Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Современные технологии обработки экономической информации»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель кафедры экономической информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. О. Петрович |
|  | \_\_\_.\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Автоматизированная система контроля**

**экспорта товаров»**

|  |  |
| --- | --- |
|  | Выполнил студент группы 972302  Бородын Виктор Викторович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2022  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

**РЕФЕРАТ**

**Бохан, М. А.** Автоматизированная система контроля экспорта товаров / М. А. Бохан. – Минск: БГУИР, 2023 – 106 с.

Пояснительная записка 99 с., 90 рис., 11 источников, 3 приложения

Ключевые слова: система, автоматизированная, бронирование, гостиница, номер, разработка, информационная, модель, схема, моделирование, диаграмма, анализ, тестирование, пользователь, администратор.

Объект исследования: туристическая компания, имеющая система интернет-бронирования отелей, для формирования заявок в конкретной стране или городе.

Предмет исследования: способы и пути повышения качества обслуживания клиентов и сокращение времени на обработку заявок.

Цель курсового проекта: является создание автоматизированной системы бронирования номеров в гостинице для упрощения и автоматизации процесса бронирования номеров для гостей, сокращении времени на обработку заявок и улучшении качества обслуживания.

Методология проведения работы: в процессе разработки автоматизированной системы использованы методы анализа, систематизации, классификации, обобщения данных, качественно-количественные методы обработки данных, функциональный анализ процессов, принципа построения баз данных, моделирование системы с помощью UML-диаграмм.

Результаты работы: проведен теоретический анализ, а также моделирование деятельности по осуществлению бронирования номеров в гостинице. Выполнено проектирование и конструирование автоматизированной системы бронирования номеров в гостинице. Осуществлено тестирование и проверка работоспособности системы. Предоставлена инструкция по развертыванию автоматизированной системы, а также сквозной тестовый пример реализации некоторых вариантов использования.

Область применения результатов: услуги по бронированию гостиничных номеров.

**СОДЕРЖАНИЕ**

[Перечень условных обозначений, символов и терминов 7](#_Toc121074599)

[Введение 8](#_Toc121074600)

[1 Описание процессов бронирования номера в гостинице 10](#_Toc121074601)

[1.1 Описание предметной области 10](#_Toc121074602)

[1.2 Разработка функциональной модели системы бронирования номеров в гостинице 11](#_Toc121074603)

[1.3 Определение требований к автоматизированной системе бронирования гостиничных номеров и примеры существующих решений на рынке 16](#_Toc121074604)

[2 Постановка задачи на разработку автоматизированной системы бронирования номеров в гостинице 21](#_Toc121074605)

[2.1 Определение задачи моделирования представлений автоматизированной системы бронирования номеров в гостинице 21](#_Toc121074606)

[2.2 Определение задачи проектирования информационной модели, хранящей данные о гостиничных номерах и пользователях 23](#_Toc121074607)

[2.3 Определение задачи программной реализации автоматизированной системы бронирования номеров в гостинице 23](#_Toc121074608)

[3 Моделирование представлений автоматизированной системы бронирования гостиничных номеров](#_Toc121074609)  25

[3.1 Диаграмма вариантов использования 25](#_Toc121074606)

[3.2 Диаграмма последовательности 27](#_Toc121074607)

[3.3 Диаграмма состояний 29](#_Toc121074608)

[3.4 Диаграмма классов 31](#_Toc121074606)

[3.5 Диашрамма развертывания 34](#_Toc121074607)

[4 Информационная модель автоматизированной системы бронирования номеров в гостинице](#_Toc121074609)  37

[5 Описание алгоритмов, реализующих бизнес-логику автоматизированной системы бронирования номеров в гостинице](#_Toc121074611) 41

[6 Обоснование выбора технологий, подходов для программной реализации автоматизированной системы бронирования номеров в гостинице 43](#_Toc121074612)

[7 Руководство пользователя автоматизированной системы системы бронирования номеров в гостинице 45](#_Toc121074615)

[7.1 Инструкция по развертыванию автоматизированной системы бронирования номеров в гостинице 45](#_Toc121074616)

[7.2 Рукоовдство пользователя автоматизированной системы бронирования номеров в гостинице 46](#_Toc121074617)

[Заключение 52](#_Toc121074621)

[Список использованных источников 54](#_Toc121074622)

[Приложение А (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат» 55](#_Toc121074623)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих основную бизнес-логику 56](#_Toc121074624)

[Приложение В (обязательное) Листинг SQL-скрипта 72](#_Toc121074625)

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

СУБД – Система управления базами данных

AWS – *Amazon Web Services* – облачная платформа для развертывания приложений

DI – *Dependency Injection* – принцип получения объектами других объектов по требованию

HTTP – *Hypertext Transfer Protocol* – протокол передачи данных в сети

IDEF0 – *Integrated DEFinition 0* – методология функционального моделирования

IDEF1X – *Integrated DEFinition 1 Extended* – методология моделирования описания данных

Java EE – *Java Enterprise Edition* – набор спецификаций, описывающий архитектуру для средних и крупных предприятий

JPA – *Java Persistence API* – Java API, предоставляющий механизм взаимодействия с базой данных посредством технологии ORM

JTA – *Java Transaction API* – Java API, предоставляющий механизм управления транзакциями в Java EE

JWT – *JSON Web Token* – стандарт создания токенов доступа в формате JSON

MVC – *Model View Controller* – архитектурный шаблон построения приложений, при котором логика разделена на модель, представление и контроллер

ORM – *Object-Relational Mapping* – технология отображения сущностей базы данных на POJO-классы

POJO – *Plain Old Java Object –* объект Java, не наследующийи не реализующий какой-либо функционал

REST – *Representational State Transfer* – архитектурный стиль взаимодействия компонентов распределенного приложения в сети

SQL – *Structured Query Language –* декларативный язык взаимодействия с реляционной базой данных

UML – *Unified Modeling Language –* графический язык визуализации, специфицирования, конструирования систем

Введение

В настоящее время грузовые перевозки остаются одним из важнейших звеньев в общей структуре функционирования экономики. Благодаря экспортным отношениям обеспечивается работа промышленности и сельского хозяйства, создаются условия для оптимального обращения товаров, укрепляются хозяйственные связи между различными регионами. От экспорта в огромной мере зависит эффективная деятельность организаций и фирм, занятых в сфере торговли. Рациональное использование транспорта позволяет более оперативно доставлять миллионы тонн товаров от места их изготовления до конечных потребителей продукта.

В условиях изменения системы хозяйственных связей внутри страны и интеграции экономики Республики Беларусь в мировой рынок, перевозка грузов должна осуществляться с минимальными транспортными издержками при одновременном обеспечении высококачественных транспортных услуг.

Недостаточный контроль водителя и состояния транспортного средства повышает риск хищения и утраты грузов, увеличивает расходы на топливо, простой и нецелевое использование транспорта. Так, например, незнание реального местонахождения транспорта ставит под угрозу своевременную доставку груза.

Автоматизированная система контроля экспорта товаров может решить задачу по оптимизации деятельности организации в сфере грузовых перевозок.

*Целью курсового проекта* является совершенствование организации экспорта товаров путем создания автоматизированной системы контроля временных издержек на формирование рейсов.

Поставленная цель требует решения следующих задач:

* анализ деятельности по осуществлению экспорта товаров для дальнейшего построения функциональной модели;
* определение ключевых требований, предъявляемых к автоматизированной системе, а также способов их исполнения;
* моделирование представлений автоматизированной системы;
* проектирование информационной модели, хранящей данные о совершаемых экспортных поставках;
* описание алгоритмов, реализующих ключевые бизнес-задачи автоматизированной системы;
* предоставление пользователю подробного руководства по взаимодействию с автоматизированной системой, включающего инструкцию по развертыванию, а также сквозные примеры функционала;
* тестирование и проверка работоспособности каждого модуля автоматизированной системы.

Объектом исследования курсового проекта является транспортная компания, осуществляющая как региональные, так и международные экспортные перевозки.

Предметом исследования курсового проекта являются способы и пути повышения качества мониторинга осуществления перевозок товаров

Транспорт, предоставляя услуги по экспорту как промышленным гигантам, так и физическим лицам, становится все более надежным, удобным и безопасным средством доставки сырья, промышленной и сельскохозяйственной продукций.

1. Описание процессов бронирования номера в гостинице
   1. Описание предметной области

Система интернет-бронирования является автоматизированной системой, которая позволяет гостям забронировать номер в гостинице через интернет. Эта система предоставляет удобный и быстрый способ бронирования номера, который может быть осуществлен из любой точки мира.

Процесс бронирования номера начинается с поиска подходящего номера на сайте гостиницы. Гости могут выбрать различные параметры, такие как дата заезда и выезда, количество гостей, тип номера и другие параметры, чтобы найти подходящий номер. Когда гость находит подходящий номер, он может забронировать его, заполнив форму бронирования на сайте.

Форма бронирования содержит информацию о госте, такую как имя, фамилия, адрес электронной почты и номер телефона. Гость также может указать дополнительные требования, такие как требования к питанию или услугам гостиницы. Когда гость заполняет форму бронирования, он должен также указать данные своей кредитной карты для оплаты бронирования.

Система интернет-бронирования автоматически проверяет наличие свободных номеров на выбранные даты и резервирует номер для гостя. Если номер недоступен на выбранные даты, система предложит другие варианты номеров или даты, которые могут быть доступны.

Когда бронирование подтверждено, гость получает электронное письмо с подтверждением бронирования и информацией о гостинице, включая адрес, контактную информацию и инструкции по заезду. Гость также может получить доступ к своему бронированию через систему интернет-бронирования, чтобы проверить информацию о бронировании, изменить или отменить бронирование.

Система интернет-бронирования также может быть связана с другими системами управления гостиницей, такими как система управления бронированием и система управления номерами. Это позволяет автоматически обновлять информацию о наличии номеров и избежать двойных бронирований.

Кроме того, система интернет-бронирования может предоставлять гостям дополнительные услуги, такие как заказ трансфера из аэропорта или услуги прачечной. Гости могут заказать эти услуги при бронировании номера через систему интернет-бронирования.

Система интернет-бронирования также может предоставлять гостинице ценную информацию о потребностях гостей и популярности различных услуг. Это помогает гостинице улучшить качество своих услуг и повысить удовлетворенность гостей.

В целом, система интернет-бронирования является эффективным и удобным способом бронирования номеров в гостинице для гостей. Она позволяет сократить время на обработку заявок, улучшить качество обслуживания и повысить доходы гостиницы.

* 1. Разработка функциональной модели системы бронирования номеров в гостинице

Автоматизированная система бронирования номеров будет представлять собой интернет площадку с возможностью поиска и выбора в первую очередь временного жилья.

Пользователь устанавливает критерии для поиска и выбора жилья и дополнительных жилищных услуг в заданных им самим странах, городах, заказывает авиабилеты для осуществления их бронирования или покупки, оставляет заявку на предоставление услуг. Каждый пользователь обязан будет иметь персональный аккаунт для использования и управления предоставляемых сайтом услуг.

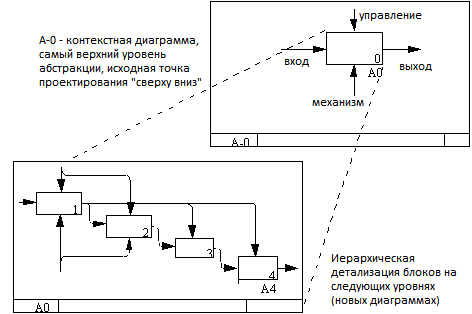
С целью отображения взаимодействия подсистемы с внешним миром построена контекстная диаграмма, которая является вершиной древовидной структуры диаграмм стандарта IDEF0.

IDEF0 обозначает определение интеграции для моделирования процессов — методологию создания общедоступных доменов, которая используется для моделирования предприятий и их процессов, чтобы их можно было понять и улучшить. Схемы IDEF0 (рисунок 1.1) обычно включают следующие компоненты:

− Контекстная схема – самая верхняя схема в модели IDEF0.

− Иерархия декомпозиции IDEF0 с использованием родительских и детских связей.

− Дерева узлов – структуры узлов в виде дерева, корневые на выбранном узле и используемые для представления полного декомпозиции IDEF0 в одной схеме.

Рисунок 1.1 - Основы графической нотации функционального моделирования бизнес-процессов IDEF0

Стрелка слева – это «вход» (обозначает начало процесса), без которого не может начаться процесс.

Сам процесс будет выполнен в конечный итог – «выход».

Стрелка «механизм» – это то, посредством чего процесс производится. Это могут быть люди или инструменты.

Стрелка «управление» – это управляющее воздействие на процесс, например, инструкция, алгоритм, спецификации.

Подводя итог возможностям и ограничениям IDEF0, еще раз подчеркнем достоинства этой методологии функционального моделирования:

− отлично подходит для иерархического проектирования «сверху вниз», от целых направлений деятельности до локальных бизнес-функций;

− позволяет комплексно представить все бизнес-процессы предприятия в виде единой системы из взаимосвязанных компонентов, отделяя ее внутреннее устройство от внешних объектов/сигналов;

− дает быстрое представление о наиболее важных характеристиках процесса (вход, выход, управление, механизмы), задействованных в его выполнении;

− является хорошим способом идентификации процессов-источников доходов и затрат.

Основную прибыль гостиница в первую очередь получает из бронирования номеров через интернет, располагая тарифами для частных лиц, а также для компаний. Для понимания процесса бронирования номера обеих категорий была воспроизведена функциональная модель в программе AllFusion.

Входные данные в данной модели представляют собой информацию о клиенте и гостиницах: данные клиента, такие как пароль и e-mail, а также имя, фамилия и дата рождения для неавторизованных пользователей.

Для управления процессом бронирования клиента выступают: документ «Политика в отношении обработки персональных данных» и закон Республики Беларусь «О защите персональных данных», которые дают гарантию безопасности и нераспространения данных клиента при предоставлении личной информации. Это необходимо так как при оформлении брони используются паспортные данные клиента.

Для бронирования номера в гостинице необходим пользователь, который является клиентом. Клиент заполняет все формы на сайте, для оформления брони. Также необходим администратор, который будет управлять процессом, держать контроль над пользователями и обрабатывать заявки со своей стороны.

В конечном итоге бронируется номер, а также, после проживания, появляется возможность оставить свой отзыв об оказанной или не оказанной услуге.

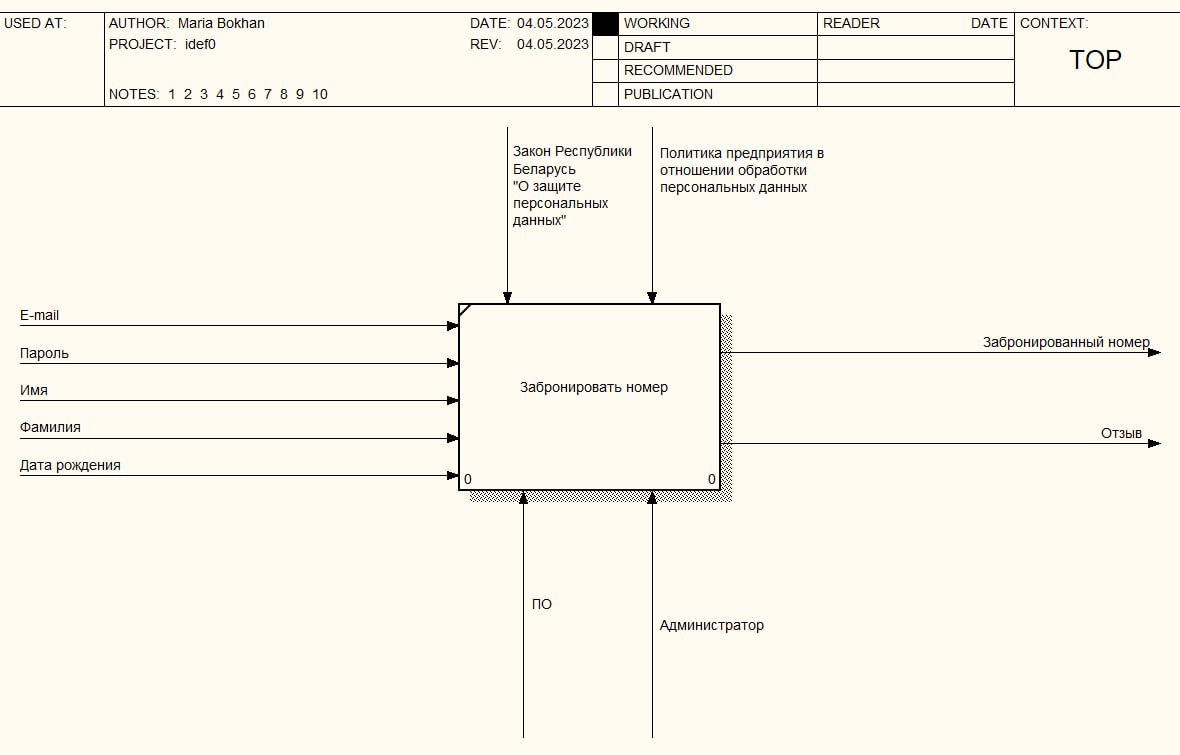
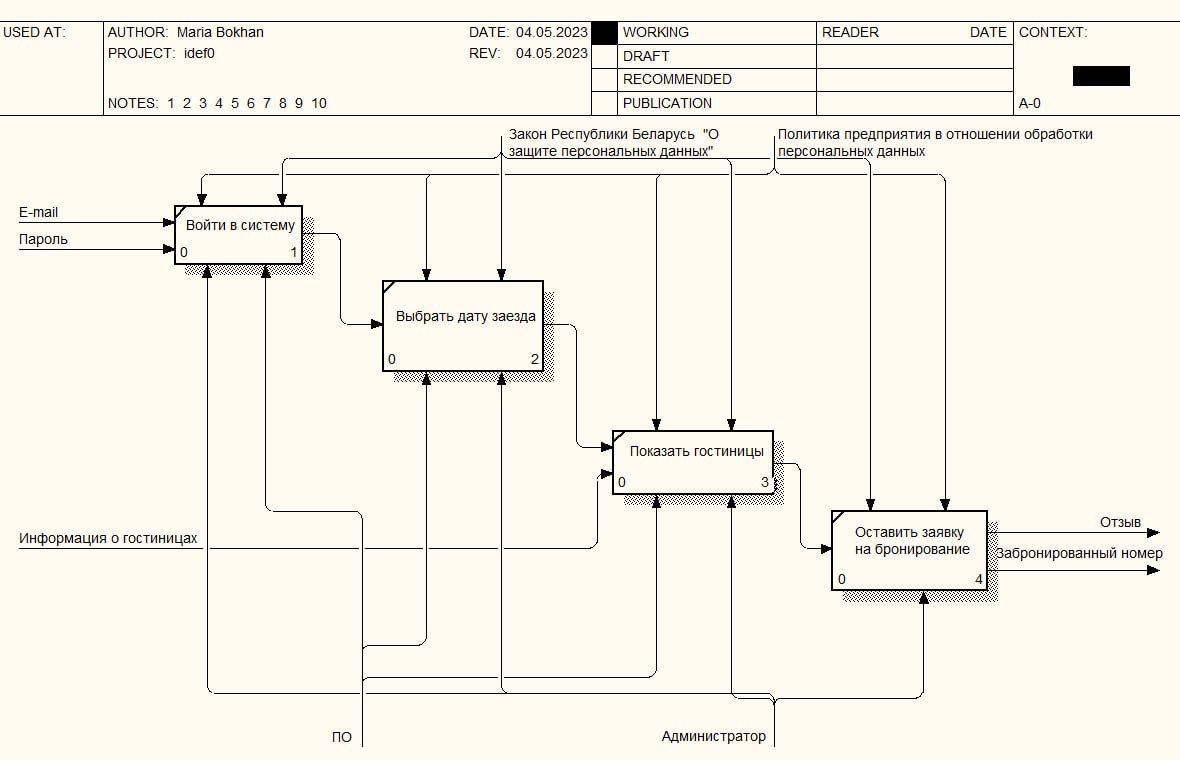
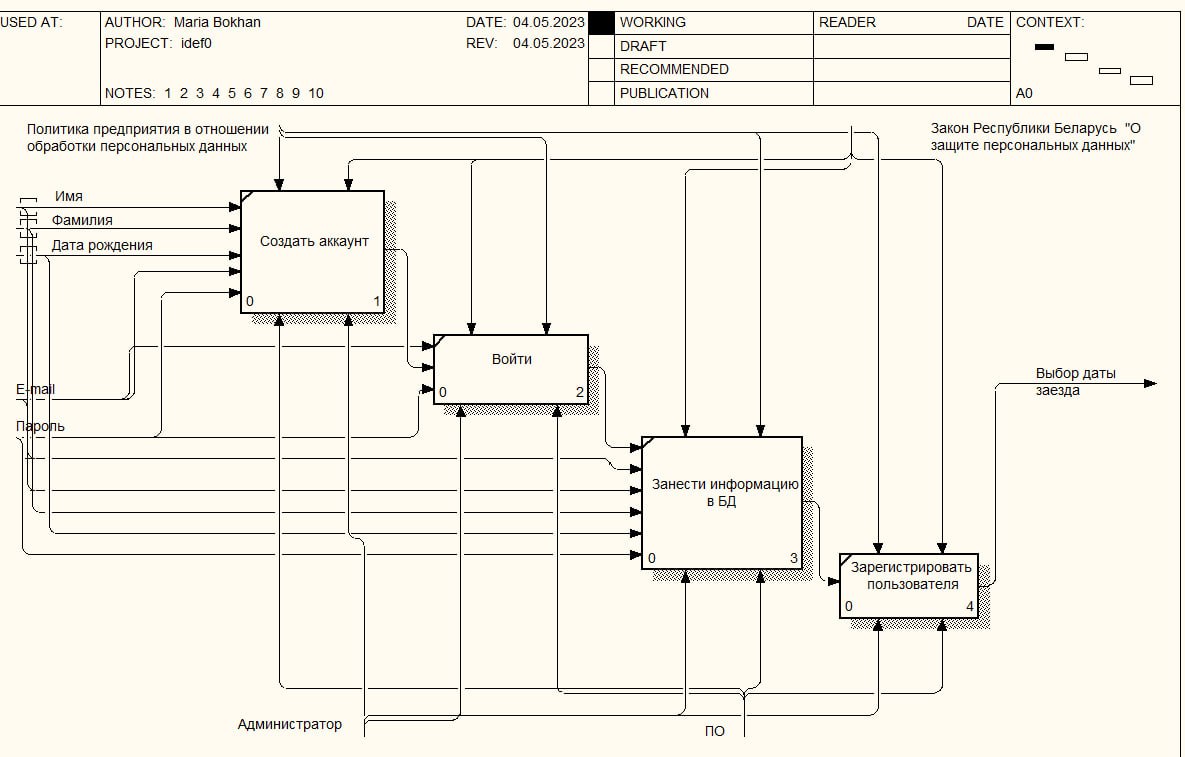
Для графического представления процесса бронирования номера в гостинице была представлена диаграмма верхнего уровня модели IDEF0 на рисунке 1.2.

Рисунок 1.2 – Контекстная диаграмма процесса «Забронировать номер в гостинице»

Чтобы забронировать номер, клиенту для начала необходимо зарегистрироваться и выполнить вход в систему. После этого необходимо указать желаемую дату своего заезда и выезда. После этого клиент может ознакомиться со списком подходящих для него гостиниц. После выбора понравившейся гостиницы клиенту необходимо оставить заявку на бронирование на сайте. Обработкой всего процесса управляют программное обеспечение и ответственный администратор. Контроль процесса осуществляется законом государства и политикой предприятия. Графическое представление данного процесса представлено на рисунке 1.3.

Рисунок 1.3 - Декомпозиция диаграммы IDEF0

Для подробного представления внутренних процессов системы сделаем декомпозицию блоков 1 (рисунок 2.4) и 3 (рисунок 2.5).

Рисунок 1.4 - Декомпозиция блока «Войти в систему»

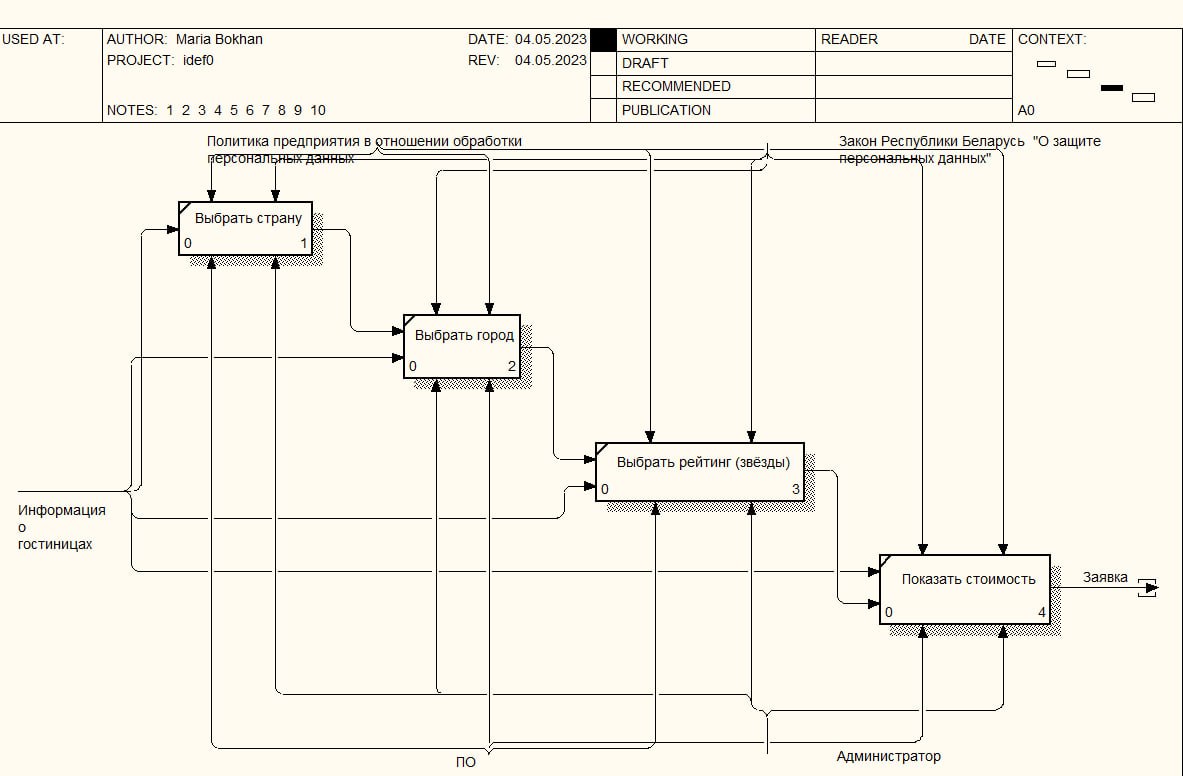


Рисунок 1.5 - Декомпозиция блока «Показать гостиницы»

Будем считать, что уровень декомпозиции рассмотренных диаграмм достаточен для отражения цели моделирования, и на диаграммах нижнего уровня в качестве наименований работ используются элементарные функции (с точки зрения пользователя системы).

Главным преимуществом строения диаграмм с помощью программного обеспечения — это возможность проверки правильности ее построения, проверка логического расположения блоков и так далее. Данная функция позволит избежать возможных ошибок в описании бизнес-процесса, сделает его более функциональным и эффективным.

* 1. Определение требований к автоматизированной системе бронирования гостиничных номеров и примеры существующих решений на рынке

В современном мире гостиничное дело развивается такими темпами, что первостепенной задачей по управлению отелем становятся высокий сервис в гостиничном бизнесе: качественные гостиничные услуги и грамотный гостиничный менеджмент. Гостиничные услуги, предъявляют все больше требований к управлению отелем. Комплексная автоматизация гостиниц, включающая в себя автоматизацию процессов бронирования, расчетов с гостями, агентами, туроператорами и хозяйственной деятельности становятся все более актуальными для предприятий гостиничной индустрии. Современные системы автоматизации гостиниц (PMS системы для гостиниц) включают в себя множество функций, среди которых:

Современные системы автоматизации гостиниц (PMS системы для гостиниц) включают в себя множество функций, среди которых:

* автоматизация баров и ресторанов;
* автоматизация систем управления гостиницей;
* системы онлайн бронирования гостиниц;
* система безопасности гостиницы;
* системы контроля доступа для гостиниц;
* система управления персоналом гостиницы.

Автоматизированная система бронирования номеров должна удовлетворять следующим требованиям:

1. Онлайн-бронирование: система должна обеспечивать возможность бронирования номеров через интернет.

2. Надежность: система должна быть надежной и стабильной, чтобы пользователи могли бронировать номера без сбоев.

3. Удобство использования: система должна быть простой в использовании, чтобы пользователи могли легко найти нужную информацию и забронировать номер.

4. Гибкость: система должна быть гибкой и адаптивной к различным типам номеров и услуг.

5. Многоплатформенность: система должна работать на различных устройствах, включая компьютеры, мобильные устройства и планшеты.

6. Интеграция: система должна интегрироваться с другими системами, такими как системы управления отелями и платежные системы.

7. Безопасность: система должна обеспечивать безопасность персональных данных пользователей и защиту от мошенничества.

8. Аналитика: система должна предоставлять аналитические данные о бронировании номеров для улучшения управления отелем и увеличения прибыли.

9. Поддержка: система должна иметь круглосуточную техническую поддержку для решения проблем пользователей.

10. Мультиязычность: система должна поддерживать несколько языков для удобства использования пользователей из разных стран.

Для определения популяризации данных требований для автоматизированных систем бронирования, рассмотрим примеры уже существующих решений на рынке, которые добились авторитета и доверия клиентов в наше время.

1. Современная гостиничная система автоматизации Shelter v.2 разработки компании UCS позволяет оформить гостя в течение считанных минут, сделать в целом процесс поселения и выселения быстрым и комфортным как для гостей, так и для персонала гостиницы.

Компания UCS является единственным разработчиком ПО, который может на базе собственных программных решений предложить полную автоматизацию всех зон гостиничного комплекса. Информационная система гостиницы Shelter взаимодействует с системой управления рестораном R-Keeper, программой автоматизации фитнес клуба и СПА - Абонемент, складской учетной системой StoreHouse.

2. Booking.com – система интернет-бронирования отелей. Бизнес-модель очень простая – отели выставляют предложения, пользователи выбирают подходящие и платят отелям, которые раз в месяц выплачивают Букингу комиссию. Главная страница сайта выполнена просто и понятно для пользователя (рисунок 1.6).

Информация о технологическом стеке и языках программирования, используемых в Booking.com, не является общедоступной. Однако известно, что Booking.com использует собственную разработку, которая базируется на микросервисной архитектуре и включает в себя компоненты, написанные на различных языках программирования, включая Java, Python, Ruby и др. Они также используют различные инструменты и технологии, такие как Docker, Kubernetes, Hadoop и другие.

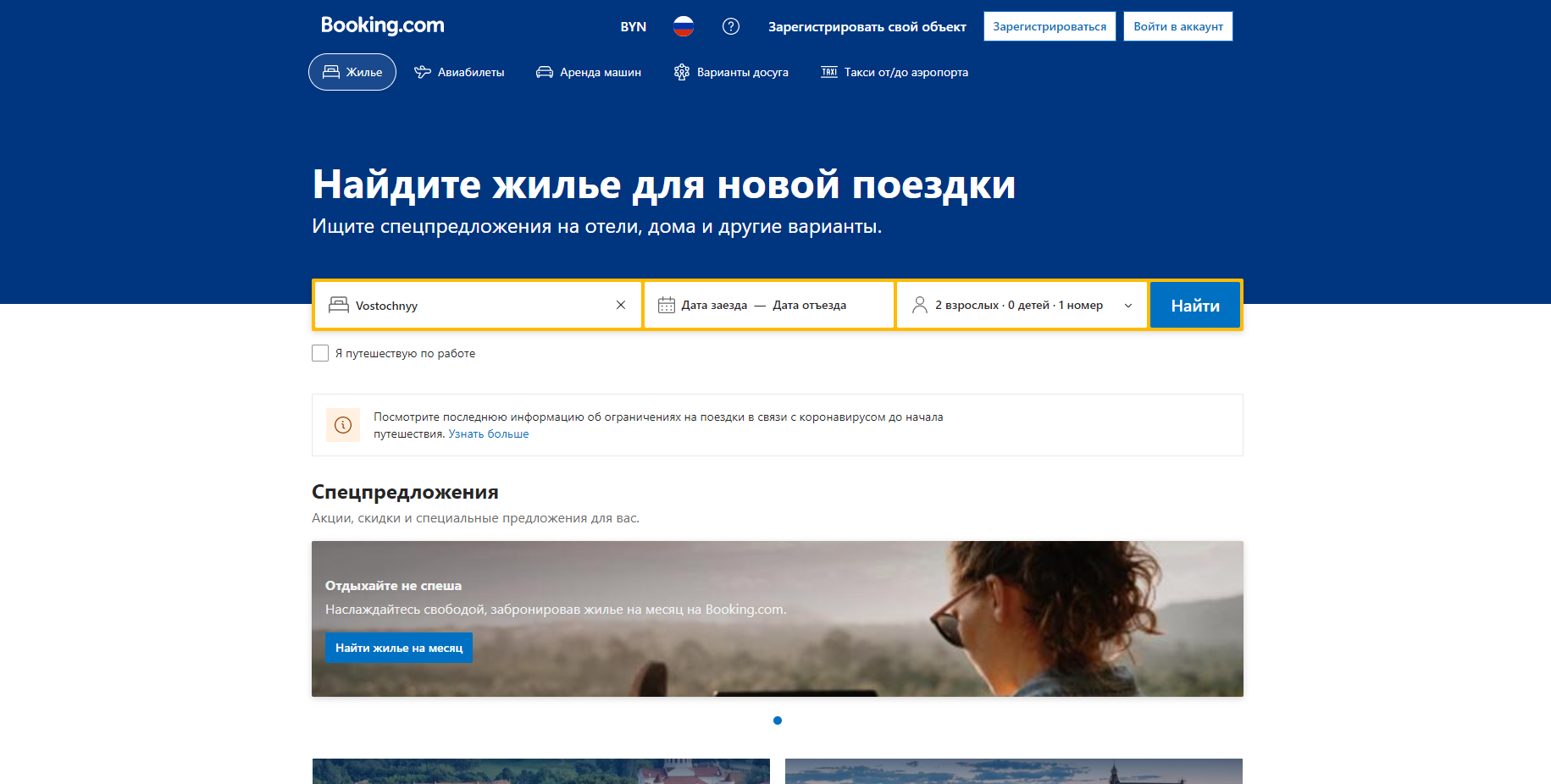


Рисунок 1.6 – Главная страница сайта Booking.com

Как видно из представления, интерфейс не предоставляет возможности изменить локализацию непосредственно при посещении главной страницы, что может стать проблемой для пользователя. При поиске отелей в системе Booking, можно установить несколько критериев в фильтре:

* звездность отелей;
* наличие завтраков в отеле;
* номер для некурящих;
* оплата (моментальная) или на месте;
* требуется ли кредитная карта при бронировании или нет;
* бесплатна отмена бронирования.

Сильные стороны: большой выбор отелей по всему миру; простота бронирования; возможность бронирования без кредитной карты; на сайте есть фото отеля, отзывы туристов, а также возможность посмотреть расположение отеле на карте (актуально для пляжного отдыха).

Слабые сторны: фото гостиницы могут быть устаревшие; фиктивные скидки; спам после регистрации.

3. Ostrovok.ru — российский сервис онлайн-бронирования отелей.

Идет большой упор на клиентскую часть, который требует того, чтобы код был тщательно организован и структурирован: выбор модульного подхода к разработке, основанный на ООП.

Это значит, что в системе активно используются классы, наследования, слабое связывание и прочие методы, которые предоставляет JavaScript.

Для каждой страницы есть отдельный JavaScript-класс, содержащий в себе какие-то вложенные модули, которые, в свою очередь, могут дробиться постоянно. Тщательная организация кода, также касается верстки: серверных и клиентских шаблонов и CSS. Требования к верстке те же самые, что и к JavaScript-приложению: функциональность, портативность, удобство сопровождения, портативность.

На главной странице в первую очередь представлен модуль бронирования (рисунок 1.7).

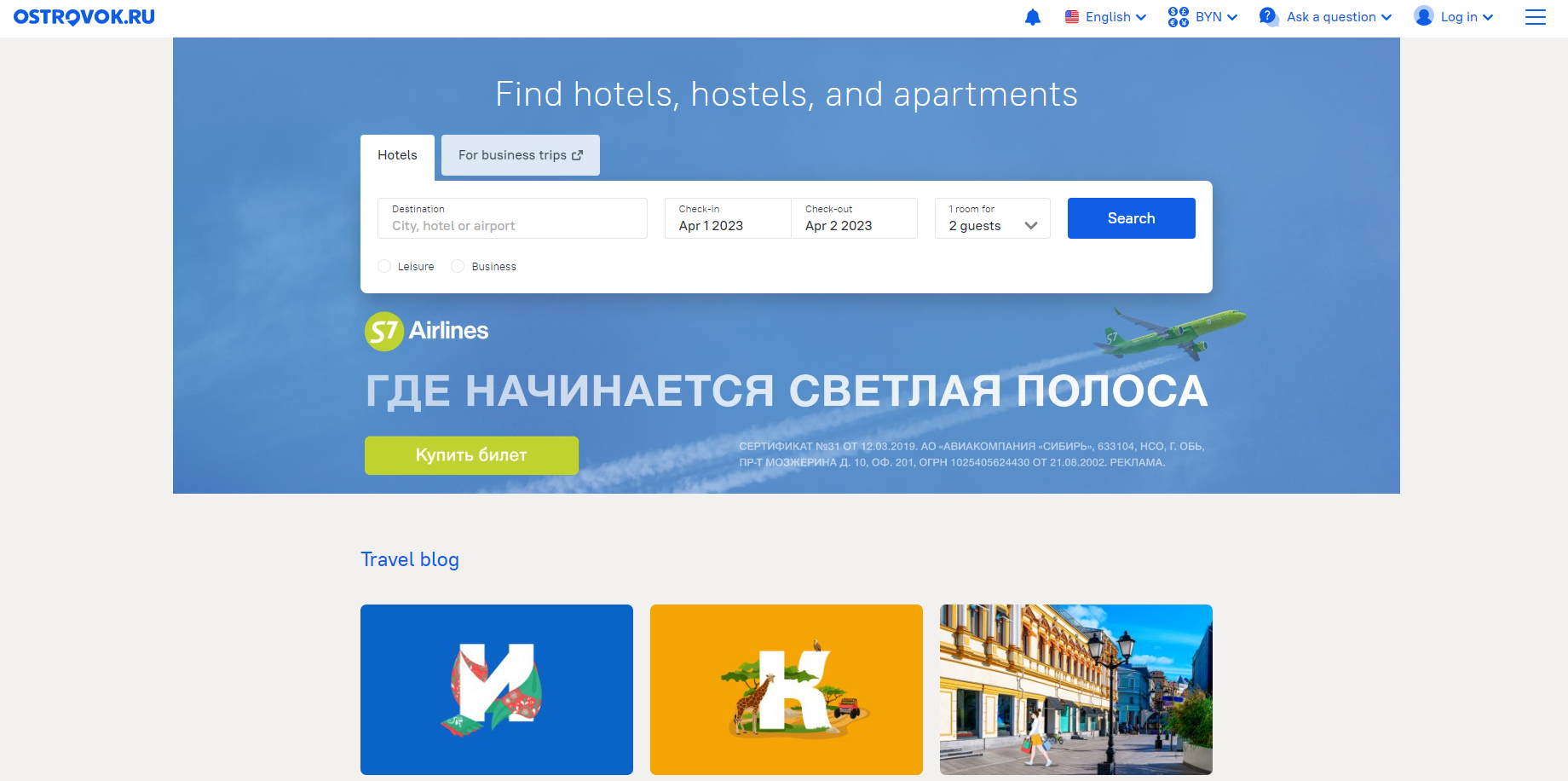


Рисунок 1.7 – Главная страница сайта Ostrovok.ru

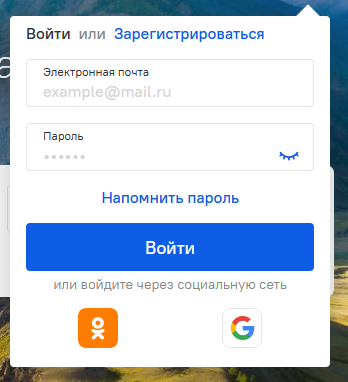
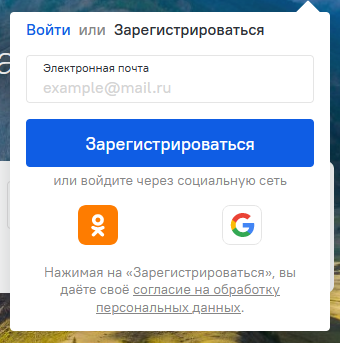
Форма авторизации и регистрации довольно простая и не требует лишней информации у пользователя (рисунок 1.8).

Рисунок 1.8 – Форма авторизации и регистрации сайта Ostrovok.ru

Плюс, с таким подходом к разработке, код проекта становится гибким и легко приспосабливаемым к нетривиальным ситуациям. Например, если перед разработчиком стоит задача кастомизировать страницу выдачи для пользователей, пришедших на наш сайт по поисковому запросу, а не с главной страницы, то задача ограничивается созданием дочернего класса и переопределением нескольких методов у родительского класса.

1. Постановка задачи на разработку автоматизированной системы бронирования номеров в гостинице

Для достижения цели, поставленной на курсовой проект, необходимо создание автоматизированной системы бронирования номеров в гостинице.

Ключевыми задачами, требующими выполнения для разработки автоматизированной системы, являются следующие:

* моделирование представлений автоматизированной системы бронирования номеров в гостинице;
* проектирование информационной модели, хранящей данные о гостиничных номерах и пользователях;
* программная реализация автоматизированной системы бронирования номеров в гостинице.

Программная реализация автоматизированной системы, исключая написание исходного кода, включает тестирование и руководство пользователя.

1. Определение задачи моделирования представлений автоматизированной системы бронирования номеров в гостинице

UML (Unified Modeling Language) - это стандартный язык моделирования, используемый для проектирования систем и программного обеспечения. Он позволяет описывать структуру, поведение и взаимодействие компонентов системы.

Диаграмма в UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами(сущностями) и ребрами (отношениями), используемое для визуализации системы с разных точек зрения. В UML выделяют 8 типов диаграмм (рисунок 2.1).

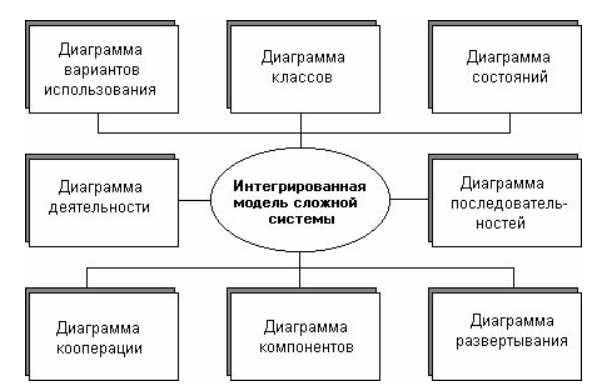
Рисунок 2.1 Интегрированная модель сложной системы в нотации UML.

Диаграмма вариантов использования показывает различные сценарии, которые могут произойти при бронировании номера клиентом.

Диаграмма классов показывает классы объектов, которые используются в процессе бронирования. Диаграмма классов также может показать связи между классами, например, что каждое бронирование связано с одним номером и одним клиентом. В целом, диаграмма классов помогает описать структуру системы бронирования номера в гостинице и идентифицировать все необходимые классы объектов для ее реализации.

Диаграмма состояний показывает все возможные состояния, которые может пройти процесс бронирования, начиная от начала процесса и заканчивая получением подтверждения бронирования.

Диаграмма последовательности показывает последовательность действий при бронировании номера клиентом.

Диаграмма развертывания показывает, какие компоненты системы используются для выполнения процесса бронирования и как они развертываются на физических устройствах. Например, диаграмма может показать, что клиент использует веб-интерфейс для ввода информации о бронировании, а сервер приложений обрабатывает эту информацию и сохраняет ее в базе данных. Диаграмма развертывания также может показать, на каких физических устройствах работают компоненты системы, например, веб-сервер может быть развернут на одном сервере, а база данных - на другом. Диаграмма развертывания помогает описать архитектуру системы и идентифицировать возможные проблемы с производительностью или масштабируемостью.

UML позволяет более наглядно и понятно описывать систему и ее компоненты, а также упрощает коммуникацию между разработчиками и заказчиками.

* 1. Определение задачи проектирования информационной модели, хранящей данные о гостиничных номерах и пользователях

Информационная модель описывает структуру и свойства объекта, его функции, процессы и взаимодействие с другими объектами. Информационная модель может быть представлена в виде диаграмм, таблиц, графов и других форматов. Она используется для проектирования систем, разработки программного обеспечения, управления бизнес-процессами и других задач. В случае системы бронирования номеров в гостинице информационная модель помогает определить необходимые функции, связи между ними и требования к базе данных.

Модель базы данных - это абстрактное представление базы данных, которое определяет ее структуру, типы данных, отношения между таблицами и другие характеристики. Она используется для проектирования и создания баз данных, а также для управления данными в них. Модель базы данных может быть представлена в виде диаграмм, таблиц или других форматов. Она позволяет определить связи между таблицами, типы данных, ограничения и правила целостности данных, а также определить доступ к данным и управление ими. В случае системы бронирования номеров в гостинице модель базы данных может помочь определить таблицы для хранения информации о номерах, клиентах, бронированиях и других сущностях, а также связи между ними.

Логическая модель базы данных описывает структуру данных и их отношения на более высоком уровне абстракции, чем физическая модель. Она определяет сущности, атрибуты и связи между ними, а также правила целостности данных. Логическая модель не зависит от конкретной СУБД и может быть использована для проектирования баз данных различных типов.

* 1. Определение задачи программной реализации автоматизированной системы бронирования номеров в гостинице

Серверная часть автоматизированной системы бронирования гостиничных номеров должна быть построена на основе набора спецификаций Java EE или фреймворка Spring, так как оба инструмента предлагают широкий спектр возможностей:

* контейнер внедрения зависимостей DI;
* менеджер управления транзакциями JTA;
* возможность объектно-реляционного отображения ORM сущностей базы данных на POJO.

Для представления доступа к защищенной информации при разработке автоматизированной системы может быть использован инструмент Spring Security, на основе JWT – токена, и протокола авторизации OAuth2.

Клиентская часть автоматизированной системы должна быть построена на технологиях React, Angular, так как на момент написания курсового проекта данные инструменты являются наиболее актуальными для построения пользовательских интерфейсов.

При разработке автоматизированной системы в качестве СУБД будет использован один из свободно распространяемых продуктов в лице MySQL, PostgreSQL, так как они подходят для построения сложных аналитических систем.

Разработка автоматизированной системы бронирования гостиничных номеров должна сопровождаться модульным и сквозным тестированиями.

По окончании разработки автоматизированной системы бронирования гостиничных номеров должны быть предоставлены руководства пользователя и по развертыванию системы.

Оптимальным при разработке будет использование паттернов проектирования, архитектуры.

1. моделирование представлений автоматизированной системы бронирования гостиничных номеров

С помощью UML можно разработать детальный план создаваемой системы, содержащий не только ее концептуальные элементы, такие как системные функции и бизнес-процессы, но и конкретные особенности, например, классы, написанные на каком-либо языке программирования, схемы баз данных и повторно используемые программные компоненты [6].

В данной главе будут представлены следующие модели представления:

* диаграмма вариантов использования;
* диаграмма последовательности;
* диаграмма состояний;
* диаграмма классов;
* диаграмма развертывания.
  1. Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case diagram) является одним из инструментов моделирования, который используется для описания взаимодействия между пользователем и системой. Она помогает анализировать требования системы и выделить основные функциональности, которые должны быть реализованы.

В центре диаграммы находится актёр – лицо или система, которые взаимодействуют с системой. Это может быть, к примеру, пользователь, другая система, внешний сервис, или даже другое ПО. Он изображается в виде человечка, пиктограммы или просто прямоугольника с именем (рисунок 3.1)

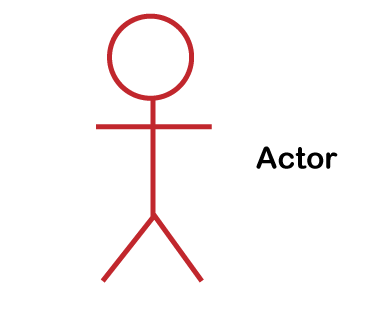
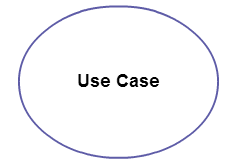
Вокруг актёра располагаются варианты использования (Use Case). Это называется привязкой вариантов использования к актёрам. Вариант использования описывает конкретный сценарий взаимодействия между актёром и системой. Он изображается в виде эллипса с именем (рисунок 3.1).

Рисунок 3.1 – Пример графического изображения актера и варинтов использования

.

Между актёрами и вариантами использования можно провести связи в виде стрелок. Они показывают, что актёр может вызвать определенный вариант использования и какой результат будет получен. Например, на диаграмме может быть показано, что пользователь может зарегистрироваться в системе и войти в свою учетную запись.

Диаграмма вариантов использования позволяет проанализировать возможности и ограничения системы, увидеть взаимосвязи между вариантами использования и сконцентрироваться на ключевых функциях, которые необходимо реализовать. Она также может использоваться для общения с заказчиком и другими заинтересованными сторонами, чтобы согласовать требования к системе.

Для информационной системы интернет-бронирования номеров гостиницы предусмотрены следующие роли (сущности-актеры): клиент, администратор гостиницы. Каждый актер системы (или вариант использования) включает в себя определенный набор действий.

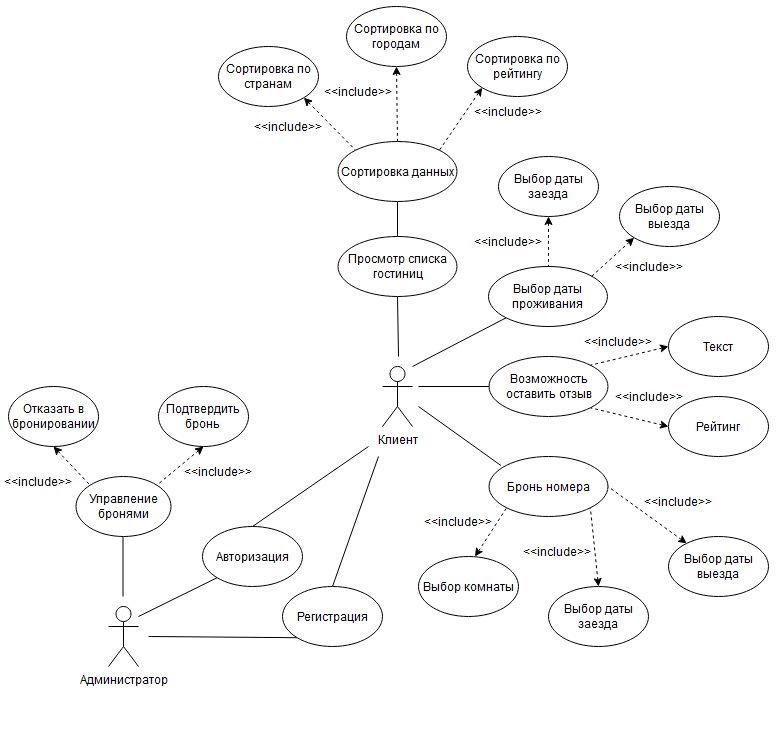
Клиент информационной системы интернет-бронирования номеров гостиницы имеет следующие функции:

* регистрация;
* авторизация;
* просмотр всех;
* подбор по критериям;
* бронь;
* отзыв.

Администратор гостиницы выполняет четыре действия:

* просмотр заявок;
* подтверждение заявки;
* отклонение заявки;
* возможность редактировать информацию о гостинице.

Диаграмма вариантов использования автоматизированной системы бронирования номеров представлена на рисунке 3.2.

Рисунок 3.2 - Диаграмма вариантов использования

* 1. Диаграмма последовательности

Диаграмма последовательности (sequence diagram) отображает взаимодействие между объектами в системе в хронологическом порядке. Каждый объект представлен в виде вертикальной линии, называемой линией жизни.

Контекст и назначение диаграммы последовательности:

1. Диаграмма последовательности показывает последовательность сообщений, передаваемых между объектами в системе.
2. Она используется для моделирования основных процессов в системе и их порядка выполнения.
3. Диаграмма последовательности может использоваться для представления алгоритмов и потока выполнения программного кода.
4. Её можно использовать для отображения процессов взаимодействия клиента и сервера, что позволяет легко улучшать информационное положение.

Пример диаграммы последовательности представлен на рисунке 3.3.

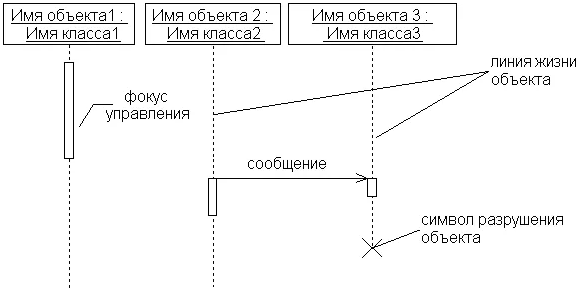


Рисунок 3.3 – Пример диаграммы последовательности

Пользователь взаимодействует с системой бронирования. Чтобы забронировать гостиницу, ему необходимо войти в систему и открыть нужную страницу, затем осуществить выборку гостиницы по заданным критериям. В ответ система выдаст список подходящих билетов для пользователя. В альтернативном блоке рассматривается ситуация, при которой искомый пользователем билет не будет найден в базе данных. При данной ситуации система сообщит пользователю о нехватке билета.

Диаграмма последовательности последовательности бронирования номера в гостинице представлена на рисунке 3.4.

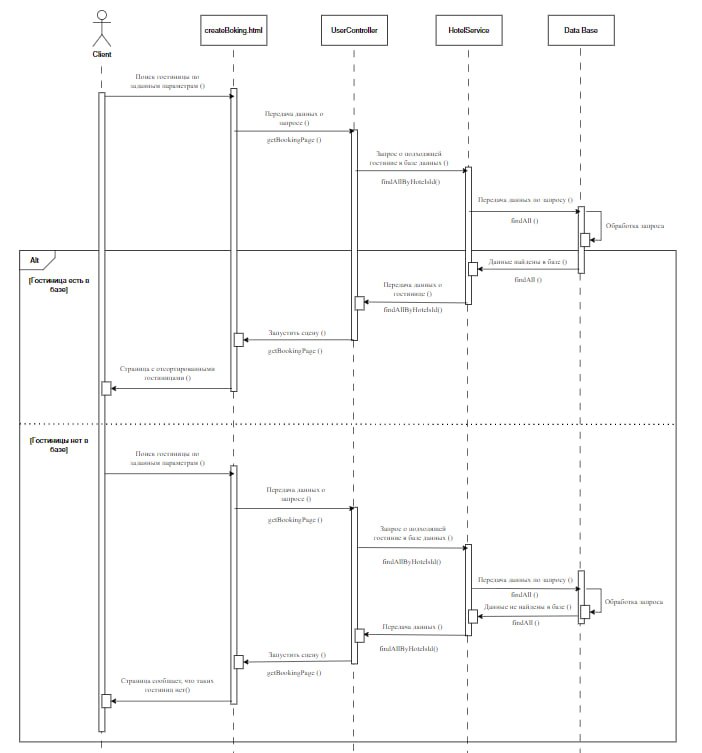


Рисунок 3.4 – Диаграмма последовательности бронирования номера в гостинице

* 1. Диаграмма состояний

Диаграмма состояний (Statechart diagram) – это диаграмма, предназначенная для моделирования динамических аспектов поведения систем. Она позволяет описать все возможные состояния объекта или системы и переходы между ними. Диаграмма состояний в UML помогает описать поведение объекта, его жизненный цикл и действия, которые должны быть выполнены при переходе между состояниями.

Основными элементами диаграммы состояний являются:

1. Состояние (State) - это условие, в котором находится объект или система. Состояние может быть описано в текстовом виде или графически представлено в виде прямоугольника с названием состояния.
2. Переход (Transition) - это действие, которое вызывает изменение состояния объекта или системы. Переход может быть вызван внешними событиями или внутренними условиями и описывается стрелкой, которая указывает направление перехода между состояниями.
3. Событие (Event) - это внешнее воздействие, которое вызывает переход между состояниями. Событие может быть вызвано пользователем, другими объектами или системой.
4. Действие (Action) - это действие, которое должно быть выполнено при переходе между состояниями. Действие может быть выполнено внутри объекта или системы, и описывается в виде текста или графически.

Диаграмма состояния процесса бронирования номера в гостинице представлена на рисунке 3.5.

Рисунок 3.5 - Диаграмма состояния процесса бронирования

Данная диаграмма описывает состояние бронирования гостиничного номера. Сначала пользователь выбирает дату своего заезда и выезда, если данные были введены правильно, то он переходит на следующий этап. Пользователь дальше подбирает гостиницу по критериям, сервер их обрабатывает, в случае успеха пользователю выводятся подходящие для него варианты. Если же гостиница, подходящая по всем критериям, не будет найдена, то пользователь возвращается на первоначальный этап. Далее пользователь выбирает необходимую для него гостиницу и оставляет заявку на бронирование. Затем сервером обрабатываются полученные данные. После решения администратора, на сервере подтверждается бронирование, и пользователь получает его. Если же администратор отказал в брони, то пользователь должен повторно ввести свои данные. Далее на сервере просматривается статус бронирования пользователя.

* 1. Диаграмма классов

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Целью создания диаграммы классов является графическое представление статической структуры декларативных элементов системы (классов, типов и т. п.). Она содержит в себе также некоторые элементы поведения (например – операции), однако их динамика должна быть отражена на диаграммах других видов (диаграммах коммуникации, диаграммах состояний). При представлении сущностей реального мира разработчику требуется отразить их текущее состояние, их поведение и их взаимные отношения.

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Примеры отображения класса представлены на рисунке 3.6.

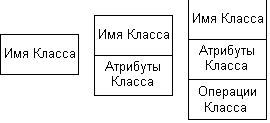


Рисунок 3.6 – Примеры отображения класса

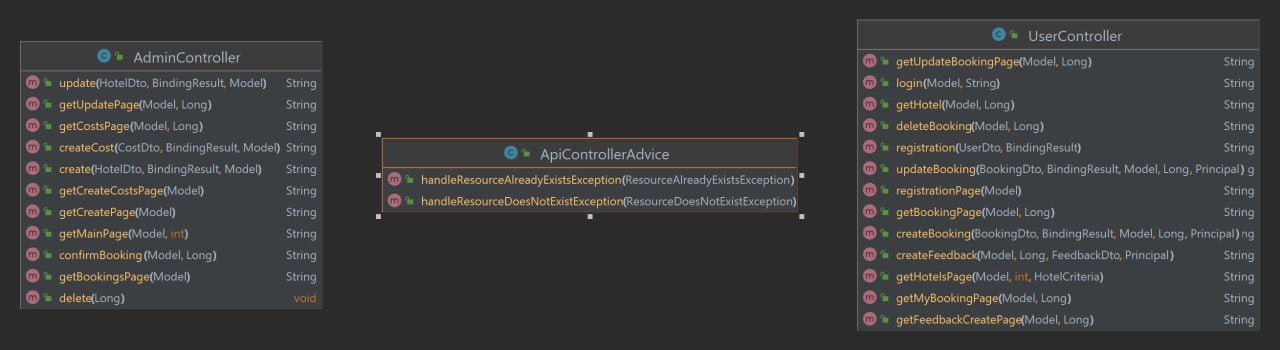
Диаграмма классов серверного программного средства представлена на рисунке 3.7.

Рисунок 3.7 - Диаграмма классов серверного программного средства

Контроллер смотрит на объект Request, создает и возвращает объект Response. Ответ может быть HTML-страницей, JSON, XML, сохраняемым файлом, редиректом, ошибкой 404 или чем-то другим. Контроллер может запускать любую произвольную логику, которая нужна приложению для отображения содержимого страницы.

Класс AdminController предназначен для управления работы открывающихся окон со стороны администратора.

Класс UserController предназначены для управления работой каждого окна, представленного для клиента.

Класс ApiControllerAdvice представляет собой класс рекомендаций контроллера для обработки исключений. Он перехватывает определенные исключения, создаваемые приложением, и сопоставляет их с соответствующим кодом состояния HTTP и сообщением. Сообщения, возвращаемые этими обработчиками, используются для информирования клиента об ошибке.

Сущности – это абстракции, которые являются основными элементами модели, связи соединяют их между собой, а диаграммы группируют представляющие интерес наборы сущностей. Диаграмма сущностей является интерпритацией таблиц из базы данных (рисунок 3.8).

БД позволяет хранить информацию в таблицах, для того чтобы это осуществить, у hibernate существует специальный механизм — mapping. Mapping-проецирование (сопоставление) Java классов с таблицами базы данных. Реализация маппинга произведена через аннотации: @Entity, @Table.

Рисунок 3.8 - Диаграмма классов сущностей

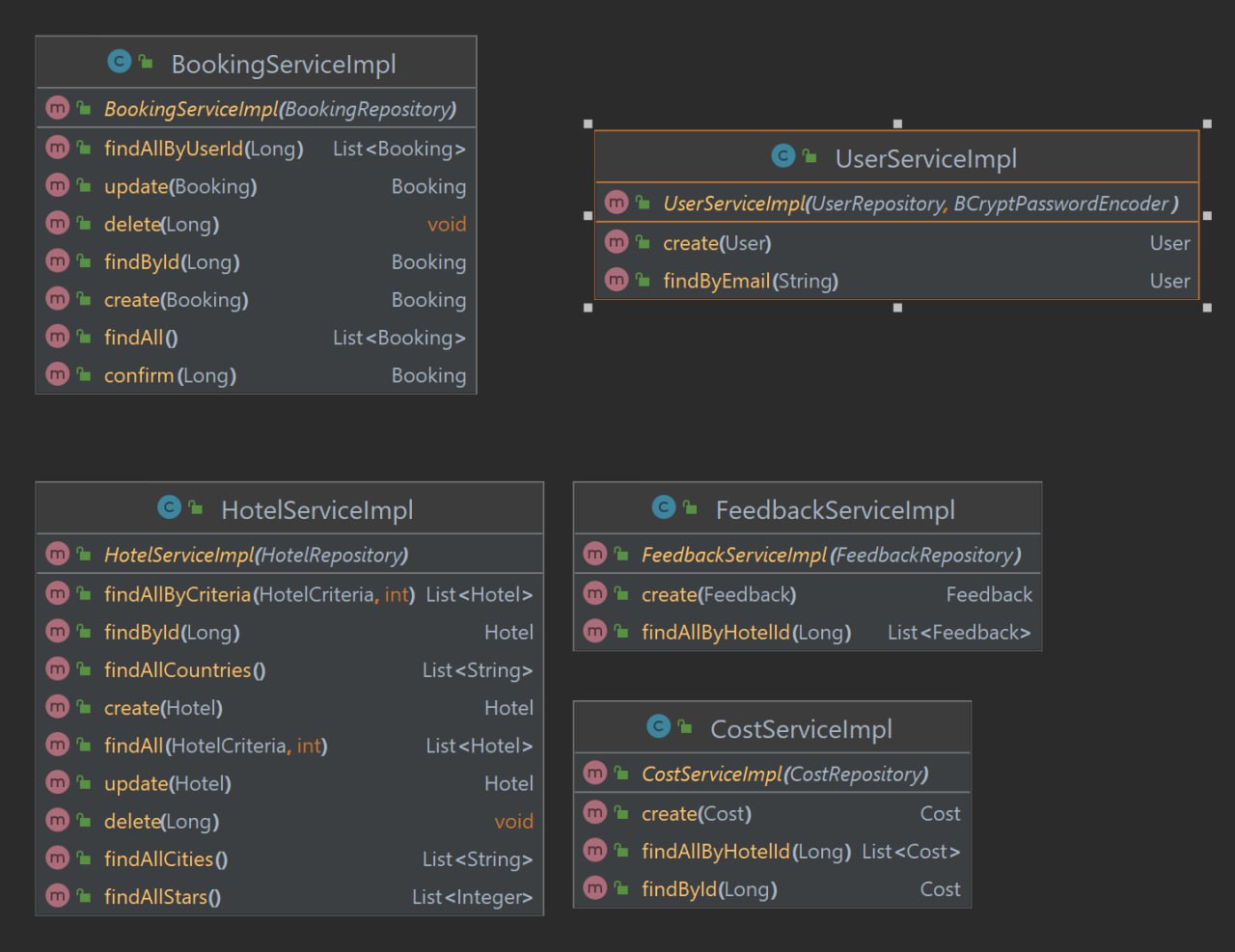
В полях классов сервисного уровня расположены объекты получения данных, которые содержат в себе все CRUD операции. Сервисный уровень объединяет все эти методы, чтобы определить более сложный функционал системы. Диаграмма классов сервисного уровня представлена на рисунке 3.9.

Рисунок 3.9 – Диаграмма классов сервисного уровня

* 1. Диаграмма развертывания

Диаграмма развертывания (Deployment diagram) в UML представляет собой модель, которая описывает аппаратную конфигурацию и топологию системы. Она показывает, как компоненты ПО распределены на устройствах и как они взаимодействуют друг с другом.

На диаграмме развертывания можно увидеть следующие элементы:

1. Устройства (Devices) – это физические или виртуальные устройства, на которых работает ПО. Например, серверы, клиентские компьютеры, мобильные устройства и т.д.

2. Компоненты (Components) – это части ПО, которые выполняют определенную функцию и могут быть установлены на различных устройствах. Например, базы данных, веб-серверы, приложения и т.д.

3. Связи (Connections) – это линии, которые показывают, как компоненты связаны между собой и с устройствами. Пример изображения связи представлен на рисунке 3.10.



Рисунок 3.10 – Пример изображения связи.

4. Узлы (Nodes) – это логические группы устройств, которые могут быть объединены в один узел. Например, кластер серверов может быть представлен как один узел на диаграмме развертывания. Пример изображения узла представлен на рисунке 3.11.

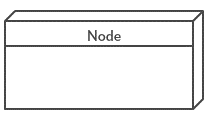


Рисунок 3.11 – Пример изображения узла.

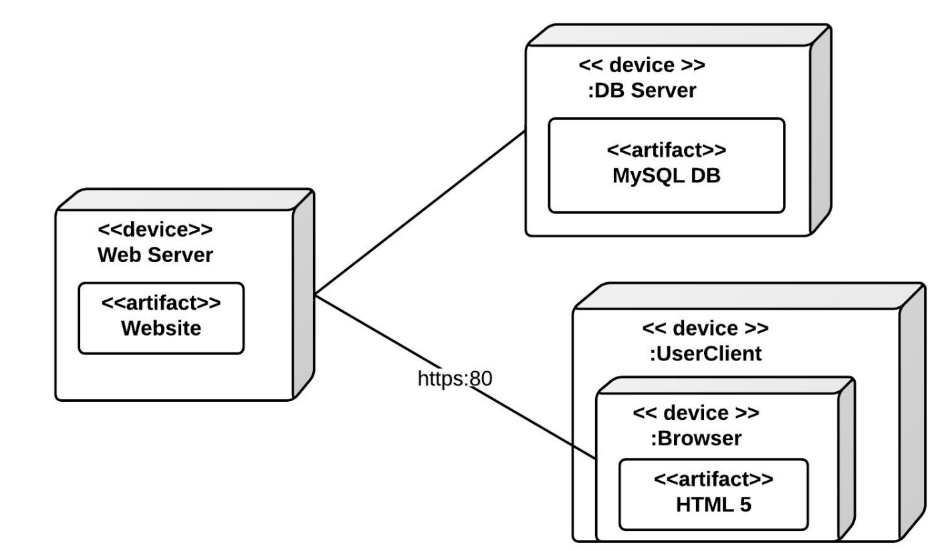
Диаграмма развертывания помогает разработчикам и архитекторам ПО лучше понимать, как система работает и как компоненты связаны между собой. Она также может использоваться для оптимизации производительности и масштабируемости системы. Пример изображения диаграммы развертывания представлен на рисунке 3.12.

Рисунок 3.12 – Пример изображения диаграммы развертывания

На рисунке 3.13 представлена диаграмма развертывания, где в качестве девайса выступает машинное обеспечение пользователя. В качестве сервера была использована программа Docker, которая позволяет облегчить и ускорить работу приложения. Данные, которые отправляются серверу, потом поступают в базу данных. Базой данных в данной курсовой работе выступает PostgreSQL.

Сервер взаимодействует с устройством посредством протокола HTTP.



Рисунок 4.6 - Диаграмма развертывания

В данной главе описано построение моделей представления автоматизированной системы контроля экспорта товаров. Рассмотрен унифицированный язык моделирования UML и его применение. Для построения диаграммы вариантов использования была разъяснена специфика отношений между вариантами. Для построения диаграммы последовательности были описаны основные виды сообщений. Для построения диаграммы состояний были изучены основные понятия в теории автоматов. Для построения диаграмм классов и развертывания были представлены примеры изображений используемых сущностей, а также разъяснены базовые отношения между ними.

Выполненный перечень работ позволяет перейти к дальнейшей разработке автоматизированной системы бронирования номеров в гостинице.

1. информационная модель автоматизированной системы бронирования номеров в гостинице

Для представления структуры хранимых данных программного средства выбрана информационная методология IDEF1X.

IDEF1X (Integrated Definition Language for Information Modeling Express) - язык моделирования данных, разработанный в 1970-х годах для описания структур баз данных. Он используется для создания диаграмм, которые показывают сущности (таблицы), их атрибуты (столбцы) и связи между ними. IDEF1X также предоставляет нотацию для определения ограничений целостности данных, таких как первичные ключи, внешние ключи и уникальные ограничения. Этот язык широко используется в инженерии баз данных и представляет собой стандарт ANSI/ISO.

1. Сущность (Entity) - это объект, который хранится в базе данных и имеет свои атрибуты. Сущность может быть физическим или концептуальным объектом.

2. Атрибут (Attribute) - это свойство сущности, которое описывает ее. Атрибуты могут быть простыми (например, имя, адрес) или составными (например, дата рождения).

3. Связь (Relationship) - это связь между двумя или более сущностями. Она может быть однонаправленной или двунаправленной, и может иметь определенный тип (например, один-ко-многим или многие-ко-многим).

4. Ключ (Key) - это уникальный идентификатор сущности в базе данных. Он может быть первичным ключом (Primary Key), который уникально идентифицирует каждую запись в таблице, или внешним ключом (Foreign Key), который связывает две таблицы между собой.

5. Ограничение целостности (Integrity Constraint) - это правило, которое определяет допустимые значения для атрибутов и связей в базе данных. Ограничения целостности могут быть первичными ключами, внешними ключами, уникальными ограничениями или ограничениями на значения атрибутов.

Ниже на рисунке 4.1 приведена информационная модель базы данных.

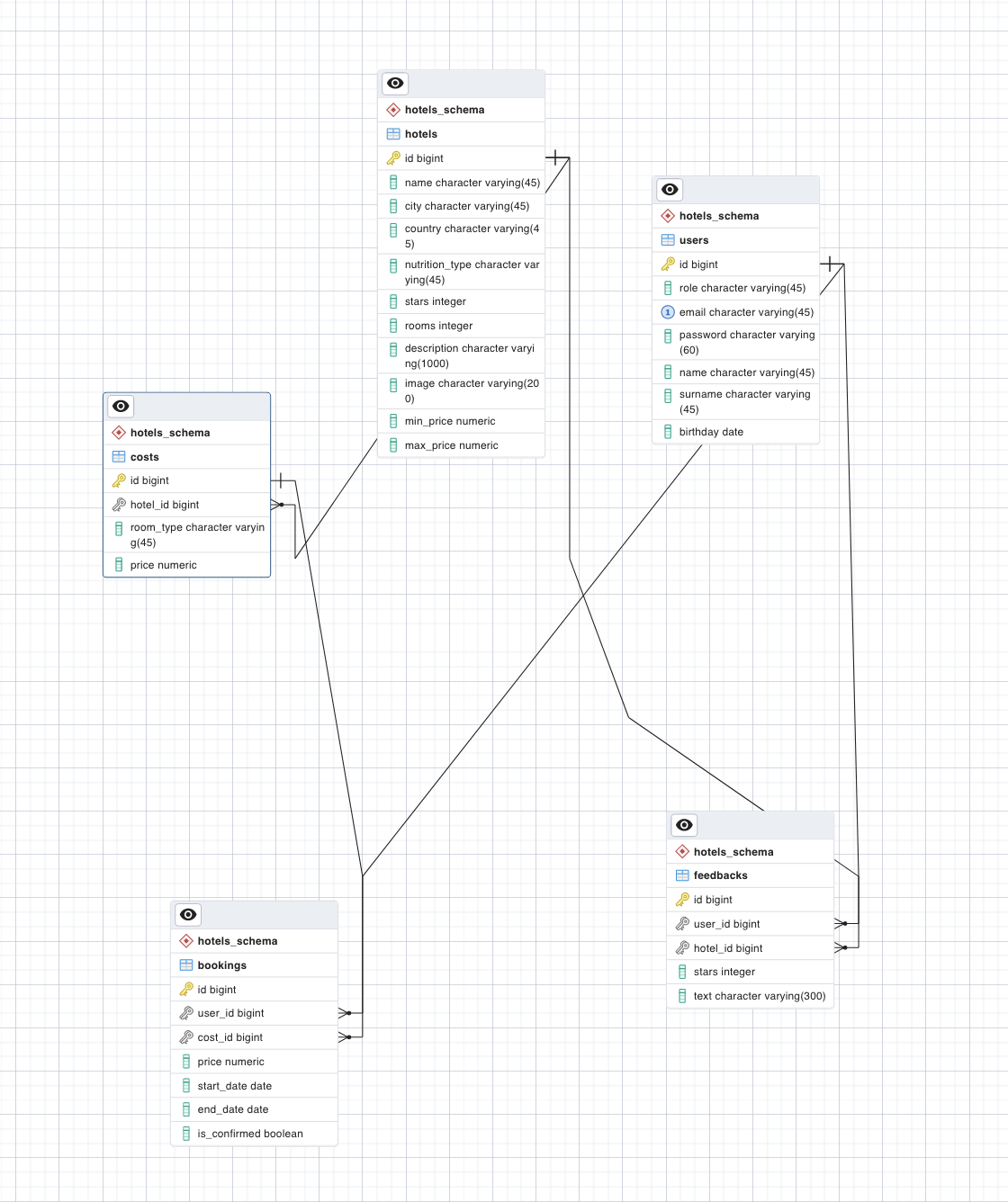


Рисунок 4.1 - Информационная модель базы данных

Перейдём к рассмотрению сущностей, используемых в курсовом проекте.

Сущность «hotels» хранит всю информацию о представленных гостиницах, с которыми пользователь может ознакомиться на сайте. Атрибуты сущности «hotels»:

* id (тип данных bigint) – первичный ключ и уникальный идентификатор гостиницы;
* name (тип данных character varying) – название гостиницы;
* city (тип данных character varying) – город нахождения гостиницы.
* country (тип данных character varying) – страна нахождения гостиницы.
* nutrition\_type (тип данных character varying) – организация питания в гостинице;
* stars (тип данных integer) – рейтинг гостиниц, представленный на сайте;
* rooms (тип данных integer) – категории номеров в гостинице;
* description (тип данных character varying) – описание гостиницы;
* image (тип данных character varying) – фотографии гостиницы;
* min\_price (тип данных numeric) – минимальная цена проживания;
* max\_price (тип данных numeric) – максимальная цена проживания;

Сущность «users» хранит информацию об учетных записях авторизовавшихся пользователей. Атрибуты сущности «users»:

* id (тип данных bigint) – первичный ключ и уникальный идентификатор пользователя;
* role (тип данных character varying) – роль пользователя;
* email (тип данных character varying) – электронный почтовый ящик пользователя;
* password (тип данных character varying) – пароль пользователя;
* name (тип данных character varying) – имя пользователя;
* surname (тип данных character varying) – фамилия пользователя;
* birthday (тип данных date) – дата рождения пользователя;

Сущность «costs» хранит информацию о тарифах на проживание для разных комнат и гостиниц. Атрибуты сущности «costs»:

* id (тип данных bigint) – первичный ключ и уникальный идентификатор стоимости;
* hotels\_id (тип данных bigint) – уникальный идентификатор гостиницы;
* rooms\_type (тип данных character varying) – тип комнаты;
* price (тип данных numeric) – цена;

Сущность «booking» хранит информацию о поступивших бронях и их состояниях. Атрибуты сущности «bookings»:

* id (тип данных bigint) – первичный ключ и уникальный идентификатор брони;
* user\_id (тип данных bigint) – уникальный идентификатор пользователя;
* cost\_id (тип данных bigint) – уникальный идентификатор стоимости;
* price (тип данных numeric) – цена;
* start\_date (тип данных date) – дата заезда;
* end\_date (тип данных date) – дата выезда;
* is\_confirmed (тип данных boolean) – подтверждение брони;

Сущность «feedbacks» хранит отзывы, оставленные проживающими. Атрибуты сущности «feedbacks»:

* id (тип данных bigint) – первичный ключ и уникальный идентификатор брони;
* user\_id (тип данных bigint) – уникальный идентификатор пользователя;
* hotels\_id (тип данных bigint) – уникальный идентификатор гостиницы;
* stars (тип данных integer) – рейтинг, выставляемый пользователем;
* text (тип данных character varying) – сообщение пользователя о гостинице (отзыв);

Ниже определим форму базы данных.

Первая нормальная форма. Устранение повторяющиеся группы в отдельных таблицах. Создание отдельной таблицы для каждого набора связанных данных. Идентификация каждого набор связанных данных с помощью первичного ключа.

Вторая нормальная форма. Создание отдельных таблиц для наборов значений, относящихся к нескольким записям. Связь этих таблиц с помощью внешнего ключа. Записи могут зависеть только от первичного ключа таблицы.

Третья нормальная форма. Устранение полей, не зависящих от ключа. Чтобы нормализовать базу данных до третьей нормальной формы, необходимо сделать так, чтобы в таблицах отсутствовали неключевые столбцы, которые зависят от других неключевых столбцов.

Неключевые столбцы не должны пытаться играть роль ключа в таблице, т.е. они действительно должны быть неключевыми столбцами, такие столбцы не дают возможности получить данные из других столбцов, они дают возможность посмотреть на информацию, которая в них содержится, так как в этом их назначение.

Созданная модель является третьей нормальной формой.

Нормализация не является обязательной, но приносит следующие преимущества:

1. Упрощается процесс выборки**.** Речь идет об упрощении работы по составлению запросов, то есть пользователь сможет получать нужную информацию относительно простыми запросами;
2. Обеспечивается целостность данных. Можно говорить о минимизации искажения информации и снижении вероятности потери данных;
3. Улучшается масштабируемость**.** При соблюдении правил нормализации формируются благоприятные предпосылки к росту БД;
4. Отсутствует избыточность (data redundancy). Избыточность —проблема непродуктивного использования свободного места на жестком диске, затрудняющая обслуживание БД.
5. Отсутствие несогласованных зависимостей. Несогласованные зависимости затрудняют доступ к данным, ведь путь к такой информации может быть неправилен и нелогичен.
6. Описание алгоритмов, реализующих бизнес-логику автоматизированной системы БРОНИРОВАНИЯ НОМЕРОВ В ГОСТИНИЦЕ

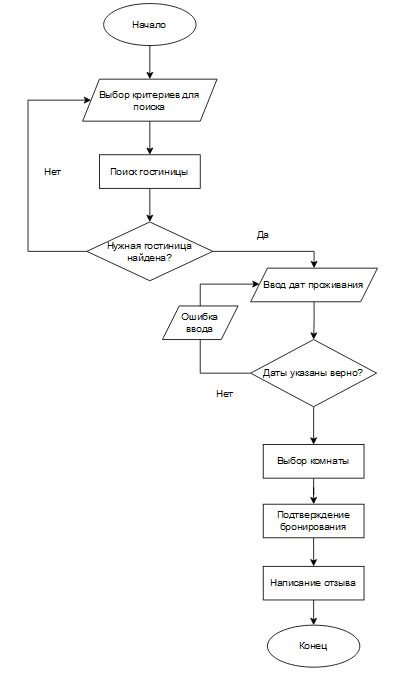
Алгоритм программы – это последовательность инструкций, которые выполняются компьютером для достижения определенной цели.

Описание алгоритма программы необходимо для того, чтобы разработчик мог лучше понять, как программа должна работать, и чтобы другие разработчики могли прочитать и понять, как программа работает. Описание алгоритма также может помочь в тестировании программы и устранении ошибок.

Бизнес-логика системы – это набор правил, процессов и операций, которые определяют, как система должна работать для достижения бизнес-целей организации. Она определяет, какие данные должны быть обработаны и как они должны быть обработаны для выполнения задач, связанных с бизнес-процессами. Бизнес-логика системы может быть описана в виде диаграмм, таблиц, текстовых описаний и других форматов. Она является ключевым компонентом при разработке и внедрении информационных систем в организации.

Опишем последовательность действий для построения алгоритма. После начала запуска алгоритма идет выбор критериев для поиска пользователем. Клиент должен выбрать страну проживания, город и рейтинг гостиницы по которым будет производиться поиск. Если нужная гостиница не найдена, пользователь возвращается к выбору критериев. Если нужная гостиница найдена, то далее необходимо выбрать даты заезда и выезда. Даты проверяются на правильность ввода (дата заезда должна быть раньше, чем дата выезда, и выставляться в настоящем или будущем времени по автоматическому календарю). Если пользователь указал неверные данные, система сообщает об ошибке. После выбора дат пользователю предлагается выбрать комнату, затем администратор подтверждает бронь, после чего открывается возможность написать отзыв.

Рассмотрим графическое представление основного бизнес-процесса для нашего веб-приложения. На рисунке 5.1 представлена блок-схема процесса бронирования номера в гостинице пользователем.

Рисунок 5.1 – Блок-схема процесса бронирования номера в гостинице.

В данной главе описаны ключевые алгоритмы, которые представляют бизнес-логику разрабатываемой автоматизированной системы бронирования номеров в гостинице.

1. Обоснование выбора технологий, подходов для программной реализации Автоматизированной системы бронирования номеров в гостинице

Разрабатываемая в курсовом проекте автоматизированная система должна осуществлять бронь гостиничных номеров.

Согласно требованию на курсовой проект, серверная часть автоматизированной системы должна быть построена на основе набора спецификаций Java EE или фреймворка Spring.

В проекте были выбраны инструменты, предоставляемые фреймворком Spring, так как он широко используется для создания enterprise-приложений.

Spring – это популярный фреймворк для разработки на Java, который используют для создания корпоративных приложений, например CRM. П рименять его могут не только Java-разработчики, но и те, кто работает с Kotlin или Groovy.

Spring Boot – это технология, которая позволяет упростить и ускорить процесс разработки приложений на Java. Она основана на Spring Framework и предоставляет инструменты для автоматической конфигурации и управления зависимостями. Сам Spring Boot представляет собой набор утилит, автоматизирующих настройки фреймворка. Вот что он берёт на себя:

* упаковывает зависимости в starter-пакеты;
* автоматически конфигурирует приложения с помощью jar-зависимостей;
* создаёт веб-сервер, что позволяет локально запускать на нём приложения.

Встроенный в Spring Boot сервер Apache Tomcat работает в качестве веб-сервера и прослушивает запросы на порту localhost 8080.

Для реализации автоматизированной системы бронирования гостиничных номеров на базе Spring Boot можно использовать следующие технологии:

Spring MVC - это фреймворк для разработки веб-приложений на Java. Он позволяет быстро создавать контроллеры, сервлеты, представления и работать с данными.

Spring Data JPA - это инструмент для работы с данными на Java. Он позволяет работать с базами данных через объектно-реляционное отображение (ORM) и генерировать SQL-запросы автоматически.

Thymeleaf - это движок шаблонов для веб-приложений на Java. Он позволяет создавать динамические страницы, включая данные из модели и контроллера.

В качестве основной базы данных была использована программа Postgres SQL. В первую очередь Postgres SQL является одной из самых надежных и стабильных СУБД, что позволяет избежать потери данных и сбоев в работе системы, может легко масштабироваться для работы с большими объемами данных и высокой нагрузкой, поддерживает множество расширений и плагинов, что позволяет адаптировать базу данных под конкретные потребности и требования проекта, обладает высоким уровнем безопасности и поддерживает множество механизмов защиты данных, таких как шифрование, аутентификация и авторизация.

Postgres SQL является открытым и бесплатным программным обеспечением, что позволяет сократить затраты на разработку и поддержку системы.

Для тестирования серверной части автоматизированной системы контроля экспорта товаров использован фреймворк Junit, так как он является частью проекта spring-boot-starter-test.

Для тестирования клиентской части использован фреймворк Cypress, который предоставляет возможности сквозного тестирования (подробная эмуляция пользовательской среды), а также имеет преимущество в асинхронном тестировании перед иными фреймворками.

При разработке автоматизированной системы контроля экспорта товаров использована СУБД PostgreSQL, так как она является оптимальным решением для построения сложных систем с большим числом запросов.

1. руководство пользователя автоматизированной системы бронирования номеров в гостинице
   1. Инструкция по развертыванию автоматизированной системы бронирования номеров в гостинице

Для возможности корректного использования системы требуется наличие на компьютере 32х или 64ч разрядной операционной системы Windows 7 и выше или Mac OS X 10.6 и выше. Данный программный продукт использует PostgreSQL для работы с базой данных. Также необходимо наличие на компьютере для клиента и сервера JDK версии 17 и выше.

Для запуска приложения необходимо запустить из консоли AppInitializer.java. При первом запуске приложения будет создана база данных, если до этого не была создана, то она сгенерируется автоматически с помощью SQL-скрипта, который находится в приложении В.

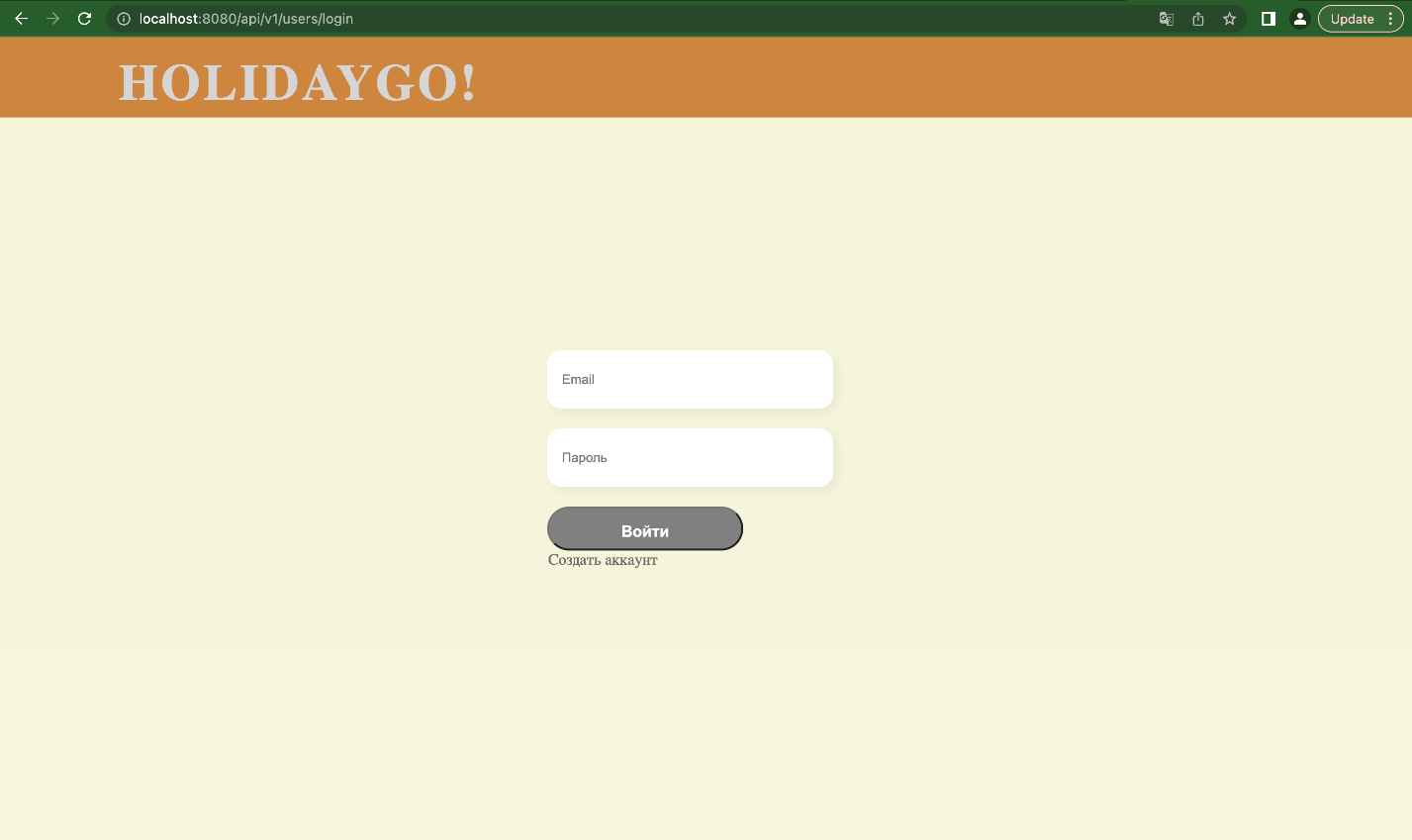
В результате описанных выше действий пользователь может корректно запустить данное программное приложение на любом удовлетворяющем требованиям устройстве.

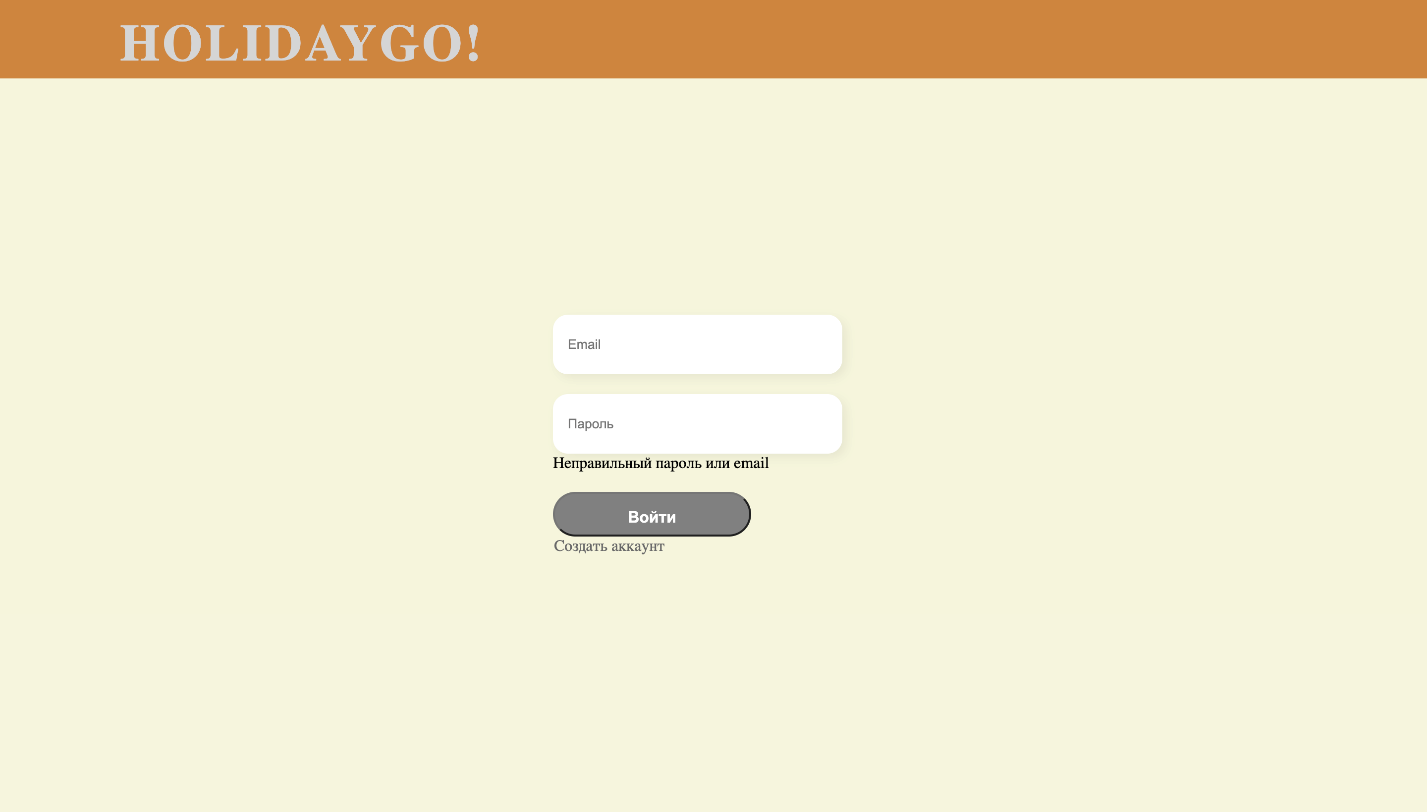
С помощью Docker, программного обеспечения для автоматизации развертывания, можно развернуть приложение на любом устройстве, где установлен Docker. Для полного развертывания приложения на сервере нужно выполнить команду «docker compose up» в консоли. Docker установит все зависимости и создаст базу данных, необходимую для приложения. После этого приложение будет готово к использованию.

Этот руководство поможет развернуть автоматизированную систему бронирования в Docker.

1. Установите Docker на вашей машине, используя инструкции на сайте Docker.
2. Скачайте и распакуйте файлы системы бронирования из источника. Система бронирования должна содержать все необходимые файлы, включая Dockerfile.
3. Откройте терминал и перейдите в папку с файлами системы бронирования.
4. Создайте образ Docker используя команду: docker build -t booking\_system.
5. Запустите контейнер на основе образа командой: docker run -p 8080:80 booking\_system. Эта команда запускает контейнер и привязывает его к порту 8080 на локальной машине. Вы можете изменить порт на свой выбор.
6. Перейдите по адресу http://localhost:8080 в вашем браузере для того, чтобы убедиться, что система бронирования работает нормально.
   1. Руководство пользователя автоматизированной системы бронирования гостиничных номеров

После выполнения всех этапов развертывания программы перед клиентом приложения появляется страница авторизации пользователя (рисунок 8.1), которая запрашивает электронную почту и пароль, а также дает переход в другую форму по кнопке «Создать аккаунт» для неавторизованных пользователей. Для избегания системных ошибок выполняется проверка на корректность введения электронного почтового адреса (Правильная валидация email: *@*). Система выдает ошибку, если пользователь введет адрес неправильно. (рисунок 8.2)

Рисунок 8.1 - Форма входа в систему

Рисунок 8.2 – Ошибка авторизации

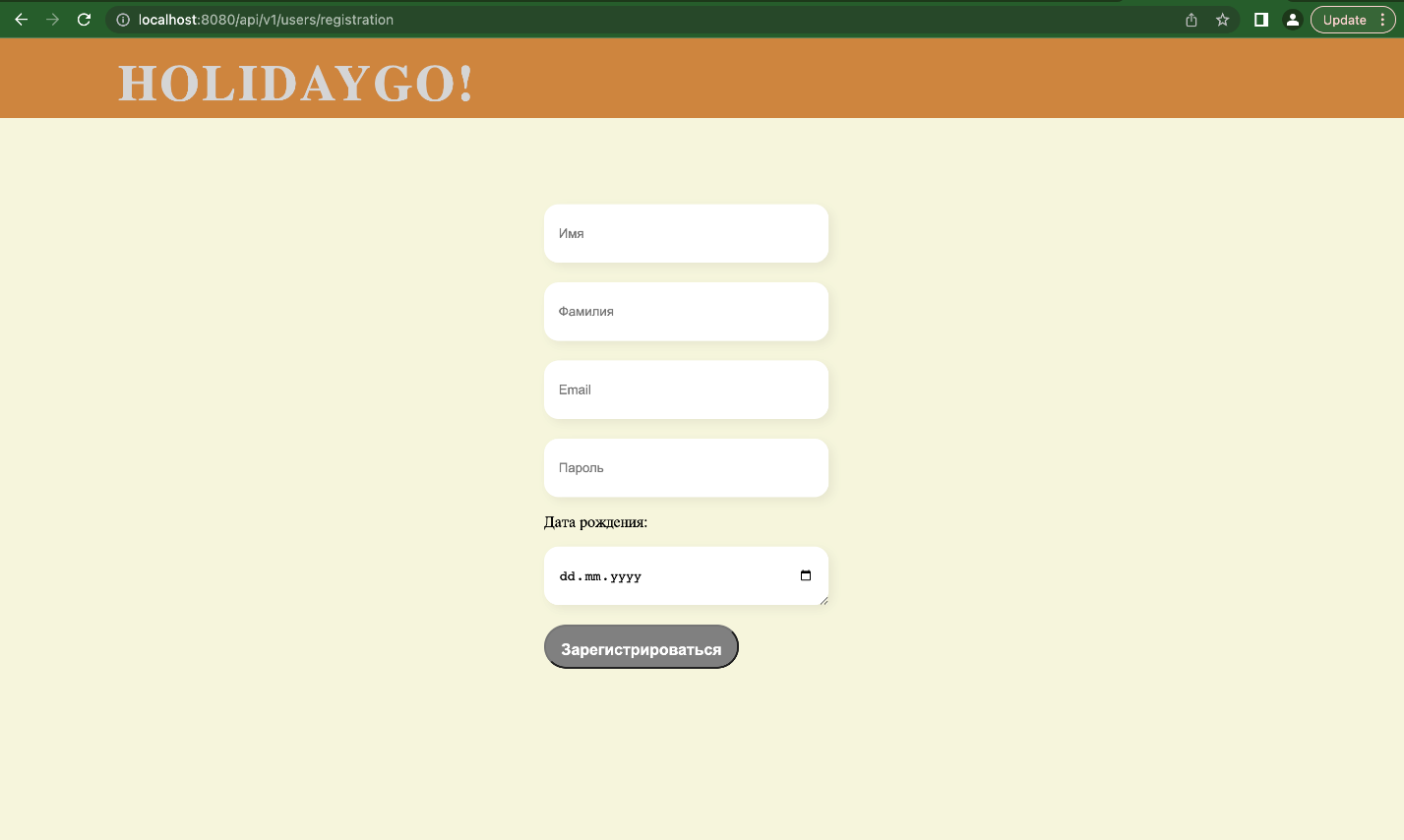
На рисунке 8.3 показана форма регистрации. Новый пользователь должен ввести имя, фамилию, электронный почтовый адрес, пароль и дату своего рождения.

Рисунок 8.3 – Форма регистрации нового пользователя

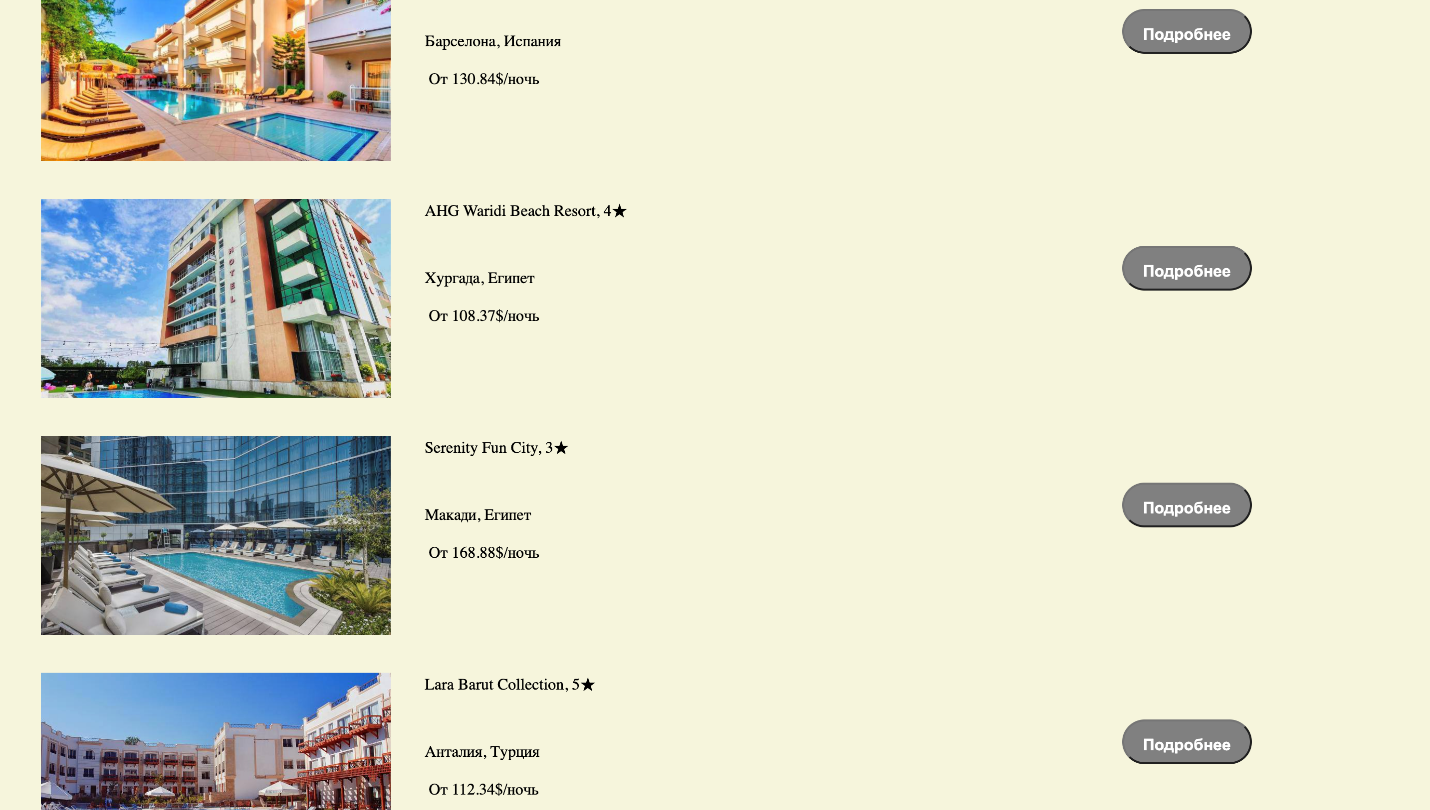
После успешного входа в систему клиенту становится доступным просмотр его бронирований. Также ниже представлен фильтр для выбора гостиницы по критериям: страна, город, рейтинг, дата заезда/дата выезда. При прокрутке страницы можно обнаружить список всех представленных гостиниц, где есть фотографии и указаны минимальный тариф на проживание, рейтинг, страна и город.

Рисунок 8.4 – Список всех доступных гостиниц

Также доступен просмотр каждой гостиницы в отдельности (рисунок 8.5), где можно узнать больше информации (включено ли питание, какие достопримечательности есть рядом).

Рисунок 8.5 – Информация о гостинице на отдельной странице

После выбора гостиницы пользователь должен перейти в форму бронирования (рисунок 8.6). Здесь пользователь может выбрать категорию номера в соответствии со стоимостью, определиться с датой заезда (выбрать дату нужно корректно, чтобы дата заезда была раньше, чем дата выезда). После правильно введённых данных запрос клиента отправляется администратору.

Рисунок 8.6 – Форма бронирования номера в гостинице

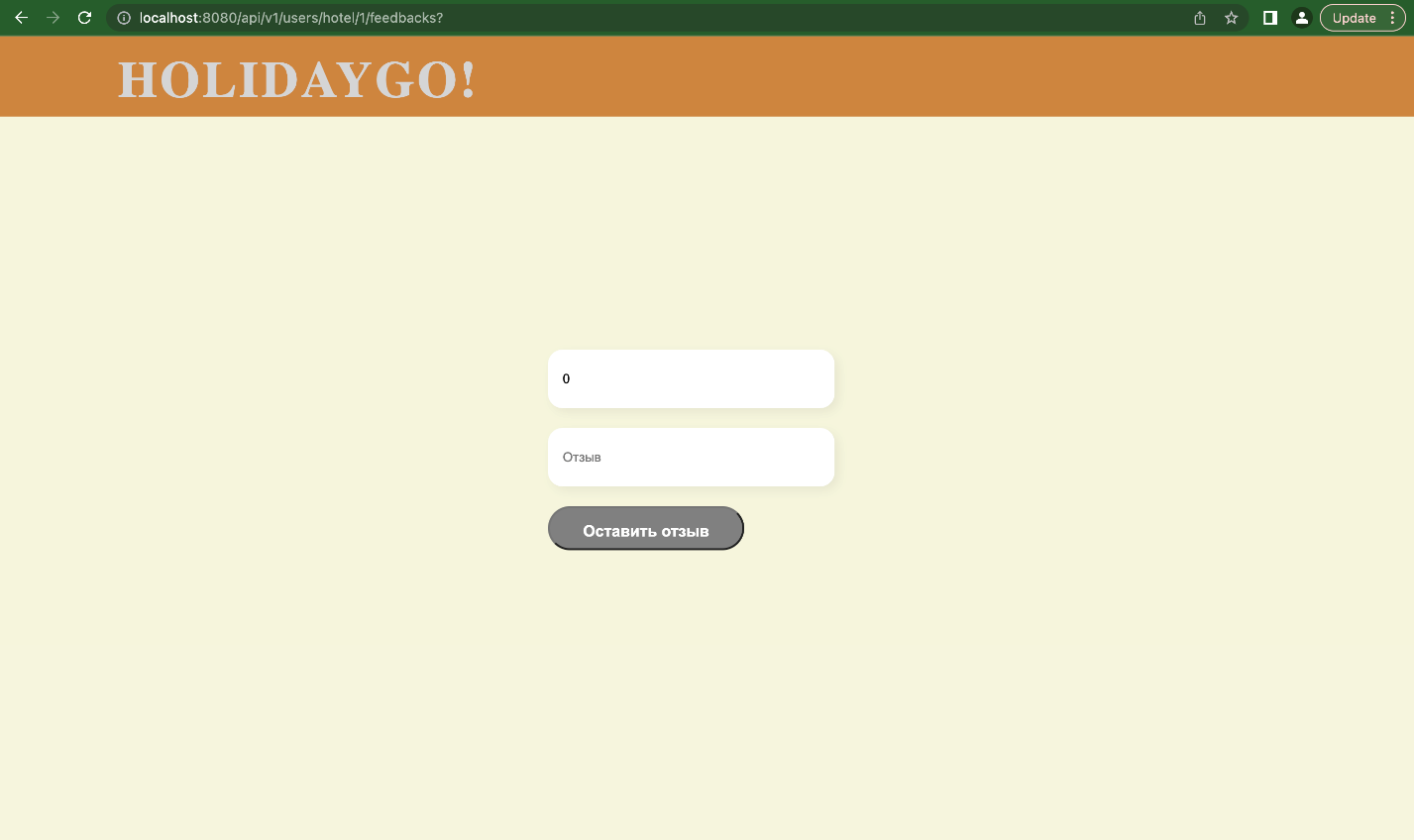
После подтверждения брони со стороны администратора, пользователю открывается возможность оставить свой текстовый отзыв о гостинице и выставить рейтинг.

Рисунок 8.7 – Форма для отзывов.

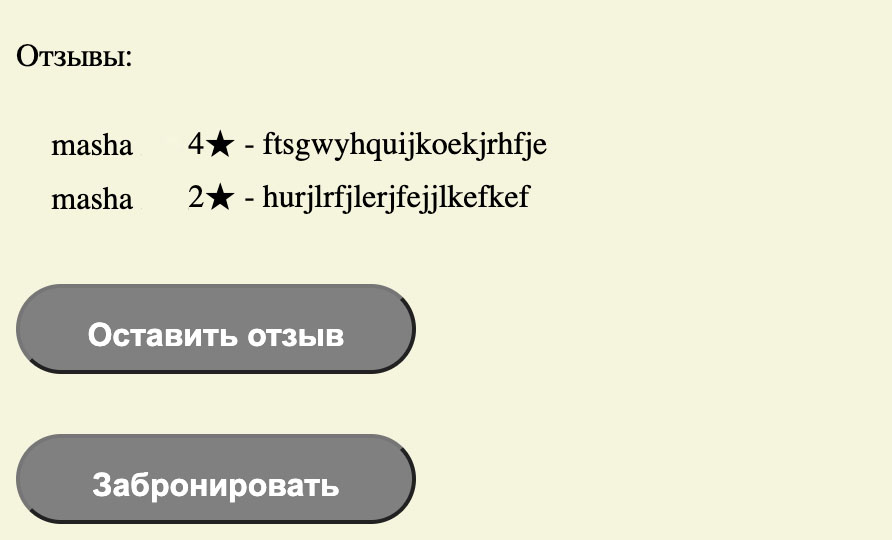
Гость также может просматривать отзывы, оставленные другими пользователями, также эта форма предлагает вновь забронировать номер в выбранной им гостинице (рисунок 8.8).

Рисунок 8.8 – Форма просмотра отзывов

Пользователи с ролью «Администратора» имеют те же функции, что и другие пользователи. Однако, администратор имеет возможность управления входящими заявками на бронирование: администратор может подтвердить бронь (рисунок 8.9).

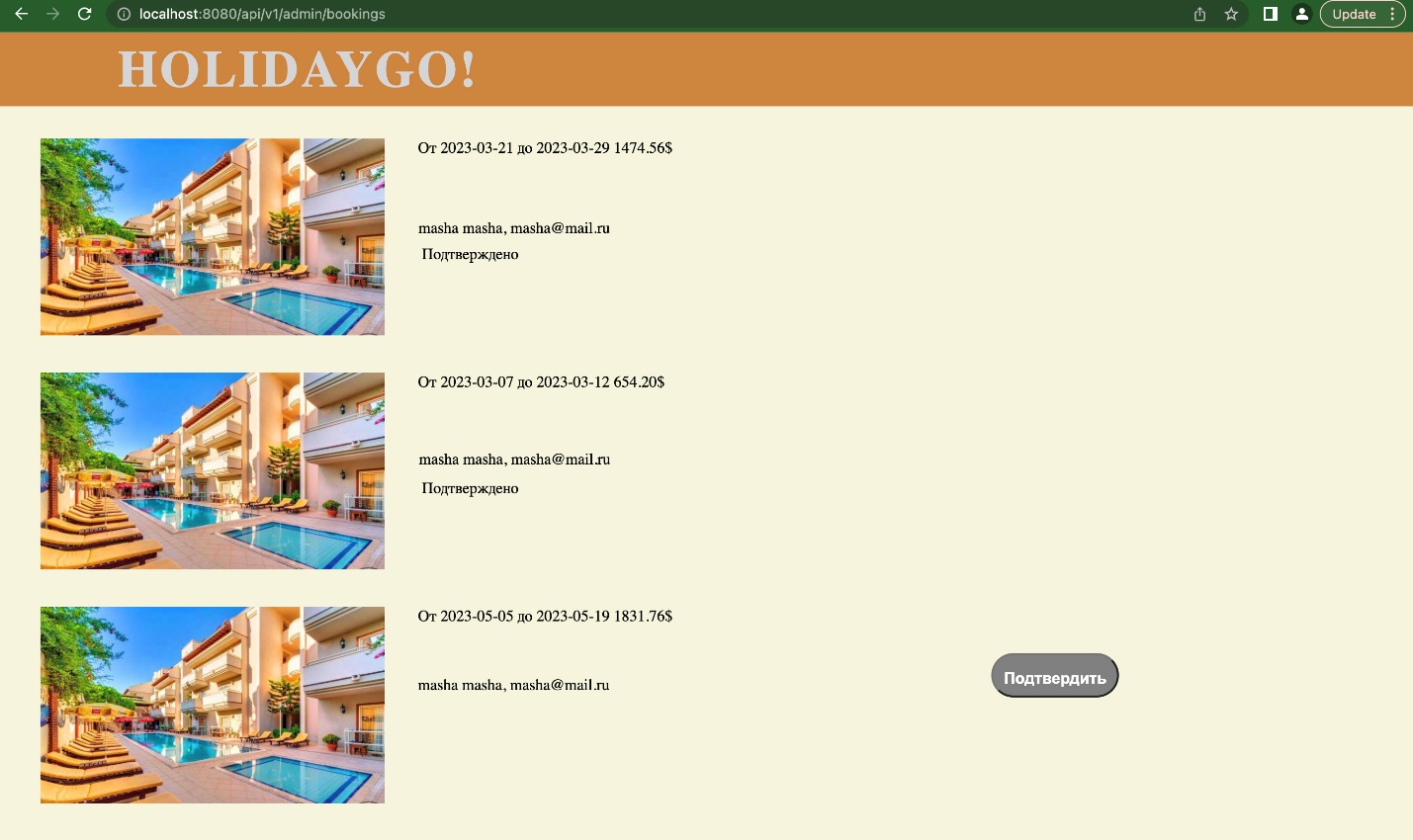


Рисунок 8.9 – Форма подтверждения брони.

По результатам работы тестирования программного продукта, видно, что поставленные задачи решены в полном объеме. Разработанная система позволит пользователям регистрироваться и авторизоваться в системе, изменять свои личные данные, бронировать авиабилеты, добавлять новые и анализировать данные. Данное приложение облегчит работу гостиницам и сделает возможность быстрого и легкого бронирования для пользователей

Заключение

Согласно заданию, целью курсового проекта является совершенствование организации по оказанию услуг бронирования в интернете и повышению качества обслуживания пользователей.

Поставленная цель потребовала решения следующих задач:

* анализ деятельности по осуществлению бронирований в интеренете;
* определение ключевых требований, предъявляемых к автоматизированной системе, а также способов их исполнения;
* моделирование представлений автоматизированной системы;
* проектирование информационной модели, хранящей данные о пользователях и гостиницах;
* описание алгоритмов, реализующих ключевые бизнес-задачи автоматизированной системы;
* предоставление пользователю подробного руководства по взаимодействию с автоматизированной системой, включающего инструкцию по развертыванию, а также руководство пользователя;

Для разработки автоматизированной системы были изучены особенности организации экспорта товаров. Результатом изучения организации стала диаграмма IDEF0, состоящая из контекстной диаграммы и декомпозиций. Создание диаграммы было осуществлено в программном продукте AllFusion Process Modeler.

Изученная организация бронирования номеров в гостинице позволила разработать базу данных под управлением СУБД PostgreSQL, хранящую данные о совершённых экспортных поставках. Проектирование информационной модели осуществлено в программном продукте pgAdmin4.

Для разработки автоматизированный системы были смоделированы алгоритмы, реализующие ключевую бизнес-логику системы. Моделирование алгоритмов выполнено на онлайн-платформе: https://app.diagrams.net/.

Средствами фреймворка Spring разработано приложение, исполняемое на стороне сервера. При разработке использованы такие проекты как Spring Boot, Spring Data, Spring Security.

Предоставлена инструкция по развертыванию автоматизированной системы контроля экспорта товаров на платформе Docker. Для конечного пользователя продемонстрированы примеры реализации функционала.

В результате работы над курсовым проектом была разработана автоматизированная система бронирования номеров в гостинице, которая имеет следующие особенности:

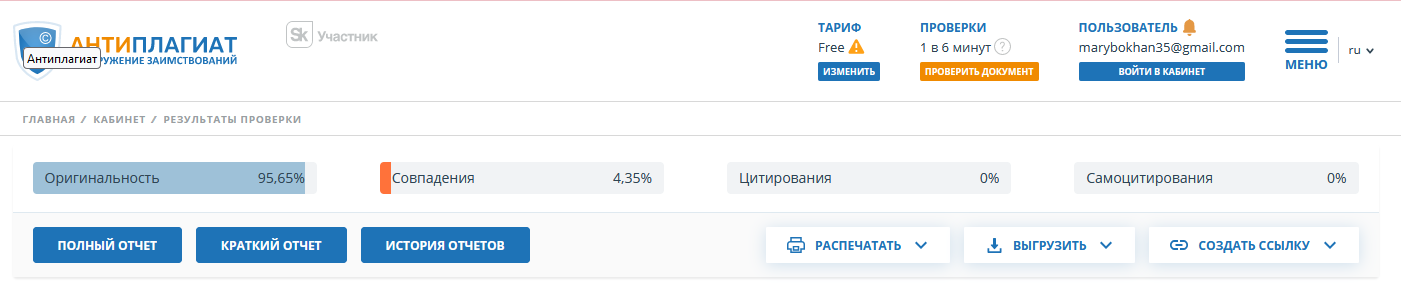
* удобный для пользователя интерфейс SPA-приложения;
* возможность локализации системы;
* широкий спектр возможностей администрирования автоматизированной системы;
* масштабируемость.

По результатам проверки системой «Антиплагиат» на заимствования выявлено, что оригинальность автоматизированной системы бронирования номеров в гостинице составляет 95,65%, что является приемлемым результатом.

Список использованных источников

1. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2006. – 496 с.
2. Диаграммы классов [Электронный ресурс]. – Режим доступа : <https://flexberry.github.io/ru/gpg_class-diagram.html>
3. Диаграммы последовательности [Электронный ресурс]. – Режим доступа : https://studizba.com /lectures/10 -informatika-i-programmirovanie /368 -sovremennye-tehnologiiprogrammirovaniya /4999-7-diagrammy-posledovatelnosti.html
4. Диаграммы развертывания [Электронный ресурс]. – Режим доступа : <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/>
5. Комличенко, В. Н. Компьютерные сети : лаб. практикум. – Минск : БГУИР, 2013. – 76 с.
6. Леоненков, А. В. Самоучитель UML 2 / А. В. Леоненков. – СПб. : БХВ-Петербург, 2007. – 576 с.
7. Международный обмен товарами и услугами [Электронный ресурс]. – Режим доступа : <https://mgimo.ru/upload/iblock/6ee/6eeb72cc378a24f030b0376e988f6fe8.pdf>
8. Тиленс, Т. М. React в действии / Т. М. Тиленс. – СПб. : Питер, 2019. – 368 с.
9. Уоллс, К. Spring в действии / К. Уоллс. – М. : ДМК, 2013. – 750 с.
10. Черемных, С. В. Моделирование и анализ систем. IDEF-технологии / С. В. Черемных. – М. : Финансы и статистика, 2006. – 188 с.
11. Экспорт и импорт товаров в мировой торговле [Электронный ресурс]. – Режим доступа : <https://www.grandars.ru/student/mirovaya-ekonomika/eksport-i-import.html>

Приложение А  
(обязательное)  
Отчет о проверке на заимствования в системе  
 «Антиплагиат»



Приложение Б  
(обязательное)  
Листинг кода алгоритмов, реализующих основную  
бизнес-логику

**BookingServiceImpl.java**

package com.app.hotels.service.impl;

import com.app.hotels.domain.Booking;

import com.app.hotels.domain.exception.DateNullPointerException;

import com.app.hotels.domain.exception.IllegalDateDurationException;

import com.app.hotels.domain.exception.ResourceDoesNotExistException;

import com.app.hotels.repository.BookingRepository;

import com.app.hotels.service.BookingService;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import java.math.BigDecimal;

import java.time.Duration;

import java.util.List;

import java.util.Optional;

@Service

@RequiredArgsConstructor

public class BookingServiceImpl implements BookingService {

private final BookingRepository bookingRepository;

@Override

public Booking create(Booking booking) {

if (booking.getEndDate() == null || booking.getStartDate() == null ){

throw new DateNullPointerException("Введите дату!");

}

booking.setConfirmed(false);

long dayAmount = Duration.between(booking.getStartDate().atStartOfDay(), booking.getEndDate().atStartOfDay()).toDays();

if (dayAmount<=0) throw new IllegalDateDurationException("Выберите верные даты");

BigDecimal price = booking.getCost().getPrice().multiply(BigDecimal.valueOf(dayAmount));

booking.setPrice(price);

return bookingRepository.save(booking);

}

@Override

public Booking update(Booking booking) {

return bookingRepository.save(booking);

}

@Override

public void delete(Long id) {

bookingRepository.deleteById(id);

}

@Override

public List<Booking> findAllByUserId(Long id) {

return bookingRepository.findAllByUserId(id);

}

@Override

public List<Booking> findAll() {

return bookingRepository.findAll();

}

@Override

public Booking confirm(Long id) {

Optional<Booking> booking = bookingRepository.findById(id);

if (booking.isEmpty()) {

throw new ResourceDoesNotExistException("Не существует бронирования с id " + id);

}

Booking bookingExisted = booking.get();

bookingExisted.setConfirmed(true);

return bookingRepository.save(bookingExisted);

}

@Override

public Booking findById(Long id) {

return bookingRepository.findById(id).get();

}

}

**HotelServiceImpl.java**

package com.app.hotels.service.impl;

import com.app.hotels.domain.Hotel;

import com.app.hotels.domain.criteria.HotelCriteria;

import com.app.hotels.domain.exception.ResourceDoesNotExistException;

import com.app.hotels.repository.HotelRepository;

import com.app.hotels.service.HotelService;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceContext;

import jakarta.persistence.criteria.\*;

import lombok.RequiredArgsConstructor;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.List;

@Service

@RequiredArgsConstructor

public class HotelServiceImpl implements HotelService {

private static final int PAGE\_SIZE = 20;

private final HotelRepository hotelRepository;

@PersistenceContext

private EntityManager entityManager;

@Override

public List<Hotel> findAll(HotelCriteria hotelCriteria, int currentPage) {

List<Hotel> hotels;

if(currentPage == 0){

currentPage = 0;

}

if(hotelCriteria.getCities() != null || hotelCriteria.getStars() != null || hotelCriteria.getMinPrice() != null ||

hotelCriteria.getMaxPrice() != null || hotelCriteria.getCountries() != null) {

hotels = findAllByCriteria(hotelCriteria, currentPage);

} else {

CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();

CriteriaQuery<Hotel> criteriaQuery = criteriaBuilder.createQuery(Hotel.class);

Root<Hotel> hotelRoot = criteriaQuery.from(Hotel.class);

Order hotelOrder = criteriaBuilder.asc(hotelRoot.get("id"));

criteriaQuery.orderBy(hotelOrder);

Pageable pageable = PageRequest.of(currentPage, PAGE\_SIZE);

hotels = entityManager.createQuery(criteriaQuery)

.setFirstResult((int) pageable.getOffset())

.setMaxResults(pageable.getPageSize())

.getResultList();

}

return hotels;

}

public List<Hotel> findAllByCriteria(HotelCriteria hotelCriteria, int currentPage){

CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();

CriteriaQuery<Hotel> criteriaQuery = criteriaBuilder.createQuery(Hotel.class);

Root<Hotel> hotelRoot = criteriaQuery.from(Hotel.class);

List<Predicate> predicates = new ArrayList<>();

List<String> cities = hotelCriteria.getCities();

List<Predicate> cityPredicates = new ArrayList<>();

if (cities != null && !cities.isEmpty()) {

for (String city : cities) {

cityPredicates.add(criteriaBuilder.equal(hotelRoot.get("city"), city));

}

Predicate cityFinalPredicate = criteriaBuilder.or(cityPredicates.toArray(new Predicate[0]));

predicates.add(cityFinalPredicate);

}

List<String> countries = hotelCriteria.getCountries();

List<Predicate> countryPredicates = new ArrayList<>();

if (countries != null && !countries.isEmpty()) {

for (String country : countries) {

countryPredicates.add(criteriaBuilder.equal(hotelRoot.get("country"), country));

}

Predicate countryFinalPredicate = criteriaBuilder.or(countryPredicates.toArray(new Predicate[0]));

predicates.add(countryFinalPredicate);

}

List<Integer> stars = hotelCriteria.getStars();

List<Predicate> starsPredicates = new ArrayList<>();

if(stars != null && !stars.isEmpty()){

for(int star: stars){

starsPredicates.add(criteriaBuilder.equal(hotelRoot.get("stars"), star));

}

Predicate starsFinalPredicate = criteriaBuilder.or(starsPredicates.toArray(new Predicate[0]));

predicates.add(starsFinalPredicate);

}

BigDecimal minPrice = hotelCriteria.getMinPrice();

if (minPrice != null) {

Predicate minPricePredicate = criteriaBuilder.greaterThanOrEqualTo(hotelRoot.get("maxPrice"), minPrice);

predicates.add(minPricePredicate);

}

BigDecimal maxPrice = hotelCriteria.getMaxPrice();

if (maxPrice != null) {

Predicate maxPricePredicate = criteriaBuilder.lessThanOrEqualTo(hotelRoot.get("minPrice"), maxPrice);

predicates.add(maxPricePredicate);

}

Predicate finalPredicate = criteriaBuilder.and(predicates.toArray(new Predicate[0]));

criteriaQuery.where(finalPredicate);

Pageable pageable = PageRequest.of(currentPage, PAGE\_SIZE);

List<Hotel> result = entityManager.createQuery(criteriaQuery)

.setFirstResult((int) pageable.getOffset())

.setMaxResults(pageable.getPageSize())

.getResultList();

return result;

}

@Override

public Hotel findById(Long id) {

return hotelRepository.findById(id)

.orElseThrow(() -> new ResourceDoesNotExistException("Отель с id " + id + " не существует"));

}

@Override

public Hotel create(Hotel hotel) {

return hotelRepository.save(hotel);

}

@Override

public Hotel update(Hotel hotel) {

return hotelRepository.save(hotel);

}

@Override

public void delete(Long id) {

hotelRepository.deleteById(id);

}

@Override

public List<String> findAllCountries() {

return hotelRepository.findAllCountries();

}

@Override

public List<String> findAllCities() {

return hotelRepository.findAllCities();

}

@Override

public List<Integer> findAllStars() {

return hotelRepository.findAllStars();

}

}

**UserServiceImpl.java**

package com.app.hotels.service.impl;

import com.app.hotels.domain.User;

import com.app.hotels.domain.exception.ResourceAlreadyExistsException;

import com.app.hotels.domain.exception.ResourceDoesNotExistException;

import com.app.hotels.repository.UserRepository;

import com.app.hotels.service.UserService;

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.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service

public class UserServiceImpl implements UserService, UserDetailsService {

@Autowired

private UserRepository userRepository;

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

@Override

@Transactional

public User create(User user) {

if (userRepository.existsByEmail(user.getEmail())) {

throw new ResourceAlreadyExistsException("Пользователь с email " + user.getEmail() + " уже существует");

}

user.setRole(User.Role.ROLE\_USER);

user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));

userRepository.save(user);

return user;

}

@Override

public User findByEmail(String email) {

return userRepository.findByEmail(email)

.orElseThrow(() -> new ResourceDoesNotExistException("Не существует пользователя с email " + email));

}

@Override

public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

return findByEmail(email);

}

}

**FeedbackServiceImpl.java**

package com.app.hotels.service.impl;

import com.app.hotels.domain.Feedback;

import com.app.hotels.repository.FeedbackRepository;

import com.app.hotels.service.FeedbackService;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

@RequiredArgsConstructor

public class FeedbackServiceImpl implements FeedbackService {

private final FeedbackRepository feedbackRepository;

@Override

public Feedback create(Feedback feedback) {

return feedbackRepository.save(feedback);

}

@Override

public List<Feedback> findAllByHotelId(Long id) {

return feedbackRepository.findAllByHotelId(id);

}

}

**СostServiceImpl.java**

package com.app.hotels.service.impl;

import com.app.hotels.domain.Cost;

import com.app.hotels.repository.CostRepository;

import com.app.hotels.service.CostService;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

@RequiredArgsConstructor

public class CostServiceImpl implements CostService {

private final CostRepository costRepository;

@Override

public List<Cost> findAllByHotelId(Long id) {

return costRepository.findAllByHotelId(id);

}

@Override

public Cost create(Cost cost) {

return costRepository.save(cost);

}

@Override

public Cost findById(Long id) {

return costRepository.findById(id).get();

}

}

**UserController.java**

package com.app.hotels.web.controller;

import com.app.hotels.domain.\*;

import com.app.hotels.domain.criteria.HotelCriteria;

import com.app.hotels.service.\*;

import com.app.hotels.web.dto.BookingDto;

import com.app.hotels.web.dto.FeedbackDto;

import com.app.hotels.web.dto.UserDto;

import com.app.hotels.web.mapper.\*;

import jakarta.validation.Valid;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.\*;

import java.security.Principal;

import java.util.List;

@Controller

@RequestMapping("/api/v1/users")

@RequiredArgsConstructor

public class UserController {

private final UserService userService;

private final UserMapper userMapper;

private final HotelService hotelService;

private final HotelMapper hotelMapper;

private final CostService costService;

private final CostMapper costMapper;

private final BookingService bookingService;

private final BookingMapper bookingMapper;

private final FeedbackService feedbackService;

private final FeedbackMapper feedbackMapper;

@GetMapping("/login")

public String login(Model model, @RequestParam(value = "error", required = false) String error) {

if (error != null) {

model.addAttribute("error", "Неправильный пароль или email");

}

return "login";

}

@GetMapping("/registration")

public String registrationPage(Model model) {

UserDto user = new UserDto();

model.addAttribute("user", user);

return "registration";

}

@PostMapping(value = "/registration")

public String registration(@Valid @ModelAttribute("user") UserDto user, BindingResult bindingResult, Model model) {

if (bindingResult.hasErrors()) {

model.addAttribute("user", new UserDto());

return "registration";

}

User userMapped = userMapper.toEntity(user);

userService.create(userMapped);

return "redirect:/api/v1/users/login";

}

@GetMapping("/hotels/{currentPage}")

public String getHotelsPage(Model model, @PathVariable(required = false) int currentPage, @ModelAttribute("hotelCriteria") HotelCriteria hotelCriteria, Principal principal) {

List<Hotel> hotels = hotelService.findAll(hotelCriteria, currentPage);

model.addAttribute("hotels", hotelMapper.toDto(hotels));

List<String> cities = hotelService.findAllCities();

model.addAttribute("cities", cities);

List<String> countries = hotelService.findAllCountries();

model.addAttribute("countries", countries);

List<Integer> stars = hotelService.findAllStars();

model.addAttribute("stars", stars);

model.addAttribute("minCost", 0);

model.addAttribute("maxCost", 1000);

User user = userService.findByEmail(principal.getName());

model.addAttribute("user", user);

return "hotels";

}

@GetMapping("/hotel/{id}")

public String getHotel(Model model, @PathVariable("id") Long id){

Hotel hotel = hotelService.findById(id);

model.addAttribute("hotel", hotelMapper.toDto(hotel));

List<Feedback> feedbacks = feedbackService.findAllByHotelId(id);

model.addAttribute("feedbacks", feedbackMapper.toDto(feedbacks));

return "hotel";

}

@GetMapping("/hotel/{id}/booking")

public String getBookingPage(Model model, @PathVariable("id") Long id){

Hotel hotel = hotelService.findById(id);

model.addAttribute("hotel", hotelMapper.toDto(hotel));

List<Cost> costs = costService.findAllByHotelId(id);

model.addAttribute("costs", costs);

Booking booking = new Booking();

model.addAttribute("booking", booking);

return "createBooking";

}

@PostMapping("/hotel/{hotel}/{hotelId}/booking")

public String createBooking(@Valid @ModelAttribute("booking") Booking booking, BindingResult bindingResult,

Model model, @PathVariable(name = "hotelId") Long hotelId, Principal principal) {

if (bindingResult.hasErrors()){

Hotel hotel = hotelService.findById(hotelId);

model.addAttribute("hotel", hotelMapper.toDto(hotel));

List<Cost> costs = hotel.getCosts();

model.addAttribute("costs", costMapper.toDto(costs));

model.addAttribute("booking");

return "createBooking";

}

User user = userService.findByEmail(principal.getName());

booking.setUser(user);

bookingService.create(booking);

return "redirect:/api/v1/users/hotels/0";

}

@GetMapping("/hotel/bookings/{userId}")

public String getMyBookingPage(Model model, @PathVariable("userId") Long userId){

List<Booking> bookings = bookingService.findAllByUserId(userId);

model.addAttribute("bookings", bookings);

return "myBookings";

}

@PostMapping("/hotel/bookings/{bookingId}")

public String deleteBooking(Model model, @PathVariable("bookingId") Long bookingId){

bookingService.delete(bookingId);

return "hotels";

}

@GetMapping("/hotels/bookings/update/{id}")

public String getUpdateBookingPage(Model model, @PathVariable("id") Long id){

Booking booking = bookingService.findById(id);

model.addAttribute("booking", bookingMapper.toDto(booking));

return "myBookings";

}

@PutMapping("/hotels/booking")

public String updateBooking(@Valid @ModelAttribute("booking") BookingDto booking, BindingResult bindingResult,

Model model, @PathVariable Long id, Principal principal) {

if (bindingResult.hasErrors()){

model.addAttribute("booking");

return "updateBooking";

}

Booking bookingMapped = bookingMapper.toEntity(booking);

bookingService.update(bookingMapped);

return "redirect:/myBookings";

}

@GetMapping("/hotel/{hotelId}/feedbacks")

public String getFeedbackCreatePage(Model model, @PathVariable("hotelId") Long hotelId){

Hotel hotel = hotelService.findById(hotelId);

model.addAttribute("hotel", hotelMapper.toDto(hotel));

FeedbackDto feedback = new FeedbackDto();

model.addAttribute("feedback", feedback);

return "createFeedback";

}

@PostMapping("/hotel/{hotel}/{hotelId}/feedbacks")

public String createFeedback(@Valid @ModelAttribute("feedback") FeedbackDto feedback, BindingResult bindingResult,

@PathVariable("hotelId") Long hotelId, Principal principal, Model model){

if (bindingResult.hasErrors()) {

Hotel hotel = hotelService.findById(hotelId);

User user = userService.findByEmail(principal.getName());

feedback.setUser(user);

feedback.setHotel(hotel);

model.addAttribute("hotel", hotel);

model.addAttribute("feedback", feedback);

return "createFeedback";

}

Feedback feedbackMapped = feedbackMapper.toEntity(feedback);

Hotel hotel = hotelService.findById(hotelId);

User user = userService.findByEmail(principal.getName());

feedbackMapped.setUser(user);

feedbackMapped.setHotel(hotel);

feedbackService.create(feedbackMapped);

return "redirect:/api/v1/users/hotels/0";

}

}

Приложение В  
(обязательное)  
Листинг SQL-скрипта

create schema if not exists hotels\_schema;

set schema 'hotels\_schema';

create table if not exists users

(

id bigserial,

role varchar(45) not null,

email varchar(45) not null unique,

password varchar(60) not null,

name varchar(45) not null,

surname varchar(45) not null,

birthday date not null,

primary key (id)

);

create table if not exists hotels

(

id bigserial,

name varchar(45) not null,

city varchar(45) not null,

country varchar(45) not null,

nutrition\_type varchar(45) not null,

stars integer not null,

rooms integer not null,

description varchar(1000) not null,

image varchar(200) not null,

min\_price decimal not null,

max\_price decimal not null,

primary key (id)

);

create table if not exists costs

(

id bigserial,

hotel\_id bigint not null,

room\_type varchar(45) not null,

price decimal not null,

foreign key (hotel\_id) references hotels (id) on update cascade on delete cascade,

primary key (id)

);

create table if not exists bookings

(

id bigserial,

user\_id bigint not null,

cost\_id bigint not null,

price decimal,

start\_date date not null,

end\_date date not null,

is\_confirmed boolean not null default false,

foreign key (cost\_id) references costs (id) on update cascade on delete cascade,

foreign key (user\_id) references users (id) on update cascade on delete cascade,

primary key (id)

);

create table if not exists feedbacks

(

id bigserial,

user\_id bigint not null,

hotel\_id bigint not null,

stars integer not null,

text varchar(300) not null,

foreign key (hotel\_id) references hotels (id) on update cascade on delete cascade,

foreign key (user\_id) references users (id) on update cascade on delete cascade,

primary key (id)

);