

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
Гомельский государственный технический университет имени
П.О. Сухого

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

специальность 1-40 04 01 «Информатика и технологии программирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**к дипломной работе
на тему**

«Программный комплекс автоматизации обслуживания жилого фонда студенческого общежития»

Разработал студент гр. ИП-42	_____	<u>Пархоменко П.Л.</u>
	(подпись)	(Ф.И.О.)
Руководитель работы	_____	<u>ст. преподаватель Шибeko В.Н.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Консультант по экономической части	_____	<u>доцент, к.э.н. Соловьева Л.Л.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Консультант по охране труда и технике безопасности	_____	<u>профессор, д.т.н., Кудин В.П.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Нормоконтроль	_____	<u>Самовендюк Н.В.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Рецензент	_____	_____
	(подпись)	(ученое звание, ученая степень, должность, организация, Ф.И.О.)

Дипломная работа (_____ с.) допущена к защите
в Государственной экзаменационной комиссии.

Зав. кафедрой	_____	<u>доцент, к.т.н, Трохова Т.А.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)

Гомель 2023

Лист задания

Реферат

ПРОГРАММНЫЙ КОМПЛЕКС АВТОМАТИЗАЦИИ ОБСЛУЖИВАНИЯ ЖИЛОГО ФОНДА СТУДЕНЧЕСКОГО ОБЩЕЖИТИЯ: дипломная работа / П. Л. Пархоменко. – Гомель : ГГТУ им. П.О. Сухого, 2023. – Дипломная работа: 82 страницы, 38 рисунков, 35 таблиц, 13 источников, 2 приложения.

Ключевые слова: студент, студенческое общежитие, управление, комендант, отчетность, оптимизация процессов.

Объектом разработки является программный продукт, направленный на автоматизацию обслуживания жилого фонда студенческого общежития.

Целью дипломной работы является создание программного комплекса автоматизации жилого фонда студенческого общежития. Данный программный комплекс будет предназначен для упрощения и автоматизации процессов, связанных с проживанием студентов и поддержанием работы всего жилого фонда общежития.

В процессе работы было сделано следующее: выполнен анализ существующего программного обеспечения по автоматизации студенческих общежитий, создана база данных для хранения и обработки учетных записей и личных данных пользователей, а также новостей, событий, заявок; произведен экономический анализ и обоснована рентабельность разработки; проведен анализ внедрения разработки с учетом аспектов энерго- и ресурсосбережения.

Областью практического использования является применение этого программного обеспечения в студенческих общежитиях.

Студент-дипломник подтверждает, что дипломная работа выполнена самостоятельно, приведенный в дипломной работе материал объективно отражает состояние разрабатываемого объекта, пояснительная записка проверена в системе «Антиплагиат» «АО "Антиплагиат"» (режим доступа: <https://www.antiplagiat.ru/>). Процент оригинальности составляет 78.73%. Все заимствованные из литературных и других источников, теоретические и методологические положения и концепции сопровождаются ссылками на источники, указанные в «Списке использованных источников».

Резюме

Тема дипломной работы «Программный комплекс автоматизации обслуживания жилого фонда студенческого общежития».

В процессе выполнения данной дипломной работы был разработан программный продукт, который предназначен для упрощения и автоматизации процессов, связанных с проживанием студентов и поддержанием работы всего жилого фонда общежития.

Объектом разработки является программный продукт, направленный на автоматизацию обслуживания жилого фонда студенческого общежития.

Поставленная задача была выполнена в полном объеме.

Рэзюме

Тэма дыпломнай працы «Праграмны комплекс аўтаматызацыі абслугоўвання жылога фонду студэнцкага інтэрната».

У працэсе выканання гэтай дыпломнай работы быў распрацаваны праграмны прадукт, які прызначаны для спрашчэння і аўтаматызацыі працэсаў, звязаных з пражываннем студэнтаў і падтрыманнем работы ўсяго жылога фонду інтэрната.

Аб'ектам распрацоўкі з'яўляецца праграмны прадукт, накіраваны на аўтаматызацыю абслугоўвання жылога фонду студэнцкага інтэрната.

Пастаўленая задача была выканана ў поўным аб'ёме.

Summary

The theme of the thesis is «Software complex for automation of maintenance of the housing stock of a student hostel».

In the process of performing this thesis work, a software product was developed that is designed to simplify and automate the processes associated with student accommodation and maintaining the work of the entire housing stock of the hostel.

The object of development is a software product aimed at automating the maintenance of the housing stock of a student hostel.

The assigned task has been completed in full.

СОДЕРЖАНИЕ

Введение.....	7
1 Аналитический обзор существующих методов и средств автоматизации жилого фонда студенческого общежития	8
1.1 Обзор существующих систем автоматизации.....	8
1.2 Анализ разработки и проектирования веб приложений.....	9
1.3 Анализ используемых технологий для реализации поставленной задачи.....	10
1.4 Анализ инструментальных средств автоматизации разработки и тестирования.....	13
1.5 Техническое задание для клиент-серверного программного продукта «Программный комплекс автоматизации обслуживания жилого фонда студенческого общежития»	17
2 Анализ предметной области и алгоритмы.....	18
2.1 Анализ предметной области	18
2.2 Функциональная модель программного комплекса.....	18
2.3 Информационная модель программного комплекса.....	21
3 Программная реализация комплекса.....	27
3.1 Архитектура приложения.....	27
3.2 Уровень представления клиентской части приложения.....	28
3.3 Уровень бизнес-логики приложения	40
3.4 Уровень данных в приложении	45
4 Тестирование, верификация и валидация программного комплекса	47
4.1 Тестирование пользовательских форм	47
4.2 Модульное тестирование бизнес-логики.....	50
4.3 Интеграционное тестирование бизнес-логики.....	55
5 Экономическое обоснование дипломной работы	59
5.1 Техничко-экономическое обоснование целесообразности разработки программного продукта.....	59
5.2 Расчёт общей трудоемкости разработки программного обеспечения	59
5.3 Расчёт объёма капитальных вложений при создании программного продукта	62
5.4 Расчёт текущих затрат разработки программного продукта.....	62
5.5 Расчёт договорной цены разрабатываемого программного продукта	68
5.6 Определение экономической эффективности разработки программного продукта.....	70
6 Охрана труда и техника безопасности. психофизиологические и эргономические основы охраны труда	73
6.1 Охрана труда. Цели и принципы	73
6.2 Психологические аспекты охраны труда	74
6.3 Физиологические аспекты охраны труда	75
6.4 Эргономические аспекты охраны труда	77
7 Ресурсо- и энергосбережение при использовании программного продукта ...	79
Заключение	81

Список использованных источников	82
Приложение А Листинг программы.....	83
Приложение Б Результаты расчета экономического обоснования	108

ВВЕДЕНИЕ

Автоматизация процессов позволяет увеличить эффективность работы, сократить время и затраты на выполнение задач, снизить вероятность ошибок и улучшить качество услуг. Также автоматизация помогает оптимизировать использование ресурсов, повысить скорость реакции на изменения внешней среды и упростить процессы для пользователя.

В настоящее время, среди аналогов разрабатываемого программного продукта можно выделить различные интернет-страницы общежитий. Однако, эти аналоги часто ограничиваются предоставлением общей информации об общежитии, не уделяя должного внимания взаимодействию между воспитателем, комендантом и студентом.

Целью дипломной работы является создание программного комплекса автоматизации жилого фонда студенческого общежития. Данный программный комплекс будет предназначен для автоматизации процессов, связанных с проживанием студентов и поддержанием работы всего жилого комплекса.

Задачами дипломной работы являются:

- изучение методик разработки клиент-серверных приложений на базе стека *MERN*;
- классификация ролей пользователей и их ролевые политики;
- изучение методов реализации серверной части для приложения;
- проектирование структуры приложения, базы данных для хранения информации, формирование пользовательских правил для доступа к ресурсам и функциям приложения;
- разработка программных модулей, обеспечивающих авторизацию и аутентификацию пользователей; работу с данными с помощью графического интерфейса;
- верификация и опытная эксплуатация разработанного программного обеспечения.

В результате разработки данного программного комплекса ожидается значительное улучшение эффективности управления жилым фондом студенческого общежития, сокращение времени и затрат, а также повышение уровня сервиса для студентов.

1 АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ АВТОМАТИЗАЦИИ ЖИЛОГО ФОНДА СТУДЕНЧЕСКОГО ОБЩЕЖИТИЯ

1.1 Обзор существующих систем автоматизации

Многие университеты имеют свои личные жилые фонды (общежития), но не многие пытаются их автоматизировать, ведь в них протекают много различных и сложных процессов. Например, управление жилплощадью, распределение мест, контроль доступа, управление уборкой и ремонтом, а также учет платежей за проживание и услуги – все эти процессы могут быть оптимизированы и автоматизированы с помощью специальных программных решений. Такая автоматизация помогает университетам повысить эффективность управления общежитиями, снизить затраты и улучшить качество обслуживания студентов.

В большинстве случаев университеты предоставляют общую информацию об общежитии, сформированную в виде одной онлайн страницы. Например, на сайте университета БГПУ приведены следующие данные:

- адреса общежитий;
- фамилия, имя и отчество заведующего;
- фамилии, имена и отчества воспитателей и педагогов-организаторов;
- общее описание здания (количество мест, год постройки);
- наличие в общежитии общественных пространств (теннисные и тренажерные залы, актовый зал, кафе, прачечная, комната самоподготовки).

На сайте «Белорусского государственного университета» имеется информация, выраженная в виде рекомендаций и фотографий, например:

- какие бытовые приборы лучше всего взять с собой;
- как распланировать бюджет родителям.

Также существуют другие средства автоматизация жилого фонда студенческого общежития. Эти системы включают в себя различные программные и аппаратные компоненты, такие как системы безопасности, системы управления доступом, системы учета и бронирования проживающих, системы мониторинга и управления энергопотреблением и т.д. Такие системы могут быть полезны для упрощения управления общежитием и улучшения комфорта проживания студентов. Например, системы управления доступом позволяют управлять доступом в здание и в отдельные комнаты, что обеспечивает безопасность и предотвращает несанкционированный доступ. Системы мониторинга и управления энергопотреблением позволяют управлять энергоресурсами и экономить деньги на коммунальных услугах.

Кроме того, системы учета и бронирования проживающих позволяют упростить процесс бронирования мест в общежитии и учета проживающих, что может значительно сократить время, затрачиваемое на административные процессы.

В целом, автоматизация студенческих общежитий может быть очень полезна для улучшения условий проживания студентов и оптимизации управления общежитием.

1.2 Анализ разработки и проектирования веб приложений

Разработка и проектирование веб-приложений – это процесс создания программного обеспечения, которое работает на сервере и обеспечивает взаимодействие с пользователем через веб-браузер. Веб-приложения могут быть использованы для различных целей, включая электронную коммерцию, социальные сети, онлайн-банкинг, различные средства автоматизации и т.д.

Согласно [1], для создания веб-приложения нужно выполнить несколько шагов. Сначала определяются требования к приложению, которые могут включать в себя функциональные и нефункциональные требования, дизайн интерфейса, спецификации базы данных и т.д. Затем происходит проектирование архитектуры приложения, разработка базы данных и программного кода, а также тестирование и развертывание приложения на сервер.

Веб-приложение – это программное обеспечение, которое работает через веб-браузер и взаимодействует с пользователем посредством веб-интерфейса. В общем случае, веб-приложение состоит из клиентской и серверной части, базы данных, *API* и дополнительных компонентов.

Клиентская часть (*frontend*) – это часть веб-приложения, которая выполняется в браузере пользователя и отображает интерфейс. Она написана на языке *HTML*, *CSS* и *JavaScript*, и обеспечивает взаимодействие пользователя с приложением.

Серверная часть (*backend*) – это часть веб-приложения, которая работает на сервере и обрабатывает запросы пользователя, взаимодействует с базой данных и обеспечивает доступ к ресурсам и функциям приложения. Серверная часть может быть написана на различных языках программирования, таких как *PHP*, *Python*, *Java*, *Ruby* и т.д.

База данных – это хранилище данных, которые используются в приложении. База данных может быть реляционной или нереляционной и хранить информацию о пользователях, продуктах, заказах, и т.д.

API (*Application Programming Interface*) – это интерфейс, который позволяет взаимодействовать между клиентской и серверной частями приложения. *API* может использовать различные протоколы, такие как *HTTP* или *WebSocket*, и форматы данных, такие как *JSON* или *XML*.

Дополнительные компоненты – веб-приложение может содержать и другие компоненты, такие как библиотеки, плагины, сервисы сторонних разработчиков и т.д.

Компоненты веб-приложения взаимодействуют между собой и обеспечивают полноценное функционирование приложения. Они позволяют пользователям работать с приложением, выполнять различные действия и получать результаты их работы.

Разработка веб-приложений может включать в себя использование различных языков программирования, фреймворков, библиотек, систем управления базами данных и других инструментов. Например, для создания веб-приложения может использоваться язык программирования *JavaScript*, фреймворк *React*, база данных *MySQL* и сервер *Apache*.

Разработка и проектирование веб-приложений требует определенных навыков и знаний в области программирования, веб-технологий, дизайна пользовательского интерфейса и баз данных. Для успешной разработки веб-приложений необходимо понимание требований к приложению и его пользователей, умение разрабатывать эффективную архитектуру, использовать соответствующие инструменты и библиотеки, а также тестировать и развертывать приложение на сервер. Важным аспектом является также поддержка и обновление приложения после его запуска, а также обеспечение безопасности при обработке пользовательских данных и защите от взломов.

1.3 Анализ используемых технологий для реализации поставленной задачи

Веб-приложения могут быть написаны на разных языках программирования и фреймворках, но одним из наиболее популярных стеков технологий для разработки веб-приложений является стек *MERN*. *MERN* – это аббревиатура, состоящая из четырех популярных инструментов для веб-разработки: *MongoDB*, *Express*, *React* и *Node.js*.

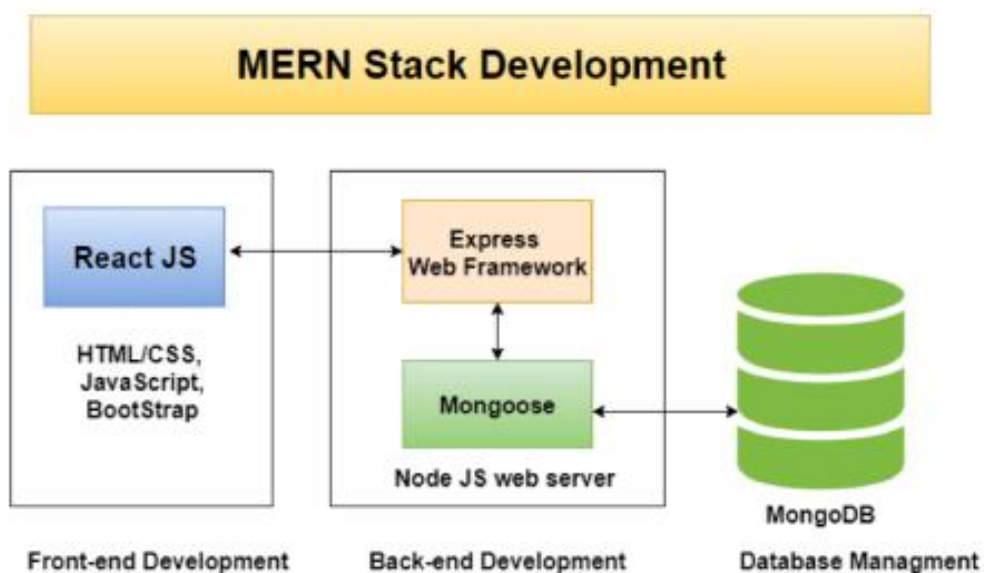


Рисунок 1.1 – Компоненты стека *MERN*

В основе этих фреймворков и библиотек находится язык программирования *JavaScript*.

JavaScript – мультипарадигменный (с одновременным использованием множества парадигм) язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации *ECMAScript* (стандарт *ECMA-262*). Его обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты:

- динамическая типизация;
- слабая типизация;
- автоматическое управление памятью;
- прототипное программирование;
- функции как объекты первого класса.

JavaScript включает в себя объектную модель браузера – браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера – управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом *window*, центральным объектом *DOM*. Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке *WHATWG* и *W3C*.

В *JavaScript* используется в *AJAX*, популярном подходе к построению интерактивных пользовательских интерфейсов веб-приложений, заключающемся в «фоновом» асинхронном обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе (без применения *AJAX*) [2].

MongoDB – это документо-ориентированная *NoSQL* база данных, которая используется для хранения, управления и обработки больших объемов данных. Она позволяет хранить данные в формате документов *BSON (Binary JSON)*, которые имеют структуру, аналогичную формату *JSON*, но могут содержать бинарные данные, даты и другие типы данных.

MongoDB разрабатывалась с учетом масштабируемости и производительности, что делает ее особенно привлекательной для проектов, которые требуют быстрой и эффективной обработки больших объемов данных. Она имеет множество функций, которые делают ее полезной для многих типов приложений, включая:

- гибкий документо-ориентированный подход – *MongoDB* предоставляет гибкую модель документо-ориентированной базы данных, что позволяет хранить данные любого формата, включая структурированные, полуструктурированные и неструктурированные данные;
- высокая доступность – *MongoDB* предоставляет высокую доступность благодаря встроенному механизму репликации, который позволяет хранить несколько копий данных на разных серверах;
- масштабируемость – *MongoDB* поддерживает горизонтальное и вертикальное масштабирование, что позволяет ей обрабатывать большие объемы данных;
- мощный язык запросов – *MongoDB* предоставляет мощный язык запросов, который позволяет быстро и эффективно извлекать данные из базы данных;
- индексы – *MongoDB* поддерживает множество типов индексов, которые позволяют ускорить поиск и фильтрацию данных.

MongoDB широко используется в различных областях, включая социальные сети, онлайн-магазины, приложения для анализа данных и многие другие.

Ее популярность объясняется ее гибкостью, производительностью и масштабируемостью [3].

Express.js – это легковесный фреймворк для *Node.js*, который используется для разработки веб-приложений и *API*. Он предоставляет удобный и гибкий механизм для обработки запросов и ответов, маршрутизации и создания модульной структуры приложений. *Express.js* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений. Это позволяет разработчикам быстро решать задачи и получать поддержку при возникновении проблем.

Express.js облегчает разработку веб-приложений благодаря простому и интуитивно понятному *API*. Он позволяет быстро создавать маршруты, обрабатывать запросы и ответы, а также работать с различными *middleware*-пакетами. Это позволяет разработчикам создавать приложения с различными функциональными возможностями. *Express.js* не навязывает строгую структуру приложения, что дает возможность гибко настраивать его под конкретные задачи.

Благодаря встроенным механизмам, *Express.js* позволяет легко масштабировать приложения, что особенно важно для больших и сложных проектов. Он поддерживает работу с кластерами и позволяет распределять нагрузку между несколькими серверами.

Middleware в *Express.js* – это специальные функции, которые предоставляет широкие возможности для работы с фреймворком, что позволяет улучшать функциональность приложения и повышать его безопасность. С помощью *middleware* можно добавлять авторизацию, обработку ошибок, логгирование и многое другое.

Данный фреймворк используется для создания различных типов приложений, включая *API*, веб-серверы, приложения для обработки данных и многие другие. Он позволяет быстро создавать и масштабировать приложения, обеспечивая при этом гибкость и удобство разработки [4].

React – это библиотека *JavaScript*, разработанная *Facebook*, которая используется для создания пользовательских интерфейсов. *React* использует декларативный подход для описания компонентов пользовательского интерфейса, что делает его более простым и понятным для разработчиков. Он позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов.

Основные преимущества *React*:

- декларативный подход – *React* использует декларативный подход для описания пользовательского интерфейса, что делает его более понятным для разработчиков;
- переиспользуемые компоненты – *React* позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов;
- эффективный – *React* использует виртуальный *DOM*, который позволяет изменять только те элементы, которые действительно изменились;
- большое сообщество – *React* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений;

– простота – *React* является относительно простым и понятным инструментом для создания пользовательских интерфейсов.

React используется для создания интерактивных пользовательских интерфейсов, включая веб-приложения, мобильные приложения, игры и многое другое. Он позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов [5].

Node.js – это среда выполнения *JavaScript* на стороне сервера, которая позволяет разрабатывать высокопроизводительные и масштабируемые веб-приложения. Она основана на движке *V8*, разработанном *Google* для браузера *Chrome*, и позволяет использовать *JavaScript* для создания приложений на серверной стороне.

Основные преимущества *Node.js*:

– высокая производительность – *Node.js* основан на движке *V8*, который обеспечивает быстрое выполнение;

– масштабируемость – *Node.js* позволяет создавать масштабируемые приложения с помощью механизма обработки запросов в нескольких потоках;

– широкие возможности – *Node.js* имеет большое количество библиотек и модулей, которые позволяют упростить разработку и расширить функциональность приложения;

– единый язык – *Node.js* использует *JavaScript* как единый язык для программирования на серверной и клиентской стороне, что упрощает разработку и повышает эффективность работы разработчика;

– активное сообщество – *Node.js* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений.

Node.js используется для создания различных приложений на серверной стороне, включая веб-приложения, микросервисы, *API* и многое другое. Он позволяет разработчикам создавать высокопроизводительные и масштабируемые приложения с использованием *JavaScript* на стороне сервера, что упрощает разработку и повышает эффективность работы разработчика.

1.4 Анализ инструментальных средств автоматизации разработки и тестирования

Для создания программного комплекса по обслуживанию жилого фонда студенческого общежития будет использована среда разработки *Webshtorm* от компании *JetBrains*. Для визуализации базы данных *MongoDb* лучше всего подходит приложение *MongoDb Compass*. Хранить исходный код только на локальном компьютере плохая практика, поэтому будет создан удаленный репозиторий на *GitHub*. Для построения различных диаграмм, которые позволят упростить разработку, а также предоставят полное понимание работы приложения, будет использоваться *StarUml*.

Рассмотрим подробнее каждый из этих инструментов.

Webshtorm – это интегрированная среда разработки (*IDE*), которая предоставляет обширный функционал для разработки веб-приложений. Среда разработки позволяет работать с различными языками программирования, включая

JavaScript, TypeScript, HTML, CSS, Node.js, Angular, React и другие. *Webshtorm* включает в себя встроенные инструменты отладки, систему автодополнения кода, автоматическую проверку ошибок, систему контроля версий, анализаторы кода и многие другие полезные функции.

Webshtorm обладает многоплатформенностью, что позволяет разрабатывать на разных операционных системах, включая *Windows, macOS* и *Linux*. Среда разработки также имеет мощную систему плагинов, которая позволяет расширять функционал *IDE*, добавляя поддержку новых языков программирования и инструментов. Она также имеет множество инструментов для работы с базами данных, включая поддержку *MongoDB, MySQL, PostgreSQL, Oracle* и других, обеспечивает интеграцию с браузерами для отладки веб-приложений в режиме реального времени.

Одним из основных преимуществ *Webshtorm* является его эффективность и производительность. Среда разработки использует многопоточную архитектуру и оптимизированный механизм работы с памятью, что обеспечивает быстрое действие и позволяет работать с большими проектами. Кроме того, *Webshtorm* имеет обширную документацию и активное сообщество пользователей, которые создают полезные плагины, советы и обучающие ресурсы, что делает процесс разработки еще более комфортным и эффективным.

Webshtorm – это мощная среда разработки, которая облегчает и ускоряет процесс создания высококачественных веб-приложений, идеально подходящая для опытных и начинающих разработчиков [6].

MongoDB Compass – это визуальный интерфейс для работы с базами данных *MongoDB*. Этот инструмент позволяет разработчикам и администраторам баз данных *MongoDB* легко просматривать, анализировать и манипулировать данными. Он предоставляет графический интерфейс для создания, редактирования и удаления коллекций, документов и индексов базы данных *MongoDB*. Он также обеспечивает визуализацию структуры коллекций, что помогает быстро понять структуру данных.

MongoDB Compass имеет встроенный механизм запросов, который позволяет быстро и легко создавать и выполнять запросы к базе данных. Он также обеспечивает удобный механизм фильтрации и сортировки данных, а также поддержку агрегационных запросов.

Среди других возможностей *MongoDB Compass* – поддержка графического интерфейса для выполнения команд в *MongoDB*, создание и сохранение запросов для повторного использования, анализ статистики запросов и многое другое.

MongoDB Compass также обеспечивает удобный механизм подключения к серверу базы данных *MongoDB*, что позволяет работать с удаленными базами данных. Он также поддерживает механизм аутентификации и авторизации, что обеспечивает безопасность при работе с базами данных.

В целом, *MongoDB Compass* – это удобный и мощный инструмент для работы с базами данных *MongoDB*, который обеспечивает широкий спектр возможностей для работы с данными, а также удобный интерфейс для управления базой данных. Этот инструмент идеально подходит для разработчиков и администраторов баз данных *MongoDB*, которые хотят упростить и ускорить работу с

данными [7].

GitHub – это веб-сервис для хранения и совместной работы над *Git*-репозиториями. Это платформа, которая позволяет разработчикам хранить и совместно работать над кодом, отслеживать ошибки, создавать ветки, а также просматривать и редактировать код, управлять версиями, изменениями и запросами на слияние.

Репозиторий в *GitHub* – это хранилище для кода и других файлов, которые могут быть загружены и управляемы в *Git*. Репозиторий в *GitHub* позволяет хранить и управлять кодом в облаке, а также предоставляет множество функций для работы с кодом, включая возможность комментирования кода, управления задачами и многое другое.

Если хранить код только на локальном компьютере, то это может привести к потере данных в случае сбоя жесткого диска или других проблем с компьютером. Кроме того, хранение кода на локальном компьютере не обеспечивает возможность совместной работы и синхронизации изменений между различными разработчиками.

Таким образом, *GitHub* – это мощный и удобный инструмент для хранения и совместной работы над кодом. Он предоставляет широкий спектр возможностей для управления кодом, удобный интерфейс для просмотра и редактирования кода, а также возможность совместной работы и синхронизации изменений между различными разработчиками.

StarUML – это приложение для создания *UML*-диаграмм, которое позволяет разработчикам создавать модели проектов, планировать архитектуру и дизайн приложений (рисунок 1.2). *StarUML* имеет графический интерфейс пользователя, который предоставляет множество инструментов для создания и редактирования диаграмм, включая диаграммы классов, диаграммы последовательностей, диаграммы состояний, диаграммы активностей и многое другое.

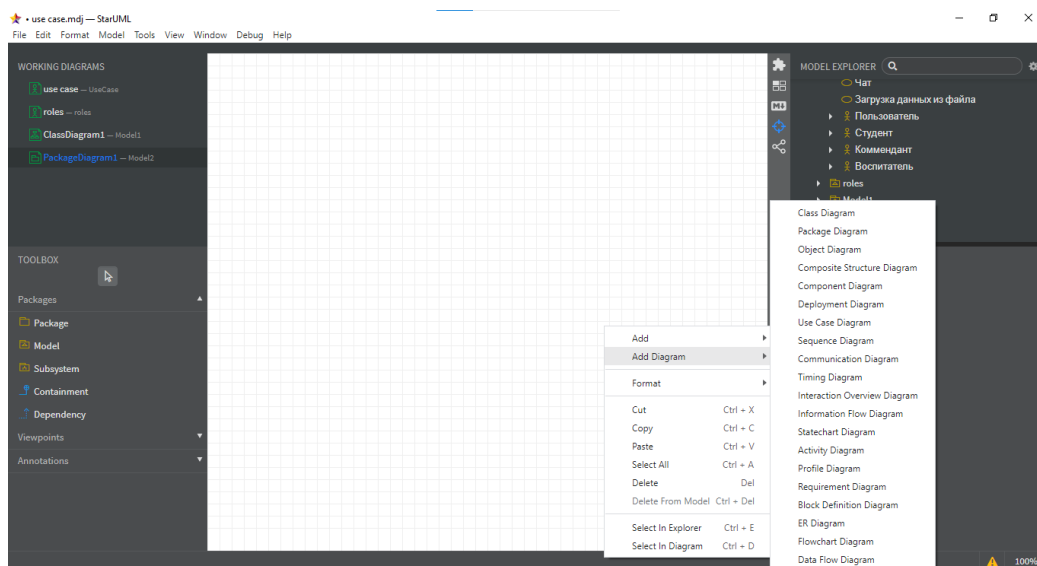


Рисунок 1.2 – Интерфейс главного окна *StarUml*

В *StarUML* можно создавать различные типы диаграмм, включая:

- диаграммы классов – используются для описания структуры классов и их отношений;
- диаграммы последовательностей – используются для описания взаимодействия между объектами и процессов, происходящих во времени;
- диаграммы состояний – используются для описания жизненного цикла объекта и его состояний;
- диаграммы компонентов – используются для описания структуры и отношений между компонентами системы;
- диаграммы развертывания – используются для описания физического размещения компонентов и системы в целом;
- диаграммы активностей – используются для описания последовательности действий и процессов в системе.

На рисунке 1.2 в виде выпадающего меню представлен весь список возможных диаграмм.

StarUML – это мощный и удобный инструмент для создания *UML*-диаграмм, который предоставляет широкий спектр возможностей для создания и редактирования диаграмм, а также импорта и экспорта диаграмм в различных форматах файлов. Он также имеет множество функций, которые облегчают создание и редактирование диаграмм, и может быть расширен за счет пользовательских элементов, шаблонов и плагинов [8].

Postman – это инструмент для тестирования *API*, который позволяет разработчикам быстро и удобно отправлять запросы к *API*, тестировать их и анализировать ответы.

С помощью *Postman* можно отправлять запросы на сервер и получать ответы в разных форматах, таких как *JSON*, *XML*, *HTML* и другие. Также в *Postman* есть множество функциональных возможностей, таких как автоматическое генерирование документации, управление авторизацией, создание и выполнение тестовых сценариев и другие. Он поддерживает различные типы запросов, включая *GET*, *POST*, *PUT*, *DELETE* и многие другие. Вы можете отправлять запросы с параметрами, заголовками и телом запроса в формате *JSON* или форм-данных.

Один из самых удобных и полезных аспектов *Postman* – это коллекции запросов, которые можно создавать и организовывать для более эффективного управления тестированием *API*. Коллекции могут быть экспортированы и импортированы в различных форматах, таких как *JSON* и *XML*.

Postman имеет множество преимуществ, которые делают его популярным среди разработчиков, вот некоторые из них:

- удобный интерфейс – *Postman* интуитивно понятен и удобен в использовании;
- простота в использовании – *Postman* не требует от разработчика особых навыков, чтобы начать использовать его;
- расширяемость – *Postman* можно легко расширять, используя плагины: добавить поддержку авторизации на основе *OAuth* или создать пользовательский набор инструментов для тестирования *API*;
- автоматизация – *Postman* позволяет автоматизировать тестирование *API*;
- множество функций – *Postman* имеет множество функций, таких как

создание коллекций, генерация документации и выполнение тестовых сценариев, что делает его полезным инструментом для разработки и тестирования приложений;

– бесплатность – *Postman* имеет бесплатную версию, которая покрывает большинство потребностей разработчиков, а также платную версию с дополнительными функциями.

Postman является полезным и удобным инструментом для тестирования *API*, который может существенно ускорить процесс разработки приложений [9].

1.5 Техническое задание для клиент-серверного программного продукта «Программный комплекс автоматизации обслуживания жилого фонда студенческого общежития»

Цель разработки: разработать приложение, предназначенное для автоматизации обслуживания жилого фонда студенческого общежития.

Данный программный комплекс предназначен для студентов и работников студенческого общежития.

Приложение позволит коменданту следующее:

– вести учет технического оборудования, которое ввозят студенты для личного пользования;

– вести учет количества свободных и занятых мест с возможностью фильтрации за год или за месяц;

– для заполнения списка студентов комендант сможет импортировать файл формата *Excel* в базу данных;

– мониторинг оплаты за проживание.

Для студентов будет реализовано:

– личный кабинет, где они смогут следить за текущими новостями и событиями;

– чат с воспитателем;

– возможность создавать заявки на починку бытового оборудования.

Для воспитателей приложение предусматривает следующие функции:

– создание и ведение ленты новостей;

– создание и ведение ленты событий;

– начисление баллов студентам;

– создание чата со студентом или общей беседы;

– оставлять замечания в личном кабинете студента.

Приложение должно иметь следующую структуру и функциональность:

– работать как веб-приложение основанное на *REST* архитектуре;

– являться кроссплатформенным;

– являться кроссбраузерным;

– иметь принцип работы, аналогичный веб-приложениям (реализовывать *REST* архитектуру, быть доступным посредством веб-браузера, иметь пользовательский интерфейс).

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И АЛГОРИТМЫ

2.1 Анализ предметной области

Анализ предметной области «Автоматизация обслуживания жилого фонда студенческого общежития» позволяет выделить следующие основные аспекты:

- учет и распределение комнат – для автоматизации данного процесса необходимо иметь возможность вести учет свободных и занятых комнат, а также осуществлять распределение студентов в свободные комнаты;
- регистрация и учет студентов – необходимо иметь возможность регистрировать студентов и вести учет их проживания в общежитии, а также осуществлять контроль за своевременной оплатой за проживание;
- предоставление различной информации – необходимо иметь возможность создавать и просматривать новостные ленты для того;
- связь воспитателей и студентов – необходимо иметь возможность связываться с студентом для решения вопросов, связанных с проживанием в общежитии.

Для автоматизации этих процессов может быть использована специализированная информационная система, разработанная с учетом конкретных потребностей студенческого общежития. Она будет включать в себя базу данных для хранения информации о студентах и комнатах, модули для управления ресурсами и контроля доступа, а также возможности для сотрудников общежития добавлять и изменять информацию о конкретном студенте.

Таким образом, разработка автоматизированной системы позволит упростить и ускорить процессы управления жилым фондом студенческого общежития, повысить эффективность использования ресурсов и улучшить качество обслуживания студентов. Благодаря использованию базы данных, все данные будут храниться в одном месте, и сотрудники общежития смогут быстро получать необходимую информацию о студентах и комнатах. Модули управления ресурсами и контроля доступа позволят более точно контролировать использование ресурсов и обеспечить безопасность в общежитии. Разработка системы также может способствовать улучшению учебного процесса, так как облегчит жизнь студентам и позволит им сконцентрироваться на учебе, а не на решении бытовых вопросов.

2.2 Функциональная модель программного комплекса

Для того, чтобы создать качественное приложение, нужно четко понимать, какие роли у нас будут и какой функционал им будет доступен. Для этого нужно знать, какие объекты попадают в предметную область проектируемой системы и какие логические связи между ними существуют. Для формирования такого понимания используются логические модели предметной области. Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области. Для стабильной работы

программного обеспечения необходимо чёткое распределение на роли, на основе которых будут формироваться функции взаимодействия со внутренней средой приложения.

Актер – множество логически связанных ролей в *UML*, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Актером может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Актеры в диаграммах прецедентов *UML* используются для представления ролей, которые взаимодействуют с системой или используют её функциональность. Они могут быть реальными людьми, группами людей, другими программными системами или даже внешними сервисами.

Прецеденты представляют действия, выполняемые системой в интересах актеров. Проще говоря, прецедент – это описание последовательности действий (или нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат. Один актер может использовать несколько элементов прецедентов, и наоборот, один прецедент может иметь несколько актеров, использующих его. Каждый прецедент задает определенный путь использования системы. Набор всех прецедентов определяет полные функциональные возможности системы.

Взаимодействие актеров с прецедентами происходит через использование прецедентов. Прецеденты описывают функциональные возможности системы, доступные актерам. В результате взаимодействия актеров с прецедентами система выполняет определенные действия и предоставляет соответствующие результаты.

Приложение должно реализовывать три роли: комендант, воспитатель, студент.

Основные функции для роли «Комендант»:

- учет личного технического оборудования;
- учет количества свободных и занятых мест;
- возможность импортировать данные о студентах формата *Excel*;
- производить мониторинг оплаты за проживание.

Основные функции для роли «Студент»:

- просмотр новостей и событий;
- чат с воспитателем;
- создавать заявки на починку бытового оборудования.

Основные функции для роли «Воспитатель»:

- создание и ведение ленты новостей и событий;
- начисление баллов студентам;
- создание чата со студентом;
- создание общей беседы;
- возможность оставлять замечания в личном кабинете студента.

В качестве среды для разработки и отображения функциональной структуры программы будет использована *StarUml*. На рисунке 2.1 представлена *Use Case* диаграмма приложения.

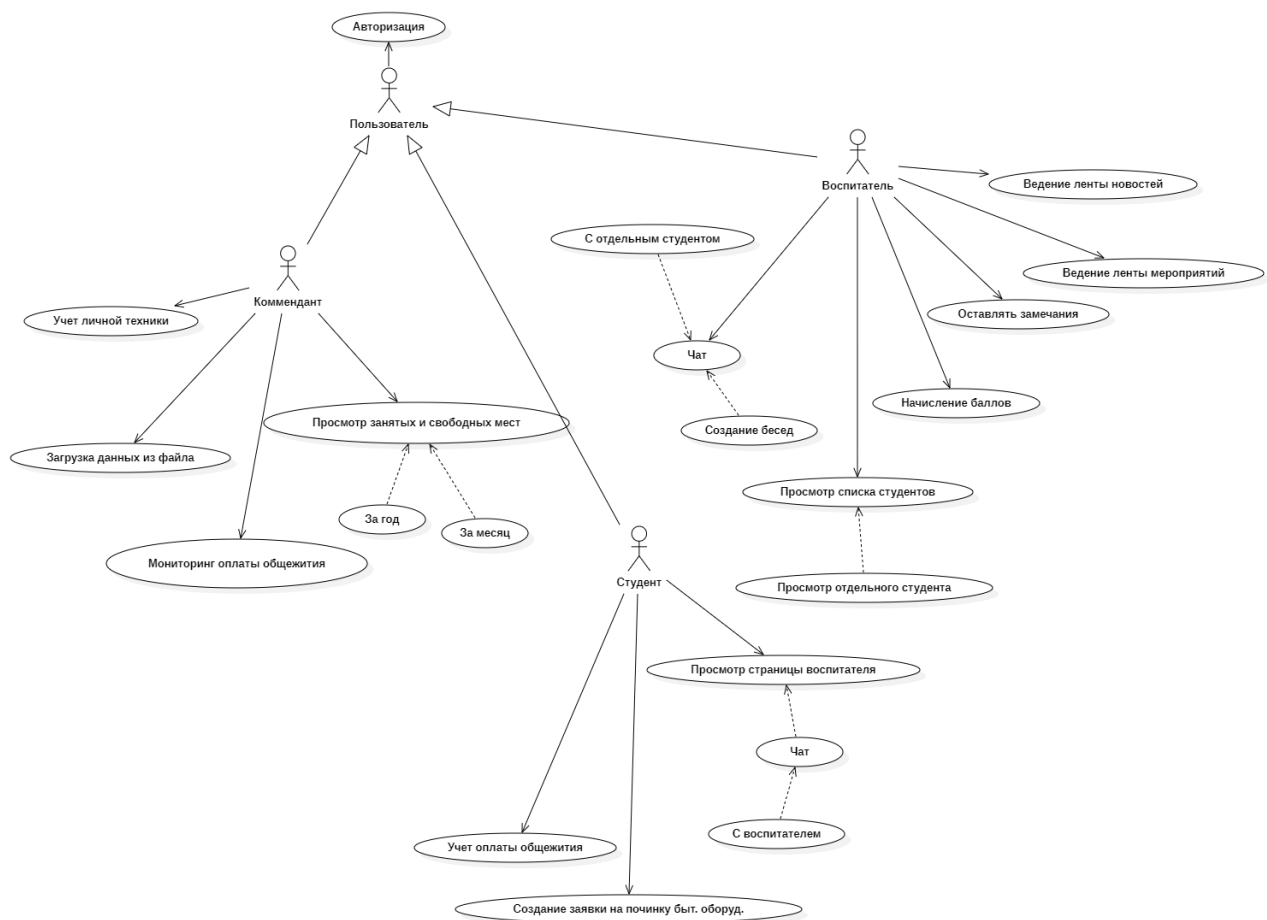


Рисунок 2.1 – Диаграмма прецедентов и актеров

Далее приведено описание прецедентов для роли «Комендант».

Прецедент «Учет личного техники» предназначен для просмотра и управления личным техническим оборудованием студента.

Прецедент «Загрузка данных из файла» предназначен для выгрузки личных данных студентов в базу данных.

Прецедент «Мониторинг оплаты общежития» предназначен для просмотра задолженности перед общежитием, а также для информирования студента.

Прецедент «Просмотр занятых и свободных мест» предназначен для мониторинга занятости комнат.

Далее приведено описание прецедентов для роли «Воспитатель».

Прецедент «Чат» предназначен для общения с отдельным студентом или с неким количеством.

Прецедент «Просмотр списка студентов» предназначен для просмотра информации о студентах в целом или более подробно об отдельном студенте.

Прецедент «Начисление баллов» предназначен для начисления баллов отдельному студенту.

Прецедент «Оставлять замечания». Данный прецедент необходим для того, чтобы воспитатель мог оповестить студента о каком-то совершенном нарушении в его личном кабинете.

Прецедент «Ведение ленты новостей» предназначен для просмотра и создания новых новостей.

Прецедент «Ведение ленты мероприятий» предназначен для просмотра и создания с целью оповещения новых событий.

Диаграмма прецедентов и актеров отображает функциональность системы с точки зрения пользователей. Она показывает взаимодействие между пользователем и системой в рамках конкретной задачи или сценария использования.

2.3 Информационная модель программного комплекса

На основе анализа предметной области определим набор коллекций и их свойства. В *MongoDB* коллекция представляет собой группу документов, которые хранятся в базе данных. Коллекции в *MongoDB* являются аналогом таблиц в реляционных базах данных. Они содержат документы в формате *JSON*, которые могут иметь различную структуру и не обязательно должны иметь одинаковые поля.

В *MongoDB* каждый документ представляет отдельную запись в коллекции и может содержать произвольное количество полей и значений. Это отличает *MongoDB* от реляционных баз данных, где таблицы обычно имеют фиксированную структуру с определенными столбцами и типами данных.

В коллекциях *MongoDB* можно определять индексы, которые позволяют ускорить поиск и запросы к данным. Индексы могут быть созданы на одном или нескольких полях документа и помогают оптимизировать производительность при выполнении запросов.

Коллекции в *MongoDB* позволяют гибко организовывать и хранить данные в базе данных, а также обеспечивают возможность работы с разнообразными структурами документов. Это позволяет разработчикам эффективно моделировать предметные области и управлять данными в *MongoDB*. Также *MongoDB* не требует предварительного определения схемы данных. Это означает, что разработчики могут свободно добавлять, изменять и удалять поля в документах без необходимости изменения структуры всей коллекции. Такая гибкость позволяет легко адаптировать схему данных к изменяющимся требованиям предметной области.

Для отображения информационной модели разрабатываемого продукта были выделены следующие коллекции:

- коллекция «Студент», которая хранит в себе информацию о студентах, а также о комнате, где они живут;
- коллекция «Аккаунт», которая хранит в себе данные для входа в личный аккаунт пользователя;
- коллекция «Сотрудники», которая хранит в себе личные данные сотрудников общежития;
- коллекция «Новость», которая хранит в себе список новостей;
- коллекция «Чат», которая хранит список всех чатов;
- коллекция «Заявки», которая хранит информацию о заявке студента на починку бытового оборудования.

На картинке 2.2 приведена схема базы данных.

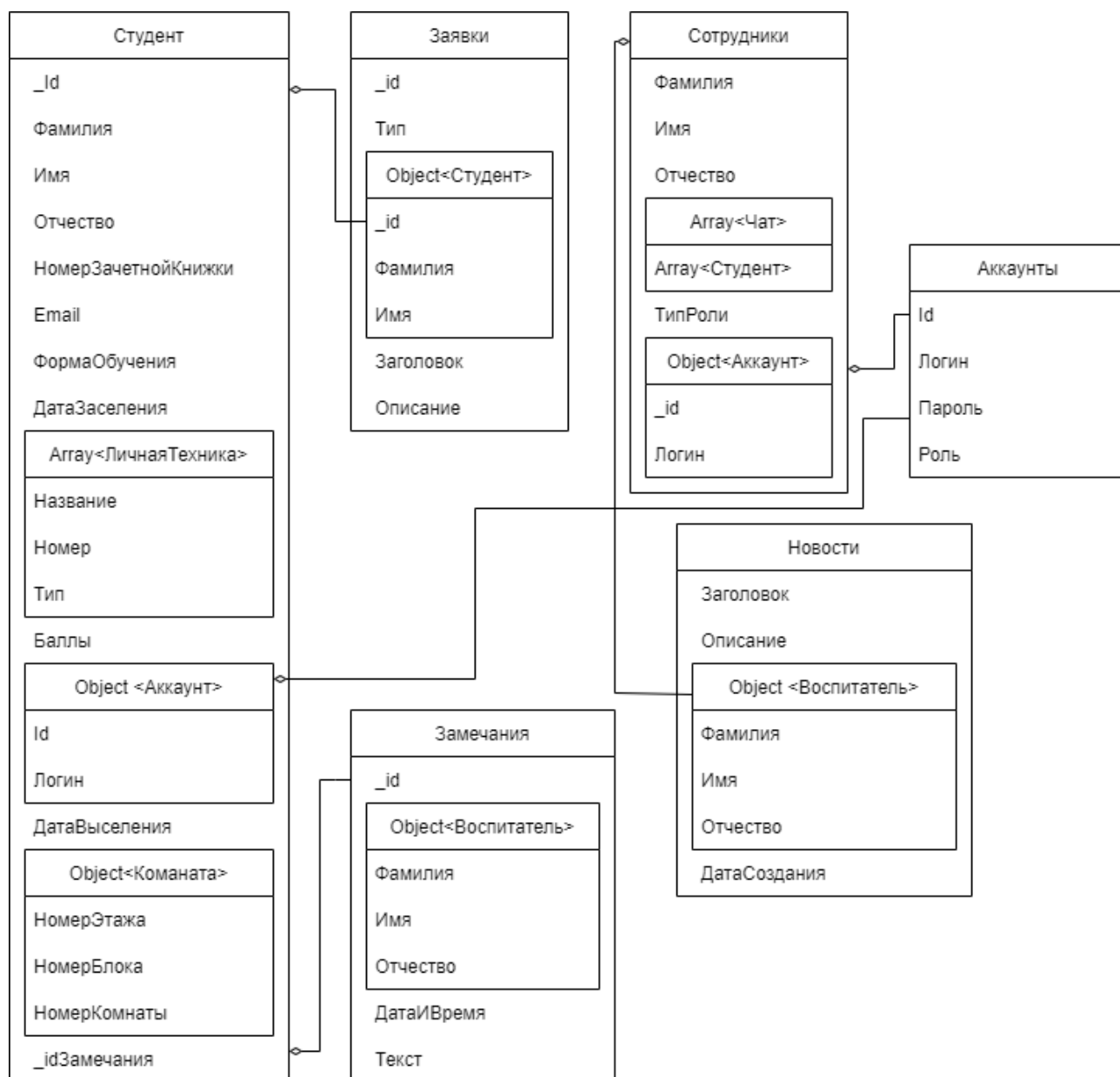


Рисунок 2.2 – Схема базы данных

Поля в документах могут быть как обязательными, так и не обязательными. Например, если в коллекции *MongoDB* есть поле *name*, оно может быть обязательным, то есть требовать наличия значения, или необязательным, где оно может быть пустым или иметь значение по умолчанию, если не указано явно. В случае, если поле не обязательное, оно может иметь значение по умолчанию.

Тип *ObjectId* является специальным типом, предоставляем *Mongoose*. *ObjectId* представляет собой уникальный идентификатор, который состоит из 12-байтового значения и содержит информацию о времени создания, машине и случайном числе. Этот тип данных обеспечивает уникальность идентификаторов в пределах коллекции *MongoDB*.

Подробное описание сущностей приведено в таблицах 2.1 – 2.6.

Таблица 2.1 – Коллекция «Студент»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>firstName</i>	<i>String</i>	Да	Нет
<i>secondName</i>	<i>String</i>	Да	Нет
<i>middleName</i>	<i>String</i>	Да	Нет
<i>numberTest</i>	<i>String</i>	Да	Да
<i>email</i>	<i>String</i>	Нет	Да
<i>formEducation</i>	<i>Union</i> ("платное" "бесплатное")	Да	Нет
<i>dateEntry</i>	<i>String</i>	Да	Нет
<i>balls</i>	<i>Number</i>	Да	Нет
<i>privateTechs</i>	<i>Array<Techs></i>	Нет	Нет
<i>privateTech.model</i>	<i>String</i>	Нет	Нет
<i>privateTech.number</i>	<i>String</i>	Нет	Нет
<i>privateTech.type</i>	<i>String</i>	Нет	Нет
<i>room</i>	<i>Object</i>	Да	Нет
<i>room.floor</i>	<i>Number</i>	Да	Нет
<i>room.block</i>	<i>Number</i>	Да	Нет
<i>room.apartament</i>	<i>Number</i>	Да	Нет
<i>remarks</i>	<i>Array<Remark></i>	Нет	Нет
<i>remark.dateAndTime</i>	<i>String</i>	Да	Нет
<i>remark.header</i>	<i>String</i>	Да	Нет
<i>remark.status</i>	<i>String</i>	Да	Нет
<i>remark.text</i>	<i>String</i>	Да	Нет
<i>remark.mentor</i>	<i>Object</i>	Да	Нет
<i>remark.mentor.firstName</i>	<i>String</i>	Да	Нет
<i>remark.mentor.secondName</i>	<i>String</i>	Да	Нет
<i>remark.mentor.middleName</i>	<i>String</i>	Да	Нет
<i>account</i>	<i>Object</i>	Да	Да
<i>account.login</i>	<i>String</i>	Да	Да

В коллекции, описанной в таблице 2.1 поле *formEducation* имеет специальный тип *union*, предоставляемый *TypeScript*. Данный тип представляет объединение нескольких строк, которые разрешено присваивать данной переменной.

Таблица 2.2 – Коллекция «Новости»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>header</i>	<i>String</i>	Да	Нет
<i>description</i>	<i>String</i>	Да	Нет
<i>mentor</i>	<i>Object</i>	Да	Нет
<i>mentor.firstName</i>	<i>String</i>	Да	Нет
<i>mentor.second- Name</i>	<i>String</i>	Да	Нет
<i>mentor.mid- delName</i>	<i>String</i>	Да	Нет
<i>dateCreate</i>	<i>Date</i>	Да	Нет

Как видно из таблицы 2.2, коллекция «Новости» хранит в себе данные воспитателя. Это необходимо для того, чтобы не осуществлять поиск по коллекции «Сотрудники» для установления данных создателя документа, что позволит нам сэкономить время загрузки данных.

Таблица 2.3 – Коллекция «Аккаунты»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>login</i>	<i>String</i>	Да	Да
<i>password</i>	<i>String</i>	Да	Нет

В таблице 2.3 приведено описание полей коллекции «Аккаунты». Данная коллекция нужна для быстрого поиска пользователя в системе во время его авторизации.

Таблица 2.4 – Коллекция «Чат»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
1	2	3	4
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>name</i>	<i>String</i>	Да	Нет
<i>messages</i>	<i>Array<Chat></i>	Да	Нет
<i>messages.who</i>	<i>Object</i>	Да	Нет
<i>messages.who.firstName</i>	<i>String</i>	Да	Нет
<i>messages.who.secondName</i>	<i>String</i>	Да	Нет

Продолжение таблицы 2.4

1	2	3	4
<i>messages.who.middleName</i>	<i>String</i>	Да	Нет
<i>messages.when</i>	<i>Date</i>	Да	Нет
<i>messages.message</i>	<i>String</i>	Да	Нет

В таблице 2.4 описана коллекция «Чат». Данная коллекция хранит названия чата и его сообщения. Поле *messages* хранит массив сообщений. Каждое сообщение хранит дату создания, отправителя и текст.

Таблица 2.5 – Коллекция «Сотрудники»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>firstName</i>	<i>String</i>	Да	Нет
<i>secondName</i>	<i>String</i>	Да	Нет
<i>middleName</i>	<i>String</i>	Да	Нет
<i>role</i>	<i>Union</i> ("mentor" "main")	Да	Нет
<i>chats</i>	<i>Array<ObjectId></i>	Нет	Нет

В таблице 2.5 описаны поля для коллекции «Сотрудники». Все поля, кроме *chats*, обязательные. Поле *role* является типом *Union* и может принимать один из двух значений: *mentor* или *main*. Поле *chats* хранит массив типа *ObjectId*. Данный массив заполняется *id* из коллекции «Чат», описанной в таблице 2.4.

Таблица 2.6 – Коллекция «Заявки»

Название	Тип	Обязательное (<i>required</i>)	Уникальное (<i>unique</i>)
1	2	3	4
<i>_id</i>	<i>ObjectId</i>	Да	Да
<i>type</i>	<i>String</i>	Да	Нет
<i>student</i>	<i>Object</i>	Да	Нет
<i>student.firstName</i>	<i>String</i>	Да	Нет
<i>student.secondName</i>	<i>String</i>	Да	Нет
<i>student.room</i>	<i>Number</i>	Да	Нет

Продолжение таблицы 2.6

1	2	3	4
<i>student.room.floor</i>	<i>Number</i>	Да	Нет
<i>student.room.block</i>	<i>Number</i>	Да	Нет
<i>student.room.apartment</i>	<i>Number</i>	Да	Нет
<i>header</i>	<i>String</i>	Да	Нет
<i>description</i>	<i>String</i>	Да	Нет

В таблице 2.6 представлена коллекция «Заявки». Данная коллекция хранит созданные студентом заявки на починку бытового оборудования. Поле *student* является объектом и хранит фамилию, имя, отчество и номер комнаты. Поля *type* необходимо для того, чтобы комендант смог отфильтровать входящие заявки по типу: электрик, сантехник, столяр.

Как можно заметить, каждая коллекция в *MongoDB* может иметь некоторые поля, которые могут быть обязательными или необязательными. Кроме того, поля могут быть уникальными или неуникальными в пределах коллекции. Поле, которое является уникальным, не позволяет вставлять документ с таким же значением, как уже имеющийся в коллекции. Поле, которое является обязательным, требует, чтобы его значение было указано при вставке документа. Если обязательное поле не указано, то операция вставки завершится неудачей. Коллекция также может иметь индексы, которые помогают ускорить выполнение запросов к базе данных.

MongoDB поддерживает несколько типов данных для удобного хранения информации. Вот некоторые из основных типов данных в *MongoDB*:

- *String* – используется для хранения текстовых данных;
- *Number* – используется для хранения числовых значений, как целых чисел, так и чисел с плавающей запятой;
- *Boolean* – используется для хранения логических значений *true* (истина) или *false* (ложь);
- *Array* – используется для хранения упорядоченных коллекций значений;
- *Date* – используется для хранения даты и времени;
- *Null* – используется для хранения отсутствующих или пустых значений.

В целом, использование *MongoDB* позволяет эффективно хранить и управлять данными, предоставляя удобный интерфейс для доступа к ним.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОМПЛЕКСА

3.1 Архитектура приложения

Приложение использует трёхуровневую архитектуру (рисунок 3.1). Трёхуровневая архитектура приложения – это подход к проектированию программного обеспечения, в котором приложение разбивается на три слоя: представление (*presentation layer*), бизнес-логика (*business logic layer*) и уровень доступа к данным (*data access layer*).

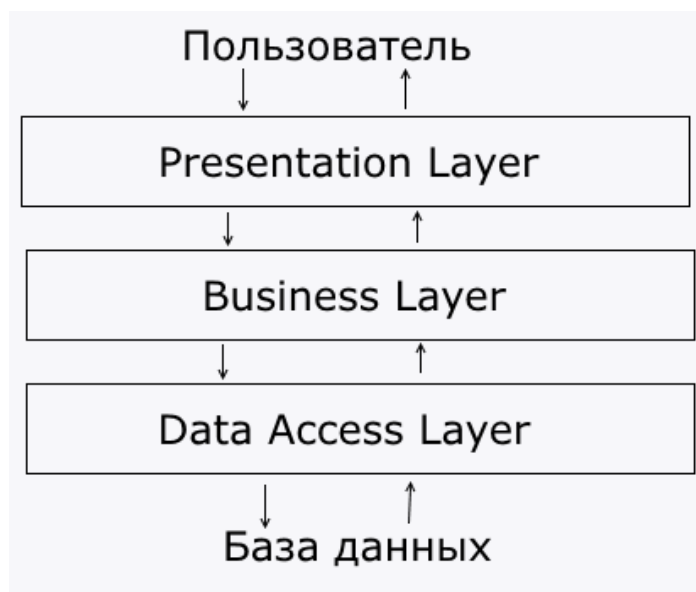


Рисунок 3.1 – Схема архитектуры приложения

Трёхуровневая архитектура приложения на примере стека *MERN* может быть описана следующим образом:

- уровень представления (*Presentation Layer*) – в *MERN*-стеке уровень представления представлен *React*-фреймворком, который используется для разработки клиентской части приложения и позволяет создавать переиспользуемые компоненты, которые управляют отображением данных на странице;

- уровень бизнес-логики (*Application Layer*) – в *MERN*-стеке этот уровень реализован на *Node.js* и *Express.js*: *Node.js* – серверная платформа, которая позволяет запускать *JavaScript* на стороне сервера, *Express.js* – это веб-фреймворк для *Node.js*, который предоставляет удобный *API* для работы с запросами и ответами;

- уровень доступа к данным (*Data Access Layer*) – этот уровень отвечает за доступ к базе данных *MongoDb*, которая хранит данные в формате документов *JSON* и обеспечивает функциональность для чтения, записи, обновления и удаления данных в базе данных.

Таким образом, трёхуровневая архитектура приложения разделяет приложение на три основных уровня: представление, бизнес-логику и уровень доступа к данным. Это позволяет разработчикам легче поддерживать и расширять

приложение, а также упрощает его тестирование и развертывание.

3.2 Уровень представления клиентской части приложения

Уровень представления содержит следующие страницы:

- *Login* – страница для входа в приложение;
- *Registration* – страница для регистрации студента в системе;
- *MainStudent* – главная страница студента;
- *ContainerTech* – данная страница предназначена для просмотра личной техники студентом;
- *PayHostel* – страница для учета оплаты общежития;
- *Claim* – страница для просмотра замечаний;
- *NewsPanel* – страница для просмотра списка новостей;
- *BallsInfo* – страница для просмотра истории начисления баллов;
- *FullNews* – страница для подробного просмотра новости;
- *EventPanel* – страница для просмотра событий;
- *Chat* – страница, которая позволяет вести беседу между воспитателем и студентом;
- *EmployeeList* – страница для просмотра списка воспитателей;
- *FullMentor* – страница для просмотра подробной информации о воспитателе;
- *CreateRepair* – страница для создания студентом заявки на починку бытового оборудования;
- *MainMentor* – главная страница воспитателя;
- *StudentsList* – страница, на которой находится таблица с перечислением краткой информации о студентах;
- *FullStudent* – страница для просмотра полной информации о студенте;
- *CreateClaim* – страница для создания замечания для определенного студента;
- *ChangeBalls* – страница для начисления воспитателем баллов студенту;
- *NewNewsForm* – страница с формой для создания новости;
- *CreateEvent* – страница с формой для создания нового события;
- *CreateChat* – страница для создания нового чата со студентами;
- *MainAdmin* – главная страница для коменданта;
- *AddStudent* – страница с формой для добавления нового студента;
- *RepairView* – страница, на которой комендант может просмотреть входящие заявки на починку бытового оборудования;
- *CreateTechForm* – страница с формой для добавления новой техники для определенного студента;
- *ImportStudents* – страница, на которой комендант может импортировать данный студентов из файла;
- *ViewPayHostel* – страница для просмотра отчетов об оплате за общежитие;
- *CreateMentorForm* – страница с формой для регистрации нового воспитателя;

- *AccPlaces* – страница, на которой отображены занятые места;
- *FreePlaces* – страница, на которой отображены свободные места;
- *Reports* – страница, на которой комендант может запросить различные отчеты.

Страница *Login* (рисунок 3.2) является одной из начальных страниц, которую должны заполнить все роли при первоначальном запуске программы. Данная форма содержит два поля: логин и пароль. Все поля являются обязательными. В случае, если студент еще не зарегистрирован в системе, ему доступны ссылки «Регистрация» и «На главную».

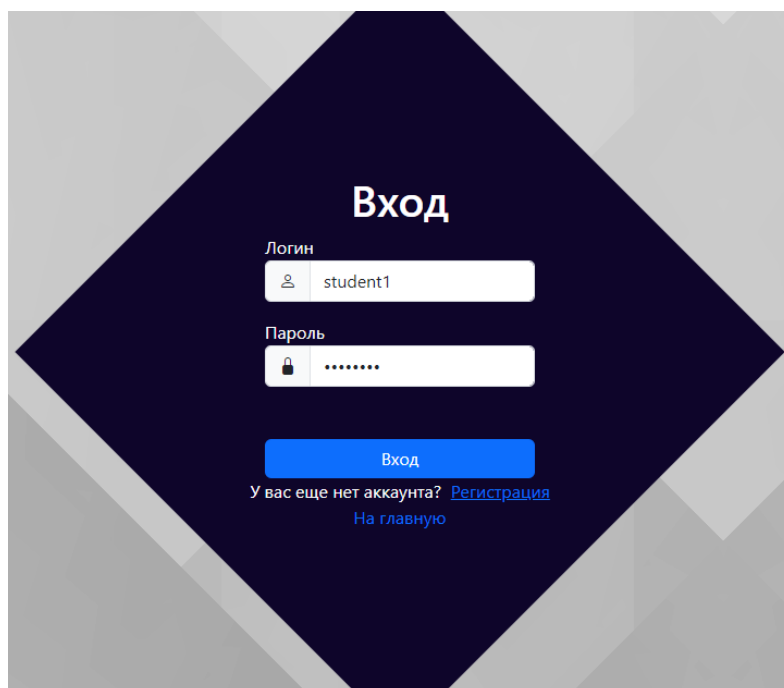
The image shows a login form titled "Вход" (Login) centered on a dark blue diamond-shaped background. Below the title, there are two input fields: "Логин" (Login) with the text "student1" and "Пароль" (Password) with masked characters. A blue button labeled "Вход" (Login) is positioned below the password field. At the bottom, there is a link "У вас еще нет аккаунта? Регистрация" (Don't you have an account? Registration) and a link "На главную" (Home).

Рисунок 3.2 – Вид страницы «*Login*»

Страница *Registration* (рисунок 3.3) предназначена для регистрации студента в системе. На данной странице представлена форма со следующими полями: номер зачетки и *email*. Все поля являются обязательными для заполнения.

Форма регистрации обычно используется для сбора необходимой информации от пользователей, чтобы создать учетную запись и позволить им получить доступ к системе. В данном случае, для успешной регистрации в системе, пользователь должен заполнить оба поля: номер зачетки и *email*.

Номер зачетки представляет собой уникальный идентификатор студента в системе образования. Это числовое значение, которое однозначно идентифицирует студента. Номер зачетки является обязательным полем, поскольку он необходим для создания уникальной учетной записи студента.

Email используется для связи с пользователем и отправки уведомлений или информации о его аккаунте. *Email* должен быть правильно введен, чтобы система могла связаться со студентом и подтвердить его регистрацию. Обязательное заполнение всех полей в форме регистрации обеспечивает достоверность предоставленной информации и позволяет системе создать учетную запись.

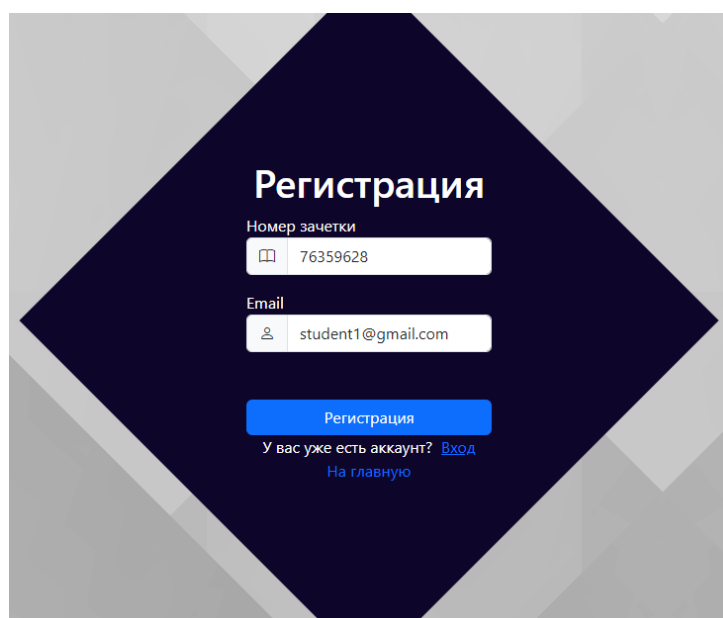


Рисунок 3.3 – Страница регистрации студента

После заполнения всех полей и нажатия на кнопку «Регистрация», студент получит уведомление о том, что заранее сгенерированные системой логин и пароль, были высланы ему на почту. Эти данные студент должен ввести в форму *Login* для успешного входа в приложение.

После успешной авторизации студента откроется страница *MainStudent* (рисунок 3.4).

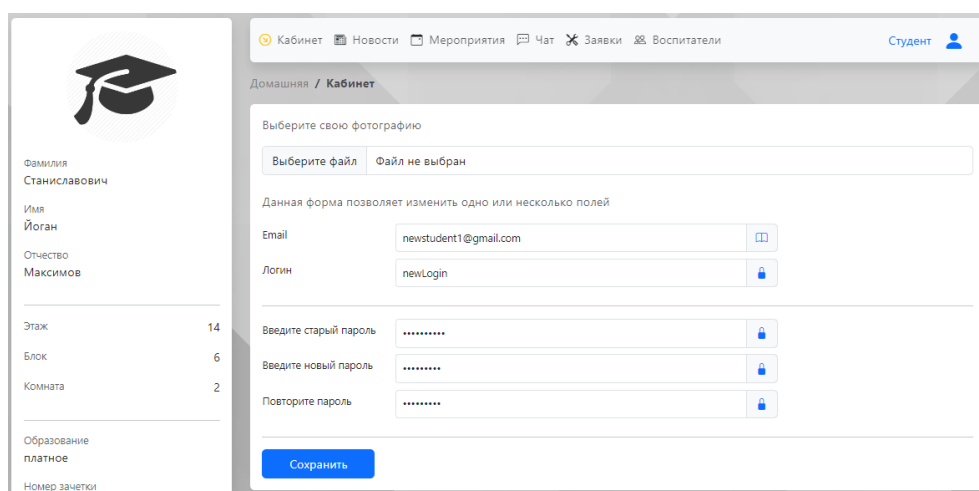


Рисунок 3.4 – Личный кабинет студента

В левой части личного кабинета находится панель с информацией: фамилия, имя, отчество, этаж, блок, комната, форма образования, номер зачетной книжки. В верхней части расположено меню и навигационная панель. По центру расположена форма для изменения личных данных. Данная форма является универсальной, так как позволяет изменить только необходимые данные без заполнения всей формы. После заполнения хотя бы одного поля и нажатия на кнопку

«Сохранить», данные отправятся на сервер. После успешного или не успешного сохранения данных в базе данных, пользователь получит соответствующее уведомление.

Страница *NewsPanel* представляет ленту новостей (рисунок 3.5). Данная лента новостей одинаковая для роли «Студент» и «Воспитатель». В каждой новости имеется ссылка «Прочитать новость полностью», которая откроет новость на новой странице с более подробным содержанием.

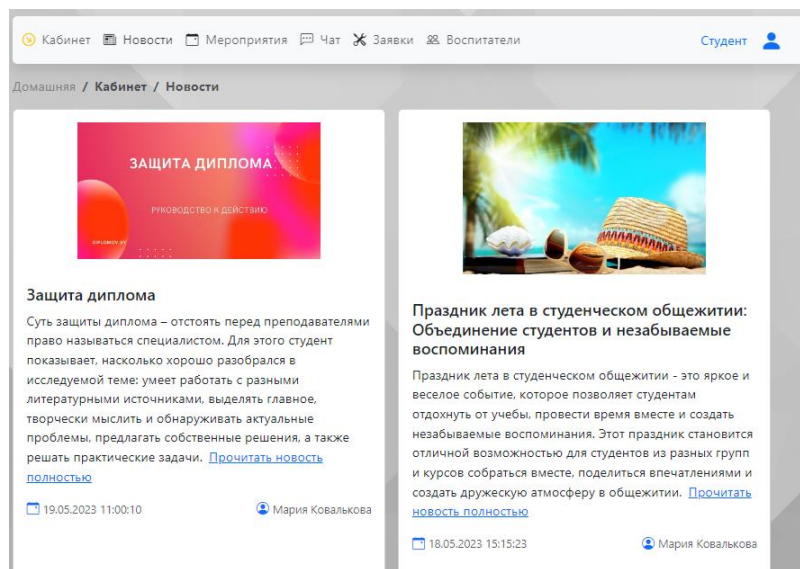


Рисунок 3.5 – Страница с новостями

Страница *CreateRepair* (рисунок 3.6) предназначена для создание заявок на починку бытового оборудования. На данной странице расположена форма и история заявок студента. После нажатия кнопки «Сохранить», данные, введенные студентом в форму, отправляются на сервер и сохраняются в базе данных. При успешном сохранении, студент получает подтверждение о создании заявки на починку оборудования. В истории заявок отображаются все его предыдущие заявки, включая новую.

Рисунок 3.6 – Форма для оформления заявки на починку оборудования

В личном кабинет студент может делать отчет об оплате общежития. Для этого была реализована страница *PayHostel* (рисунок 3.7). Студенту необходимо заполнить форму, а затем нажать кнопку «Сохранить». На данной странице также находится история оплаты, текущая стоимость общежития и оплаченная сумма.

The screenshot shows a web interface for reporting on hostel payment. At the top, there is a breadcrumb trail: «Домашняя / Кабинет / Отчет об оплате». Below this, the date and time are displayed as «Дата и время 05.06.2023 18:35:56». The form contains two input fields: «Номер квитанции» with the value «RT5YU7364GR» and «Сумма» with the value «20». A blue button labeled «Сохранить» is positioned below the inputs. Below the button, the text «Задолженности нет. На счету 0 BYN» is shown in green. Further down, it states «Оплачено всего: 35 (BYN)» and «Текущая стоимость проживания: 35 (BYN)». At the bottom, there is a section titled «История оплаты» containing a list of two payments: «1. 10 (BYN) 24.05.2023 19:19:06 Последняя оплата» and «2. 25 (BYN) 23.05.2023 20:37:02».

Рисунок 3.7 – Форма для отчетности по оплате за проживания

На странице *Claim* (рисунок 3.8) предоставлен список замечаний, которые воспитатель сделал определенному студенту. Для того, чтобы изменить статус с «Новое» на «Прочитано», студенту необходимо нажать на кнопку «Я понял», после чего отправится запрос на сервер для изменения статуса. В противном случае, воспитатель увидит, что данное замечание студентом не прочитано.

The screenshot shows a web interface for a note. At the top, the title «Замечания» is displayed. Below it, there is a note card. The card has a checkbox icon and the date «23.05.2023 20:57:51» on the left, and the status «Новое» on the right. The main text of the note is «Уборка в комнате» followed by «Необходимо убратсья в комнате к пятнице». At the bottom of the card, the name «Метлушко Светлана Александровна» is shown next to a user icon, and a blue button labeled «Я понял» is on the right.

Рисунок 3.8 – Замечание для студента

Воспитатель может создавать новости. Для этого была реализована страница новостей, которая представлена формой *NewNewsForm* (рисунок 3.9). После заполнения всех полей, воспитателю необходимо нажать кнопку «Создать секцию», после чего создается массив объектов с полями *header*, *file*, *description*.

Таким образом воспитатель создает полноценную новость, которая состоит из нескольких секций. После создания нескольких секций, воспитателю необходимо нажать кнопку «Сохранить новость» для сохранения ее в базе данных.

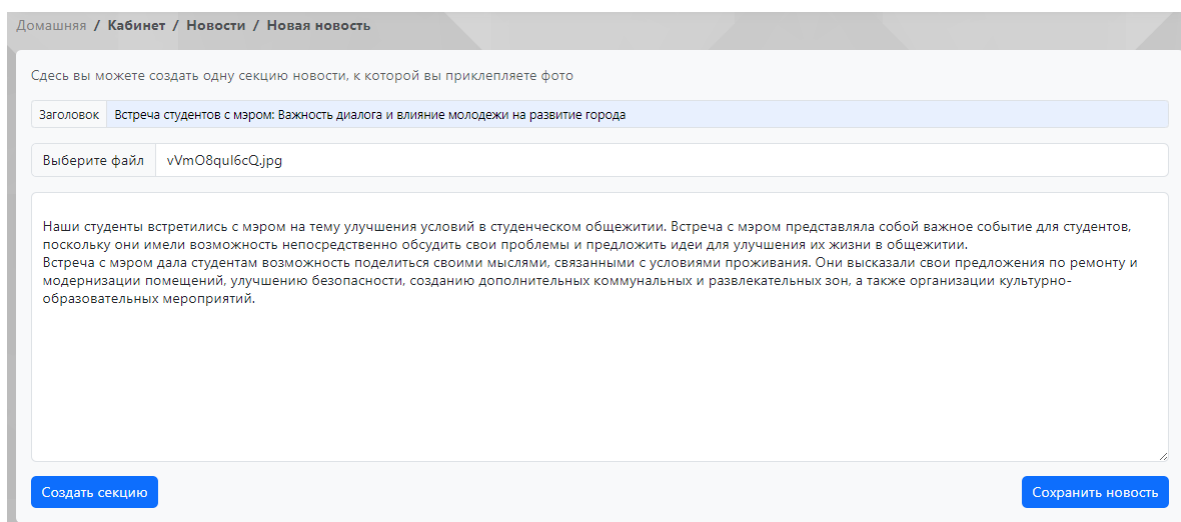


Рисунок 3.9 – Форма для создания новости

Для создания мероприятий была реализованная страница *CreateEvent* (рисунок 3.10). Поле «Дата проведения» является специальной компонентой *DatePicker*, которая позволяет выбрать дату и время. Все поля являются обязательными. Для сохранения данных необходимо нажать на кнопку «Сохранить».

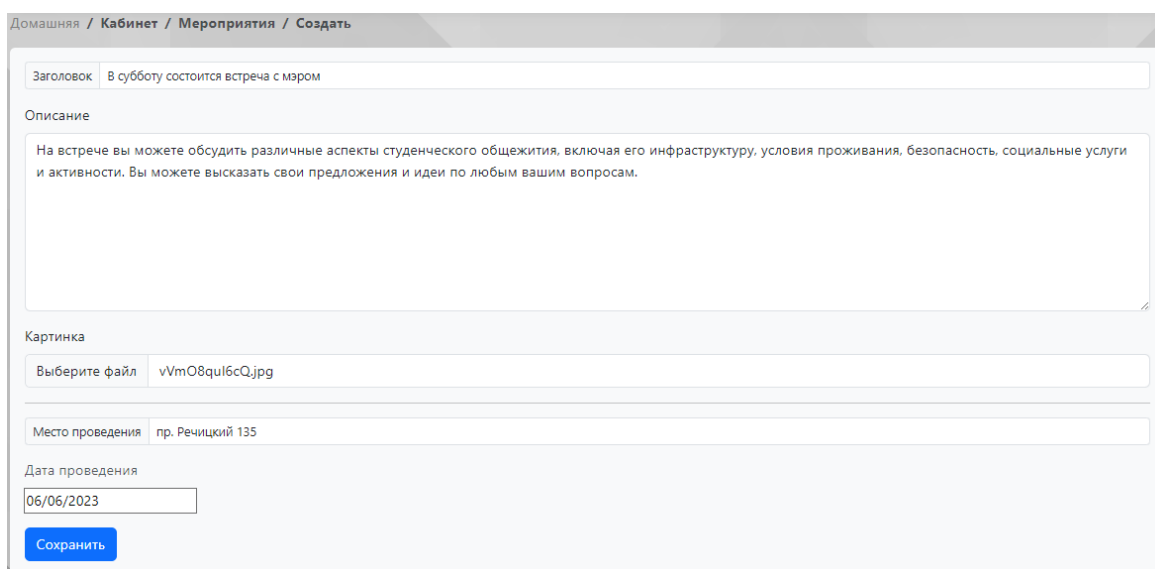


Рисунок 3.10 – Форма для создания событий

Для общения воспитателя со студентами был реализован чат. Чат может создавать только воспитатель. Для этого была реализована страница с полем для ввода названия чата, списком студентов и кнопкой «Создать чат» (рисунок 3.11). Чтобы создать новый чат, воспитателю необходимо ввести названия чата и

выбрать одного или нескольких студентов. После ввода названия чата и выбора студентов, воспитатель нажимает на кнопку «Создать чат». В результате новый чат создается и становится доступным для общения между воспитателем и выбранными студентами.

Название чата

Перед созданием чата, сначала выберите студентов

Создать чат

#		Фамилия	Имя	Отчество	
1	<input type="checkbox"/>	Станиславович	Йоган	Максимов	
2	<input type="checkbox"/>	Алексеевич	Александр	Филиппов	
3	<input type="checkbox"/>	Вадимович	Савва	Смирнов	
4	<input type="checkbox"/>	Сергеевич	Семён	Владимиров	
5	<input type="checkbox"/>	Алексеевич	Жигер	Кириллов	
6	<input type="checkbox"/>	Валериевич	Дан	Лукин	
7	<input type="checkbox"/>	Богданович	Доминик	Комаров	
8	<input type="checkbox"/>	Андреевич	Артём	Рымар	

Рисунок 3.11 – Создание нового чата

После того, как воспитатель создаст чат, студент и воспитатель могут начать общение. Для этого студенту и воспитателю необходимо перейти на страницу *Chat* (рисунок 3.12).

14 этаж

Здравствуйте
Светлана Александровна 24.05.2023 20:34:31

Завтра проведем собрание
Светлана Александровна 24.05.2023 20:34:52

Хорошо
Йоган Максимов 24.05.2023 20:35:21

Уточните время
Йоган Максимов 24.05.2023 20:35:27

Сообщение

Рисунок 3.12 – Вид страницы «Чат»

Для просмотра списка студентов воспитатель может перейти на страницу *StudentsList* (рисунок 3.13). На данной странице расположен поиск по параметрам и таблица с краткой информацией о студентах. Для того, чтобы посмотреть полную информации о конкретном студенте, необходимо нажать на иконку «Список» в правой части строки таблицы.

Чтобы быстро находить нужного студента в общем списке, воспитателю необходимо выбрать нужный параметр, например, имя, и далее ввести в строку «Поиск» нужную подстроку, после чего нажать на кнопку «Поиск».

<input type="radio"/> Имя <input type="radio"/> Отчество <input type="radio"/> Фамилия <input type="radio"/> Форма образования <input type="radio"/> Факультет <input type="radio"/> Группа <input type="radio"/> Баллы <input type="radio"/> Номер зачетки <input type="radio"/> Этаж <input type="radio"/> Блок <input type="radio"/> Комната					
Поиск				Поиск	Показать всех
#	Фамилия	Имя	Отчество	Факультет/Группа	Комната
1	Станиславович	Йоган	Максимов	ЭФ / ИТИ-21	14-6-2
2	Алексеевич	Александр	Филиппов	ФАИС / ИТИ-21	10-2-1
3	Вадимович	Савва	Смирнов	ЭФ / ИТИ-21	11-7-2
4	Сергеевич	Семён	Владимиров	ЭФ / ИТИ-21	6-3-2
5	Алексеевич	Жигер	Кириллов	ЭФ / ИП-42	8-7-1
6	Валериевич	Дан	Лукин	ФАИС / ИТИ-21	10-6-1

Рисунок 3.13 – Список студентов

Страница *ChangeBalls* (рисунок 3.14) предназначена для того, чтобы воспитатель мог изменить количество баллов определенному студенту. Для этого ему необходимо выбрать нужный пункт из таблицы «Деятельность, повышающая или понижающая рейтинг» и ввести рекомендованное количество баллов. После чего нажать на кнопку «Сохранить».

Участие с докладом на СНТК
 30 баллов

п/п	Деятельность, повышающая и понижающая рейтинг	Кол-во баллов
1	<input type="radio"/> Подготовка работы на Республиканский конкурс студенческих НИР или научная публикация	70
2	<input type="radio"/> Участие с докладом на МНТК	50
3	<input checked="" type="radio"/> Участие с докладом на СНТК	30
4	<input type="radio"/> Выполнение обязанностей старосты группы	от 20 до 50
5	<input type="radio"/> Выполнение обязанностей старосты этажа, председателя либо члена студсовета	от 20 до 50
6	<input type="radio"/> Участие в городских культурно-массовых мероприятиях проводимых с участием вуза	от 10 до 50
7	<input type="radio"/> Непосредственное участие в организации культурно-массовых мероприятий проводимых в рамках вуза	50

Рисунок 3.14 – Страница изменения количества баллов

Страница *ImportStudents* (рисунок 3.15) предназначена для добавления новых студентов. Добавление происходит путем импорта файла *Excel* в базу данных. Для того, чтобы осуществить импорт, коменданту необходимо выбрать файл и один из двух пунктов: заменить все данные или добавить новые данные. В первом случае данные студентов, указанные в импортируемом файле, полностью заменят существующие данные в базе данных. Это означает, что все старые данные студентов будут перезаписаны, а предыдущие данные будут помещены в архив. Такой подход полезен, когда необходимо обновить информацию о студентах из нового файла, полностью заменяя все предыдущие данные. Во втором случае данные студентов, указанные в импортируемом файле, будут добавлены к существующим данным в базе данных. Таким образом, новые студенты будут

добавлены к уже существующим записям о студентах.

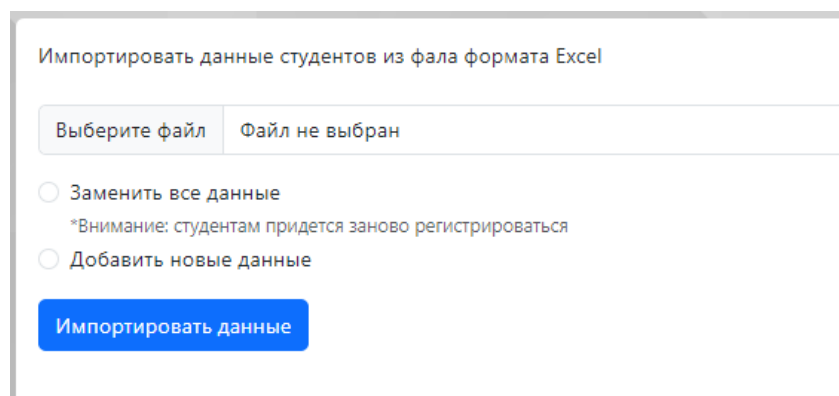


Рисунок 3.15 – Вид страницы «Импорт студентов»

Также комендант может добавить одного студента, используя форму на странице *AddStudent* (рисунок 3.16). Форма состоит из следующих полей: имя, фамилия, отчество, форма образования, номер зачетной книжки, номер этажа, блок, комната, факультет, группа. В данной форме все поля являются обязательными. Для сохранения данных коменданту необходимо нажать на кнопку «Сохранить», после чего форма отправляется на сервер и проходит проверку на валидность.

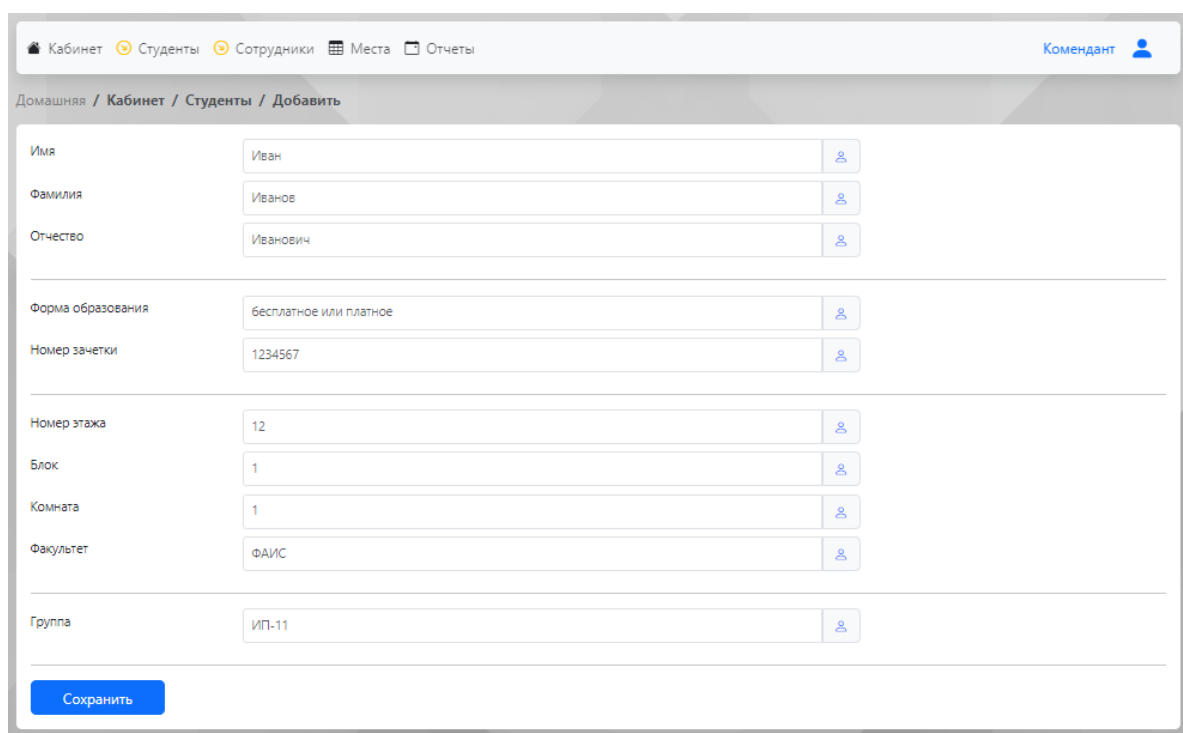
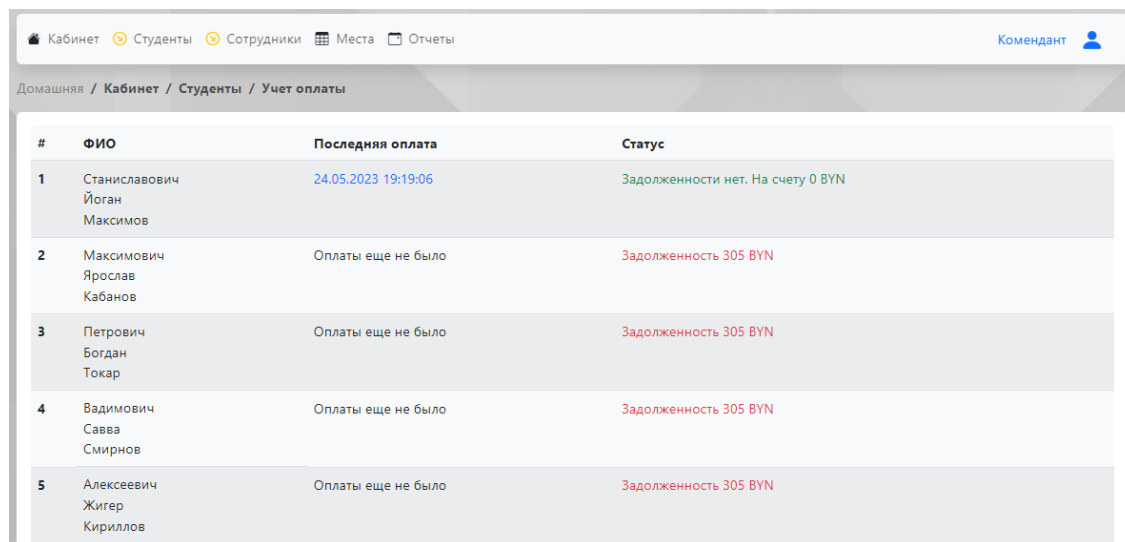


Рисунок 3.16 – Форма «Добавление студента»

Страница *ViewPayHostel* (рисунок 3.17) предназначена для мониторинга оплаты за проживание в общежитии. На данной странице комендант может провести учет оплаты всех студентов, которые обучаются на платной основе. В

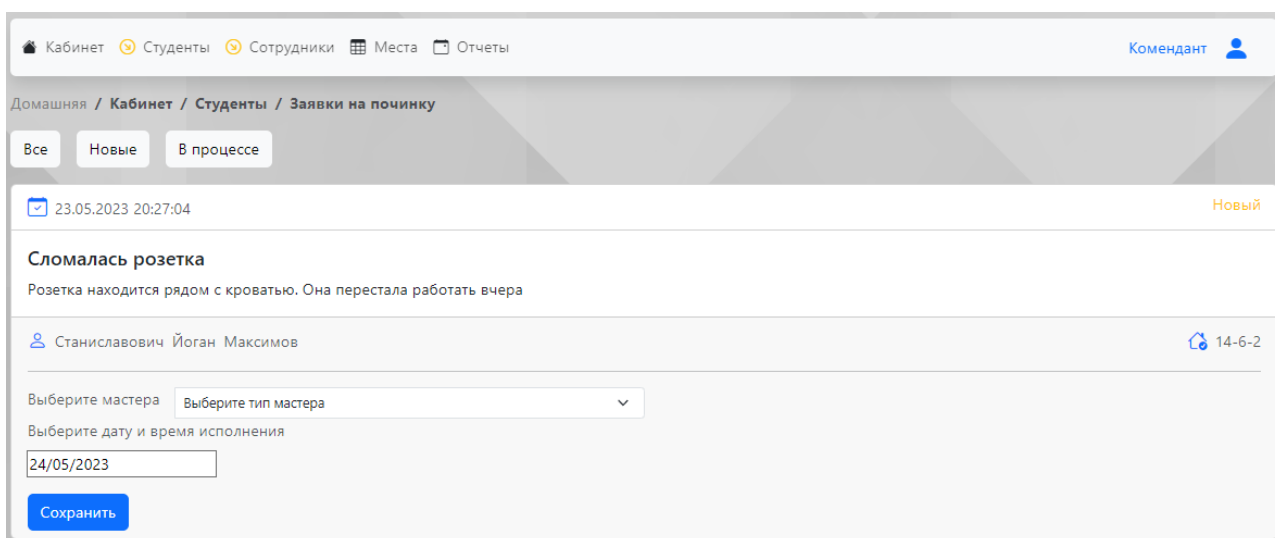
случае, если у студента имеется задолженность – эта задолженность выводится красным цветом. Если задолженность отсутствует, комендант увидит его сумму на счету.



#	ФИО	Последняя оплата	Статус
1	Станиславович Йоган Максимов	24.05.2023 19:19:06	Задолженности нет. На счету 0 BYN
2	Максимович Ярослав Кабанов	Оплаты еще не было	Задолженность 305 BYN
3	Петрович Богдан Токар	Оплаты еще не было	Задолженность 305 BYN
4	Вадимович Савва Смирнов	Оплаты еще не было	Задолженность 305 BYN
5	Алексеевич Жигер Кириллов	Оплаты еще не было	Задолженность 305 BYN

Рисунок 3.17 – Просмотр оплаты за проживание

Страница *RepairView* (рисунок 3.18) предназначена для просмотра заявок на починку оборудования. Для удобства просмотра была предусмотрена фильтрация, с помощью которой комендант может просмотреть все, только новые, или те, которые сейчас имеют статус «В процессе». После того, как студент сделал новую заявку, комендант может принять её. Для этого нужно выбрать соответствующего мастера, дату и время исполнения и нажать на кнопку «Сохранить».



Домашняя / Кабинет / Студенты / Заявки на починку

Все Новые В процессе

23.05.2023 20:27:04 Новый

Сломалась розетка
Розетка находится рядом с кроватью. Она перестала работать вчера

Станиславович Йоган Максимов 14-6-2

Выберите мастера

Выберите дату и время исполнения

Рисунок 3.18 – Вид страницы «Просмотр заявок на починку оборудования»

После того, как комендант примет заявку, она получит статус «В процессе». Для того, чтобы завершить заявку, коменданту необходимо нажать

кнопку «Завершить» (рисунок 3.19).

23.05.2023 20:27:04 В процессе

Сломалась розетка
Розетка находится рядом с кроватью. Она перестала работать вчера

Станиславович Йоган Максимов 14-6-2

Мастер Электрик
Дата и время 24.05.2023 19:45:07

[Завершить](#)

Рисунок 3.19 – Вид заявки со статусом «В процессе»

Страница *AccPlaces* (рисунок 3.20) предназначена для мониторинга занятых мест в общежитии. В случае, если на этаже все комнаты пустые, комендант увидит соответствующее сообщение. Эта функциональность позволяет коменданту быстро и удобно отслеживать текущее состояние заселенности общежития.

[Посмотреть свободные места](#)

[0-0-0](#) - отображены занятые комнаты

1	Этаж пустой
2	Этаж пустой
3	Этаж пустой
4	Этаж пустой
5	5-6-1 5-4-2 5-5-2 5-5-2 Занято: 4
6	6-3-2 6-6-1 Занято: 2
7	7-2-1 Занято: 1
8	8-7-1 8-1-1 Занято: 2

Рисунок 3.20 – Список занятых мест

Страница *FreePlaces* (рисунок 3.21) предназначена для мониторинга свободных мест в общежитии. Комендант может визуально оценить доступные варианты размещения и принять решение о заселении студента на основе актуальных данных.

На данной странице представлены свободные комнаты на соответствующих этажах. Предполагается, что в каждой комнате два места, и в случае, если эти места свободны, они отображаются с помощью постфикса «/2». Эта функциональность значительно облегчает задачу коменданта в отслеживании текущего состояния заселенности общежития. Вместо того, чтобы проверять каждую комнату отдельно, комендант может быстро оценить загрузку каждого этажа и определить доступные места для новых студентов.

Мониторинг свободных мест на странице *FreePlaces* позволяет коменданту легко планировать и управлять размещением студентов в общежитии. Если есть свободные места на определенном этаже, комендант может решить о заселении нового студента в доступную комнату.

14-1-1/2 - отображены комнаты, где свободно два места

Этаж	Комнаты
1	1-1-1/2, 1-1-2/2, 1-2-1/2, 1-2-2/2, 1-3-1/2, 1-3-2/2, 1-4-1/2, 1-4-2/2, 1-5-1/2, 1-5-2/2, 1-6-1/2, 1-6-2/2, 1-7-1/2, 1-7-2/2, 1-8-1/2, 1-8-2/2
2	2-1-1/2, 2-1-2/2, 2-2-1/2, 2-2-2/2, 2-3-1/2, 2-3-2/2, 2-4-1/2, 2-4-2/2, 2-5-1/2, 2-5-2/2, 2-6-1/2, 2-6-2/2, 2-7-1/2, 2-7-2/2, 2-8-1/2, 2-8-2/2
3	3-1-1/2, 3-1-2/2, 3-2-1/2, 3-2-2/2, 3-3-1/2, 3-3-2/2, 3-4-1/2, 3-4-2/2, 3-5-1/2, 3-5-2/2, 3-6-1/2, 3-6-2/2, 3-7-1/2, 3-7-2/2, 3-8-1/2, 3-8-2/2

Рисунок 3.21 – Свободные места в общежитии

Страница *Reports* (рисунок 3.22) предназначен для генерации отчетов, которые являются файлами формата *Excel*. Для того, чтобы сгенерировать отчет, коменданту необходимо нажать одну из нескольких кнопок «Получить», после чего начнется автоматическое скачивание файла с заранее сгенерированным именем.

1. Отчет по количеству баллов

Отчет представляет собой файл формата Excel

Получить

2. Отчет по оплате

Отчет представляет собой файл формата Excel

Получить

Рисунок 3.22 – Вид страницы «*Reports*»

Уровень представления реализован с использованием библиотеки *Bootstrap*. Это позволило значительно упростить и ускорить процесс разработки пользовательского интерфейса, а также уделить больше внимания созданию понятной структуры приложения. Уровень представления выполнен в минималистичном стиле, что позволит пользователю сосредоточиться на содержимом и

функциональности интерфейса, вместо того чтобы заниматься деталями стилизации и внешнего вида.

3.3 Уровень бизнес-логики приложения

Маршруты в *Express* позволяют определить, как приложение должно обрабатывать запросы, направленные на конкретные *URL*-адреса. Каждый маршрут может иметь ассоциированный обработчик (*handler*), который выполняет определенные действия при получении запроса. При создании обработчика маршрута в *Express* нужно указывать соответствующий путь или шаблон *URL*-адреса, который будет обрабатываться этим обработчиком. Когда клиент делает запрос на указанный маршрут, *Express* вызывает соответствующий обработчик и передает ему объекты запроса (*request*) и ответа (*response*), которые позволяют взаимодействовать с клиентом и сервером. Обработчики маршрутов в *Express* представляют собой функции, которые выполняются при обращении к определенному *URL*-адресу. Они могут выполнять различные действия, такие как отправка ответа клиенту, обработка данных из запроса, взаимодействие с базой данных и другие операции.

Уровень бизнес-логики представлен следующими маршрутами:

- *auth.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с авторизацией пользователей;
- *student.route* – маршрут, отвечающий за обработку данных уровня представления, связанного со студентом;
- *mentor.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с воспитателем;
- *news.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с новостями;
- *event.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с мероприятиями;
- *chat.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с чатом;
- *common.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с чатом, воспитателем и со студентом;
- *admin.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с комендантом;
- *repair.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с заявками на починку оборудования.

Обработчики для маршрута *auth.route* приведены в таблице 3.1.

Таблица 3.1 – Обработчики маршрута *auth.route*

Обработчик	Параметры	Описание
1	2	3
<i>router.post('/login')</i>	<i>string login, string password</i>	Возвращает данные пользователя

Продолжение таблицы 3.1

1	2	3
<i>router.post('/registr')</i>	<i>number numberTest, string email</i>	Возвращает сообщение о том, что логин и пароль были отправлены на почту

Обработчики для маршрута *student.route* приведены в таблице 3.2.

Таблица 3.2 – Обработчики маршрута *student.route*

Обработчик	Параметры	Описание
1	2	3
<i>router.post('/import-students')</i>	<i>Array<Student> students, string type</i>	Возвращает сообщение и статус запроса
<i>router.get('/get-info')</i>	—	Возвращает всех студентов из базы данных
<i>router.get('/get-student-id')</i>	<i>string id</i>	Возвращает найденного по <i>id</i> студента
<i>router.post('/change-status-claim')</i>	<i>number numberTest, number ind</i>	Возвращает новый объект <i>Claim</i> с измененным статусом
<i>router.post('/create-claim')</i>	<i>number numberTest, string header, string text, Object mentor, Date dateAndTime</i>	Создает новое замечание и возвращает новый объект <i>Claim</i>
<i>router.get('/get-places')</i>	—	Возвращает массив занятых студентами мест в общежитии
<i>router.post('/add-tech')</i>	<i>number numberTest, string model, string type</i>	Добавляет новую запись в массив <i>Techs</i> и возвращает созданный объект
<i>router.post('/update-info')</i>	<i>number numberTest, string email, string login, string newPassword, string oldPassword</i>	Изменяет личные данные студента и возвращает новый объект <i>Student</i>
<i>router.post('/get-remarks')</i>	<i>number numberTest</i>	Возвращает массив замечаний

Продолжение таблицы 3.2

1	2	3
<i>router.post('/add-student')</i>	<i>string firstName, string secondName, number number floor, number block, number apartament, string faculty, string group</i>	Добавляет нового студента в коллекцию <i>Students</i>
<i>router.post('/pay-hostel')</i>	<i>string receipt, number numberTest, number payment</i>	Добавляет новые данные об оплате общежития
<i>router.post('/get-pay')</i>	<i>number numberTest</i>	Возвращает массив оплаты за проживание в общежитии
<i>router.post('/get-balls')</i>	<i>number numberTest</i>	Возвращает массив баллов

Обработчики для маршрута *mentor.route* приведены в таблице 3.3.

Таблица 3.3 – Обработчики маршрута *mentor.route*

Обработчик	Параметры	Описание
<i>router.post('/info-mentor')</i>	<i>string login</i>	Возвращает объект <i>Mentor</i> , найденный по логину
<i>router.post('/update-info')</i>	<i>number id, string login, string newPassword, string oldPassword</i>	Изменяет личные данные воспитателя и возвращает новый объект <i>Mentor</i>
<i>router.post('/get-students-by-impact')</i>	<i>Object impact</i>	Возвращает массив студентов
<i>router.post('/update-balls')</i>	<i>number numberTest, number num, string summary</i>	Обновляет количество баллов у студента и возвращает обновленные данные

Обработчики для маршрута *news.route* приведены в таблице 3.4.

Таблица 3.4 – Обработчики маршрута *news.route*

Обработчик	Параметры	Описание
1	2	3
<i>router.post('/create-section')</i>	<i>String header, string description, number id</i>	Создает новую секцию новости
<i>router.post('/create-news')</i>	<i>Object mentor</i>	Создает новость и возвращает <i>id</i> созданного объекта

Продолжение таблицы 3.4

1	2	3
<i>router.get('/get-news')</i>	—	Возвращает все новости из базы данных
<i>router.get('/get-news-id')</i>	<i>Number id</i>	Возвращает новость по <i>id</i>

Обработчики для маршрута *event.route* приведены в таблице 3.5.

Таблица 3.5 – Обработчики маршрута *event.route*

Обработчик	Параметры	Описание
<i>router.post('/create-event')</i>	<i>string header, string description, Date dateEvent, string placeEvent</i>	Создает мероприятие и возвращает <i>id</i> созданного объекта
<i>router.get('/get-events')</i>	—	Возвращает все мероприятия из базы данных

Обработчики для маршрута *chat.route* приведены в таблице 3.6.

Таблица 3.6 – Обработчики маршрута *chat.route*

Обработчик	Параметры	Описание
<i>router.post('/create-chat')</i>	<i>Array<string> arrayId, string name, string idMentor</i>	Создает новый чат и возвращает <i>id</i> созданного объекта
<i>router.post('/get-chats')</i>	<i>Array<string> idChats</i>	Возвращает все чаты для переданного массива
<i>router.post('/get-messages')</i>	<i>string id</i>	Возвращает все сообщение для найденного по <i>id</i> чата

Обработчики для маршрута *common.route* приведены в таблице 3.7.

Таблица 3.7 – Обработчики маршрута *common.route*

Обработчик	Параметры	Описание
<i>router.get('/load')</i>	<i>file img</i>	Возвращает картинку с названием <i>img</i>
<i>router.get('/get-cost-hostel')</i>	—	Возвращает текущую стоимость проживания в общежитии

Обработчики для маршрута *admin.route* приведены в таблице 3.8.

Таблица 3.8 – Обработчики маршрута *admin.route*

Обработчик	Параметры	Описание
<i>router.get('/get-admin-info')</i>	<i>number id</i>	Возвращает объект <i>Admin</i> , найденный по <i>id</i> в базе данных
<i>router.get('/get-employee')</i>	–	Возвращает массив воспитателей
<i>router.post('/create-employee')</i>	<i>string secondName,</i> <i>string string firstName,</i> <i>string middleName,</i> <i>number impactFrom,</i> <i>number impactTo,</i> <i>string login</i>	Добавляет нового воспитателя в коллекцию <i>Mentors</i>
<i>router.get('/delete-student')</i>	<i>number id</i>	Удаляет данные студента из базы данных
<i>router.get('/get-payments')</i>	–	Возвращает массив <i>Payments</i> для всех студентов, у которых поле <i>formEducation</i> равно «платное»
<i>router.get('/get-free-places')</i>	–	Возвращает массив свободных мест в общежитии
<i>router.get('/report-balls')</i>	–	Возвращает сгенерированный <i>Excel</i> файл с отчетом по количеству баллов
<i>router.get('/report-pays')</i>	–	Возвращает сгенерированный <i>Excel</i> файл с отчетом по оплате

Обработчики для маршрута *repair.route* приведены в таблице 3.9.

Таблица 3.9 – Обработчики маршрута *repair.route*

Обработчик	Параметры	Описание
1	2	3
<i>router.get('/get-repairs-all')</i>	–	Возвращает все заявки на починку оборудования
<i>router.post('/get-repairs-id')</i>	<i>number id</i>	Возвращает заявку на починку оборудования, найденный по <i>id</i>

Продолжение таблицы 3.9

1	2	3
<code>router.post('/create')</code>	<i>Object user, string header, string description, Object room</i>	Создает новый объект <i>Repair</i> и возвращает созданный объект
<code>router.post('/change-status')</code>	<i>number id</i>	Изменяет статус заявки и возвращает измененный статус

Каждый обработчик содержит дополнительный функционал для обработки ошибок. В случае возникновения ошибки, клиент получит соответствующее уведомление, а результат запроса завершится с кодом 400 или 500.

3.4 Уровень данных в приложении

Уровень данных в приложении обеспечивает связь между приложением и базой данных *MongoDB*. Этот уровень отвечает за работу с данными, их сохранение, извлечение, обновление и удаление. Он представлен в виде моделей. Модели представляют собой схемы и методы для работы с данными в базе данных *MongoDB*. Они названы в соответствии с названиями коллекций, схема которых приведена на рисунке 2.2, только в единственном числе.

Для подключения к базе данных необходимо указать *URL* в окне подключения *MongoDb Compass*. URL для подключения к базе данных *MongoDB* обычно состоит из нескольких компонентов: протокол, хост, порт и данные для аутентификации.

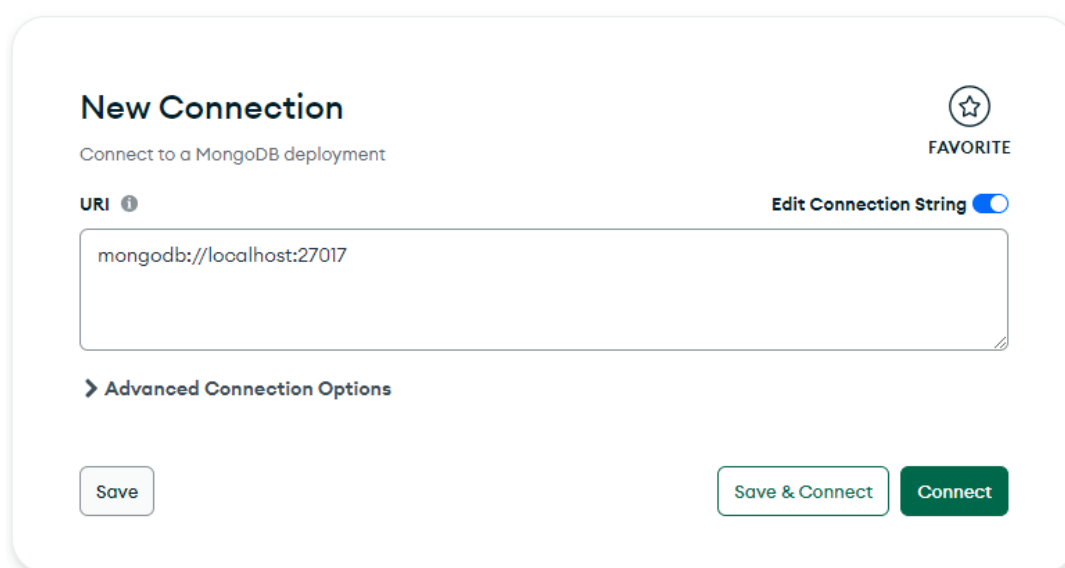


Рисунок 3.23 – Окно подключения к базе данных

Модель *Mongoose* позволяет определить структуру документов, которые будут храниться в коллекции *MongoDB*. Схема определяет поля, типы данных,

валидацию, уникальные ограничения и другие атрибуты для каждого поля. Определение схемы облегчает работу с данными, позволяет контролировать их структуру и обеспечивает целостность данных.

Подключение к базе данных *MongoDB* с использованием *Mongoose* – это процесс установления соединения между приложением и базой данных *MongoDB*, а также настройка параметров для взаимодействия с базой данных. Для подключения была реализована функция *start* (рисунок 3.22), которая использует метод *connect* объекта *mongoose* со строкой подключения.

```
async function start() {  
  try {  
    await mongoose.connect("mongodb://127.0.0.1:27017/diplom_db");  
    app.listen(PORT, () => console.log("Server has been started on port 3000!"));  
  }  
  catch (e) {  
    console.log("Server Error: ", e.message);  
    process.exit(1);  
  }  
}
```

Рисунок 3.24 – Функция для подключения к базе данных

После успешного подключения к базе данных, запускается сервер Express на указанном порту с помощью метода *app.listen()*. Однако, важно учитывать возможность ошибок при запуске сервера. Если сервер не может быть запущен по какой-либо причине, нужно предусмотреть обработку таких ситуаций. В данном случае приложение экстренно завершит работу с кодом 1.

Для реализации дополнительных действий при обработке ошибок при запуске сервера *Express*, можно воспользоваться обработчиком ошибок (*error handler*). В Express это можно сделать путем определения специального маршрута с использованием функции *app.use()* или метода *app.use()*. Вместо простого завершения работы с кодом 1 можно добавить логирование ошибки, чтобы иметь информацию о причине сбоя сервера. Это может быть полезно при последующем анализе и устранении проблемы. Дополнительно, можно реализовать механизм автоматического перезапуска сервера через определенное время после ошибки, чтобы попытаться восстановить его работу.

Данное решение позволяет контролировать поведение приложения в случае ошибки при запуске сервера. Также можно реализовать дополнительные действия при обработке ошибок, например, отправить уведомление или выполнить дополнительные операции перед завершением приложения.

4 ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ ПРОГРАММНОГО КОМПЛЕКСА

4.1 Тестирование пользовательских форм

Тестирование пользовательских форм относится к проверке корректности работы и удобства использования форм в интерфейсе пользователя. Формы представляют собой элементы интерфейса, через которые пользователи вводят и отправляют данные, например, регистрационные данные, почтовые адреса, кредитные карты и т.д. Некорректное функционирование форм или неудобный интерфейс могут привести к ошибкам при заполнении данных, потере информации, негативному опыту пользователей и ухудшению общей эффективности системы. Цель тестирования пользовательских форм заключается в обнаружении и исправлении потенциальных проблем, которые могут возникнуть в процессе взаимодействия пользователя с формой.

В таблице 4.1 проведены пошаговые тесты, которые в конечном итоге должны привести к ожидаемому результату.

Таблица 4.1 – Примеры проводимых тестов

Краткое описание	Шаги по воспроизведению	Ожидаемый результат	Полученный результат
1	2	3	4
Проверка совпадения паролей	1. Войти в личный кабинет воспитателя или студента; 2. Ввести логин или <i>email</i> ; 3. Ввести старый пароль; 4. Ввести новый пароль « <i>password12</i> »; 5. Повторить пароль « <i>password22</i> »; 6. Нажать кнопку «Сохранить».	Приложение должно выводить следующее уведомление: «Пароли должны совпадать»	Приложение выводит уведомление «Пароли должны совпадать»
Проверка правильности <i>email</i>	1. Войти в личный кабинет воспитателя или студента; 2. Заполнить поле <i>email</i> строкой <i>myemail.gmail.com</i> ; 3. Нажать кнопку «Сохранить».	Приложение должно выводить следующее уведомление: «Адрес электронной почты должно содержать символ @»	Приложение выводит уведомление «Адрес электронной почты должно содержать символ @»

Продолжение таблицы 4.1

1	3	4	5
Проверка поля обязательного к заполнению	1. Открыть любую форму; 2. Нажать кнопку «Сохранить».	Приложение должно выводить уведомление: «Заполните это поле»	Приложение выводит уведомление «Заполните это поле»
Проверка на количество символов	1. Войти в личный кабинет воспитателя; 2. Перейти на страницу создания нового события; 3. Заполнить все поля не более, чем 5 символами; 4. Нажать кнопку «Сохранить».	Приложение должно выводить следующее уведомление: «Введите данные в указанном формате»	Приложение выводит уведомление: «Введите данные в указанном формате»
Проверка введенных данных на число	1. Перейти на форму, где требуется ввести число; 2. Попытаться ввести строку.	Приложение должно не позволять вводить не числа	Приложение не позволило вводить не числа

На рисунке 4.1 приведено уведомление, которое появляется при попытке сохранить форму с неправильным повторением пароля. Это уведомление предупреждает пользователя о том, что значения, введенные в поле пароля и поле повторного ввода пароля, не совпадают.

The screenshot shows a web application interface for a 'Кабинет' (Cabinet) of a 'Студент' (Student). At the top, there is a navigation bar with links for 'Заявки' (Applications) and 'Воспитатели' (Tutors). A red error message banner at the top states: 'Пароли должны совпадать' (Passwords must match). Below the banner, a message says: 'Данная форма позволяет изменить одно или несколько полей' (This form allows you to change one or more fields). The form contains several input fields: 'Email', 'Логин' (Login) with the value 'student1', 'Введите старый пароль' (Enter old password), 'Введите новый пароль' (Enter new password), and 'Повторите пароль' (Repeat password). Each password field has a lock icon to toggle visibility. At the bottom of the form is a blue button labeled 'Сохранить' (Save).

Рисунок 4.1 – Вид уведомления при неправильном повторении пароля

На рисунке 4.2 представлен результат проверки введенного *email* на

правильность.

Данная форма позволяет изменить одно или несколько полей

Email:

Логин:

Введите старый пароль:

Введите новый пароль:

Повторите пароль:

Сохранить

! Адрес электронной почты должен содержать символ "@". В адресе "maemail.gmail.com" отсутствует символ "@".

Рисунок 4.2 – Результат проверки *email*

На рисунке 4.3 представлен результат проверки на обязательное поле. Так как поле «Сумма» должно быть числом, при попытке ввести не число, программа не позволит это сделать и покажет подсказку «Сумма должна быть числом».

Дата и время 27.05.2023 19:34:03

Номер квитанции:

Номер квитанции находится сразу

Сумма:

Сохранить

! Заполните это поле.

Сумма должна быть числом

Рисунок 4.3 – Проверка на обязательное поле и на число

На рисунке 4.4 представлен результат проверки на разрешенное количество символов. Данная проверка ограничивает минимальное и максимальное разрешенное количество введенных символов.

Картинка

Выберите файл:

Место проведения:

Дата проведения:

Сохранить

! Введите данные в указанном формате. Минимум 5 и максимум 40 символов

Рисунок 4.4 – Результат проверки на количество символов

Тестирование пользовательских форм имеет важное значение, поскольку формы являются ключевым элементом взаимодействия между пользователем и приложением. Некорректная обработка данных в формах может привести к ошибкам или сбоям в приложении. Тестирование форм помогает выявить и исправить потенциальные проблемы, такие как отсутствие обработки специальных символов, неправильная валидация или некорректная обработка ошибок.

4.2 Модульное тестирование бизнес-логики

Бизнес-логика является основным компонентом программного продукта, отвечающим за обработку данных, принятие решений и выполнение бизнес-правил. Тестирование бизнес-логики позволяет убедиться, что она работает правильно, соответствует ожидаемым результатам и выполняет необходимые проверки и валидации данных. Оно помогает выявить потенциальные ошибки, пропуски или некорректные реализации в бизнес-логике еще до того, как они повлияют на работу приложения в целом. Это позволяет оперативно исправить проблемы и предотвратить возможные сбои или неправильное поведение системы.

Модульное тестирование бизнес-логики с помощью *Jest* позволяет повысить качество и надежность программного продукта, упростить разработку и поддержку кода, а также предоставить документацию и понимание ожидаемого поведения системы. Тесты бизнес-логики также могут служить своего рода документацией, позволяя разработчикам и другим заинтересованным сторонам лучше понять ожидаемое поведение и требования к функциональности. Тесты могут служить наглядным примером использования бизнес-логики и помогать при анализе и отладке проблемных ситуаций. Модульные тесты позволяют проверить различные сценарии использования бизнес-логики и убедиться, что она работает должным образом. Это способствует повышению качества и надежности программного продукта.

В таблице 4.2 приведен набор модульных тестов для обработчиков `auth.route`, отвечающий за обработку запросов, связанных с авторизацией пользователей.

Таблица 4.2 – Набор модульных тестов для `auth.route`

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
1	2	3	4	5
<code>router.post('/auth/login')</code>	Проверка корректного логина	<code>login: login123</code> <code>password: pass123</code>	Сообщение «Такого логина не существует» и код 400	Сервер возвращает сообщение «Такого логина не существует» и код 400

Продолжение таблицы 4.2

1	2	3	4	5
<i>router.post</i> (<i>'/auth/login'</i>)	Проверка корректного пароля	<i>login: login123</i> <i>password: pas</i>	Сообщение «Неверный пароль» и код 400	Сервер воз- вращает со- общение «Неверный логин» и код 400
<i>router.post</i> (<i>'/auth/login'</i>)	Проверка корректного входа	<i>login: student1</i> <i>password: stu-</i> <i>dent1</i>	Сообщение «Вход вы- полнен успешно» и код 200	Сервер воз- вращает со- общение «Вход вы- полнен успешно» и код 200
<i>router.post</i> (<i>'/auth/registr'</i>)	Проверка корректного номера зачет- ной книжки	<i>numberTest: 123</i> <i>email:</i> <i>mail@gmail.com</i>	Сообщение «Неверный номер за- четной книжки» и код 400	Сервер воз- вращает со- общение «Неверный номер за- четной книжки» и код 400
<i>router.post</i> (<i>'/auth/registr'</i>)	Проверка су- ществования аккаунта	<i>numberTest:</i> <i>46753842</i> <i>email:</i> <i>mail@gmail.com</i>	Сообщение «У вас уже есть акка- унт»	Сервер воз- вращает со- общение «У вас уже есть аккаунт»
<i>router.post</i> (<i>'/auth/registr'</i>)	Проверка корректного email	<i>numberTest:</i> <i>46753842</i> <i>email:</i> <i>email.gmail.com</i>	Сообщение «Неверный <i>email</i> » и код 400	Сервер воз- вращает со- общение «Неверный <i>email</i> »
<i>router.post</i> (<i>'/auth/registr'</i>)	Проверка успешной ре- гистрации	<i>numberTest:</i> <i>1583657</i> <i>email:</i> <i>pavel@gmail.com</i>	Сообщение «Сообще- ние отпра- влено на по- чту» и код 200	Сервер воз- вращает сообщение «Сообще- ние отпра- влено на по- чту» и код 200

На картинке 4.5 приведены результаты исполнения набора тестов из таблицы 4.2.

```

PASS src/tests/auth.test.ts (9.895 s)
  Тестирование маршрута /auth/login
    ✓ Проверка корректного логина (8 ms)
    ✓ Проверка корректного пароля (1 ms)
    ✓ Проверка корректного входа (1 ms)
  Тестирование маршрута /auth/registr
    ✓ Проверка корректного номера зачетной книжки (3 ms)
    ✓ Проверка существования аккаунта (2 ms)
    ✓ Проверка корректного email (1 ms)
    ✓ Проверка успешной регистрации (1 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        10.447 s, estimated 11 s
Ran all test suites.

```

Рисунок 4.5 – Результаты тестирования *auth.route*

В таблице 4.3 приведен набор модульных тестов для обработчиков *mentor.route*, отвечающий за обработку запросов, связанных с воспитателем.

Таблица 4.3 – Набор модульных тестов для *mentor.route*

Название маршрута	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
1	2	3	4	5
<i>router.post</i> ('/update-info')	Проверка успешного обновления данных воспитателя	<i>id: 2, login: mentor1, newPassword: newPass, oldPassword: oldPass</i>	Объект Mentor, сообщение «Данные обновлены» и код 200	Сервер возвращает объект Mentor, сообщение «Данные обновлены» и код 200
<i>router.post</i> ('/update-info')	Проверка совпадения паролей	<i>Id: 2, newPass: newPassword, oldPassword: oldPass</i>	Сообщение «Пароль должен совпадать со старым паролем»	Сервер возвращает сообщение «Пароль должен совпадать со старым паролем»
<i>router.post</i> ('/update-info')	Проверка существования логина	<i>id: 2 login: mentor2</i>	Сообщение «Такой логин уже существует» и код 400	Сервер возвращает сообщение «Такой логин уже существует» и код 400

Продолжение таблицы 4.3

1	2	3	4	5
<i>router.post</i> (<i>'/update-balls'</i>)	Проверка успешного обновления данных	<i>numberTest</i> : 4767246, <i>num</i> : 30, <i>summary</i> : Участие с докладом на МНТК	Сообщение «Данные обновлены» и код 200	Сервер возвращает сообщение «Данные обновлены» и код 200
<i>router.post</i> (<i>'/update-balls'</i>)	Проверка корректности данных	<i>numberTest</i> : 4767246, <i>num</i> : 3ст, <i>summary</i> : Участие с докладом на МНТК	Сообщение «Количество должно быть числом» и код 400	Сервер возвращает сообщение «Количество должно быть числом» и код 400

На картинке 4.6 приведены результаты исполнения набора тестов из таблицы 4.3.

```

PASS src/tests/mentor.test.ts (12.01 s)
  Тестирование маршрута /mentor/update-info
    ✓ Проверка успешного обновления данных воспитателя (12 ms)
    ✓ Проверка совпадения паролей (3 ms)
    ✓ Проверка существования логина (2 ms)
  Тестирование маршрута /mentor/update-balls
    ✓ Проверка успешного обновления данных (5 ms)
    ✓ Проверка корректности данных (3 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:  0 total
Time:        12.692 s
Ran all test suites.

```

Рисунок 4.6 – Результаты тестирования *mentor.route*

В таблице 4.4 приведен набор модульных тестов для обработчиков *news.route*, отвечающий за обработку запросов, связанных с новостями.

Таблица 4.4 – Набор модульных тестов для *news.route*

Название маршрута	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
1	2	3	4	5
<i>router.post</i> (<i>'/create-section'</i>)	Проверка успешного создания секции новости	Header: строка из 25 символов, <i>id</i> : 3	Сообщение «Новость создана» и код 200	Сервер возвращает сообщение «Новость создана» и код 200

Продолжение таблицы 4.4

1	2	3	4	5
<i>router.post</i> (<i>'/create-section'</i>)	Проверка валидного количества знаков	<i>Header</i> : строка из 3 символов, <i>description</i> : строка из 1000 символов, <i>id</i> : 3	Сообщение «Не верная длина заголовка или описания»	Сервер возвращает сообщение «Не верная длина заголовка или описания»
<i>router.post</i> (<i>'/create-section'</i>)	Проверка на пустые поля	<i>Header</i> : строка из 0 символов, <i>desc</i> : строка из 0 символов, <i>id</i> : 3	Сообщение «Поля обязательные к заполнению»	Сервер возвращает сообщение «Поля обязательные к заполнению»
<i>router.post</i> (<i>'/create-news'</i>)	Проверка успешного создания новости	<i>Object Mentor</i>	Сообщение «Новость создана», <i>id</i> новости	Сервер возвращает сообщение «Новость создана», <i>id</i> новости
<i>router.get</i> (<i>'/get-news-id'</i>)	Проверка успешного возвращения новости по <i>id</i>	<i>id</i> : 3	Найденная новость по переданному <i>id</i>	Сервер вернул новость по переданному <i>id</i>

На картинке 4.7 приведены результаты исполнения набора модульных тестов из таблицы 4.4.

```

PASS src/tests/news.test.ts (10.423 s)
  Тестирование маршрута /mentor/create-section
    ✓ Проверка успешного создания секции новости (7 ms)
    ✓ Проверка валидного количества знаков (2 ms)
    ✓ Проверка на пустые поля (3 ms)
  Тестирование маршрута /mentor/create-news
    ✓ Проверка успешного создания новости (2 ms)
  Тестирование маршрута /mentor/get-news-id
    ✓ Проверка успешного возвращения новости по id (2 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        10.989 s, estimated 12 s
Ran all test suites.

```

Рисунок 4.7 – Результаты тестирования *news.route*

Express основан на концепции маршрутизации, и правильное

функционирование маршрутов является критически важным для веб-приложений. Модульное тестирование маршрутов *Express* с помощью *Jest* позволяет удостовериться, что они работают согласно ожиданиям и возвращают правильные ответы на запросы. Тесты позволили нам проверить различные сценарии и условия работы приложения, включая различные *HTTP*-методы, передачу параметров, обработку ошибок и другие аспекты, а также обнаружить и исправить потенциальные уязвимости, неправильное поведение или сбои в работе приложения.

Тестирование помогло нам повысить стабильность и надежность приложения, поскольку мы смогли выявить и исправить ошибки и проблемы до их попадания в продуктивную среду. Тесты также обеспечили нам уверенность в том, что приложение работает согласно заданным требованиям и ожиданиям. Благодаря проведенным тестам мы смогли улучшить качество нашего приложения, снизить риск возникновения сбоев и проблем в работе, а также повысить удовлетворенность пользователей.

4.3 Интеграционное тестирование бизнес-логики

Интеграционное тестирование – это процесс проверки взаимодействия и корректной работы различных компонентов или модулей программного продукта в единой системе. Оно выполняется с целью обнаружения ошибок, связанных с интеграцией компонентов, перед тем как система будет введена в эксплуатацию.

Основная задача интеграционного тестирования заключается в проверке взаимодействия компонентов в рамках системы и обнаружении возможных ошибок, которые могут возникнуть при интеграции. В процессе интеграционного тестирования проверяются такие аспекты, как передача данных между компонентами, правильность вызова функций и методов, обработка возвращаемых значений, а также согласованность и целостность данных между различными компонентами системы.

Интеграционное тестирование выполняется на различных уровнях, начиная от интеграции отдельных модулей до тестирования системы в целом. Это позволяет обнаружить проблемы как на ранних стадиях разработки, так и перед введением системы в эксплуатацию, что в свою очередь способствует повышению качества и надежности программного продукта.

В отличие от модульного тестирования, интеграционное тестирование фокусируется на проверке взаимодействия между различными компонентами и интерфейсами системы. Оно предназначено для проверки совместной работы этих компонентов и обеспечения корректного функционирования системы в целом.

В таблице 4.5 представлен интеграционный тест «Изменение логина и пароля новым студентом». Данный тест проверяет успешное добавление нового студента в базу данных комендантом, получение этих данных, вход нового студента в систему по сгенерированному логину и паролю, изменение логина и пароля и затрагивает роли «Студент» и «Комендант». Ожидаемый результат: новый студент успешно войдет в систему, используя новый логин и пароль.

Таблица 4.5 – Интеграционный тест «Изменение логина и пароля новым студентом»

Название маршрута	Описание шага	Исходные данные
<i>router.post('/add-student')</i>	Добавление нового студента в базу данных	<i>firstName: Иван, secondName: Иванов, middleName: Иванович, floor: 10, block: 3, apartment: 1, faculty: ФАИС, group: ИП-42</i>
<i>router.get('/get-student-id')</i>	Получение данных нового студента	<i>numberTest: 3657823</i>
<i>router.post('/login')</i>	Вход нового студента в систему	<i>login: fth5392jk, password: yett2u434</i>
<i>router.post('/update-info')</i>	Изменение логина и пароля	<i>numberTest: 3657823, login: newLogin, newPassword: newPassword</i>
<i>router.post('/login')</i>	Вход, используя новый логин и пароль	<i>login: newLogin, password: newPassword</i>

На рисунке 4.8 представлен результат интеграционного теста. Данный результат полностью совпадает с ожидаемым.

```

PASS src/tests/int1.test.ts (9.936 s)
  Добавление нового студента
    ✓ Шаг 1. Добавление нового студента в базу данных (9 ms)
    ✓ Шаг 2. Получение данных нового студента (1 ms)
    ✓ Шаг 3. Вход нового студента в систему (1 ms)
    ✓ Шаг 4. Изменение логина и пароля (2 ms)
    ✓ Шаг 5. Вход, используя новый логин и пароль (1 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        10.51 s, estimated 11 s
Ran all test suites.

```

Рисунок 4.8 – Результат интеграционного теста «Изменение логина и пароля новым студентом»

В таблице 4.6 представлен интеграционный тест «Создание новостей». Данный тест проверяет успешное создание новостей воспитателем, а также просмотр созданной новости студентом. Ожидаемый результат: успешный просмотр созданной новости студентом.

Таблица 4.6 – Интеграционный тест «Создание новостей»

Название маршрута	Описание шага	Исходные данные
<i>router.post</i> (<i>'/create-news'</i>)	Создание новости и получение <i>id</i>	<i>Object <Mentor></i>
<i>router.get</i> (<i>'/create-section'</i>)	Создание новой секции новости	<i>header</i> : строка из 25 символов, <i>description</i> : строка из 200 символов, <i>id</i> : 10
<i>router.get</i> (<i>'/get-news'</i>)	Получение массива новостей	–
<i>router.post</i> (<i>'/get-news-id'</i>)	Получение студентом новости по <i>id</i>	<i>id</i> : 10

На рисунке 4.9 представлен результат интеграционного теста. Данный результат полностью совпадает с ожидаемым.

```

PASS src/tests/int2.test.ts (9.755 s)
  Создание новостей
    ✓ Шаг 1. Создание новости и получение id (6 ms)
    ✓ Шаг 2. Создание новой секции новости (2 ms)
    ✓ Шаг 3. Получение студентом массива новостей (1 ms)
    ✓ Шаг 4. Получение студентом новости по id (4 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        10.24 s
Ran all test suites.

```

Рисунок 4.9 – Результат интеграционного теста «Создание новостей»

В таблице 4.7 представлен интеграционный тест «Создание и изменение статуса заявки на починку бытового оборудования». Данный тест проверяет успешное создание новой заявки студентом, изменение статуса заявки комендантом, а также просмотр этой заявки. Ожидаемый результат: студент получит заявку со статусом 2 – «Завершен».

Таблица 4.7 – Интеграционный тест «Создание и изменение статуса заявки на починку бытового оборудования»

Название маршрута	Описание шага	Исходные данные
1	2	3
<i>router.post</i> (<i>'/create'</i>)	Создание заявки. Статус заявки равен 0 – «Новое»	<i>header</i> : строка из 15 символов, <i>description</i> : строка из 45 символов, <i>room</i> : <i>IRoom</i>

Продолжение таблицы 4.7

1	2	3
<code>router.get('/get-repairs-all')</code>	Получение массива заявок	—
<code>router.get('/get-repairs-id')</code>	Получение заявки по id	<i>id: 5</i>
<code>router.post('/change-run')</code>	Изменение комендантом статуса заявки на 1 – «В процессе»	<i>Master: Электрик, id: 5</i>
<code>router.post('/change-status-to-2')</code>	Изменение комендантом статуса заявки на 2 – «Завершен»	<i>id: 5</i>
<code>router.get('/get-repairs-id')</code>	Получение заявки по id. Статус заявки должен быть равен 2	<i>id: 5</i>

На рисунке 4.10 представлен результат интеграционного теста. Данный результат полностью совпадает с ожидаемым.

```

PASS src/tests/int3.test.ts (9.607 s)
  Создание и изменение статуса заявки на починку бытового оборудования
    ✓ Шаг 1. Создание заявки. Статус заявки равен 0 – «Новое» (5 ms)
    ✓ Шаг 2. Получение массива заявок (1 ms)
    ✓ Шаг 3. Получение заявки по id (1 ms)
    ✓ Шаг 4. Изменение комендантом статуса заявки на 1 – «В процессе» (1 ms)
    ✓ Шаг 5. Изменение комендантом статуса заявки на 2 – «Завершен» (2 ms)
    ✓ Шаг 6. Получение заявки по id. Статус заявки должен быть равен 2 – «Завершен» (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:  0 total
Time:        10.088 s
Ran all test suites.

```

Рисунок 4.10 – Результат интеграционного теста «Создание и изменение статуса заявки на починку бытового оборудования»

Интеграционное тестирование проверяет взаимодействие и корректную работу различных компонентов или модулей программного продукта в единой системе. Проведя интеграционное тестирование, мы проверили, что все компоненты программного продукта корректно взаимодействуют друг с другом. Данные передаются между компонентами системы без искажений и потерь информации.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ

5.1 Технико-экономическое обоснование целесообразности разработки программного продукта

Программный продукт, создаваемый в ходе выполнения дипломной работы, представляет собой средство для улучшения взаимодействия воспитателя и коменданта со студентом. Для этого воспитателю предоставляются средства для ведения ленты новостей и событий, создания различных замечаний. Коменданту предоставляется возможность вести учет оплаты и мест в общежитии. Для быстрого взаимодействия воспитателя и студента в приложении имеется чат.

Среди аналогов данного программного продукта можно выделить различные интернет-страницы общежитий, на которых указаны данные воспитателей, различные фотографии, режим работы, карта с подробным маршрутом и прочее. Однако функциональные возможности данных аналогов в целом сводятся к предоставлению общей информации, часть которой так же затрагивается в реализуемом решении, почти полностью игнорируя взаимодействие воспитателя и коменданта со студентом.

Основной экономический эффект достигается за счёт экономии времени воспитателя и коменданта на составление новостных бумажных плакатов и ведение *excel* таблиц для мониторинга оплаты и мест в общежитии. Однако стоит отметить, что экономический эффект является не единственным важным мотиватором для создания программного продукта. Другим важным аспектом является желание предоставить студентам и сотрудникам общежития удобную платформу для взаимодействия, тем самым ускорив работу воспитателей и коменданта.

5.2 Расчёт общей трудоемкости разработки программного обеспечения

Общий объём трудоемкости разработки системы по автоматизации жилого фонда студенческого общежития (V_0) определяется исходя из объёма функций, реализуемых программой, по каталогу функций ПО по формуле (5.1):

$$V_0 = \sum_{i=1}^n V_i \quad (5.1)$$

где V_i – объём отдельной функции ПО;

n – общее число функций.

Уточненный объём ПО (V_y) определяется по формуле (5.2):

$$V_y = \sum_{i=1}^n V_{yi} \quad (5.2)$$

где V_{yi} – уточненный объем отдельной функции ПО в строках исходного кода (LOC).

Результаты расчетов представлены в таблице Б.1.

Разработанное в ходе выполнения дипломной работы приложение относится к третьей категории сложности.

На основании принятого к расчёту (уточнённого) объёма (V_y) и категории сложности ПО определяется нормативная трудоемкость ПО (T_n) выполняемых работ, которая приведена в таблице 5.1.

Таблица 5.1 – Нормативная трудоёмкость на разработку ПО (T_n)

Уточнённый объем, V_y	3-я категория сложности ПО	Номер нормы
6980	290	55

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (K_c) (формула 5.3).

$$K_c = 1 + \sum_{i=1}^n K_i \quad (5.3)$$

где K_i – коэффициент соответствующий степени повышения сложности;
 n – количество учитываемых характеристик.

$$K_c = 1 + 0,07 = 1,07$$

Новизна разработанного ПО определяется путем экспертной оценки данных, полученных при сравнении характеристик разрабатываемого ПО с имеющимися аналогами. Влияние фактора новизны на трудоёмкость учитывается путем умножения нормативной трудоёмкости на соответствующий коэффициент, учитывающий новизну ПО (K_n). Разработанная программа обладает категорией новизны В, а значение $K_n = 0,63$.

В данном программном комплексе используется от 40% до 60% стандартных модулей, что соответствует значению коэффициента $K_t = 0,65$.

Приложение разработано на языке JavaScript, что соответствует коэффициенту функционирования в глобальных сетях, учитывающему средства разработки ПО, $K_{yp} = 0,65$.

Значения коэффициентов удельных весов трудоёмкости стадий разработки ПО в общей трудоемкости ПО определяются с учётом установленной категории новизны ПО. При этом сумма значений коэффициентов удельных весов всех стадий в общей трудоёмкости равна единице [10].

Значения коэффициентов приведены в таблице Б.2.

Нормативная трудоемкость ПО (T_n) выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: повышения сложности ПО (K_c),

учитывающих новизну ПО (K_n), учитывающих степень использования стандартных модулей (K_m), средства разработки ПО ($K_{y.p}$) и определяются по формулам:

- для стадии ТЗ рассчитывается по формуле (5.4):

$$T_{y.т.з} = T_n \cdot K_{т.з} \cdot K_c \cdot K_n \cdot K_{y.p}; \quad (5.4)$$

- для стадии ЭП рассчитывается по формуле (5.5):

$$T_{y.э.п} = T_n \cdot K_{э.п} \cdot K_c \cdot K_n \cdot K_{y.p}; \quad (5.5)$$

- для стадии ТП рассчитывается по формуле (5.6):

$$T_{y.т.п} = T_n \cdot K_{т.п} \cdot K_c \cdot K_n \cdot K_{y.p}; \quad (5.6)$$

- для стадии РП рассчитывается по формуле (5.7):

$$T_{y.p.п} = T_n \cdot K_{p.п} \cdot K_c \cdot K_n \cdot K_t \cdot K_{y.p}; \quad (5.7)$$

- для стадии ВН рассчитывается по формуле (5.8):

$$T_{y.в.н} = T_n \cdot K_{в.н} \cdot K_c \cdot K_n \cdot K_{y.p}, \quad (5.8)$$

где $K_{т.з}$, $K_{э.п}$, $K_{т.п}$, $K_{p.п}$ и $K_{в.н}$ – значения коэффициентов удельных весов;

Коэффициенты K_n , K_c и $K_{y.p}$ вводятся на всех стадиях разработки, а коэффициент K_m вводится только на стадии РП.

$$T_{y.т.з} = 290 \cdot 0,08 \cdot 0,65 \cdot 0,63 \cdot 1,3 = 10 \text{ чел.-дн},$$

$$T_{y.э.п} = 290 \cdot 0,19 \cdot 0,65 \cdot 0,63 \cdot 1,3 = 24 \text{ чел.-дн},$$

$$T_{y.т.п} = 290 \cdot 0,28 \cdot 0,65 \cdot 0,63 \cdot 1,3 = 36 \text{ чел.-дн},$$

$$T_{y.p.п} = 290 \cdot 0,34 \cdot 0,65 \cdot 0,63 \cdot 0,77 \cdot 1,3 = 28 \text{ чел.-дн},$$

$$T_{y.в.н} = 290 \cdot 0,11 \cdot 0,65 \cdot 0,63 \cdot 1,3 = 14 \text{ чел.-дн}.$$

Общая трудоёмкость разработки ПО (T_o) определяется суммированием нормативной (скорректированной) трудоёмкости ПО по стадиям разработки и рассчитывается по формуле (5.9):

$$T_o = \sum_{i=1}^n T_{yi}, \quad (5.9)$$

где T_{yi} – нормативная (скорректированная) трудоёмкость разработки ПО на i -й

стадии, чел.-дн.;

n – количество стадий разработки.

$$T_o = 10 + 24 + 36 + 28 + 14 = 112 \text{ чел.-дн.}$$

Результаты расчётов по определению нормативной и скорректированной трудоёмкости ПО по стадиям разработки и общую трудоёмкость разработки ПО (T_o) представлены в таблице Б.3.

5.3 Расчёт объёма капитальных вложений при создании программного продукта

В общем виде совокупность капитальных вложений в проект может быть рассчитана следующим образом формуле (5.10):

$$K = K_{об} + K_{на} - K_{л} + K_{пр}, \quad (5.10)$$

где $K_{об}$ – стоимость устанавливаемого оборудования, руб.;

$K_{на}$ – недоамортизированная часть стоимости демонтируемого оборудования, руб.;

$K_{л}$ – ликвидационная стоимость демонтируемого оборудования, руб.;

$K_{пр}$ – стоимость приобретенных программных продуктов, руб.

В стоимость оборудования ($K_{об}$) входят расходы на его приобретение по прейскурантам, прайс-листам и другим источникам, а также расходы на приёмку и хранение оборудования (примерно 2 % от стоимости). Также в стоимость оборудования включаются транспортно-заготовительные расходы, т. е. расходы по его доставке и стоимость монтажа устанавливаемого оборудования. Как правило, их принимают в размере 5–10 % от стоимости нового оборудования. Прейскурант на приобретение оборудования приведен в таблице Б.4.

$$K_{об} = 2150 \cdot (1 + 0,02 + 0,05) = 2300,5 \text{ руб.,}$$

$$K = 2300,5 + 0 - 0 + 0 = 2300,5 \text{ руб.}$$

5.4 Расчёт текущих затрат разработки программного продукта

В состав затрат на разработку ПП входят следующие статьи расходов:

– затраты труда на создание ПП (затраты по основной, дополнительной заработной плате и соответствующие отчисления) ($Z_{тр}$);

– затраты на изготовление эталонного экземпляра ($Z_{эт}$);

– затраты на технологию (затраты на освоение и приобретение программных продуктов, используемых при разработке ПП) ($Z_{тех}$);

– затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) ($Z_{м.в}$);

- затраты на материалы (информационные носители) ($Z_{\text{мат}}$);
- общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) ($Z_{\text{общ.пр}}$);

- непроизводственные (коммерческие) расходы (затраты, связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) ($Z_{\text{непр}}$) [10].

Суммарные затраты на разработку ПО (Z_p) определяются по формуле (5.11):

$$Z_p = Z_{\text{тр}} + Z_{\text{эт}} + Z_{\text{тех}} + Z_{\text{м.в}} + Z_{\text{мат}} + Z_{\text{общ.пр}} + Z_{\text{непр}}. \quad (5.11)$$

Параметры расчета затрат на разработку ПО приведены в таблице А.4.

Расходы на оплату труда разработчиков с отчислениями ($Z_{\text{тр}}$) определяются по формуле (5.12):

$$Z_{\text{тр}} = ЗП_{\text{осн}} + ЗП_{\text{доп}} + ОТЧ_{\text{зп}}, \quad (5.12)$$

где $ЗП_{\text{осн}}$ – основная заработная плата разработчиков, руб.;

$ЗП_{\text{доп}}$ – дополнительная заработная плата разработчиков, руб.;

$ОТЧ_{\text{зп}}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная заработная плата разработчиков рассчитывается по формуле (5.13):

$$ЗП_{\text{осн}} = C_{\text{ср.час}} \cdot T_o \cdot K_{\text{ув}}, \quad (5.13)$$

где $C_{\text{ср.час}}$ – средняя часовая тарифная ставка, руб./ч;

T_o – общая трудоёмкость разработки, чел.-ч;

$K_{\text{ув}}$ – коэффициент доплаты стимулирующего характера ($K_{\text{ув}} = 1,8$).

Средняя часовая тарифная ставка определяется по формуле (5.14):

$$C_{\text{ср.час}} = \frac{\sum_i C_{\text{чи}} \cdot n_i}{\sum_i n_i}, \quad (5.14)$$

где $C_{\text{чи}}$ – часовая тарифная ставка разработчика i -й категории, руб./ч;

n_i – количество разработчиков i -й категории.

Часовая тарифная ставка ($C_{\text{чи}}$) вычисляется путём деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени 168 ч ($F_{\text{мес}}$) и рассчитывается по формуле (5.15):

$$C_{\text{чи}} = \frac{C_{\text{ми}} \cdot K_{\text{к}}}{F_{\text{мес}}} = \frac{C_{\text{м1}} \cdot T_{\text{ки}} \cdot K_{\text{к}}}{F_{\text{мес}}}, \quad (5.15)$$

где $C_{\text{ми}}$ – месячная тарифная ставка;

$C_{м1}$ – базовая ставка;

T_{ki} – тарифный коэффициент установленного тарифного разряда,

K_k – корректирующий коэффициент базовой ставки.

Общая трудоёмкость разработки (T_o) в чел.-ч получается с помощью перевода общей трудоёмкости разработки в чел.-дн., в восьмичасовой рабочий день путём домножения на 8 рабочих часов в день.

$$C_{\text{ср.час}} = C_q = \frac{228 \cdot 1,57 \cdot 2,08}{168} = 4,43 \text{ руб.},$$

$$ЗП_{\text{осн}} = 4,43 \cdot 112 \cdot 8 \cdot 1,8 = 7144,704 \text{ руб.}$$

Дополнительная заработная плата рассчитывается по формуле (5.16):

$$ЗП_{\text{доп}} = \frac{ЗП_{\text{осн}} \cdot H_{\text{доп}}}{100}, \quad (5.16)$$

где $H_{\text{доп}}$ – норматив на дополнительную заработную плату разработчиков.

$$ЗП_{\text{доп}} = \frac{7144,704 \cdot 15}{100} = 1071,71 \text{ руб.}$$

Отчисления от основной и дополнительной заработной платы рассчитываются по формуле (5.17):

$$ОТЧ_{\text{с.н}} = \frac{(ЗП_{\text{осн}} + ЗП_{\text{доп}}) \cdot H_{\text{з.п}}}{100}, \quad (5.17)$$

где $H_{\text{з.п}}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($H_{\text{з.п}} = 34,6\%$).

$$ОТЧ_{\text{с.н}} = \frac{(7144,704 + 1071,71) \cdot 34,6}{100} = 2842,88 \text{ руб.},$$

$$З_{\text{тр}} = 7144,704 + 1071,71 + 2842,88 = 11059,29 \text{ руб.}$$

Затраты машинного времени ($З_{\text{м.в}}$) определяются по формуле (5.18):

$$З_{\text{м.в}} = C_q \cdot K_m \cdot t_{\text{эвм}}, \quad (5.18)$$

где C_q – стоимость 1 ч машинного времени, руб./ч;

K_m – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ ($K_m = 1$);

$t_{\text{ЭВМ}}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

Стоимость 1 машино-часа определяется по формуле (5.19):

$$C_{\text{ч}} = \frac{3\Pi_{\text{об}} + 3_{\text{ар}} + 3_{\text{ам}} + 3_{\text{э.п}} + 3_{\text{в.м}} + 3_{\text{т.р}} + 3_{\text{пр}}}{F_{\text{ЭВМ}}}, \quad (5.19)$$

где $3\Pi_{\text{об}}$ – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год ($3\Pi_{\text{об}} = 0$, т.к. обслуживающий персонал отсутствует);

$3_{\text{ар}}$ – стоимость аренды помещения под размещение вычислительной техники, руб./год ($3_{\text{ар}} = 0$, т.к. помещение не арендуется);

$3_{\text{ам}}$ – амортизационные отчисления за год, руб./год;

$3_{\text{э.п}}$ – затраты на электроэнергию, руб./год;

$3_{\text{в.п}}$ – затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), руб./год;

$3_{\text{т.р}}$ – затраты на текущий и профилактический ремонт ЭВМ, руб./год;

$3_{\text{пр}}$ – прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

$F_{\text{ЭВМ}}$ – действительный фонд времени работы ЭВМ, час/год.

Сумма годовых амортизационных отчислений ($3_{\text{ам}}$) определяется по формуле (5.20):

$$3_{\text{ам}} = \frac{\sum_i 3_{\text{при}}(1 + K_{\text{доп}})m_i \cdot N_{\text{ам}i}}{Q_{\text{ЭВМ}}}, \quad (5.20)$$

где $3_{\text{при}}$ – затраты на приобретение i -го вида основных фондов, руб.;

$K_{\text{доп}}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования ($K_{\text{доп}} = 13\%$ от $3_{\text{при}}$);

$3_{\text{при}}(1 + K_{\text{доп}})$ – балансовая стоимость ЭВМ, руб.;

m_i – количество оборудования i -го вида;

$N_{\text{ам}i}$ – норма амортизации для i -го вида ЭВМ, %.

$$3_{\text{ам}} = \frac{2300,5 \cdot (1 + 0,13) \cdot 1 \cdot 0,125}{1} = 324,95 \text{ руб.}$$

Стоимость электроэнергии, потребляемой за год, определяется по формуле (5.21):

$$3_{\text{э.п}} = \frac{M_{\text{сум}} \cdot F_{\text{ЭВМ}} \cdot C_{\text{эл}} \cdot A}{Q_{\text{ЭВМ}}}, \quad (5.21)$$

где $M_{\text{сум}}$ – суммарная мощность техники (ПЭВМ и периферийной техники), кВт;

$C_{\text{эл}}$ – стоимость одного кВт · ч электроэнергии, руб.;

A – коэффициент интенсивного использования мощности, $A = 0,98 \dots 0,9$.

Действительный годовой фонд времени работы ПЭВМ ($F_{\text{ЭВМ}}$) рассчитывается по формуле (5.22):

$$F_{\text{ЭВМ}} = (D_{\text{Г}} - D_{\text{ВЫХ}} - D_{\text{ПР}}) F_{\text{СМ}} \cdot K_{\text{СМ}} (1 - K_{\text{ПОТ}}), \quad (5.22)$$

где $D_{\text{Г}}$ – общее количество дней в году ($D_{\text{Г}} = 365$ дн.);

$D_{\text{ВЫХ}}$, $D_{\text{ПР}}$ – число выходных и праздничных дней в году ($D_{\text{ВЫХ}} + D_{\text{ПР}} = 113$ дн.);

$F_{\text{СМ}}$ – продолжительность 1 смены ($F_{\text{СМ}} = 8$ ч);

$K_{\text{СМ}}$ – коэффициент сменности ($K_{\text{СМ}} = 1$);

$K_{\text{ПОТ}}$ – коэффициент, учитывающий потери рабочего времени, связанные с профилактикой и ремонтом ЭВМ ($K_{\text{ПОТ}} = 0,15$).

$$F_{\text{ЭВМ}} = (365 - 113) \cdot 8 \cdot 1 \cdot (1 - 0,15) = 1713 \text{ ч.}$$

В итоге получаем, что годовая стоимость электроэнергии ($З_{\text{Э.П}}$) равна:

$$З_{\text{Э.П}} = \frac{0,05 \cdot 1713 \cdot 0,2459 \cdot 0,96}{1} = 20,22 \text{ руб.}$$

Затраты на материалы ($З_{\text{В.М}}$), необходимые для обеспечения нормальной работы ПЭВМ составляют около 1% от балансовой стоимости ПЭВМ и определяются по формуле (5.23):

$$З_{\text{В.М}} = \frac{\sum_i З_{\text{при}} (1 + K_{\text{доп}}) m_i}{Q_{\text{ЭВМ}}} K_{\text{М.З}}, \quad (5.23)$$

где $З_{\text{при}}$ – затраты на приобретение (стоимость) ЭВМ, руб.;

$K_{\text{доп}}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования ($K_{\text{доп}} = 12\%$ от $З_{\text{при}}$);

$K_{\text{М.З}}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{\text{М.З}} = 0,01$).

$$З_{\text{В.М}} = \frac{2300,5 \cdot (1 + 0,13) \cdot 0,01}{1} = 26 \text{ руб.}$$

Затраты на текущий и профилактический ремонт ($З_{\text{Т.Р}}$) принимаются равными 5% от балансовой стоимости ЭВМ и вычисляются по формуле (5.24):

$$З_{\text{Т.Р}} = \frac{\sum_i З_{\text{при}} (1 + K_{\text{доп}}) m_i}{Q_{\text{ЭВМ}}} K_{\text{Т.Р}}, \quad (5.24)$$

где $K_{т.р}$ – коэффициент, характеризующий затраты на текущий и профилактический ремонт ($K_{т.р} = 0,05$).

$$З_{т.р} = \frac{2300,5 \cdot (1+0,13) \cdot 0,05}{1} = 129,98 \text{ руб.}$$

Прочие затраты, связанные с эксплуатацией ЭВМ ($З_{пр}$) состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5 % от балансовой стоимости, и вычисляется по формуле (5.25):

$$З_{пр} = \frac{\sum_i З_{при} (1+D_{доп}) m_i}{Q_{ЭВМ}} K_{пр}, \quad (5.25)$$

где $K_{пр}$ – коэффициент, характеризующий размер прочих затрат, связанных с эксплуатацией ЭВМ ($K_{пр} = 0,05$).

$$З_{пр} = \frac{2300,5 \cdot (1+0,13) \cdot 0,05}{1} = 129,98 \text{ руб.}$$

Для расчета машинного времени ЭВМ ($t_{ЭВМ}$, час), необходимого для разработки и отладки проекта, следует использовать формулу (5.26):

$$t_{ЭВМ} = (t_{р.п} + t_{вн}) F_{см} \cdot K_{см}, \quad (5.26)$$

где $t_{р.п}$ – срок реализации стадии «Рабочий проект» (РП), дн.;

$t_{вн}$ – срок реализации стадии «Ввод в действие» (ВП), дн.;

$F_{см}$ – продолжительность рабочей смены, ч ($F_{см} = 8$ ч.);

$K_{см}$ – количество рабочих смен ($K_{см} = 1$).

$$t_{ЭВМ} = (45 + 10) \cdot 8 \cdot 1 = 440 \text{ ч,}$$

$$C_{ч} = \frac{0+0+324,95+20,22+26+129,98+129,98}{1713} = 0,37 \text{ руб./ч,}$$

$$З_{м.в} = 0,37 \cdot 1 \cdot 440 = 162,11 \text{ руб.}$$

Затраты на материалы, необходимые для обеспечения нормальной работы ЭВМ, рассчитываются по формуле (5.26):

$$З_{мат} = \sum_{i=1}^n (Ц_i \cdot N_i (1 + K_{т.з}) - Ц_{oi} \cdot N_{oi}), \quad (5.26)$$

где C_i – цена i -го наименования материала полуфабриката, комплектующего, руб.;

N_i – потребность в i -м материале, полуфабрикате, комплектующем, натур. ед.;

$K_{т.з}$ – коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров ($K_{т.з} = 0,1$);

C_{oi} – цена возвратных отходов i -го наименования материала, руб.;

N_{oi} – количество возвратных отходов i -го наименования, натур. ед.;

n – количество наименований материалов, полуфабрикатов, комплектующих. Перечень материалов представлен в таблице Б.5.

$$З_{\text{мат}} = (0,2 \cdot 100 \cdot (1 + 0,25) - 0) + (22 \cdot 1 \cdot (1 + 0,25) - 0) = 52,5 \text{ руб.}$$

Общепроизводственные затраты ($З_{\text{общ.пр}}$) рассчитываются по формуле (5.27):

$$З_{\text{общ.пр}} = \frac{З_{\text{Посн}} \cdot Н_{\text{доп}}}{100 \%}, \quad (5.27)$$

где $Н_{\text{доп}}$ – норматив общепроизводственных затрат.

$$З_{\text{общ.пр}} = \frac{7144,704 \cdot 50}{100 \%} = 3572,35 \text{ руб.}$$

Непроизводственные затраты рассчитываются по формуле (5.28):

$$З_{\text{непр}} = \frac{З_{\text{Посн}} \cdot Н_{\text{непр}}}{100 \%}, \quad (5.28)$$

где $Н_{\text{непр}}$ – норматив непроизводственных затрат.

$$З_{\text{непр}} = \frac{7144,704 \cdot 10}{100 \%} = 714,47 \text{ руб.,}$$

$$З_p = 11059,29 + 0 + 0 + 162,11 + 52,5 + 3572,35 + 714,47 = 15560,72 \text{ руб.}$$

Результаты расчета суммарных затрат представлены в таблице Б.7.

5.5 Расчёт договорной цены разрабатываемого программного продукта

Оптовая цена ПП ($C_{\text{опт}}$) определяется по формуле (5.29):

$$C_{\text{опт}} = C(З_p) + П_p, \quad (5.29)$$

где $C(З_p)$ – себестоимость ПО, руб.;

$П_p$ – прибыль от реализации ПП, руб.;

Прибыль от реализации рассчитывается по формуле (5.30):

$$П_p = \frac{C(З_p) \cdot Y_p}{100}, \quad (5.30)$$

где Y_p – уровень рентабельности ПП, % ($Y_p = 30\%$).

$$П_p = \frac{15560,72 \cdot 30}{100} = 4668,22 \text{ руб.},$$

$$Ц_{\text{опт}} = 15560,72 + 4668,22 = 20228,93 \text{ руб.}$$

Прогнозируемая отпускная цена ПП без НДС рассчитывается по формуле (5.31):

$$Ц_{\text{отп}} = Ц_{\text{опт}} = C(З_p) + П_p. \quad (5.31)$$

Прогнозируемая отпускная цена ПП рассчитывается по формуле (5.32):

$$Ц_{\text{отп}} = C(З_p) + П_p + P_{\text{ндс}}, \quad (5.32)$$

где $P_{\text{ндс}}$ – налог на добавленную стоимость, руб.

Налог на добавленную стоимость ($P_{\text{ндс}}$) рассчитывается по формуле (5.33):

$$P_{\text{ндс}} = \frac{(C(З_p) + П_p) \cdot H_{\text{ндс}}}{100}, \quad (5.33)$$

где $H_{\text{ндс}}$ – ставка налога на добавленную стоимость, % ($H_{\text{ндс}} = 20\%$).

$$P_{\text{ндс}} = \frac{(15560,72 + 4668,22) \cdot 20}{100} = 4045,79 \text{ руб.},$$

$$Ц_{\text{отп.ндс}} = 15560,72 + 4668,22 + 4045,79 = 24274,72 \text{ руб.}$$

Розничная цена на ПП ($Ц_{\text{розн}}$) определяется по формуле (5.34):

$$Ц_{\text{розн}} = Ц_{\text{отп.ндс}} + T_n, \quad (5.34)$$

где T_n – торговая наценка при реализации программного обеспечения через специализированные магазины (торговых посредников), руб. ($T_n = 15\%$ от $Ц_{\text{отп.ндс}}$).

$$\Pi_{\text{розн}} = 24274,72 + 3641,21 = 27915,93 \text{ руб.}$$

Стоимость разработки ПП, определённая путём составления калькуляции, представлена в таблице Б.8.

5.6 Определение экономической эффективности разработки программного продукта

Эффект (прибыль) рассчитывается по формуле (5.35):

$$\mathcal{E}(\Pi) = \mathcal{Z}_{\text{баз}} - \mathcal{Z}_{\text{нов}}, \quad (5.35)$$

где $\mathcal{Z}_{\text{баз}}$ – текущие и инвестиционные затраты по базовому варианту, включающие затраты на приобретение продукта, его эксплуатацию, руб. ($\mathcal{Z}_{\text{баз}} = 43000$ руб.);

$\mathcal{Z}_{\text{нов}}$ – текущие и инвестиционные затраты по новому (разрабатываемому) проекту, руб.

$$\mathcal{E}(\Pi) = 43000 - 27915,93 = 15084,07 \text{ руб.}$$

Рентабельность затрат (\mathcal{P}) или инвестиций (\mathcal{I}) на новую информационную технологию, программный продукт рассчитываются по формуле (5.36):

$$\mathcal{P} = \frac{\mathcal{E}(\Pi)}{\mathcal{Z}(\mathcal{I})} \cdot 100\%. \quad (5.36)$$

$$\mathcal{P} = \frac{15084,07}{27915,93} \cdot 100\% = 54\%.$$

Простой срок окупаемости проекта – это период времени, по окончании которого чистый объем поступлений (доходов) покрывает объем инвестиций (расходов) в проект и соответствует периоду, при котором накопительное значение чистого потока наличности изменяется с отрицательного на положительное. Вычисляется он по формуле (5.37):

$$T_{\text{пр}} = \frac{\mathcal{Z}(\mathcal{I})}{\mathcal{E}(\Pi)}. \quad (5.37)$$

$$T_{\text{пр}} = \frac{27915,93}{15084,07} = 1,85 \text{ г.}$$

Годовой экономический эффект определяется по формуле (5.38):

$$\Gamma \mathcal{E} \mathcal{E} = \mathcal{E}(\Pi) - \mathcal{P}_{\text{баз}} \cdot \mathcal{Z}(\mathcal{I}). \quad (5.38)$$

где $P_{\text{баз}}$ – рентабельность затрат (инвестиций) базового варианта, % ($P_{\text{баз}} = 34\%$).

$$\Gamma\text{ЭЭ} = 15084,07 - 0,34 \cdot 27915,93 = 5592,66 \text{ руб.}$$

Поскольку простой срок окупаемости затрат превышает год, необходимо вычислить динамический срок окупаемости.

Оценка эффективности инвестиций базируется на сопоставлении ожидаемого чистого дохода от реализации проекта с затратами инвестиционного характера. На основании чистого потока наличности рассчитываются основные показатели оценки эффективности инвестиций: чистый дисконтированный доход (ЧДД), индекс рентабельности (ИР), динамический срок окупаемости ($T_{\text{дин}}$).

Для расчета этих показателей применяется коэффициент дисконтирования, который используется для приведения будущих потоков и оттоков денежных средств за каждый расчетный период реализации проекта к начальному периоду времени.

Коэффициент дисконтирования рассчитывается по формуле (5.39):

$$K_t = \frac{1}{(1+r)^t}, \quad (5.39)$$

где r – норма дисконта (применяется на уровне ставки рефинансирования) ($r = 10\%$, как ставка рефинансирования на 03.05.2023);

t – период реализации проекта ($t = 4$ года).

Годовой доход, получаемый как результат экономии пользовательского времени на некоторых операциях при внедрении программного обеспечения, составляет 5630 руб.

Если инвестиционные затраты, связанные с разработкой программного продукта и приобретением компьютерной техники, периферийных устройств, кабелей и т.д. производится только в год разработки, а первые доходы ожидаются в следующем году, то ЧДД можно вычислить по формуле (5.40):

$$\text{ЧДД} = -I_0 + \frac{D_1}{(1+r)^1} + \frac{D_2}{(1+r)^2} + \dots + \frac{D_n}{(1+r)^n}, \quad (5.40)$$

где D_n – доходы (эффекты) от внедрения информационных технологий, руб.;

I_0 – затраты инвестиционного характера (единовременные, капитальные) на разработку и внедрение информационных технологий, руб.

Расчёт чистого дисконтированного дохода приведён в таблице Б.9.

Как видно из вышеуказанной таблицы, чистый дисконтированный доход к концу 4-го года реализации составит 1342,7 руб.

Индекс рентабельности (доходности) (ИР) – это отношение суммарного дисконтированного дохода к суммарным дисконтированным затратам. Проект эффективен, если норма дисконта оказывается больше или равной ставки

рефинансирования, требуемой инвестором, кредитором. ИР рассчитывается по формуле (5.41):

$$\text{ИР} = \frac{\sum_{t=0}^n \frac{D_t}{(1+r)^t}}{\sum_{t=0}^n \frac{I_t}{(1+r)^t}}. \quad (5.41)$$

Инвестиционные проекты эффективны при $\text{ИР} > 1$.

$$\text{ИР} = \frac{(18830,97+4652,89+4229,9+3845,37)}{30216,43} = 1,044.$$

Динамический срок окупаемости ($T_{\text{дин}}$). Расчёт динамического срока окупаемости проекта осуществляется по накопительному дисконтированному чистому потоку наличности. Динамический срок окупаемости учитывает стоимость капитала и рассчитывается по формуле (5.42):

$$T_{\text{дин}} = \frac{\sum_{t=0}^n \frac{I_t}{(1+r)^t}}{\sum_{t=0}^n \frac{D_t}{(1+r)^t}}, \quad (5.42)$$

$$T_{\text{дин}} = 2 + \frac{4229,9}{6732,56} = 3.59 \text{ г.}$$

Внутренняя норма доходности (рентабельность) представляет собой ставку дисконта (ВНД). Её вычисление является итеративным процессом, который начинается с барьерной ставки (r), если при этом ЧДД положительный, то в следующей итерации используют более высокую ставку, если отрицательная – то более низкую.

Точное значение ВНД вычисляется по формуле (5.43):

$$\text{ВНД} = r_{\text{чдд}(+)} + \frac{\text{ЧДД}(+)}{\text{ЧДД}(+) - \text{ЧДД}(-)}, \quad (5.43)$$

где $r_{\text{чдд}(+)}$ – значение ставки дисконта;

В таблице Б.10 приведён расчёт внутренней нормы дисконта.

$$\text{ВНД} = 12,7 + \frac{113,39}{113,39+121,42} = 13,18 \text{ \%}.$$

Таким образом, по результатам проведенных вычислений величина ЧДД > 0 , значение ИД > 1 , а рассчитанная внутренняя норма дисконта превышает фактическое значение ($13,18 \% > 10 \%$). Это позволяет сделать вывод о том, что вложение инвестиций в разработку данного проекта является экономически целесообразным [10].

Срок окупаемости будет составлять 3.59 года. Техничко-экономические показатели проекта представлены в таблице Б.11.

6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ. ПСИХОФИЗИОЛОГИЧЕСКИЕ И ЭРГОНОМИЧЕСКИЕ ОСНОВЫ ОХРАНЫ ТРУДА

6.1 Охрана труда. Цели и принципы

Охрана труда – это комплекс мер и действий, направленных на предотвращение травматизма, сохранение здоровья и обеспечение безопасности работников в процессе трудовой деятельности. Охрана труда включает в себя анализ и оценку рисков, связанных с рабочей средой и условиями труда, разработку и внедрение соответствующих мероприятий, а также контроль и управление над их соблюдением. Целью охраны труда является создание таких условий труда, при которых работники могут свободно и эффективно выполнять свои профессиональные обязанности, минимизируя риск возникновения профессиональных заболеваний, травм и других негативных последствий для их здоровья и благополучия. Она включает в себя множество мероприятий и действий, направленных на обеспечение безопасности и здоровья работников.

Цели охраны труда:

- предотвращение производственных травм и несчастных случаев на рабочем месте;
- защита здоровья и благополучия работников;
- снижение рисков и опасностей, связанных с трудовой деятельностью сотрудников;
- создание безопасных и здоровых условий труда;
- повышение эффективности, производительности, полезности и оптимальности труда;
- соответствие нормативным требованиям и законодательству в области охраны труда.

Принципы охраны труда:

- предварительное планирование и оценка рисков включает в себя анализ рабочих процессов, определение потенциальных опасностей и оценку возможных рисков для здоровья и безопасности работников;
- профилактика и предупреждение направлена на предотвращение возникновения опасных ситуаций и исключение возможности травм и несчастных случаев на рабочем месте;
- обучение и информирование – работники должны быть ознакомлены с правилами и инструкциями по охране труда, а также получать необходимое обучение и информацию о мерах безопасности и здоровья на рабочем месте;
- использование коллективного опыта – опытные работники и специалисты должны делиться своими знаниями и опытом в области охраны труда с остальными членами коллектива;
- системный подход, в котором охрана труда должна рассматриваться как интегральная часть управления предприятием или организацией, включая планирование, организацию, контроль и непрерывное улучшение условий и процессов труда.

6.2 Психологические аспекты охраны труда

Психологические аспекты охраны труда включают изучение взаимосвязи между психическим состоянием работника и его трудовой деятельностью, а также разработку мероприятий, направленных на поддержание психического здоровья и благополучия работников. Ключевыми аспектами являются стресс и работа с тяжелыми эмоциональными нагрузками.

Стресс является одним из важных психологических аспектов, оказывающих влияние на охрану труда. Он может возникать в результате несоответствия требованиям работы, конфликтных ситуаций, перегрузки информацией, неопределенности, недостатка контроля над рабочим процессом и других факторов. Воздействие стресса на работника может быть значительным и иметь негативные последствия для его здоровья и безопасности. Эмоциональное и физическое напряжение, вызванное стрессом, может снижать концентрацию, приводить к ошибкам и повышать риск возникновения несчастных случаев на рабочем месте. Кроме того, длительный стресс может оказывать воздействие на иммунную систему, повышать утомляемость, вызывать психосоматические расстройства и способствовать развитию психических заболеваний. Поэтому предотвращение и управление стрессом на рабочем месте являются важной задачей в области охраны труда.

Оценка рисков и предупреждение стресса являются важными аспектами в управлении и поддержании здоровья и благополучия как в личной жизни, так и в профессиональной сфере. Она включает в себя идентификацию потенциальных опасностей, анализ вероятности и возможных последствий неблагоприятных событий или ситуаций. Это позволяет разработать стратегии и меры для предотвращения, снижения или управления рисками. Оценка рисков может проводиться как на индивидуальном уровне, так и в организационном контексте, чтобы защитить интересы и безопасность людей.

Предупреждение стресса направлено на предотвращение или снижение воздействия факторов, которые могут вызвать негативные эмоциональные или физические реакции у человека. Это может включать планирование и организацию работы, управление временем, развитие навыков управления стрессом, поддержку и обеспечение здоровой рабочей среды. Предупреждение стресса помогает поддерживать психологическое и физическое благополучие, повышает эффективность и продуктивность работы.

Ряд профессий связан с высокой эмоциональной нагрузкой, которая может оказывать значительное влияние на работников. Работники в сферах здравоохранения и услуг, социального обслуживания, экстренных служб, психологии, юстиции и других отраслях сталкиваются с тяжелыми эмоциональными ситуациями, которые могут вызывать эмоциональное и физическое истощение и повышенный риск профессионального выгорания.

Организация времени и отдыха играют важную роль при работе с тяжелыми эмоциональными нагрузками. Эмоционально напряженная работа может быть физически и психологически истощающей, поэтому необходимо уделить особое внимание поддержанию баланса и восстановлению.

Один из ключевых аспектов организации времени в таких условиях – установление четких границ между работой и личной жизнью. Важно создать структуру и расписание, чтобы иметь возможность отдыхать и регулярно восстанавливаться. Это может включать установку четких временных рамок для работы и планирования перерывов и отдыха.

Регулярные перерывы и отдых имеют большое значение при работе с эмоциональными нагрузками. Они позволяют отключиться от работы, расслабиться и восстановить энергию. Важно использовать этот временной отрезок для занятий, которые помогут снять стресс и поддержать психологическое благополучие, такие как физические упражнения, медитация, чтение, общение с друзьями и семьей и занятия хобби.

Физические упражнения являются отличным способом расслабления и снятия напряжения после продолжительного периода работы. Короткая физическая активность, такая как прогулка или набор простых упражнений, может помочь расслабить мышцы, улучшить кровообращение и увеличить уровень эндорфинов – гормонов счастья.

Медитация и дыхательные практики также могут быть полезны для снятия стресса и улучшения психологического состояния. Практики глубокого дыхания и осознанности помогают сосредоточиться, успокоить ум и улучшить общее самочувствие.

Чтение является отличным способом уйти от рабочих мыслей и погрузиться в другой мир. Увлекательная книга или интересная статья могут помочь расслабиться и отвлечься от повседневных забот.

Эти меры помогут уменьшить негативное влияние на работников и способствовать созданию более безопасной и здоровой рабочей среды. Охрана труда должна включать не только физическую, но и психологическую составляющую для обеспечения полноценной безопасности и благополучия работников [11].

6.3 Физиологические аспекты охраны труда

Физиологические аспекты охраны труда относятся к изучению влияния физических факторов рабочей среды на здоровье и благополучие работников. Они включают в себя анализ физического напряжения, воздействия вредных веществ и других факторов, которые могут оказывать негативное воздействие на организм человека.

Физическое напряжение является одним из важных аспектов охраны труда, поскольку неправильные рабочие позы, тяжелые физические нагрузки и длительные периоды статического напряжения могут оказывать негативное влияние на здоровье и благополучие работников. При недостаточном внимании к физическому напряжению на рабочем месте могут возникать травмы опорно-двигательной системы, мышечные заболевания, а также повышенный риск профессиональных заболеваний.

Разнообразие задач и периодические перерывы играют важную роль в обеспечении охраны труда и снижении физического напряжения. В рабочей среде, где выполняются повторяющиеся и монотонные задачи, часто возникают

риски для здоровья и безопасности работников. Одним из способов снижения этих рисков является введение разнообразия задач и регулярных перерывов.

Разнообразие задач включает в себя расширение спектра деятельности работника, включение различных видов задач и процессов. Это позволяет разнообразить двигательные активности, нагрузку на различные группы мышц и двигаться в разных позициях. Периодическое изменение задач помогает предотвратить перенапряжение конкретных мышц и суставов, а также уменьшить риск развития монотонных движений и повреждений, связанных с повторяемыми задачами.

Перерывы также имеют важное значение для охраны труда и снижения физического напряжения. Регулярные перерывы позволяют работнику отдохнуть, восстановиться и снять физическую нагрузку. Они также предоставляют возможность выполнить растяжку или физические упражнения для расслабления и укрепления мышц.

Рекомендуется проводить перерывы через определенные интервалы времени в течение рабочего дня. Это может быть краткосрочные перерывы каждые 1-2 часа и более продолжительные перерывы для отдыха и приема пищи. Важно использовать эти перерывы для смены активности, перемещения, растяжки и расслабления, чтобы уменьшить физическую нагрузку и снизить риск развития мышечных заболеваний и травм.

Физическое напряжение является серьезным фактором риска на рабочем месте, но с правильными мерами по охране труда его воздействие может быть существенно снижено. Предотвращение травм и повышение комфорта работников способствуют созданию безопасной и продуктивной рабочей среды.

Шум и вибрация также носят негативный физиологический характер, поскольку их воздействие на работников может приводить к различным проблемам со здоровьем. Длительное и интенсивное воздействие шума и вибрации может вызывать повреждение слуха, нарушения нервной системы, а также оказывать отрицательное влияние на психологическое и физиологическое состояние человека.

Инженерные и организационные меры играют важную роль в снижении негативного воздействия шума и вибрации. Вот некоторые из них:

Инженерные меры:

- использование звукоизоляционных материалов – установка звукоизолирующих панелей, покрытий и материалов для снижения шума в рабочей среде;
- использование акустических экранов – размещение специальных экранов или перегородок для блокировки и поглощения звуковых волн;
- звукоизоляция оборудования – установка акустических оболочек или изоляции на источниках шума, чтобы снизить уровень шума, излучаемого оборудованием;
- проведение регулярного технического обслуживания – поддержание оборудования в исправном состоянии, чтобы избежать ненужных шумовых вибраций и звуковых отклонений.

Организационные меры:

- правильное планирование распределения рабочих мест – размещение

рабочих мест и оборудования таким образом, чтобы минимизировать негативное воздействие шума и вибрации на сотрудников;

- расписание работы – определение периодов покоя и отдыха, чтобы дать сотрудникам возможность отдохнуть от негативного воздействия шума и вибрации;

- предоставление индивидуальных средств защиты – выдача специальных наушников, наушников с активным шумоподавлением и других средств защиты от шума для сотрудников, работающих в шумной среде;

- обучение и информирование – проведение обучающих программ и предоставление информации о правильных методах работы и защиты от негативного воздействия шума и вибрации.

Шум и вибрация являются серьезными факторами риска на рабочем месте, однако с применением соответствующих мер по охране труда можно существенно снизить их воздействие на работников. Защита от шума и вибрации не только способствует сохранению здоровья работников, но и повышает их производительность и комфортность работы [12].

6.4 Эргономические аспекты охраны труда

Эргономические аспекты охраны труда являются важным элементом создания безопасной и комфортной рабочей среды. Эргономика изучает взаимодействие человека и его окружающей среды с целью оптимизации условий труда и предотвращения негативных последствий для здоровья и производительности работников. Из наиболее значимых аспектов можно выделить рациональное проектирование рабочих мест и использование подходящих инструментов и оборудования.

Рациональное проектирование рабочих мест направлено на создание комфортной и безопасной рабочей среды для работников. Эффективное проектирование рабочих мест помогает предотвратить травмы, усталость и другие негативные последствия, связанные с неправильным расположением оборудования, инструментов и мебели.

Эргономические принципы относятся к проектированию рабочих мест, оборудования и рабочих процессов с учетом потребностей и возможностей работников, с целью снижения риска возникновения травм и заболеваний, а также повышения комфорта и производительности.

Вот некоторые основные эргономические принципы охраны труда:

- рациональное планирование рабочего места – учет эргономических аспектов при размещении оборудования, инструментов и материалов, чтобы сократить ненужные перемещения и усилия работников;

- подходящая мебель и оборудование – предоставление рабочих столов, стульев, клавиатур, мониторов и другого оборудования, соответствующего анатомическим и физиологическим потребностям работников;

- соблюдение правильной позы и движений – обучение работников правильным рабочим позам и методам выполнения задач для снижения напряжения и риска возникновения мышечных и скелетных заболеваний;

– поддержка психологического комфорта – создание условий, которые способствуют психологическому комфорту работников, такие как поддержка социальной интеграции, устранение излишнего стресса и создание приятной рабочей атмосферы.

Рациональное проектирование рабочих мест имеет значительное значение для охраны труда. Оно способствует созданию условий, при которых работники могут выполнять свои задачи без излишнего напряжения, усталости и риска возникновения травм. Правильное размещение оборудования, оптимальная высота рабочих поверхностей и адаптивность к индивидуальным потребностям работников обеспечивают комфортную и эффективную рабочую среду.

Использование подходящих инструментов и оборудования является существенным элементом охраны труда. Корректный выбор и правильное использование инструментов и оборудования на рабочем месте помогают предотвратить травмы, усталость и повышают эффективность работы.

Вот несколько ключевых принципов, связанных с использованием подходящих инструментов и оборудования:

– оценка рабочих задач – перед выбором инструментов и оборудования необходимо проанализировать требования рабочих задач, что включает понимание типа работы, специфических требований к инструментам и оборудованию, а также учет физических особенностей работников;

– эргономика и безопасность – инструменты должны быть удобными для использования, легкими и иметь эргономичные ручки, а оборудование должно быть стабильным, надежным и обеспечивать безопасные условия работы;

– обучение и инструктаж – работники должны быть обучены правильному использованию инструментов и оборудования.

Применение эргономических принципов в охране труда способствует снижению рисков травм и заболеваний, повышает комфорт и производительность работников, а также улучшает общую рабочую среду.

7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ ПРИ ИСПОЛЬЗОВАНИИ ПРОГРАММНОГО ПРОДУКТА

В Республике Беларусь в области ресурсо- и энергосбережения активно применяются меры и стандарты, направленные на оптимизацию использования ресурсов и снижение энергопотребления. В этой сфере действует межгосударственный стандарт, разработанный Межгосударственным комитетом по стандартизации, метрологии и сертификации (МТК).

Межгосударственный стандарт, разработанный МТК, устанавливает требования и рекомендации по энергосбережению и энергоэффективности в различных секторах экономики. Он предоставляет руководство и регламентирует использование передовых технологий, методов и подходов, способствующих сокращению потребления энергии и оптимизации ресурсов.

МТК сотрудничает с органами государственного управления, предприятиями, общественными организациями и другими заинтересованными сторонами для проведения обучения, консультаций и информационной поддержки по внедрению стандарта и повышению энергоэффективности в стране. Разработанный стандарт служит основой для разработки и внедрения мер по сокращению потребления энергии, повышению эффективности использования ресурсов и снижению негативного влияния на окружающую среду.

Внедрение межгосударственного стандарта по ресурсо- и энергосбережению в Республике Беларусь имеет целью улучшение энергетической эффективности, сокращение затрат на энергию, снижение выбросов вредных веществ и обеспечение устойчивого развития экономики страны. Этот стандарт способствует повышению конкурентоспособности предприятий, снижению зависимости от импорта энергоносителей и общему сокращению негативного влияния на окружающую среду.

Стандарт «Энергетическая эффективность» в Республике Беларусь является важным нормативным документом, разработанным с целью регулирования и стимулирования повышения энергоэффективности в различных секторах экономики Беларуси. Он определяет требования, методы и процедуры, направленные на сокращение энергопотребления и оптимизацию энергетических процессов, а также устанавливает правила и нормы, которые должны соблюдаться организациями и предприятиями в целях повышения эффективности использования энергетических ресурсов. Он предусматривает различные меры, такие как проведение энергетического аудита, внедрение современных технологий энергосбережения, оптимизацию систем энергоснабжения и другие мероприятия, направленные на сокращение потребления энергии.

Применение ресурсо- и энергосберегающих практик при использовании программного обеспечения (ПО) имеет важное значение для экономии ресурсов, снижения негативного влияния на окружающую среду и повышения эффективности работы систем. Осознанное использование ресурсов и энергии является важным аспектом экологической ответственности. Время, энергия и материалы, затрачиваемые на разработку, производство, эксплуатацию и утилизацию компьютерных систем и инфраструктуры, могут оказывать негативное воздействие

на окружающую среду. Применение ресурсо- и энергосберегающих методов в ПО помогает уменьшить потребление энергии, сократить использование материалов и снизить выбросы вредных веществ.

Энергосбережение позволяет использовать ресурсы более эффективно и экономно. Оптимизированное использование вычислительной мощности, памяти, сетевых ресурсов и других компонентов системы позволяет сократить затраты на оборудование, улучшить производительность и уменьшить нагрузку на сетевую инфраструктуру. Это также может привести к снижению расходов на энергию, охлаждение и обслуживание. Также это может повысить эффективность работы систем и улучшить пользовательский опыт. Оптимизация алгоритмов, управление энергопотреблением, виртуализация и другие техники могут сократить время обработки задач, ускорить процессы и повысить отзывчивость системы. Это в свою очередь способствует повышению производительности бизнеса и улучшению пользовательского удовлетворения [13].

Применение ресурсо- и энергосберегающих стандартов в ПО способствует созданию более устойчивых систем и инфраструктуры. Это особенно важно в условиях растущей нагрузки на вычислительные ресурсы и ограниченности природных ресурсов. Сокращение потребления энергии, оптимизация ресурсов и эффективное использование инфраструктуры позволяют улучшить устойчивость системы и снизить зависимость от энергетических и ресурсных рисков.

В итоге, межгосударственный стандарт, разработанный МТК в Республике Беларусь, является важным инструментом для содействия ресурсо- и энергосбережению. Он способствует повышению энергоэффективности в различных отраслях экономики и приводит к более устойчивому и экологически ответственному развитию страны. Применение ресурсо- и энергосбережения в ПО является важным аспектом современной разработки и использования технологий. Оно способствует экологической ответственности, экономии ресурсов, повышению эффективности и устойчивости систем, а также улучшению восприятия бренда и репутации компании.

ЗАКЛЮЧЕНИЕ

В ходе написания дипломной работы был разработан программный комплекс автоматизации фонда студенческого общежития. Была подобрана и проанализирована литература о разработке приложений, используя стек MERN. В ходе анализа задачи было выявлено, что создание хорошего приложения требует сочетания продуманного интерфейса и корректной обработки различных ошибок на стороне сервера и клиента. Для достижения этой цели были выбраны наиболее подходящие технологии, которые обладают богатым набором инструментов и компонентов.

Результатом дипломной работы является *web*-приложение для управления студенческим общежитием. В ходе разработки программного продукта были решены следующие задачи:

- проведен анализ требований пользователей и определены основные функциональности приложения. На основе этой информации был разработан дизайн интерфейса, включающий в себя компоненты, макеты и взаимодействие элементов;
- выполнен процесс верстки, включающий создание *HTML*-структуры и стилей *CSS*;
- проведено корректное размещение элементов интерфейса, которые были реализованы с учетом современных стандартов и рекомендаций;
- разработана эффективная навигация и логика взаимодействия между различными компонентами интерфейса, что включает в себя реализацию кнопок, форм, меню, всплывающих окон и других элементов, необходимых для удобной работы с приложением;
- применены оптимизационные методы для улучшения производительности интерфейса: минимизация запросов к серверу, оптимизация загрузки изображений и ресурсов, а также кэширование данных для более быстрого доступа;
- проводились тесты интерфейса и бизнес-логики, чтобы убедиться в его корректной работе и соответствии требованиям.

Разработанный программный продукт предназначен для использования студентами, воспитателями и комендантом, которые хотят быстро взаимодействовать друг с другом. Так как приложений такого плана в открытом доступе практически нет, разработанное приложение может стать очень ценным для студенческих общежитий.

Список использованных источников

1. Проектирование и разработка *web*-приложений. – Электрон. данные. – Режим доступа: <https://urait.ru/book/proektirovanie-i-razrabotka-web-prilozheniy-512113>. – Дата доступа: 05.03.2023.
2. Бобров, К.А. Изучаем программирование на *JavaScript* / К.А.Бобров, А.П. Кантор. – 1-е изд., стер. – Литературный дом, 2018. – 294 с.
3. Кузнецов А.А. *MongoDb* и *JavaScript*. Разработка *WEB*-приложений / А.А. Кузнецов ; под общ. ред. А.О. Чекмарев. – М. : Геркулес, 2008. – 340 с.
4. Разработка веб-приложений с использованием *Node.js* и *Express*. – Электрон. данные. – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs. – Дата доступа: 08.03.2023.
5. Разработка интерактивных пользовательских интерфейсов с использованием библиотеки *React*. – Электрон. данные. – Режим доступа: react.dev/ru/learn/describing-the-ui. – Дата доступа: 08.03.2023.
6. Использование *IDE WebStorm* для разработки веб-приложений. – Электрон. данные. – Режим доступа: <https://prettier.io/docs/en/webstorm.html>. – Дата доступа: 08.03.2023.
7. Управление базой данных *MongoDB* с помощью *MongoDB Compass*. – Электрон. данные. – Режим доступа: <https://www.hostland.ru/articles/mongodb-compass#:~:text=GUI%20MongoDB%20Compass&text=Это%20удобный%20Клиент%2C%20разработанный%20MongoDB,OC%20Linux%2C%20Mac%20и%20Windows>. – Дата доступа: 16.03.2023.
8. Проектирование и моделирование систем с использованием *StarUML*. – Электрон. данные. – Режим доступа: <https://soware.ru/products/staruml>. – Дата доступа: 17.03.2023.
9. Тестирование и отладка *API* с помощью *Postman*. – Электрон. данные. – Режим доступа: <https://blog.skillfactory.ru/glossary/postman/>. – Дата доступа: 02.04.2023.
10. Кожевников, Е. А. Расчёт экономической эффективности разработки программных продуктов: метод. указания по подготовке организационно-экономического раздела дипломных работ для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения / Е. А. Кожевников, Н. В. Ермалинская. – Гомель : ГГТУ им. П. О. Сухого, 2012. – 68 с.
11. Панфилов, Г. И. Психофизиология труда / Г.И. Панфилов, Ю.Г. Баранов. – 2-е изд., стер. – Академия, 2011. – 384 с.
12. Баев, А. С. Основы эргономики и охраны труда / А. С. Баев; под общ. ред. В. Г. Климова. – М. : Волга, 2014. – 174 с.
13. Щебов, М.А. Энергосбережение при разработке программного обеспечения / М.А. Щебов ; под общ. ред. Степанеев Д.Д. – М. : Линейка, 2017. – 148 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

App.js

```
import React, { useState } from 'react'
import './App.css'
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom'
import { Login } from './pages/Auth/Login'
import { Registration } from './pages/Auth/Registration'
import { Home } from './pages/Home'
import { StudentRoom } from './pages/StudentRoom'
import { NewsPanel } from './componets/NewsAction/NewsPanel'
import { EventPanel } from './componets/ViewEvent/EventPanel';
import { MentorRoom } from './pages/MentorRoom'
import { MyContext } from './context'
import { NewNews } from './componets/NewsAction/NewNews'
import { MainStudent, Claim } from './componets/ForStudent'
import { ContainerTech } from './componets/ForStudent/Tech/ContainerTech'
import { FullNews } from './componets/NewsAction/FullNews'
import { MainMentor } from './componets/ForMentor/MainMentor';
import { CreateClaim } from './componets/ForMentor/CreateClaim';
import { CreateEvent } from './componets/ForMentor/CreateEvent';
import { Chat } from './componets/Chat';
import { CreateChat } from './componets/Chat/CreateChat'
import { AdminRoom } from './pages/AdminRoom'
import { MainAdmin } from './componets/ForAdmin/MainAdmin'
import { ImportStudents } from './componets/ForAdmin/ImportStudents'
import { EmployeeList } from './componets/ForAdmin/EmployeeList'
import { FullMentor } from './componets/FullMentor';
import { AccPlaces } from './componets/ForAdmin/AccPlaces';
import { CreateMentor } from './componets/ForAdmin/CreateMentor';
import { CreateTech } from './componets/ForAdmin/CreateTech'
import { CreateRepair } from './componets/ForStudent/CreateRepair'
import { AddStudent } from './componets/ForAdmin/AddStudent'
import { Reports } from './componets/ForAdmin/Reports'
import { PayHostel } from './componets/ForStudent/PayHostel'
import { ViewPayHostel } from './componets/ForAdmin/ViewPayHostel';
import { RepairViewAdmin } from './componets/ForAdmin/RepairAction/RepairViewAdmin';
import { FullStudentMentor } from './componets/ForMentor/FullStudentMentor';
import { FullStudentAdmin } from './componets/ForAdmin/FullStudentAdmin';
import { StudentListAdmin } from './componets/ForAdmin/StudentListAdmin';
import { StudentsListMentor } from './componets/ForMentor/StudentsListMentor';

import { ToastContainer, toast } from 'react-toastify'
import 'react-toastify/dist/ReactToastify.css'
import { ChangeBalls } from './componets/ForMentor/ChangeBalls';
import { BallsInfo } from './componets/ForStudent/BallsInfo';
import { FreePlaces } from './componets/ForAdmin/FreePlaces';

function App() {
  const [contextState] = useState({
    toast
  })

  return (
    <MyContext.Provider value={contextState}>
      <ToastContainer position="top-center" autoClose={5000} hideProgressBar={true}>
```

```

newestOnTop={false} closeOnClick rtl={false} pauseOnFocusLoss draggable
pauseOnHover theme="light"
/>
<Router>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/login" element={<Login />} />
    <Route path="/reg" element={<Registration />} />
    <Route path="/student" element={<StudentRoom />} />
    <Route index element={<MainStudent />} />
    <Route path="tech" element={<ContainerTech />} />
    <Route path="pay-hostel" element={<PayHostel />} />
    <Route path="claim" element={<Claim />} />
    <Route path="news" element={<NewsPanel />} />
    <Route path="balls-info" element={<BallsInfo />} />
    <Route path="news/:id" element={<FullNews />} />
    <Route path="events" element={<EventPanel />} />
    <Route path="chat" element={<Chat />} />
    <Route path="employee-info" element={<EmployeeList />} />
    <Route path="employee-info/:id" element={<FullMentor />} />
    <Route path="create-repair" element={<CreateRepair />} />
  </Route>
  <Route path="/mentor" element={<MentorRoom />} />
  <Route index element={<MainMentor />} />
  <Route path="students" element={<StudentsListMentor />} />
  <Route path="students/:id" element={<FullStudentMentor />} />
  <Route path="students/:id/create" element={<CreateClaim />} />
  <Route path="students/:id/add-tech" element={<CreateTech />} />
  <Route path="students/:id/change-balls" element={<ChangeBalls />} />
  <Route path="news" element={<NewsPanel />} />
  <Route path="news/:id" element={<FullNews />} />
  <Route path="news/create" element={<NewNews />} />
  <Route path="events" element={<EventPanel />} />
  <Route path="events/create" element={<CreateEvent />} />
  <Route path="chat" element={<Chat />} />
  <Route path="chat/create" element={<CreateChat />} />
</Route>
  <Route path="/admin" element={<AdminRoom />} />
  <Route index element={<MainAdmin />} />
  <Route path="students" element={<StudentListAdmin />} />
  <Route path="students/add" element={<AddStudent />} />
  <Route path="students/repairs" element={<RepairViewAdmin />} />
  <Route path="students/:id" element={<FullStudentAdmin />} />
  <Route path="students/:id/add-tech" element={<CreateTech />} />
  <Route path="students/import" element={<ImportStudents />} />
  <Route path="students/view-pay-hostel" element={<ViewPayHostel />} />
  <Route path="employee/:id" element={<FullMentor />} />
  <Route path="employee/create" element={<CreateMentor />} />
  <Route path="places" element={<AccPlaces />} />
  <Route path="places/free-places" element={<FreePlaces />} />
</Route>
</Routes>
</Router>
</MyContext.Provider>
)
}
export default App

```

AdminRoom.jsx

```

import React from 'react'
import { Header } from "../../components/Layout/Header"

```

```

import { Menu } from "../../componets/Menu"
import { Outlet } from "react-router-dom"
import { Footer } from "../../componets/Layout/Footer"
import { BreadCrumbs } from "../../componets/BreadCrumbs"
import { MENU_ADMIN, ROUTES_ADMIN } from '../../mocks'

export function AdminRoom() {
  return (
    <div>
      <div className="container pt-3 st-room__container">
        <div className="w-100">
          <Header>
            <Menu menu={MENU_ADMIN} role='Комендант' />
          </Header>
          <BreadCrumbs routes={ROUTES_ADMIN} />
          <div className="rounded w-100">
            <Outlet />
          </div>
        </div>
      </div>
      <Footer />
    </div>
  )
}

```

MentorRoom.jsx

```

import React from 'react'
import { Header } from "../../componets/Layout/Header"
import { Menu } from "../../componets/Menu"
import { Outlet } from "react-router-dom"
import { Footer } from "../../componets/Layout/Footer"
import { BreadCrumbs } from "../../componets/BreadCrumbs"
import { MENU_MENTOR, ROUTES_MENTOR } from '../../mocks'

export function MentorRoom() {
  return (
    <div>
      <div className="container pt-3 st-room__container">
        <div className="w-100">
          <Header>
            <Menu menu={MENU_MENTOR} role="Воспитатель"/>
          </Header>
          <BreadCrumbs routes={ROUTES_MENTOR} />
          <div className="rounded w-100">
            <Outlet />
          </div>
        </div>
      </div>
      <Footer />
    </div>
  )
}

```

StudentRoom.jsx

```

import React, { useEffect, useState } from 'react'
import './styles.css'
import { Avatar } from "../../componets/Layout/Avatar";
import { Outlet } from 'react-router-dom'
import { Header } from '../../componets/Layout/Header'
import { Menu } from '../../componets/Menu'

```

```

import { Footer } from '../componets/Layout/Footer'
import { MENU_STUDENT, ROUTES_STUDENT } from '../mocks'
import { BreadCrumbs } from '../componets/BreadCrumbs'
export function StudentRoom() {
  const [student, setStudent] = useState({})

  useEffect(() => {
    setStudent(JSON.parse(localStorage.getItem("user")))
  }, [])

  return (
    <div>
      <div className="container pt-3 st-room__container">
        <Avatar data={student} />
        <div className="w-100">
          <Header>
            <Menu menu={MENU_STUDENT} role="Студент"/>
          </Header>
          <BreadCrumbs routes={ROUTES_STUDENT} />
          <div className="rounded w-100">
            <Outlet />
          </div>
        </div>
      </div>
      <Footer />
    </div>
  )
}

```

Login.jsx

```

import React, { useState } from 'react'
import { useHttp } from '../hooks'
import { fieldsLogin } from '../mocks'
import { Form } from './Form'
import './styles.css'
import { Link, useNavigate } from "react-router-dom";
import { SERVER } from '../constants'
import { toastMess } from '../helpers'

export function Login() {
  const [form, setForm] = useState({
    login: "",
    password: "",
  })

  const [errors, setErrors] = useState(new Map())
  const navigate = useNavigate();

  const { request } = useHttp();

  function handleInput(event) {
    setForm({...form, [event.target.name]: event.target.value})
  }

  async function loginHandle() {
    try {
      const { data, message, errors, status } = await request(SERVER + '/auth/login', 'POST', {...form})
      if(!errors && data.role === 'mentor'){
        localStorage.setItem('user', JSON.stringify(data._doc));
        navigate("/mentor")
      }
    }
  }
}

```

[illegible]

```
import React, { useState } from 'react'
import { Link, useNavigate } from "react-router-dom"
import { useHttp } from '../hooks'
import { fieldsRegistration } from '../mocks'
import { Form } from './Form'
import { SERVER } from '../constants'
import './styles.css'
import { toastMess } from "../helpers";

export function Registration() {
  const [form, setForm] = useState({
    numberTest: "",
    email: ""
  })
  const [errors, setErrors] = useState(new Map())
  const navigate = useNavigate();

  const { request } = useHttp();

  function handleInput(event) {
    setForm({ ...form, [event.target.name]: event.target.value })
  }
}
```



```

    getIdFetch()
  }, [])

function handleSaveSection(event) {
  event.preventDefault()
  console.log(file)
  if(description.length < 5 || description.length > 1000) {
    toastMess(false, "Неверная длина описания")
    return
  }
  toastMess(true, "Секция создана")
  setSections([...sections, {header, description, file, id: idNews}])
  setDescription("")
  setHeader("")
}

const handleHeaderChange = (event) => {
  setHeader(event.target.value);
};

const handleDescriptionChange = (event) => {
  setDescription(event.target.value);
};

const handleFileChange = (event) => {
  setFile(event.target.files[0]);
};

async function handleSaveNews() {
  if(!sections.length){
    toastMess(false, "Сначала нужно создать секцию")
  }
  const formDataArray = sections.map((item, index) => {
    const formData = new FormData();
    Object.keys(item).forEach(key => {
      formData.append(`${key}`, item[key]);
    });
    return formData;
  });

  let requests = formDataArray.map(item => fetch(SERVER + '/news/create-section', {
    method: 'POST',
    body: item,
  }));

  Promise.all(requests)
    .then(responses => responses.forEach(
      response => toastMess(response.ok, "Обработка запроса...")
    ));
}

return (
  <
    <form onSubmit={handleSaveSection}>
      <div className="bg-light rounded p-3">
        <p className="text-muted">
          Здесь вы можете создать одну секцию новости, к которой вы прикрепляете фото
        </p>
        <div>
          <div className="input-group input-group-sm">
            <span className="input-group-text" id="input-group-sm-example">
              Заголовок

```

```

    </span>
    <input
      type="text"
      className="form-control"
      aria-label="Small input group"
      aria-describedby="input-group-sm"
      name="header"
      value={header}
      onChange={handleHeaderChange}
      required
    />
  </div>
</div>
<div>
  <input
    className="form-control mt-3"
    type="file"
    id="file"
    name="file"
    onChange={handleFileChange}
    required
  />
</div>
<div className="form-floating mt-3">
  <textarea className="form-control"
    placeholder="Напишите здесь текст"
    id="floatingTextarea"
    style={{ minHeight: "300px" }}
    name="description"
    value={description}
    onChange={handleDescriptionChange}
    required
  />
</div>
<div className="d-flex justify-content-between bd-highlight mt-3">
  <button
    type="submit"
    className="btn bg-primary border-0 text-light rounded"
  >
    Создать секцию
  </button>
  <button type="button"
    className="btn bg-primary border-0 text-light rounded"
    onClick={handleSaveNews}
  >
    Сохранить новость
  </button>
</div>
</div>
</form>
<div className="bg-light rounded p-3 mt-3">
  {sections.map((item, ind) => (
    <p className="text-muted" key={ind}>Секция {ind + 1}: {item.header}</p>
  ))}
</div>
</>
)
}

```

AccPlaces.jsx

```
import React, { useEffect, useState } from 'react'
```

```

import { useHttp } from "../../hooks";
import { SERVER } from "../../constants";
import { Loading } from "../../Loading";
import { Link } from "react-router-dom";

export function AccPlaces() {
  const { loading, request } = useHttp()
  const [res, setRes] = useState([])

  useEffect(() => {
    const fetchData = async () => {
      const { data, message } = await request(SERVER + '/student/get-places', "GET")
      setRes([...data])
    }
    fetchData()
  }, [])

  if(loading) return <Loading />

  return(
    <div className="bg-light rounded p-3">
      <button className="btn btn-light mb-3">
        <Link className="text-primary text-decoration-none" to={ 'free-places' }>Посмотреть свободные места</Link>
      </button>
      <div className="d-flex">
        <div className="bg-primary rounded ms-2 px-3 text-light"
          style={{ height: '25px' }}
        >
          <span>0</span>
          <span>0</span>
          <span>0</span>
        </div>
        <p> - отображены занятые комнаты</p>
      </div>
      <hr className="hr"/>
      <div className="d-flex flex-column">
        {res.map((item, ind) => {
          if(ind === 0) return
          if(!item) {
            return (
              <div key={ind} className="d-flex border-bottom mb-2">
                <p className="me-5" style={{ width: '40px' }}>{ind}</p>
                <p>Этаж пустой</p>
              </div>
            )
          } else {
            return (
              <div className="d-flex border-bottom mb-2 flex-wrap" key={ind}>
                <p className="me-5" style={{ width: '40px' }}>{ind}</p>
                {item.map((place, ind) => (
                  <div key={ind} className="bg-primary rounded ms-2 px-3 h-50 text-light">
                    <span>{place.floor}</span>
                    <span>{place.block}</span>
                    <span>{place.apartment}</span>
                  </div>
                )))
                <p className="text-muted ms-5">Занято: {item.length}</p>
              </div>
            )
          }
        })}
      </div>
    </div>
  )
}

```

```
)
}
```

AddStudent.jsx

```
import React, { useState } from 'react'
import { SimpleForm } from "../../SimpleForm";
import { ADD_STUDENT_FORM } from "../../mocks";
import { useHttp } from "../../hooks";
import { SERVER } from "../../constants";
import { toastMess } from "../../helpers";

export function AddStudent() {

  const [form, setForm] = useState({
    firstName: "",
    secondName: "",
    middleName: "",
    formEducation: 0,
    floor: 0,
    block: 0,
    apartament: 0,
    faculty: "",
    group: ""
  })

  const { request } = useHttp()

  function handleInput(event) {
    setForm({...form, [event.target.name]: event.target.value})
  }

  async function handleClick(event) {
    event.preventDefault()
    const { message, status } = await request(SERVER + '/student/add-student', 'POST', {
      ...form
    })
    toastMess(status, message)
  }

  return (
    <div className="w-100 p-3 bg-white rounded">
      <SimpleForm
        fields={ADD_STUDENT_FORM}
        onChange={handleInput}
        onClick={ (event) => handleClick(event) }
        buttonName="Сохранить"
        errors={null}
        messFromServer=""
      />
    </div>
  )
}
```

ChangeBalls.jsx

```
import React, { useState } from 'react'
import { useHttp } from "../../hooks";
import { SERVER } from "../../constants";
import { useParams } from 'react-router-dom'
import { dateFormat, toastMess } from "../../helpers"
import { BallsView } from "../../BallsView";
```

```

import { BALLS } from '../../mocks'

export function ChangeBalls() {
  const { request } = useHttp();
  const { id } = useParams()
  const [category, setCategory] = useState(null)
  const [balls, setBalls] = useState("")

  function handleTable(event) {
    setCategory({ ...BALLS[event.target.value] })
  }

  async function handleChangeBalls() {
    const { message, status } = await request(SERVER + '/mentor//update-balls-v2', "POST", {
      numberTest: id,
      num: balls,
      summary: category.summary
    })
    toastMess(status, message)
  }

  return(
    <div>
      <div className="bg-light rounded mb-3 p-3">
        {
          category
          ?
          <div>
            <p>
              <span>{ category.summary }</span><br/>
              <span>{ category.num } баллов</span>
            </p>
            <input
              type="string"
              className="form-control w-25"
              id="formControlInputBalls"
              placeholder={ category.num }
              value={ balls }
              onChange={ (event) => setBalls(event.target.value) }
            />
            <button type="button" className="btn btn-primary mt-3" onClick={ handleChangeBalls }>
              Сохранить
            </button>
          </div>
          :
          <p>Выберите категорию</p>
        }
      </div>
      <BallsView onClick={ handleTable }/>
    </div>
  )
}

```

PayHostel.jsx

```

import React, { useMemo, useState, useEffect } from 'react'
import formEducationImg from '../../assets/formEducation.png'
import { PAY_HOSTEL_FORM } from "../../mocks"
import { SimpleForm } from "../../SimpleForm"
import { useHttp } from '../../hooks'
import { SERVER } from '../../constants'
import { toastMess, dateFormat, getDaysDifference, paymentHelp, sumHostelStudent } from "../../helpers"

```

```

export function PayHostel() {
  const user = useMemo(() => JSON.parse(localStorage.getItem('user')), [])
  const [payments, setPayments] = useState(user.pay || [])
  const [paysState, setPaysState] = useState({})

  const [form, setForm] = useState({
    receipt: "",
    payment: ""
  })

  const { request } = useHttp()

  useEffect(() => {
    const fetchData = async () => {
      console.log('fetchData')
      const { data } = await request(SERVER + '/student/get-pay', 'POST', {
        numberTest: user.numberTest
      })
      setPaysState(data //pays/ sumHostel/ curPay
      setPayments(data.pays.reverse())
    }
    fetchData()
  }, [])

  function handleInput(event) {
    setForm({...form, [event.target.name]: event.target.value})
  }

  async function handleClick(event) {
    event.preventDefault()
    const { message, status } = await request(SERVER + '/student/pay-hostel', 'POST', {
      numberTest: user.numberTest,
      receipt: form.receipt,
      payment: form.payment
    })
    toastMess(status, message)
    if(status) {
      setPayments([
        {
          date: String(new Date()),
          receipt: form.receipt,
          payment: Number(form.payment)
        }, ...payments])
    }
  }

  if(user.formEducation === 'бесплатное') {
    return (
      <div className="bg-light p-3 rounded min-vh-100">
        <p className="text-muted">Отчет о оплате за проживание в общежитии доступен только для студентов,
          которые обучаются на платной основе</p>
        <img alt="edu-img" src={formEducationImg} />
      </div>
    )
  }

  return(
    <div className="bg-light p-3 rounded min-vh-100">
      <p>
        <span className="text-muted">Дата и время</span> {dateFormat()}
      </p>
      <SimpleForm
        fields={PAY_HOSTEL_FORM}

```

```

    onChange={handleInput}
    onClick={ (event) => handleClick(event) }
    buttonName="Сохранить"
    errors={ null }
    messFromServer={ "" }
  />
  <div className="mt-3">
    { Object.keys(paysState).length !== 0
    ?
      <div>
        <p>{ paymentHelp(user.dateInHostel, paysState.sumHostel, sumHostelStudent(payments)) }</p>
        <p>Оплачено всего: {sumHostelStudent(payments)} (BYN)</p>
      </div>
      :
      <p>Загрузка...</p>
    }
  </div>
  <div className="mt-3">
    <p>Текущая стоимость проживания: <span>{paysState?.curPay} (BYN)</span></p>
    <p>История оплаты</p>
    {payments.map((item, ind) => (
      <div key={ind}>
        <span>{ind + 1}</span>
        <span className="mx-3">{item.payment} (BYN)</span>
        <span className="text-primary">{dateFormat(new Date(item.date))}</span>
        { ind === 0 ? <span className="mx-2 text-warning">Последняя оплата</span> : null }
      </div>
    ))}
  </div>
</div>
)
}

```

CreateRepair.jsx

```

import React, { useState, useEffect, useContext } from 'react'
import { SimpleForm } from "../SimpleForm";
import { CREATE_REPAIR_FORM } from '../../mocks'
import { SERVER } from "../../constants";
import { useHttp } from "../../hooks";
import { ViewRepairs } from "../../ViewRepairs";
import { MyContext } from '../../context'

export function CreateRepair() {
  const [form, setForm] = useState({
    header: "",
    description: "",
  })
  const [res, setRes] = useState([])

  useEffect(() => {
    const fetchData = async () => {
      const numberTest = JSON.parse(localStorage.getItem('user')).numberTest
      console.log(numberTest)
      const { data } = await request(SERVER + `~/repair/get-repairs-id`, "POST", {
        numberTest: numberTest
      })
      setRes([...data])
    }
    fetchData()
  }, [])
}

```

```

const { request } = useHttp()
const { toast } = useContext(MyContext)

function handleInput(event) {
  setForm({...form, [event.target.name]: event.target.value})
}

async function handleClick(event) {
  event.preventDefault()
  const user = JSON.parse(localStorage.getItem('user'))
  const repair = {
    ...form,
    room: user.room,
    user: {
      firstName: user.firstName,
      secondName: user.secondName,
      middleName: user.middleName,
      numberTest: user.numberTest
    }
  }
  const { data, message } = await request(SERVER + '/repair/create', 'POST', {...repair})
  setRes([...res, data])
  toast.success(message)
}

return (
  <>
    <div className="w-100 p-3 bg-white rounded">
      <SimpleForm
        fields={CREATE_REPAIR_FORM}
        onChange={handleInput}
        onClick={(event) => handleClick(event)}
        buttonName="Сохранить"
        errors={null}
        messFromServer=""
      />
    </div>
    <div>
      <ViewRepairs repairs={res.reverse()} />
    </div>
  </>
)
}

import path from "path"
import { Router } from 'express'
import { News } from '../models/news'
import { getDateAndTime } from '../utils'

const router = Router()

const multer = require("multer");

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads/news')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now() + '.' + file.originalname.split('.').pop())
  }
})

interface MulterRequest extends Request {

```



```

    file: any;
  }

const upload = multer({ storage: storage })
router.post('/create-section', upload.single("file"), async (req, res) => {
  try {
    const { header, description, id } = req.body
    const img = (req as unknown as MulterRequest).file.filename
    const candidateNews = await News.findById(id)
    console.log('-----')
    console.log(candidateNews.body)

    await News.updateOne({ _id: id }, {
      body: [...candidateNews.body, { header, description, img: img}],
    })
    return res.status(200).json({ message: 'Новость создана' })
  } catch(err) {
    console.log(err.message)
    return res.status(500).json({ message: 'Что-то пошло не так - сервер' })
  }
})

```

news.route.ts

```

router.post('/create-news', async (req, res) => {
  try {
    const { mentor } = req.body
    const news = new News({
      mentor,
      dateCreate: getDateAndTime(),
    })
    const newNews = await news.save()
    return res.status(200).json({ data: { id: newNews._id }, message: 'Новость создана' })
  } catch(err) {
    return res.status(500).json({ message: 'Что-то пошло не так - сервер' })
  }
})

```

```

router.get('/get-news', async (req, res) => {
  try {
    const data = await News.find()
    const news = []
    for(let i = 0; i < data.length; i++) {
      if(!data[i].body.length) {
        await News.deleteOne({ _id: data[i]._id });
      } else {
        news.push(data[i])
      }
    }

    return res.status(200).json({ data: news, message: 'Новости получены' })
  } catch(err) {
    return res.status(500).json({ message: 'Что-то пошло не так - сервер' })
  }
})

```

```

router.get('/get-news-id', async (req, res) => {
  try {
    const id = req.query.id
    const data = await News.findOne({ _id: id })
    return res.status(200).json({ data: data, message: 'Новость получена' })
  } catch(err) {
  }
}

```

```

    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

router.get('/load', function(req, res) {
  const img: string = String(req.query.img)
  const imagePath = path.join("D:", "diplom-app", "server", 'uploads', 'news', img);
  res.sendFile(imagePath);
});

module.exports = router

```

server.ts

```

const express = require('express');
const mongoose = require('mongoose');
const bp = require('body-parser')
const cors = require('cors')

export const app = express();

app.use(cors())
app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*')
  next()
})

app.use(bp.json())
app.use(bp.urlencoded({ extended: true }))
app.use(express.static(__dirname));

app.use('/auth', require('./routers/auth.route'))
app.use('/student', require('./routers/student.route'))
app.use('/mentor', require('./routers/mentor.route'))
app.use('/news', require('./routers/news.route'))
app.use('/event', require('./routers/event.route'))
app.use('/chat', require('./routers/chat.route'))
app.use('/common', require('./routers/common.route'))
app.use('/admin', require('./routers/admin.route'))
app.use('/repair', require('./routers/repair.route'))

const PORT = 5000

async function start() {
  try {
    await mongoose.connect(`mongodb://127.0.0.1:27017/diplom_db`);
    app.listen(PORT); //, () => console.log(`Server has been started on port ${PORT}!`)
  }
  catch (e) {
    console.log(`Server Error: ${e.message}`);
    process.exit(1);
  }
}

start();

```

student.route.ts

```

import path from "path";
import { Router } from 'express'
import { Student } from '../models/student'

```

```

import { Account } from '../models/account'
import { Hostel } from '../models/hostel';
import { getDefaultPlaces } from '../utils'

const router = Router();
router.post('/import-students',
  async (req, res) => {
    try {
      const { students, type } = req.body
      const hostel = (await Hostel.find())[0]
      const freePlaces = hostel.places
      let busyPlaces = []
      let stObjs = []
      for (let i = 1; i < students.length; i++) {
        busyPlaces.push(`${students[i][5]}-${students[i][6]}-${students[i][7]}`)
        stObjs.push({
          firstName: students[i][0],
          secondName: students[i][1],
          middleName: students[i][2],
          formEducation: students[i][3],
          balls: 0,
          dateInHostel: String(new Date()),
          numberTest: students[i][4] as number,
          room: {
            floor: students[i][5] as number,
            block: students[i][6] as number,
            apartment: students[i][7] as number,
          },
          faculty: students[i][8],
          group: students[i][9],
        })
      }
      if (type === 'add') {
        await Student.insertMany(stObjs)
        for(let i = 0; i < busyPlaces.length; i++) {
          const index = freePlaces.findIndex(element => element === busyPlaces[i]);
          if (index !== -1) {
            freePlaces.splice(index, 1, 'Занято');
          }
        }
        hostel.places = freePlaces
        await hostel.save()
        return res.status(200).json({message: 'Данные успешны выгружены'})
      }
      else {
        const places = getDefaultPlaces()
        for(let i = 0; i < busyPlaces.length; i++) {
          const index = places.findIndex(element => element === busyPlaces[i]);
          if (index !== -1) {
            places.splice(index, 1, 'Занято');
          }
        }
        hostel.places = places
        await hostel.save()
        await Account.deleteMany({role: 'student'})
        await Student.deleteMany()
        await Student.insertMany(stObjs)
        return res.status(200).json({message: 'Данные успешны выгружены'})
      }
    } catch (err) {
      return res.status(500).json({message: 'Что-то пошло не так'})
    }
  }

```

```

});

router.get('/get-info', async (req, res) => {
  try {
    const data = await Student.find()
    return res.status(200).json({data: data, message: 'Данные загружены'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
})

router.get('/get-student-id', async (req, res) => {
  try {
    const nt = req.query.id
    const data = await Student.findOne({numberTest: nt})
    return res.status(200).json({data: data, message: 'Данные загружены'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
})

router.post('/change-status-claim', async (req, res) => {
  try {
    const {numberTest, ind} = req.body
    const data = await Student.findOne({numberTest: numberTest})
    data.remarks[ind].status = 1
    await data.save()
    return res.status(200).json({message: 'Статус изменен'})
  } catch (err) {
    console.log(err)
    return res.status(500).json({message: 'Что-то пошло не так - сервер'})
  }
})

router.post('/create-claim', async (req, res) => {
  try {
    const {numberTest, header, text, mentor, dateAndTime} = req.body
    const student = await Student.findOne({numberTest: numberTest})
    const update = {
      remarks: [
        ...student.remarks, {
          dateAndTime,
          header,
          text,
          mentor,
          status: 0
        }
      ]
    }
    await Student.findOneAndUpdate({numberTest: numberTest}, update)
    return res.status(200).json({message: 'Данные обновлены'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
})

router.get('/get-places', async (req, res) => {
  try {
    const data = await Student.find()
    const resStudents = []

    for (let i = 0; i < data.length; i++) {
      if (!resStudents[data[i].room?.floor]) {

```

```

        resStudents[data[i].room?.floor] = [data[i].room]
      } else {
        resStudents[data[i].room?.floor] = [...resStudents[data[i].room?.floor], data[i].room]
      }
    }
    return res.status(200).json({data: resStudents, message: 'Данные загружены'})
  } catch (err) {
    console.log(err.message)
    return res.status(500).json({message: 'Что-то пошло не так - server'})
  }
})

router.post('/add-tech', async (req, res) => {
  try {
    const {numberTest, model, number, type} = req.body
    const student = await Student.findOne({numberTest: numberTest})
    const update = {
      privateTechs: [
        ...student.privateTechs, {
          model,
          number,
          type,
        }
      ]
    }
    await Student.findOneAndUpdate({numberTest: numberTest}, update)
    return res.status(200).json({message: 'Новая техника добавлена'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
})

router.post('/update-balls', async (req, res) => {
  try {
    const {numberTest, balls} = req.body
    const student = await Student.findOne({numberTest: numberTest})
    let update = {}
    if (!student.balls) {
      update = {
        balls: Number(balls)
      }
    } else {
      update = {
        balls: Number(balls) + Number(student.balls)
      }
    }
    const newStudent = await Student.findOneAndUpdate({numberTest: numberTest}, update, {
      new: true
    })
    return res.status(200).json({data: newStudent.balls, message: 'Данные обновлены'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
})

const multer = require("multer");

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads/students')
  },
  filename: function (req, file, cb) {

```

```

    cb(null, file.fieldname + '-' + Date.now() + '.' + file.originalname.split('.').pop())
  }
})

const upload = multer({ storage: storage })

interface MulterRequest extends Request {
  file: any;
}

router.post('/update-info', upload.single("file"), async (req, res) => {
  try {
    const { numberTest, email, login, newPassword, oldPassword } = req.body
    const img = (req as unknown as MulterRequest).file?.filename
    const student = await Student.findOne({ numberTest: numberTest })
    const acc = await Account.findOne({ login: student.account?.login })

    if (acc.password !== oldPassword && oldPassword !== "") {
      return res.status(400).json({ message: 'Пароль должен совпадать со старым паролем' })
    }

    const previewAcc = await Account.findOne({ login: login })
    if (previewAcc) {
      return res.status(400).json({ message: 'Такой логин уже существует' })
    }

    let update = {
      email: student.email,
      'account.login': student.account?.login,
    }
    let accUpdate = {
      login: acc.login,
      password: acc.password
    }

    if (email !== "") update.email = email
    if (login !== "") {
      update['account.login'] = login
      accUpdate.login = login
    }
    if (newPassword !== "") accUpdate.password = newPassword

    await Account.findOneAndUpdate({ login: student.account?.login }, accUpdate)
    await Student.findOneAndUpdate({ numberTest: numberTest }, update)
    if (img) {
      await Student.findOneAndUpdate({ numberTest: numberTest }, {
        img: img
      })
    }
    const studentNew = await Student.findOne({ numberTest: numberTest })
    return res.status(200).json({ data: studentNew, message: 'Данные обновлены' })
  } catch (err) {
    console.log(err.message)
    return res.status(500).json({ message: 'Что-то пошло не так' })
  }
})

router.get('/load', function (req, res) {
  try {
    const img = req.query.img
    if (img === 'undefined') {
      const imageDefaultPath = path.join("D:", "diplom-app", "server", 'uploads', 'student.png');
      res.sendFile(imageDefaultPath);
    }
  }
})

```

```

    return
  }
  const imagePath = path.join("D:", "diplom-app", "server", "uploads", "students", `${img}`);
  if (!imagePath) return res.status(400)
  res.sendFile(imagePath);
} catch (err) {
  return res.status(400)
}
});

router.post('/get-remarks', async (req, res) => {
  try {
    const { numberTest } = req.body
    const data = await Student.findOne({ numberTest: numberTest })
    return res.status(200).json( {data: data.remarks || [], message: 'Данные загружены'})
  } catch (err) {
    console.log(err)
    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

router.post('/add-student', async (req, res) => {
  try {
    const {
      firstName, secondName, middleName,
      formEducation, floor, block, apartment, faculty, group
    } = req.body

    const student = new Student({
      firstName, secondName, middleName, formEducation,
      dateInHostel: String(new Date()),
      room: {
        floor, block, apartment
      },
      faculty, group
    })
    await student.save()
    return res.status(200).json( {message: 'Новый студент добавлен'})
  } catch (err) {
    console.log(err)
    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

router.post('/pay-hostel', async (req, res) => {
  try {
    const { receipt, numberTest, payment } = req.body
    const student = await Student.findOne({ numberTest })
    let update = []
    if(student.pay) {
      update = [
        ...student.pay, {
          date: String(new Date()),
          receipt, payment
        }
      ]
    } else {
      update = [{
        date: String(new Date()),
        receipt, payment
      }]
    }
    student.pay = update
  }
})

```

```

    await student.save()
    return res.status(200).json( {message: 'Квитанция добавлена'})
  } catch (err) {
    console.log(err)
    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

```

```

router.post('/get-pay', async (req, res) => {
  try {
    const { numberTest } = req.body
    const student = await Student.findOne({ numberTest })
    const hostel = (await Hostel.find())[0]
    return res.status(200).json({
      data: {
        pays: student.pay || [],
        sumHostel: hostel.costsHostel,
        curPay: hostel.costHostel,
      },
      message: 'Квитанция добавлена'
    })
  } catch (err) {
    console.log(err)
    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

```

```

router.post('/get-balls', async (req, res) => {
  try {
    const { numberTest } = req.body
    const student = await Student.findOne({ numberTest })
    return res.status(200).json(
      {
        data: {
          balls: student.balls || 0,
          ballsInfo: student.ballsInfo || []
        }, message: 'Данные получены'})
  } catch (err) {
    return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
  }
})

```

module.exports = router

mentor.route.ts

```

import path from "path"
import { Router } from 'express'
import { Mentor } from '../models/mentor'
import { IMentor } from '../interfaces'
import { Account } from "../models/account"
import { Student } from "../models/student";
const multer = require("multer")

```

const router = Router()

```

router.post('/info-mentor', async (req, res) => {
  try {
    const { login } = req.body
    const model: IMentor | null = await Mentor.findOne({ "account.login": login });
    if(!model) return res.status(400).json( {message: 'Данных нет'})
  }
}

```



```

    return res.status(200).json({data: model, message: 'Данные найдены'})
  } catch (err) {
    return res.status(500).json({message: 'Что-то пошло не так'})
  }
});

router.get('/load', function(req, res) {
  const img: string = String(req.query.img)
  if(img === 'undefined') return res.status(400)
  const imagePath = path.join("D:", "diplom-app", "server", 'uploads', img);
  res.sendFile(imagePath);
})

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now() + '.' + file.originalname.split('.').pop())
  }
})

const upload = multer({ storage: storage })

interface MulterRequest extends Request {
  file: any;
}

router.post('/update-info', upload.single("file"), async (req, res) => {
  try {
    const { id, login, newPassword, oldPassword, phone, email } = req.body
    const img = (req as unknown as MulterRequest).file?.filename
    const mentor = await Mentor.findOne({_id: id})
    const acc = await Account.findOne({login: mentor.account?.login})

    if(acc.password !== oldPassword && oldPassword !== "") {
      return res.status(400).json({message: 'Пароль должен совпадать со старым паролем'})
    }

    const previewAcc = await Account.findOne({login: login})
    if(previewAcc) {
      return res.status(400).json({message: 'Такой логин уже существует'})
    }

    let update = {
      'account.login': login || mentor.account?.login,
      img: img || mentor.img,
      phone: phone || mentor.phone || "",
      email: email || mentor.email || "",
    }

    let accUpdate = {
      login: login || acc.login,
      password: newPassword || acc.password
    }

    await Account.findOneAndUpdate({login: mentor.account?.login}, accUpdate)
    const newMentor = await Mentor.findOneAndUpdate({_id: id}, update, {
      new: true
    })

    return res.status(200).json({data: newMentor, message: 'Данные обновлены'})
  } catch (err) {

```

```

    console.log(err.message)
    return res.status(500).json({ message: 'Что-то пошло не так'})
  }
})

router.post('/get-students-by-impact', async (req, res) => {
  try {
    const { impact } = req.body
    const data = await Student.find()
    const students = []
    for(let i = 0; i < data.length; i++) {
      if(data[i].room.floor >= impact.from && data[i].room.floor <= impact.to) {
        students.push(data[i])
      }
    }
    return res.status(200).json({ data: students, message: 'Данные загружены'})
  } catch (err) {
    return res.status(500).json({ message: 'Что-то пошло не так'})
  }
})

router.post('/update-balls-v2', async (req, res) => {
  try {
    const { numberTest, num, summary } = req.body
    const student = await Student.findOne({ numberTest: numberTest})

    let update = {}
    if (!student.ballsInfo || student.ballsInfo?.length === 0) {
      update = {
        ballsInfo: [{
          num: Number(num), summary
        }],
        balls: Number(num),
      }
    } else {
      update = {
        ballsInfo: [...student.ballsInfo, { num, summary }],
        balls: Number(num) + Number(student.balls)
      }
    }
    const newStudent = await Student.findOneAndUpdate({ numberTest: numberTest}, update, {
      new: true
    })
    return res.status(200).json({ data: newStudent.balls, message: 'Данные обновлены'})
  } catch (err) {
    return res.status(500).json({ message: 'Что-то пошло не так'})
  }
})

module.exports = router

```

chat.route.ts

```

import { Student } from '../models/student'
import { Mentor } from '../models/mentor'
const { Router } = require('express')
import { Chat } from '../models/chat'

const router = Router()

router.get('/create-chat', async (req, res) => {
  try {

```

```

const chat = new Chat({ })
const newChat = await chat.save()
return res.status(200).json( {data: { id: newChat._id }, message: 'Чат создан'})
} catch(err) {
return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
}
})

```

```

router.post('/create-chat',async (req, res) => {
try {
const { arrayId, name, idMentor } = req.body
console.log(arrayId, name, idMentor)
const chat = await Chat.create({
name,
messages: [],
})
for (let i = 0; i < arrayId.length; i++) {
const doc = await Student.findById(arrayId[i])
console.log(doc.chats)
if(!doc.chats) {
doc.chats = []
}
doc.chats = [...doc.chats, chat._id]
await doc.save();
}
const mentor = await Mentor.findById(idMentor)
if(!mentor.chats) {
mentor.chats = []
}
mentor.chats = [...mentor.chats, chat._id]
await mentor.save()
return res.status(200).json( {data: "", message: 'Чат создан'})
} catch(err) {
console.log(err.message)
return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
}
})

```

```

router.post('/get-chats',async (req, res) => {
try {
const { idChats } = req.body
const chats = []
for (let i = 0; i < idChats.length; i++) {
const doc = await Chat.findById(idChats[i])
chats.push(doc)
}
return res.status(200).json( {data: chats, message: 'Чат создан'})
} catch(err) {
return res.status(500).json( {message: 'Что-то пошло не так - сервер'})
}
})

```

ПРИЛОЖЕНИЕ Б

(Обязательное)

Результаты расчёта экономического обоснования

Таблица Б.1 – Перечень и объем функции программного обеспечения

Код функций	Наименование (содержание) функций	Объем функции строк исходного кода (<i>LOC</i>)	
		по каталогу (V_o)	уточненный (V_y)
102	Контроль, предварительная обработка и ввод информации	490	180
109	Управление вводом-выводом	1970	920
202	Формирование базы данных	1980	1150
206	Манипулирование данными	7860	2840
305	Формирование файла	2130	680
601	Проведение тестовых испытаний прикладных программ в интерактивном режиме	3780	1130
Итого		18210	6900

Таблица Б.2 – Значения коэффициентов удельных весов трудоёмкости стадий разработки ПО в общей трудоёмкости ПО

Категория новизны ПО	Без применения CASE-технологии				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{ТЗ}$	$K_{ЭП}$	$K_{ТП}$	$K_{РП}$	$K_{ВН}$
В	0,08	0,19	0,28	0,34	0,11

Таблица Б.3 – Расчет общей трудоемкости разработки ПО

№ п/п	Показатели	Стадии разработки					Итого
		ТЗ	ЭП	ТП	РП	ВН	
1	2	3	4	5	6	7	8
1	Общий объем ПО (V_o), количество строк LOC	—	—	—	—	—	18210
2	Общий уточненный объем ПО (V_y), количество строк LOC	—	—	—	—	—	6900
3	Категория сложности разрабатываемого ПО	—	—	—	—	—	3
4	Нормативная трудоемкость разработки ПО (T_n), чел.-дн.	—	—	—	—	—	290
5	Коэффициент повышения сложности ПО (K_c)	1,07	1,07	1,07	1,07	1,07	—
6	Коэффициент, учитывающий новизну ПО (K_n)	0,63	0,63	0,63	0,63	0,63	—
7	Коэффициент, учитывающий степень использования стандартных модулей (K_t)	—	—	—	0,65	—	—
8	Коэффициент, учитывающий средства разработки ПО ($K_{y.p}$)	0,65	0,65	0,65	0,65	0,65	—
9	Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{т.з}$, $K_{э.п}$, $K_{т.п}$, $K_{р.п}$, $K_{в.н}$)	0,08	0,19	0,28	0,34	0,11	1,0
10	Распределение нормативной трудоемкости ПО по стадиям, чел.-дн.	23	55	81	99	32	290
11	Распределение скорректированной (с учетом K_c , K_n , K_t , $K_{y.p}$) трудоемкости ПО по стадиям, чел.-дн.	10	24	36	28	14	112
12	Общая трудоемкость разработки ПО (T_o), чел.-дн.	—	—	—	—	—	112

Таблица Б.4 – Прейскурант на приобретение оборудования

№ п/п	Наименование	Кол-во (m_i)	Стоимость приобре- тения, руб.	Балансовая стоимость единицы ЭВМ, руб.	Суммарная стоимость ЭВМ, руб.
1	Ноутбук <i>Acer Aspire A315-25-15</i>	1	1800	1926	1926
2	Мышь компью- терная <i>Sven RX-G880</i>	1	230	246,1	246,1
3	Коврик для мыши <i>Defender Black Ultra XXL</i>	1	120	128,4	128,4
Итого		–	–	–	2300,5

Таблица Б.5 – Расходы на материалы

№ п/п	Наименование	Количество, (шт)	Цена с НДС за ед. изд., руб.	Сумма с НДС, руб.
1	Бумага для прин- тера	100	0,2	20
2	<i>Flesh</i> -накопитель	1	22	22
Итого		–	–	44

Таблица Б.6 – Параметры расчета затрат на разработку ПО

Параметр	Значение
1	2
Базовая ставка, руб.	228
Разряд разработчика	8
Тарифный коэффициент	1,57
Корректирующий коэффициент базовой тарифной ставки 1-го разряда для ИТ компаний	2,08
Норматив отчислений на доп. зарплату разработчиков ($H_{\text{доп}}$), %	15
Количество ЭВМ ($Q_{\text{ЭВМ}}$), шт.	1
Затраты на приобретение единицы ЭВМ, руб.	2300,5

Продолжение таблицы Б.6

1	2
Стоимость одного кВт · ч электроэнергии ($C_{эл}$), руб.	0,2459
Затраты на изготовление эталонного экземпляра ($Z_{эт}$), руб.	0
Затраты на технологию ($Z_{тех}$), руб.	0
Норматив общепроизводственных затрат ($H_{доп}$), %	50
Норматив непроизводственных затрат ($H_{непр}$), %	10

Таблица Б.7 – Результаты расчёта суммарных затрат на разработку ПО

№ п/п	Статья затрат	Итого
1	Затраты на оплату труда разработчиков ($Z_{тр}$)	11059,29
1.1	Основная заработная плата разработчиков	7144,704
1.2	Дополнительная заработная плата разработчиков	1071,71
1.3	Отчисления от основной и дополнительной заработной платы	2842,88
2	Затраты машинного времени ($Z_{м.в}$)	162,11
2.1	Стоимость машино-часа	0,37
	Затраты на заработную плату обслуживающего персонала	0
	Годовые затраты на аренду помещения	0
	Сумма годовых амортизационных отчислений	324,95
	Стоимость электроэнергии, потребляемой за год	20,22
	Действительный годовой фонд времени работы ПЭВМ	1713
	Затраты на материалы	26
	Затраты на текущий и профилактический ремонт	129,98
	Прочие затраты, связанные с эксплуатацией ЭВМ	129,98
2.2	Машинное время ЭВМ	440
3	Затраты на изготовление эталонного экземпляра ($Z_{эт}$)	0
4	Затраты на технологию ($Z_{тех}$)	0
5	Затраты на материалы ($Z_{мат}$)	52,5
6	Общепроизводственные затраты ($Z_{общ.пр}$)	3572,35
7	Непроизводственные (коммерческие) затраты ($Z_{непр}$)	714,47
8	Суммарные затраты на разработку ПО (Z_p)	15560,72

Таблица Б.8 – Плановая калькуляция разработки программного продукта

№ п/п	Наименование статьи расходов	Условные обозначения	Значение
1	Затраты на оплату труда разработчиков	$З_{тр}$	11059,28732
1.1	Основная заработная плата разработчиков		7144,704
1.2	Дополнительная заработная плата разработчиков		1071,7056
1.3	Отчисления от основной и дополнительной заработной платы		2842,88
2	Затраты машинного времени	$З_{м.в}$	162,11
3	Затраты на изготовление эталонного экземпляра	$З_{эт}$	0
4	Затраты на технологию	$З_{тех}$	0
5	Затраты на материалы	$З_{мат}$	52,5
6	Общепроизводственные затраты	$З_{общ.пр}$	3572,35
7	Производственная себестоимость		14846,25
8	Непроизводственные (коммерческие) затраты	$З_{непр}$	714,47
9	Полная себестоимость (суммарные затраты на разработку ПО)	$З_p$	15560,72
10	Прибыль от реализации ПО	$П_p$	4668,22
11	Отпускная цена ПО без НДС	$Ц_{отп}$	20228,93
12	Налог на добавленную стоимость	$P_{ндс}$	4045,79
13	Отпускная цена ПО с НДС	$Ц_{отп.ндс}$	24274,72
14	Торговая наценка	T_n	3641,21
15	Розничная цена ПО	$Ц_{розн}$	27915,93

Таблица Б.9 – Расчёт чистого дисконтированного дохода

Показатель	Годы реализации проекта				
	0-й	1-й	2-й	3-й	4-й
1	2	3	4	5	6
Отток денежных средств	30216,43	—	—	—	—
Капитальные вложения	2300,5	—	—	—	—
Затраты на разработку ПО	27915,93	—	—	—	—

Продолжение таблицы Б.9

1	2	3	4	5	6
Приток денежных средств	—	20714,07	5630	5630	5630
Экономический эффект от производства нового ПО	—	15094,07	—	—	—
Чистый экономический эффект	—	5630	5630	5630	5630
Чистый поток денежных средств	-30216,43	20714,07	5630	5630	5630
Коэффициент дисконтирования (при $r = 10\%$)	1	0,91	0,83	0,75	0,68
Текущая стоимость потока	-30216,43	18830,97	4652,89	4229,9	3845,37
Накопленная стоимость потока	-30216,43	-11385,45	-6732,56	-2502,66	1342,71

Таблица Б.10 – Расчёт внутренней нормы дисконта

Год	Денежные потоки	при $r = 12,5\%$		при $r = 13\%$	
		K_t	текущий поток	K_t	текущий поток
0	-30216,43	1	-30216,43	1	-30216,43
1	20714,07	0,89	18412,51	0,88	18331,04
2	5630	0,79	4448,39	0,78	4409,12
3	5630	0,7	3954,13	0,69	3901,87
4	5630	0,62	3514,78	0,61	3452,98
		ЧДД = 113,39		ЧДД = -121,42	

Таблица Б.11 – Техничко-экономические показатели проекта

№ п/п	Наименование показателя	Единица измерения	Базовый вариант	Проектный вариант
1	2	3	4	5
Показатели затрат на разработку				
1	Общая трудоёмкость разработки ПО	чел.-дн.	—	112

Продолжение таблицы Б.11

2	Капитальные вложения в проект	руб.	—	2300,5
3	Затраты на разработку программы	руб.	—	15560,72
3.1	Затраты на оплату труда разработчиков	руб.	—	11059,29
3.2	Затраты машинного времени	руб.	—	162,11
3.3	Затраты на изготовление эталонного экземпляра	руб.	—	0
3.4	Затраты на технологию	руб.	—	0
3.5	Затраты на материалы	руб.	—	52,5
3.6	Общепроизводственные затраты	руб.	—	3572,35
3.7	Непроизводственные (коммерческие) затраты	руб.	—	714,47
Показатели стоимости				
4	Отпускная цена ПП с НДС	руб.	—	24274,72
5	Розничная цена ПП	руб.	43000	27915,93
Показатели экономической эффективности				
6	Рентабельность затрат	%	—	54
7	Простой срок окупаемости проекта	лет	—	1,85
8	Годовой экономический эффект	руб.	—	5592,66
9	Чистый дисконтированный доход	руб.	—	1342,7
10	Внутренняя норма доходности	%	—	13,18
11	Индекс рентабельности (доходности)	%	—	1,044
12	Динамический срок окупаемости	лет	—	3,59