

Федеральное государственное образовательное бюджетное  
учреждение высшего образования  
**«Финансовый университет  
при Правительстве Российской Федерации»  
(Финансовый университет)**

Департамент анализа данных и машинного обучения

Пояснительная записка к курсовой работе по дисциплине  
«Современные технологии программирования»  
на тему:

**«Информационно-справочная система общежития вуза»**

Выполнил:  
Студент группы ДПИ22-1  
Молчанов А.Е.

Научный руководитель:  
к.т.н., ст. преподаватель  
Пальчевский Е.В.

**Москва – 2025**

## Оглавление

Введение .....	3
1. Описание программы .....	5
1.1. Алгоритмические решения .....	5
1.2. Описание интерфейса программы .....	6
1.2.1. Поиск и фильтры .....	7
1.2.2. Дополнительные элементы управления системой .....	9
1.2.3. Вывод информации о студентах .....	10
1.2.4. Редактирование записи о студенте .....	12
1.2.5. Добавление нового студента .....	15
1.3. Архитектура приложения .....	16
1.3.1. Зависимости проекта .....	16
1.3.2. Клиент .....	18
1.3.3 База данных .....	19
2. Структура классов и их назначение в рамках проекта .....	21
2.1. Сервер .....	21
2.1.1. <i>StudentManagerApplication</i> .....	22
2.1.2. <i>MvcConfig</i> .....	22
2.1.3. Модели данных .....	22
2.1.4. Репозитории .....	22
2.1.5. Сервисный слой .....	22
2.1.6. <i>REST API</i> Контроллеры .....	22
2.2. Клиент .....	23
2.2.1. <i>Templates</i> .....	23
2.2.2. <i>Static</i> .....	23
Заключение .....	24
Список использованных источников .....	25

## **Введение**

В последние десятилетия потребность в эффективных информационно-справочных системах значительно возросла в связи с расширением образовательных учреждений, увеличением числа студентов и необходимостью автоматизации процессов управления. Вузы, как крупные образовательные и социальные организации, сталкиваются с множеством задач, требующих высокоорганизованной системы учета и управления. Одним из таких важных аспектов является управление и обслуживание общежитий, которые предоставляют студентам жилье на время их учебы. Эффективное управление общежитием требует надежной информационно-справочной системы, способной оперативно решать задачи учета, мониторинга и предоставления необходимых данных о жильцах.

Процесс управления общежитием требует аккуратного учета большого количества информации, включая сведения о студентах, комнатах, оплате проживания. В связи с этим, автоматизация этих процессов становится важным инструментом для обеспечения эффективного функционирования общежития и улучшения качества обслуживания студентов.

Целью данной курсовой работы является разработка информационно-справочной системы для управления общежитием вуза, которая обеспечит удобство и эффективность работы для администрации общежития. В рамках проекта будут решены следующие задачи:

- разработка клиент-серверного приложения на языке программирования Java для автоматизации процесса управления общежитием;
- создание серверной части приложения с использованием фреймворка Spring Boot, которая будет отвечать за обработку запросов от пользователей и предоставление необходимой информации;

- разработка клиентской части с графическим интерфейсом с помощью HTML, CSS и JavaScript, обеспечивающей удобное взаимодействие с сервером для пользователей и получения информации о проживающих студентах;

- реализация модели MVC (Model-View-Controller) для повышения структурности кода и удобства его сопровождения;

- создание базы данных с помощью MySQL для эффективного хранения и управления информацией о жильцах и ресурсах общежития.

Предложенная система будет не только облегчать повседневные задачи, связанные с управлением общежитием, но и обеспечит прозрачность и доступность информации для сотрудников вуза.

## 1. Описание программы

### 1.1. Алгоритмические решения

В разработанном веб-приложении используется архитектурный шаблон MVC (Model-View-Controller), который разделяет структуру системы на три взаимосвязанных компонента: модель (Model), представление (View) и контроллер (Controller). Это позволяет упростить поддержку и масштабирование проекта, а также обеспечивает удобное разделение логики, интерфейса и работы с данными (рисунок 1).

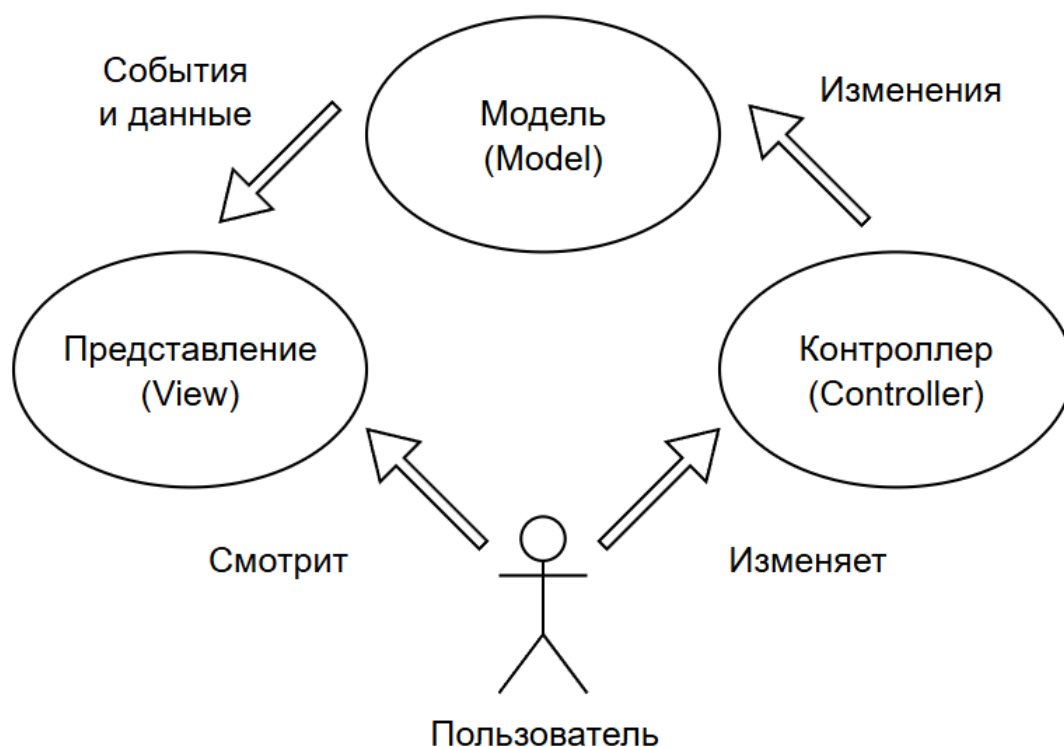


Рисунок 1 – Взаимодействие компонентов в паттерне MVC

Модель (Model) отвечает за работу с данными. В контексте данной системы она включает классы, представляющие основные сущности, например, Student, Room, и соответствующие репозитории (например, StudentRepository), которые содержат методы для выполнения операций с базой данных: получения, добавления, изменения и удаления данных о студентах.

Представление (View) отвечает за отображение данных пользователю. Оно реализовано с использованием HTML, CSS и JavaScript, предоставляя визуальный

интерфейс, через который пользователь может взаимодействовать с системой. Также в проекте применяется клиентская логика на JavaScript для динамического обновления интерфейса без полной перезагрузки страницы.

Контроллер (Controller) является связующим звеном между моделью и представлением. Он обрабатывает запросы от пользователя, вызывает соответствующие методы модели, и возвращает данные обратно в представление. Например, контроллер `StudentRestController` получает информацию о студентах по API-запросам и передаёт её на клиент.

Последовательность взаимодействия пользователя с системой:

- Пользователь открывает веб-приложение в браузере.
- Клиентская часть отправляет запрос к серверу по определённому маршруту (например, `/api/students`).
- Контроллер обрабатывает полученный запрос и, при необходимости, взаимодействует с репозиториями для получения или изменения данных в базе данных.
- После обработки данные возвращаются на клиентскую часть.
- JavaScript отображает полученные данные в пользовательском интерфейсе в виде таблицы или карточек.

При дальнейшем взаимодействии (поиск, фильтрация, редактирование, удаление) отправляются новые запросы, и процесс повторяется.

Таким образом, система предоставляет пользователю удобный способ взаимодействия с базой данных студентов общежития вуза через браузер, обеспечивая высокую отзывчивость и актуальность данных за счёт реализации архитектуры MVC и технологии одностраничного приложения (SPA).

## **1.2. Описание интерфейса программы**

В данном разделе будет описан интерфейс информационно-справочной системы общежития, разработанный с использованием технологий HTML, CSS и

JavaScript. Основной целью интерфейса является создание удобного и интуитивно понятного средства для взаимодействия пользователей с системой. Интерфейс включает в себя разнообразные элементы управления, такие как кнопки, текстовые поля, выпадающие списки, таблицы и другие компоненты, которые обеспечивают эффективное выполнение всех необходимых операций.

Проект реализован в формате SPA (Single Page Application), что позволяет загружать данные динамически без полной перезагрузки страницы. Такой подход обеспечивает более высокую скорость работы интерфейса и улучшает пользовательский опыт за счёт плавной и быстрой навигации по разделам системы.

### *1.2.1. Поиск и фильтры*

В интерфейсе реализован удобный поиск и фильтрация данных в верхней части экрана (рисунок 2).

Поиск по студентам общежития осуществляется с помощью ввода Фамилии, Имени или Отчества студента в поле ввода по центру экрана (рисунок 2). Для удобства пользователя поиск осуществляется сразу после ввода какого-либо символа, без необходимости дополнительно нажимать кнопку поиска.

Для снижения количества запросов, отправляемых на сервер при вводе текста в поле поиска, и, как следствие, уменьшения нагрузки на сервер, была реализована функция дебаунса (debounce) с задержкой в 300 миллисекунд. Это позволяет отправлять запрос только после того, как пользователь прекращает ввод на короткое время, предотвращая множественные обращения к серверу при каждом нажатии клавиши.

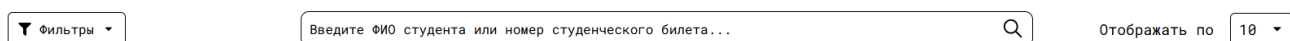


Рисунок 2 – Поиск и фильтры

Для расширения возможностей поиска и сортировки студентов, проживающих в общежитии, в интерфейсе реализован выпадающий список «Фильтры» (см. рисунок 3). При выборе одного из доступных фильтров

пользователем, рядом автоматически отображается дополнительный выпадающий список со значениями, соответствующими выбранному фильтру (рисунок 4). Эти значения динамически загружаются с сервера, что позволяет обеспечить актуальность данных и гибкость в настройке параметров поиска.

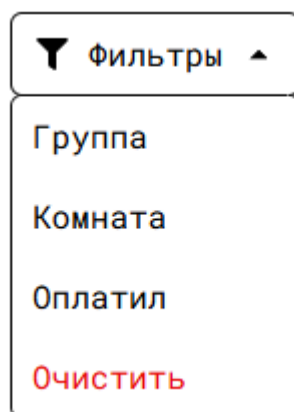


Рисунок 3 – Фильтры

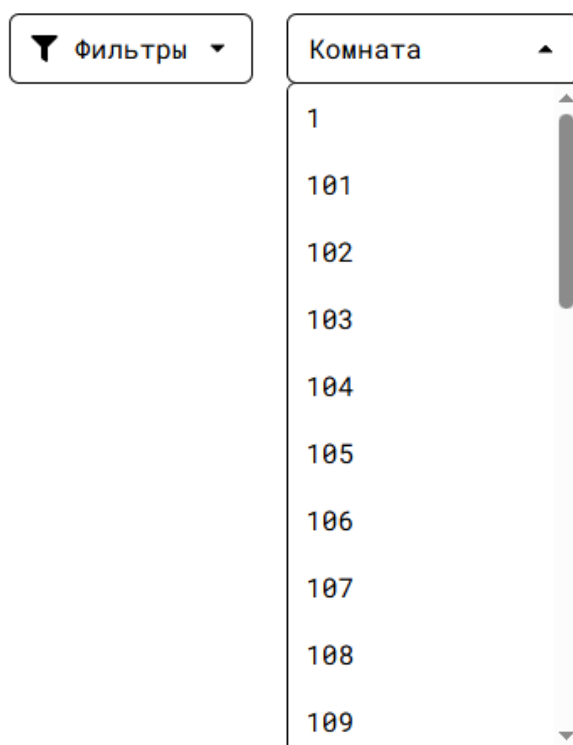


Рисунок 4 – Дополнительные фильтры



Также справа от поля ввода реализован выпадающий список «Отображать по» (рисунок 5), который позволяет пользователю выбрать количество студентов, отображаемых на одной странице. Это обеспечивает удобную навигацию по списку и позволяет адаптировать интерфейс под предпочтения пользователя, особенно при работе с большим объемом данных.

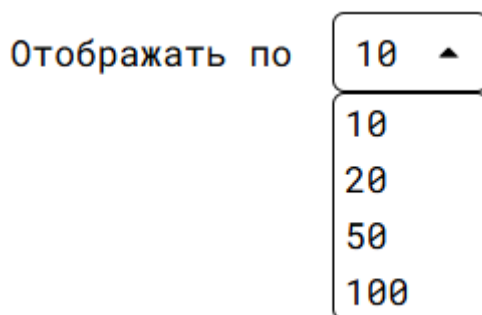


Рисунок 5 – Выпадающий список «Отображать по»

### *1.2.2. Дополнительные элементы управления системой*

В правой части экрана расположены дополнительные элементы управления, обеспечивающие расширенный функционал интерфейса.

— кнопка справки — при нажатии на кнопку со знаком вопроса (рисунок 6) открывается модальное окно «Инструкция по использованию информационно-справочной системы общежития». Оно содержит справочную информацию и рекомендации по работе с системой, что облегчает её освоение для новых пользователей.



Рисунок 6 – Кнопка вызова справки

— кнопки смены вида представления информации — при нажатии на кнопку с изображением таблицы или карточки (иконка меняется в зависимости от текущего режима) осуществляется переключение формата отображения основной информации о студентах (рисунок 7). Это позволяет пользователю выбрать

наиболее удобный способ визуализации данных: табличный (рисунок 9) или карточный (рисунок 10). Выбранный формат сохраняется в localStorage, что обеспечивает сохранение предпочтений пользователя при обновлении страницы или повторном входе в систему.



Рисунок 7 – Кнопки смены вида представления информации

— кнопка смены цветовой темы интерфейса — при нажатии на кнопку с изображением солнца или полумесяца (иконка меняется в зависимости от текущей темы) происходит переключение между светлой и тёмной цветовой темой страницы (рисунок 8). Это повышает комфорт работы с системой в разное время суток и адаптирует интерфейс под индивидуальные предпочтения пользователя. Текущая тема также сохраняется в localStorage, что позволяет системе автоматически применять выбранную пользователем тему при следующем открытии.



Рисунок 8 – Кнопки смены цветовой темы страницы

### *1.2.3. Вывод информации о студентах*

Для отображения информации о студентах, проживающих в общежитии, в разработанной системе реализованы два формата представления данных: табличный (рисунок 9) и карточный (рисунок 10).

Таблица обеспечивает наглядность, структурированность и удобство восприятия информации. Благодаря построчному отображению данных и разделению по столбцам пользователь может быстро находить нужную

информацию, сравнивать записи и ориентироваться даже при большом объёме данных.

Карточный формат обеспечивает визуальную выразительность и акцент на ключевой информации. Благодаря крупному отображению фамилии, имени, отчества и группы, карточки позволяют быстро идентифицировать студентов и использовать систему в условиях ограниченного экрана или при работе с небольшим количеством записей. Такой формат особенно удобен для пользователей со слабым зрением, так как крупный шрифт и чёткое визуальное разделение элементов упрощают восприятие информации.

Для повышения удобства восприятия обеих форматов предусмотрены визуальные элементы, позволяющие быстро выделять важные записи. Студенты, у которых статус «Оплатил» равен «Нет», автоматически выделяются розовым фоном как в таблице, так и в карточках. При использовании тёмной темы оформления цвет подсветки изменяется на оранжевый для сохранения контраста и читаемости. Это даёт возможность быстро визуальное определить наличие задолженностей по оплате проживания.

Постраничная навигация (пагинация) реализована для обоих представлений и расположена в нижней части интерфейса. Пользователь может переключаться между страницами без перегрузки интерфейса и без ожидания полной загрузки всей базы данных. Количество отображаемых студентов на одной странице регулируется через выпадающий список «Отображать по» (рисунок 5), что обеспечивает гибкость в работе с данными и повышает производительность системы.

Вся таблица формируется динамически на основе данных, полученных с сервера через API. Это обеспечивает актуальность отображаемой информации и высокую отзывчивость интерфейса при работе пользователя.

Фамилия	Имя	Отчество	Группа	Комната	Номер студенческого билета	Оплатил		
Петров	Антон	Сергеевич	ИБТ-21-7	128	001	Да		
Орлоа	Антон	Даниилович	ИБТ-21-4	1	667	Нет		
Скворцов	Артём	Вадимович	ИБТ-21-6	184	003	Да		
Сидоров	Тимур	Иванович	ИБТ-21-10	128	004	Нет		
Киселёв	Максим	Вадимович	ИБТ-21-5	111	005	Да		
Иванов	Глеб	Владимирович	ИБТ-21-4	101	006	Да		
Сидоров	Роман	Владимирович	ИБТ-21-3	109	007	Да		
Сидоров	Максим	Кириллович	ИБТ-21-10	130	008	Да		
Киселёв	Данила	Кириллович	ИБТ-21-2	120	009	Да		
Кондрашов	Иван	Артёмович	ИБТ-21-8	109	010	Да		

1 2 ... 14

Добавить студента

Рисунок 9 – Вывод информации о студентах в табличном формате

<b>Петров</b> Антон Сергеевич ИБТ-21-7	<b>Орлоа</b> Антон Даниилович ИБТ-21-4	<b>Скворцов</b> Артём Вадимович ИБТ-21-6
<b>Сидоров</b> Тимур Иванович ИБТ-21-10	<b>Киселёв</b> Максим Вадимович ИБТ-21-5	<b>Иванов</b> Глеб Владимирович ИБТ-21-4
<b>Сидоров</b> Роман Владимирович ИБТ-21-3	<b>Сидоров</b> Максим Кириллович ИБТ-21-10	<b>Киселёв</b> Данила Кириллович ИБТ-21-2
<b>Кондрашов</b> Иван Артёмович ИБТ-21-8	<b>Чесноков</b> Тимур Иванович ИБТ-21-1	<b>Петров</b> Тимур Григорьевич ИБТ-21-9

Рисунок 10 – Вывод информации о студентах в карточном формате

#### 1.2.4. Редактирование записи о студенте

При нажатии на кнопку с изображением карандаша (в табличном представлении информации) или на саму карточку студента (в карточном

представлении) открывается модальное окно редактирования данных (рисунок 11). В этом окне представлены кнопки «Редактировать» и «Сохранить».

По умолчанию поля недоступны для редактирования — это сделано с целью предотвращения случайного изменения информации. Чтобы внести изменения, необходимо сначала нажать кнопку «Редактировать». После внесения изменений следует нажать кнопку «Сохранить». Без этого действия новые данные не будут сохранены, и изменения в базу данных не попадут.

Перед отправкой данных на сервер выполняется очистка пробелов и проверка на заполненность всех полей, чтобы избежать сохранения пустых значений. После этого на серверной стороне выполняется валидация данных:

- проверка уникальности номера студенческого билета — если номер уже существует в базе данных, сервер возвращает ошибку с соответствующим сообщением (рисунок 12).

- проверка корректности номера комнаты — если указанная комната отсутствует в базе данных, сервер возвращает ошибку.

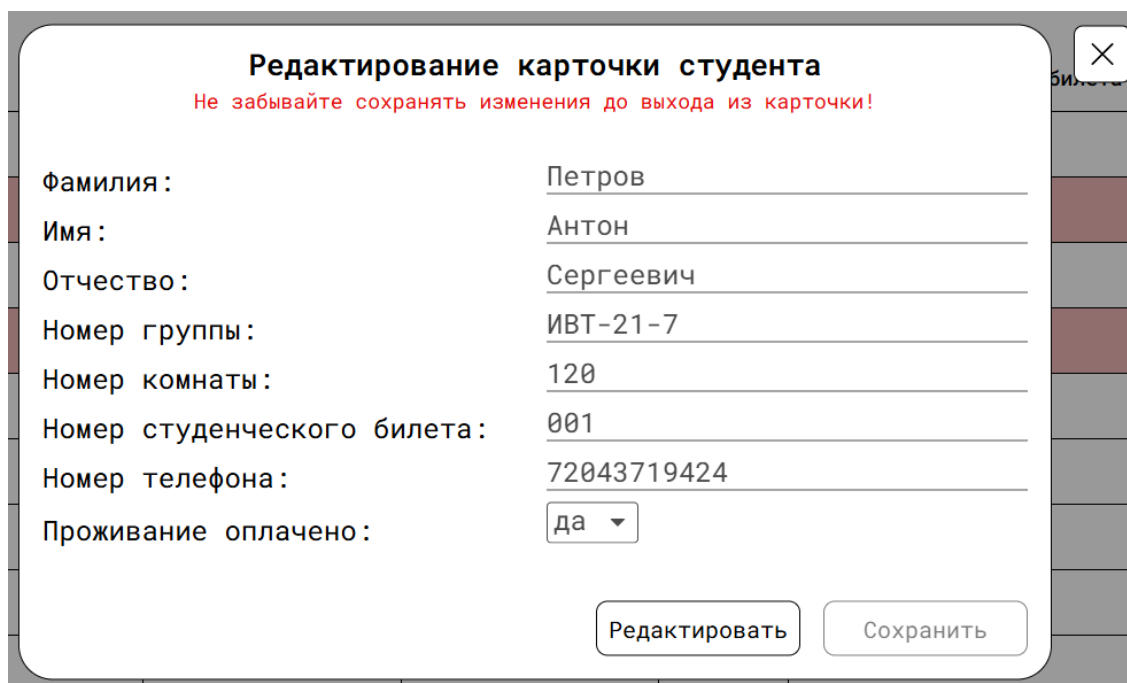
- проверка количества мест в комнате — если в комнате, рассчитанной на 4 места, уже проживает 4 студента, сервер не позволяет добавить нового студента и возвращает ошибку о переполнении комнаты.

Клиентская сторона обрабатывает статус и текст ошибки, полученные от сервера, и отображает их пользователю в соответствующем формате.

Редактирование информации о комнате недоступно обычному пользователю по соображениям безопасности и целостности данных.

Комнаты являются частью фиксированной структуры, и их изменение может повлиять на множество связанных записей в системе. Поэтому такие действия доступны исключительно разработчику. Это ограничение помогает предотвратить случайные ошибки, нарушение логики распределения студентов и гарантирует стабильную работу системы. Кроме того, ограничение на редактирование

информации о комнате исключает возможность злоупотреблений, таких как мошенничество с данными или подселение несуществующих студентов в уже полностью занятые комнаты. Таким образом, система обеспечивает не только техническую, но и административную надёжность.

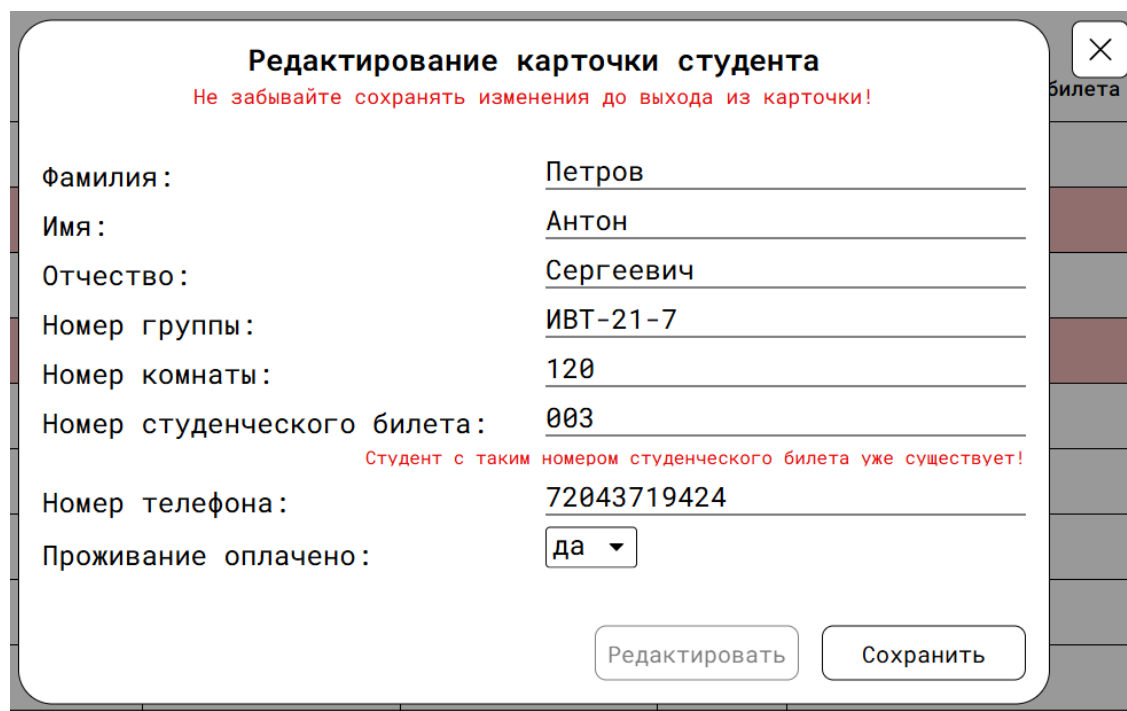


The screenshot shows a modal window titled "Редактирование карточки студента" (Editing student card). Below the title is a red warning message: "Не забывайте сохранять изменения до выхода из карточки!" (Don't forget to save changes before leaving the card!). The form contains the following fields and values:

Field	Value
Фамилия:	Петров
Имя:	Антон
Отчество:	Сергеевич
Номер группы:	ИВТ-21-7
Номер комнаты:	120
Номер студенческого билета:	001
Номер телефона:	72043719424
Проживание оплачено:	да

At the bottom right, there are two buttons: "Редактировать" (Edit) and "Сохранить" (Save).

Рисунок 11 – Модальное окно редактирования записи о студенте



This screenshot shows the same modal window as Figure 11, but with an error message displayed below the "Номер студенческого билета:" (Student ID number) field. The error message is: "Студент с таким номером студенческого билета уже существует!" (Student with this student ID number already exists!). The other fields and values remain the same as in Figure 11.

Field	Value
Фамилия:	Петров
Имя:	Антон
Отчество:	Сергеевич
Номер группы:	ИВТ-21-7
Номер комнаты:	120
Номер студенческого билета:	003
Номер телефона:	72043719424
Проживание оплачено:	да

The error message is: "Студент с таким номером студенческого билета уже существует!"

Buttons: "Редактировать" (Edit) and "Сохранить" (Save).

Рисунок 12 – Пример отображения ошибки

### 1.2.5. Добавление нового студента

При нажатии на кнопку «Добавить студента» в нижней части экрана открывается модальное окно для внесения данных о новом студенте (рисунок 13). В отличие от окна редактирования, здесь доступна только кнопка «Сохранить», так как все поля изначально открыты для заполнения.

Перед сохранением система проверяет введённые данные по тем же правилам, что и при редактировании:

- обязательные поля не должны быть пустыми;
- номер студенческого билета должен быть уникальным;
- указанная комната должна существовать в базе;
- в комнате должно быть свободное место.

Если проверка пройдена успешно, запись добавляется в базу данных, модальное окно закрывается, а новый студент сразу появляется в общем списке — в виде строки таблицы или карточки, в зависимости от выбранного режима представления информации. Теперь его можно редактировать или удалять, как и других студентов.

**Добавить студента**

Не забывайте сохранять изменения до выхода из карточки!

Фамилия: \_\_\_\_\_

Имя: \_\_\_\_\_

Отчество: \_\_\_\_\_

Номер группы: \_\_\_\_\_

Номер комнаты: \_\_\_\_\_

Номер студенческого билета: \_\_\_\_\_

Номер телефона: \_\_\_\_\_

Проживание оплачено:

Рисунок 13 – Модальное окно создания записи о новом студенте

### 1.2.6. Удаление записи о студенте

При нажатии на иконку с изображением человека и знака минуса в табличном представлении информации, либо на крестик в карточном формате, открывается модальное окно подтверждения удаления записи студента из базы данных (рисунок 14).

Модальное окно содержит основные идентификационные данные студента — фамилию, имя и отчество, а также две кнопки: «Да» и «Отмена».

При нажатии на кнопку «Да» на сервер отправляется запрос на удаление соответствующей записи. После подтверждения студент исключается из базы данных.

При нажатии на «Отмена» окно закрывается, и операция удаления не выполняется.

Такой механизм реализован для предотвращения случайного удаления данных, а также для повышения безопасности и осознанности действий пользователя при работе с системой.

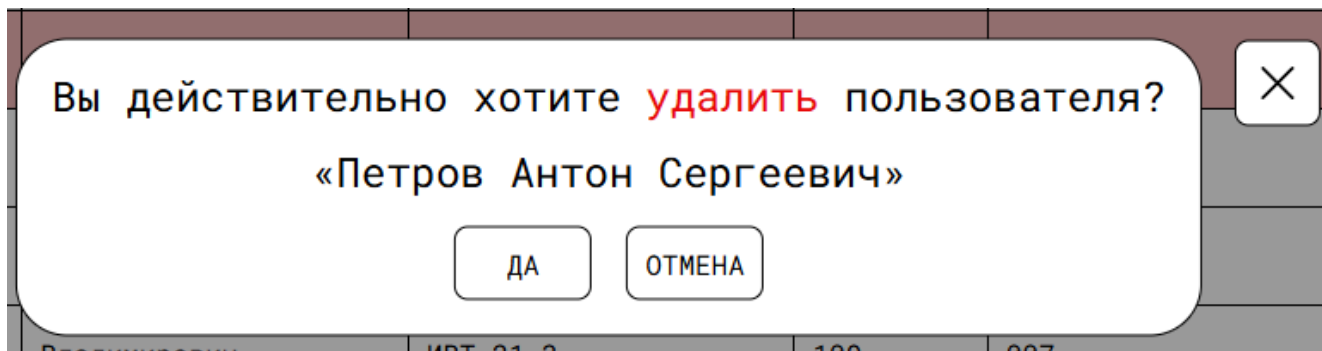


Рисунок 14 – Модальное окно удаления записи о студенте

## 1.3. Архитектура приложения

### 1.3.1. Зависимости проекта

В файле build.gradle представлены все необходимые зависимости проекта, которые определяют используемые технологии и фреймворки. Ниже приведен анализ ключевых зависимостей:



Основные технологические стеки:

Spring Boot Starter-ы:

- spring-boot-starter-web - базовая зависимость для создания веб-приложений (включает Tomcat, Spring MVC)
- spring-boot-starter-data-jpa - реализация ORM для работы с БД через Hibernate
- spring-boot-starter-security - система аутентификации и авторизации
- spring-boot-starter-graphql - поддержка GraphQL API
- spring-boot-starter-webflux - реактивное программирование
- Интеграционные модули:
- vaadin-spring-boot-starter - фреймворк для веб-интерфейсов
- graphql-dgs-spring-graphql-starter - Netflix DGS для GraphQL
- spring-modulith-starter-core - модульная архитектура приложения

Специфические зависимости:

- jakarta.persistence-api:3.1.0 - спецификация JPA 3.1
- thymeleaf-extras-springsecurity6 - интеграция Thymeleaf и Spring Security
- htmx-spring-boot-thymeleaf - библиотека HTMX для динамических интерфейсов
- spring-data-rest-hal-explorer - автоматическое REST API для репозитория

СУБД и драйверы:

- mysql-connector-j — драйвер MySQL
- ojdbc11 — драйвер Oracle
- mariadb-java-client — драйвер MariaDB

Система сборки:

- Плагин `com.netflix.dgs.codegen` - генерация кода GraphQL
- Плагин `gg.jte` - шаблонизатор JTE
- Плагин `com.vaadin` - поддержка Vaadin
- Нативная сборка через `org.graalvm.buildtools.native`

Ключевые особенности конфигурации:

- Управление версиями через BOM (рисунок 15)
- Генерация кода GraphQL-клиента из схемы (generateJava)
- Шаблонов JTE с предкомпиляцией (jte)

Данная конфигурация зависимостей позволяет реализовать:

- Многослойную архитектуру приложения
- Поддержку REST, GraphQL и WebSocket
- Реактивное программирование
- Современные веб-интерфейсы (Vaadin + HTMX)
- Гибкую работу с различными СУБД

Полный список зависимостей содержит 28 элементов, что обеспечивает комплексную функциональность системы управления студенческим общежитием.



```

<dependencyManagement>{
  <imports>{
    <mavenBom> "org.springframework.modulith:spring-modulith-bom:${springModulithVersion}"
    <mavenBom> "com.netflix.graphql.dgs:graphql-dgs-platform-dependencies:${netflixDgsVersion}"
    <mavenBom> "com.vaadin:vaadin-bom:${vaadinVersion}"
  }
}

```

Рисунок 15 – Управление версиями через BOM

### 1.3.2. Клиент

Клиентская часть приложения реализована как одностраничное приложение (SPA) с использованием современного стека веб-технологий. Основу интерфейса составляют HTML5, CSS3 и JavaScript (ES6+), которые взаимодействуют с серверной частью через REST API.

HTML5 используется для создания семантической структуры приложения, обеспечивая корректную разметку контента и улучшая доступность для пользователей и поисковых систем. Весь контент на страницах обновляется динамически, что является характерной особенностью SPA-архитектуры. Это позволяет избежать полной перезагрузки страницы и обеспечивает более плавное взаимодействие с пользователем.

CSS3 разработан с использованием методологии БЭМ (Блок-Элемент-Модификатор), что позволяет организовать стили по компонентному принципу. Такой подход способствует поддержанию чистоты и удобочитаемости кода, минимизирует вероятность конфликтов между стилями и облегчает масштабирование и поддержку проекта. Каждый компонент приложения стилизован независимо, что делает систему гибкой и упрощает внесение изменений в будущем.

JavaScript (ES6+) используется для реализации всей динамической логики на клиентской стороне. Современные возможности языка, такие как асинхронные функции и стрелочные функции, позволяют создавать чистый и эффективный код. Взаимодействие с сервером осуществляется через REST API, что обеспечивает легкость и гибкость интеграции с серверной частью и другими внешними сервисами.

### *1.3.3 База данных*

Настройки подключения к базе данных и таблицам находятся в файле `application.properties` (рисунок 16) в проекте. В этом файле определяются параметры подключения к базе данных MySQL, а также настройки для работы с JPA (Java Persistence API).

```

1  vaadin.launch-browser=true
2  spring.application.name=StudentDormitory
3  spring.datasource.url=jdbc:mysql://localhost:3306/dormitory
4  spring.datasource.username=root
5  spring.datasource.password=root
6
7  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
8
9  spring.jpa.hibernate.ddl-auto=none
10 spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
11
12 logging.level.org.hibernate.SQL=DEBUG
13 logging.level.org.hibernate.type=TRACE
14
15 spring.security.user.name=root
16 spring.security.user.password=root
17 spring.security.user.roles=manager
18
19 spring.cache.type=none
20 spring.web.resources.add-mappings=true

```

Рисунок 16 – Файл настроек подключения

В данном примере настройки подключения к базе данных MySQL и использование JPA для работы с базой данных прописаны следующим образом:

`spring.datasource.url` — URL-адрес для подключения к базе данных MySQL (в данном случае подключение осуществляется к базе данных `dormitory`, расположенной на локальном сервере).

— `spring.datasource.username` — имя пользователя базы данных (`root`).

— `spring.datasource.password` — пароль пользователя базы данных (`root`).

— `spring.jpa.properties.hibernate.dialect` — диалект Hibernate для работы с MySQL 8.

— `spring.jpa.hibernate.ddl-auto` — настройка для автоматического обновления схемы базы данных (в данном случае установлено значение `none`, что означает отсутствие автоматического изменения схемы).

— `spring.jpa.hibernate.naming.physical-strategy` — настройка для стандартной стратегии именования физических таблиц и колонок.

Также в файле настроек можно встретить следующие блоки:

—`logging.level.org.hibernate.SQL` и `logging.level.org.hibernate.type` — для включения детализированного логирования SQL-запросов и типов данных.

—`spring.security.user.name`, `spring.security.user.password` и `spring.security.user.roles` — настройки безопасности для пользователя, который имеет доступ к приложению (пользователь `root` с ролью `manager`).

—`spring.cache.type` — настройка кэширования (в данном случае кэширование отключено).

—`spring.web.resources.add-mappings` — настройка для обработки статических ресурсов.

Эти параметры позволяют приложению подключаться к базе данных MySQL и использовать JPA для взаимодействия с данными, а также задают дополнительные настройки для логирования и безопасности.

## 2. Структура классов и их назначение в рамках проекта

### 2.1. Сервер

Серверная часть реализована с использованием Spring Boot и состоит из 6 классов и 2 интерфейсов (рисунок 17). Архитектура приложения следует принципам MVC (Model-View-Controller) и разделению ответственности между компонентами.

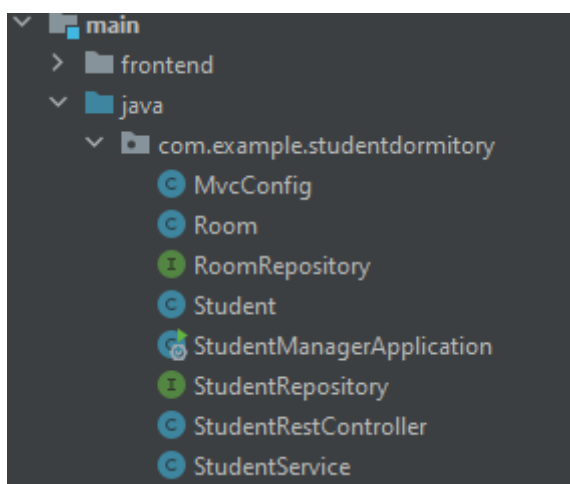


Рисунок 17 – Структура классов

### *2.1.1. StudentManagerApplication*

Главный класс приложения, содержащий метод `main()`. Отвечает за запуск Spring Boot приложения и автоматическую конфигурацию всех компонентов. Аннотация `@SpringBootApplication` объединяет три основные функции: автоконфигурацию, сканирование компонентов и возможность определения дополнительных бинов.

### *2.1.2. MvcConfig*

Класс `MvcConfig` является конфигурационным компонентом Spring MVC, реализующим интерфейс `WebMvcConfigurer`. В текущей реализации класс служит заготовкой для будущей настройки веб-слоя приложения, но пока не содержит активных конфигураций. Пустой метод `addViewControllers()` специально оставлен для возможного расширения функциональности при развитии проекта.

### *2.1.3. Модели данных*

`Student` - представляет сущность студента, проживающего в общежитии

`Room` - сущность, описывающая комнату в общежитии

### *2.1.4. Репозитории*

`StudentRepository` - интерфейс для работы со студентами в БД, расширяет `JpaRepository`

`RoomRepository` - интерфейс для управления комнатами, также расширяет `JpaRepository`

### *2.1.5. Сервисный слой*

Класс `StudentService` является центральным компонентом бизнес-логики приложения, обрабатывающим все операции, связанные со студентами.

### *2.1.6. REST API Контроллеры*

`StudentRestController` - обрабатывает HTTP-запросы и предоставляет API для CRUD операций со студентами, поиска и фильтрации

## 2.2. Клиент

Всего на стороне клиента используется 1 HTML файл, 1 файл иконки, 1 CSS файл стилей и 1 JavaScript файл (рисунок 17).

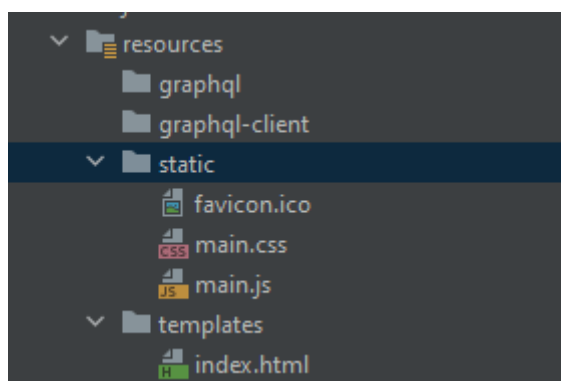


Рисунок 17 – Структура клиентской части

### 2.2.1. Templates

В данной папке содержится один HTML файл — `index.html`, который является статической частью для SPA приложения. Этот файл содержит основную структуру интерфейса, включая разметку модальных окон, кнопок, заголовков для таблиц и оболочек, в которые позже будет динамически добавляться HTML-код через `main.js`. Важно, что `index.html` служит только как базовая структура, и динамическое обновление контента происходит с использованием JavaScript.

### 2.2.2. Static

В папке `static` содержатся два файла:

`main.js` — это файл, отвечающий за логику клиентской части приложения. Он управляет состоянием модальных окон, отрисовкой информации о студентах в нужном формате и отправкой запросов на сервер. Основная задача этого скрипта — взаимодействие с сервером и динамическое обновление контента на странице без необходимости перезагрузки.

`main.css` — это файл стилей для клиентской части приложения. Он содержит все стили, которые применяются к элементам на странице.

## **Заключение**

В результате выполнения курсовой работы было разработано веб-приложение для управления данными студентов в общежитии с использованием фреймворка Spring. Основной целью проекта было создать систему, которая позволяет эффективно управлять информацией о студентах, их размещении в комнатах и оплате проживания, а также обеспечивать удобный интерфейс для взаимодействия с данными.

Для достижения этой цели были выполнены следующие задачи: разработка клиент-серверного приложения на языке Java, создание серверной части приложения с использованием фреймворка Spring Boot, разработка клиентской части приложения с использованием HTML, CSS и JavaScript, а также внедрение принципов архитектуры MVC для разделения логики приложения на отдельные компоненты. Также была реализована связь между фронтенд- и бэкенд-частью через REST API.

Результатом работы стало создание удобного и функционального веб-приложения, которое позволяет эффективно управлять данными студентов, отслеживать их размещение в комнатах и контролировать состояние оплаты проживания. Разработанное приложение может быть использовано в учебных заведениях и других организациях для автоматизации работы с общежитиями.

Кроме того, во время выполнения курсовой работы были приобретены полезные навыки разработки веб-приложений с использованием фреймворка Spring, а также опыт работы с клиент-серверной архитектурой.



### **Список использованных источников**

1. Walls, C. Spring в действии / C. Walls. - М.: ДМК Пресс, 2019. - 616 с.
2. Sharma, R., Chopra, K. Spring 5 для профессионалов / R. Sharma, K. Chopra. - М.: Питер, 2018. - 624 с.
3. Long, J. Spring Boot в действии / J. Long. - М.: ДМК Пресс, 2018. - 432 с.
4. Bishop, S. RESTful Web Services / S. Bishop. - 2nd ed. - O'Reilly Media, 2014. - 360 с.
5. Crockford, D. JavaScript: The Good Parts / D. Crockford. - O'Reilly Media, 2008. - 176 с.
6. Duckett, J. HTML5 and CSS3: Design and Build Websites / J. Duckett. - Wiley, 2011. - 512 с.