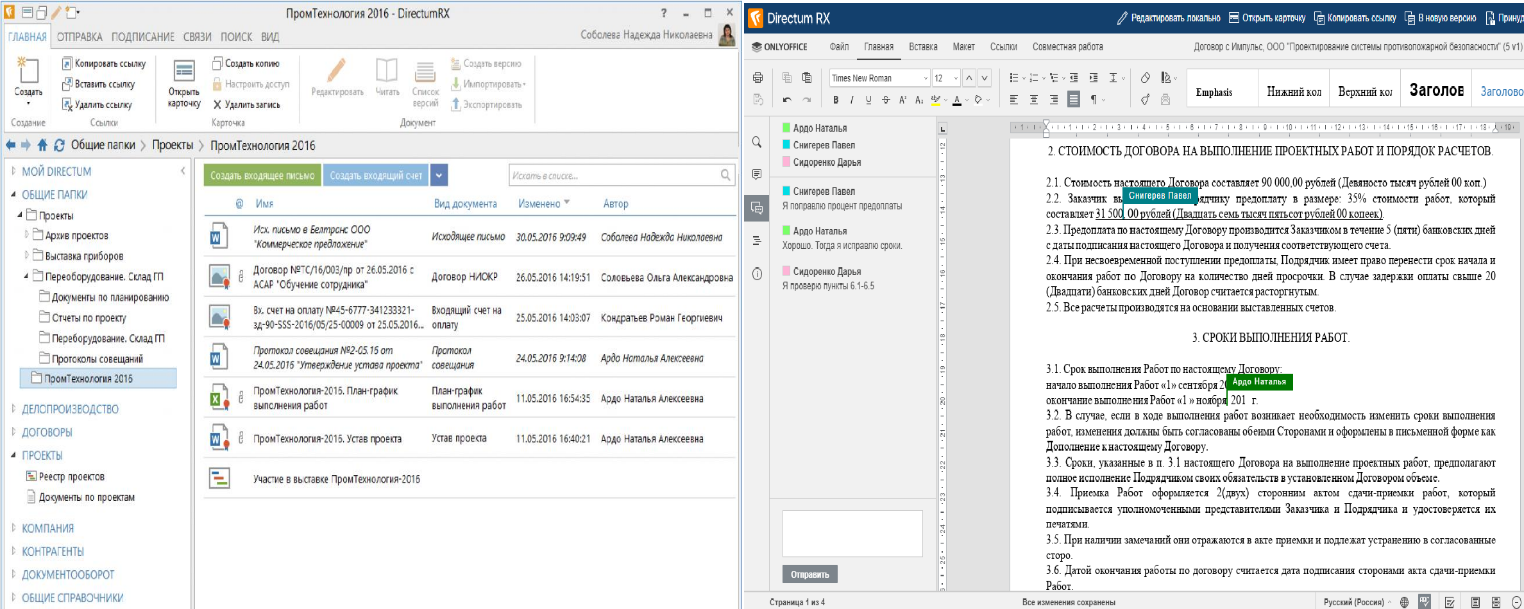


Рисунок 1.4 – Интерфейс программы *Directum*

Данная программа обладает следующими минусами:

* высокая стоимость покупки;
* универсальная архитектура, из-за чего программа содержит большое количество функций и настроек, которые не востребованы для задачи автоматизации заполнения плана нагрузки преподавателей.

Однако она обладает и рядом преимуществ. Среди них:

* наличие интернет-версии, так как это позволяет использовать систему в независимости от операционной системы;
* наличие системы уведомлений о подтверждении документов, истечении сроков и необходимости просмотра тех или иных документов;
* возможность настройки пути документов после выполнения разных этапов работы, например, отправка руководителю после составления документа, а после подтверждения от руководителя отправка документа в соответствующий отдел предприятия.

Система электронного документа оборота Атлас часто используется в учреждениях образования для схожих целей. Её интерфейс представлен на рисунке 1.5.

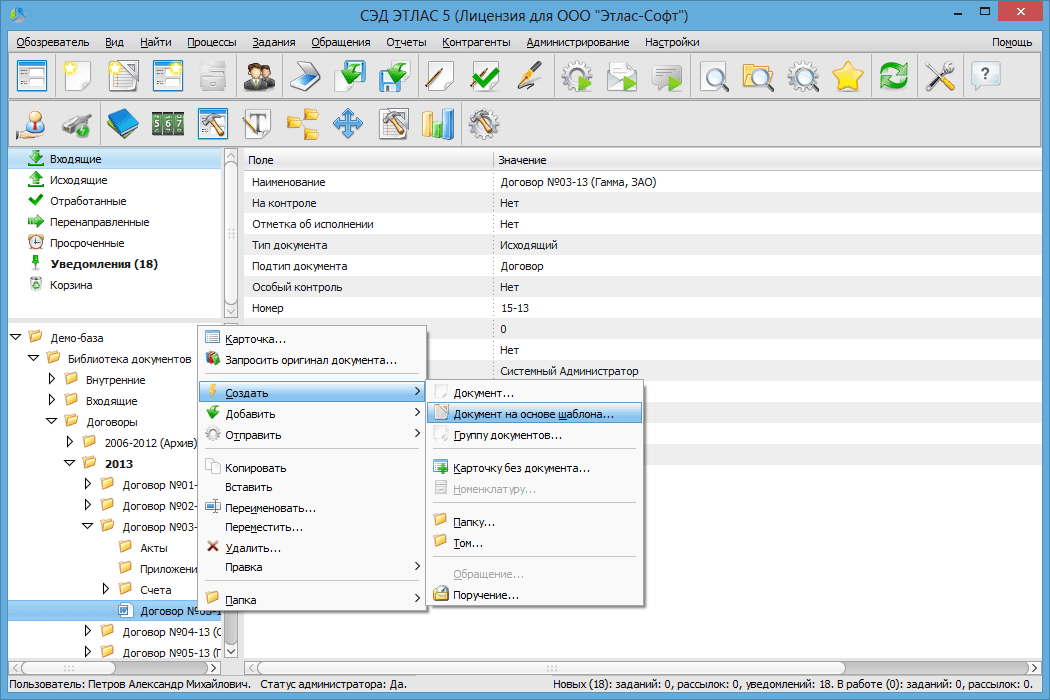


Рисунок 1.5 – Интерфейс программы Атлас

К её недостаткам можно отнести:

* сложность подключения к системе, то есть настройка и подключение к системе может потребовать значительных усилий и знаний в области информационных технологий;
* необходимость оформления платной подписки для использования системы.

Однако данная система обладает и рядом преимуществ, а именно:

* + возможность настройки интерфейса пользователя в соответствии с его ролью;
  + удобная система поиска и фильтрации данных.

Таким образом можно выделить следующие недостатками конкурентов, которых не должно быть в разрабатываемой системе:

* высокая стоимость использования, что важно в сфере образования;
* ориентация на определённые операционные системы, что снижает круг возможных пользователей;
* наличие лишнего функционала, который усложняет использование и повышает входной порог для пользователей;
* аналоги не учитывают специфики предметной области.

**1.2 Постановка задачи**

На основании произведённого анализа, а также требований, которые были выдвинуты в задании по практике. Создаваемое программное обеспечение должно включать в себя такие функции, как:

* рассылка уведомлений о необходимости составления плана, о скором окончании сроков сдачи;
* автоматическая генерация *excel*-отчётов на основе обрабатываемых данных;
* автоматическая проверка распределения данных;
* возможность удалённой работы с данными;
* авторизацию и аутентификацию пользователей как по ролям, так и по кафедрам.
* хранение и просмотр отчётов за различные промежутки времени;
* фильтрацию данных по различным параметрам, при просмотре отчётов.

Из требований к архитектуре можно выделить, что приложение должно иметь возможность установки на компьютеры с различной операционной системой, а также предоставлять веб-интерфейс для конечных пользователей.

## **1.3 Выбор средств реализации для сервера приложения**

Основываясь на выдвинутых выше требованиях для создания сервера информационной системы, был выбран объектно-ориентированный язык программирования *C#* под управлением платформы *.Net 6*.

*.Net* – платформа для разработки, включающая общеязыковую исполняющую среду (*Common Language Runtime*), которая управляет выполнением кода и обеспечивает создателей программ богатыми библиотеками классов для написания приложений [3, с. 48].

* Платформа *.Net* обладает большим количеством положительных качеств. Например,поддержка кроссплатформенности обеспечивает возможность создания приложений для различных операционных систем, включая *Windows, macOS* и *Linux*. Это позволяет разработчикам использовать один язык программирования и одни и те же библиотеки для создания приложений, что снижает общее время разработки.

Платформа позволяет разработчикам создавать масштабируемые приложения, способные выдерживать большую нагрузку и обрабатывать большие объёмы данных. С помощью таких технологий, как параллельное программирование и асинхронные операции, можно эффективно использовать ресурсы и повысить производительность.

Вокруг *.NET* сформировалось активное и разнообразное сообщество разработчиков, которые делятся знаниями, опытом и решениями проблем. Это обеспечивает доступ к обширной базе знаний, обновлениям, учебным материалам и поддержке, что помогает разработчикам легче справляться с задачами и находить ответы на свои вопросы.

Она поддерживает использование *COM*-объектов, взаимодействие с базами данных и интеграцию с различными протоколами связи. Это позволяет разрабатывать приложения, которые легко взаимодействуют с существующими решениями и инфраструктурой.

*.NET* обладает мощным *JIT*-компилятором, который обеспечивает высокую производительность выполнения кода. Благодаря компиляции в машинный код во время выполнения, приложения на *.NET* имеют быструю загрузку и выполнение, а также оптимизированное использование системных ресурсов.

*Microsoft* активно поддерживает и обновляет платформу *.NET*. Разработчики могут рассчитывать на постоянное обновление и поддержку платформы, что помогает обеспечить безопасность и стабильность приложений.

Интеграция с другими сервисами *Microsoft* также облегчает разработку.

Язык *С#* выбран среди других языков данной платформы так как он является универсальным, безопасным в отношении типов, объектно-ориентированным языком программирования. Цель *C#* заключается в обеспечении продуктивности работы программистов. Для этого в языке соблюдается баланс между простотой, выразительностью и производительностью. С самой первой версии главным архитектором языка *C#* был Андерс Хейлсберг (создатель *Turbo Pascal* и архитектор *Delphi*). Язык *C#* нейтрален в отношении платформ и работает с рядом исполняющих сред, специфических для платформ [4, c. 33].

Строгая типизация в *C#* позволяет выявлять ошибки на этапе компиляции и обеспечивает более безопасную работу с типами данных, улучшает производительность приложений.

Наличие активного и большого сообщества пользователей, включая крупные компании, которые осуществляют обмен опытом и помогают в решении проблем, способствует ускорению разработки.

В качестве среды разработки была выбрана *Visual Studio Code*.

*Visual Studio Code* – современная и упрощённая кросс-платформенная среда разработки, разработанная компанией *Microsoft*. Среду можно запустить во всех распространённых операционных системах, включая *Windows, macOS* и множество разновидностей *Linux*, таких как *Red Hat Enterprise Linux* (*RHEL*) и *Ubuntu*.

*Visual Studio Code* хорошо подходит для современной кросс-платформенной разработки, поскольку имеет богатый пополняемый набор расширений для поддержки многих языков, помимо *C#*.

*Visual Studio Code* содержит широкий спектр инструментов для веб-разработки.

*Visual Studio Code* поддерживается процессорами *ARM*, поэтому на ней можно разрабатывать приложения на компьютерах *Apple Silicon и Raspberry Pi.*

На сегодняшний день *Visual Studio Code* — самая популярная интегрированная среда разработки: по результатам опроса Stack Overflow 2021 более 70 % профессиональных разработчиков выбрали именно её [5, c. 41].

Для взаимодействия с базой данных был выбран фреймворк *Entity Framework* *Core*.

Высокий уровень абстракции облегчает разработку, так как позволяет работать с базой данных по средствам обычных объектов, что позволяет не тратить время на составление различных запросов к базе данных и абстрагироваться от устройства базы данных. Также *Entity Framework Core* обеспечивает автоматическое управление отношениями между объектами и транзакциями базы данных. Он позволяет легко устанавливать и поддерживать связи между таблицами, гарантирует целостность данных с помощью управления транзакциями.

В данном фреймворке реализована поддержка асинхронных запросов и кэширования данных, что положительно сказывается на производительности;

Фреймворк поддерживает возможность автоматического создания миграций базы данных, то есть обновление архитектуры базы данных с переносом данных, что полезно при увеличении функционала и поддержке разрабатываемого приложения [6];

Фреймворк поддерживает инкрементальную загрузку данных, что означает, что связанные данные загружаются по мере необходимости. Это помогает уменьшить количество данных, извлекаемых из базы данных, и повышает производительность приложения.

*Entity Framework* *Core* предоставляет удобные инструменты для тестирования и отладки кода. Например, возможно использовать тестовую реализацию контекста данных для создания модульных тестов, а также использовать инструменты отладки для отслеживания и анализа запросов к базе данных.

Встроенная система логирования операций позволяет при возникновении ошибок взаимодействия с базой данных быстро их локализировать и устранить.

Для генерации *excel* документов на основе заполненных данных выбрана библиотека *GemBox*. Главными преимуществами данной библиотеки являются:

* независимость от наличия *Microsoft Excel*;
* возможность использования большей части функционала *Microsoft Excel*, что при необходимости позволит в будущем без особых проблем дополнить отчёты нужными данными;
* наличие качественной документации;
* поддержка работы с документами разных форматов *Excel(XLSX, XLS, XLSB, ODS)*, текстовые файлы (*CSV, TXT*) и *HTML*-файлы [7].

Для обработки входящих запросов был выбран фреймворк *ASP.NET Core*. Данный фремворк обладает множеством преимуществ, среди них:

* модульность – различные компоненты фреймворка можно установить через пакетный менеджер *Nuget*, а отсутствие лишних модулей уменьшает вероятность конфликта версий между ними, объем ресурсов необходимых для работы приложения;
* встроенная система управления зависимостями, что позволяет писать красивый, читабельный код;
* собственный кросс-платформенный *web*-сервер *Kestrel*, благодаря чему приложение с использованием *ASP.NET Core* может быть запущенно на любой операционной системе;
* высокая производительность – фреймворк использует инновационные технологии для обработки запросов и оптимизации производительности, такие как кэширование, асинхронность;
* безопасность – *ASP.NET Core* предоставляет встроенную поддержку для защиты от атак и уязвимостей, таких как кросс-сайтовые скрипты;
* современные подходы к разработке – *ASP.NET Core* поддерживает современные подходы к разработке *web*-приложений, такие как *REST API и SPA*, а также предоставляет встроенную поддержку для *Docker и Kubernetes*;

Для увеличения производительности и упрощения разработки при предварительной проверке данных входящих запросов выбрана библиотека *FluentValidation*.

Для обеспечения функций авторизации и аутентификации выбран фреймворк *ASP.NET Core Identity*. Плюсами данной библиотеки является:

* встроенная защита от атак – *ASP.NET Core Identity* включает в себя механизмы защиты от атак, такие как защита от подделки маркеров, предотвращение многократной отправки форм и проверка безопасности паролей;
* гибкость и расширяемость *– ASP.NET Core Identity* является гибкой и расширяемой системой, которая позволяет разработчикам добавлять свои собственные функции и настраивать систему аутентификации и авторизации в соответствии с требованиями приложения;
* интеграция с другими сервисами – *ASP.NET Core Identity* интегрируется с другими сервисами, такими как системы управления учетными записями *Active Directory и LDAP*, что облегчает взаимодействие с другими системами;
* автоматизация настройки базы данных – *ASP.NET Core Identity* содержит инструменты для автоматического создания таблиц, их связей и ограничений в базе данных.

Для осуществления отправки уведомлений выбран сервис *Google SMPT*. Его преимущества это:

* надежность – *Google SMTP* обеспечивает высокую надежность, система масштабируется и устойчива к сбоям, что позволяет ей обрабатывать большой объем почты без потерь;
* безопасность – *Google SMTP* обеспечивает высокий уровень безопасности, используя *SSL/TLS* для шифрования и защиты передаваемых данных, что означает даёт пользователям уверенность в том, что их данные и личная информация не будут доступны злоумышленникам;
* легкость настройки – *Google SMTP* легко настраивается и может быть использован с любым клиентом электронной почты, который поддерживает протокол *SMTP*; это означает, что пользователи могут продолжать использовать свой любимый клиент электронной почты, но в то же время получат преимущества, предоставляемые *Google SMTP*;
* высокая скорость доставки – *Google SMTP* предоставляет высокую скорость доставки электронной почты, которая обеспечивается благодаря высокоскоростной инфраструктуре *Google* и оптимизации процессов доставки;
* доступность – *Google SMTP* предоставляет услуги абсолютно бесплатно.

## **1.4 Выбор хранилища данных**

Отчёт о нагрузке по преподавателям имеет сложную структуру данных, попытка нормализации подобных данных приведёт к большому времени ожидания получения результатов. Один из способов решение данной проблемы – использование документной базы данных.

Документная база данных – это тип нереляционных баз данных, предназначенный для хранения и запроса данных в виде документов в формате, подобном *JSON*. Документные базы данных позволяют хранить и запрашивать данные в БД с помощью той же документной модели, которую они используют в коде приложения. Гибкий, полуструктурированный, иерархический характер документов и документных баз данных позволяет им развиваться в соответствии с потребностями приложений.

Документные базы данных обеспечивают гибкость индексации, производительность выполнения стандартных запросов и аналитику наборов документов [8].

*MongoDb* – одна из самых популярных документных баз данных на сегодняшний момент. Данная база данных абсолютно бесплатная и обладает следующим набором преимуществ:

* кроссплатформенность – *mongoDB* написана на *C++*, поэтому ее легко портировать на самые разные платформы, например, *MongoDB* может быть развёрнута на платформах *Windows, Linux, MacOS, Solaris*;
* балансировка нагрузки – *MongoDB* использует концепцию шардинга для горизонтального масштабирования с помощью разделения данных между несколькими экземплярами БД, она может работать на нескольких серверах, балансируя нагрузку и дублируя данные, чтобы поддерживать работоспособность системы в случае аппаратного сбоя;
* специальные запросы – *MongoDB* поддерживает поиск по полям, диапазонные запросы и поиск по регулярным выражениям, могут быть сделаны запросы для возврата определённых полей в документах;

Однако в процессе заполнения отчётов, а также для работы с данными пользователей необходимо поддерживать высокую связанность данных, обеспечить возможность выполнения транзакций. Для этого необходимо использовать дополнительную реляционную базу данных. Оптимальным выбором для неё является *MySql*.

Главными преимуществами *MySql* являются:

* многочисленное сообщество пользователей и разработчиков, что облегчает поиск помощи и поддержки при возникновении проблем, а большое количество документации, форумов и онлайн-ресурсов, которые помогут быстро найти ответы на интересующие вопросы;
* надежность и стабильность в работе, которые обеспечиваются благодаря встроенным механизмам резервного копирования, восстановления данных и обеспечения целостности базы данных;
* наличие возможности шифрования данных и авторизации пользователей, что обеспечивая высокий уровень безопасности;
* гибкость, заключающаяся в том, что *MySql* поддерживает различные операционные системы, такие как *Windows, Linux, macOS* и другие, а также интегрируется с различными языками программирования, такими как *C#, Python, Java* и другими, что делает его универсальным решением для разработки приложений;
* высокая производительность, благодаря чему *MySQL* может обрабатывать большие объемы данных с высокой скоростью, он оптимизирован для работы с транзакциями и параллельными запросами, что делает его отличным выбором для приложений с высокой нагрузкой;
* бесплатная и открытая система, что даёт возможность бесплатно загружать, использовать и модифицировать ее исходный код в соответствии с лицензией *GNU General Public License (GPL)*, такое положение дел делает *MySQL* доступным для широкого круга пользователей и разработчиков.

Таким образом для работы приложения выбраны две базы данных – одна для хранения информации необходимой для заполнения отчётов и самих отчётов, вторая для авторизации и аутентификации пользователей, а также работы с их учётными записями.

## **1.5 Выбор средств реализации графического интерфейса приложения**

Для создания интерфейса пользователя был выбран язык *JavaScript*, а не *C#* так как:

* + - *JavaScript* является языком, который выполняется непосредственно в браузере, что позволяет создавать динамические и интерактивные *web*-страницы без необходимости отправлять запросы на сервер и загружать страницы заново; эти делает *web*-страницы более быстрыми и отзывчивыми для пользователей;
    - *JavaScript* имеет более широкую поддержку и экосистему для *web*-разработки, чем *C#,* т.к. Существует огромное количество инструментов и библиотек для *JavaScript*, которые облегчают разработку и расширяют возможности языка, а Некоторые из них, такие как *React* и *Angular*, являются одними из самых популярных фреймворков для разработки веб-приложений;
    - *JavaScript* является более удобным языком для работы с *HTML*;
    - *JavaScript* является стандартом для *web*-разработки, то есть большинство *web*-браузеров поддерживают его, в то время как *C#* является языком для платформы *.NET*, который требует специального программного обеспечения, чтобы быть выполненным в браузере.

Так как чистый *JavaScript* не имеет высокоуровневых интерфейсов, для разработки крупных веб-приложений, то дополнительно необходимо использовать фреймворк предоставляющий данный функционал. Наиболее оптимальным является *React*. Он имеет множество преимуществ.

*React* использует виртуальный *DOM (Virtual DOM)*, который обеспечивает более быстрое обновление страницы и улучшает производительность в целом. Виртуальный *DOM* является легковесной копией реального *DOM*, которая позволяет сравнивать изменения между предыдущим и новым состоянием страницы, и обновлять только те части, которые действительно изменились;

Благородя разбиению кода на отдельные блоки-компоненты, появляется возможность сделать код более модульным, переиспользуемым и легко поддерживаемым. Компоненты могут быть повторно использованы в различных частях приложения, что упрощает разработку и уменьшает количество дублирующегося кода;

Использование языка *JSX* – расширения для языка *JavaScript*, которое позволяет использовать *HTML*-подобный синтаксис для описания пользовательского интерфейса, делает код более понятным и легко читаемым;

Большое сообщество и экосистема разработчиков, которые создали множество полезных инструментов и библиотек для упрощения разработки. Экосистема *React* насчитывает множество сторонних библиотек и инструментов, что делает его более гибким и удобным для разработки;

*React* был создан командой *Facebook*, которая продолжает его развивать и поддерживать. Это означает, что фреймворк имеет высокую степень стабильности и надежности.

Добавление динамической функциональности является весьма объёмной задачей с обычным *JavaScript*. Для упрощения данной задачи был выбран фреймворк *jQuery*. Он был разработан именно для того, чтобы сделать наиболее общие задачи тривиальными. Например, чтобы создать таблицу с разным цветом фона для чётных и нечётных строк, дизайнеру потребуется написать до 10 строк кода на языке *JavaScript*. А аналогичный скрипт, написанный с использованием *jQuery* поместиться в рамках одной строки [9, c. 26].

Упрощение работы происходит за счёт того, что *jQuery* предоставляет простой синтаксис для выбора элементов на веб-странице с использованием селекторов *CSS*. Это упрощает поиск и манипуляцию элементами *DOM*, что делает код более читабельным и понятным.

*jQuery* предоставляет простой способ привязки обработчиков событий к элементам *DOM*. Вы можете легко реагировать на события, такие как щелчок мыши, наведение курсора и нажатие клавиш, и выполнять соответствующие действия.

Также стоит отметить, что *jQuery* облегчает выполнение *AJAX*-запросов к серверу. Он предоставляет удобные методы для отправки и получения данных асинхронно, без перезагрузки страницы. Это позволяет создавать динамические веб-приложения с возможностью взаимодействия с сервером без прерывания пользовательского опыта.

## **1.6 Выбор средств доставки и развёртывания**

Разрабатываемый программный комплекс имеет сложную структуру. Это делает проблематичной задачи сборки и установки комплекса. Помимо того, что процесс настройки среды может оказаться довольно сложным, также можно столкнуться с проблемой, что на мишине-хостинге будут отсутствовать необходимые программные компоненты или, наоборот, будут присутствовать компоненты, конфликтующие с устанавливаемыми. Данную проблему можно решить, прибегнув к контейнеризации.

Контейнеризация – это процесс развёртывания программного обеспечения, который объединяет код приложения со всеми файлами и библиотеками, необходимыми для запуска в любой инфраструктуре. С помощью контейнеризации можно создать единый программный пакет или контейнер, который будет работать на всех типах устройств и операционных систем [10].

Для создания контейнеров и их последующего запуска выбран инструмент *Docker*.

Созданный контейнер можно автоматически доставить на компьютер-хост и запустить тем, самым устранив необходимость вручную следить за обновлениями и устанавливать их. Помимо этого, также можно автоматически и упаковывать создавать данный контейнер при релизе новой версии.

Подобная автоматизация упаковки, доставки и установки называется методологией непрерывной доставки и интеграции*.*

Для также для выполнения данной задачи необходимо воспользоваться системой контроля версий. Помимо обеспечения процесса непрерывной интеграции и развёртывания система контроля версий имеет много других важных положительных качеств:

* СКВ предоставляет возможность сохранять различные версии документов или кода, что позволяет легко переходить между ними, находить ошибки и изменения.
* СКВ позволяет нескольким пользователям работать над одним проектом и вносить изменения в документы и код, учитывая все внесенные изменения и отслеживая их историю.
* СКВ сохраняет историю изменений, поэтому вы можете легко восстановить документ или код в случае сбоя или потери данных.
* СКВ автоматически отслеживает изменения, которые могут вызвать конфликты в коде, и предоставляет инструменты для их разрешения;
* СКВ позволяет легко отслеживать, кто внес изменения в код, когда это произошло, и что было изменено, а это позволяет легко узнать, какие части проекта ещё не были завершены.
* СКВ может ускорить процесс разработки, поскольку позволяет быстро возвращаться к предыдущим версиям, а также параллельно работать над несколькими задачами.
* СКВ автоматически сохраняет историю изменений, что упрощает подготовку документации по проекту.

Сохранение проектов на удалённом сервере позволяет уменьшить риск потери данных при отказе оборудования;

В качестве системы контроля версий выбрана *GitLab*. Она обладает следующими преимуществами:

* управление непрерывной интеграцией и доставкой осуществляется по средствами встроенной в системы непрерывной интеграции и доставки, которая позволяет быстро создавать, тестировать и доставлять программное обеспечение, конкуренты также предоставляют инструменты для обеспечения данных процессов, но они доступны только в платных весях;
* *GitLab* позволяет установить платформу на своём сервере, что даёт большую гибкость и контроль над конфигурацией и безопасностью;
* встроенный редактор кода, который позволяет быстро редактировать код в браузере без необходимости скачивать его на компьютер;
* высокий уровень безопасности, за счёт предоставления широкого набора инструментов, таких как проверка кода на наличие уязвимостей и механизмы защиты от атак.

**1.7 Вывод по разделу**

На основании произведённого анализа для реализации серверной части приложения был выбран объектно-ориентированный язык программирования C# под управлением платформы .Net, так как подобное сочетание языка программирования и платформы позволяет использовать большую библиотеку разработанных классов, а также разнообразные инструменты, упрощающие разработку.

Для улучшения пользовательского опыта в качестве баз данных было выбрано две системы: *MongoDb* и *MySql*. Данный подход позволяет сохранять связанность данных во время работы и обеспечить быструю загрузку уже обработанных данных.

Для взаимодействия с базами данных был выбран фреймворк *Entity Framework Core*, так как он позволяет абстрагироваться от устройства баз данных, что особенно полезно в случае использования различных баз данных. Помимо этого, данный фреймворк содержит внутренние инструменты кэширования данных и предоставляет возможность использования асинхронных операций, что положительно сказывается на производительности.

Для работы с *html* запросами был выбран фреймворк *ASP.NET Core*. Данный фреймворк обладает хорошей производительностью, обширным набором инструментов, например, всторенной системы внедрения зависимостей, что существенно упрощает разработку.

Для авторизации и аутентификации пользователей был выбран фреймворк *ASP.NET Core Identity,* так как он обладает механизмами для лёгкой интеграции с системами на *ASP.NET Core,* а также поддерживает автоматическое таблиц в базе данных и установления связей между ними.

Для реализации графического интерфейса был выбран язык *JavaScript* и фреймворки *jQuery* и *React*. Данный подход позволяет создать динамический интерфейс с модульной структурой, а это в свою очередь уменьшает время разработки и улучшает производительность итогового продукта.

**2 АРХИТЕКТУРА ПРИЛОЖЕНИЯ**

**2.1 Функциональная схема**

На рисунке 2.1 представлена функциональная схема, составленная для данной системы. Функциональная схема необходима, чтобы показать взаимодействия между пользователями и системой.



Рисунок 2.1 – Функциональная схема

В системе можно выделить следующих актёров:

* актёр «Администратор» выполняет контроль над данными пользователей и настройками системы;
* актёр «Заведующий» занимается распределением нагрузки;
* актёр «Преподаватель» может ознакомится со своим планом нагрузки.

В системе можно выделить следующие прецеденты:

* прецедент «Авторизация и аутентификация» – прецедент, который является общим для всех актёров и вызывается перед началом работы с приложением;
* прецедент «Просмотр списка пользователей как заведующий» вызывается актёром «Заведующий» для просмотра списка пользователей и выбора пользователя для управления его рабочими ставками;
* прецедент «Редактирование информации о количестве рабочих ставок» вызывается актёром «Заведующий» для редактирования информации о количестве рабочих ставок у того или иного пользователя;
* прецедент «Просмотр планов распределения нагрузки» вызывается актёром «Заведующий» для просмотра списка всех планов распределения нагрузки, относящихся к кафедре актёра, расширяется прецедентами «Распределение нагрузки по преподавателям в текущем году»;
* прецедент «Распределение нагрузки по преподавателям в текущем году» расширяется прецедентом «Отправка планов распределения нагрузки для ознакомления» и позволяет создавать и редактировать план распределения нагрузки на текущий год;
* прецедент «Отправка планов распределения нагрузки для ознакомления» вызывается при завершении работы над планом и предоставляет пользователям ознакомится с их планом нагрузки в текущем году;
* прецедент «Ознакомление с планом нагрузки» вызывается актёром «Преподаватель» и позволяет ему ознакомится с планом нагрузки на текущий год;
* прецедент «Просмотр списка пользователей как администратор» вызывается актёром «Администратор» для просмотра списка всех пользователей, расширяется прецедентом «Редактирование информации о пользователе»;
* прецедент «Редактирование информации о пользователе» вызывается для изменения информации о пользователе;
* прецедент «Установка параметров системы» вызывается актёром «Администратор» для настройки данных системы, расширяется прецедентами «Установка сроков подготовки и проверки распределения нагрузки» и «Установка сроков отправки уведомлений», которые осуществляют отправку уведомлений пользователям.

## **2.2 Структурная схема**

Структура программы включает в себя описание модулей, их взаимосвязей, а также данных, используемых в программе. Для разработки сложных программ необходимо применять специальные приемы, позволяющие создать рациональную структуру программы и сократить объем программирования. Структура программы может быть представлена в виде иерархической схемы, отражающей подчиненность модулей. Однако она не отображает порядок вызова модулей и функционирование программы. Выделение структурной схемы позволяет снизить порог опыта, необходимого разработчику для реализации продукта и сократить количество ошибок при разработке. Это также помогает снизить время разработки программного продукта. Структурная схема изображена на рисунке 2.2.

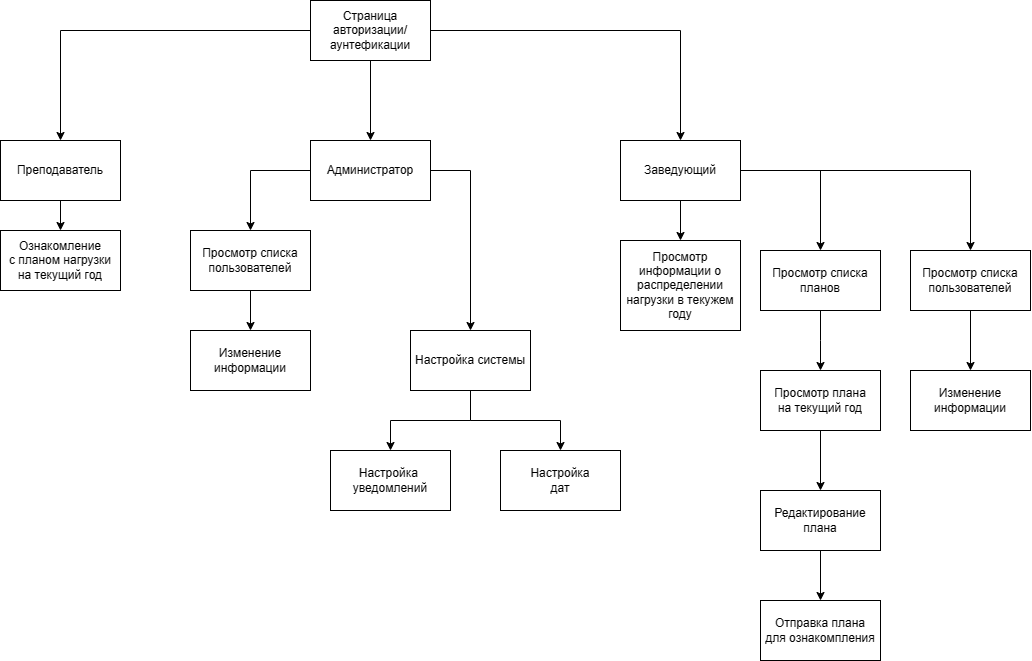


Рисунок 2.2 – Структурная схема приложения

**2.3 Моделирование архитектуры приложения**

В основе программного комплекса лежит микросервисная архитектура. Микросервисы — это небольшие, автономные, совместно работающие сервисы [11, c. 7]. Подобная атомизация ускоряет разработку, так как упрощает добавление нового функционала и понимание структуры проекта.

В качестве архитектуры сервера по обработке бизнес-логики оптимальным выбором является использование *API* построенного с учётом принципов *REAST* и применением паттерна *CQRS.*

Суть паттерна *CQRS* заключается в разделении запросов на две категории:

* получение данных;
* изменение данных.

С выделением отдельных классов по обработке разных видов запросов и создании разнообразных моделей. Схема работы данного паттерна изображена на рисунке 2.3.

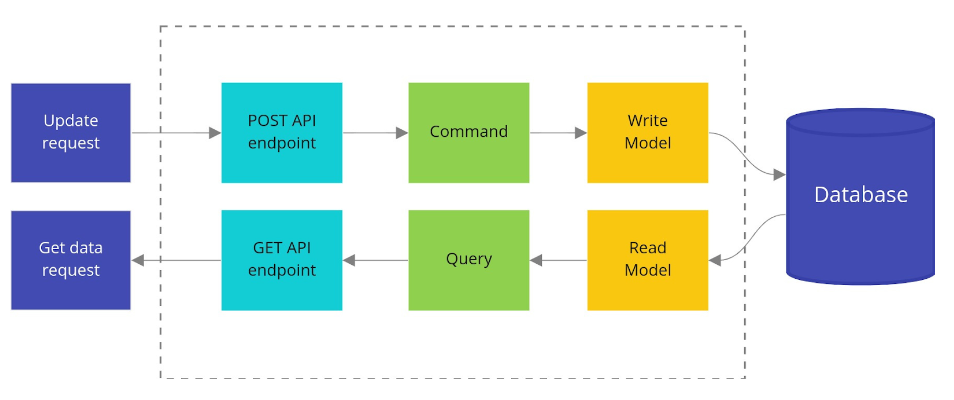


Рисунок 2.3 – Схема работы паттерна *CQRS*

Данный паттерн внедрён для упрощения работы с несколькими базами данных. Как упоминалось выше для увеличения скорости работы приложения используется *NoSql* база данных.

Во время заполнения отчёта, все данные хранятся в *Sql* базе данных в нормализованном виде, однако, после того как данные о нагрузке будут распределены, они копируются в *NoSql* базу данных тем самым устраняя необходимость постоянного объединения таблиц для получения отчётов.

*REST* принципы также упрощают взаимодействие с программным продуктом. и являются общепризнанным стандартом при разработке *Api*.

Принцип единого интерфейса является конструктивной основой любого веб-сервиса *REST*. Это указывает на то, что сервер передаёт информацию в стандартном формате. Этот формат может отличаться от внутреннего представления ресурса в серверном приложении. Например, сервер может хранить данные в виде текста, но отправлять их в формате представления *HTML*. Единый интерфейс накладывает следующие архитектурных ограничения:

* + запросы должны идентифицировать ресурсы. Это происходит за счет единого идентификатора ресурсов;
  + клиенты имеют достаточно информации в представлении ресурса, чтобы при желании изменить или удалить ресурс, например, сервер выполняет это условие, отправляя метаданные, которые дополнительно описывают ресурс;
  + Клиенты получают информацию о дальнейшей обработке представлений. Сервер реализует это, отправляя описательные сообщения, где содержатся метаданные о том, как клиент может использовать их оптимальным образом.

Принцип отсутствия сохранения состояния в архитектуре *REST* относится к методу связи, при котором сервер выполняет каждый клиентский запрос независимо от всех предыдущих запросов. Клиенты могут запрашивать ресурсы в любом порядке, и каждый запрос либо изолирован от других запросов, либо его состояние не сохраняется. Это конструктивное ограничение *REST API* подразумевает, что сервер может каждый раз полностью понять и выполнить запрос.

Принцип кэширования состоит в том, что веб-службы *REST* поддерживают кэширование, то есть процесс сохранения некоторых ответов на клиенте или на посреднике для сокращения времени ответа сервера. Например, вы заходите на веб-сайт с общими изображениями верхнего и нижнего колонтитулов на каждой странице. Каждый раз, когда вы посещаете новую страницу веб-сайта, сервер должен повторно отправлять одни и те же изображения. Чтобы избежать этого, клиент кэширует или сохраняет эти изображения после первого ответа, а затем использует изображения из кэша.

Принцип предоставления кода по запросу. В архитектурном стиле *REST* серверы могут временно расширять или настраивать функциональные возможности клиента, передавая код программного обеспечения. Например, когда вы заполняете регистрационную форму на каком-либо веб-сайте, ваш браузер сразу же выделяет все допущенные ошибки, например, неверные номера телефонов. Это происходит благодаря коду, отправленному сервером [12].

Следованием данным принципам позволяет создать продукт, с которым можно взаимодействовать, используя различные языки программирования. Благодаря чёткому разделению компонентов появляется возможность создавать независимые компоненты, что существенно упрощает масштабирование проекта. Отсутствие необходимости хранения состояния позволяет уменьшить нагрузку при обработке запросов.

Так как в данном веб-приложении многие элементы, такие как заголовки таблицы и элементы меню, будут обновляться редко, в отличие от рабочей области с данными, из которых создаётся отчёт, то для увеличения производительности следует использовать подход с построением одностраничного приложения. Суть данного подхода состоит в отправке на компьютер пользователя не готовых *html-*страниц, а скриптов, которые выполняют генерацию отдельных модулей страницы непосредственно в браузере.

Данный подход позволит снизить не только интернет-трафик пользователя и время ожидания, но ещё уменьшит нагрузку на сервер.

**2.4 Моделирование базы данных**

Модель базы данных – описание содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных.

Для правильно функционирования программного комплекса необходимо, модель данных, которая бы адекватно отображала реалии той предметной области, для которой она разрабатывалась.

Модель базы данных, которая отображает информацию об учётных записях пользователей и их ролях представлена на рисунке 2.5:

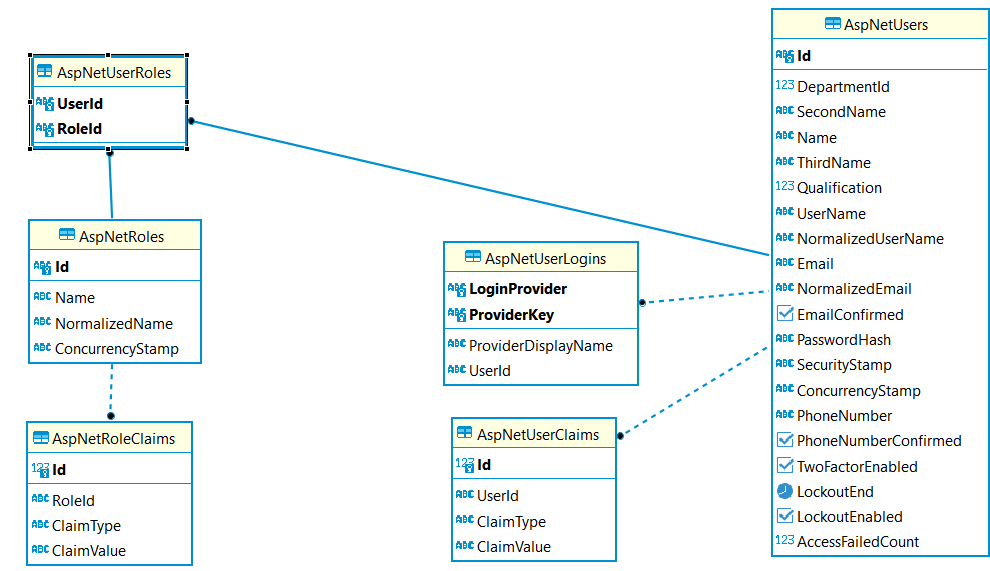


Рисунок 2.5 – Модель базы данных, отображающая информацию об учётных записях пользователей

Модель базы данных, которая хранит информацию о нагрузке на преподавателей изображена на рисунке 2.6.

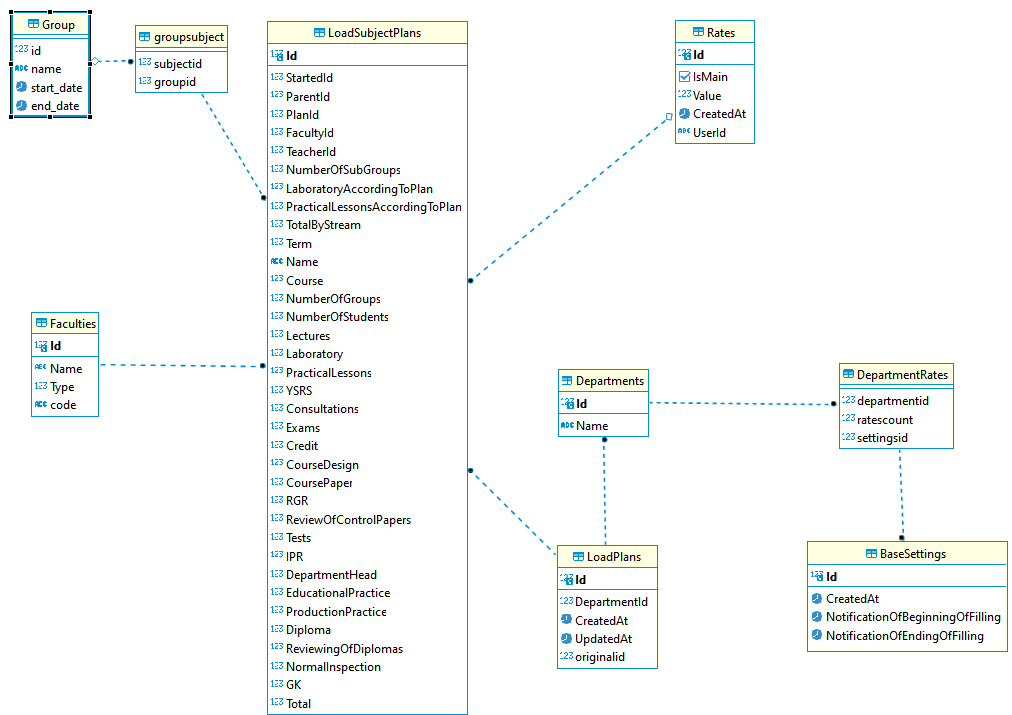


Рисунок 2.6 – базы данных, которая хранит информацию о нагрузке на преподавателей

Описание таблиц базы данных приведено в таблице 2.1

Таблица 2.1 – Описание таблиц базы данных

|  |  |
| --- | --- |
| Название таблицы | Описание |
| 1 | 2 |
| *AspNetUsers* | Хранит основную информацию о пользователе |
| *AspNetUserLogins* | Хранит данные необходимые для авторизации и аутентификации |
| *AspNetRoles* | Хранит список ролей пользователей |
| *AspNetUserRoles* | Промежуточная таблица, связывающая пользователей и их роли |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| 1 | 2 |
| *AspNetUserClaims* | Хранит дополнительные ограничения пользователей, например, их принадлежность той или иной кафедре |
| *AspNetRoleClaims* | Хранит ограничения, накладываемые ролями |
| *Group* | Хранит информацию об студенческих группах, например, ИП-41 |
| *GroupSubject* | Хранит информацию о связи группы и дисциплины |
| *Faculties* | Хранит информацию о факультетах |
| *LoadPlans* | Хранит информацию о нагрузочном плане |
| *Department* | Хранит информацию о кафедрах |
| *DepartmentRate* | Хранит информацию о допустимом количестве ставок для кафедры |
| *BaseSetting* | Хранит базовые настройки системы |
| *Rates* | Хранит информацию о рабочих ставках |

Описание атрибутов таблицы «*AspNetUserClaims*» находится в таблице 2.2.

Таблица 2.2 – Описание атрибутов таблицы «*AspNetUserClaims*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Идентификатор записи | Да | Да |
| *Userid* | *Nvarchar(128)* | Идентификатор пользователя | Нет | Да |
| *Claimtype* | *Nvarchar(max)* | Название  Ограничения | Нет | Нет |
| *Claimvalue* | *Nvarchar(max)* | Значение  Ограничения для пользователя | Нет | Нет |

Описание атрибутов таблицы «*AspNetRoles*» представлено в таблице 2.3.

Таблица 2.3 – Описание атрибутов таблицы «*AspNetRoles*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Nvarchar* | Идентификатор роли | Да | Да |
| *Name* | *Nvarchar(128)* | Название роли | Нет | Нет |
| *Normalaizedname* | *Nvarchar(128)* | Название роли в нормализированной форме | Нет | Нет |

Описание атрибутов таблицы «*AspNetUserRoles*» представлено в таблице 2.4.

Таблица 2.4 – Описание атрибутов таблицы «*AspNetUserRoles*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Userid* | *Nvarchar(128)* | Идентификатор пользователя | Да | Да |
| *Roleid* | *Nvarchar(128)* | Идентификатор роли | Нет | Да |

Описание атрибутов таблицы «*AspNetUserLogins*» дано в таблице 2.5.

Таблица 2.5 – Описание атрибутов таблицы «*AspNetUserLogins*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Loginprovider* | *Nvarchar(128)* | Название поставщика, предоставившего информацию о пользователе | Да | Да |
| *Providerkey* | *Nvarchar(128)* | Ключ поставщика | Нет | Нет |
| *Userid* | *Nvarchar(128)* | Идентификатор пользователя | Нет | Да |

Описание атрибутов таблицы «*AspNetUsers*» представлено в таблице 2.6.

Таблица 2.6 – Описание атрибутов таблицы «*AspNetUsers*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Nvarchar(128)* | Идентификатор пользователя | Да | Нет |
| *Email* | *Nvarchar(256)* | Адрес электронной почты пользователя | Нет | Нет |
| *Passwordhash* | *Nvarchar(MAX)* | Хеш код пароля пользователя | Нет | Нет |
| *Username* | *Nvarchar(256)* | Имя пользователя в системе | Нет | Нет |
| *Departmentid* | *Int?* | Идентификатор кафедры пользователя | Нет | Да |
| *Name* | *Nvarchar(128)* | Имя | Нет | Нет |
| *Secondname* | *Nvarchar(128)* | Фамилия | Нет | Нет |
| *Thirdname* | *Nvarchar(128)* | Отчество | Нет | Нет |
| *Qualification* | *Int?* | Учёная степень пользователя | Нет | Нет |

Описание атрибутов таблицы «*Group*» приведено в таблице 2.7.

Таблица 2.7 – Описание атрибутов таблицы «*Group*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| 1 | 2 | 3 | 4 | 5 |
| *Id* | *Int* | Идентификатор учебной группы | Да | Нет |
| *Full\_name* | *Nvarchar(128)* | Полное название группы | Нет | Нет |
| *Code* | *Nvarchar(128)* | Сокращённое название группы | Нет | Нет |
| *Startdate* | ***Date*** | Дата создания группы | Нет | Нет |
| *Enddate* | ***Date*** | Дата закрытия группы | Нет | Нет |

Продолжение таблицы 2.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *1* | ***2*** | 3 | 4 | 5 |
| *Numberofstudents* | ***Int*** | Количество  Студентов в группе | Нет | Нет |
| *Numberofsubgroups* | ***Int*** | Количество подгрупп | Нет | Нет |

Описание атрибутов таблицы «*Faculties*» приведено в таблице 2.8.

Таблица 2.8 – Описание атрибутов таблицы «*Faculties*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Идентификатор факультета | Да | Нет |
| *Code* | *Nvarchar(128)* | Сокращённое название | Нет | Да |
| *Name* | *Nvarchar(128)* | Название | Нет | Нет |

Описание атрибутов таблицы «*Departments*» приведено в таблице 2.9.

Таблица 2.9 – Описание атрибутов таблицы «*Departments*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Идентификатор кафедры | Да | Нет |
| *Name* | *Nvarchar(128)* | Название кафедры | Нет | Нет |

Описание атрибутов таблицы «*Rates*» приведено в таблице 2.10.

Таблица 2.10 – Описание атрибутов таблицы «*Rates*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| 1 | 2 | 3 | 4 | 5 |
| *Id* | *Int* | Идентификатор кафедры | Да | Нет |

Продолжение таблицы 2.10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *IsMain* | *Nvarchar(128)* | Название кафедры | Нет | Нет |
| *Value* | *Int* | Значение | Нет | Нет |
| *CreateAt* | *Date* | Дата создания | Нет | Нет |
| *UserId* | *Nvarchar(128)* | Идентификатор пользователя | Нет | Да |

Описание атрибутов таблицы «*GroupSubject*» приведено в таблице 2.11.

Таблица 2.11 – Описание атрибутов таблицы «*GroupSubject*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *SubjectId* | *Int* | Идентификатор дисциплины | Да | Да |
| *GroupId* | *Int* | Идентификатор группы | Да | Да |

Описание атрибутов таблицы «*LoadPlans*» приведено в таблице 2.12

Таблица 2.12 – Описание атрибутов таблицы «*LoadPlans*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Идентификатор  Плана | Да | Нет |
| *DepartmentId* | *Int* | Идентификатор кафедры пользователя | Нет | Да |
| *CreatedAt* | *Date* | Дата создания плана | Нет | Нет |
| *UpdatedAt* | *Date* | Дата последнего изменения | Нет | Нет |
| *OriginalId* | *Int* | Идентификатор  Составителя | Нет | Да |

Описание атрибутов таблицы «*LoadSbjectPlans*» приведено в таблице 2.13.

Таблица 2.13 – Описание атрибутов таблицы «*LoadSbjectPlans*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| 1 | 2 | 3 | 4 | 5 |
| *Id* | *Int* | Идентификатор  Дисциплины | Да | Нет |
| *Name* | *Nvarchar(128)* | Название дисциплины | Нет | Нет |
| *Parentid* | *Int?* | Идентификатор  Лекции, с которой связан данный предмет | Нет | Да |
| *Planid* | *Int* | Идентификатор  Учебного плана | Нет | Нет |
| *Facultyid* | *Double* | Идентификатор  Факультета | Нет | Нет |
| *Rateid* | *Double* | Идентификатор  Рабочей ставки | Нет | Нет |
| *Laboratoryaccordingtoplan* | *Double* | Часов для лабораторных запланировано | Нет | Нет |
| *Laboratory* | *Double* | Часов лабораторных | Нет | Нет |
| Practicallessonsaccordingtoplan | *Double* | Часов для практических запланировано | Нет | Нет |
| Practicallessons | *Double* | Часов практических | Нет | Нет |
| *Ysrs* | *Double* | Часов по управляемой самостоятельной работе студентов | Нет | Нет |
| *Consultations* | *Double* | Часов консультаций | Нет | Нет |
| *Exams* | *Double* | Часов экзаменов | Нет | Нет |

Продолжение таблицы 2.13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *1* | *2* | 3 | 4 | 5 |
| *Credit* | *Double* | Часов зачётов | Нет | Нет |
| *CoursePaper* | *Double* | Часов по написанию курсовых работ | Нет | Нет |
| *RGR* | *Double* | Часов по расчётно-графическим работам | Нет | Нет |
| *ReviewOfControlPapers* | *Double* | Часов по рецензированию контрольных работ | Нет | Нет |
| *Tests* | *Double* | Часов тестирования | Нет | Нет |
| *EducationalPractice* | *Double* | Часов производственной практики | Нет | Нет |
| *ProductionPractice* | *Double* | Часов учебной практики | Нет | Нет |
| *Diploma* | *Double* | Часов дипломного проектирования | Нет | Нет |
| *NormalInspection* | *Double* | Часов по наркоконтролю | Нет | Нет |
| *Total* | *Double* | Часов всего | Нет | Нет |

**3 СТРУКТУРА И АЛГОРИТМЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**3.1 Структура программного комплекса**

Общая структура программного комплекса, разработанная согласно архитектуре, описанной в предыдущем разделе приведена на рисунке 3.1.

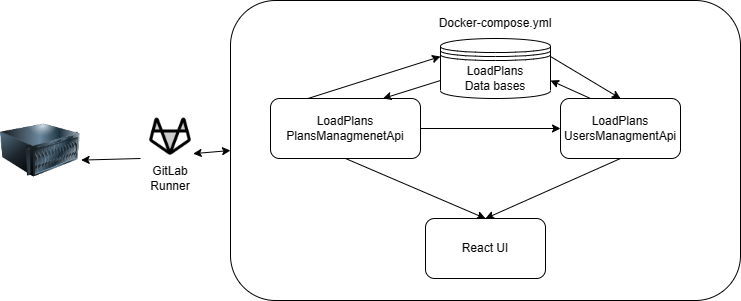


Рисунок 3.1 – Общая структура программного комплекса

Взаимодействие между проектами осуществляется по протоколу *https*, данный подход позволяет повысить безопасность в сравнении с обычным *http* благодаря шифрованию, передаваемой информации, это делает сложным перехват и чтение этих данных злоумышленниками.

**3.2 Серверная часть программного комплекса**

*UsersManagmentApi* – *Api*, который отвечает за операции над нагрузочными планами, генерацией отчётов, а также назначением преподавателей на те или иные дисциплины.

В основе данного проекта, как и писалось выше лежит применение паттерна *CQRS*, а также разбиение на несколько слоёв в зависимости от исполняемых задач.

Слой *DataAccess* – отвечает за взаимодействие с базой данных. На этом уровне происходит настройка класса контекста базы данных, а также определяются модели, по средствам которых и будет осуществляться взаимодействие.

Также на данном слое храниться классы-миграции базы данных. В данном проекте использовался подход *Code-First* суть которого состоит в создании моделей классов, а также настройки их связи в классе контексте базы данных.

После настройки класса контроллера, автоматически генерируются миграции, которые позволяют одной командой создать все необходимые таблицы в базе данных и настроить их связи и ограничения.

По мимо класса контекста базы данных также на этом слое содержатся *enum* типы, описывающие квалификацию пользователя и модель самого пользователя.

С со слоем *DataAccess* взаимодействует слой под названием *CQRS*. На этом слое определенны обработчики запроса, а также модели самих запросов. Каждый обработчик находится в своей директории.

Список обработчиков отвечающих за получение данных их описание приведены в таблице 3.1.

Таблица 3.1 – Обработчики запросов на получение данных и их описание

|  |  |
| --- | --- |
| Название | Задача |
| *GetAll* | Получение списка пользователей |
| *GetById* | Получение модели пользователя для изменения администратором по *Id* |
| *GetRateById* | Получение модели пользователя для изменения заведующим кафедрой по Id |
| *GetAllRates* | Получение списка всех ставок по кафедре |

Список обработчиков отвечающих за изменение данных их описание приведены в таблице 3.2.

Таблица 3.2 – Обработчики запросов на изменение данных и их описание

|  |  |
| --- | --- |
| Название | Задача |
| *Login* | Выполнение авторизации пользователя |
| *Registration* | Выполнение регистрации пользователя |
| *Update* | Изменение информации пользователя администратором |
| *UpdateUserRate* | Изменение информации о рабочих ставках пользователя заведующим кафедрой |
| *SynchronizeData* | Импортировать данные о пользователях |
| *SignOut* | Выполнить выход из учётной записи |

Каждый обработчик запроса связан с соответствующим эндпоинтом. Передача данных о запросе между эндпоинтом и обработчиком осуществляется с использованием паттерна «Медиатор». Суть данного потерна в использовании посредника, который отвечает за координацию взаимодействий между группой обработчиков и контроллером. Он избавляет входящие в группу объекты от необходимости явно ссылаться друг на друга, так как все объекты располагают информацией только о посреднике, что позволяет уменьшить количество взаимосвязей [13. c. 320].

Схема, отображающая работу данного паттерна изображена на рисунке 3.2.

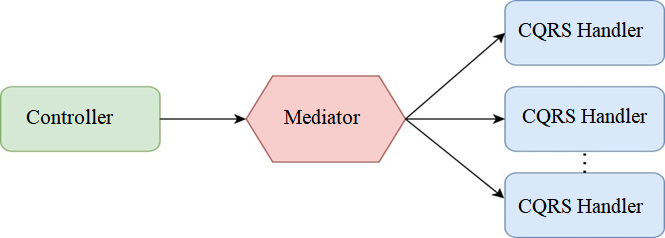


Рисунок 3.2 – Схема отображающая работу паттерна «Медиатор»

В для реализации данного паттерна была использована *nuget* библиотека «*MediatR*».

Для выполнения авторизации и аутентификации используются *JWT*-токены. Данный способ был выбран потому, что он позволяет единожды авторизоваться и использовать токен для общения между всеми сервисами, что важно так как всего в системе присутствует два отдельных *Api-*сервиса.

Обработка токенов осуществляется на соответствующем промежуточном слое обработки запросов *ASP.NET Core*, который внедряется в систему путём вызова методов расширения «*UseAuthentication*» и «*UseAuthorization*» у интерфейса «*IApplicationBuilder*».

Взаимодействия между объектами происходит за счёт использования интерфейсов и системы внедрения зависимостей, которая, используя механизмы рефлексии, подставляет инициализированные объекты, реализующие те или иные интерфейсы.

Указание классов, реализующих интерфейсы, происходит путём вызова методов расширений у интерфейса «*IServiceCollection*»:

* *AddTransient*, который создаёт новый объект сервиса при каждом обращении к сервису, а так как в течение одного запроса может быть несколько обращений к сервису то, соответственно, при каждом обращении будет создаваться новый объект;
* *AddScoped*, который создаёт один объект сервиса для каждого запроса, то есть если в во время обработки одного запроса есть несколько обращений к одному сервису, то при всех этих обращениях будет использоваться один и тот же объект;
* *AddSingleton*, который указывает, что объект сервиса будет создан единожды и при всех последующих вызовах будет использоваться именно этот объект.

Общее дерево всех файлов, входящих в проект изображено на рисунке 3.3.

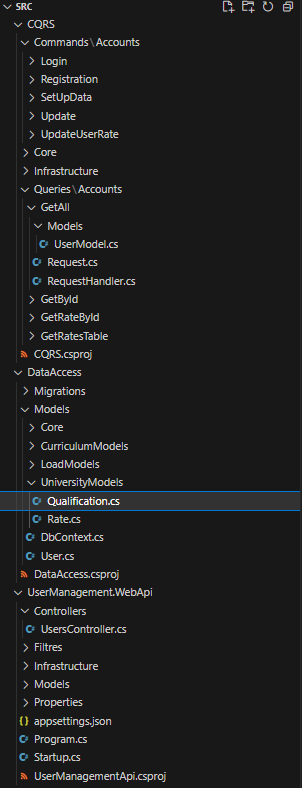


Рисунок 3.3 – Дерево файлов, входящих в проект *UsersManagmentApi*

*PlansManagmentApi* – часть программного комплекса, которая отвечает за получение информации о нагрузочных планах, её изменении и генерации отчётов.

Общее дерево всех файлов, входящих в *PlansManagmentApi* представлено на рисунке 3.4­.

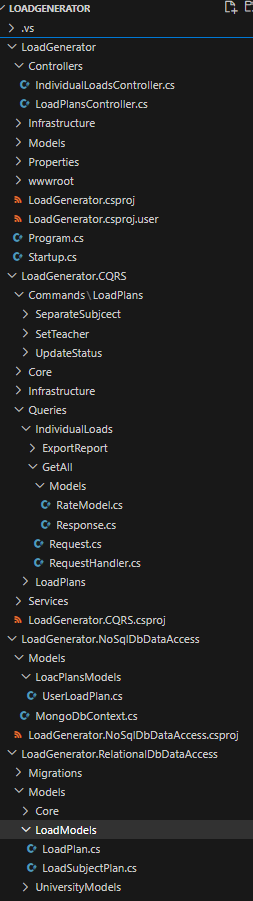


Рисунок 3.4 – Дерево файлов, входящих в проект *PlansManagmentApi*

Данный *Api* разделён на три отдельный логических слоя, однако два модуля занимают общий логический уровень. Схема взаимодействия модулей изображена на рисунке 3.5.

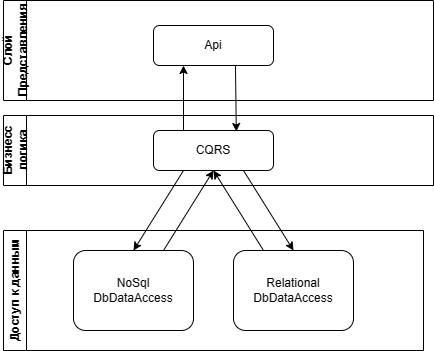


Рисунок 3.5 – Схема взаимодействия модулей в *PlansManagmentApi*

На слое *Api* определенно два контроллера:

* IndividualLoads *–* отвечает за получение информации о нагрузке на конкретных преподавателей, а также за импортданных о нагрузке;
* *LoadPlans –* отвечает за работу с общим планом нагрузки, назначением преподавателей на те или иные дисциплины.

На этом слое также происходит внедрение зависимостей по алгоритму схожему с таковым у *UsersManagmentApi.*

Стоит отметить, что разработка велась с учётом *SOLID* принципов. Для того, чтобы соблюсти принцип инверсии зависимости (*Dependency-Inversion Principle*) –модули верхнего уровня не должны зависеть от модулей нижнего уровня, и те и другие должны зависеть от абстракций, а абстракции не должны зависеть от деталей; детали должны зависеть от абстракций, на каждом слое были созданы классы, расширений, которые внедряли реализацию интерфейсов на своих уровнях.

Данный подход позволяет не концентрироваться на деталях реализации на более высоких логических уровнях, что способствует сохранению времени, ведь именно в модулях верхнего уровня инкапсулированы важные стратегические решения и бизнес-модели приложения. Эти модули и отличают одно приложение от другого. Но если они зависят от модулей нижнего уровня, то изменение последних может напрямую отразиться на модулях верхнего уровня и может стать причиной их изменения [14. с, 191].

На уровне бизнес-логики, в данном случае подпроекте *CQRS* также как и в предыдущем *Api* располагаются обработчики запросов. Стоит отметить, что в отличии от предыдущего *Api*, *PlansManagmentApi* содержит несколько контроллеров, поэтому обработчики сгруппированы согласно контроллерам, к которым они относятся.

Описание обработчиков *Quires* запросов на получение данных приведено в таблице 3.3.

Таблица 3.3 – Описание обработчиков *Quires* запросов в *PlansManagmentApi*

|  |  |  |
| --- | --- | --- |
| Название | Контроллер | Описание |
| 1 | 2 | 3 |
| *ExportReport* | *IndividualLoads* | Выполняет создание *XLSX* документа, хранящего информацию о нагрузке на преподавателя |
| *GetAll* | *IndividualLoads* | Возвращает список нагрузочных планов на преподавателей. Если заходит преподаватель – возвращаются только его нагрузочные планы, а если заходит заведующий кафедрой, то возвращается список нагрузочных планов по всем преподавателям кафедры и совместителям |
| *GetOriginal* | *LoadPlans* | Возвращает оригинал плана о распределении нагрузки по преподавателям в рамках кафедры |
| *GetAll* | *LoadPlans* | Возвращает список всех планов о распределении нагрузки по преподавателям в рамках кафедры |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| *GetById* | *LoadPlans* | Возвращает отредактированный план о распределении нагрузки по преподавателям в рамках кафедры |
| *GetTeacherRates* | *LoadPlans* | Возвращает список преподавателей, которых можно назначить на ту или иную дисциплину, например, только преподаватели с учёным званием выше, чем «преподаватель» могут быть назначенными на лекционные занятия |

Относительно *Command* запросов стоит сказать, что они относятся только к *LoadPlans* контроллеру. Описание обработчиков *Command* запросов на приведено в таблице 3.3.

Таблица 3.3 – Описание обработчиков *Command* запросов в *PlansManagmentApi*

|  |  |
| --- | --- |
| Название | Описание |
| *SeparateSubjcect* | Выполняет разделение практических занятий, в которых задействовано несколько подгрупп студентов, на отдельные занятия |
| *SetTeacher* | Выполняет назначение рабочей ставки преподавателя на ту или иную дисциплину |
| *UpdateStatus* | Помечает, что вся нагрузка распределена |
| *Sync* | Выполняет синхронизацию плана о распределении нагрузки по преподавателям в рамках кафедры за текущий и прошлый год, путём автоматического назначения на дисциплины, согласно плану прошлого года. Выполняется поиск по названиям дисциплин, групп и именам преподавателей. |

*RelationalDbDataAccess* – модуль, который отвечает за взаимодействие с базой данных *MySql.* В данном модуле определенны модели, которые описывают предметную область, относящуюся к нагрузочным планам:

* Group;
* GroupSubject;
* Faculties;
* LoadPlans;
* Department;
* DepartmentRate;
* BaseSetting;
* Rates.

Также в этом модуле определён класс контекста базы данных.

*NoSqlDbDataAccess –* модуль, который отвечает за взаимодействие с *MongoDb*. На данном уровне определён единственный класс-модель – *UserLoadPlan,* который хранит информацию нагрузке по конкретному преподавателю.

Класс *MongoDbContext* является реализацией паттерна «Фасад» над классом *MongoClient.* Сам класс наследуется on *MongoContext,* поставляемого в рамках *nuget* пакета *MongoFramework.*

Взаимодействие с документами в нём происходит по такому-же принципу, как и взаимодействие с таблицами в *DbContext.* Использование схожего синтаксиса упрощает разработку и способствует уменьшению времени разработки.

**3.3 Клиентская часть программного комплекса**

Для отображения пользовательского интерфейса, как описано выше, используется отдельное *React* приложение. В основе данного приложения лежит компонентный подход.

Базовая конфигурация страницы html элементов на странице определена в файле *Layout.js.* Данный файл содержит функцию, которая отвечает за рендер навигационного меню, и контейнера, в рамках которого и будут отображаться вложенные компоненты.

В файле *App.js* происходит настройка маршрутизации, то есть сопоставление Url адресов и компонентов, которые необходимо отобразить при переходе по этим адресам. Для этого используется стандартный компонент React приложений – *React-router.* Для упрощения процесса стилизации приложения использовался фреймворк *Bootstrap 5*, который предоставляет как набор классов для стилизации html-элементов, так и не посредственно компоненты *React*.

Вышеописанные элементы находятся в базовой директории проекта. Остальные элементы структурированы по директориям в зависимости от задач, которые они выполняют. Общее дерево компонентов проекта изображено на рисунке 3.6.

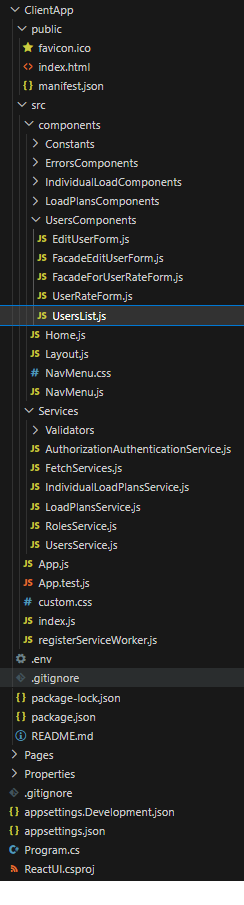


Рисунок 3.6 – Дерево компонентов *React* приложения

Для упрощения написания компонентов были выделены сервисы, содержащие статические методы для обращения к *Api*.

**4 Тестирование, верификация и валидация**

**4.1 Тестирование**

Тестирование программного кода — это процесс проверки программного кода с целью обнаружения ошибок, дефектов и недочётов, а также оценки его соответствия ожиданиям. Тестирование помогает убедиться, что программа работает правильно, корректно выполняет свои функции и взаимодействует с другими компонентами системы.

Для проверки функционирования всей системы необходимо протестировать, как отдельные модули, так и их взаимодействие друг с другом. Для этого применяются разные виды тестирования.

Для того, чтобы проверить правильность функционирования модулей применяются модульные тесты. Модульные тесты должны обладать рядом свойств, а именно: должен тестироваться отдельно взятый модуль, тесты должны быть быстрыми и изолированными друг от друга.

В большую часть работы в программном комплексе выполняют довольно сложные компоненты, взаимодействующие друг с другом, однако для валидации *command* запросов используются отдельные классы-валидаторы, которые и были протестированы в модульных тестах.

Для тестирования взаимодействия модулей были написаны интеграционные тесты, стоит отметить, что фреймворк *ASP.NET Core* позволяет запустить всю систему во время прохождения тестов, тем самым проверить обработку запросов.

Скриншоты листинга тестов приведены на рисунке 4.1.

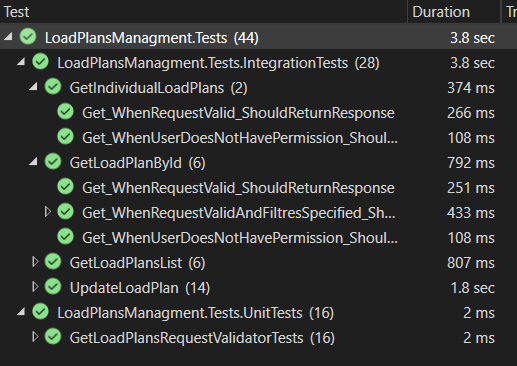
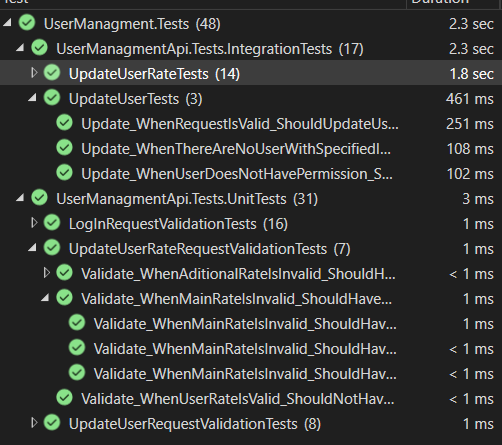


Рисунок 4.1 – Листинг модельных и интеграционных тестов

Так как разрабатываемым комплексом будет пользоваться не один человек, а персонал университета, то крайне важно произвести нагрузочное тестирование. Данное тестирование позволит проверить, сможет ли справиться приложение с поставленной задачей в реальных условиях, а также поможет выяснить требования к системе.

Для тестирования использовался инструмент *apache jmeter*. Тестовая конфигурация приложения изображена на рисунке 4.2.

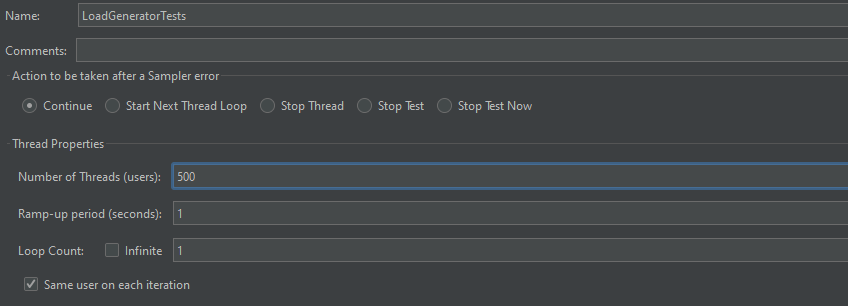


Рисунок 4.2 – Конфигурация *apache jmeter*

Тестировались следующие страницы:

* экспорт плана нагрузки преподавателя
* получение нагрузочного плана кафедры по id
* список пользователей для изменения
* список планов нагрузки по преподавателям

Результат нагрузочного тестирования представлен на рисунке 4.3.

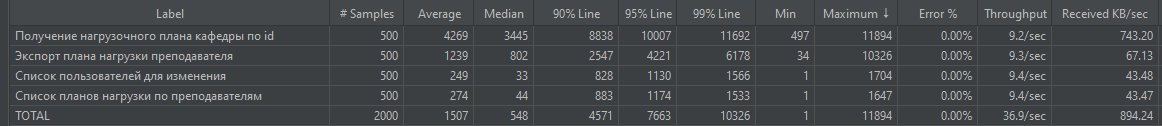


Рисунок 4.3 – Результат нагрузочного тестирования

Таким образом, даже при получении большого количества запросов, приложение продолжает стабильно работать, хотя время ожидания получения запросов увеличивается и в обычной ситуации. Таким образом можно сделать вывод, что для университета сопоставимого с ГГТУ имени П. О. Сухого хватит конфигурации c 8 гигабайтами оперативной памяти, 1 жёсткого диска со скоростью вращения 7200 оборотов в минуту и процессора I5-8300.

**4.2 Верификация**

Для верификации программного комплекса был выбран функциональный способ.

При входе на начальную страницу приложения пользователь видит краткое описание приложения в нижнем левом углу, справа располагается логотип приложения. В навигационном меню для неавторизированного пользователя отображаются две кнопки: «Вход». Начальная страница изображена на рисунке 4.4.

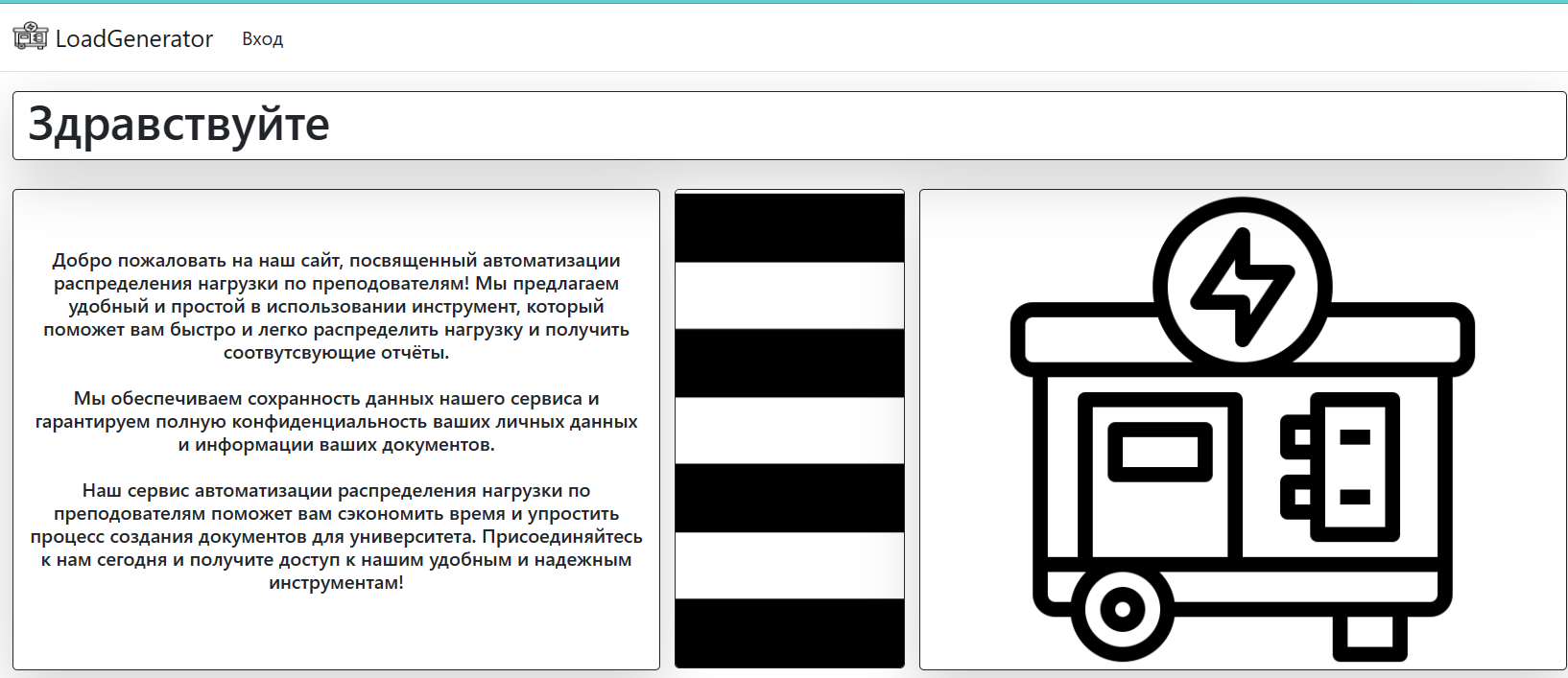


Рисунок 4.4 – Начальная страница приложения

После нажатия на кнопку «Вход» пользователь перемещается на страницу входа в учётную запись. На данной странице присутствуют поля для ввода пароля и электронной почты пользователя. Страница регистрации изображена на рисунке 4.5.

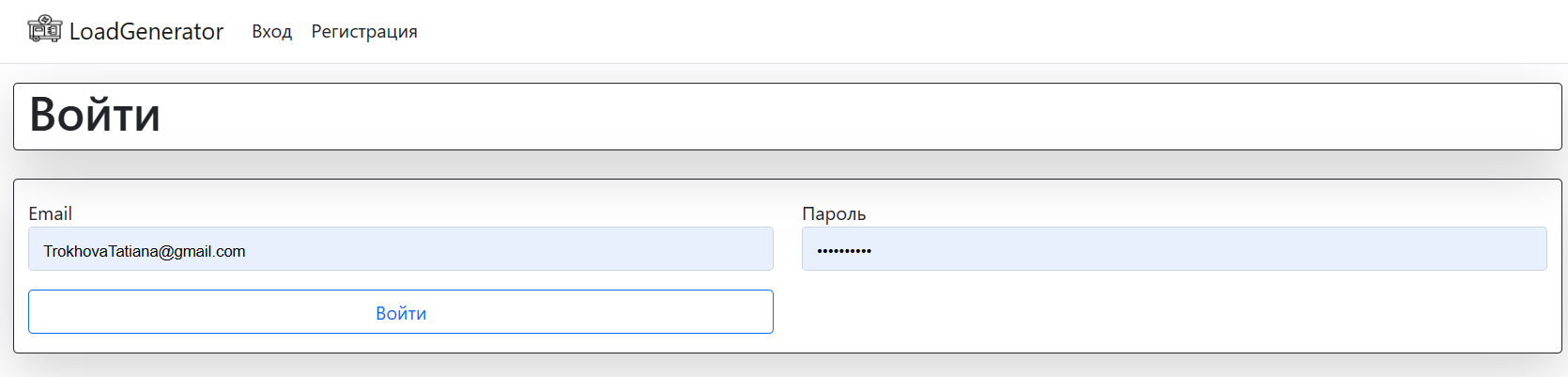


Рисунок 4.5 – Страница регистрации пользователя

В случае ввода не верных данных пользователю отображается сообщение об ошибке. Пример сообщения об ошибке в случае, если не существует учётной записи с таким логином и паролем отображён на рисунке 4.6.

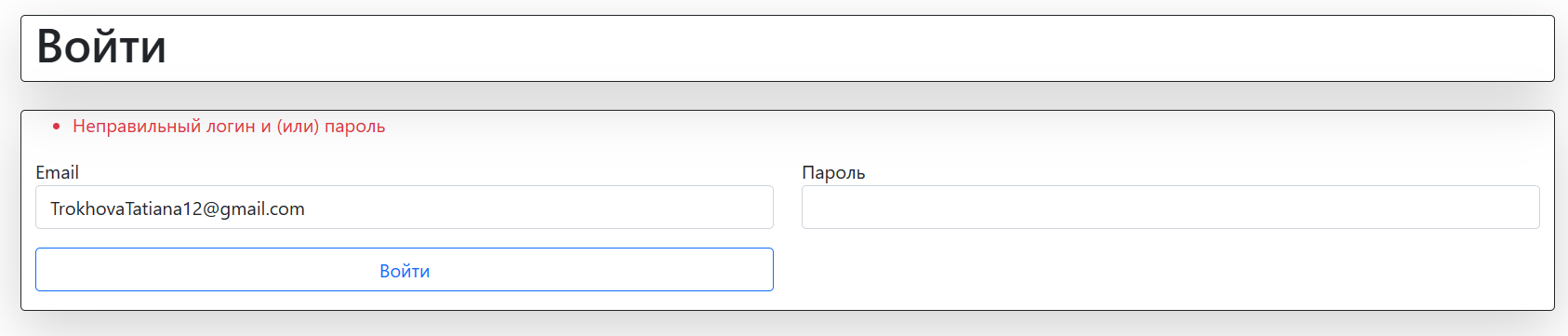


Рисунок 4.6 – Пример отображения сообщения об ошибке

После осуществления входа в учётную запись, обладающую ролью «Заведующий кафедрой», пользователю отображается обновлённое навигационное меню, которое включает такие пункты, как «Нагрузочные планы», «Нагрузка пользователей – статистика», «Нагрузка пользователей – планы», «Пользователи», «Выйти». Навигационное меню после входа в учётную запись, обладающую ролью «Заведующий кафедрой» изображено на рисунке 4.7.

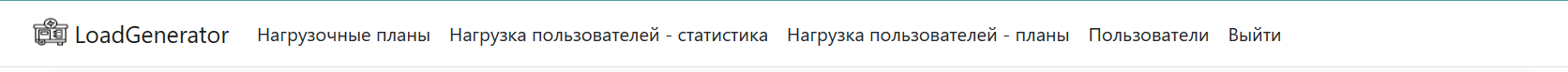


Рисунок 4.7 – Навигационное меню, для учётной записи, обладающей ролью «Заведующий кафедрой»

При нажатии кнопки «Нагрузочные планы», пользователю отображается список нагрузочных планов, относящихся к данной кафедре. Страница с нагрузочными планами изображена на рисунке 4.8.

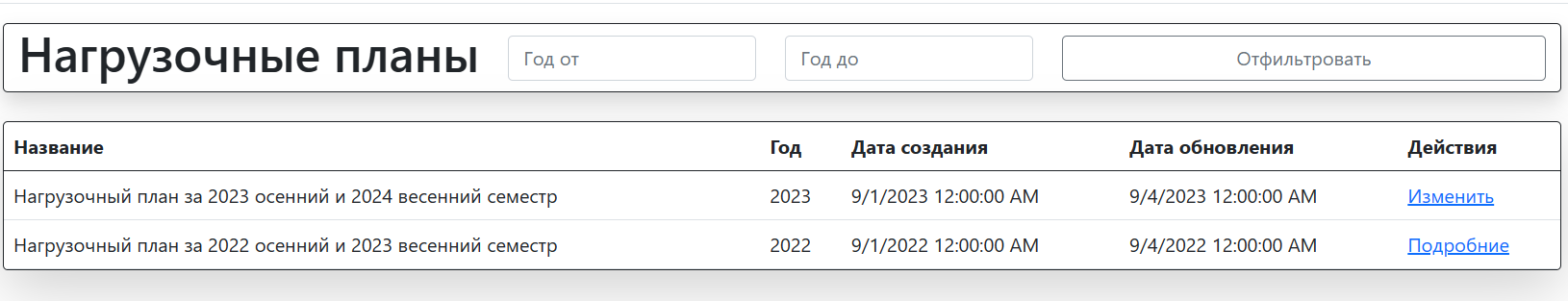


Рисунок 4.8 – Страница с нагрузочными планами

В зависимости от состояния нагрузочного плана напротив него отображаются кнопки. «Изменить» – если план ещё не закончен. «Подробнее» – если план помечен как распределенный. Также присутствует возможность отфильтровать данные за определённый год. Пример страницы, с отфильтрованными данными, за 2023 год изображён на рисунке 4.9.

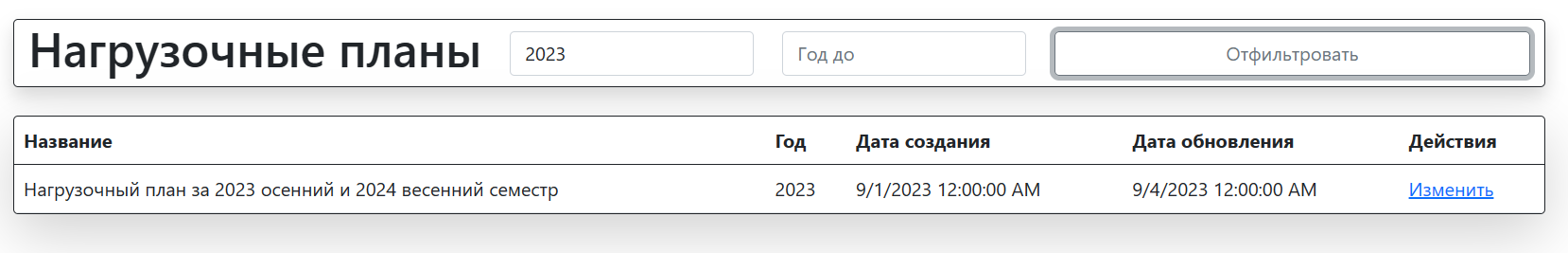


Рисунок 4.9 – Нагрузочный план за 2023 год

При нажатии на кнопку изменить, пользователь попадает на страницу нагрузочного плана. На данной странице присутствует заголовок, с тремя кнопками: «Фильтры», «Оригинал», «Изменить статус».

При нажатии на кнопку «Фильтры», пользователю отображается панель с фильтрами, при повторном нажатии на кнопку, панель скрывается. Всего присутствует 4 фильтра:

* по названию факультета;
* по названию предмета;
* по статусу предмета – назначен ли преподаватель на данный предмет или нет;
* по имени преподавателя.

При фильтрации добавлении фильтрации, под таблицей отображается блок, содержащий информацию о общем количестве часов по дисциплинам, согласно фильтрации.

Важной особенностью является то, что при отображении в виде таблицы, каждая строка имеет разные цвет. Если на какую-то дисциплину назначен преподаватель, то данная строка отображается зелёным цветом. Если преподаватель не назначен, и строка не поддерживает разбиение, то строка отображается жёлтым цветом, а если строка поддерживает разбиение, и преподаватель не назначен, то она отображается красным цветом.

При нажатии на кнопку «Оригинал», пользователь переходит на страницу, на которой отображается не изменённая версия плана распределения нагрузки. А все строки в данной таблице имеет белый фон. Также на данной странице отсутствует кнопка изменения статуса плана, однако присутствует кнопка «В работе», по нажатию которой пользователь возвращается на предыдущую страницу.

Верхняя часть страницы нагрузочного плана с отображением панели фильтрации изображена на рисунке 4.10.

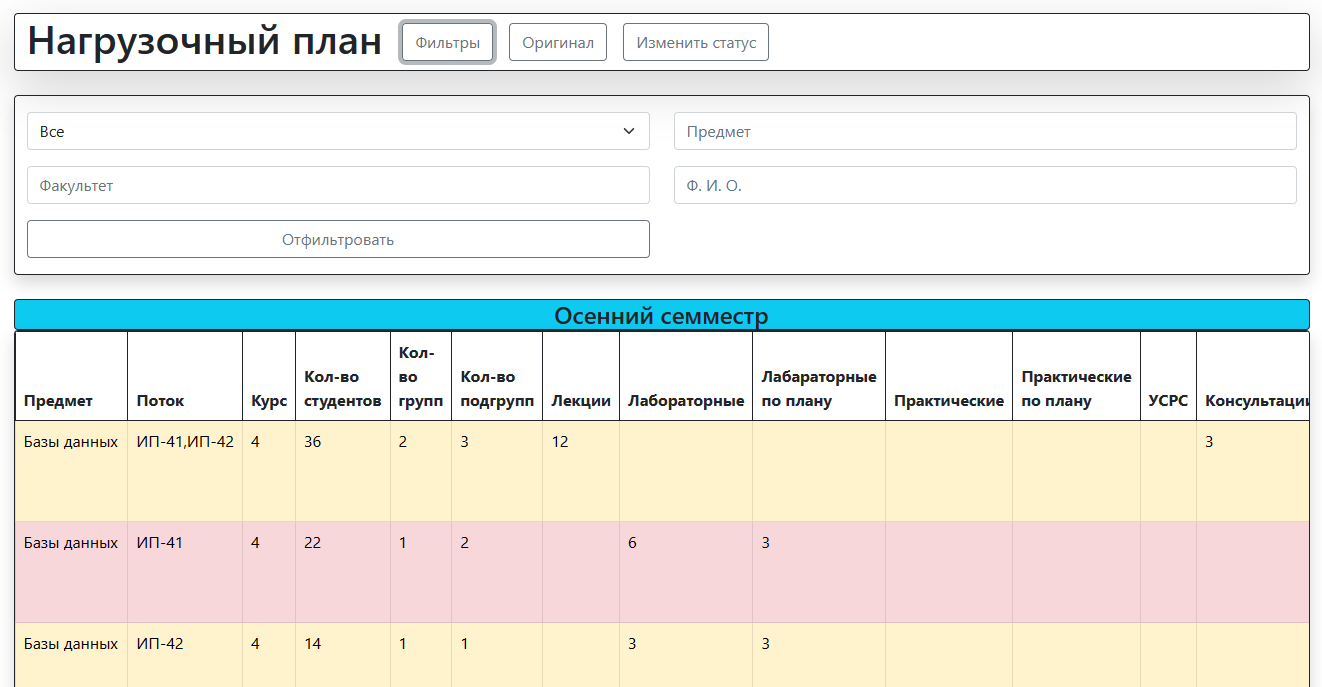


Рисунок 4.10 – Верхняя часть страницы нагрузочного плана с отображением панели фильтрации

Преподаватель назначается на ту или иную дисциплину через выпадающее меню. Данное меню, находится в конце строки, в колонке «Преподаватель». В меню отображаются инициалы преподавателя и объём его ставки. Само выпадающее меню также поддерживает поиск по имени преподавателя. Меню выбора преподавателя изображено на рисунке 4.11.

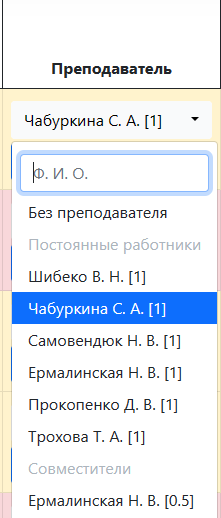


Рисунок 4.11 – Меню выбора преподавателя

Стоит отметить, то преподаватели в данном меню разделены по группам:

* постоянные работники;
* совместители.

Таблица предметов за осенний семестр изображена на рисунке 4.12.



Рисунок 4.12 – Талица предметов за осенний семестр

Нижняя часть страницы нагрузочного плана с отображением панели подведения итогов изображена на рисунке 4.13.

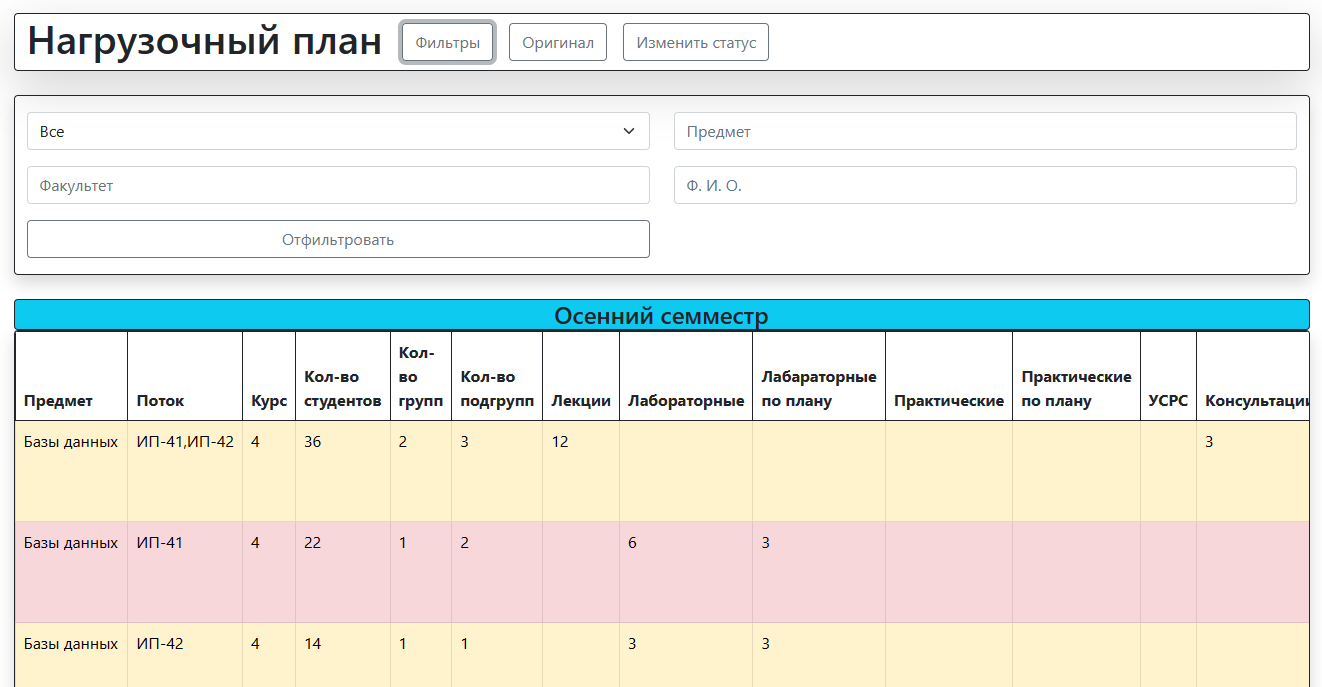


Рисунок 4.13 – Нижняя часть страницы нагрузочного плана с отображением панели подведения итогов

Изображение страницы с оригиналом документа приведено на рисунке 4.14.

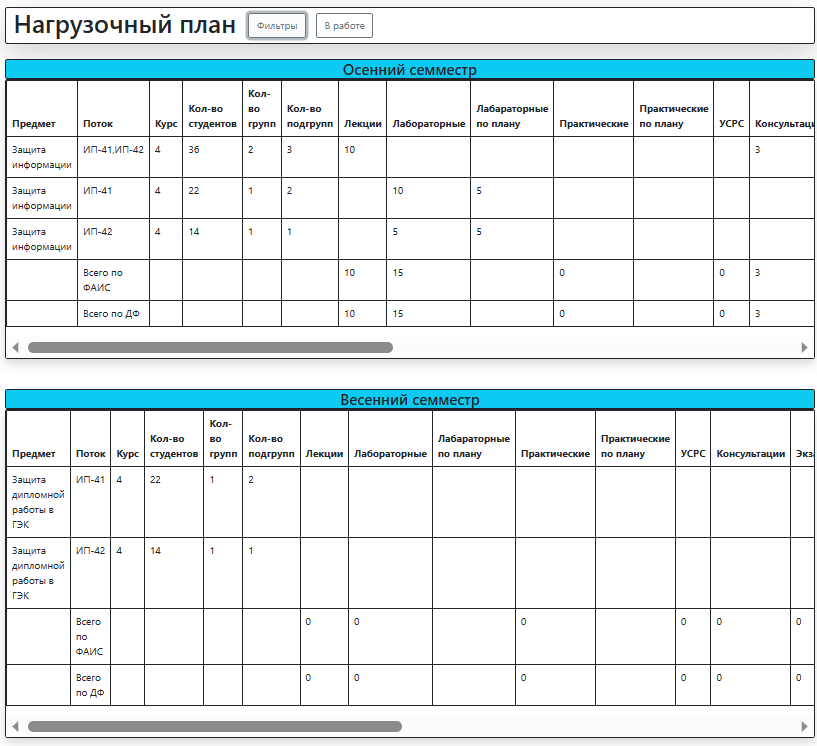


Рисунок 4.14 – Страница с оригиналом документа

При нажатии на кнопку «Нагрузка пользователей – статистика» пользователю отображается страница с таблицей, содержащей информации о количестве часов, распределённых на пользователя. Страница со статистикой о нагрузке на преподавателей изображена на рисунке 4.15.



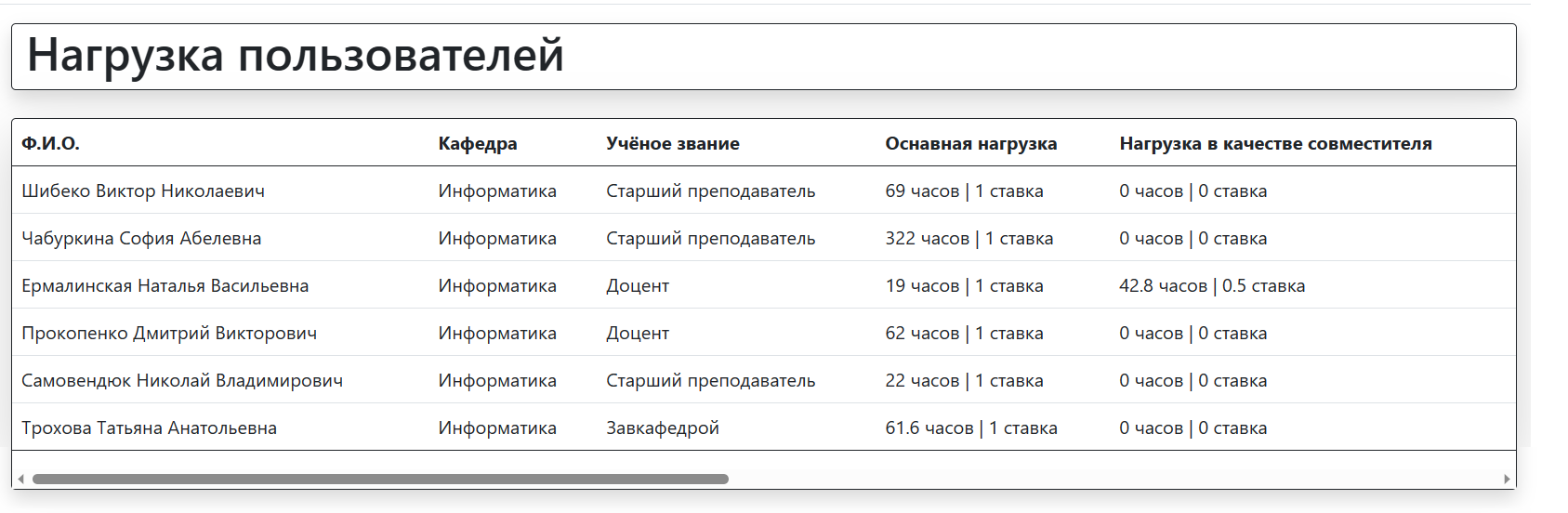


Рисунок 4.15 – Страница со статистикой о нагрузке на преподавателей

При нажатии на кнопку «Нагрузка преподавателей – планы» пользователь переходит на страницу скачивания индивидуальных нагрузочных планов на преподавателя. В случае если заходи заведующий кафедрой, то ему отображается список планов на всех работников кафедры. Страница, отображающая нагрузочные планы пользователей изображена на рисунке 4.16.

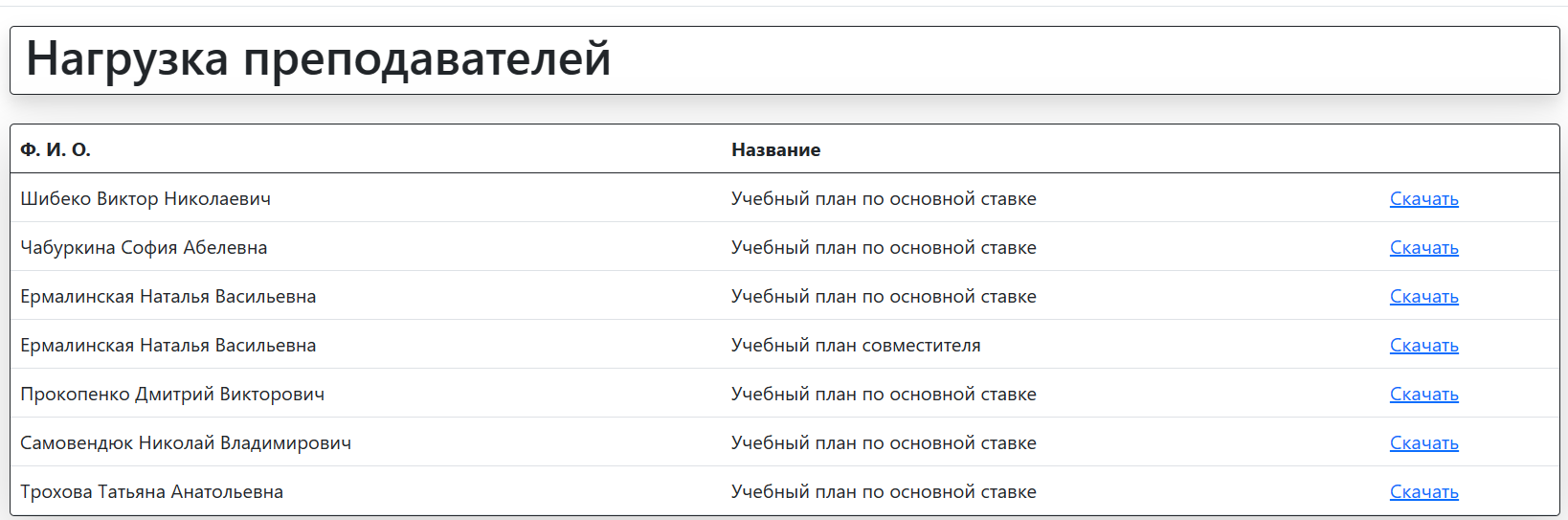


Рисунок 4.16 – Страница, отображающая нагрузочные планы пользователей

При нажатии на кнопку «Скачать» происходит скачивание отчёта о нагрузке на преподавателя в виде файла с *XLSX* форматом. Содержимое данного файла изображено на рисунке 4.17.

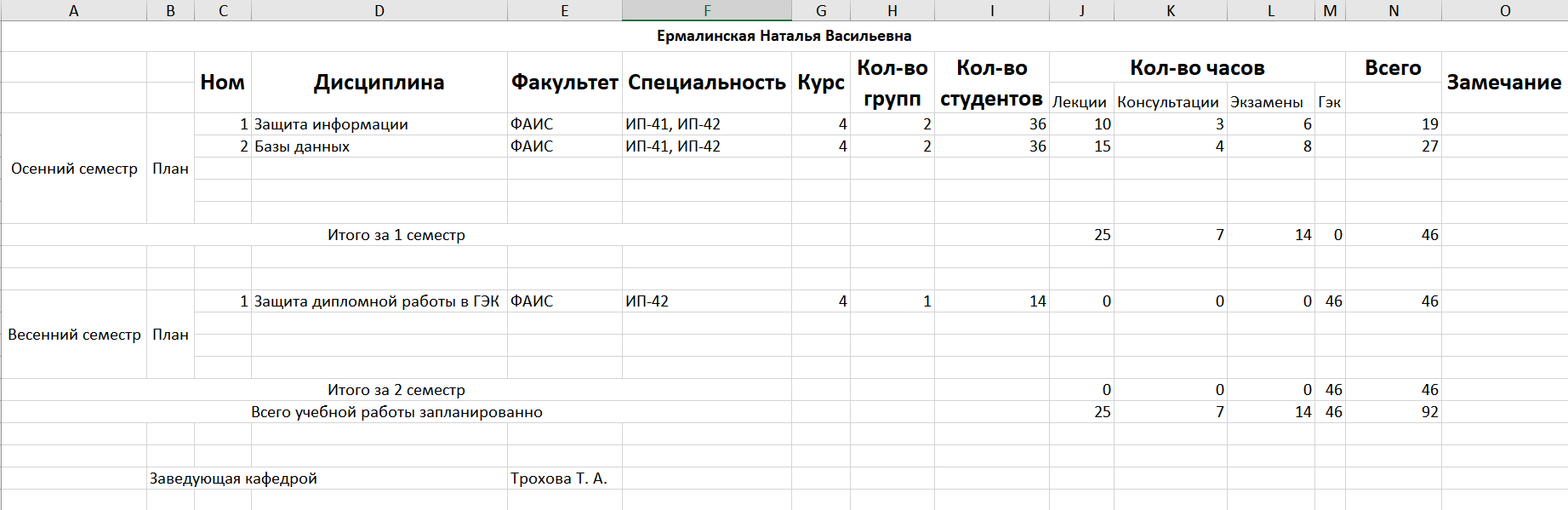


Рисунок 4.17 – Содержимое файла-отчёта о нагрузке на преподавателя

При нажатии на кнопку «Пользователи» из навигационного меню, пользователю отображается список всех работников, которых он может назначить на учебные дисциплины в рамках своей кафедры.  
 Напротив каждого пользователя отображается кнопка «Редактировать», а также отображается заголовок с двумя параметрами доступными для фильтрации по имени пользователя и по его кафедре.

Страница, отображающая список пользователей изображена на рисунке 4.18

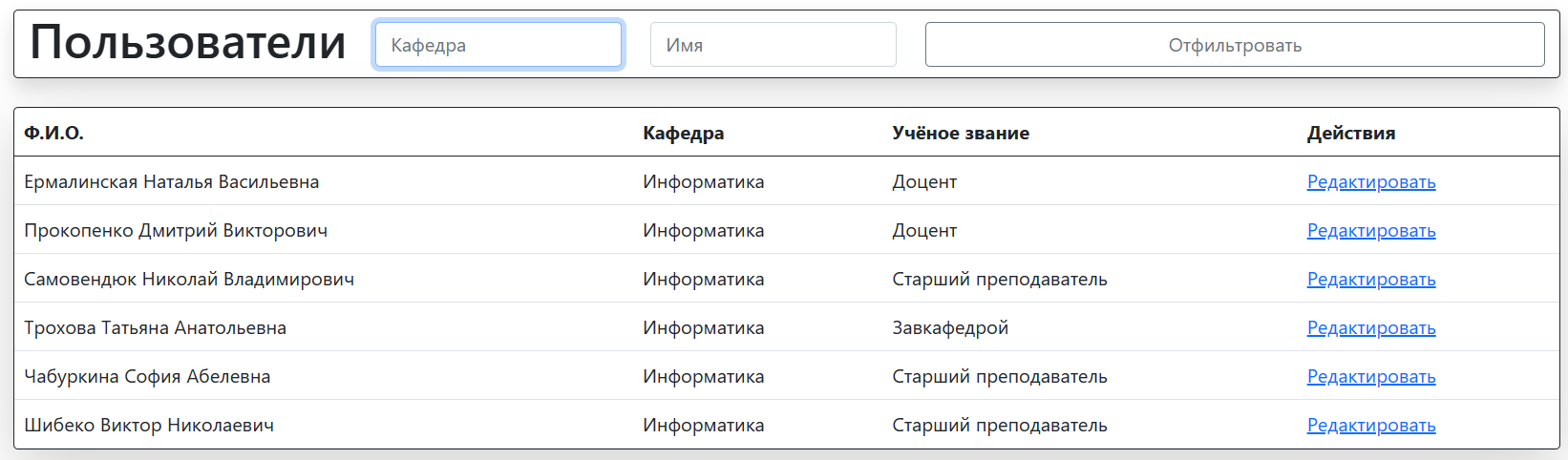


Рисунок 4.18 - Страница, отображающая список пользователей

При нажатии на кнопку «Редактировать», пользователь переходит на страницу редактирования информации о ставках работника. Причём, если работник относится к кафедре данного заведующего, то для редактирования доступно два значения: «ставка в качестве совместителя», «основная ставка».

Если же работник относится к другой кафедре, или вообще не имеет кафедры, то тогда у него можно настроить только поле «ставка в качестве совместителя». Страница, на которой отображён процесс редактирования ставок работника, с кафедры заведующего изображена на рисунке 4.19.

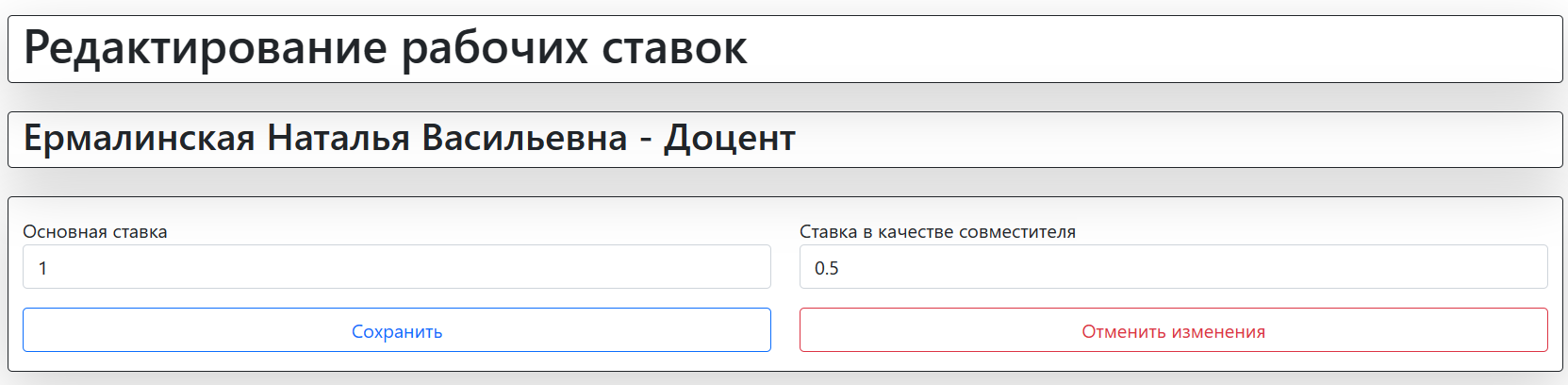


Рисунок 4.19 – Страница редактирования ставок работника, с кафедры заведующего

Страница, на которой отображён процесс редактирования ставок работника, с чужой кафедры изображена на рисунке 4.20.

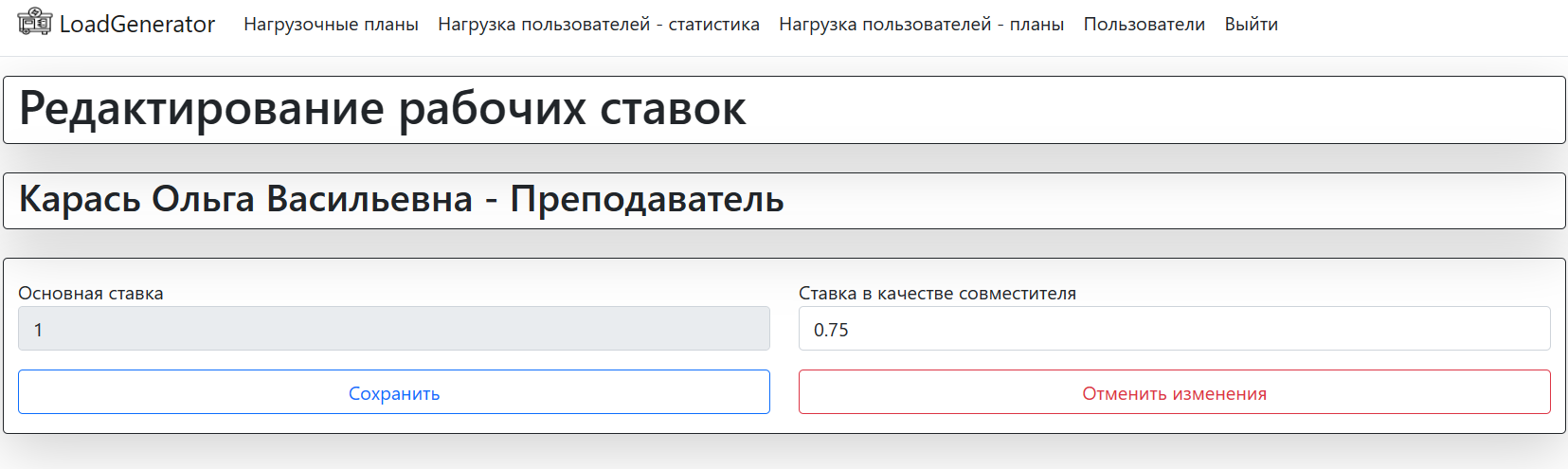


Рисунок 4.20 – Страница редактирования ставок работника, с чужой кафедры

Допустимыми значениями являются только 0, 0.25, 0.5, 0.75, 1 ставка.

В случае если введены неверные данные, то отображается соответствующее сообщение об ошибке около поля ввода. Изображение сообщения об ошибке при редактировании информации о ставках работника приведено на рисунке 4.21.

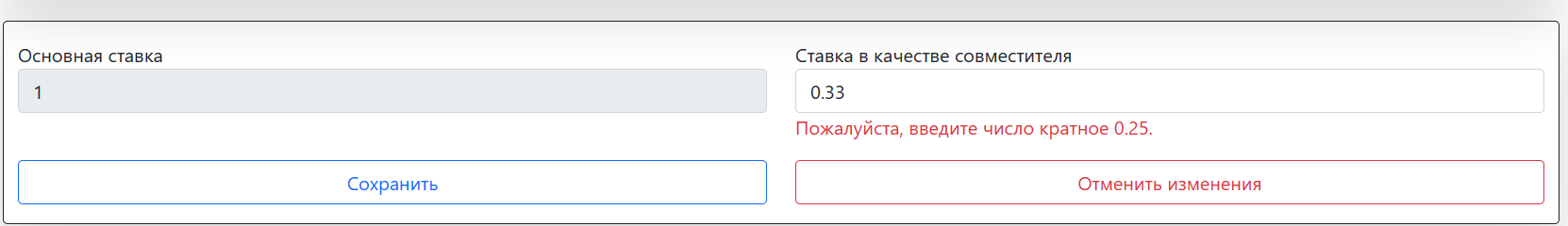


Рисунок 4.21 – Сообщение об ошибке при редактировании информации о ставках работника

При входе в учётную запись, которая обладает правами администратора, пользователю отображается навигационное меню с тремя кнопками: «Пользователи», «Выйти», «Настройки».

Навигационное меню пользователя «Администратор» изображено на рисунке 4.22.



Рисунок 4.22 – Навигационное меню пользователя «Администратор»

При нажатии на кнопку «Пользователи» осуществляется переход на страницу с интерфейсом схожим с таковым на рисунке 4.18, однако при нажатии кнопки «Редактировать» открывается страница редактирования информации о пользователе, а не его рабочих ставках.

На данной странице есть поля для редактирования информации об имени, фамилии, отчестве, электронной почте, учёном звании. Также можно сменить роли пользователя. В нижнем краю присутствует две кнопки: «Сохранить изменения» и «Отменить изменения».

Страница редактирования информации пользователя изображена на рисунке 4.23.

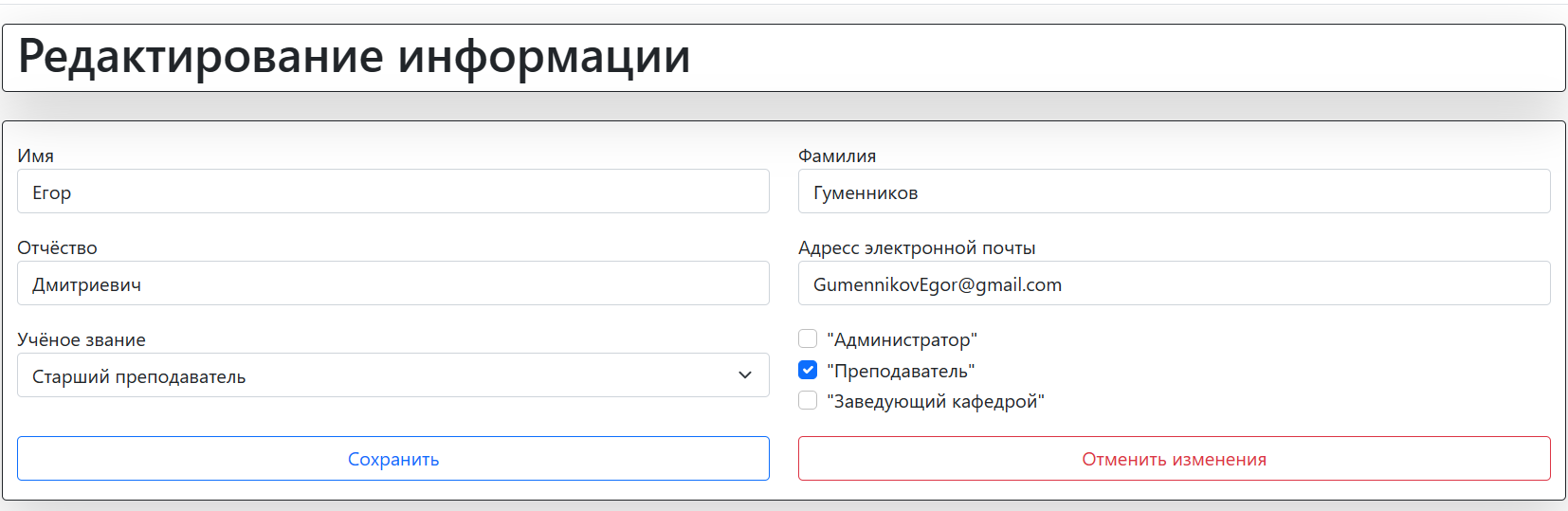


Рисунок 4.23 – Редактирование информации пользователя

При входе в учётную запись, которая обладает правами преподавателя, пользователю отображается навигационное меню с двумя кнопками: «Нагрузочные планы», «Выйти». Навигационное меню пользователя «Преподаватель» изображено на рисунке 4.24.



Рисунок 4.24 – Навигационное меню пользователя «Преподаватель»

При нажатии на кнопку «Нагрузочные планы» пользователь попадает страницу аналогичную той, что изображена на рисунке 4.16, однако на ней отображаются только нагрузочные планы данного пользователя.

**6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ**

**6.1 Охрана труда в Республике Беларусь**

Охрана труда — это система обеспечения безопасности жизни и здоровья, работающих в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-гигиенические, лечебно-профилактические, реабилитационные и иные мероприятия и средства [15. c, 6].

Основными законодательными актами, регулирующими охрану труда в Республике Беларусь, являются:

* закон Республики Беларусь «О труде», который устанавливает общие принципы и требования по охране труда, права и обязанности работодателей и работников;
* закон Республики Беларусь «О безопасности и охране труда», который определяет нормативные требования и основные принципы охраны труда, включая обязательность разработки инструкций по охране труда, проведение обучения и медицинских осмотров, а также регулирование производственных проверок и контроля;
* нормативные акты Министерства труда и социальной защиты Республики Беларусь, которые разрабатываются для более детального регулирования конкретных аспектов охраны труда, таких как требования к организации рабочих мест, предотвращению профессиональных заболеваний;
* государственные стандарты и нормы безопасности труда, которые определяют конкретные требования и рекомендации по организации безопасных и здоровых условий труда в различных отраслях и профессиях.

Важной составляющей охраны труда в Беларуси является обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний, которое обеспечивает компенсации для пострадавших работников.

**6.2 Система управления охраной труда на предприятии**

Охрана труда является одним из основных приоритетов внутренней политики в Республике Беларусь, так как позволяет обеспечить право каждого человека на безопасные и здоровые условия труда. Для эффективной реализации этого права необходимо разрешать соответствующие правовые, социально-экономические, организационные, технические, санитарно-гигиенические и другие вопросы в области управления охраной труда.

Управление охраной труда представляет собой процесс воздействия на систему «человек – машина – производственная среда», с целью достижения установленных значений, характеризующих состояние условий работы.

Для решения различных задач в области охраны труда активное участие принимают руководители предприятий, структурные подразделения, функциональные службы, отделы охраны труда и профсоюзные комитеты. Практически все работники, начиная от главных руководителей до рабочих, являются вовлечёнными в эту работу.

Организация деятельности администрации и служб предприятия по осуществлению комплекса мер для повышения уровня охраны труда осуществляется через систему управления охраной труда.

Система управления охраной труда является одной из подсистем, входящих в систему управления предприятием. Для успешного функционирования данной системы, как и любой другой, необходимо чётко определить задачи и функции, иметь стабильную структуру связей.

Главную роль в системе управления охраной труда играет руководитель предприятия и главный инженер, если таковой имеется; в подразделениях и цехах данную роль играют руководители соответствующих подразделений и служб.

СУОТ включает следующие задачи:

* составление плана работ по охране труда;
* контроль за соблюдением законодательства об охране труда;
* определение опасностей и рисков;
* финансирование мероприятий по охране труда.

Составление плана работ по охране труда – процесс определения задач для подразделений, служб и должностных лиц предприятия, участвующих в организации охраны труда.

План мероприятий по охране труда может является как частью коллективного договора трудящихся, так и самостоятельным документом.

Планы, обычно, разделяются на три вида:

* перспективные – составляются на срок 3-5 лет;
* текущие – составляются на год;
* оперативные – составляется на квартал или месяц.

Задачи, которые могут рассматриваются в планах:

* сокращение рабочих мест, несоответствующих нормам охраны труда;
* вывод из эксплуатации объектов, не подлежащих ремонту или реконструкции и не обеспечивающих соблюдение норм охраны труда;
* предоставление работникам санитарно-бытовых помещений согласно установленным нормам;
* сокращение тяжёлых физических работ и работников, занятых ручным трудом.

Исходными данными для составления планов являются:

* паспорта санитарно-технического состояния условий труда в цехах и структурных подразделениях;
* мероприятия, предусмотренные документами специального расследования несчастных случаев с тяжелыми последствиями и актами по форме Н-1;
* приказы нанимателя по вопросам охраны труда;
* мероприятия, предусмотренные коллективным договором;
* журналы административно-общественного контроля за состоянием охраны труда;
* предписания органов государственного надзора и контроля, службы охраны труда предприятия;
* представления профсоюзов и предложения работников по улучшению условий труда;
* информация о передовом опыте работы по созданию здоровых и безопасных условий труда [16. c, 36].

К данным, которые необходимо указывать в плане, относятся: названия мероприятий, сроки их выполнений, ответственные лица, источники финансирования и ожидаемые улучшения.

Мероприятия, которые не требующие существенных затрат, а также мероприятия, улучшающие условия труда большому числу работников включаются в план в обязательном порядке.

Мероприятия, внесённые в план, являются обязательными для исполнения. В случае их неисполнения, ответственные лица могут понести наказание в соответствии с законодательством РБ.

Важной задачей является финансовое и материальное обеспечение мероприятий, отражённых в плане.

В организациях, подчинённых Правительству Республики Беларусь, согласно постановлению Совета Министров Республики Беларусь, финансирование мероприятий по улучшению условий и охраны труда осуществляется за счёт инновационных фондов. Местным распорядительным и исполнительным органам рекомендовано предусматривать финансирование подобных задач за счёт местных бюджетов.

Также организация может осуществлять финансирование путём:

* использования средств, расходы по которым учитываются в себестоимости продукции, если эти мероприятия имеют некапитальный характер и тесно связаны с участием работников в производственном процессе;
* использования средств из амортизационного фонда, если мероприятия проводятся одновременно с капитальным ремонтом основных средств;
* получения банковского кредита, если мероприятия входят в комплекс затрат, которые банк готов кредитовать для внедрения новой техники или расширения производства;
* инвестирования в основной капитал, включая фонд накопления, если мероприятия имеют капитальный характер.

Помимо этого, согласно действующему законодательству, наниматель также имеет право принимать решения о финансировании мероприятий по охране труда из других источников в установленном порядке.

Материальные и финансовые ресурсы, предназначенные для осуществления мероприятий по охране труда, не могут расходоваться для достижения других целей. В случае, если в ходе проведения мероприятия, средства, выделенные на их проведение, были израсходованы не в полном объеме, оставшиеся ресурсы должны быть направлены нанимателем на проведение дополнительных мероприятий по охране труда, кроме организаций, финансируемых из государственного бюджета.

Стоит отметить, что на данный момент в Республике Беларусь действует стандарт 18001-2009 «Системы управления охраной труда. Требования» устанавливающий требования к системе управления охраной труда с целью оказания помощи организациям в управлении рисками и повышении результативности данного управления.

# **ЗАКЛЮЧЕНИЕ**

Целью, данной преддипломной практики являлось проектирование программного комплекса для создания *web*-приложения по автоматизации распределения нагрузки по преподавателям.

В процессе прохождения технологической(производственной) практики был спроектирован программный комплекс, полностью отвечающий функциональным требованиям. Пользовательский интерфейс был спроектирован с использованием языка *JavaScript* и фреймворка *React*, что позволило уменьшить нагрузку на серверную часть и увеличить скорость взаимодействия с продуктом.

Серверная часть была выполнена с спроектирована с использованием языка программирования *C#*. Основные фреймворки, используемые для разработки это *Asp Net Core, EntityFrameworkCore*.

Для хранения данных были выбраны *MySqlServer* и *MongoDb*.

Серверная часть имеет в своей основе микросервисную архитектуру, а уровень представления построен в соответствии с паттерном *SinglePage Application*.

Практическая польза данного комплекса заключается в том, что она может быть использован для автоматизации распределения нагрузки по преподавателям, что позволит разгрузить руководителей кафедры.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 10 плюсов от внедрения 1С Документооборот. – Электрон. Данные. – Режим доступа: <https://gomel.1cbit.by/blog/10-plyusov-ot-vnedreniya-1s-dokumentooborot/>. – Дата доступа 15.04.2023

2 Справочники (1c.ru). – Электрон. Данные. – Режим доступа: https://v8.1c.ru/platforma/spravochniki/– Дата доступа 15.04.2023

3 Прайс М. Д. C# 7 и .Net Core. кросс-платформенная разработка для профессионалов / М. Д. Прайс; под ред. С. В. Черников. – 3-е изд., – Санкт-Петербург: Питер, 2019. – 636 с.

4 Албахари Д. C# 9.0. Справочник. Полное описание языка / Д. Албахари; под ред. Ю.Н. Артеменко. – Чехов: ООО «Диалектика», 2021. – 1056 с.

5 Прайс М. Д. C# 10 и .NET 6. Современная кросс-платформенная разработка. / М. Д. Прайс; под ред. С. В. Черников. – Санкт-Петербург: Питер, 2023. – 848 с.

6 Что такое Entity Framework?. – Электрон. Данные. – Режим доступа: https://www.entityframeworktutorial.net/what-is-entityframework.aspx. – Дата доступа 15.04.2023

7 Компонент C# / VB.NET Excel - GemBox.Spreadsheet. – Электрон. Данные. – https://www.gemboxsoftware.com/spreadsheet. – Дата доступа 15.04.2023

8 Что такое документная база данных?. – Электрон. Данные. – https://aws.amazon.com/ru/nosql/document/. – Дата доступа 15.04.2023

9 Бибо Б. jQury. Подробное руководство по продвинутому JavaScript / Б. Бибо, И. Кац; под общ. ред. А. С. Галунов. – 2-е изд., – Санкт-Петербург: Символ-Плюс, 2017. – 612 с.

10 Что такое контейнеризация? – Описание контейнеризации – AWS (amazon.com). – Электрон. Данные. – https://aws.amazon.com/ru/what-is/containerization/. – Дата доступа 15.04.2023

11 Ньюмен С. Создание микросервисов / С. Ньюмен; под ред. С. В. Черников. – 2-е изд., – Санкт-Петербург: Питер, 2023. – 624 с.

12 Что такое RestFull Api. – Электрон. Данные. – https://aws.amazon.com/ru/what-is/restful-api/ – Дата доступа 15.04.2023

13 Гамма Э. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Д. М. Влиссидес; под ред. К. А. Тульцева. – 2-е изд., – Санкт-Петербург: Символ-Плюс, 2020. — 448.

14 Сессил Р. М. Принципы, паттерны и методики гибкой разработки на языке C# / Р. М. Сессил; под общ. ред. А. С. Галунов. – 2-е изд., – Санкт-Петербург: Символ-Плюс, 2011. – 768 с.