МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Филиал федерального государственного бюджетного образовательного учреждения высшего образования

«Национальный исследовательский университет «МЭИ» в г. Смоленске

Кафедра вычислительной техники

УДК: 004.9 (079.2)

П-68

**Прядихин Максим Юрьевич**

СИСТЕМА УПРАВЛЕНИЯ ГОСТИНИЧНЫМ КОМПЛЕКСОМ

Выпускная квалификационная работа на соискание квалификации

**Бакалавр**

Направление подготовки

**09.03.01 – Информатика и вычислительная техника**

Профиль подготовки

**Программное обеспечение средств вычислительной техники и автоматизированных систем**

Студент группы ПО-20 М.Ю. Прядихин

Научный руководитель

к.т.н., доцент Я.А. Федулов

*Допускается к защите*

Заведующий кафедрой

д.т.н., профессор А.С. Федулов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2024 г.

Смоленск – 2024

**Аннотация**

Система управления гостиничным комплексом. Прядихин Максим Юрьевич. Работа содержит 83 страницы, 38 иллюстраций, 4 приложения, 7 таблиц. Год выполнения: 2024.

Данная работа посвящена разработке системы управления гостиничным комплексом с использованием Java, Spring, Hibernate, PostgreSQL и Docker. Описаны функциональные и нефункциональные требования, спроектированы и реализованы основные модули и компоненты приложения, включая управление номерами, клиентами и персоналом. Проведено модульное и интеграционное тестирование, а также нагрузочное тестирование с Apache JMeter. Приложение развернуто в Docker-контейнерах для обеспечения портативности и легкости развертывания. Система надежна и масштабируема, удовлетворяет потребности современных гостиниц.

**Annotation**

Hotel Management System. Pryadikhin Maksim Yuryevich. The work consists of 83 pages, 38 illustrations, 4 appendices and 7 tables. Year of completion: 2024.

This work is dedicated to the development of a hotel management system using Java, Spring, Hibernate, PostgreSQL, and Docker. Functional and non-functional requirements are described, and the main modules and components of the application are designed and implemented, including room, client, and staff management. Unit and integration testing, as well as load testing with Apache JMeter, are conducted. The application is deployed in Docker containers for portability and ease of deployment. The system is reliable and scalable, meeting the needs of modern hotels.

**Содержание**

[ВВЕДЕНИЕ 5](#_Toc169175892)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc169175893)

[1.1 Обзор гостиничного бизнеса и его особенностей 6](#_Toc169175894)

[1.2 Обзор существующих систем управления гостиничным комплексом 6](#_Toc169175895)

[1.3 Постановка задачи разработки 8](#_Toc169175896)

[1.4 Идентификация требований к системе управления 10](#_Toc169175897)

[1.5 Технологический обзор для разработки веб-приложения 11](#_Toc169175898)

[1.6 Выводы по главе 12](#_Toc169175899)

[2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНЫХ СРЕДСТВ 14](#_Toc169175900)

[2.1 Формулирование функциональных требований к веб-приложению 14](#_Toc169175901)

[2.2 Проектирование архитектуры системы 14](#_Toc169175902)

[2.2.1 Возможности пользователей 14](#_Toc169175903)

[2.2.2 Диаграмма классов 16](#_Toc169175904)

[2.2.3 Диаграмма компонентов 20](#_Toc169175905)

[2.2.4 Диаграмма последовательности 23](#_Toc169175906)

[2.2.5 Блок схема алгоритма валидации бронирования дат 25](#_Toc169175907)

[2.3 Проектирование структуры базы данных 28](#_Toc169175908)

[2.3.1 Физическая модель 28](#_Toc169175909)

[2.4 Проектирование конфигурации и развертывания приложения в Docker-контейнерах 31](#_Toc169175910)

[2.5 Выводы по главе 32](#_Toc169175911)

[3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 34](#_Toc169175912)

[3.1 Начальная конфигурация проекта 34](#_Toc169175913)

[3.2 Создание архитектурного шаблона MVC и его компонентов 36](#_Toc169175914)

[3.3 Создание основных модулей и компонентов веб-приложения на Java с использованием Spring Framework 40](#_Toc169175915)

[3.4 Реализация взаимодействия c базой данных PostgreSQL 47](#_Toc169175916)

[3.5 Создание страниц представлений 48](#_Toc169175917)

[3.6 Развертывания приложения в Docker-контейнере 50](#_Toc169175918)

[3.7 Выводы по главе 51](#_Toc169175919)

[4 ТЕСТИРОВАНИЕ И ОЦЕНКА КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 53](#_Toc169175920)

[4.1 Планирование и проведение модульного тестирования Java-классов и компонентов Spring-приложения 53](#_Toc169175921)

[4.2 Интеграционное тестирование приложения 55](#_Toc169175922)

[4.3 Тестирование производительности с использованием Apache JMeter для анализа нагрузки на сервер и оценки производительности приложения 58](#_Toc169175923)

[4.4 Тестирование базового пути 60](#_Toc169175924)

[4.5 Оценка качества программного продукта 64](#_Toc169175925)

[4.6 Выводы по главе 79](#_Toc169175926)

[ЗАКЛЮЧЕНИЕ 81](#_Toc169175927)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 82](#_Toc169175928)

[ПРИЛОЖЕНИЕ А (ТЕХНИЧЕСКОЕ ЗАДАНИЕ) 84](#_Toc169175929)

[ПРИЛОЖЕНИЕ Б (КОД ПРОГАММЫ) 90](#_Toc169175930)

[ПРИЛОЖЕНИЕ В (КОНФИГУРАЦИОННЫЕ ФАЙЛЫ) 107](#_Toc169175931)

[ПРИЛОЖЕНИЕ Г (HTML ПРЕДСТАВЛЕНИЯ) 110](#_Toc169175932)

[ПРИЛОЖЕНИЕ Д (КОД БАЗЫ ДАННЫХ) 130](#_Toc169175933)

ВВЕДЕНИЕ

Ежегодно гостиничный бизнес становится более требовательным к качеству обслуживания. Современные гостиничные комплексы нуждаются в эффективных системах управления, которые способны автоматизировать процессы бронирования, учета гостей, финансового учета и других аспектов их деятельности. Веб-приложения становятся важным инструментом для управления гостиничным бизнесом, предоставляя гибкость, масштабируемость и доступность.

Целью данной выпускной квалификационной работы является разработка веб-приложения, которое будет выполнять функции системы управления гостиничным комплексом. Проект будет выполнен с использованием Java, Spring, Hibernate ORM, PostgreSQL и Docker, Redis, что обеспечит надежность, производительность и масштабируемость разработанной системы.

В рамках работы будет проведен анализ предметной области гостиничного бизнеса, определены его основные потребности и требования, а также выбраны наиболее подходящие технологии для её разработки. После будет выполнено проектирование, реализация и тестирование системы, с целью обеспечения её функциональности, надежности и производительности.

Результаты данной работы будут полезны как для гостиничного бизнеса, желающего внедрить современную систему управления, так и для разработчиков, которые заинтересованы в создании веб-приложений для подобных предприятий.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

**1.1 Обзор гостиничного бизнеса и его особенностей**

Гостиничный бизнес – отрасль сферы услуг, направленная на обеспечение потребителя жильем, питанием и связанными услугами бытового обслуживания.

Целью данного бизнеса является удовлетворением потребительских потребностей, за счет обеспечения высокого уровня комфорта и получение прибыли за оказанные услуги.

Гостиница – предприятие, задачей которого является размещении посетителей в номерах и оказании им стандартных услуг.

Особенности гостиничного бизнеса:

* Сезонность и спрос: гостиничный бизнес довольно часто зависит от сезонности, что требует гибкости в управлении загрузкой и регулированием ценообразования.
* Конкуренция: из-за высокой конкуренции в данной сфере деятельности необходимо постоянно совершенствовать сервис и создавать новые предложения.
* Управление ресурсами: эффективное использование персонала, помещений, оборудования и других ресурсов играет важную роль в обеспечении прибыльности и конкурентоспособности гостиничного бизнеса.

Для успешного функционирования гостиничного комплекса необходима система управления, способная автоматизировать процессы бронирования, учета гостей, финансового учета и других аспектов бизнеса.

1.2 Обзор существующих систем управления гостиничным комплексом

На сегодняшний день на рынке представлено большое количество систем для управления гостиничным бизнесом. Рассмотрим некоторые из них, а именно системы: OtelMS, Logus HMS и Эдельвейс.

Ниже приведен их краткий обзор с выделением достоинств и недостатков:

**OtelMS** – профессиональная система управления отелем, автоматизирующая бизнес-процессы.

*Достоинства*:

* Широкий функционал: данный продукт предлагает широкий набор функций, таких как бронирование, финансовый учет, управление персоналом и другие.
* Гибкость настройки: возможность настройки системы под потребности конкретного предприятия.
* Интеграция: система позволяет интегрироваться с другими программными сервисами и продуктами.

*Недостатки*:

* Как было описано выше, продукт имеет широкий функционал, но для некоторых типов бизнеса он может быть недостаточным или же избыточным.
* Сложность использования: для новых пользователей может потребоваться время на освоение интерфейса и всех доступных функций.

**Logus HMS** – Российская система управления отелем, созданная с учетом специфики российской гостиничной индустрии.

*Достоинства*:

* Простота использования: программа предоставляет простой и интуитивно понятный интерфейс.
* Внимание на базовые функции: продукт может быть сконцентрирован на основных функция, что привлекательно для небольших гостиниц.

*Недостатки*:

* Ограниченный функционал: по сравнению с более мощными системами, функционал может быть ограничен.
* Ограниченная гибкость настройки: некоторый функционал имеет менее гибкие настройки, по сравнению со своими конкурентами.

**Эдельвейс –** профессиональная система управления отелем с широким набором функциональных модулей.

*Достоинства*:

* Интеграция с дополнительными сервисами: Эдельвейс предоставляет возможность интеграции с различными сервисами.
* Гибкость настройки: дает возможность настройки под конкретные потребности бизнеса.

*Недостатки*:

* Сложность использования: интерфейс является менее интуитивно понятным и требует обучения новых сотрудников.
* Ограниченный функционал: для некоторых типов бизнеса количество функций может оказаться недостаточным.

Несмотря на существующие решения на рынке, создание собственной системы управления гостиничным комплексом позволит нам адаптировать её под уникальные потребности нашего бизнеса. Мы сможем полностью адаптировать функционал под особенности бизнеса, избежим излишней сложности, которая может присутствовать в готовых системах. А также собственная разработка даст возможность гибко управлять развитием и сопровождением системы в будущем.

1.3 Постановка задачи разработки

Современные системы управления гостиничным комплексом (Property Management Systems, PMS) обладают различным функционалом и уровнем сложности, однако не всегда могут удовлетворить специфические потребности каждой гостиницы. В процессе анализа существующих решений, таких как OtelMS, Logus HMS и Эдельвейс, были выявлены следующие основные недостатки:

- Высокая стоимость лицензий: Многие существующие системы требуют значительных затрат на приобретение и последующее обслуживание.

Ограниченная кастомизация: Большинство систем сложно адаптируются под специфические потребности отдельных гостиничных комплексов, что снижает их гибкость.

- Сложность в использовании: Некоторые системы требуют длительного обучения персонала, что увеличивает временные и финансовые затраты на внедрение.

На основании выявленных недостатков, была поставлена задача разработки новой системы управления гостиничным комплексом, которая должна удовлетворять следующим требованиям:

1. Экономичность:

- Минимизация затрат на приобретение и обслуживание системы.

- Открытая лицензия или разумная стоимость для небольших гостиничных комплексов.

2. Кастомизация:

- Возможность гибкой настройки системы под индивидуальные потребности каждой гостиницы.

- Поддержка пользовательских настроек и расширений без необходимости значительных изменений в коде.

3. Простота использования:

- Интуитивно понятный интерфейс для минимизации времени обучения персонала.

- Простой и удобный процесс установки и настройки системы.

4. Масштабируемость:

- Возможность расширения функционала по мере роста и развития гостиничного бизнеса.

- Поддержка многопользовательского режима и распределенной архитектуры для обеспечения высокой производительности и отказоустойчивости.

Реализация новой системы должна основываться на современных технологиях и фреймворках, таких как Java, Spring, Hibernate, PostgreSQL и Docker. Это позволит обеспечить высокую производительность, надежность и безопасность системы, а также упростит процесс ее развертывания и эксплуатации.

В результате выполнения данной задачи будет создана система управления гостиничным комплексом, которая будет соответствовать требованиям современного гостиничного бизнеса и обеспечит высокое качество обслуживания клиентов.

**1.4 Идентификация требований к системе управления**

Для успешного создания собственной системы управления гостиничным комплексом необходимо анализ потребностей и выявить основные требования к функциональности и характеристикам системы. Это важно для того, чтобы система была максимально эффективной и соответствовала бизнес-процессам и целям комплекса. Ниже приведены основные аспекты, которым должна удовлетворять система:

*Бронирование и управление номерами:*

* Возможность бронирования номеров.
* Система должна вести учет недоступных и свободных номеров, а также позволять управлять их статусом.

*Учёт гостей и услуг:*

* Создание и хранение профилей гостей, их персональных данных.
* Возможность быстрого поиска информации о гостях.
* Учет предпочтений и специальных запросов клиентов.
* Возможность предоставления дополнительных услуг.

*Финансовый учёт и отчетность:*

* Учет оплаты за проживание и дополнительные услуги.
* Возможность генерации финансовой отчетности.
* Формирование статистики о загрузке отеля, доходности, оценках гостей и т.д.

*Управление персоналом:*

* Учет рабочего времени сотрудников, графики работы, отпусков и смен.
* Возможность контроля доступа к функционалу системы для персонала.
* Распределение задач и обязанностей.

*Безопасность и конфиденциальность:*

* Защита персональных данных гостей в соответствии с требованиями законодательства.
* Обеспечение безопасности данных от несанкционированного доступа.

Идентификация и учет потребностей необходимы для разработки системы управления гостиничным комплексом, которая будет эффективно поддерживать деятельность комплекса, повышать удовлетворенность гостей и улучшать финансовые показатели бизнеса.

**1.5 Технологический обзор для разработки веб-приложения**

При разработке веб-приложения для управления гостиничным комплексом необходимо выбрать подходящие технологии и инструменты, которые смогут обеспечить эффективную и надежную работу системы. Далее приведен перечень основных технологий, которые будут использованы в разработке данного проекта.

*Java:* язык программирования Java будет использован для написания бизнес-логики приложения. Java обладает высокой производительностью, масштабируемостью и большим сообществом разработчиком.

*Spring Framework:* является одним из самых популярных фреймворков для разработки веб-приложений на языке Java. Он предоставляет большое количество инструментов для управления зависимостями, обеспечения безопасности, разработки RESTful API и других аспектов приложения.

*Spring Data:* предоставляет удобный интерфейс, для облегчения работы с базами данных. С его помощью можно сократить количество шаблонного кода и упростить реализацию слоя доступа к данным.

*Spring Security:* обеспечивает средства авторизации и аутентификации в веб-приложениях. С его помощью можно настроить защиту ресурсов, контролировать доступ пользователей и управлять их аутентификацией.

*Hibernate:* используется в качестве ORM инструмента для работы с базой данных PostgreSQL. Позволяет эффективно взаимодействовать с базой данных, предоставляя удобный интерфейс для работы с объектами.

*PostgreSQL:* применяется в качестве реляционной базы данных. Обладает высокой производительностью, надежностью и масштабируемостью, что делает его отличным выбором для системы управления гостиничным комплексом.

*Thymeleaf:* является шаблонизатором для Java, который интегрируется со Spring Framework. Он предоставляет возможность создания динамических HTML страниц, что делает его удобным для разработки пользовательского интерфейса веб-приложения.

*Docker:* используется для удобного развертывания и масштабирования приложения. Он позволяет помещать приложение и его зависимости в контейнеры, что упрощает управление инфраструктурой и обеспечивает повторяемость развертывания.

*Redis:* используется для кэширования данных, что позволяет ускорить доступ к часто запрашиваемым данным.

Выбор этих технологий обусловлен их популярностью, масштабируемостью, надежностью и удобством использования, что делает их идеальным выбором при разработке веб-приложения для управления гостиничным комплексом.

1.6 Выводы по главе

В данной главе проведен всесторонний анализ предметной области, касающейся управления гостиничным комплексом. Были рассмотрены основные особенности гостиничного бизнеса и его специфика, включая важные аспекты, такие как бронирование номеров, управление персоналом и взаимодействие с клиентами.

Проведен обзор существующих систем управления гостиничными комплексами, таких как OtelMS, Logus HMS и Эдельвейс. В ходе анализа были выявлены ключевые недостатки этих систем, включая высокую стоимость лицензий, ограниченные возможности кастомизации, сложность интеграции с внешними сервисами и сложность использования.

На основании выявленных недостатков была сформулирована задача разработки новой системы управления гостиничным комплексом. Основные требования к новой системе включают экономичность, гибкую кастомизацию, широкие возможности интеграции, простоту использования и масштабируемость.

В рамках постановки задачи разработки было определено, что новая система должна основываться на современных технологиях и фреймворках, таких как Java, Spring, Hibernate, PostgreSQL и Docker. Это обеспечит высокую производительность, надежность и безопасность системы, а также упростит процесс ее развертывания и эксплуатации.

Таким образом, выполнение поставленной задачи позволит создать современную и эффективную систему управления гостиничным комплексом, которая будет удовлетворять потребности современного гостиничного бизнеса и обеспечит высокий уровень качества обслуживания клиентов.

2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНЫХ СРЕДСТВ

2.1 Формулирование функциональных требований к веб-приложению

Функциональные требования определяют основные функции и возможности, которые должно предоставлять веб-приложение для управления гостиничным комплексом.

Функциональные требования заключаются в следующем:

* Авторизация в системе.
* Создание профиля нового пользователя.
* Изменение ролей пользователей приложения.
* Управление номерами и категория номеров, включая добавление, редактирование и удаление.
* Отображение свободных или всех номеров.
* Управление профилями клиентов.
* Поиск клиентов по ФИО.
* Бронирование номеров.
* Заселение клиентов в номера.
* Управление персоналом и их профессиями.
* Просмотр статистики загруженности и доходности.

2.2 Проектирование архитектуры системы

2.2.1 Возможности пользователей

Приложение предоставляет разные возможности в зависимости от роли пользователя. Для наглядности представления функциональности используются диаграммы вариантов, которые отображают информацию о взаимодействии между пользователями и системой. Действующими лицами системы являются: администратор и пользователь. Пользователями выступают сотрудники гостиницы. Ниже представлено графическое представление того, что должна делать система.

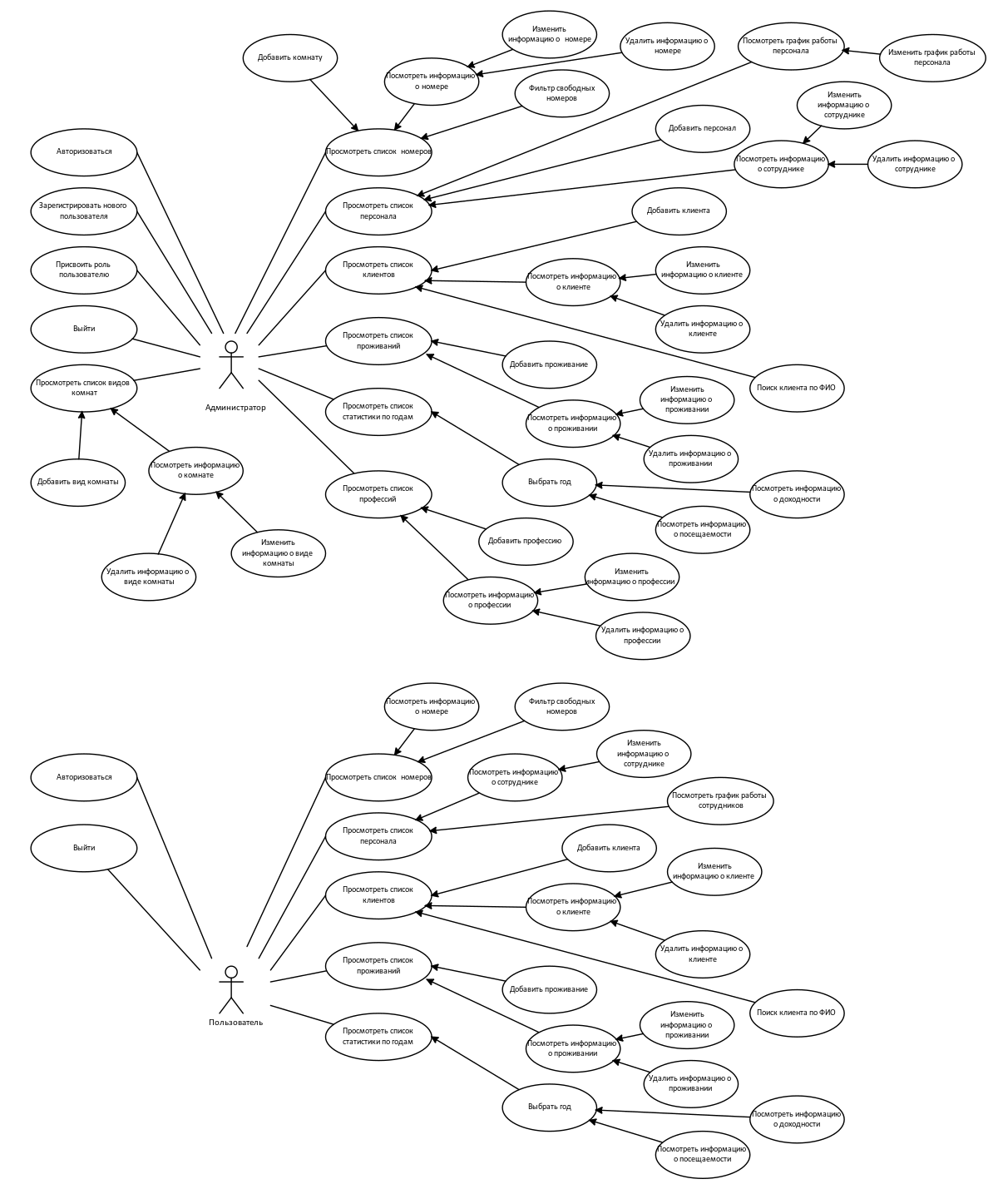


Рисунок 1 – Диаграмма вариантов использования

На диаграмме вариантов использования (рис. 1) простой пользователь после авторизации может просматривать информацию о номерах, персонале, клиентах, проживаниях, статистике, графике работы. Применять фильтр свободных номеров, изменять информацию о сотрудниках, управлять записями о клиентах и проживаниях (изменение, добавление, удаление).

Администратор обладает расширенными правами доступа и возможностями в системе, такими как регистрация новых пользователей и управление их ролями, просмотр и управление всеми записями, к которым есть доступ у простого пользователя, а также к записям о видах номеров и профессиям сотрудников. Эти дополнительные функции администратора обеспечивают контроль и управление всеми аспектами работы приложения и обеспечивают безопасность и эффективность его использования.

2.2.2 Диаграмма классов

Для наглядного представления структуры приложения и взаимосвязей между его компонентами разработаны диаграммы классов (рис. 2-4). В них отображены классы приложения, их поля и методы, а также взаимосвязи между ними.

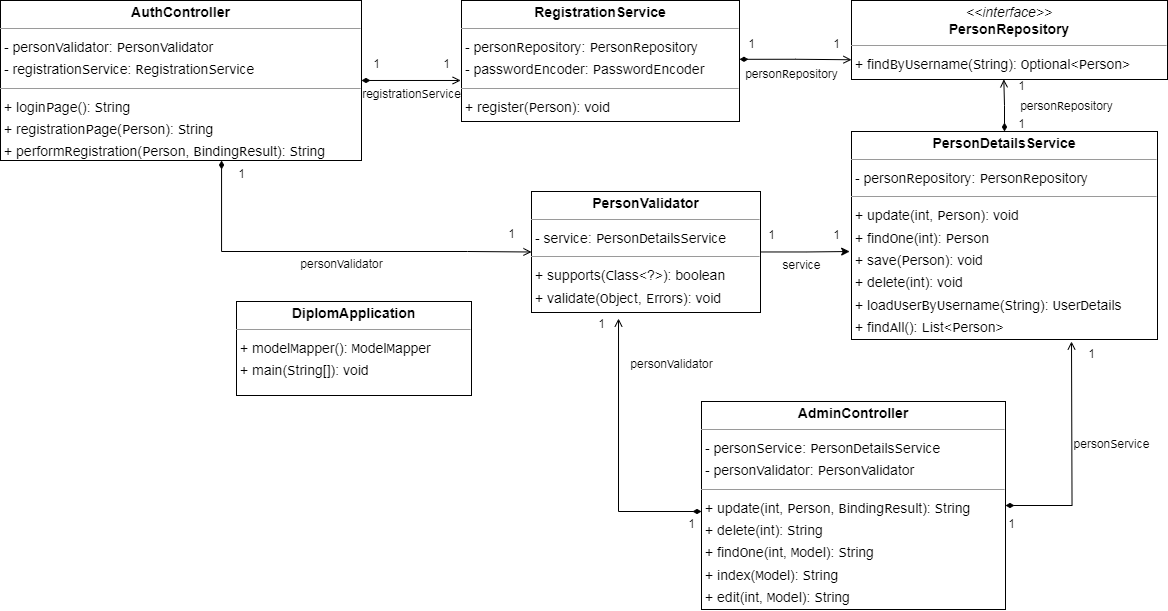
****

Рисунок 2 – Диаграмма классов 1

Диаграмма классов 1 включает в себя следующие классы: *AuthController* – класс-контроллер, отвечающий за аутентификацию пользователей в системе. Он содержит методы для обработки запросов, связанных с аутентификацией. *RegistrationService* – сервисный класс, отвечающий за регистрацию новых пользователей в системе. Он содержит методы для создания новых учетных записей пользователей и их сохранения в базе данных. *PersonValidator* – класс, выполняющий валидацию данных, введенных пользователем при регистрации. Он проверяет корректность введенных данных и выдает сообщение об ошибке в случае несоответствия заданным критериям. *PersonDetailService* – сервисный класс, предоставляющий дополнительные сведения о пользователях системы. Он содержит методы для получения информации о пользователях, такой как их роли, права доступа и другие характеристики. *AdminController* – класс-контроллер, предназначенный для администрирования системы и управления пользователями. Он содержит методы для работы с учетными записями пользователей, их ролями и правами доступа. *PersonRepository* – репозиторий, отвечающий за взаимодействие с базой данных для операций над объектами типа Person (пользователь). Он содержит методы для сохранения, обновления, поиска и удаления объектов Person из базы данных. *DiplomApplication –* основной класс для запуска приложения.

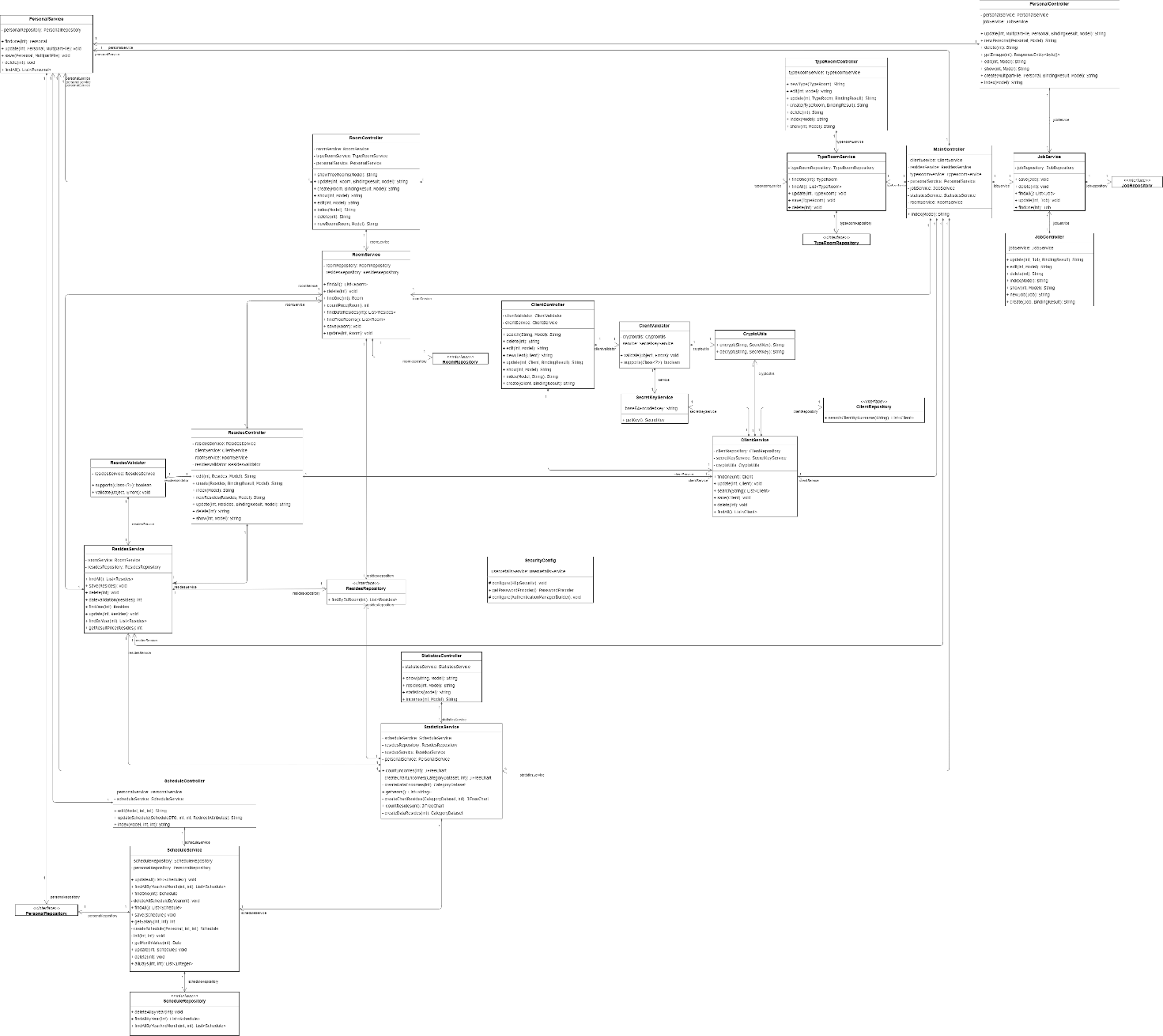


Рисунок 3 – Диаграмма классов 2

Диаграмма классов 2 содержит следующие классы: *PersonalService –* сервисный класс, отвечающий за бизнес-логику, связанную с сотрудниками (персоналом). *PersonalController* – класс-контроллер, обрабатывающий запросы связанные с сотрудниками. *PersonalRepository* – репозиторий, обеспечивающий доступ к данным о сотрудниках в базе данных. *RoomService* – сервисный класс, содержащий бизнес-логику, связанную с управлением номерами. *RoomContoller* – класс-контроллер, обрабатывающий запросы, связанные с номерами. *RoomRepository* – репозиторий, отвечающий за доступ к данным о номерах в базе данных. *TypeRoomService* – сервисный класс, содержащий бизнес-логику, связанную с управлением типами (классами) номеров, *TypeRoomController* – класс-контролер, обрабатывающий запросы, связанные с типами номеров, *TypeRoomRepository* – репозиторий, отвечающий за доступ к данным о типах номеров в базе данных, *ResidesService* – сервисный класс, содержащий бизнес-логику для управления проживаниями, *ResidesController* – класс-контролер, обрабатывающий запросы, связанные с проживанием, *ResidesValidator* – класс, проверяющий корректность данных о проживании, *ResidesRepository* – репозиторий, обеспечивающий доступ к данным о проживании в базе данных, *ClientService*  - сервисный класс, содержащий бизнес-логику, связанную с управлением клиентами, *ClientController* – класс-контролер, обрабатывающий запросы, связанные с клиентами, *ClientRepository* – репозиторий, отвечающий за доступ к данным о клиентах в базе данных, *CryptoUtils* – класс, предоставляющий методы для криптографических операций, *SecretKeyService* - класс, управляющий секретными ключами, используемыми в криптографии, *StatisticService* – сервисный класс, содержащий бизнес-логику для генерации статистических данных, *StatisticController* – класс-контролер, обрабатывающий запросы для получения статистики, *ScheduleService* – сервисный класс, содержащий бизнес-логику для управления графиком работы персонала, *ScheduleController* – класс-контролер, обрабатывающий запросы, связанные с графиком работы персонала, *ScheduleRepository* – репозиторий, обеспечивающий доступ к данным о графике работы персонала в базе данных, *JobService* – сервисный класс, содержащий бизнес-логику для работы с профессиями персонала, *JobController* – класс-контролер, обрабатывающий запросы, связанные с профессиями персонала, *JobRepository* – репозиторий, отвечающий за доступ к данным о профессиях в базе данных, *MainContoller* – класс-контролер, обрабатывающий общие запросы и управляющий навигацией по приложению, *SecurityConfig* – класс, содержащий конфигурацию безопасности приложения.



Рисунок 4 – Диаграмма классов 3

Диаграмма классов 3 содержит следующие классы: *Statistic* – класс, содержащий данные о статистических показателях работы гостиничного комплекса, *TypeRoom* – класс, содержащий информацию о типах (классах) номеров, *Room* – класс, содержащий информацию о номерах, *Resides* – класс, содержащий информацию о проживании гостей, *Job* – класс, содержащий информацию о профессиях персонала, *Personal* – класс, содержащий информацию о персонале, *Client* – класс, содержащий информацию о клиентах, *Person* – класс, содержащий информацию о пользователе приложения, *PersonDetails* – класс, содержащий дополнительную информацию о пользователе приложения, *Schedule* – класс, содержащий информацию о графике работы персонала, *ScheduleDTO* – класс, предоставляющий объект передачи данных для графика работы сотрудников.

2.2.3 Диаграмма компонентов

Диаграмма компонентов предоставляет визуальное представление структуры системы, отображая взаимодействие различных программных компонентов и их взаимосвязи. Эта диаграмма помогает понять, как разные части системы взаимодействуют друг с другом и как данные передаются между компонентами.

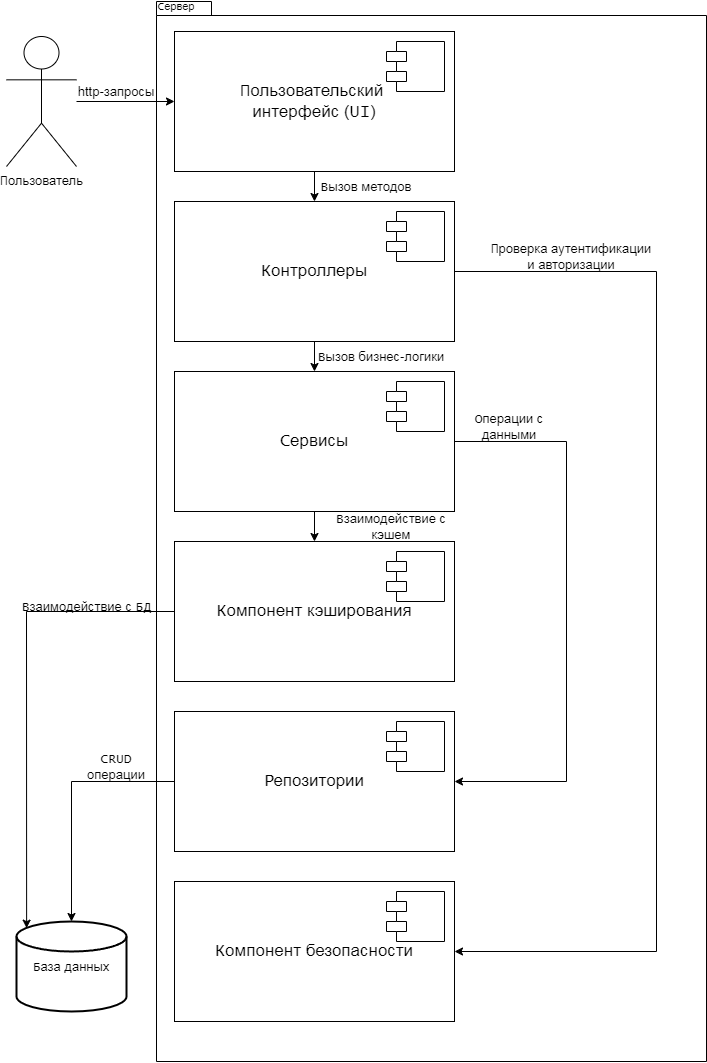


Рисунок 5 – Диаграмма компонентов

*Компоненты системы*

1. Пользовательский интерфейс

Описание: компонент пользовательского интерфейса отвечает за взаимодействие с пользователем. Включает веб-страницы и элементы управления.

Основные технологии: HTML, CSS, JavaScript, Thymeleaf.

1. Контроллеры

Описание: контроллеры обрабатывают входящие HTTP-запросы, взаимодействуют с бизнес-логикой и возвращают соответствующие ответы.

Основные технологии: Spring MVC.

1. Сервисы

Описание: сервисный слой содержит бизнес-логику приложения. Сервисы выполняют операции с данными и вызывают методы репозиториев.

Основные технологии: Spring Framework.

1. Репозитории

Описание: репозитории предоставляют методы для выполнения CRUD-операций с базой данных. Они абстрагируют взаимодействие с базой данных от остальной части приложения.

Основные технологии: Spring Data JPA, Hibernate.

1. База данных

Описание: компонент базы данных хранит все данные приложения. Взаимодействие с базой данных осуществляется через репозитории.

Основные технологии: PostgreSQL.

1. Компоненты безопасности

Описание: обеспечивает аутентификацию и авторизацию пользователей, защищая ресурсы приложения.

Основные технологии: Spring Security.

1. Компоненты кэширования

Описание: используется для временного хранения данных с целью уменьшения нагрузки на базу данных и повышения производительности.

Основные технологии: Redis.

*Взаимодействие компонентов*

1. Пользовательский интерфейс отправляет HTTP-запросы к контроллерам.
2. Контроллеры принимают запросы и вызывают соответствующие методы сервисов для выполнения бизнес-логики.
3. Сервисы взаимодействуют с репозиториями для выполнения операций с базой данных.
4. Репозитории обращаются к базе данных для выполнения CRUD-операций.
5. Компонент безопасности проверяет подлинность и полномочия пользователей при каждом запросе.
6. Компонент кэширования используется сервисами для уменьшения нагрузки на базу данных и ускорения доступа к часто запрашиваемым данным.

На диаграмме компонентов представлены основные компоненты системы и их взаимодействие. Стрелки показывают направление передачи данных и вызовов между компонентами.

Диаграмма компонентов помогает разработчикам и архитекторам системы понять структуру приложения и взаимодействия его частей. Это позволяет эффективно планировать и разрабатывать новые функции, а также оптимизировать существующие.

2.2.4 Диаграмма последовательности

Диаграмма последовательности представляет собой графическое изображение взаимодействий между объектами в определенном порядке. Она демонстрирует, как объекты взаимодействуют друг с другом в рамках выполнения одной или нескольких функций системы. В контексте системы управления гостиничным комплексом диаграмма последовательности может быть полезна для отображения таких сценариев, как процесс бронирования номера или аутентификация пользователя.

*Пример: процесс бронирования номера*

Для создания диаграммы последовательности процесса бронирования номера, где пользователь взаимодействует с системой через веб-интерфейс, необходимы следующие шаги:

1. Пользователь инициирует запрос на бронирования номера через интерфейс.
2. Контроллер принимает запрос и передает его в сервис бронирования.
3. Сервис бронирования выполняет бизнес-логику, проверят доступность номера и сохраняет информацию о бронировании.
4. Сервис бронирования обращается к репозиторию для сохранения данных о бронировании.
5. Репозиторий взаимодействует с базой данных для сохранения информации.
6. Результат операции возвращается через контроллер к интерфейсу пользователя.

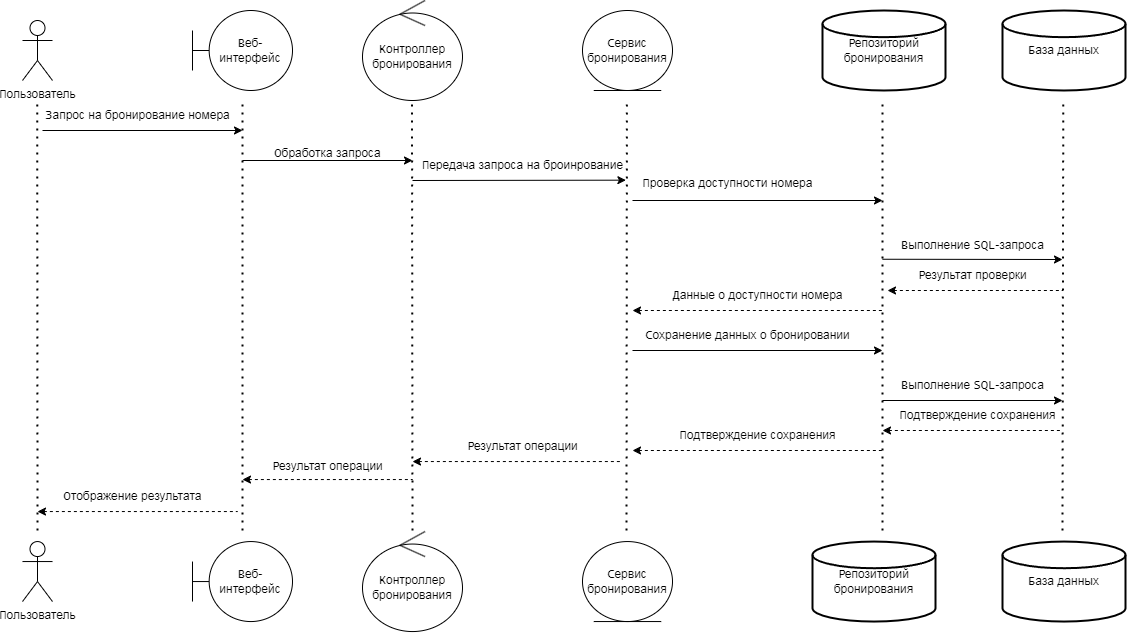


Рисунок 6 – Диаграмма последовательности

Данная диаграмма иллюстрирует процесс бронирования номера от запроса пользователя до сохранения данных в базе данных и отображения результата пользователю.

Описание:

*Пользователь* инициирует запрос на бронирование номера через веб-интерфейс. *Веб-интерфейс* направляет запрос контроллеру бронирования. *Контроллер бронирования* передает запрос сервису бронирования для выполнения бизнес-логики. *Сервис бронирования* проверяет доступность номера, взаимодействуя с репозиторием бронирования. *Репозиторий бронирования* выполняет SQL-запрос к базе данных для проверки доступности номера. *База данных* возвращает результаты проверки репозиторию бронирования. *Репозиторий бронирования* передает данные о доступности номера обратно сервису бронирования. *Сервис бронирования* сохраняет данные о бронировании, передавая их в репозиторий бронирования. *Репозиторий бронирования* выполняет SQL-запрос к базе данных для сохранения данных. *База данных* подтверждает сохранение данных. *Репозиторий бронирования* возвращает подтверждение сохранения сервису бронирования. *Сервис бронирования* передает результат операции контроллеру бронирования.*Контроллер бронирования* возвращает результат веб-интерфейсу.*Веб-интерфейс* отображает результат пользователю.

Диаграмма последовательности помогает понять, как происходит взаимодействие между компонентами системы и как данные проходят через различные уровни приложения в процессе выполнения задачи.

2.2.5 Блок схема алгоритма валидации бронирования дат

Представлена блок-схема алгоритма валидации дат бронирования, реализованного в методе *dateValidation.* Этот алгоритм используется для проверки доступности номера на заданные даты.

*Описание алгоритма*

Метода *dateValidation* принимает объект *Resides,* который содержит информацию о бронировании (идентификатор комнаты, дата въезда, дата выезда и т.п.). Алгоритм проверяет, доступны ли указанные даты для бронирования, возвращая 1, если бронирование возможно, и 0, если даты заняты.

*Код метода*

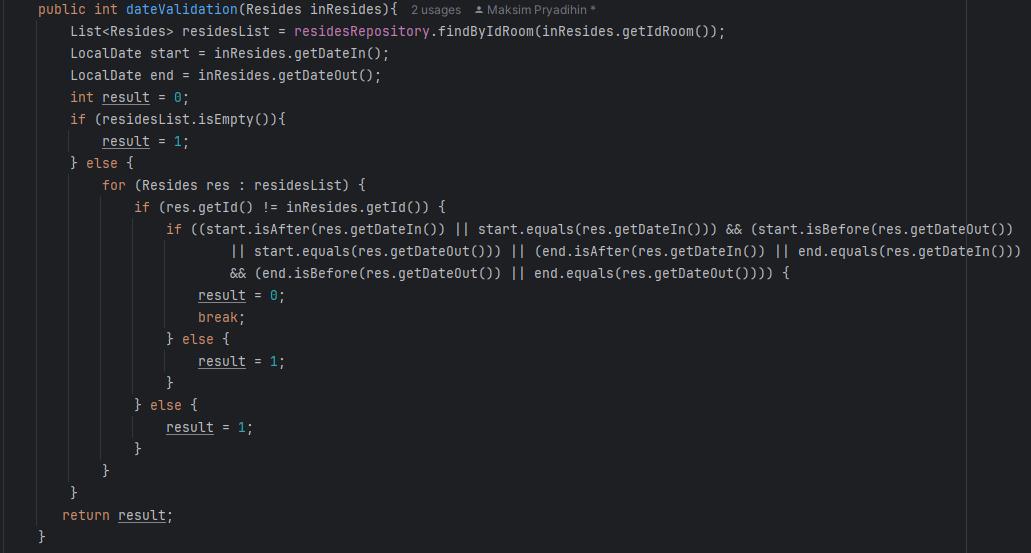


Рисунок 7 – Код метода *dateValidation*

*Блок-схема*

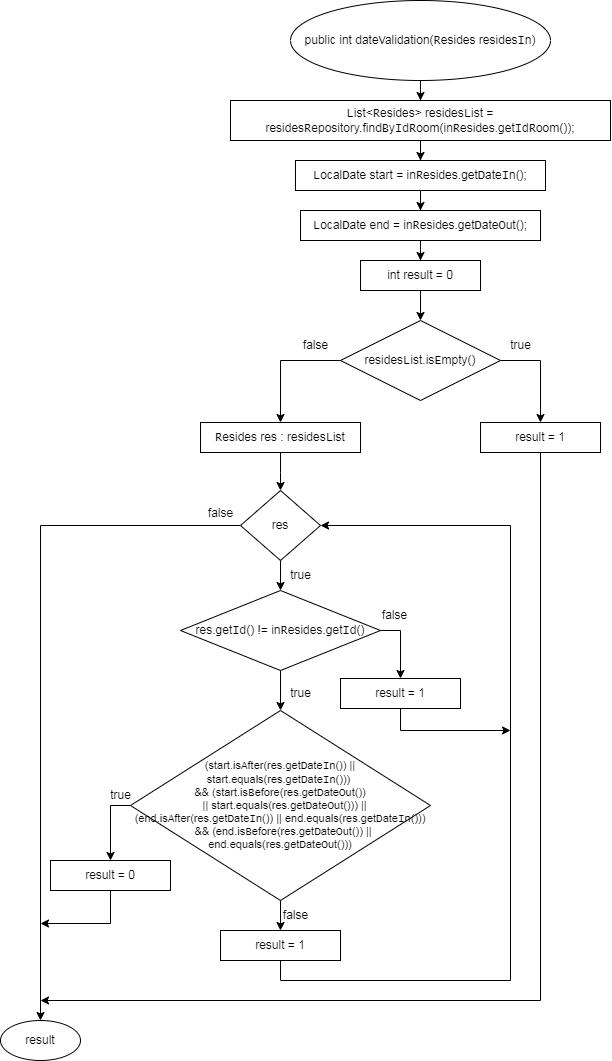


Рисунок 8 – Блок-схема алгоритма *dateValidation*

*Объяснение блок-схемы*

1. Получаем список бронирований для заданной комнаты.
2. Если список бронирования пуст, устанавливаем *result* в 1 (бронирование возможно).
3. Если список бронирования не пуст, получаем даты начала и окончания нового бронирования и устанавливаем *result* в 0.
4. Проходим по списку существующих бронирований

- Если ID текущего бронирования не равен ID нового бронирования, то проверяем, попадают ли даты нового бронирования в диапазон существующего бронирования.

- Если попадают, устанавливаем *result* в 0 и выходим из цикла.

- Если не попадают, устанавливаем *result* в 1.

- Если ID текущего бронирования равен ID нового бронирования, устанавливаем *result* в 1.

5. Возвращаем значение *result.*

Блок-схема помогает наглядно представить процесс валидации дат бронирования и упрощает понимание логики метода.

2.3 Проектирование структуры базы данных

2.3.1 Физическая модель

Физическая модель базы данных определяет структуру данных и способы их хранения на уровне базы данных. В контексте системы управления гостиничным комплексом, физическая модель будет определять таблицы, их поля и связи между ними, а также индексы и ограничения, необходимые для обеспечения целостности данных и эффективного выполнения запросов. Ниже приведена физическая модель базы данных (рис. 9).

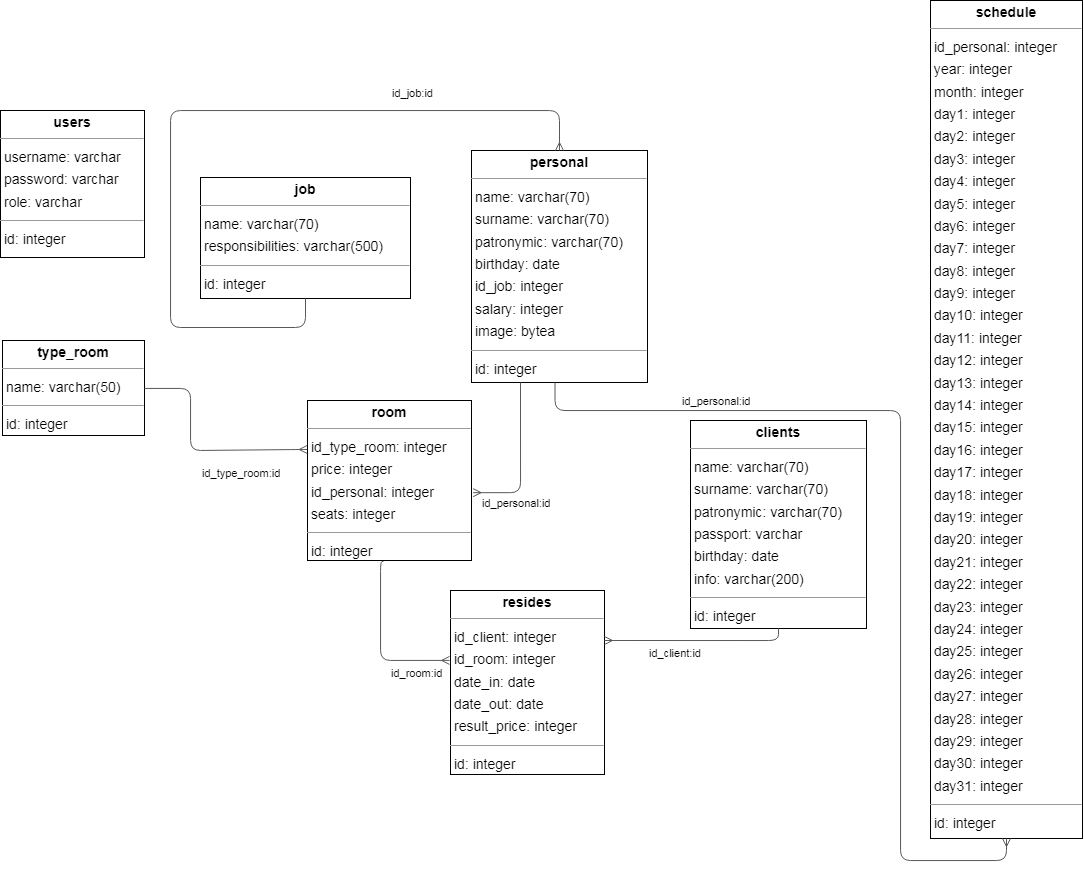


Рисунок 9 – Физическая модель БД

Данная модель состоит из восьми таблиц:

1. users:

* id INTEGER PRIMARY KEY
* username VARCHAR NOT NULL
* password VARCHAR NOT NULL
* role VARCHAR NOT NULL

1. typeRoom

* id INTEGER PRIMARY KEY
* name VARCHAR(50)

1. job

* id INTEGER PRIMARY KEY
* name VARCHAR(70)
* responsibilities VARCHAR(500) DEFAULT ‘ ’ NOT NULL

1. personal

* id INTEGER PRIMARY KEY
* name VARCHAR(70) NOT NULL
* surname VARCHAR(70) NOT NULL
* patronymic VARCHAR(70)
* birthday DATE NOT NULL
* id\_job INTEGER REFERENCES job ON DELETE CASCADE
* salary INTEGER CHECK (salary > 0)
* image BYTEA

1. room

* id INTEGER PRIMARY KEY
* id\_type\_room INTEGER REFERENCES type\_room ON DELETE CASCADE
* price INTEGER NOT NULL CHECK (price > 0)
* id\_personal INTEGER REFERENCES personal ON DELETE CASCADE
* seats INTEGER NOT NULL CHECK (seats > 0)

1. resides

* id INTEGER PRIMARY KEY
* id\_client INTEGER REFERENCES clients ON DELETE CASCADE
* id\_room INTEGER REFERENCES rooms ON DELETE CASCADE
* date\_in DATE NOT NULL
* date\_out DATE NOT NULL
* result\_price INTEGER

1. schedule

* id INTEGER PRIMARY KEY
* id\_personal INTEGER REFERENCES personal ON DELETE CASCADE
* year INTGER NOT NULL
* month INTEGER NOT NULL
* day1 INTEGER DEFAULT 0
* day2 INTEGER DEFAULT 0
* day3 INTEGER DEFAULT 0
* day4 INTEGER DEFAULT 0
* day5 INTEGER DEFAULT 0

…

* day31 INTEGER DEFAULT 0

1. clients

* id INTEGER PRIMARY KEY
* name VARCHAR(70) NOT NULL
* surname VARCHAR(70) NOT NULL
* patronymic VARCHAR(70)
* passport VARCHAR NOT NULL UNIQUE
* birthday DATE NOT NULL
* info VARCHAR(200)

2.4 Проектирование конфигурации и развертывания приложения в Docker-контейнерах

Для обеспечения удобного развертывания и масштабирования приложения будет использована технология контейнеризации Docker. Контейнеры представляют собой средства инкапсуляции приложения вместе с его зависимостями[2, с. 17]. Ниже представлены основные этапы проектирования конфигурации и развертывания приложения в Docker-контейнерах:

1. Определение образов Docker:

* Веб-сервер: создание Docker-образа для веб-приложения, использующего Java и фреймворк Spring. Образ должен содержать все необходимые зависимости, исходные файлы приложения и настройки Spring.
* Сервис кэширования Redis: Создание Docker-образа для Redis, который будет использоваться в качестве сервиса кэширования. Образ должен содержать конфигурационные файлы и настройки Redis.

1. Организация многоконтейнерного приложения: разделение приложения на отдельные сервисы и контейнеры по функциональности.
2. Настройка docker-compose.yml: создание файла docker-compose.yml для определения структуры многоконтейнерного приложения. В docker-compose.yml указываются сервисы, используемые образы, сети, порты и другие параметры конфигурации.
3. Запуск контейнеров: после создания образов и написания docker-compose.yml выполняется сборка образов Docker с помощью команды *docker build,* а затем запуск контейнеров с помощью команды *docker-compose up.*

Этот процесс обеспечивает эффективное и удобное развертывание приложения в Docker-контейнерах, что обеспечивает его надежность, масштабируемость и переносимость между различными окружениями.

2.5 Выводы по главе

В данной главе были определены и проработаны основные архитектурные аспекты системы управления гостиничным комплексом.

Были разработаны:

1. Диаграмма возможностей пользователей, показывающая функции, доступные различным категориям пользователей.
2. Диаграмма классов, отражающая основные сущности системы и их взаимосвязи, что стало основой для проектирования базы данных и логики приложения.
3. Диаграмма компонентов, демонстрирующая логическое разбиение системы на отдельные модули и их взаимодействие, что позволило определить границы ответственности каждого компонента.
4. Диаграмма последовательности, показывающая динамику взаимодействия между объектами системы на примере ключевого сценария, что важно для реализации бизнес-логики.
5. Блок-схема алгоритма для валидации дат бронирования, описывающая логику проверки пересечения дат новых бронирований с уже существующими.
6. Физическая модель базы данных, включая структуру таблиц, связи между ними и основные атрибуты.

Эти результаты обеспечили ясное понимание структуры и логики системы, что будет использовано для реализации и тестирования компонентов, создания базы данных и написание кода приложения. Проектирование архитектуры системы закладывает прочный фундамент для последующей разработки, тестирования и внедрения системы управления гостиничным комплексом.

3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**3.1 Начальная конфигурация проекта**

Данный проект разработан в среде IntelliJ IDEA с использованием системы управления зависимостями Maven. Система управления зависимостями Maven обеспечивает эффективное управление зависимостями проекта и автоматическую сборку приложения. Maven позволяет легко добавлять в проект сторонние библиотеки и управлять версиями зависимостей, что упрощает процесс разработки и поддержки приложения.

Для управления зависимостями Maven используется конфигурационный pom.xml файл (рис. 10). Используемые элементы POM (Project Object Model):

1. Метаданные проекта: информация о проекте, такая как его идентификатор (groupId, artifactId, version), название описание, разработчики и лицензия.
2. Зависимости: указывают на внешние библиотеки, которые нужны для сборки и работы проекта. Maven автоматический загружает эти библиотеки из удаленных репозиториев и включает в проект.
3. Сборка проекта: содержит плагины для сборки проекта.

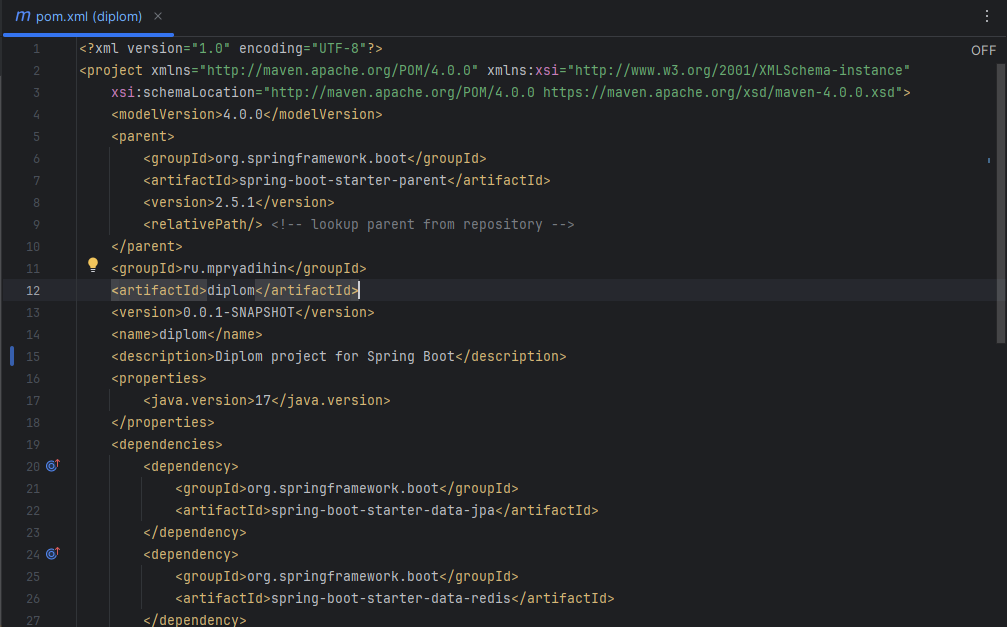


Рисунок 10 – Конфигурация POM файла

В данном проекте используются следующие зависимости:

*spring-boot-starter-data-jpa* – предоставляет стартовый набор зависимостей для использования Spring Data JPA, который облегчает взаимодействие с базой данных через Java Persistence API (JPA).

*spring-boot-starter-data-redis* – предоставляет стартовый набор зависимостей для работы с Redis, распределенным кэшем и хранилищем данных.

*junit-jupiter-api –* предоставляет библиотеку JUnit 5 для написания и запуская модульных тестов в Java.

*lombol ­–* предоставляет аннотации для уменьшения кода в Java, такие как *@Setter, @Getter* и другие, что делает код более читабельным и компактным.

*spring-boot-starter-thymeleaf –* предоставляет стартовый набор зависимостей для использования Thymeleaf, шаблонизатора для создания динамических веб-страниц в приложениях Spring Boot.

*spring-boot-starter-security -* предоставляет стартовый набор зависимостей для использования Spring Security, которая обеспечивает аутентификацию и авторизацию пользователей в приложении.

*spring-boot-starter-validation -* предоставляет стартовый набор зависимостей для использования встроенной валидации в Spring Boot, которая позволяет проводить проверку данных на стороне сервера.

*spring-boot-starter-web -* предоставляет стартовый набор зависимостей для создания веб-приложений с использованием Spring MVC.

*jfreechart –* предоставляет библиотеку для построения графиков и диаграмм в Java.

*postgresql* – предоставляет драйвер JDBC для подключения к базе данных PostgreSQL в приложении.

*modelmapper* – предоставляет инструмент для автоматического сопоставления объектов разных типов в Java.

*sping-security-test –* предоставляет классы и утилиты для тестирования Spring Security в Java-приложениях.

*jakarta.validation-api –* предоставляет API для проведения валидации данных в Java на основе стандарта Bean Validation (JSR 380).

*hibernate-java8 –* предоставляет поддержку работы с Java 8 Date/Time API в Hibernate

А также следующие плагины:

*spring-boot-maven-plugin –* предоставляет возможность собирать и запускать приложение Spring Boot с помощью Maven, автоматический упаковывает приложение в JAR или WAR-файл, в зависимости от настроек проекта.

*maven-dependency-plugin* – предоставляет различные возможности работы с зависимостями проекта, такими как анализ и отображение информации о зависимостях, копирование файлов зависимостей в указанные места, а также управление их версиями и обновлениями. Он часто используется для различных задач с зависимостями в процессе сборки и развертывания приложения.

Использование данных зависимостей и плагинов позволяет создать надежное, эффективное и масштабируемое веб-приложение с использованием современных технологий разработки.

**3.2 Создание архитектурного шаблона MVC и его компонентов**

Данный проект построен на основе архитектурного шаблона MVC (Model-View-Controller). MVC разделяет приложение на три основных компонента (рис.11):

Модель (Model): отвечает за представление данных и бизнес-логику приложения. Включает классы, отражающие структуру данных и логику их обработки, такие как сущности базы данных, сервисы, репозитории и т.д.

Представление (View): отвечает за отображение данных пользователю. Включает в себя шаблоны, HTML-страницы, CSS-стили и JavaScript-код, необходимые для создания пользовательского интерфейса.

Контроллер (Controller): отвечает за обработку запросов от пользователей и взаимодействие с моделью. Включает в себя классы, которые обрабатывают HTTP-запросы, вызывают соответствующие методы модели для выполнения бизнес-логики и выбирают подходящее представление для отображения результатов.



Рисунок 11 – Схема архитектуры MVC

1. Пользователь отправляет запрос контроллеру;
2. Обращение контроллера к репозиторию произойдет при обработке запроса пользователя, полученные данные будут помещены в MVC-модель и далее будут использованы для формирования представления;
3. Контроллер обрабатывает запрос и сохраняет результаты обработки в модель;
4. Контроллер сообщает модели о том, что нужно сформировать новую структуру для дальнейшего отображения;
5. Представление формирует нужную структуру на основе данных из модели;
6. Отображение данных пользователю.

Исходя из выбранного архитектурного шаблона файловая структура имеет следующий вид (рис. 12):

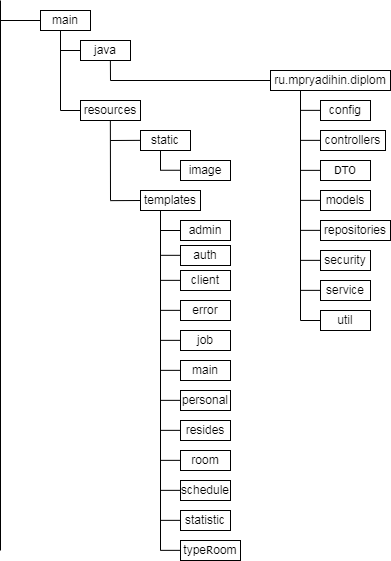


Рисунок 12 – Файловая структура

Каждая папка в структуре проекта выполняет определенную функцию и содержит определенные компоненты приложения:

*config –* содержит файлы конфигурации приложения. Файлы конфигурации определяют параметры и поведение приложения и могу быть использованы для настройки различных аспектов его работы.

*controllers –* содержит классы контроллеров, которые отвечают за обработку HTTP-запросов от клиентов и управление потоком выполнения в приложении. Контроллеры получают запросы от пользователей, вызывают соответствующие методы модели для выполнения бизнес-логики и выбирают представление для отображения результатов.

*DTO* – содержит классы объектов передачи данных, которые используются для передачи данных между различными компонентами приложения.

*models –* содержит классы моделей, которые отображают структуру данных в приложении. Модели представляют собой объекты, содержащие информацию о сущностях в приложении.

*repositories –* содержит интерфейсы репозиториев, которые отвечают за взаимодействие с базой данных. Репозитории предоставляют методы для выполнения операций CRUD (create, read, update, delete) с объектами модели, а также выполняют запросы к базе данных с использованием ORM (Object-Relation Mapping).

*security –* содержит классы, относящиеся к аспектам безопасности приложения.

*service –* содержит классы сервисов, которые содержат бизнес-логику приложения. Сервисы обрабатывают запросы от контроллеров, выполняют операции с данными и взаимодействуют с репозиториями для доступа к данным.

*util –* содержит класс, которые не относятся к бизнес-логике приложения, но предоставляют вспомогательные функции для его работы.

*static –* содержит статические ресурсы, которые используются для оформления и функциональности пользовательского интерфейса.

*templates –* содержит шаблоны представлений (view) в формате HTML или других поддерживаемых шаблонизатором файлов.

Файловая структура проекта соответствует архитектурному шаблону MVC, что обеспечивает четкую организацию и легкость поддержки. Каждая папка в структуре проекта выполняет определенную функцию и содержит соответствующие компоненты приложения. Такая организация обеспечивает чистоту кода и улучшает его читаемость. Архитектурный шаблон MVC в сочетании с правильной файловой структурой делает проект более масштабируемым, гибким и поддерживаемым.

**3.3 Создание основных модулей и компонентов веб-приложения на Java с использованием Spring Framework**

Для реализации основных модулей и компонентов веб-приложения на Java с использованием Spring Framework были созданы следующие элементы:

*Контроллеры* : для каждого функционального блока приложения были созданы соответствующие контроллеры (рис. 13). Каждый контроллер аннотирован аннотацией *@Controller* для обозначения его роли в MVC архитектуре. Также в контроллерах с помощью аннотации *@PreAuthorize* был настроен уровень доступа к компонентам в зависимости от роли. Аннотации *@GetMapping, @PostMapping, @PatchMapping, @DeleteMapping* указывают, что методы обрабатывают HTTP GET, POST, PATCH и DELETE запросы. *@RequestMapping* указывает базовый путь для всех методов контроллера. *@Autowired* инжектирует экземпляры сервисный классов в контроллер, Spring осуществляет это автоматический. *@PathVariable* связывает параметры метода с переменной из URI запроса. *@RequestBody* используется для привязки тела входящего HTTP запроса к параметру метода конструктора. *@ModelAttribute* используется для привязки объекта к модели, чтобы он был доступен в представлении. *@Valid* указывает на то, что объект должен быть проверен перед выполнением метода.

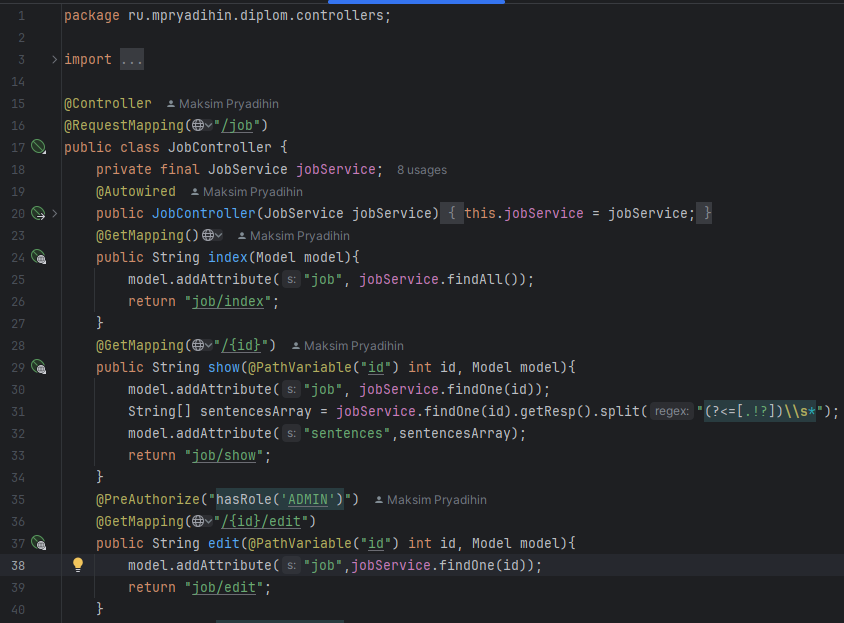


Рисунок 13 – Контроллер профессий сотрудников

*Сервисы* : для обработки бизнес-логики приложения были созданы сервисы (рис. 14). Они содержат методы для выполнения операций над данным. Каждый сервис аннотирован аннотацией *@Service*. Аннотация *@Transactional* указывает, что все методы в классе выполняются в транзакционном контексте. *@Autowired* используется для автоматического внедрения репозиториев. *@Cacheable* указывает Spring на то, что результат выполнения метода должен быть кэширован, чтобы при последующих вызовах метода с теми же аргументами можно было использовать кэшированный результат, что улучшает производительность. *@CachePut* указывает на, что результат выполнения метода должен быть добавлен в кэш или обновлен в кэше, независимо от того, существует ли уже результат для данного ключа. *@CacheEvict* указывает на то, что результат выполнения метода должен быть удален из кэша.

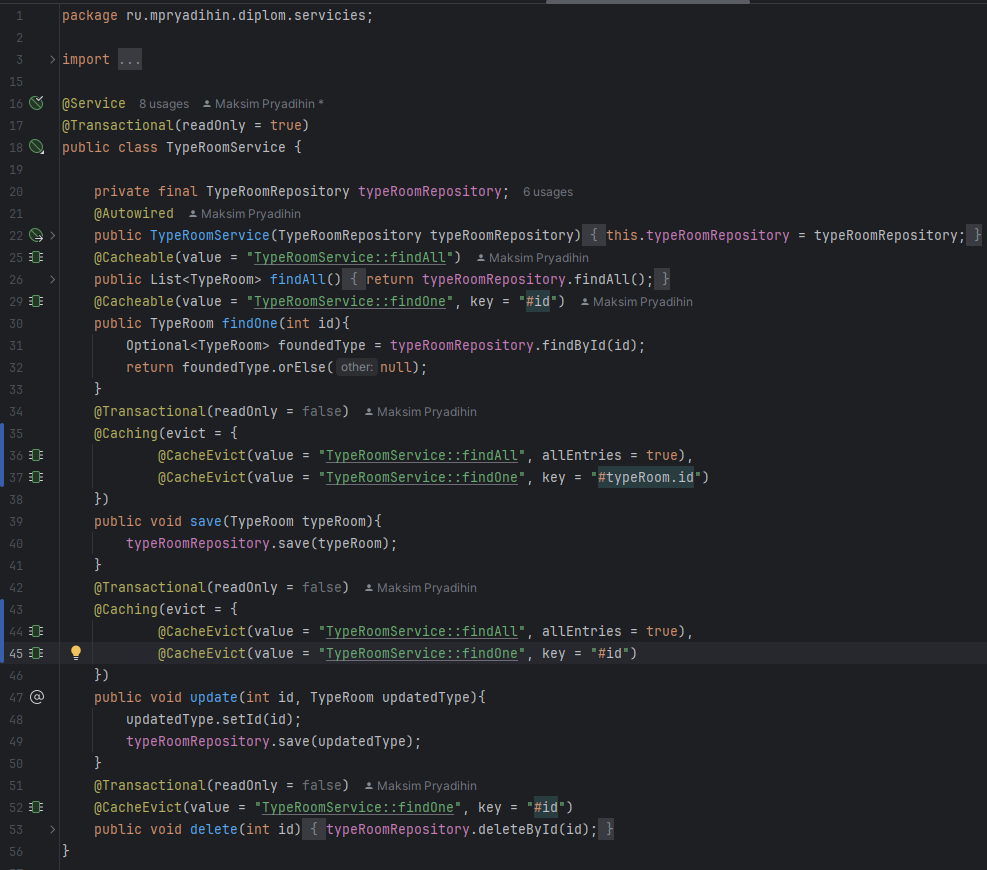


Рисунок 14 – Сервисный класс типов номеров

*Репозитории* : для взаимодействия с базой данных были созданы репозитории, которые предоставляют абстракцию доступа к данным (рис. 15). Репозитории используются для выполнения запросов к базе данных с использованием Spring Data JPA. Каждый репозиторий аннотирован аннотацией *@Repository.* Аннотация *@Query* указывает на то, что этот метод должен выполнять пользовательский запрос к базе данных.

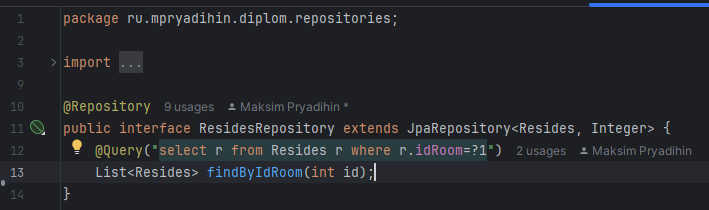


Рисунок 15 – Репозиторий проживаний

*Модели* : были созданы классы моделей, которые отражают структуру данных приложения (рис. 16). Каждый класс модели содержит поля, соответствующие атрибутам сущности в базе данных, а также методы доступа к этим полям. Классы моделей аннотированы аннотацией *@Entity* для использования ORM. Аннотации *@Getter* и *@Setter* являются аннотациями Lombok, которые автоматический генерируют методы getter и setter для полей класса. *@Table* указывает имя таблицы базы данных, с которой связана сущность. *@Column* позволяет настроить атрибуты столбца в таблице базы данных. *@Id* указывает поле, которое является первичным ключом в таблице базы данных. *@GeneratedValue* указывает способ генерации значений для первичного ключа. *@NotEmpty, @NotNull, @Size* и другие являются частью Java Bean Validation API и используются для валидации значений полей. *@MapsId* указывает Hibernate, что поле, к которому она применена, использует идентификатор другой сущности как свой собственный идентификатор. *@ManyToOne* указывает на то, что данное поле представляет собой отношение многие к одному с другой сущностью. *@OneToMany* указывает на отношение один ко многим между текущей сущностью и сущностью, с которой она связана. *@JoinColumn* определяет внешний ключ, который используется для связи сущности с сущностью, на которую она ссылается.

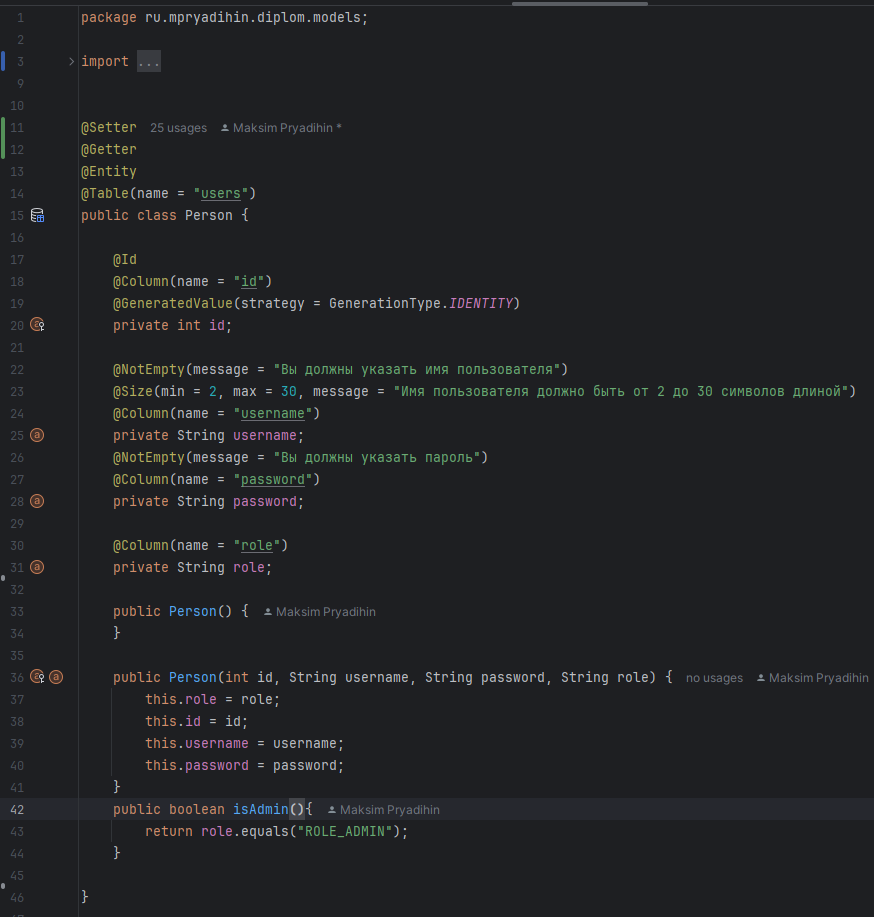


Рисунок 16 – Модель пользователя

*Конфигурационные классы* : был создан конфигурационный класс SecurityConfig, для настройки безопасности веб-приложения с использованием Spring Security (рис. 17). Аннотация *@Configuration* обозначает класс как конфигурационный, что позволяет Spring использовать его для настройки приложения. *@EnabeWebSecurity* включает поддержку безопасности на основе веба в приложении. *@EnableGlobalMethodSecurity* включает глобальную аннотацию безопасности метода, позволяя использовать аннотации *@PreAuthorize* и *@PostAuthorize.* WebSecurityConfigurerrAdapter расширяет данный класс, он предоставляет методы для настройки безопасности веб-приложения. Метод *configure(HttpSecurity http)* используется для настройки правил доступа к различным URL-адресам приложения, в нем определяются правила авторизации, аутентификации и другие аспекты безопасности. Метод *configure(AuthenticationManagerBuilder auth)* используется для настройки механизма аутентификации, в нем определяются, какие пользовательские данные использовать для аутентификации, а также способ их кодирования. Метод *getPasswordEncoder()* создает экземпляр класса *BCryptPasswordEncoder,* который используется для хэширования паролей пользователей перед сохранением их в базе данных. Поле *UserDetailsService* используется для загрузки пользовательских данных в систему аутентификации Spring Security.

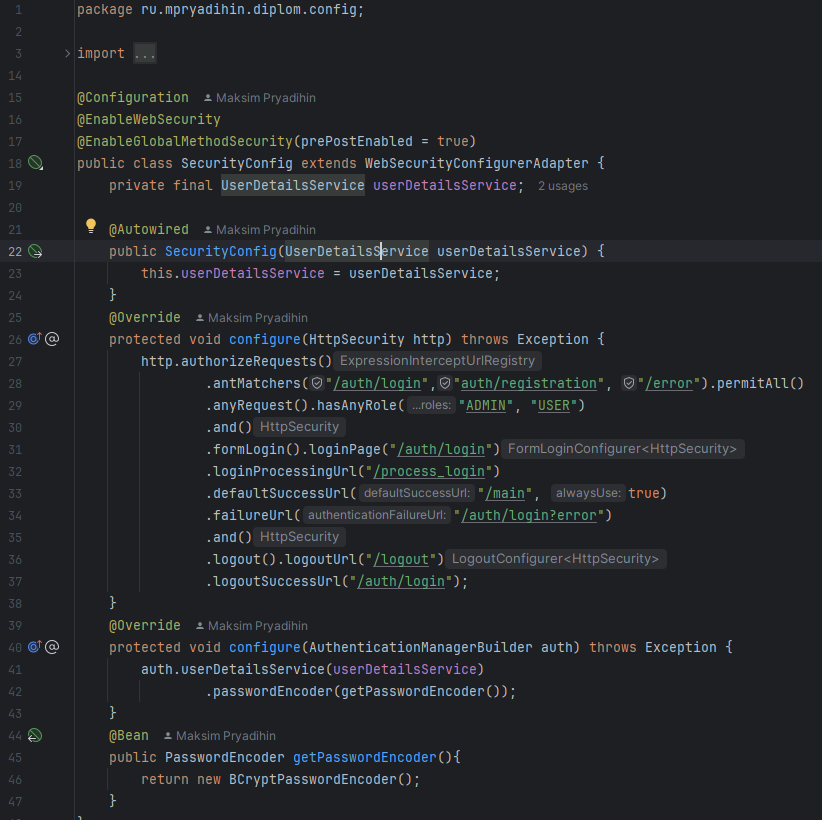


Рисунок 17 – Конфигурационный класс безопасности веб-приложения

*DTO* : для передачи данных между слоями были созданы классы DTO, они содержат только необходимые данные для обмена между клиентом и сервером, что повышает эффективность передачи данных (рис. 18).

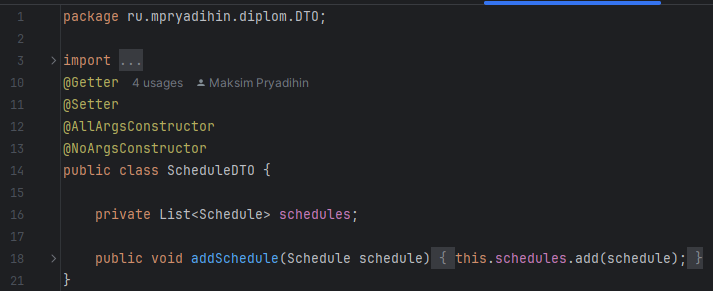


Рисунок 18 – Объект передачи данных для графика работы сотрудников

*Модуль безопасности* : PersonDetails – класс, реализующий интерфейс *UserDatails* (рис. 19)*.* Этот класс используется для предоставления пользовательских данных в контексте аутентификации и авторизации.

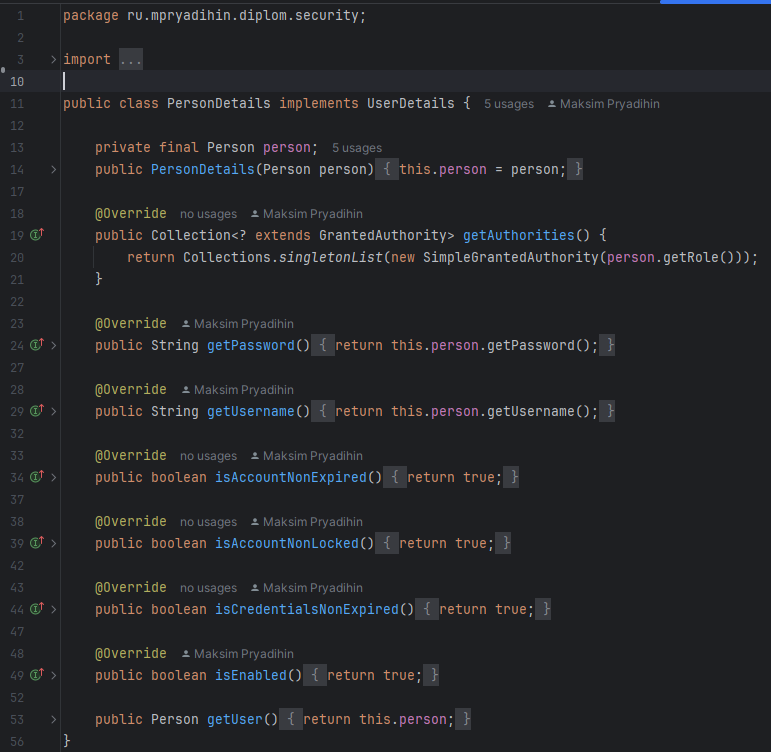


Рисунок 19 – Класс аутентификации и авторизации

*Утилиты* : были созданы дополнительные классы для выполнения различных операций, таких как валидация данных и шифрование пользовательских персональных данных (рис. 20). Дополнительные классы помечаются аннотацией *@Component.*

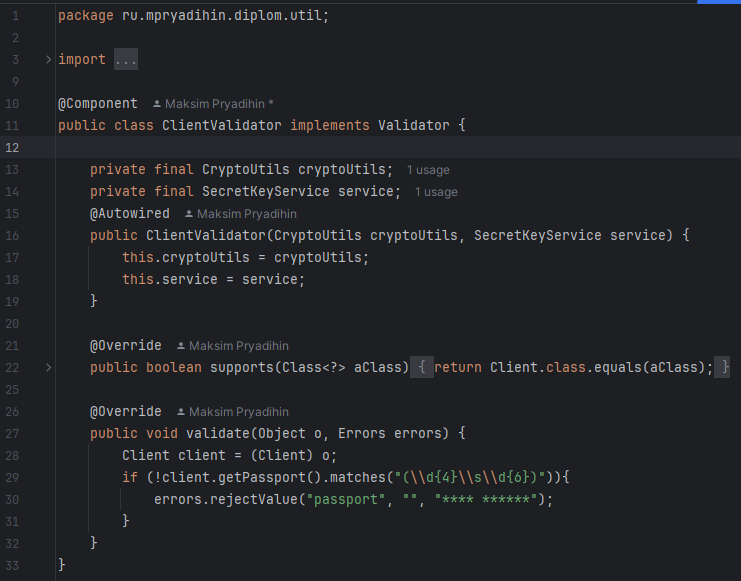


Рисунок 20 – Класс валидации данных клиента

**3.4 Реализация взаимодействия c базой данных PostgreSQL**

В приложении pgAdmin согласно физической модели из пункта 2.3.1 была создана база данных postgreSQL (рис. 21).

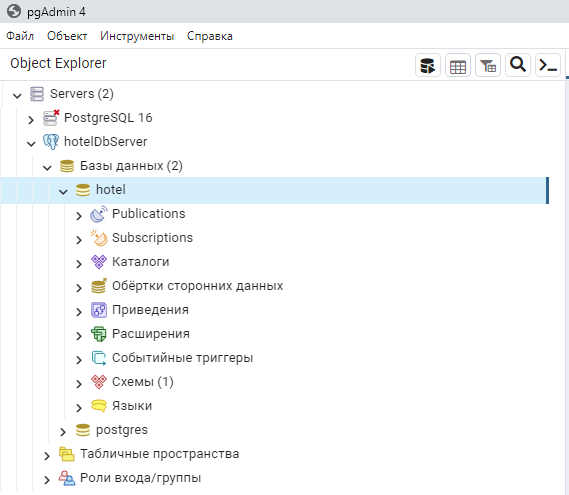


Рисунок 21 – База данных PostgreSQL

Для взаимодействия с базой данных PostgreSQL в проекте были выполнены следующие шаги:

1. Настройка подключения к базе данных: в файле *application.properties* были указаны параметры подключения к базе данных (рис. 22).

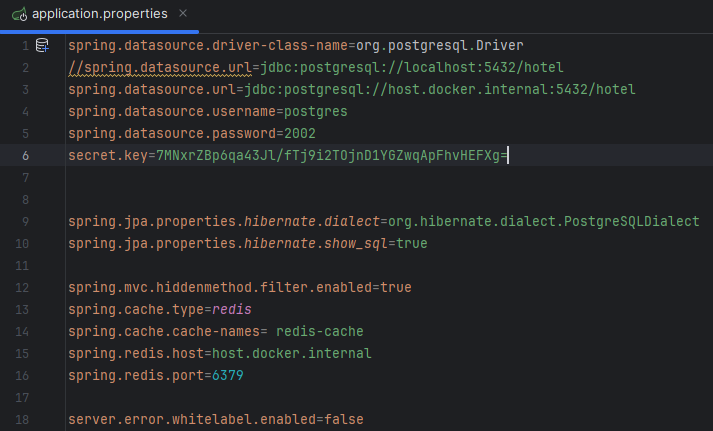


Рисунок 22 – Файл *application.properties*

1. Создание сущностей: были созданы классы-сущности, отражающие структуру таблиц в виде базы данных (см. п. 3.3).
2. Определение репозиториев : для каждой сущности был создан соответствующий репозиторий (см. п. 3.3).
3. Использование JPQL: в некоторых случаях были написаны пользовательские запросы на языке JPQL для выполнения сложных операций с данными, которые не могут быть выполнены с помощью готовых методов репозитория.

В результате выполнения этих шагов была реализована функциональность взаимодействия с базой данных PostgreSQL, что позволяет приложению сохранять, извлекать, обновлять и удалять данные базы данных в соответствии с бизнес-логикой приложения.

**3.5 Создание страниц представлений**

Для создания страниц представлений в веб-приложении были выполнены следующие шаги:

*Использование шаблонизатора Thymeleaf* – необходим для создания страниц представлений. Thymeleaf интегрируется напрямую с Spring Framework и позволяет создавать шаблоны HTML страниц с вставками кода Java или выражений Spring Expression Language.

*Создание HTML-шаблонов –* были созданы HTML-шаблоны страниц представлений, которые содержат статические и динамические элементы, такие как текст, изображения, формы, таблицы и другие. В шаблонах также использовались Thymeleaf-атрибуты для вставки данных из модели или выполнения логики на стороне сервера.

*Использование CSS и JavaScript –* для стилизации и добавления интерактивности страницам представлений были использованы CSS и JavaScript. CSS-стили позволяют задавать внешний вид элементов страницы, а JavaScript обеспечивает интерактивное поведение.

*Интеграция с контроллерами –* в контроллерах были созданы методы, которые обрабатывают HTTP-запросы от клиентов и возвращают соответствующие страницы представлений. Возвращаемые страницы представлений указываются в виде строковых значений, соответствующих названиям HTML-шаблонов.

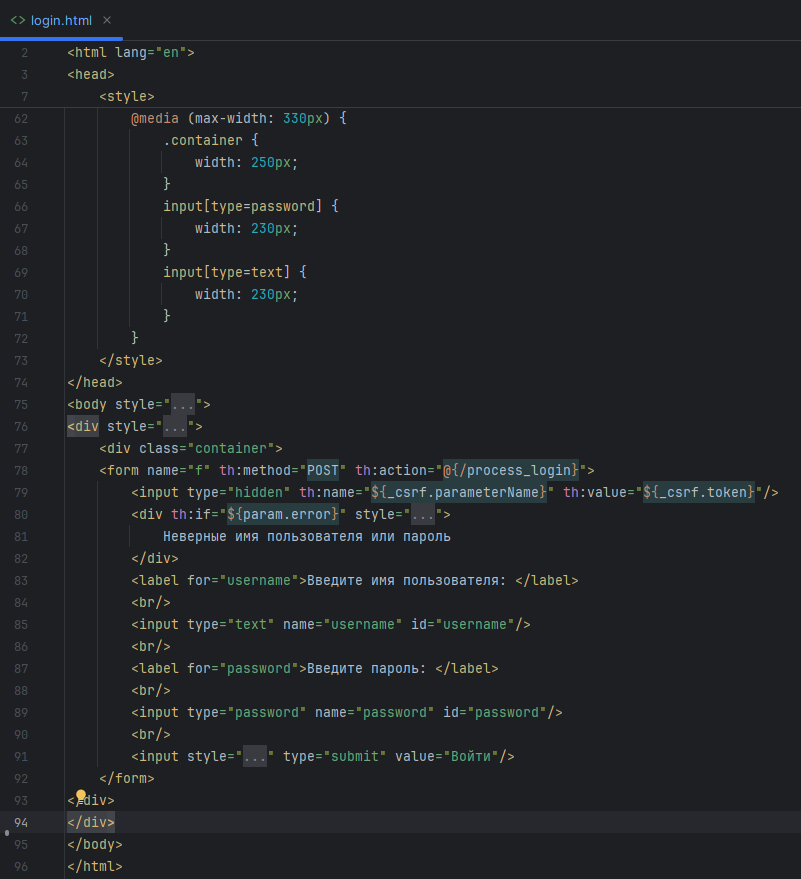


Рисунок 23 – HTML-код страницы аутентификации

Также с помощью аннотации в CSS *@Media* – была реализована кроссплатформенность приложения, что позволяет использовать его на мобильных устройствах.

**3.6 Развертывания приложения в Docker-контейнере**

Образ веб-приложения автоматический создается на основе *Dockerfile.* В файле конфигурации Docker Compose, который используется для определения многоконтейнерных приложений были определены службы, их настройки и зависимости (рис. 24). *Version –* указывает версию синтаксиса Docker Compose. *Volumes –* определяет набор томов данных, которые могут быть использованы в контейнерах. *Services –* содержит описание служб, которые будут запущены в контейнерах.

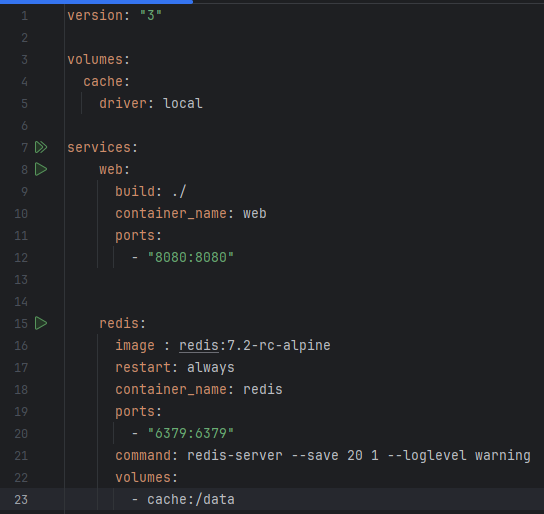


Рисунок 24 – файл Docker Compose

*Web –* определяет службу для веб-приложения. Приложение будет доступно на порту 8080 хоста. *Redis –* определяет службу для сервера Redis. Используя образ *redis:7.2-rc-alpine,* определяет имя контейнера *redis.* Порт 6379 будет проброшен на хост.

На основе данной конфигурации было собрано многоконтейнерное приложение, которое можно запустить на любом хосте с установленным Docker.

3.7 В**ыводы по главе**

В данной главе была проведена работа по реализации системы управления гостиничным комплексом, которая включала следующие ключевые этапы:

*1. Начальная конфигурация проекта:*

- Создание и настройка базового проекта на основе Spring Boot.

- Настройка зависимостей и конфигураций для обеспечения корректной работы приложения.

*2. Создание архитектурного шаблона MVC и его компонентов:*

- Реализация шаблона MVC (Model-View-Controller) для разделения логики приложения на модель, представление и контроллер.

- Настройка контроллеров для обработки запросов и управления потоком выполнения в приложении.

- Создание моделей данных и настройка взаимодействия с базой данных.

*3. Создание основных модулей и компонентов веб-приложения на Java с использованием Spring Framework:*

- Разработка сервисов для реализации бизнес-логики приложения.

- Внедрение зависимостей с использованием аннотаций Spring.

- Настройка компонентов безопасности и аутентификации пользователей с помощью Spring Security.

*4. Реализация взаимодействия с базой данных PostgreSQL:*

- Настройка соединения с базой данных PostgreSQL.

- Создание и настройка репозиториев для выполнения операций CRUD с базой данных.

- Реализация и тестирование запросов к базе данных с использованием Spring Data JPA.

*5. Создание страниц представлений:*

- Разработка пользовательского интерфейса с использованием шаблонов Thymeleaf.

- Настройка страниц для отображения данных, получения ввода от пользователей и взаимодействия с контроллерами.

*6. Развертывание приложения в Docker-контейнере:*

- Создание Dockerfile для сборки образа приложения.

- Настройка Docker Compose для упрощенного развертывания приложения и базы данных.

Эти этапы обеспечили создание функционального, масштабируемого и удобного в использовании веб-приложения для управления гостиничным комплексом. Применение современных технологий и фреймворков, таких как Spring Framework, PostgreSQL и Docker, позволило достичь высокого уровня производительности, безопасности и удобства в эксплуатации системы.

В результате реализации всех указанных этапов была разработана готовая к использованию система, которая удовлетворяет требованиям, сформулированным в первой главе, и обладает возможностями для дальнейшего расширения и улучшения.

4 ТЕСТИРОВАНИЕ И ОЦЕНКА КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1 Планирование и проведение модульного тестирования Java-классов и компонентов Spring-приложения

Модульное тестирование – это методика тестирования программного обеспечения, при которой отдельные компоненты программы тестируются независимо от других компонентов. Модульное тестирование является неотъемлемой частью процесса разработки, которая помогает обеспечить качество кода и функциональность приложения. Ниже представлены подробные этапы этого процесса:

*Выбор тестируемых компонентов –* на этом этапе определяются классы и компоненты, которые требуют модульного тестирования. Необходимо выбрать классы, которые имеют значительное влияние на работу системы или могут привести к критическим ошибкам при неправильной работе.

*Разработка тестовых сценариев –* для каждого выбранного компонента разрабатываются тестовые сценарии. Они должны покрывать основные случаи использования компонента.

*Выбор фреймворка для тестирования ­*– выбирается подходящий фреймворк для модульного тестирования Java-классов и компонентов Spring-приложения.

*Написание тестов* – на этом этапе разрабатываются сами модульные тесты для выбранных компонентов с использованием выбранного фреймворка. Тесты должны быть написаны таким образом, чтобы они выли независимыми, повторяемыми и проверяли только функциональность конкретного компонента.

*Запуск тестов –* после написания тестов они запускаются для всех выбранных компонентов с использованием инструментов сборки, например Maven.

В рамках данного приложения используется модульное тестирование для основных методов классов контроллеров и сервисов. Для написания тестов применяются фреймворки JUnit и Mockito, которые позволяют эффективно создавать и запускать тесты, а также создавать мок-объекты для изоляции тестируемых компонентов от зависимостей. Модульные тесты охватывают основные функциональности контроллеров и сервисов, включая обработку HTTP-запросов, выполнение бизнес-логики, взаимодействие с базой данных и другими внешними сервисами (рис. 25 - 26).

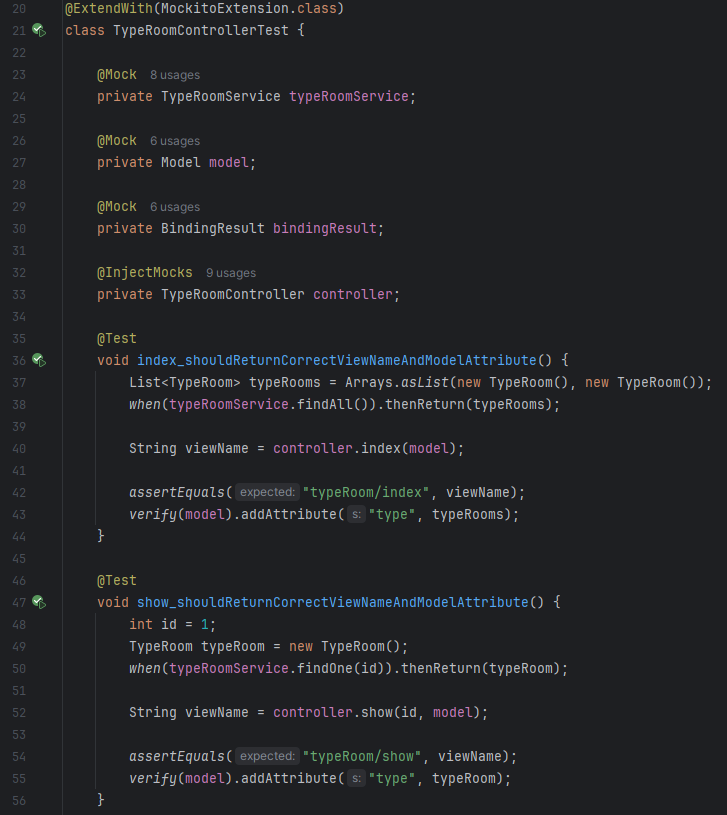


Рисунок 25 – Тестирование контроллера типов номеров



Рисунок 26 – Тестирования сервиса регистрации

В приведенных примерах используются следующие аннотации:

*@ExtendWith(MockitoExtension.class)* – применяется для настройки JUnit для использования расширения MockitoExtensiion, которое автоматически создает и внедряет мок-объекты в тестируемый объект.

*@Mock –* используется для создания мок-объектов для зависимостей, которые должны быть использованы в тестируемом объекте.

*@InjectMocks –* необходима для внедрения созданных мок-объектов в тестируемый объект.

*@BeforeEach ­–* используется в JUnit для обозначения метода, который должен быть выполнен перед каждым тестовым методом в классе тестов.

*@Test –* обозначает метод, как тестовый, которые должен быть выполнен JUnit.

Планирование и проведение модульного тестирования Java-классов и компонентов Spring-приложения помогает обеспечить стабильность и надежность приложения, ускоряет процесс разработки и улучшает качество программного обеспечения.

4.2 Интеграционное тестирование приложения

Интеграционное тестирование является важной частью процесса разработки программного обеспечения, оно позволяется убедиться, что все части приложения работают корректно вместе, а не только по отдельности, как при модульном тестировании. При интеграционном тестировании приложения проверяется взаимодействие между его различными компонентами и модулями в целом. Для интеграционного тестирования обычно используются реальные зависимости и ресурсы, такие как базы данных, внешние API и т.д.

Основные шаги интеграционного тестирования включают в себя:

*Определение сценариев использования* – определяются типичные сценарии использования приложения, которые будут воспроизводиться в тестах.

*Подготовка тестовых данных –* создание и загрузка тестовых данных, необходимых для выполнения интеграционных тестов. Включает в себя создание записей базы данных, подготовку файлов или настройку внешних систем.

*Выполнение тестовых сценариев –* запуск интеграционных тестов.

*Анализ результатов –* анализ результатов тестирования для выявления ошибок, несоответствий и проблем во взаимодействии между компонентами приложения.

*Улучшение и оптимизация –* внесение исправлений и улучшений в код приложения на основе обнаруженных проблем и результатов интеграционного тестирования.

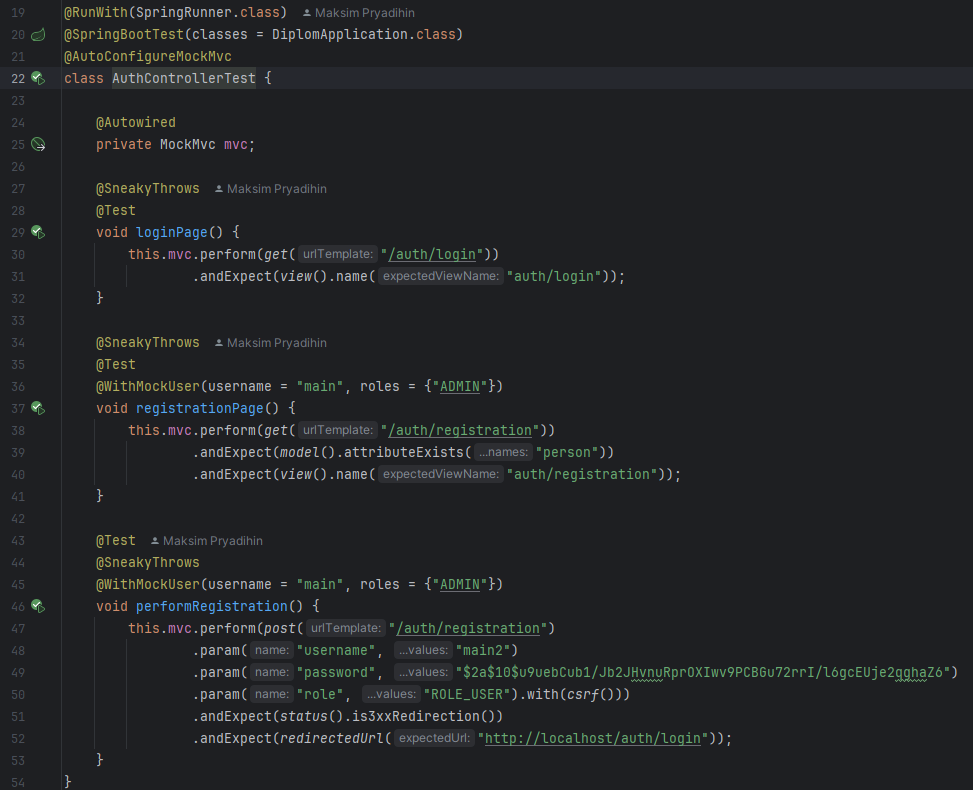


Рисунок 27 – Интеграционное тестирование контроллера аутентификации

В рисунке 27 приведен пример интеграционного тестирования класса-контроллера аутентификации, который обрабатывает запросы аутентификации и регистрации пользователей.

В данном коде применяется следующие аннотации:

*@RunWith(SpringRunner.class) –* указывает на то, что тест будет выполнятся с помощью Spring Runner, что позволяет интегрировать тесты с контекстом Spring.

*@SpringBootTest(classes = DiplomApplication.class) –* запускает тест с использованием контекста приложения, указанного в аргументе *classes*.

*@AutoConfigureMockMvc* – автоматически конфигурирует *MockMvc* для использования в тесте.

*@WithMockUser* – создает фиктивного пользователя для выполнения теста. В данном случае создается пользователь с именем *main* и ролью *ADMIN.*

Поле *private MockMvc mvc* – объект, предоставляющий возможность отправки HTTP-запросов в контроллер и проверки результатов.

Применяются следующие методы тестирования:

*this.mvc.perform() ­*– отправляет HTTP-запрос и проверяет ожидаемые результаты.

*get(“/auth/login”)* и *get(“/auth/registration”)* – формируют GET-запросы к указанным адресам.

*post(“/auth/registration”) ­*– формируют POST-запрос к указанному адресу с передачей параметров.

*param()* – устанавливает параметры запроса.

*with(csrf()) ­–* добавляет CSRF-токен к запросу, если CSRF-защита включена в приложении.

*andExpect()* – проверяет ожидаемый результат выполнения запроса.

Интеграционное тестирование обеспечивает уверенность в том, что приложение работает как ожидается в реальных условиях использования, а также помогает выяснить и устранить потенциальные проблемы в его функциональности и взаимодействии между компонентами.

4.3 Тестирование производительности с использованием Apache JMeter для анализа нагрузки на сервер и оценки производительности приложения

Для оценки производительности приложения был использован Apache JMeter, мощный инструмент для проведения различных видов нагрузочных тестов. Ниже представлены основные шаги и результаты тестирования производительности.

*Настройка тестового сценария:* разрабатывается тестовый сценарий, включающий в себя типичные действия пользователей, такие как авторизация, регистрация, просмотр страниц с информацией и т.д. Тестовый сценарий настраивается с учетом ожидаемого поведения пользователей и характеристик приложения.

*Настройка параметров тестирования:* устанавливаются необходимые параметры тестирования, такие как количество пользователей, частота запросов, продолжительность теста и др. (рис. 28), чтобы смоделировать ожидаемую нагрузку на сервер.

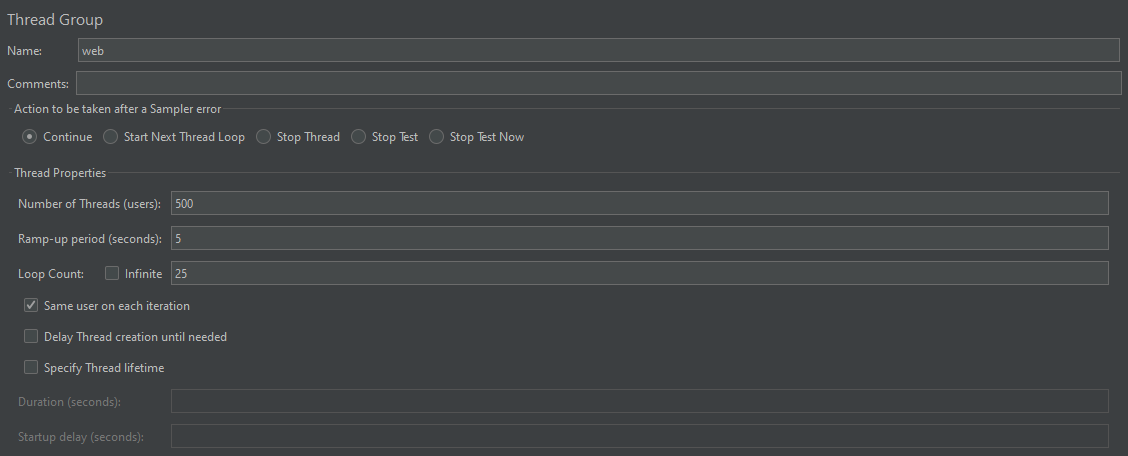
**

Рисунок 28 – Настройка параметров нагрузочного тестирования

*Number of Thread –* указывает количество потоков (пользователей, подключенных к веб-приложению).

*Loop Count –* указывает количество итераций (количество раз выполнения тестирования).

*Ramp-Up Period –* время, в течение которого будут прибавляться пользователи.

*Запуск тестов:* после настройки тестового сценария и параметров, проводится несколько серий тестов для анализа производительности приложения при различных условиях нагрузки.

*Анализ результатов:* полученные результаты анализируются для выявления проблем производительности (рис. 29, 30).

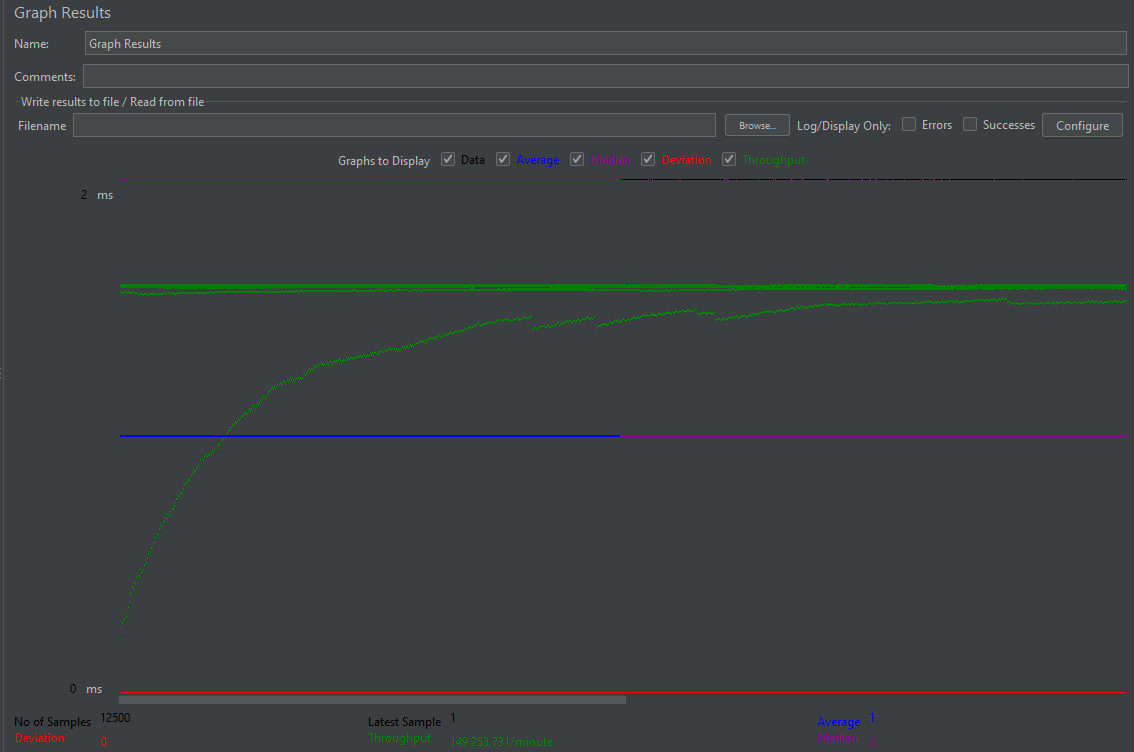


Рисунок 29 – График производительности веб-приложения

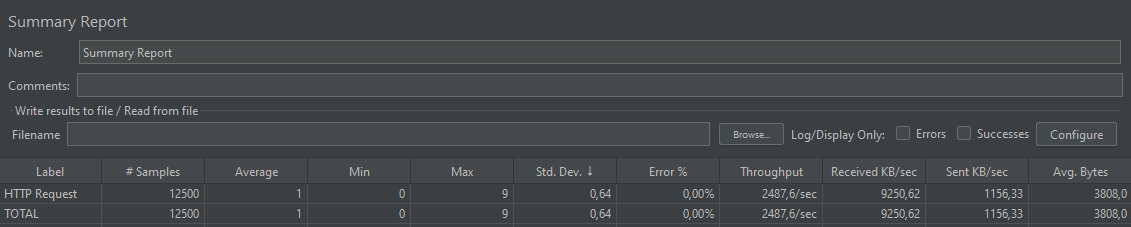


Рисунок 30 – Отчет нагрузочного тестирования

За время выполнения тестирования было выполнено 12500 запросов – все они завершились успешно, без ошибок. Среднее время выполнения одного запроса составило 1 миллисекунду. Пропускная способность 2487,6 запросов в секунду, это указывает на способность системы обрабатывать большой поток запросов.

Тестирование производительности с использованием Apache JMeter позволяет получить данные о работе приложения под нагрузкой, идентифицировать возможные проблемы и улучшить его производительность.

4.4 Тестирование базового пути

Тестирование базового пути – это метод, который обеспечивает проверку выполнения всех возможных уникальных путей через программное обеспечение. Этот вид тестирования гарантирует, что каждая линия кода в программе будет выполнена хотя бы один раз. Для тестирования базового пути были выбраны основные сценарии использования системы управления гостиничным комплексом, чтобы убедиться в правильности их выполнения.

На графе базового пути (рис. 31) обозначены следующие вершины:

1. Начало
2. Авторизация пользователя?
3. Ввод учетных данных
4. Проверка учетных данных
5. Загрузка пользовательского интерфейса
6. Создание новой брони?
7. Ввод данных бронирования
8. Проверка доступности комнаты
9. Сохранение брони в БД
10. Изменение существующей брони?
11. Выбор брони
12. Ввод новых данных
13. Проверка доступности комнаты
14. Сохранение изменений в БД
15. Удаление брони?
16. Выбор брони
17. Удаление брони из БД
18. Просмотр списка доступных комнат?
19. Запрос списка комнат
20. Отображение списка
21. Конец

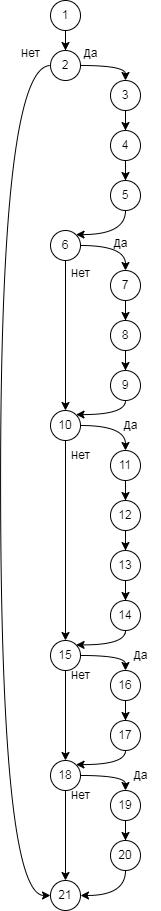


Рисунок 31 – Граф тестируемого алгоритма

Возможные пути:

1. 1 – 2 – 21
2. 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 15 – 18 – 21
3. 1 – 2 – 3 – 4 – 5 – 6 – 10 – 11 – 12 – 13 – 14 – 15 – 18 – 21
4. 1 – 2 – 3 – 4 – 5 – 6 – 10 – 15 – 16 – 17 – 18 – 21
5. 1 – 2 – 3 – 4 – 5 – 6 – 10 – 15 – 18 – 19 – 20 – 21

Тестирование условий:

Путь 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 15 – 18 – 21.

Производится авторизация пользователя (рис. 31).

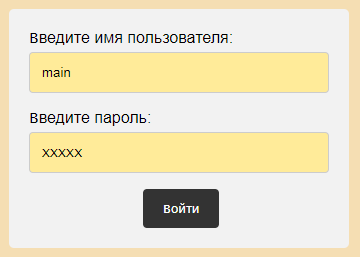


Рисунок 31 – Ввод учетных данных

Ожидаемый результат: успешная валидация данных и загрузка пользовательского интерфейса.

Полученный результат (рис. 32):

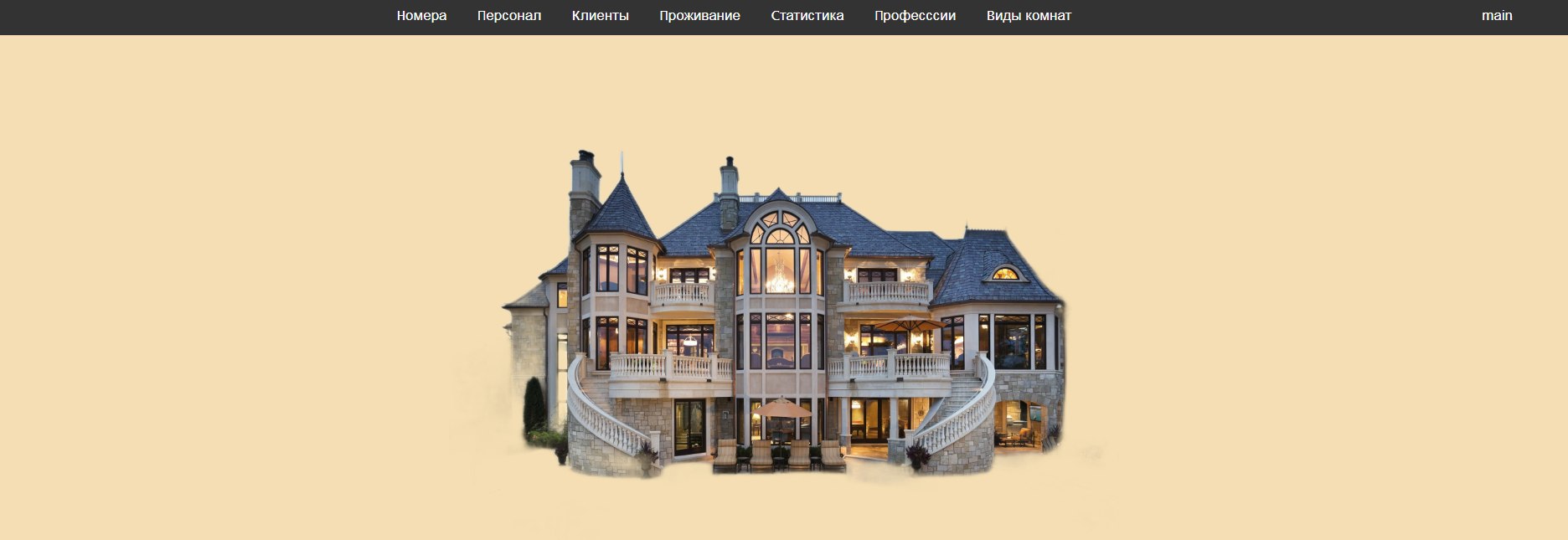


Рисунок 32 – Пользовательский интерфейс

Вводятся данные для создания новой брони (рис. 33).

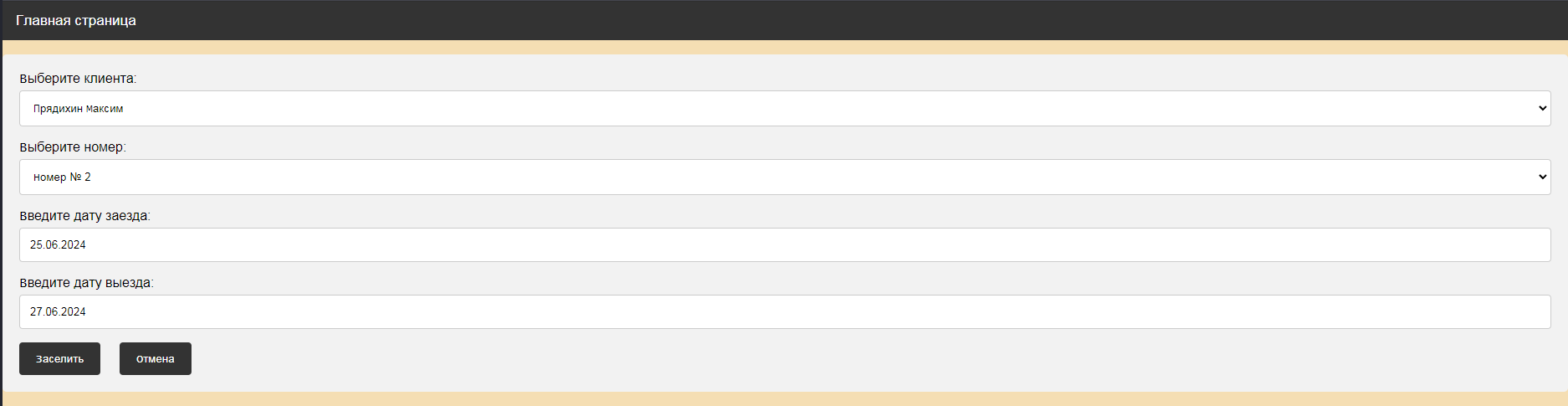


Рисунок 33 – Ввод данных бронирования

Ожидаемый результат: успешное создание новой брони и сохранение записи в базе данных.

Полученный результат (рис. 34 – 35):

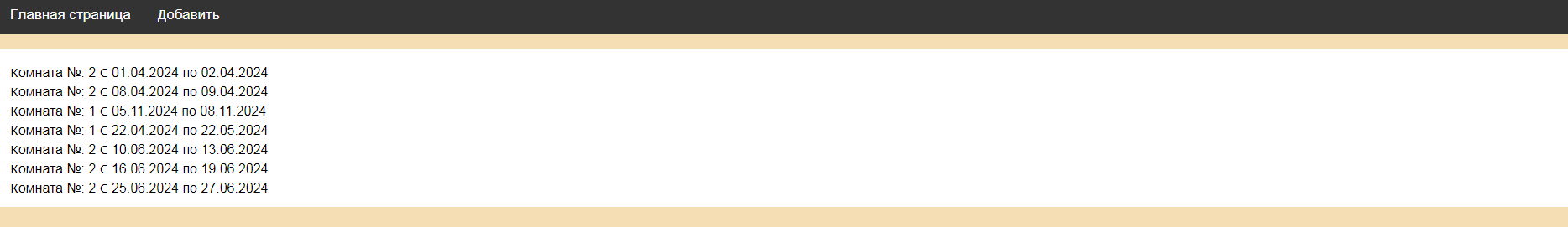


Рисунок 34 – Список бронирований

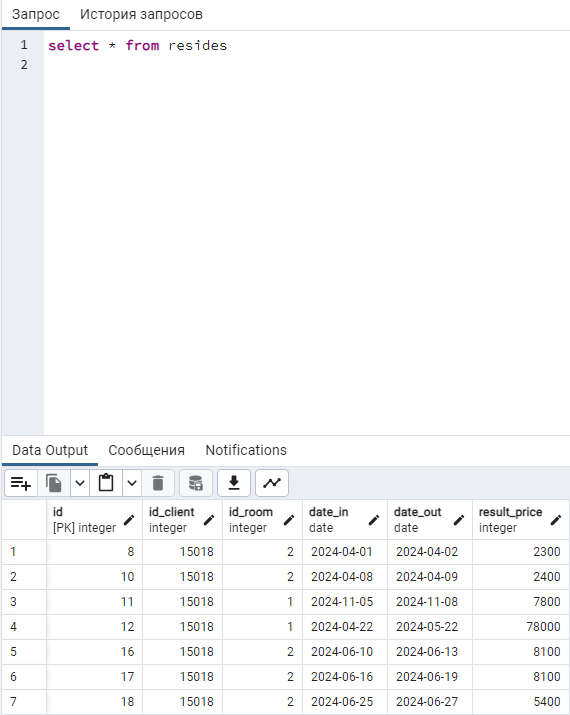


Рисунок 35 – Таблица *resides* в базе данных

Путь 1 – 2 – 3 – 4 – 5 – 6 – 10 – 15 – 18 – 19 – 20 – 21.

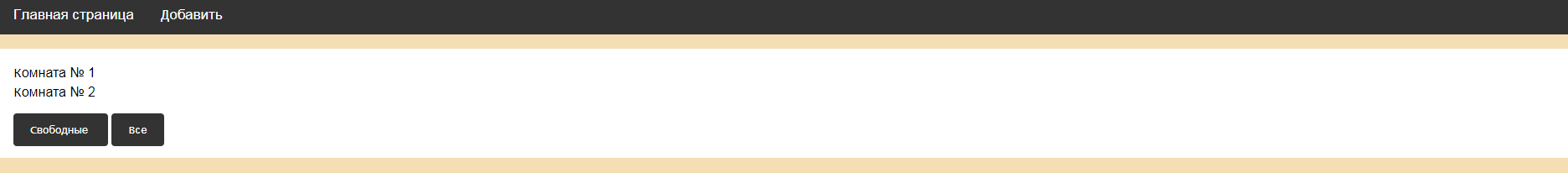


Рисунок 36 – Список номеров

Ожидаемый результат: отображение свободных номеров гостиничного комплекса.

Полученный результат (рис. 37):



Рисунок 37 – Список свободных номеров

На момент тестирования 12.06.2024 – комната №2 забронирована (рис. 38).

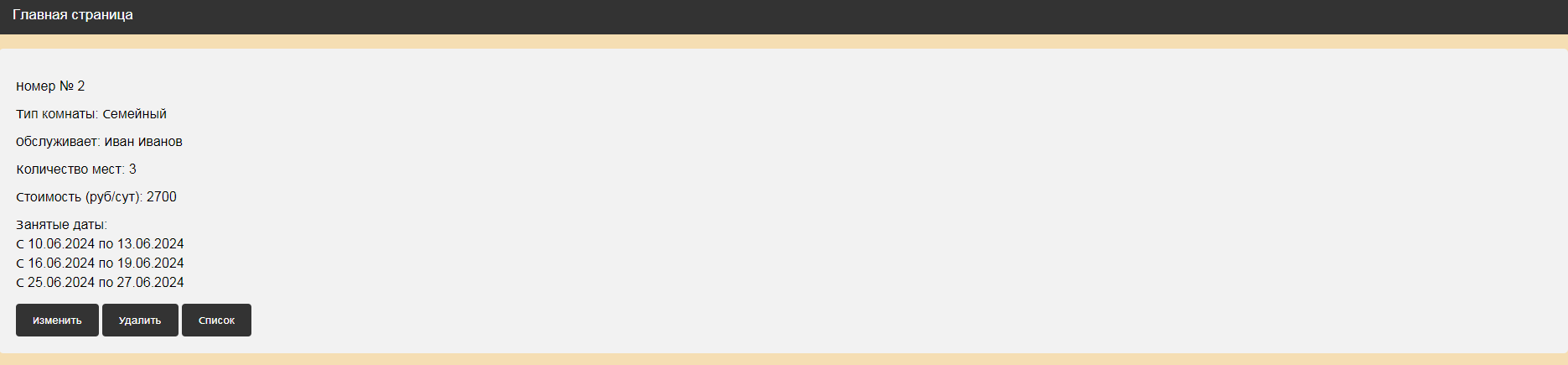


Рисунок 38 – Информация о номере

Тестирование базового пути подтвердило, что основные функции системы работают корректно. Были выявлены и исправлены незначительные ошибки в отдельных модулях, что улучшило стабильность и надежность приложения. Все основные сценарии использования были успешно протестированы, что подтверждает готовность системы к эксплуатации.

4.5 Оценка качества программного продукта

Оценка качества разрабатываемой системы осуществляется в соответствии с ГОСТ 28195-89. Данный стандарт содержит общие положения о методах оценки качества программных средств вычислительной техники (ПС), которые поставляются через фонды алгоритмов и программ (ФАП). Он также определяет номенклатуру и применимость показателей качества ПС.

В соответствии с ГОСТ 28195-89 был выбран подкласс 5012 (программные средства управления базами данных) для оценки качества ПС. Показатели качества этого подкласса приведены в таблице 3.4.1.

Таблица 4.5.1 – Оценочные элементы фактора "надежность ПС"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| Н0101 | Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных | Экспертный | 1 |
| Н0102 | Возможность обработки ошибочных ситуаций | “ | 1 |
| Н0103 | Полнота обработки ошибочных ситуаций | “ | 1 |
| Н0104 | Наличие тестов для проверки допустимых значений входных данных | “ | 1 |
| Н0105 | Наличие системы контроля полноты входных данных | “ | 1 |
| Н0106 | Наличие средств контроля корректности входных данных | “ | 1 |
| Н0107 | Наличие средств контроля непротиворечивости входных данных | “ | 1 |
| Н0201 | Наличие требований к программе по восстановлению процесса выполнения в случае сбоя операционной системы, процессора, внешних устройств | “ | 1 |
| Н0202 | Наличие требований к программе по восстановлению результатов при отказах процессора, ОС | “ | 0 |
| Н0203 | Наличие средств восстановления процесса в случае сбоев оборудования | “ | 1 |
| Н0204 | Наличие возможности разделения по времени выполнения отдельных функция программ | “ | 1 |
| Н0205 | Наличие возможности повторного старта с точки останова | “ | 0 |
| Н0108 | Наличие проверки параметров и адресов по диапазону их значений | “ | 0 |
| Н0109 | Наличие обработки граничных результатов | “ | 1 |
| Н0110 | Наличие обработки неопределенностей (деление на 0, квадратный корень, отрицательного числа и т. д.) | “ | 1 |
| Н0301 | Наличие централизованного управления процессами, конкурирующими из-за ресурсов | “ | 1 |
| Н0302 | Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления | “ | 1 |
| Н0303 | Наличие средств, обеспечивающих выполнение программы в сокращенном объеме в случае ошибок и помех | “ | 1 |
| Н0304 | Показатель устойчивости к искажающим воздействиям | Расчетный | 1 |
| Н0401 | Вероятность безотказной работы | “ | 1 |
| Н0501 | Оценка по среднему времени восстановления | “ | 1 |
| Н0502 | Оценка по продолжительности преобразования входного набора данных | “ | 1 |

Таблица 4.5.2 – Оценочные элементы фактора "сопровождаемость"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| C0803 | Наличие комментариев в точках входа и выхода программы | Экспертный | 1 |
| С0302 | Оценка простоты программы по числу точек входа и выхода | Расчетный | 1 |
| С1002 | Оценка простоты программы по числу переходов по условию | То же | 0,44 |
| С0303 | Осуществляется ли передача результатов работы модуля через вызывающий его модуль | Экспертный | 1 |
| С0304 | Осуществляется ли контроль за правильностью данных, поступающих в вызывающий модуль от вызываемого | " | 1 |
| С0604 | Оценка программы по числу циклов | " | 1 |
| С0801 | Наличие комментариев ко всем машинозависимым частям программы | " | 0 |
| С0802 | Наличие комментариев к машинозависимым операторам программы | " | 0 |
| С0901 | Соответствие комментариев принятым соглашениям | " | 0 |
| С1001 | Используется ли язык высокого уровня | " | 1 |
| С0301 | Наличие проверки корректности передаваемых данных | " | 1 |
| C0902 | Наличие комментариев-заголовков программы с указанием ее структурных и функциональных характеристик | " | 1 |
| C0601 | Использование при построении программ метода структурного программирования | " | 1 |
| C0602 | Соблюдение принципа разработки программы сверху вниз | " | 1 |
| C0201 | Наличие ограничений на размеры модуля | " | 0 |
| C0101 | Наличие модульной схемы программы | " | 0 |
| C030 | Наличие требований к независимости модулей программы от типов и форматов выходных данных | " | 0 |
| C0102 | Оценка программы по числу уникальных модулей | " | 1 |
| C0903 | Оценка ясности и точности описания последовательности функционирования всех элементов программы | " | 1 |
| C0603 | Оценка программы по числу циклов с одним входом и одним выходом | " | 1 |

Таблица 4.5.3 – Оценочные элементы фактора "удобство применения"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| У0101 | Возможность освоения программных средств по документации | Экспертный | 1 |
| У0102 | Возможность освоения ПС на контрольном примере при помощи ЭВМ | То же | 0 |
| У0103 | Возможность поэтапного освоения ПС | " | 1 |
| У0201 | Полнота и понятность документации для освоения | " | 1 |
| У0202 | Точность документации для освоения | " | 1 |
| У0203 | Техническое исполнение документации | " | 1 |
| У0301 | Наличие краткой аннотации | " | 0 |
| У0302 | Наличие описания решаемых задач | " | 1 |
| У0303 | Наличие описания структуры функций ПС | " | 1 |
| У0304 | Наличие описания основных функций ПС | " | 1 |
| У0306 | Наличие описания частных функций | " | 1 |
| У0307 | Наличие описания алгоритмов | " | 1 |
| У0308 | Наличие описания межмодульных интерфейсов | " | 0 |
| У0309 | Наличие описания пользовательских интерфейсов | " | 1 |
| У0310 | Наличие описания входных и выходных данных | " | 0 |
| У0311 | Наличие описания диагностических сообщений | " | 1 |
| У0312 | Наличие описания основных характеристик ПС | " | 0 |
| У0314 | Наличие описания программной среды функционирования ПС | " | 1 |
| У0315 | Достаточность документации ввода ПС в эксплуатацию | " | 1 |
| У0316 | Наличие информации технологии переноса для мобильных программ | " | 0 |
| У0401 | Соответствие оглавления содержанию документации | " | 1 |
| У0402 | Оценка оформления документации | " | 1 |
| У0403 | Грамматическая правильность изложения документации | " | 1 |
| У0404 | Отсутствие противоречий | " | 1 |
| У0405 | Отсутствие неправильных ссылок | " | 1 |
| У0406 | Ясность формулировок и описаний | " | 1 |
| У0407 | Отсутствие неоднозначных формулировок и описаний | " | 1 |
| У0408 | Правильность использования терминов | " | 1 |
| У0409 | Краткость, отсутствие лишней детализации | " | 1 |
| У0410 | Единство формулировок | " | 1 |
| У0411 | Единство обозначений | " | 1 |
| У0412 | Отсутствие ненужных повторений | " | 1 |
| У0413 | Наличие нужных объяснений | " | 1 |
| У0501 | Оценка стиля изложения | " | 1 |
| У0502 | Дидактическая разделенность | " | 1 |
| У0503 | Формальная разделенность | " | 1 |
| У0504 | Ясность логической структуры | " | 0 |
| У0505 | Соблюдение стандартов и правил изложения в документации | " | 1 |
| У0506 | Оценка по числу ссылок вперед в тексте документов | " | 0 |
| У0601 | Наличие оглавления | " | 0 |
| У0602 | Наличие предметного указателя | " | 0 |
| У0603 | Наличие перекрестных ссылок | " | 0 |
| У0604 | Наличие всех требуемых разделов | " | 1 |
| У0605 | Соблюдение непрерывности нумерации страниц документов | " | 1 |
| У0606 | Отсутствие незаконченных разделов абзацев, предложений | " | 1 |
| У0607 | Наличие всех рисунков, чертежей, формул, таблиц | " | 1 |
| У0608 | Наличие всех строк и примечаний | " | 1 |
| У0609 | Логический порядок частей внутри главы | " | 1 |
| У0701 | Наличие полного перечня документации | " | 0 |
| У0801 | Уровень языка общения пользователя с программой | " | 1 |
| У0802 | Легкость и быстрота загрузки и запуска программы | " | 1 |
| У0803 | Легкость и быстрота завершения работы программы | " | 1 |
| У0804 | Возможность распечатки содержимого программы | " | 1 |
| У0805 | Возможность приостановления и повторного запуска работы без потерь информации | " | 1 |
| У0901 | Соответствие меню требованиям пользователя | " | 1 |
| У0902 | Возможность прямого перехода вверх и вниз по многоуровнему меню (пропуск уровней) | " | 0 |
| У1001 | Возможность управления подробностью получаемых выходных данных | " | 0 |
| У1002 | Достаточность полученной информации для продолжения работы | " | 1 |
| У1101 | Обеспечение удобства ввода данных | " | 1 |
| У1102 | Легкость восприятия | " | 1 |
| У1201 | Обеспечение программой выполнения предусмотренных рабочих процедур | " | 1 |
| У1202 | Достаточность информации, выдаваемой программой для составления дополнительных процедур | " | 1 |

Таблица 4.5.4 – Оценочные элементы фактора "эффективность"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| Э0101 | Проблемно-ориентированные функции | Экспертный или расчетный | 0 |
| Э0102 | Машинно-ориентированные функции | То же | 0 |
| Э0103 | Функции ведения и управления | " | 1 |
| Э0104 | Функции ввода/вывода | " | 1 |
| Э0105 | Функции защиты и проверки данных | " | 1 |
| Э0106 | Функции защиты от несанкционированного доступа | " | 0 |
| Э0107 | Функции контроля доступа | " | 0 |
| Э0108 | Функции защиты от внесения изменений | " | 0 |
| Э0109 | Наличие соответствующих границ функциональных областей | " | 1 |
| Э0110 | Число знаков после запятой в результатах вычислений | " | 0 |
| Э0201 | Время выполнения программ | " | 0 |
| Э0202 | Время реакции и ответов | " | 0 |
| Э0203 | Время подготовки | " | 0 |
| Э0205 | Затраты времени на защиту данных | " | 0 |
| Э0206 | Время компиляции | " | 0 |
| Э0301 | Требуемый объем внутренней памяти | " | 18 МБ |
| Э0302 | Требуемый объем внешней памяти | " | 17 МБ |
| Э0303 | Требуемые периферийные устройства | " | Монитор, мышь, клавиатура |
| Э0304 | Требуемое базовое программное обеспечение | " | 0 |

Таблица 4.5.5 – Оценочные элементы фактора "универсальность"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| Г0101 | Оценка числа потенциальных пользователей | Экспертный | 1 |
| Г0102 | Оценка числа функций ПС | То же | 1 |
| Г0103 | Насколько набор функций удовлетворяет требованиям пользователя | " | 1 |
| Г0104 | Насколько возможности программ охватывают область решаемых пользователем задач | " | 1 |
| Г0105 | Возможность настройки формата выходных данных для конкретных пользователей | " | 1 |
| Г0201 | Наличие схемы иерархии модулей программы | " | 1 |
| Г0202 | Оценка независимости модулей | " | 1 |
| Г0203 | Оценка числа уникальных элементов/реквизитов | " | 1 |
| Г0204 | Используется ли в текущем вызове модуля информация, полученная в предыдущем вызове | " | 1 |
| Г0205 | Оценка организации точек входа и выхода модуля | " | 1 |
| Г0206 | Наличие описания атрибутов модуля | " | 0 |
| Г0301 | Оценка программ по числу переходов и точек ветвления | " | 1 |
| Г0401 | Использование метода пошагового уточнения | " | 0 |
| Г0402 | Наличие описания структуры программ | " | 1 |
| Г0403 | Наличие описания связей между элементами структуры программы | " | 0 |
| Г0404 | Наличие в программе повторного выполнения функций (подпрограмм) | " | 1 |
| Г0501 | Использование стандартных протоколов связи | " | 0 |
| Г0601 | Использование стандартных интерфейсных программ | " | 1 |
| Г0701 | Оценка зависимости программ от емкости оперативной памяти ЭВМ | " | 0 |
| Г0702 | Оценка зависимости временных характеристик программы от скорости вычислений ЭВМ | " | 0 |
| Г0703 | Оценка зависимости функционирования программы от числа внешних запоминающих устройств и их общей емкости | " | 0 |
| Г0704 | Оценка зависимости функционирования программы от специальных устройств ввода-вывода | " | 1 |
| Г0801 | Применение специальных языков программирования | " | 1 |
| Г0802 | Оценка зависимости программы от программ операционной системы | " | 1 |
| Г0803 | Зависимость от других программных средств | " | 0 |
| Г0901 | Оценка локализации непереносимой части программы | " | 1 |
| Г1001 | Оценка использования отрицательных или булевых выражений | " | 0 |
| Г1002 | Оценка программы по использованию условных переходов | " | 1 |
| Г1003 | Оценка программы по использованию безусловных переходов | " | 0 |
| Г1004 | Оформление процедур входа выхода из циклов | " | 1 |
| Г1005 | Ограничения на модификацию переменной индексации в цикле | " | 0 |
| Г1007 | Оценка программы по использованию локальных переменных | " | 1 |
| Г1006 | Оценка модулей по направлению потока управления | " | 0 |
| Г1101 | Оценка программы по числу комментариев | " | 0 |
| Г1201 | Наличие заголовка в программе | " | 0 |
| Г1202 | Комментарии к точкам ветвлений | " | 0 |
| Г1203 | Комментарии к машинозависимым частям программы | " | 0 |
| Г1204 | Комментарии к машинозависимым операторам программы | " | 0 |
| Г1205 | Комментарии к операторам объявления переменных | " | 0 |
| Г1206 | Оценка семантики операторов | " | 1 |
| Г1207 | Наличие соглашений по форме представлений комментариев | " | 0 |
| Г1208 | Наличие общих комментариев к программам | " | 0 |
| Г1301 | Использование языков высокого уровня | " | 1 |
| Г1302 | Семантика имен используемых переменных | " | 1 |
| Г1303 | Использование отступов, сдвигов и пропусков при формировании текста | " | 1 |
| Г1304 | Размещение операторов по строкам | " | 1 |
| Г1401 | Передача информации для управления по параметрам | " | 1 |
| Г1402 | Параметрическая передача входных данных | " | 1 |
| Г1403 | Наличие передачи результатов работы между модулями | " | 1 |
| Г1404 | Наличие проверки правильности данных, получаемых модулями от вызываемого модуля | " | 1 |
| Г1405 | Использование общих областей памяти | " | 0 |

Таблица 4.5.6 – Оценочные элементы фактора "корректность"

|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| К0101 | Наличие всех необходимых документов для понимания и использования ПС | Экспертный | 1 |
| К0102 | Наличие описания и схемы иерархии модулей программы | То же | 1 |
| К0103 | Наличие описания основных функций | " | 1 |
| К0104 | Наличие описания частных функций | " | 1 |
| К0105 | Наличие описания данных | " | 1 |
| К0106 | Наличие описания алгоритмов | " | 1 |
| К0107 | Наличие описания интерфейсов между модулями | " | 1 |
| К0108 | Наличие описания интерфейсов с пользователями | " | 1 |
| К0109 | Наличие описания используемых числовых методов | " | 0 |
| К0110 | Указаны ли все численные методы | " | 0 |
| К0111 | Наличие описания всех параметров | " | 0 |
| К0112 | Наличие описания методов настройки системы | " | 0 |
| К0113 | Наличие описания всех диагностических сообщений | " | 1 |
| К0114 | Наличие описания способов проверки работоспособности программы | " | 0 |
| К0201 | Реализация всех исходных модулей | " | 1 |
| К0202 | Реализация всех основных функций | " | 1 |
| К0203 | Реализация всех частных функций | " | 1 |
| К0204 | Реализация всех алгоритмов | " | 1 |
| К0205 | Реализация всех взаимосвязей в системе | " | 0 |
| К0206 | Реализация всех интерфейсов между модулями | " | 1 |
| К0207 | Реализация возможности настройки системы | " | 0 |
| К0208 | Реализация диагностики всех граничных и аварийных ситуаций | " | 1 |
| К0209 | Наличие определения всех данных (переменные, индексы, массивы и проч.) | " | 1 |
| К0210 | Наличие интерфейсов с пользователем | " | 1 |
| К0301 | Отсутствие противоречий в описании частных функций | " | 1 |
| К0302 | Отсутствие противоречий в описании основных функций в разных документах | " | 1 |
| К0303 | Отсутствие противоречий в описании алгоритмов | " | 1 |
| К0304 | Отсутствие противоречий в описании взаимосвязей в системе | " | 1 |
| К0305 | Отсутствие противоречий в описании интерфейсов между модулями | " | 1 |
| К0306 | Отсутствие противоречий в описании интерфейсов с пользователем | " | 1 |
| К0307 | Отсутствие противоречий в описании настройки системы | " | 1 |
| К0309 | Отсутствие противоречий в описании иерархической структуры сообщений | " | 1 |
| К0310 | Отсутствие противоречий в описании диагностических сообщений | " | 1 |
| К0311 | Отсутствие противоречий в описании данных | " | 1 |
| К0401 | Отсутствие противоречий в выполнении основных функций | " | 1 |
| К0402 | Отсутствие противоречий в выполнении частных функций | " | 1 |
| К0403 | Отсутствие противоречий в выполнении алгоритмов | " | 1 |
| К0404 | Правильность взаимосвязей | " | 1 |
| К0405 | Правильность реализации интерфейса между модулями | " | 1 |
| К0406 | Правильность реализации интерфейса с пользователем | " | 1 |
| К0407 | Отсутствие противоречий в настройке системы | " | 1 |
| К0408 | Отсутствие противоречий в диагностике системы | " | 1 |
| К0409 | Отсутствие противоречий в общих переменных | " | 1 |
| К0501 | Единообразие способов вызова модулей | " | 1 |
| К0502 | Единообразие процедур возврата управления из модулей | " | 1 |
| К0503 | Единообразие способов сохранения информации для возврата | " | 1 |
| К0504 | Единообразие способов восстановления информации для возврата | " | 1 |
| К0505 | Единообразие организации списков передаваемых параметров | " | 1 |
| К0601 | Единообразие наименования каждой переменной и константы | " | 1 |
| К0602 | Все ли одинаковые константы встречаются во всех программах по одинаковыми именами | " | 1 |
| К0603 | Единообразие определения внешних данных во всех программах | " | 1 |
| К0604 | Используются ли разные идентификаторы для разных переменных | " | 0 |
| К0605 | Все ли общие переменные объявлены как общие переменные | " | 1 |
| К0606 | Наличие определений одинаковых атрибутов | " | 0 |
| К0701 | Комплектность документации в соответствии со стандартами | " | 1 |
| К0702 | Правильное оформление частей документов | " | 1 |
| К0703 | Правильное оформление титульных и заглавных листов документов | " | 1 |
| К0704 | Наличие в документах всех разделов соответствии со стандартами | " | 1 |
| К0705 | Полнота содержания разделов в соответствии со стандартами | " | 1 |
| К0706 | Деление документов на структурные элементы: разделы, подразделы, пункты, подпункты | " | 1 |
| К0801 | Соответствии организации и вычислительного процесса эксплуатационной документации | " | 1 |
| К0802 | Правильность заданий на выполнение программы, правильность написания управляющие и операторов (отсутствие ошибок) | " | 1 |
| К0803 | Отсутствие ошибок в описании действий пользователя | " | 1 |
| К0804 | Отсутствие ошибок в описании запуска | " | 1 |
| К0805 | Отсутствие ошибок в описании генерации | " | 1 |
| К0806 | Отсутствие ошибок в описании настройки | " | 1 |
| К1001 | Наличие требований к тестированию программ | " | 1 |
| К1002 | Достаточность требований к тестированию программ | " | 1 |
| К1003 | Отношение числа модулей, отработавших в процессе тестирования и отладки к общему числу модулей | Расчетный | 0,3 |
| К1004 | Отношение числа логических блоков, отработавших в процессе тестирования и отладки (), к общему числу логических блоков в программе () | То же | 0,2 |

1. Определение усредненной оценки оценочного элемента по нескольким его значениям проводится по формуле:

, где t- число значений ОЭ (оценочного элемента);

k - порядковый номер метрики;

q - порядковый номер ОЭ.

1. Итоговая оценка k-й метрики j-го ведется по формуле:

, где Q - число ОЭ в k -й метрике.

Таблица 4.5.7 – Итоговые оценки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Н | С | УП | Э | Г | К |
| Итоговая оценка | 0,5 | 0,6 | 0,55 | 0,55 | 0,45 | 0,65 |

«Н» - надежность, «С» - сопровождаемость, «У» - удобство применения, «Э» - эффективность, «Г» - универсальности, «К» - корректность.

После оценки качества разработанного программного продукта в соответствии с ГОСТ 28195-89, можно заключить, что этот продукт соответствует только минимальному набору метрик, которые определяют качество программного средства. Следовательно, в своем классе и уровне, данный программный продукт является низкокачественным.

4.6 Выводы по главе

В данной главе была проведена проверка качества и надежности разработанного программного обеспечения для системы управления гостиничным комплексом. Рассмотрены основные виды тестирования, применяемые для обеспечения высокого уровня стабильности и функциональности приложения.

В разделе 4.1 описаны процессы планирования и проведения модульного тестирования Java-классов и компонентов Spring-приложения. Применение фреймворков JUnit и Mockito позволило изолировать и проверить ключевые методы, обеспечив надежность отдельных модулей.

В разделе 4.2 проведено интеграционное тестирование, которое подтвердило корректность взаимодействия различных компонентов системы. Это позволило убедиться в совместимости и правильности функционирования всех модулей в единой среде.

В разделе 4.3 осуществлено тестирование производительности с использованием Apache JMeter. Проведенные тесты нагрузки на сервер и оценка производительности приложения показали его способность эффективно обрабатывать запросы пользователей и справляться с высокими нагрузками.

В разделе 4.4 рассмотрено тестирование базового пути, что позволило удостовериться в правильности выполнения основных сценариев использования системы, таких как регистрация, авторизация, бронирование и управление бронированиями. Тестирование базового пути подтвердило, что приложение корректно обрабатывает все ключевые процессы и сценарии взаимодействия пользователя с системой.

В разделе 4.5 осуществлялась оценка качества системы в соответствии с ГОСТ 28195-89, что обеспечивает соблюдение стандартов качества и надежности программного обеспечения.

Проведенные тесты выявили ряд потенциальных улучшений и доработок, которые были оперативно внесены в приложение. Итогом тестирования стало подтверждение высокого уровня готовности системы к эксплуатации. Тестирование показало, что разработанное приложение соответствует заданным требованиям, обладает необходимой функциональностью, устойчиво к нагрузкам и обеспечивает корректное выполнение всех бизнес-процессов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной дипломной работы было разработано веб-приложение "Система управления гостиничным комплексом", основанное на Java с использованием Spring Framework, Redis, Hibernate и PostgreSQL, развернутая в Docker-контейнерах. Проект был спроектирован с учетом требований к функциональности и безопасности, что позволило создать эффективное и надежное решение для управления гостиничным бизнесом.

В рамках анализа предметной области были рассмотрены существующие решения, такие как OtelMS, Logus HMS и Эдельвейс, выявлены их достоинства и недостатки. Было принято решение о необходимости создания собственного продукта, обусловленное отсутствием полного удовлетворения требований и потребностей компании.

Проектирование приложения включало в себя определение функциональных требований, разработку архитектуры системы, создание базы данных и интерфейса пользователя. Реализация приложения включала разработку основных модулей и компонентов, взаимодействие с базой данных, создание страниц представлений с использованием Thymeleaf и обеспечение безопасности приложения с помощью Spring Security.

Проведение тестирования включало модульное тестирование Java-классов и компонентов Spring-приложения с использованием JUnit и Mockito, интеграционное тестирование приложения с использованием MockMvc, а также тестирование производительности с использованием Apache JMeter. Полученные результаты тестирования были использованы для обнаружения и устранения дефектов, а также для оптимизации производительности приложения.

Разработанное веб-приложение "Система управления гостиничным комплексом" является полнофункциональным решением, которое позволяет эффективно управлять гостиничным бизнесом, обеспечивая удобство использования для пользователей и высокий уровень безопасности и производительности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Алгоритмы для начинающих: теория и практика для разработчика / Панос Луридас – Москва: Эксмо, 2020 – 608 с. – (Мировой компьютерный бестселлер).
2. Архитектура корпоративных приложений / М. Фаулер – Москва: Эксмо, 2020. – 560с.
3. Использование Docker / Моуэт Э. / пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.: ил.
4. Основы Spring Boot / К. Браун – Москва: ДМК Пресс, 2021. – 512с.
5. Продвинутая Java: Концепции и подходы / Дж. Р. Лауэр – Москва: ДМК Пресс, 2020. – 480с.
6. Проектирование баз данных / П. В. Макаревич – Москва: ДМК Пресс, 2017. – 352с.
7. Разработка на Java EE / Р. Сиерра, К. Бейтс – СПб: Питер, 2018. – 900с.
8. Разработка на Spring и Hibernate / П. Спарроу – СПб: Питер, 2020. – 760с.
9. Системы управления базами данных / Дж. Л. Хэнин – Москва: Эксмо, 2019. – 600с.
10. Современные подходы к программированию на Java / М. Джонс – Москва: Эксмо, 2021. – 320с.
11. Философия JAVA. 4-е полное изд. / Б. Эккель – СПБ.: Питер, 2021. – 1168с.: ил. – (Серия «Классика computer science»).
12. HTML и CSS. Разработка и дизайн веб-сайтов / Джон Дакетт – Москва: Эксмо, 2019. – 480с. – (Мировой компьютерный бестселлер).
13. Hibernate Tips: Более 70 решений для обычных проблем Hibernate / Торбен Янсен – Москва: ДМК Пресс, 2019. – 304с.
14. Java для профессионалов / К. Хорстманн – СПб: Питер, 2019. – 1248с.
15. Java: Путеводитель начинающего / Херберт Шилдт – Москва: ДМК Пресс, 2021. – 864с.
16. Java: Эффективное программирование / Джошуа Блох – СПб: Питер, 2019. – 416с.
17. Java Concurrency in Practice / Брайан Гец – Москва: ДМК Пресс, 2020. – 432с.
18. Java Persistence with Hibernate / Кристиан Бауэр, Гэвин Кинг – СПб: БХВ-Петербург, 2018. – 960с.
19. PostgreSQL: Настройка и запуск / Реджина О. Об – Москва: ДМК Пресс, 2018. – 304с.
20. Pro Spring 5: Глубокое руководство по Spring Framework и его инструментам / Иулиана Космина – СПб: БХВ-Петербург, 2018. – 848с.
21. Redis в действии / Джозайя Л. Карлсон – Москва: ДМК Пресс, 2019. – 368с.
22. Spring в действии / Крэйг Уоллс – Москва: ДМК Пресс, 2019. – 592с.
23. Spring Boot в действии / Крэйг Уоллс – Москва: ДМК Пресс, 2020. – 496с.
24. Spring Data: Рецепты программирования / Майкл П. Редлих – Москва: ДМК Пресс, 2019. – 440с.
25. Spring Security в действии / Лоранциу Спилка – Москва: ДМК Пресс, 2020. – 640с.

ПРИЛОЖЕНИЕ А (ТЕХНИЧЕСКОЕ ЗАДАНИЕ)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Филиал федерального государственного бюджетного образовательного учреждения высшего образования

«Национальный исследовательский университет «МЭИ» в г. Смоленске

Кафедра вычислительной техники

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

НА СОИСКАНИЕ КВАЛИФИКАЦИИ

**Бакалавр**

Тема: Система управления гостиничным комплексом

Направление подготовки

**09.03.01 – Информатика и вычислительная техника**

Профиль подготовки

**Программное обеспечение средств вычислительной  
техники**

Студент группы ПО-20 М.Ю. Прядихин

Руководитель

к.т.н., доцент Я.А. Федулов

УТВЕРЖДАЮ

Заведующий кафедрой

д.т.н., профессор А.С. Федулов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_2024 г.

Место выполнения выпускной квалификационной работы: филиал НИУ «МЭИ» в г. Смоленске, кафедра вычислительной техники

Смоленск – 2024

1. Обоснование выбора темы выпускной квалификационной работы

Система управления гостиничным комплексом – это программное обеспечение, которое предназначено для автоматизации и оптимизации процессов управления гостиницей. Она нацелена на облегчение работы персонала гостиницы и повышение уровня обслуживания гостей.

Область применения данной системы охватывает все аспекты управления гостиничным комплексом: от бронирования номеров и учета гостей до управления ресурсами, финансами и персоналом.

1. Основание для разработки

Государственный общеобразовательный стандарт высшего образования. Направление 09.03.01 – «Информатика и вычислительная техника».

1. Назначение и цель разработки

Целью данной работы является разработка и реализация программного решения «Система управления гостиничным комплексом», предназначенного для автоматизации процессов управления гостиничным бизнесом.

Для успешной реализации проекта необходимо изучить общие методы, разработать алгоритмы и архитектуру программы, провести тестирование программы и составить документацию.

1. Технические требования

4.1 Требования к функциональным характеристикам

Разрабатываемая программа должна реализовывать следующие функции:

1. Бронирование и учет гостей:

* Автоматический учет гостей и их персональных данных.
* Возможность быстрого и точного поиска информации о гостях.

1. Управление номерами:

* Система учета доступных и занятых номеров.
* Возможность управления статусом номеров (свободный, занятый, забронированный).

1. Управление персоналом:

* Учет рабочего времени сотрудников, графики работы, отпусков и смен.
* Закрепление сотрудников за определенным номером гостиницы.

1. Финансовый учет и отчетность:

* Ведения учета финансовых операций.
* Формирования статистики посещения и финансов в виде гистограмм.

1. Безопасность и защита данных:

* Хэширование паролей для пользователей программного обеспечения.
* Шифрование персональных данных клиентов в базе данных.

Пользовательский интерфейс должен обеспечивать наглядность предоставляемой информации и удобство использования.

* 1. Требования к структуре и функционированию

Разрабатываемая программа должна предоставлять пользователю простой и понятный интерфейс, и иметь следующие основные функциональные возможности: добавление, удаление или изменение данных о клиентах, персонале, комнатах и проживании, а также получении статистики в виде гистограмм.

4.3 Требования к надёжности

Система должна осуществлять обработку внутренних ошибок и вывод информации об ошибке пользователю.

Необходимо предусмотреть контроль правильности вводимой информации.

4.4 Требования к защите информации от несанкционированного доступа

Шифрование персональных данных клиентов в базе данных и хэширование паролей пользователей программного обеспечения.

4.5 Требования по сохранению информации при авариях

Защита от сбоев в сети питания осуществляется путём использования устройства бесперебойного питания.

4.6. Требования к видам обеспечения

* + 1. Информационное обеспечение

Диалог осуществляется через пользовательский интерфейс системы.

* + 1. Лингвистическое обеспечение

Интерфейс программы и логика обработки данных осуществляется с использованием языка Java.

4.6.4 Программное обеспечение

Программное обеспечение должно быть уровня, достаточного для поддержания исправного функционирования среды программирования IntelliJ IDEA, операционная система Windows 8.1 и выше.

* + 1. Техническое обеспечение

Для работы с программным обеспечением рекомендуется компьютер, оснащенный процессором с тактовой частотой не ниже 1,8 ГГц, двухъядерный или более; 4 ГБ ОЗУ; до 210 ГБ (минимум 800 МБ) свободного места на жестком диске; видеоадаптер с минимальным разрешением 720p (1280 х 720 пикселей); клавиатура и мышь; операционная система Windows 10.

* + 1. Организационное обеспечение

Программа не требует дополнительного организационного обеспечения.

* + 1. Требования к тестированию

Тестирование должно показать полную работоспособность программы, правильное выполнение поставленных перед программой задач. Тестирование разработанной системы следует проводить в соответствии с ГОСТ Р ИСО МЭК 107-99.

1. Требования к программной документации

Программная документация будет оформлена в виде пояснительной записки. Пояснительная записка должна быть выполнена согласно ГОСТ 2.105-95.

6 План работы над выпускной квалификационной работой

| Планируемая работа | Трудоёмкость  % | Срок выполнения | Отметка о выполнении |
| --- | --- | --- | --- |
| Анализ технического задания и обзор технической литературы | 15 | 1.03.24 |  |
| Разработка алгоритмов | 20 | 1.04.24 |  |
| Реализация программного обеспечения | 25 | 5.05.24 |  |
| Тестирование | 10 | 20.05.24 |  |
| Подготовка демонстрационного примера работы программного обеспечения | 15 | 25.05.24 |  |
| Оформление пояснительной записки | 15 | 02.06.24 |  |
| Сдача проекта на проверку | - | 13.06.24 |  |

1. Перечень графического материала

* Техническое задание
* Схемы алгоритмов
* Модульная структура программы
* Результаты тестирования

8 Рекомендуемая литература

1. HTML и CSS. Разработка и дизайн веб-сайтов / Джон Дакетт – Москва: Эксмо, 2019. – 480с. – (Мировой компьютерный бестселлер).
2. Философия JAVA. 4-е полное изд. / Б. Эккель – СПБ.: Питер, 2021. – 1168с.: ил. – (Серия «Классика computer science»).
3. Алгоритмы для начинающих: теория и практика для разработчика / Панос Луридас – Москва: Эксмо, 2020 – 608 с. – (Мировой компьютерный бестселлер).

**Сведения об авторе выпускной квалификационной работы:**

Домашний адрес: г. Смоленск, ул. Твардовского, д. 6А, кв. 77.

Телефон: 8-960-580-57-35.

Электронная почта: mpryadihin@mail.ru.

ПРИЛОЖЕНИЕ Б (КОД ПРОГАММЫ)

***SecurityConfig***package ru.mpryadihin.diplom.config;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
 private final UserDetailsService userDetailsService;  
  
 @Autowired  
 public SecurityConfig(UserDetailsService userDetailsService) {  
 this.userDetailsService = userDetailsService;  
 }  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.authorizeRequests()  
 .antMatchers("/auth/login","auth/registration", "/error").permitAll()  
 .anyRequest().hasAnyRole("ADMIN", "USER")  
 .and()  
 .formLogin().loginPage("/auth/login")  
 .loginProcessingUrl("/process\_login")  
 .defaultSuccessUrl("/main", true)  
 .failureUrl("/auth/login?error")  
 .and()  
 .logout().logoutUrl("/logout")  
 .logoutSuccessUrl("/auth/login");  
 }  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService)  
 .passwordEncoder(getPasswordEncoder());  
 }  
 @Bean  
 public PasswordEncoder getPasswordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

*ClientService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Client;  
import ru.mpryadihin.diplom.repositories.ClientRepository;  
import ru.mpryadihin.diplom.util.CryptoUtils;  
  
import java.util.List;  
import java.util.Optional;  
  
  
@Service  
@Transactional(readOnly = true)  
public class ClientService {  
  
 private final ClientRepository clientRepository;  
  
 private final CryptoUtils cryptoUtils;  
  
 private final SecretKeyService secretKeyService;  
  
 @Autowired  
 public ClientService(ClientRepository clientRepository, CryptoUtils cryptoUtils, SecretKeyService secretKeyService) {  
 this.clientRepository = clientRepository;  
 this.cryptoUtils = cryptoUtils;  
 this.secretKeyService = secretKeyService;  
 }  
 @Cacheable(value = "ClientService::findAll")  
 public List<Client> findAll(){  
 return clientRepository.findAll();  
 }  
 @Cacheable(value = "ClientService::findOne", key = "#id")  
 public Client findOne(int id){  
 Optional<Client> foundClient = clientRepository.findById(id);  
 foundClient.ifPresent(client -> client.setPassport(cryptoUtils.decrypt(client.getPassport(), secretKeyService.getKey())));  
 return foundClient.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ClientService::findAll", allEntries = true),  
 @CacheEvict(value = "ClientService::findOne", key = "#client.id")  
 })  
 public void save(Client client) {  
 try {  
 client.setPassport(cryptoUtils.encrypt(client.getPassport(), secretKeyService.getKey()));  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 clientRepository.save(client);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ClientService::findAll", allEntries = true),  
 @CacheEvict(value = "ClientService::findOne", key = "#id")  
 })  
 public void update(int id, Client updatedClient){  
 updatedClient.setId(id);  
 updatedClient.setPassport(cryptoUtils.encrypt(updatedClient.getPassport(), secretKeyService.getKey()));  
 clientRepository.save(updatedClient);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "ClientService::findOne", key = "#id")  
 public void delete(int id){  
 clientRepository.deleteById(id);  
 }  
 public List<Client> search(String s){  
 List<Client> searchResult = clientRepository.searchClientBySurname(s);  
 if (searchResult.isEmpty()) {  
 Client notFound = new Client();  
 notFound.setSurname("Не найдено");  
 notFound.setName("");  
 notFound.setPatronymic("");  
 searchResult.add(notFound);  
 }  
 return searchResult;  
 }  
}

*JobService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Job;  
import ru.mpryadihin.diplom.repositories.JobRepository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional(readOnly = true)  
public class JobService {  
  
 private final JobRepository jobRepository;  
  
 @Autowired  
 public JobService(JobRepository jobRepository) {  
 this.jobRepository = jobRepository;  
 }  
  
 @Cacheable(value = "JobService::findAll")  
 public List<Job> findAll() {  
 return jobRepository.findAll();  
 }  
 @Cacheable(value = "JobService::findOne", key = "#id")  
 public Job findOne(int id){  
 Optional<Job> foundedJob = jobRepository.findById(id);  
 return foundedJob.orElse(null);  
 }  
 @Caching(evict = {  
 @CacheEvict(value = "JobService::findAll", allEntries = true),  
 @CacheEvict(value = "JobService::findOne", key = "#job.id")  
 })  
 @Transactional(readOnly = false)  
 public void save(Job job){  
 jobRepository.save(job);  
 }  
 @Caching(evict = {  
 @CacheEvict(value = "JobService::findAll", allEntries = true),  
 @CacheEvict(value = "JobService::findOne", key = "#id")  
 })  
 @Transactional(readOnly = false)  
 public void update(int id, Job updatedJob){  
 updatedJob.setId(id);  
 jobRepository.save(updatedJob);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "JobService::findOne", key = "#id")  
 public void delete(int id){  
 jobRepository.deleteById(id);  
 }  
  
}

*PersonalService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.web.multipart.MultipartFile;  
import ru.mpryadihin.diplom.models.Personal;  
import ru.mpryadihin.diplom.repositories.PersonalRepository;  
import java.io.IOException;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional(readOnly = true)  
public class PersonalService {  
  
 private final PersonalRepository personalRepository;  
  
 @Autowired  
 public PersonalService(PersonalRepository personalRepository) {  
 this.personalRepository = personalRepository;  
 }  
  
 @Cacheable(value = "PersonalService::findAll")  
 public List<Personal> findAll(){  
 return personalRepository.findAll();  
 }  
 public Personal findOne(int id){  
 Optional<Personal> foundedPersonal = personalRepository.findById(id);  
 return foundedPersonal.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "PersonalService::findAll", allEntries = true),  
 })  
 public void save(Personal personal, MultipartFile file) throws IOException {  
 byte[] imageByte = file.getBytes();  
 personal.setImage(imageByte);  
 personalRepository.save(personal);  
  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "PersonalService::findAll", allEntries = true),  
 })  
 public void update(int id, Personal updatedPersonal, MultipartFile file) throws IOException {  
 if (file.isEmpty()) {  
 updatedPersonal.setImage(null);  
 updatedPersonal.setId(id);  
 } else {  
 byte[] imageByte = file.getBytes();  
 updatedPersonal.setId(id);  
 updatedPersonal.setImage(imageByte);  
 }  
 personalRepository.save(updatedPersonal);  
 }  
 @Transactional(readOnly = false)  
 public void delete(int id){  
 personalRepository.deleteById(id);  
 }  
}

*PersonDetailsService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Person;  
import ru.mpryadihin.diplom.models.Personal;  
import ru.mpryadihin.diplom.repositories.PersonRepository;  
import ru.mpryadihin.diplom.security.PersonDetails;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional(readOnly = true)  
public class PersonDetailsService implements UserDetailsService {  
  
 private final PersonRepository personRepository;  
 @Autowired  
 public PersonDetailsService(PersonRepository personRepository) {  
 this.personRepository = personRepository;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 Optional<Person> person = personRepository.findByUsername(username);  
 if (person.isEmpty())  
 throw new UsernameNotFoundException("Пользователь не найден");  
  
 return new PersonDetails(person.get());  
 }  
  
 public List<Person> findAll(){  
 return personRepository.findAll();  
 }  
  
 public Person findOne(int id){  
 Optional<Person> foundedPerson = personRepository.findById(id);  
 return foundedPerson.orElse(null);  
 }  
  
 @Transactional(readOnly = false)  
 public void save(Person person){  
 personRepository.save(person);  
 }  
 @Transactional(readOnly = false)  
 public void update(int id, Person updatedPerson){  
 updatedPerson.setId(id);  
 personRepository.save(updatedPerson);  
 }  
 @Transactional(readOnly = false)  
 public void delete(int id){  
 personRepository.deleteById(id);  
 }  
}

*RegistrationService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Person;  
import ru.mpryadihin.diplom.repositories.PersonRepository;  
  
@Service  
public class RegistrationService {  
 private final PersonRepository personRepository;  
 private final PasswordEncoder passwordEncoder;  
 @Autowired  
 public RegistrationService(PersonRepository personRepository, PasswordEncoder passwordEncoder) {  
 this.personRepository = personRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
 @Transactional  
 public void register(Person person){  
 person.setPassword(passwordEncoder.encode(person.getPassword()));  
 person.setRole("ROLE\_USER");  
 personRepository.save(person);  
 }  
}

*ResidesService*

package ru.mpryadihin.diplom.servicies;  
  
import org.modelmapper.ModelMapper;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Resides;  
import ru.mpryadihin.diplom.models.Room;  
import ru.mpryadihin.diplom.repositories.ResidesRepository;  
  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import static java.time.temporal.ChronoUnit.DAYS;  
  
@Service  
@Transactional(readOnly = true)  
public class ResidesService {  
 private final ResidesRepository residesRepository;  
 private final RoomService roomService;  
 @Autowired  
 public ResidesService(ResidesRepository residesRepository, RoomService roomService, ModelMapper modelMapper) {  
 this.residesRepository = residesRepository;  
 this.roomService = roomService;  
 }  
 @Cacheable(value = "ResidesService::findAll")  
 public List<Resides> findAll(){  
 return residesRepository.findAll();  
 }  
 @Cacheable(value = "ResidesService::findOne", key = "#id")  
 public Resides findOne(int id){  
 Optional<Resides> foundResides = residesRepository.findById(id);  
 return foundResides.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ResidesService::findAll", allEntries = true),  
 @CacheEvict(value = "ResidesService::findOne", key = "#resides.id")  
 })  
 public void save(Resides resides){  
 resides.setResultPrice(getResultPrice(resides));  
 residesRepository.save(resides);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ResidesService::findAll", allEntries = true),  
 @CacheEvict(value = "ResidesService::findOne", key = "#id")  
 })  
 public void update(int id, Resides updatedResides){  
 updatedResides.setResultPrice(getResultPrice(updatedResides));  
 updatedResides.setIdClient(id);  
 residesRepository.save(updatedResides);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "ResidesService::findOne", key = "#id")  
 public void delete(int id){  
 residesRepository.deleteById(id);  
 }  
 public int getResultPrice(Resides resides){  
 long days = DAYS.between(resides.getDateIn(),resides.getDateOut());  
 int price = (roomService.findOne(resides.getIdRoom())).getPrice();  
 return price \* (int) days;  
 }  
  
 public int dateValidation(Resides inResides){  
 List<Resides> residesList = residesRepository.findByIdRoom(inResides.getIdRoom());  
 LocalDate start = inResides.getDateIn();  
 LocalDate end = inResides.getDateOut();  
 int result = 0;  
 if (residesList.isEmpty()){  
 result = 1;  
 } else {  
 for (Resides res : residesList) {  
 if (res.getId() != inResides.getId()) {  
 if ((start.isAfter(res.getDateIn()) || start.equals(res.getDateIn())) && (start.isBefore(res.getDateOut()) || start.equals(res.getDateOut())) || (end.isAfter(res.getDateIn()) || end.equals(res.getDateIn())) && (end.isBefore(res.getDateOut()) || end.equals(res.getDateOut()))) {  
 result = 0;  
 break;  
 } else {  
 result = 1;  
 }  
 } else {  
 result = 1;  
 }  
 }  
 }  
 return result;  
 }  
  
 public List<Resides> findByYear(int year){  
 List<Resides> resides = residesRepository.findAll();  
  
 List<Resides> resultResides = new ArrayList<>();  
 for (Resides res : resides) {  
 if (res.getDateIn().getYear() == year || res.getDateOut().getYear() == year){  
 resultResides.add(res);  
 }  
 }  
 return resultResides;  
 }  
  
}

*RoomService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Resides;  
import ru.mpryadihin.diplom.models.Room;  
import ru.mpryadihin.diplom.repositories.ResidesRepository;  
import ru.mpryadihin.diplom.repositories.RoomRepository;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional(readOnly = true)  
public class RoomService {  
 private final RoomRepository roomRepository;  
 private final ResidesRepository residesRepository;  
 @Autowired  
 public RoomService(RoomRepository roomRepository, ResidesRepository residesRepository) {  
 this.roomRepository = roomRepository;  
 this.residesRepository = residesRepository;  
 }  
 @Cacheable(value = "RoomService::findAll")  
 public List<Room> findAll(){  
 return roomRepository.findAll();  
 }  
 @Cacheable(value = "RoomService::findOne", key = "#id")  
 public Room findOne(int id){  
 Optional<Room> foundedRoom = roomRepository.findById(id);  
 return foundedRoom.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "RoomService::findAll", allEntries = true),  
 @CacheEvict(value = "RoomService::findOne", key = "#room.id")  
 })  
 public void save(Room room){  
 room.setPrice(countPrice(room));  
 roomRepository.save(room);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "RoomService::findAll", allEntries = true),  
 @CacheEvict(value = "RoomService::findOne", key = "#id")  
 })  
 public void update(int id, Room updatedRoom){  
 updatedRoom.setPrice(countPrice(updatedRoom));  
 updatedRoom.setId(id);  
 roomRepository.save(updatedRoom);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "RoomService::findOne", key = "#id")  
 public void delete(int id){  
 roomRepository.deleteById(id);  
 }  
  
 public int countPrice(Room room){  
 int temp = switch (room.getIdTypeRoom()) {  
 case 1 -> 1000;  
 case 2 -> 1200;  
 case 3 -> 1500;  
 case 4 -> 2000;  
 case 5 -> 3500;  
 case 6 -> 4200;  
 case 7 -> 5000;  
 default -> throw new IllegalStateException("Unexpected value: " + room.getIdTypeRoom());  
 };  
 return temp + (room.getSeats() \* 400);  
 }  
  
 public List<Room> findFreeRooms(){  
 List<Room> all = roomRepository.findAll();  
 List<Resides> resides = residesRepository.findAll();  
 LocalDate now = LocalDate.now();  
 for (Resides res : resides){  
 if ((now.equals(res.getDateIn()) || now.isAfter(res.getDateIn())) && now.isBefore(res.getDateOut()) || now.equals(res.getDateOut())) {  
 all.removeIf(item -> item.getId() == res.getIdRoom());  
 }  
 }  
 return all;  
 }  
 public List<Resides> findDateResides(int idRoom){  
 List<Resides> resides = residesRepository.findByIdRoom(idRoom);  
 LocalDate now = LocalDate.now();  
 resides.removeIf(r -> r.getDateOut().isBefore(now));  
 return resides;  
 }  
}

*ScheduleService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.Personal;  
import ru.mpryadihin.diplom.models.Schedule;  
import ru.mpryadihin.diplom.repositories.PersonalRepository;  
import ru.mpryadihin.diplom.repositories.ScheduleRepository;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.IntStream;  
  
@Service  
@Transactional(readOnly = true)  
public class ScheduleService {  
  
 private final ScheduleRepository scheduleRepository;  
  
 private final PersonalRepository personalRepository;  
  
 @Autowired  
 public ScheduleService(PersonalRepository personalRepository, ScheduleRepository scheduleRepository, PersonalRepository personalRepository1) {  
 this.scheduleRepository = scheduleRepository;  
 this.personalRepository = personalRepository1;  
 }  
 @Cacheable(value = "ScheduleService::findAll")  
 public List<Schedule> findAll() {  
 return scheduleRepository.findAll();  
 }  
 @Cacheable(value = "ScheduleService::findOne", key = "#id")  
 public Schedule findOne(int id){  
 Optional<Schedule> foundSchedules = scheduleRepository.findById(id);  
 return foundSchedules.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ScheduleService::findAll", allEntries = true),  
 @CacheEvict(value = "ScheduleService::findOne", key = "#schedule.id")  
 })  
 public void save(Schedule schedule) {  
 scheduleRepository.save(schedule);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "ScheduleService::findAll", allEntries = true),  
 @CacheEvict(value = "ScheduleService::findOne", key = "#id")  
 })  
 public void update(int id, Schedule updatedSchedule){  
 updatedSchedule.setId(id);  
 scheduleRepository.save(updatedSchedule);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "ScheduleService::findAll")  
 public void updateAll(List<Schedule> updatedSchedule){  
 for (Schedule s : updatedSchedule){  
 Optional<Schedule> existingScheduleOptional = scheduleRepository.findById(s.getId());  
 if (existingScheduleOptional.isPresent()) {  
 Schedule existingSchedule = existingScheduleOptional.get();  
 existingSchedule.setIdPersonal(s.getIdPersonal());  
 existingSchedule.setMonth(s.getMonth());  
 existingSchedule.setYear(s.getYear());  
 existingSchedule.setDay1(s.getDay1());  
 existingSchedule.setDay2(s.getDay2());  
 existingSchedule.setDay3(s.getDay3());  
 existingSchedule.setDay4(s.getDay4());  
 existingSchedule.setDay5(s.getDay5());  
 existingSchedule.setDay6(s.getDay6());  
 existingSchedule.setDay7(s.getDay7());  
 existingSchedule.setDay8(s.getDay8());  
 existingSchedule.setDay9(s.getDay9());  
 existingSchedule.setDay10(s.getDay10());  
 existingSchedule.setDay11(s.getDay11());  
 existingSchedule.setDay12(s.getDay12());  
 existingSchedule.setDay13(s.getDay13());  
 existingSchedule.setDay14(s.getDay14());  
 existingSchedule.setDay15(s.getDay15());  
 existingSchedule.setDay16(s.getDay16());  
 existingSchedule.setDay17(s.getDay17());  
 existingSchedule.setDay18(s.getDay18());  
 existingSchedule.setDay19(s.getDay19());  
 existingSchedule.setDay20(s.getDay20());  
 existingSchedule.setDay21(s.getDay21());  
 existingSchedule.setDay22(s.getDay22());  
 existingSchedule.setDay23(s.getDay23());  
 existingSchedule.setDay24(s.getDay24());  
 existingSchedule.setDay25(s.getDay25());  
 existingSchedule.setDay26(s.getDay26());  
 existingSchedule.setDay27(s.getDay27());  
 existingSchedule.setDay28(s.getDay28());  
 existingSchedule.setDay29(s.getDay29());  
 existingSchedule.setDay30(s.getDay30());  
 existingSchedule.setDay31(s.getDay31());  
 scheduleRepository.save(existingSchedule);  
 }  
 }  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "ScheduleService::findOne", key = "#id")  
 public void delete(int id) {  
 scheduleRepository.deleteById(id);  
 }  
 @Transactional(readOnly = false)  
 public List<Schedule> findAllByYearAndMonth(int year, int month){  
 init(year, month);  
 return scheduleRepository.findAllByYearAndMonth(year, month);  
 }  
 public List<Integer> allDays(int year, int month){  
  
 List<Integer> days = new ArrayList<>();  
  
 switch (month) {  
 case 2 -> {  
 if (year % 4 == 0) {  
 days = IntStream.rangeClosed(1, 29)  
 .boxed()  
 .collect(Collectors.toList());  
 } else {  
 days = IntStream.rangeClosed(1, 28)  
 .boxed()  
 .collect(Collectors.toList());  
 }  
 }  
 case 4, 6, 9, 11 ->  
 days = IntStream.rangeClosed(1, 30)  
 .boxed()  
 .collect(Collectors.toList());  
 case 1, 3, 5, 7, 8, 10, 12 ->  
 days = IntStream.rangeClosed(1, 31)  
 .boxed()  
 .collect(Collectors.toList());  
 }  
 return days;  
 }  
 public Date getMonthValue(int month){  
 Date monthValue = new Date();  
 monthValue.setMonth(month-1);  
 return monthValue;  
 }  
 private void deleteAllScheduleByYear(int year){ //удаляем график работы за прошлый год  
 List<Schedule> scheduleList = scheduleRepository.findAllByYear(year - 1);  
 if (!scheduleList.isEmpty()) {  
 scheduleRepository.deleteAllByYear(year - 1); //удаляем раз в год, для статистики  
 }  
 }  
 private Schedule createSchedule(Personal personal, int year, int month){  
 Schedule schedule = new Schedule();  
 schedule.setIdPersonal(personal.getId());  
 schedule.setYear(year);  
 schedule.setMonth(month);  
 schedule.setPersonal(personal);  
 return schedule;  
 }  
  
 private void init(int year, int month){  
 deleteAllScheduleByYear(year);  
 List<Schedule> findData = scheduleRepository.findAllByYearAndMonth(year, month);  
 List<Personal> allPersonal = personalRepository.findAll();  
 for (Personal p : allPersonal){  
 if (findData.stream().noneMatch(x -> x.getIdPersonal() == p.getId())){  
 save(createSchedule(p, year, month));  
 }  
 }  
 }  
 public int getSalary(int year, int month){  
 List<Schedule> schedules = scheduleRepository.findAllByYearAndMonth(year,month);  
 if (schedules.isEmpty()) {  
 return 0;  
 } else {  
 return schedules.stream().mapToInt(x -> (x.getDay1() + x.getDay2() + x.getDay3() + x.getDay4() + x.getDay5() + x.getDay6() + x.getDay7() +  
 x.getDay8() + x.getDay9() + x.getDay10() + x.getDay11() + x.getDay12() + x.getDay13() + x.getDay14() + x.getDay15() + x.getDay16() +  
 x.getDay17() + x.getDay18() + x.getDay19() + x.getDay20() + x.getDay21() + x.getDay22() + x.getDay23() + x.getDay24() + x.getDay25() +  
 x.getDay26() + x.getDay27() + x.getDay28() + x.getDay29() + x.getDay30() + x.getDay31()) \* x.getPersonal().getSalary()).sum();  
 }  
 }  
}

*SecretKeySerivce*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import javax.crypto.SecretKey;  
import javax.crypto.spec.SecretKeySpec;  
import java.util.Base64;  
  
@Service  
public class SecretKeyService {  
 @Value("${secret.key}")  
 private String base64EncodedKey;  
  
 public SecretKey getKey() {  
 byte[] keyBytes = Base64.getDecoder().decode(base64EncodedKey);  
 return new SecretKeySpec(keyBytes, "AES");  
 }  
}

*StatisticService*

package ru.mpryadihin.diplom.servicies;  
  
import org.jfree.chart.ChartFactory;  
import org.jfree.chart.JFreeChart;  
import org.jfree.chart.axis.NumberAxis;  
import org.jfree.chart.block.BlockBorder;  
import org.jfree.chart.plot.CategoryPlot;  
import org.jfree.chart.renderer.category.BarRenderer;  
import org.jfree.data.category.CategoryDataset;  
import org.jfree.data.category.DefaultCategoryDataset;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import ru.mpryadihin.diplom.models.Resides;  
import ru.mpryadihin.diplom.repositories.ResidesRepository;  
  
import java.awt.\*;  
import java.time.Month;  
import java.util.\*;  
import java.util.List;  
  
@Service  
public class StatisticsService {  
  
 private final ResidesRepository residesRepository;  
 private final ResidesService residesService;  
  
 private final ScheduleService scheduleService;  
  
  
 @Autowired  
 public StatisticsService(ResidesRepository residesRepository, ResidesService residesService, ScheduleService scheduleService) {  
 this.residesRepository = residesRepository;  
 this.residesService = residesService;  
 this.scheduleService = scheduleService;  
 }  
  
 public List<String> getYears(){  
 List<String> result = new ArrayList<>();  
 List<Resides> resides = residesRepository.findAll();  
 if (resides.isEmpty()) {  
 return Collections.singletonList("Нет данных");  
 }  
 int temp = resides.get(0).getDateOut().getYear();  
 while (temp <= resides.get(resides.size()-1).getDateOut().getYear()){  
 result.add(String.valueOf(temp));  
 temp++;  
 }  
 return result;  
 }  
 private JFreeChart createChartResides(CategoryDataset dataset, int year){  
  
 JFreeChart chart = ChartFactory.createBarChart(  
 String.valueOf(year),  
 null, // x-axis label  
 "кол-во", // y-axis label  
 dataset);  
 chart.setBackgroundPaint(Color.white);  
  
 CategoryPlot plot = (CategoryPlot) chart.getPlot();  
  
 NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();  
 rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());  
 BarRenderer renderer = (BarRenderer) plot.getRenderer();  
 renderer.setDrawBarOutline(false);  
 chart.getLegend().setFrame(BlockBorder.NONE);  
  
 return chart;  
 }  
 private CategoryDataset createDataResides(int year){  
 DefaultCategoryDataset dataset = new DefaultCategoryDataset();  
 List<Resides> resides = residesService.findByYear(year);  
 HashMap<Month,Integer> map = new HashMap<>();  
 for (Month m : Month.values()) {  
 dataset.addValue(resides.stream().filter(r -> r.getDateIn().getMonth().equals(m)).count(), "посещаемость", m.toString());  
 }  
 return dataset;  
 }  
 public JFreeChart countResides(int year){  
 CategoryDataset dataset = createDataResides(year);  
 return createChartResides(dataset, year);  
 }  
  
 private CategoryDataset createDataIncomes(int year){  
 DefaultCategoryDataset dataset = new DefaultCategoryDataset();  
 List<Resides> resides = residesService.findByYear(year);  
 HashMap<Month,Integer> map = new HashMap<>();  
 int days = 0;  
 int incomes = 0;  
 for (Month m : Month.values()){  
 int salary = scheduleService.getSalary(year, m.getValue());  
  
 for (Resides r : resides){  
 if (r.getDateIn().getMonth().equals(r.getDateOut().getMonth()) && r.getDateIn().getMonth().equals(m)){  
 incomes = r.getResultPrice();  
 map.merge(m,incomes, Integer::sum);  
 incomes = 0;  
 continue;  
 }  
 if (r.getDateIn().getMonth().equals(m) && r.getDateOut().getMonth().getValue() > m.getValue()){  
 days = m.maxLength() - r.getDateIn().getDayOfMonth();  
 incomes = r.getRoom().getPrice() \* days;  
 map.merge(m,incomes, Integer::sum);  
 incomes = 0;  
 }  
 if (r.getDateIn().getMonth().getValue() < m.getValue() && r.getDateOut().getMonth().equals(m)){  
 days = r.getDateOut().getDayOfMonth();  
 incomes = r.getRoom().getPrice() \* days;  
 map.merge(m,incomes, Integer::sum);  
 incomes = 0;  
 }  
 if (!r.getDateIn().getMonth().equals(m) && !r.getDateOut().getMonth().equals(m)) {  
 incomes = 0;  
 map.merge(m,incomes, Integer::sum);  
 }  
 }  
 dataset.addValue(map.get(m)-salary, "рубли", m.toString());  
 }  
 return dataset;  
 }  
 private JFreeChart createChartIncomes(CategoryDataset dataset, int year){  
  
 JFreeChart chart = ChartFactory.createBarChart(  
 String.valueOf(year),  
 null, // x-axis label  
 "доходность", // y-axis label  
 dataset);  
 chart.setBackgroundPaint(Color.white);  
  
 CategoryPlot plot = (CategoryPlot) chart.getPlot();  
  
 NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();  
 rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());  
 BarRenderer renderer = (BarRenderer) plot.getRenderer();  
 renderer.setDrawBarOutline(false);  
 chart.getLegend().setFrame(BlockBorder.NONE);  
  
 return chart;  
 }  
 public JFreeChart countIncomes(int year){  
 CategoryDataset dataset = createDataIncomes(year);  
 return createChartIncomes(dataset, year);  
 }  
}

*TypeRoomService*

package ru.mpryadihin.diplom.servicies;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cache.annotation.CacheEvict;  
import org.springframework.cache.annotation.CachePut;  
import org.springframework.cache.annotation.Cacheable;  
import org.springframework.cache.annotation.Caching;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import ru.mpryadihin.diplom.models.TypeRoom;  
import ru.mpryadihin.diplom.repositories.TypeRoomRepository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Transactional(readOnly = true)  
public class TypeRoomService {  
  
 private final TypeRoomRepository typeRoomRepository;  
 @Autowired  
 public TypeRoomService(TypeRoomRepository typeRoomRepository) {  
 this.typeRoomRepository = typeRoomRepository;  
 }  
 @Cacheable(value = "TypeRoomService::findAll")  
 public List<TypeRoom> findAll(){  
 return typeRoomRepository.findAll();  
 }  
 @Cacheable(value = "TypeRoomService::findOne", key = "#id")  
 public TypeRoom findOne(int id){  
 Optional<TypeRoom> foundedType = typeRoomRepository.findById(id);  
 return foundedType.orElse(null);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "TypeRoomService::findAll", allEntries = true),  
 @CacheEvict(value = "TypeRoomService::findOne", key = "#typeRoom.id")  
 })  
 public void save(TypeRoom typeRoom){  
 typeRoomRepository.save(typeRoom);  
 }  
 @Transactional(readOnly = false)  
 @Caching(evict = {  
 @CacheEvict(value = "TypeRoomService::findAll", allEntries = true),  
 @CacheEvict(value = "TypeRoomService::findOne", key = "#id")  
 })  
 public void update(int id, TypeRoom updatedType){  
 updatedType.setId(id);  
 typeRoomRepository.save(updatedType);  
 }  
 @Transactional(readOnly = false)  
 @CacheEvict(value = "TypeRoomService::findOne", key = "#id")  
 public void delete(int id){  
 typeRoomRepository.deleteById(id);  
 }  
}

ПРИЛОЖЕНИЕ В (КОНФИГУРАЦИОННЫЕ ФАЙЛЫ)

*application.properties*

spring.datasource.driver-class-name=org.postgresql.Driver  
spring.datasource.url=jdbc:postgresql://localhost:5432/hotel  
spring.datasource.url=jdbc:postgresql://host.docker.internal:5432/hotel  
spring.datasource.username=postgres  
spring.datasource.password=2002  
secret.key=7MNxrZBp6qa43Jl/fTj9i2TOjnD1YGZwqApFhvHEFXg=  
  
  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  
spring.jpa.properties.hibernate.show\_sql=true  
  
spring.mvc.hiddenmethod.filter.enabled=true  
spring.cache.type=*redis*spring.cache.cache-names= redis-cache  
spring.redis.host=host.docker.internal  
spring.redis.port=6379  
  
server.error.whitelabel.enabled=false

*docker-compose.yml*

version: "3"  
  
volumes:  
 cache:  
 driver: local  
  
services:  
 web:  
 build: ./  
 container\_name: web  
 ports:  
 - "8080:8080"  
  
  
 redis:  
 image : redis:7.2-rc-alpine  
 restart: always  
 container\_name: redis  
 ports:  
 - "6379:6379"  
 command: redis-server --save 20 1 --loglevel warning  
 volumes:  
 - cache:/data

*Dockerfile*

FROM openjdk:17  
ADD /target/diplom-0.0.1-SNAPSHOT.jar web.jar  
  
ENTRYPOINT ["java", "-jar", "web.jar"]

*pom.xml*

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.5.1</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>ru.mpryadihin</groupId>  
 <artifactId>diplom</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>diplom</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.30</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.jfree</groupId>  
 <artifactId>jfreechart</artifactId>  
 <version>1.5.4</version>  
 </dependency>  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->  
 <dependency>  
 <groupId>org.modelmapper</groupId>  
 <artifactId>modelmapper</artifactId>  
 <version>3.2.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>jakarta.validation</groupId>  
 <artifactId>jakarta.validation-api</artifactId>  
 <version>2.0.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-java8</artifactId>  
 <version>5.6.15.Final</version>  
 </dependency>  
 <dependency>  
 <groupId>org.testcontainers</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>1.19.7</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.testcontainers</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>1.19.7</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-dependency-plugin</artifactId>  
 <version>2.10</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

ПРИЛОЖЕНИЕ Г (HTML ПРЕДСТАВЛЕНИЯ)

*edit\_user*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Изменить</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 text-align: center;  
 }  
 .top-nav input {  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .button-cancel {  
 margin-left: 1%;  
 }  
 select {  
 width: 300px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container {  
 position: absolute;  
 top: 60px;  
 margin-inline: auto;  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
 input[type=password] {  
 width: 300px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical;  
 }  
 input[type=text] {  
 width: 300px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 #bt\_change{  
 margin-left: 50px;  
 }  
  
 @media (max-width: 430px) {  
 .top-nav input {  
 font-size: 20px;  
 width: 430px;  
 }  
 #bt\_change{  
 margin-left: 45px;  
 }  
 }  
 @media (max-width: 375px){  
 .top-nav input {  
 width: 375px;  
 }  
 .container {  
 width: 270px;  
 }  
 input[type=password] {  
 width: 250px;  
 }  
 input[type=text] {  
 width: 250px;  
 }  
 select {  
 width: 250px;  
 }  
 #bt\_change{  
 margin-left: 35px;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 font-size: 16px;  
 width: 330px;  
 }  
 .container {  
 width: 250px;  
 }  
 input[type=password] {  
 width: 230px;  
 }  
 input[type=text] {  
 width: 230px;  
 }  
 select {  
 width: 230px;  
 }  
 #bt\_change{  
 margin-left: 30px;  
 }  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
</div>  
<br/>  
<div style="display: flex; justify-content: center">  
<div class="container">  
 <form style="display: inline" th:method="PATCH" th:action="@{/admin/{id}(id=${user.getId()})}" th:object="${user}">  
 <label for="username">Введите имя пользователя: </label>  
 <div style="color: red" th:if="${#fields.hasErrors('username')}" th:errors="\*{username}">username error</div>  
 <br/>  
 <input type="text" th:field="\*{username}" id="username"/>  
 <br/>  
 <label for="password">Введите пароль: </label>  
 <div style="color: red" th:if="${#fields.hasErrors('password')}" th:errors="\*{password}">password error</div>  
 <br/>  
 <input type="password" th:field="\*{password}" id="password"/>  
 <br/>  
 <label for="role">Выберите роль: </label>  
 <br/>  
 <select th:object="${user}" th:field="\*{role}" id="role">  
 <option th:value="ROLE\_USER">Пользователь</option>  
 <option th:value="ROLE\_ADMIN">Администратор</option>  
 </select>  
 <br/>  
 <input id="bt\_change" type="submit" value="Изменить">  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/admin/{id}(id=${user.getId()})}">  
 <input class="button-cancel" type="submit" value="Отмена">  
 </form>  
</div>  
</div>  
</body>  
</html>

*options*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Настроки</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .data {  
 padding: 14px 16px;  
 background-color: white;  
 }  
 .otstup {  
 margin-top: 6px;  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .data{  
 display: flex;  
 justify-content: center;  
 }  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .data{  
 display: flex;  
 justify-content: center;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .data{  
 display: flex;  
 justify-content: center;  
 }  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
 <form th:method="GET" th:action="@{/auth/registration}">  
 <input type="submit" value="Зарегистрировать"/>  
 </form>  
</div>  
<div class="data">  
 <div th:each="u : ${users}">  
 <div class="otstup">  
 <a th:href="@{/admin/{id}(id=${u.getId()})}" style="text-decoration: none; color: black"  
 th:text="${u.getUsername() + ' --- ' + u.getRole()}">Users & Roles</a>  
 </div>  
 </div>  
</div>  
</body>  
</html>

*show\_user*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Пользователь</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .container {  
 padding: 20px;  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .container p{  
 display: flex;  
 justify-content: center;  
 }  
 .btn{  
 display: flex;  
 justify-content: center;  
  
 }  
 .btn input {  
 margin-right: 10px;  
 }  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .container p{  
 display: flex;  
 justify-content: center;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .container p{  
 display: flex;  
 justify-content: center;  
 }  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
</div>  
<br/>  
<div class="container">  
 <p th:text="${'Имя пользователя: ' + user.getUsername()}">username</p>  
 <p th:text="${'Роль: ' + user.getRole()}">role</p>  
 <div class="btn">  
 <form style="display: inline" th:method="GET" th:action="@{/admin/{id}/edit(id=${user.getId()})}">  
 <input type="submit" value="Изменить"/>  
 </form>  
 <form style="display: inline" th:method="DELETE" th:action="@{/admin/{id}(id=${user.getId()})}">  
 <input type="submit" value="Удалить"/>  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/admin}">  
 <input type="submit" value="Назад">  
 </form>  
 </div>  
</div>  
</body>  
</html>

*login*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Аутентификация</title>  
 <style>  
 body {  
 margin: 0 auto;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .container {  
 position: absolute;  
 top: 20px;  
 margin-inline: auto;  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
 input[type=password] {  
 width: 300px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical;  
 }  
 input[type=text] {  
 width: 300px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 @media (max-width: 375px){  
 .container {  
 width: 270px;  
 }  
 input[type=password] {  
 width: 250px;  
 }  
 input[type=text] {  
 width: 250px;  
 }  
 }  
 @media (max-width: 330px) {  
 .container {  
 width: 250px;  
 }  
 input[type=password] {  
 width: 230px;  
 }  
 input[type=text] {  
 width: 230px;  
 }  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
<div style="display: flex; justify-content: center">  
 <div class="container">  
 <form name="f" th:method="POST" th:action="@{/process\_login}">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}"/>  
 <div th:if="${param.error}" style="color: red">  
 Неверные имя пользователя или пароль  
 </div>  
 <label for="username">Введите имя пользователя: </label>  
 <br/>  
 <input type="text" name="username" id="username"/>  
 <br/>  
 <label for="password">Введите пароль: </label>  
 <br/>  
 <input type="password" name="password" id="password"/>  
 <br/>  
 <input style="margin-left: 38%" type="submit" value="Войти"/>  
 </form>  
</div>  
</div>  
  
</body>  
</html>

*registration*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Регистрация</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .container {  
 display: flex;  
 justify-content: center;  
 margin-top: 50px;  
 }  
 input[type=password] {  
 width: 400px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 input[type=text] {  
 width: 400px;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 margin-left: 120px;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 .container {  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
 @media (max-width: 430px) {  
 .container {  
 width: 330px;  
 }  
 input[type=password] {  
 width: 310px;  
 }  
 input[type=text] {  
 width: 310px;  
 }  
 .container input[type=submit] {  
 margin-left: 85px;  
 }  
 }  
 @media (max-width: 375px){  
 .container {  
 width: 270px;  
 }  
 input[type=password] {  
 width: 250px;  
 }  
 input[type=text] {  
 width: 250px;  
 }  
 .container input[type=submit] {  
 margin-left: 50px;  
 }  
 }  
 @media (max-width: 330px) {  
 .container {  
 width: 250px;  
 }  
 input[type=password] {  
 width: 230px;  
 }  
 input[type=text] {  
 width: 230px;  
 }  
 .container input[type=submit] {  
 margin-left: 45px;  
 }  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
<div style="display: flex; justify-content: center">  
<div class="container">  
 <form th:method="POST" th:action="@{/auth/registration}" th:object="${person}">  
 <label for="username">Введите имя пользователя: </label>  
 <div style="color: red" th:if="${#fields.hasErrors('username')}" th:errors="\*{username}">username error</div>  
 <br/>  
 <input type="text" th:field="\*{username}" id="username"/>  
 <br/>  
 <label for="password">Введите пароль: </label>  
 <div style="color: red" th:if="${#fields.hasErrors('password')}" th:errors="\*{password}">password error</div>  
 <br/>  
 <input type="password" th:field="\*{password}" id="password"/>  
 <br/>  
 <input type="submit" value="Зарегистрировать"/>  
 </form>  
</div>  
</div>  
</body>  
</html>

*client\_edit*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Изменить</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .container {  
 padding: 3px 14px;  
 }  
 input[type=text] {  
 width: 100%;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 .container {  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
 .button-cancel {  
 margin-left: 1%;  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 90px;  
 }  
  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 65px;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 45px;  
 }  
 }  
 </style>  
</head>  
<body style="background: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
</div>  
<br/>  
<div class="container">  
 <form style="display: inline" th:method="PATCH" th:action="@{/client/{id}(id=${client.getId()})}" th:object="${client}">  
 <label for="name">Введите имя: </label>  
 <br>  
 <input type="text" th:field="\*{name}" id="name">  
 <br>  
 <div style="color: red" th:if="${#fields.hasErrors('name')}" th:errors="\*{name}">Name error</div>  
 <label for="surname">Введите фамилию: </label>  
 <br>  
 <input type="text" th:field="\*{surname}" id="surname">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('surname')}" th:errors="\*{surname}">Surname error</div>  
 <label for="patronymic">Введите отчество (необязательно): </label>  
 <br>  
 <input type="text" th:field="\*{patronymic}" id="patronymic">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('patronymic')}" th:errors="\*{patronymic}">Patronymic error</div>  
 <label for="passport">Введите паспортные данные: </label>  
 <br>  
 <input type="text" th:field="\*{passport}" id="passport">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('passport')}" th:errors="\*{passport}">Passport error</div>  
 <label for="birthday">Введите дату рождения: </label>  
 <br>  
 <input type="text" th:field="\*{birthday}" id="birthday">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('birthday')}" th:errors="\*{birthday}">Birthday error</div>  
 <label for="info">Дополнительная информация: </label>  
 <br>  
 <input type="text" th:field="\*{info}" id="info">  
 <input class="btn" type="submit" value="Изменить">  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/client/{id}(id=${client.getId()})}">  
 <input class="button-cancel" type="submit" value="Отмена">  
 </form>  
</div>  
</body>  
</html>

*client\_index*

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org" xmlns="http://www.w3.org/1999/html">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Клиенты</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .data {  
 padding: 14px 16px;  
 background-color: white;  
 }  
 .otstup {  
 margin-top: 6px;  
 }  
 .data input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 input[type=text] {  
 width: 100%;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 110px;  
 }  
  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 85px;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 65px;  
 }  
 }  
 </style>  
</head>  
<body style="background: wheat">  
  
  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
 <form th:method="GET" th:action="@{/client/new}">  
 <input type="submit" value="Добавить"/>  
 </form>  
</div>  
<br/>  
<div class="data">  
 <form style="display: inline" th:method="GET" th:action="@{/client/search}" th:object="${client}">  
 <label for="surname">Поиск:</label>  
 <br/>  
 <input type="text" placeholder="Введите фамилию:" id="surname" name="surname"/>  
 <input class="btn" type="submit" value="Поиск"/>  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/client}">  
 <input type="submit" value="Список">  
 </form>  
</div>  
  
<div class="data">  
  
 <div th:each="person : ${clients}">  
 <div class="otstup">  
 <a th:href="@{/client/{id}(id=${person.getId()})}" style="text-decoration: none ; color: black"  
 th:text="${person.getSurname() + ' ' + person.getName() + ' ' + person.getPatronymic()}">Clients</a>  
 </div>  
 </div>  
 <div th:each="s : ${search}">  
 <div class="otstup" th:if="${s.getSurname()} == 'Не найдено'">  
 <a>Не найдено</a>  
 </div>  
 <div class="otstup" th:if="${s.getSurname()} != 'Не найдено'">  
 <a th:href="@{/client/{id}(id=${s.getId()})}" style="text-decoration: none ; color: black"  
 th:text="${s.getSurname() + ' ' + s.getName() + ' ' + s.getPatronymic()}">Find clients</a>  
 </div>  
 </div>  
</div>  
</body>  
</html>

*client\_new*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Новый клиент</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .container {  
 padding: 3px 14px;  
 }  
 input[type=text] {  
 width: 100%;  
 padding: 12px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 margin-top: 6px;  
 margin-bottom: 16px;  
 resize: vertical  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 .container {  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
 .button-cancel {  
 margin-left: 1%;  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 90px;  
 }  
  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 65px;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 margin-left: 45px;  
 }  
 }  
 </style>  
</head>  
<body style="background: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
</div>  
<br/>  
<div class="container">  
 <form style="display: inline" th:method="POST" th:action="@{/client}" th:object="${client}">  
 <label for="name">Введите имя: </label>  
 <br>  
 <input type="text" th:field="\*{name}" id="name">  
 <br>  
 <div style="color: red" th:if="${#fields.hasErrors('name')}" th:errors="\*{name}">Name error</div>  
 <label for="surname">Введите фамилию: </label>  
 <br>  
 <input type="text" th:field="\*{surname}" id="surname">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('surname')}" th:errors="\*{surname}">Surname error</div>  
 <label for="patronymic">Введите отчество (необязательно): </label>  
 <br>  
 <input type="text" th:field="\*{patronymic}" id="patronymic">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('patronymic')}" th:errors="\*{patronymic}">Patronymic error</div>  
 <label for="passport">Введите паспортные данные: </label>  
 <br>  
 <input type="text" th:field="\*{passport}" id="passport">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('passport')}" th:errors="\*{passport}">Passport error</div>  
 <label for="birthday">Введите дату рождения: </label>  
 <br>  
 <input type="text" th:field="\*{birthday}" id="birthday">  
 <br>  
 <div style="color: red" th:if="\*{#fields.hasErrors('birthday')}" th:errors="\*{birthday}">Birthday error</div>  
 <label for="info">Дополнительная информация: </label>  
 <br>  
 <input type="text" th:field="\*{info}" id="info">  
 <input class="btn" type="submit" value="Добавить">  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/client}">  
 <input class="button-cancel" type="submit" value="Отмена">  
 </form>  
</div>  
</body>  
</html>

*client\_show*

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>Клиент</title>  
 <style>  
 body {  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 }  
 .top-nav {  
 background-color: #333;  
 overflow: hidden;  
 }  
 .top-nav input {  
 float: left;  
 color: white;  
 border-style: none;  
 text-align: center;  
 padding: 14px 16px;  
 text-decoration: none;  
 font-size: 17px;  
 background-color: #333;  
 }  
  
 .top-nav input:hover {  
 background-color: #ddd;  
 color: black;  
 }  
 .container {  
 padding: 20px;  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 }  
 .container input[type=submit] {  
 background-color: #333;  
 color: white;  
 padding: 12px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 }  
 .container input[type=submit]:hover {  
 background-color: #252525;  
 }  
 @media (max-width: 430px) {  
 .top-nav input {  
 width: 430px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 display: flex;  
 justify-content: center;  
 }  
 .btn input {  
 margin-right: 10px;  
 }  
  
 }  
 @media (max-width: 375px) {  
 .top-nav input {  
 width: 375px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 display: flex;  
 justify-content: center;  
 }  
 .btn input {  
 margin-right: 10px;  
 }  
 }  
 @media (max-width: 330px) {  
 .top-nav input {  
 width: 330px;  
 display: flex;  
 justify-content: center;  
 }  
 .btn {  
 display: flex;  
 justify-content: center;  
 }  
 .btn input {  
 margin-right: 10px;  
 }  
 }  
 </style>  
</head>  
<body style="background: wheat">  
<div class="top-nav">  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница"/>  
 </form>  
</div>  
<br/>  
<div class="container">  
 <p th:text="${'ФИО: ' + client.getSurname() + ' ' + client.getName() + ' ' + client.getPatronymic()}">FIO</p>  
 <p th:text="${'Паспорт: ' + client.getPassport()}">passport</p>  
 <p th:text="${'Дата рождения: ' + client.getBirthday()}">birthday</p>  
 <p th:text="${'Дополнительно: ' + client.getInfo()}">info</p>  
 <div class="btn">  
 <form style="display: inline" th:method="GET" th:action="@{/client/{id}/edit(id=${client.getId()})}">  
 <input type="submit" value="Изменить"/>  
 </form>  
 <form style="display: inline" th:method="DELETE" th:action="@{/client/{id}(id=${client.getId()})}">  
 <input type="submit" value="Удалить"/>  
 </form>  
 <form style="display: inline" th:method="GET" th:action="@{/client}">  
 <input type="submit" value="Список">  
 </form>  
 </div>  
</div>  
</body>  
</html>

*error\_403*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0">  
 <title>403</title>  
 <style>  
 .error-403 {  
 text-align: center;  
 }  
 input {  
 border-style: none;  
 background-color: #333333;  
 color : white;  
 text-align: center;  
 padding: 14px 16px;  
 border-radius: 12px;  
 }  
 </style>  
</head>  
<body style="background-color: wheat">  
  
<div class="error-403">  
 <H1>Данная функция вам недоступна</H1>  
 <form th:method="GET" th:action="@{/main}">  
 <input type="submit" value="Главная страница">  
 </form>  
</div>  
</body>  
</html>

ПРИЛОЖЕНИЕ Д (КОД БАЗЫ ДАННЫХ)

CREATE TABLE public.clients (

id integer NOT NULL,

name character varying(70) NOT NULL,

surname character varying(70) NOT NULL,

patronymic character varying(70),

passport character varying NOT NULL,

birthday date NOT NULL,

info character varying(200)

);

ALTER TABLE public.clients ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.clients\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

ALTER TABLE public.clients ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.clients\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

ALTER TABLE public.clients ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.clients\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

ALTER TABLE public.clients ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.clients\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

ALTER TABLE public.clients ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.clients\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

CREATE TABLE public.resides (

id integer NOT NULL,

id\_client integer,

id\_room integer,

date\_in date NOT NULL,

date\_out date NOT NULL,

result\_price integer

);

CREATE TABLE public.resides (

id integer NOT NULL,

id\_client integer,

id\_room integer,

date\_in date NOT NULL,

date\_out date NOT NULL,

result\_price integer

);

CREATE TABLE public.room (

id integer NOT NULL,

id\_type\_room integer NOT NULL,

price integer NOT NULL,

id\_personal integer NOT NULL,

seats integer NOT NULL,

CONSTRAINT room\_price\_check CHECK ((price > 0)),

CONSTRAINT room\_seats\_check CHECK ((seats > 0))

);

CREATE TABLE public.room (

id integer NOT NULL,

id\_type\_room integer NOT NULL,

price integer NOT NULL,

id\_personal integer NOT NULL,

seats integer NOT NULL,

CONSTRAINT room\_price\_check CHECK ((price > 0)),

CONSTRAINT room\_seats\_check CHECK ((seats > 0))

);

CREATE TABLE public.schedule (

id integer NOT NULL,

id\_personal integer,

year integer NOT NULL,

month integer NOT NULL,

day1 integer DEFAULT 0,

day2 integer DEFAULT 0,

day3 integer DEFAULT 0,

day4 integer DEFAULT 0,

day5 integer DEFAULT 0,

day6 integer DEFAULT 0,

day7 integer DEFAULT 0,

day8 integer DEFAULT 0,

day9 integer DEFAULT 0,

day10 integer DEFAULT 0,

day11 integer DEFAULT 0,

day12 integer DEFAULT 0,

day13 integer DEFAULT 0,

day14 integer DEFAULT 0,

day15 integer DEFAULT 0,

day16 integer DEFAULT 0,

day17 integer DEFAULT 0,

day18 integer DEFAULT 0,

day19 integer DEFAULT 0,

day20 integer DEFAULT 0,

day21 integer DEFAULT 0,

day22 integer DEFAULT 0,

day23 integer DEFAULT 0,

day24 integer DEFAULT 0,

day25 integer DEFAULT 0,

day26 integer DEFAULT 0,

day27 integer DEFAULT 0,

day28 integer DEFAULT 0,

day29 integer DEFAULT 0,

day30 integer DEFAULT 0,

day31 integer DEFAULT 0

);

ALTER TABLE public.schedule ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.schedule\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

CREATE TABLE public.type\_room (

id integer NOT NULL,

name character varying(50)

);

ALTER TABLE public.type\_room ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.type\_room\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

CREATE TABLE public.users (

id integer NOT NULL,

username character varying NOT NULL,

password character varying NOT NULL,

role character varying NOT NULL

);

ALTER TABLE public.users ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (

SEQUENCE NAME public.users\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);