Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина средства и технологии анализа и разработки информационных систем

**ПОЯСНИТЕЛЬНАЯ** **ЗАПИСКА**

к курсовому проекту

на тему

**Онлайн-сервис управления гостиничным комплексом с возможностью бронирования номеров**

БГУИР КП 1-40 01 02-02 017 ПЗ

Студент Мойсеенко И. А.

Степовой В. К.

гр.672302

Руководитель Салапура М. Н.

Минск 2019

СОДЕРЖАНИЕ

[Введение 5](#_Toc8033325)

[1 Использование информационных технологий для осуществления бронирования онлайн 7](#_Toc8033326)

[1.1 Общие понятия и термины 7](#_Toc8033327)

[1.2 Определение проблемы, требующей оптимизации 8](#_Toc8033328)

[1.3 Обзор существующих решений 9](#_Toc8033329)

[2 Описание основных процессов сервиса бронирования 13](#_Toc8033330)

[2.1 Описание модели «как-есть» 14](#_Toc8033331)

[2.2 Описание модели «как-должно быть» 17](#_Toc8033332)

[3 Спецификация вариантов использования онлайн сервиса управления гостиничным комплексом с возможностью бронирования 20](#_Toc8033333)

[4 Построение информационной модели гостиницы 21](#_Toc8033334)

[5 Обоснование выбора компонент и технологий для реализации курсового проекта 25](#_Toc8033335)

[5.1 Описание выбранных компонентов и технологий 25](#_Toc8033336)

[5.2 Диаграмма последовательности гостиницы для варианта использования «просмотр доступных номеров» 27](#_Toc8033337)

[5.3 Диаграмма развертывания гостиницы 28](#_Toc8033338)

[5.4 Диаграмма компонентов гостиницы 29](#_Toc8033339)

[6 Модели представления гостиницы 30](#_Toc8033340)

[6.1 Диаграмма состояний подписки на новости гостиницы 30](#_Toc8033341)

[6.2 Блок-схема процесса бронирования номера 31](#_Toc8033342)

[6.3 Блок-схема процесса подбора номера 31](#_Toc8033343)

[7 Описание применения паттернов проетировани 32](#_Toc8033344)

[7.1 Диаграмма классов пакета controller 32](#_Toc8033345)

[7.2 Используемые паттерны 32](#_Toc8033346)

[8 Руководство пользователя 35](#_Toc8033347)

[8.1 Руководство по развертыванию приложения 35](#_Toc8033348)

[8.2 Руководство использования приложения 35](#_Toc8033349)

[Заключение 42](#_Toc8033350)

[Список использованных источников 43](#_Toc8033351)

[Приложение А (обязательное) Диаграммы 44](#_Toc8033352)

[Приложение Б (обязательное) Схема алгоритма 48](#_Toc8033353)

[Приложение В (обязательное) Листинг кода 50](#_Toc8033354)

[Приложение Г Скрипт генерации базы данных 74](#_Toc8033355)

# ВВЕДЕНИЕ

Интернет прочно вошел в повседневную жизнь современного человека. С его помощью всегда можно отыскать нужную информацию, сделать в любое время видеозвонок, оплатить различные услуги. Благодаря Интернету появились практически безграничные возможности во всех областях жизни.

Можно перечислить немало плюсов Интернета: Интернет является отличным хранилищем информации. Каждый человек может обратиться к его ресурсам и отыскать ответ на свой вопрос. Энциклопедии и справочники остались в далеком прошлом. Теперь по каждому вопросу можно найти ответ в Сети. Практически безграничные возможности для общения. Человек может общаться в социальных сетях, мессенджерах, найти друзей по интересам. В жизни подростков интернет стал важнейшим способом поделиться своим внутренним миром с другими. В Сети можно проконсультироваться со специалистом, высказать свое мнение, пообщаться. Можно наслаждаться прослушиванием музыки, просмотром картинок или играми [1].

Широкое распространение в современном мире получил онлайн-букинг. В первую очередь – это легко и удобно. Проводя бронирование в интернете, можно отгородить себя от ряда неприятных моментов, т.к. бронировать номера в гостинице, проще простого, все, что для этого нужно – это воспользоваться компьютером и интернетом. Заходя на сайт бронирования можно без труда найти и заказать подходящий номер.

Просматривая номера и гостиницы можно узнать отзывы, заранее узнать необходимую информацию о номерах и других значимых аспектах. Для того, чтобы забронировать номер, необходимо оставить заявку. В течении какого-то времени заявка будет обрабатываться, после чего придёт уведомление о бронировании.

В результате этого, можно оформить бронирование, не выходя из дома.

Для любого человека, который отправляется отдохнуть на какой-либо курорт или же в командировку, очень актуальным является вопрос: где остановиться? Сегодня есть несколько вариантов, но традиционным и в то же время самым удобным является проживание в одном из номеров гостиницы.

Среди всех этих услуг особое место занимает онлайн бронирование гостиницы, которое является гарантией того, что по приезде в другой город у человека точно будет временное место проживания. И это очень удобная услуга, так как это можно сделать даже, не выходя из своей квартиры. Этот вариант выгодный и по той причине, что дает возможность заранее подобрать номер в гостинице по своему усмотрению. Ведь если не забронировать номер заранее, то по приезде в отеле менеджер предложит только те номера, которые остались. А это означает, что они или слишком дорогие, или же располагаются в неудобном месте. И не факт, что вообще окажется хоть один свободный номер.

При бронировании номера будущий постоялец может выбрать удобный номер, с прекрасным видом из окна. Сегодня большинство гостиниц, которые предлагают такую услугу, выкладывают на страницах своих сайтов фотографии номеров, их полное описание. Также описывается, какие еще услуги предоставляет данный отель. Не очень-то приятно приехать в другой город и услышать в отеле, что номеров свободных нет. Приходится снова вызывать такси и ехать в поисках нового подобного заведения.

Целью данного курсового проекта является автоматизация процесса бронирования номеров гостиничного комплекса.

Для достижения поставленной цели необходимо выполнить ряд задач:

* осуществить анализ процесса бронирования номеров, принцип работы сайта гостиничных комплексов;
* проанализировать логическую и физическую модели представления данных;
* создать базу данных;
* реализовать серверную часть, которая позволит работать с базой данных и выполнять ряд функций, поставленных целью разработки курсового проекта;
* предусмотреть защиту данных пользователей в системе;
* разработать сайт управления гостиничным комплексом с возможностью бронирования;
* провести тестирование разработанного программного продукта;
* выполнить отладку ошибок, обнаруженных в процессе тестирования;
* предложить способы устранения ошибок.

# ИСПОЛЬЗОВАНИЕ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ДЛЯ ОСУЩЕСТВЛЕНИЯ БРОНИРОВАНИЯ ОНЛАЙН

# Общие понятия и термины

Онлайн-бронирование (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) online — на линии, на связи) — [бронирование](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D1%8D%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D0%BA%D0%B0)) через [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), в [интерактивном](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) режиме. Термин применяется по отношению к бронированию номеров в [гостиницах](https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B8%D1%86%D0%B0), [билетов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BB%D0%B5%D1%82_(%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82)) ([авиа](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D0%B8%D0%B0%D0%B1%D0%B8%D0%BB%D0%B5%D1%82), [железнодорожных](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BB%D0%B5%D1%82_%D0%BD%D0%B0_%D0%BF%D0%BE%D0%B5%D0%B7%D0%B4), автобусных и т.п), мест в [ресторанах](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%81%D1%82%D0%BE%D1%80%D0%B0%D0%BD) и [театрах](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%B0%D1%82%D1%80), [прокату автомобилей](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BA%D0%B0%D1%82_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%BE%D0%B1%D0%B8%D0%BB%D0%B5%D0%B9) и т. д.

Системами онлайн бронирования гостиниц называются системы, в которых отражено наличие реальных номеров в гостиницах и в которых информирование гостиницы о бронировании номера клиентом происходит менее чем за одну минуту с момента бронирования, бронирование номера в гостинице при этом возможно за любой минимальный период времени до начала брони т.е. более чем за одну минуту.

На сайтах гостиничного и туристического бизнеса можно встретить два вида бронирования:

Истинное онлайн-бронирование — когда клиент выбирает, бронирует и оплачивает номер самостоятельно через системы ADS или напрямую на сайте отеля. Для оплаты клиент использует пластиковую карточку,данные которой передаются в отель напрямую и через несколько секунд получает ваучер, подтверждающий бронь.

Псевдоонлайн-бронирование (также «бронирование по запросу») — когда клиент заполняет форму заявки на сайте гостиницы и отправляет её администратору. В течение некоторого времени клиент ждет звонка сотрудника службы бронирования для подтверждения брони. «Бронирование по запросу» иногда неправильно называют «онлайн-бронированием».

**Как работает онлайн-бронирование**

Система онлайн-бронирования встраивается в сайт гостиницы вместо формы бронирования по запросу.

Клиент выбирает категорию номера, дату заезда и выезда, а также другие гостиничные услуги на сайте гостиницы.

Система предлагает подтвердить выбранный набор гостиничных услуг.

Клиент оплачивает первые сутки проживания сразу на сайте и получает ваучер на гарантированное заселение. Также система поддерживает безналичный расчёт и возможность оплаты «на месте».) Также возможны и другие варианты оплаты.

Оставшуюся сумму клиент доплачивает непосредственно в кассу гостиницы при заезде.

Таким образом, клиент проходит полный цикл бронирования непосредственно на сайте в течение 3-4 минут.

Кроме формы онлайн-бронирования на сайте гостиницы, можно все формы с гостиниц собрать на одном сайте. Тогда и получается система онлайн-бронирования. Есть известные западные системы онлайн-бронирования. Кроме того, развиваются российские системы онлайн-бронирования.

Самая надежная система онлайн-бронирования гостиниц — когда информация о свободных номерах прямо из гостиницы передается на сайт и затем бронирование гостя попадает прямо в АСУ гостиницы. Тогда нет ошибок под названием «овербукинг» - двойного перебронирования номера.

Преимущества онлайн-бронирования:

Моментальная оплата заказа на сайте.

Клиент сразу получает гарантию заезда по ценам гостиницы. Часто системы онлайн-бронирования предоставляют скидку на проживание в номерах за счет своей комиссии, чтобы привлечь больше клиентов. Таким образом, стоимость номера выходит дешевле чем если бронировать его в самой гостинице.

Клиент сам выбирает период проживания, категорию номера, набор дополнительных гостиничных услуг.

Гостинице нет необходимости связываться с клиентом, так как бронирование проходит в автоматическом режиме без участия администратора.

Гостиница сама определяет размер квоты для бронирования онлайн, цены, набор дополнительных услуг.

Система работает в автономном режиме круглосуточно 24 часа 7 дней в неделю.

# Определение проблемы, требующей оптимизации

В настоящее время существует огромное количество хороших букинговых сервисов. Для того, чтобы не отставать на рынке от конкурентов, каждый сервис должен привлечь как можно больше клиентов. Всем известно, что самый эффективный способ донести информацию до потребителей – интернет. Практически каждая организация, в наше время, имеет свой собственный сайт, на котором клиенты могут получить всю необходимую информацию о продукции.

Огромную роль в решении клиентов играют оформление сайта, удобство использования, простота изложения информации и всевозможные скидки. Для того, чтобы не потерять потребителей, необходимо создать качественный сайт, который будет удовлетворять требованиям пользователей и отличаться от сайтов конкурентов.

# Обзор существующих решений

Одним из самых популярных сайтов для бронирования номеров является сервис «Booking.com» (рисунок 1.1) На данном сайте клиент может выбрать номера, гостиницы, место отправления, количество человек и т.д (рисунки 1.2 - 1.3).

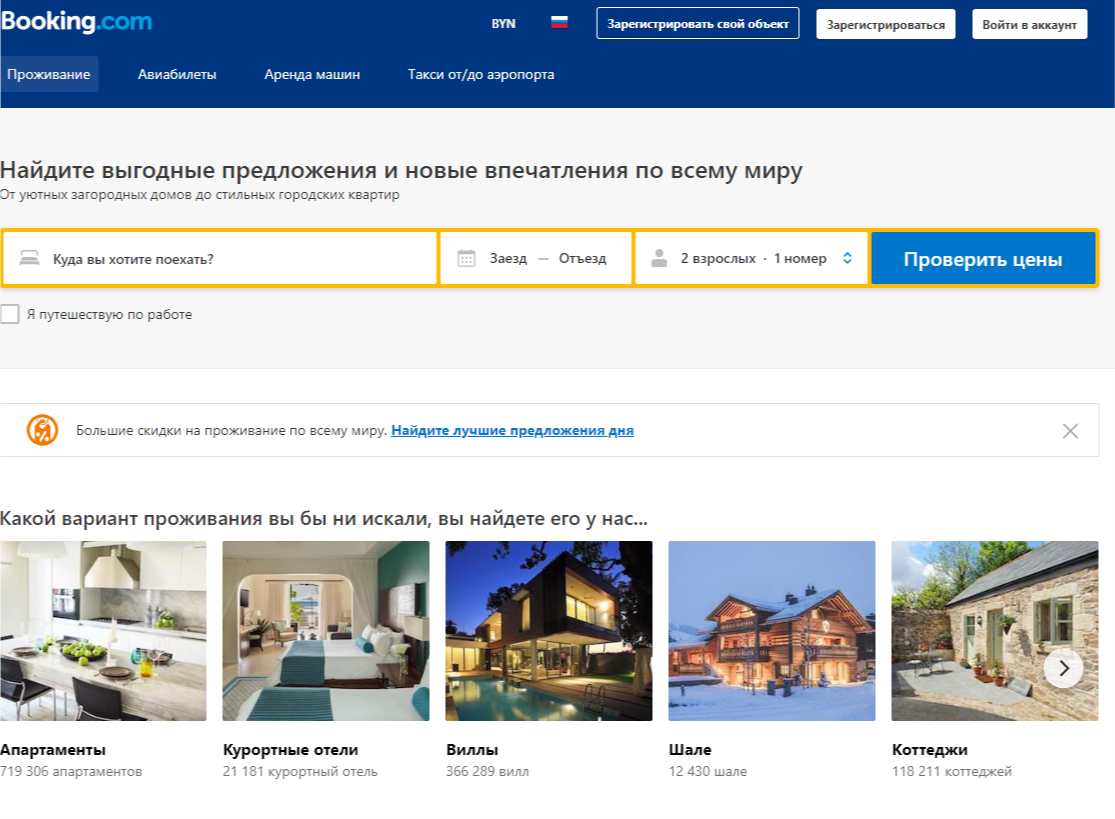


Рисунок 1.1 – Сайт «Booking.com»

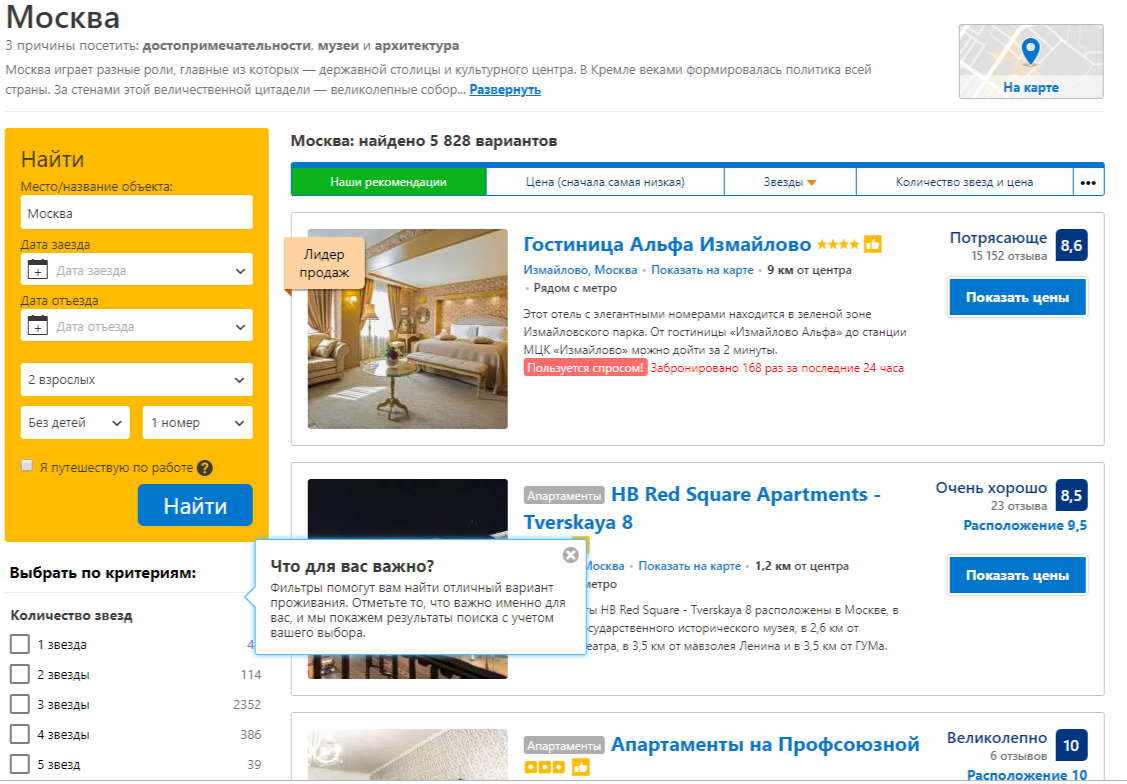


Рисунок 1.2 – Выбор гостиницы

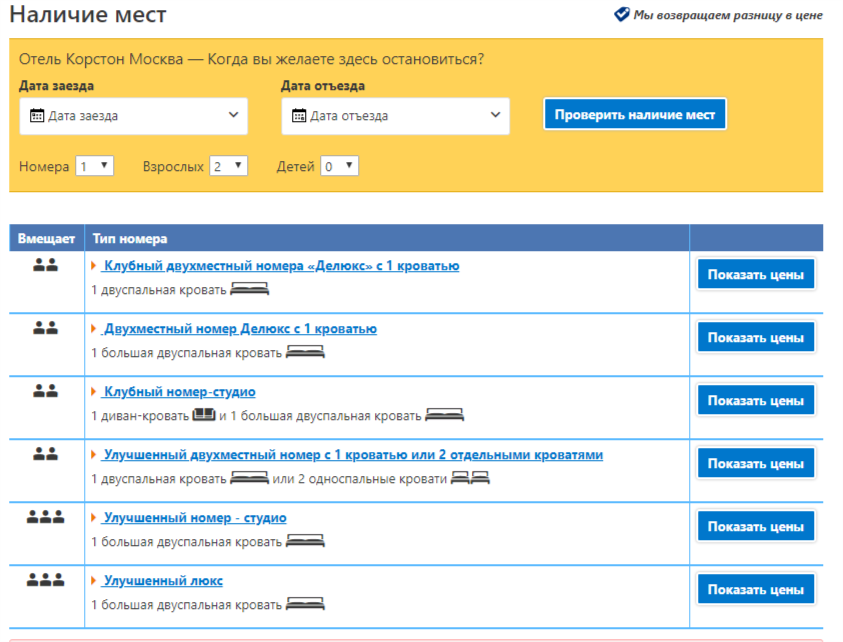


Рисунок 1.3 – Проверка наличия мест

В процессе выбора номеров клиент выбирает подходящие места и условия для дальнейшего оформления. Далее необходимо ввести всю требуемую информацию (рисунок 1.4).

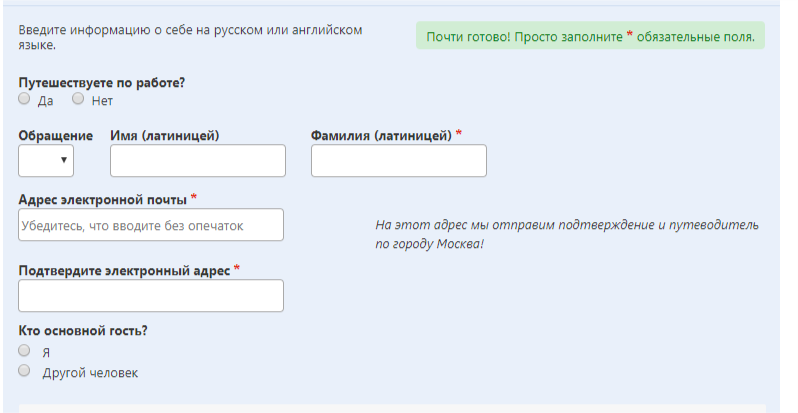


Рисунок 1.4 – Ввод информации о клиенте

После того, как клиент окончательно определился с выбором и ввел все необходимые данные ему необходимо оформить и оплатить заказ (рисунок 1.5).

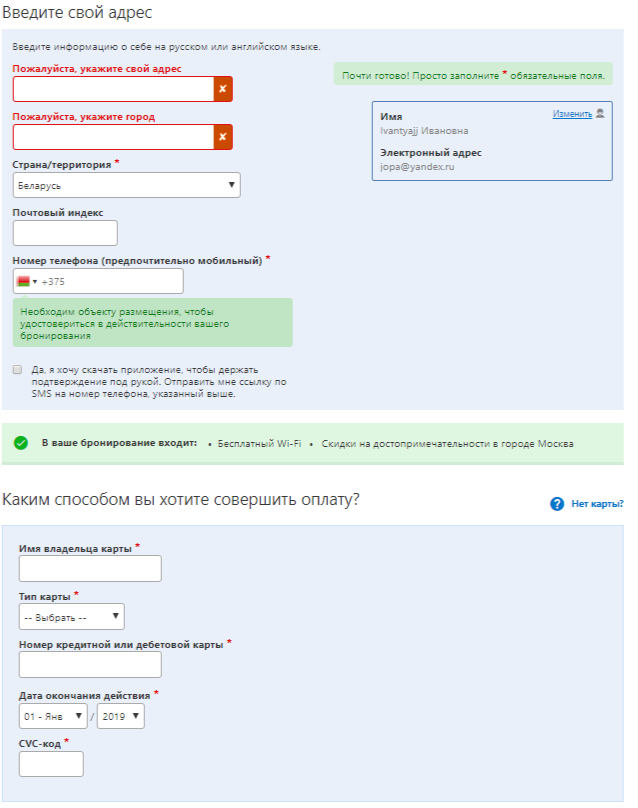


Рисунок 1.5 – Оформление и оплата

# ОПИСАНИЕ ОСНОВНЫХ ПРОЦЕССОВ СЕРВИСА БРОНИРОВАНИЯ

При анализе сервисов бронирования номеров возникает потребность в исследовании основных процессов, происходящих в системе и их взаимодействия между собой, то есть, возникает необходимость в построении функциональной модели проектируемой системы.

Такая модель помогает лучше понять взаимосвязь между процессами, определить «узкие» места, отрицательно сказывающиеся на эффективности деятельности системы, разработать мероприятия по устранению имеющихся проблем, путем перестройки отдельных бизнес-процессов или реорганизации бизнес-архитектуры в целом. Все это позволяет задать правила выполнения процессов, следование которым дает возможность достичь нужной производительности процессов, или даже повысить эффективность работы системы.

Выполнение этой задачи подразумевает под собой создание двух видов функциональных моделей:

* модели существующей организации процесса AS-IS («как-есть»);
* модели улучшенной организации процесса TO-BE («как-должно- быть»).

Моделирование бизнес-процессов может включать в себя выполнение нескольких последовательных стадий.

Для улучшения процесса необходимо понимать, как он работает в данный момент. На этой стадии определяются границы процесса, выявляются его ключевые элементы, собираются данные о работе процесса. В результате создается исходная модель процесса «как-есть». Эта модель не всегда адекватно отражает работу процесса, поэтому модель этой стадии можно назвать пробной или исходной моделью «как-есть».

Дальше идёт пересмотр, анализ и уточнение исходной модели. На этой стадии выявляются противоречия и дублирование действий в процессе, определяются ограничения процесса, взаимосвязи процесса, устанавливается необходимость изменения процесса. В результате формируется окончательный вариант модели «как-есть».

Следующим этап - разработка модели «как-должно-быть». После анализа существующей ситуации, необходимо определить желаемое состояние процесса. Это состояние представляется в модели «как- должно-быть». Такая модель показывает, как процесс должен выглядеть в будущем, после всех улучшений.

Рассмотрим процесс бронирования номера онлайн при помощи методологии IDEF0. В данной главе рассмотрим две модели, которые были построены на основе анализа сервисов бронирования номеров.

# Описание модели «как-есть»

На рисунке 2.1 представлена контекстная диаграмма верхнего уровня. Также изображены основные входы и выходы, механизмы и управление.

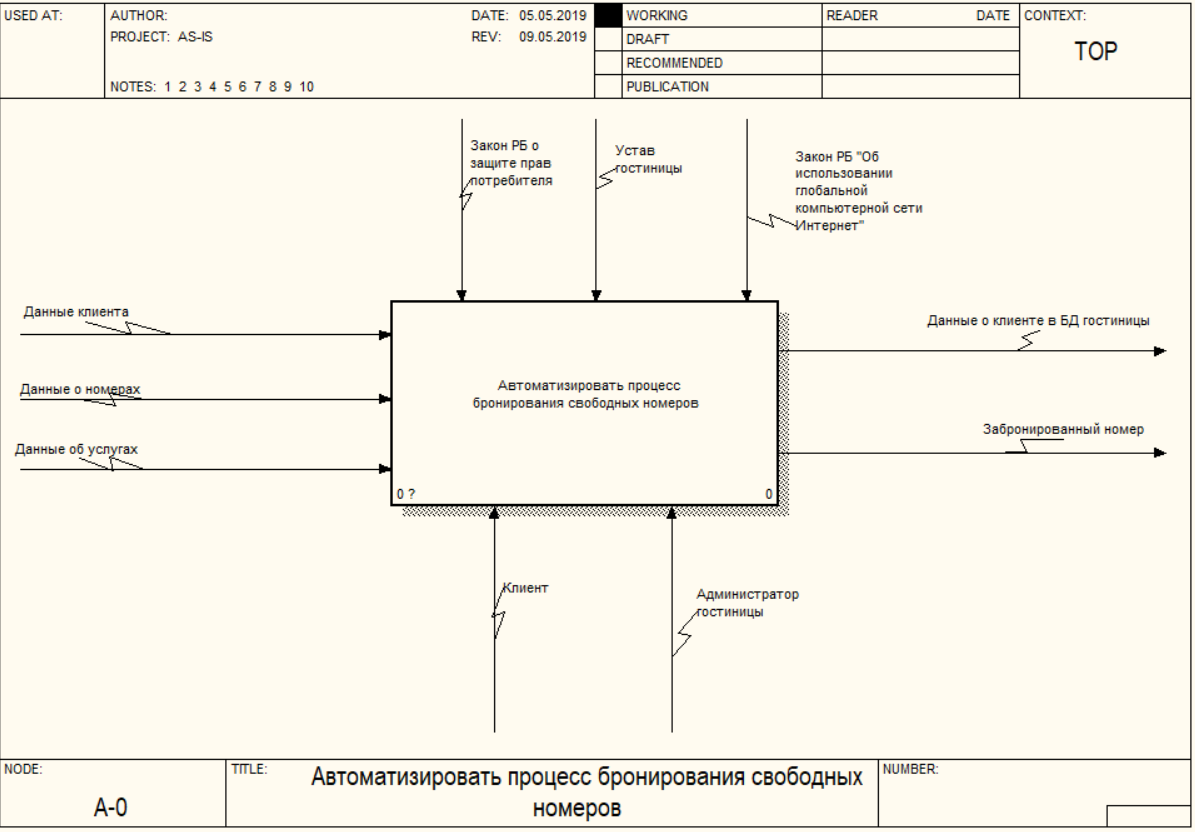
****

Рисунок 2.1 – Контекстная диаграмма верхнего уровня

Входными параметрами на диаграмме верхнего уровня являются данные о номерах, данные о клиентах и услугах. Управляющими механизмами являются администратор гостиницы и сам клиент. На выходе имеем данные о клиенте, которые отправляются в БД, забронированный номер.

Далее на рисунке 2.2 рассмотрим декомпозицию контекстной диаграммы верхнего уровня.

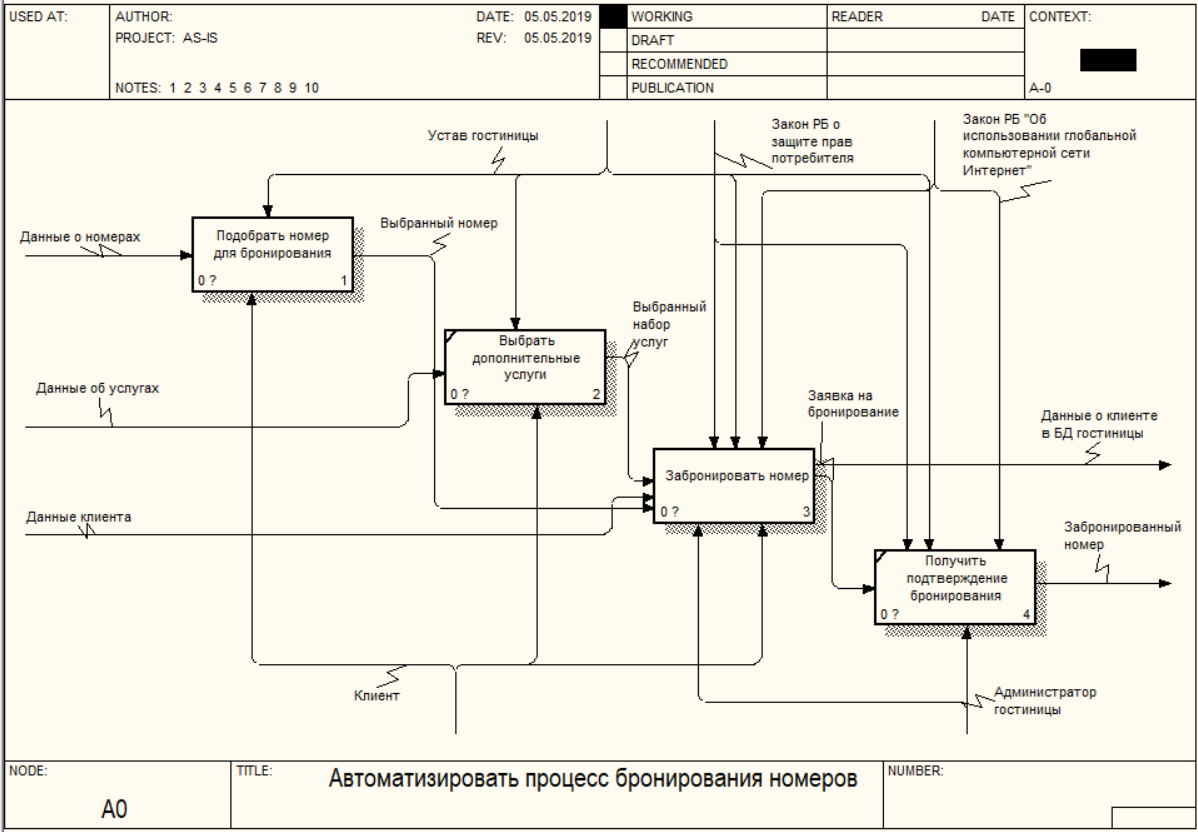


Рисунок 2.2 – Декомпозиция контекстной диаграммы верхнего уровня

Перед бронированием номера клиенту необходимо подобрать подходящий номер и выбрать необходимые услуги. Далее клиент производит бронирования номера и затем получает подтверждение бронирования.

Декомпозиции блоков «Подобрать номер для бронирования» и «Забронировать номер» представлены на рисунке 2.3 и рисунке 2.4 соответственно.

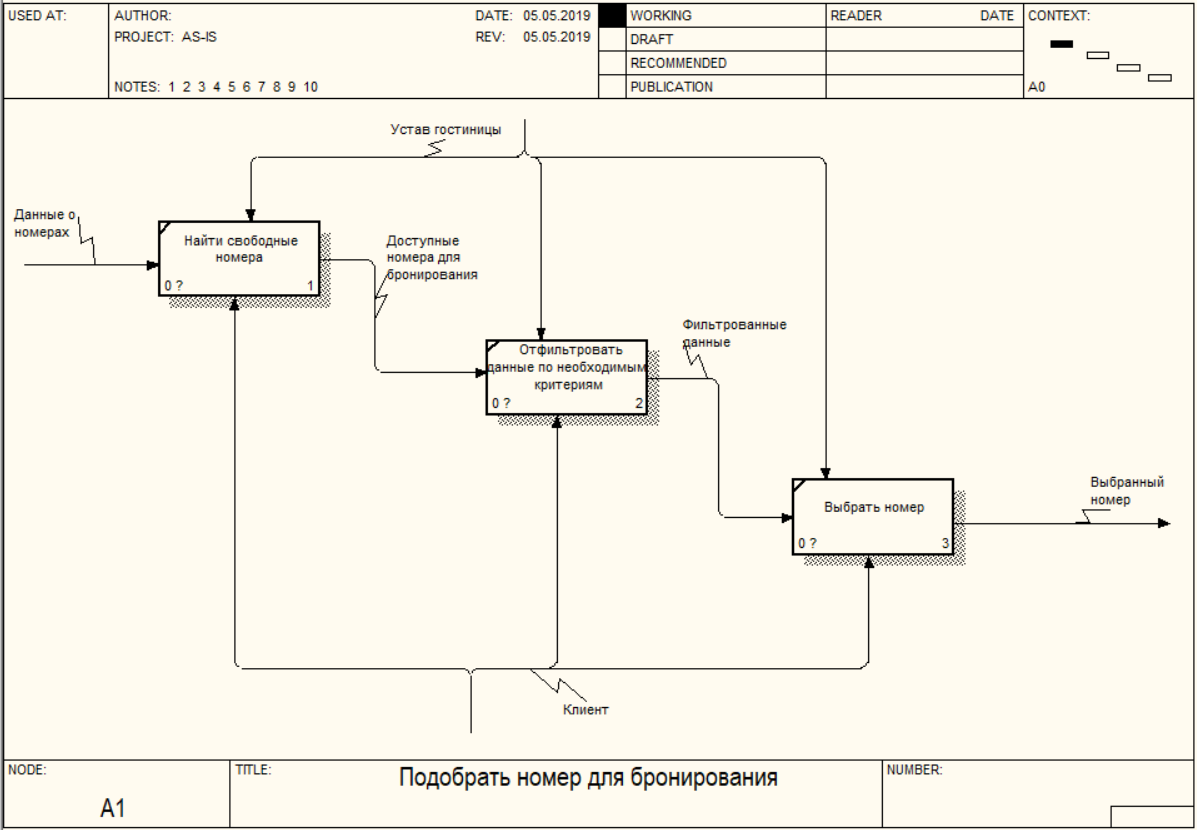


Рисунок 2.3 – Декомпозиция блока «Подобрать номер для бронирования»

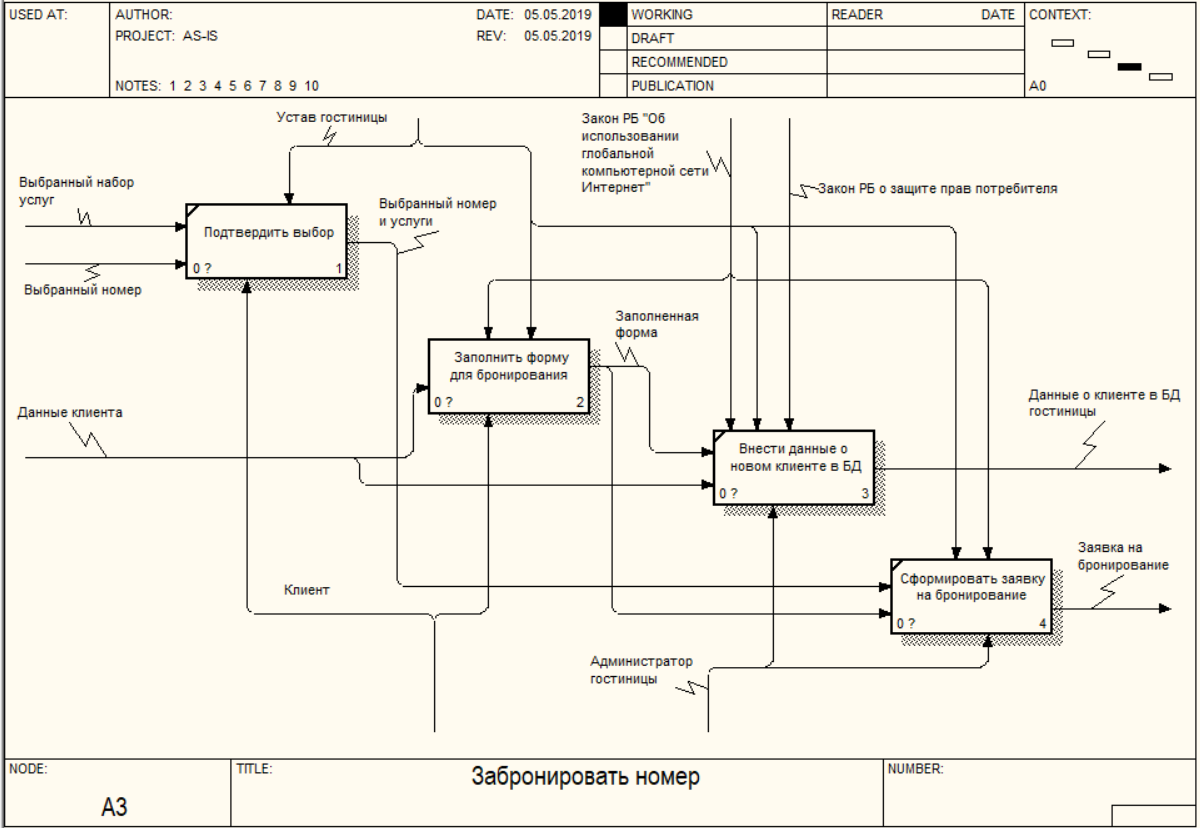


Рисунок 2.4 – Декомпозиция блока «Забронировать номер»

При бронировании номера необходимо окончательно определиться с номером, ввести все необходимые данные и нажать кнопку для формирования заявки на бронирование.

Декомпозиция блока «Заполнить форму для бронирования» представлена на рисунке 2.5. Для корректного проведения бронирования необходимо отправить все свои данные.

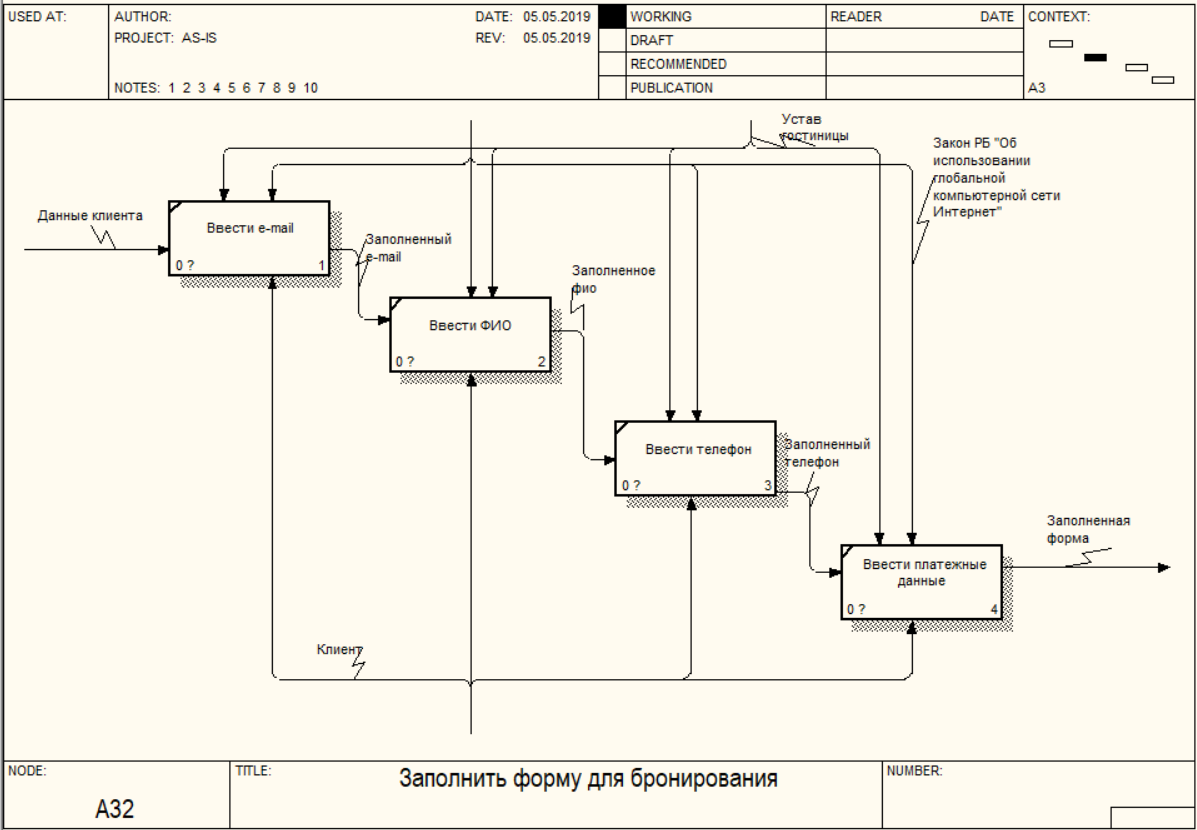


Рисунок 2.5 – Декомпозиция блока «Заполнить форму для бронирования»

# Описание модели «как-должно быть»

Далее представляется к рассмотрению модель «как-должно-быть». На рисунке 2.6 представлена контекстная диаграмма верхнего уровня.

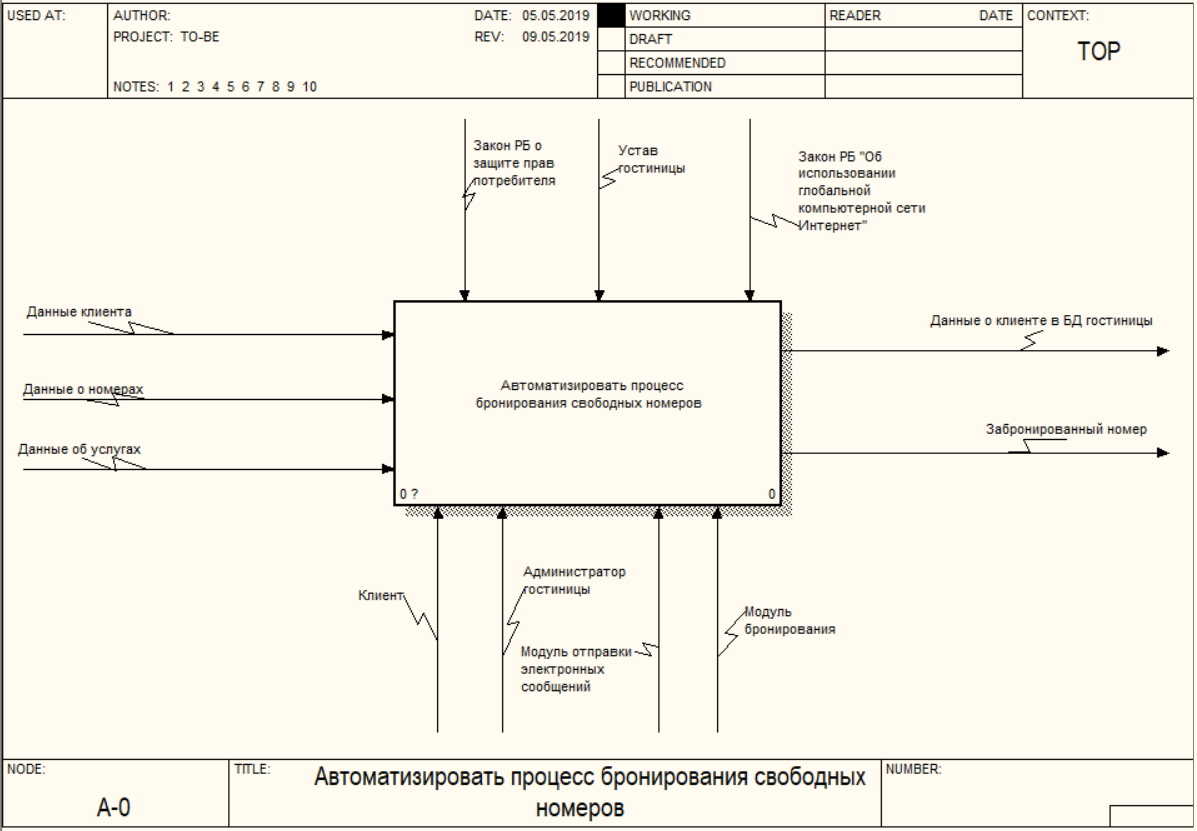


Рисунок 2.6 – Контекстная диаграмма верхнего уровня

Контекстная диаграмма верхнего уровня модели «как-должно-быть» отличается от модели «как-есть» тем, что добавлены два управляющих модуля (модуль отправки электронных сообщений и модуль бронирования), для более эффективной автоматизации процесса.

В связи с этим главные изменения коснулись блока «Забронировать номер» (рисунок 2.7).

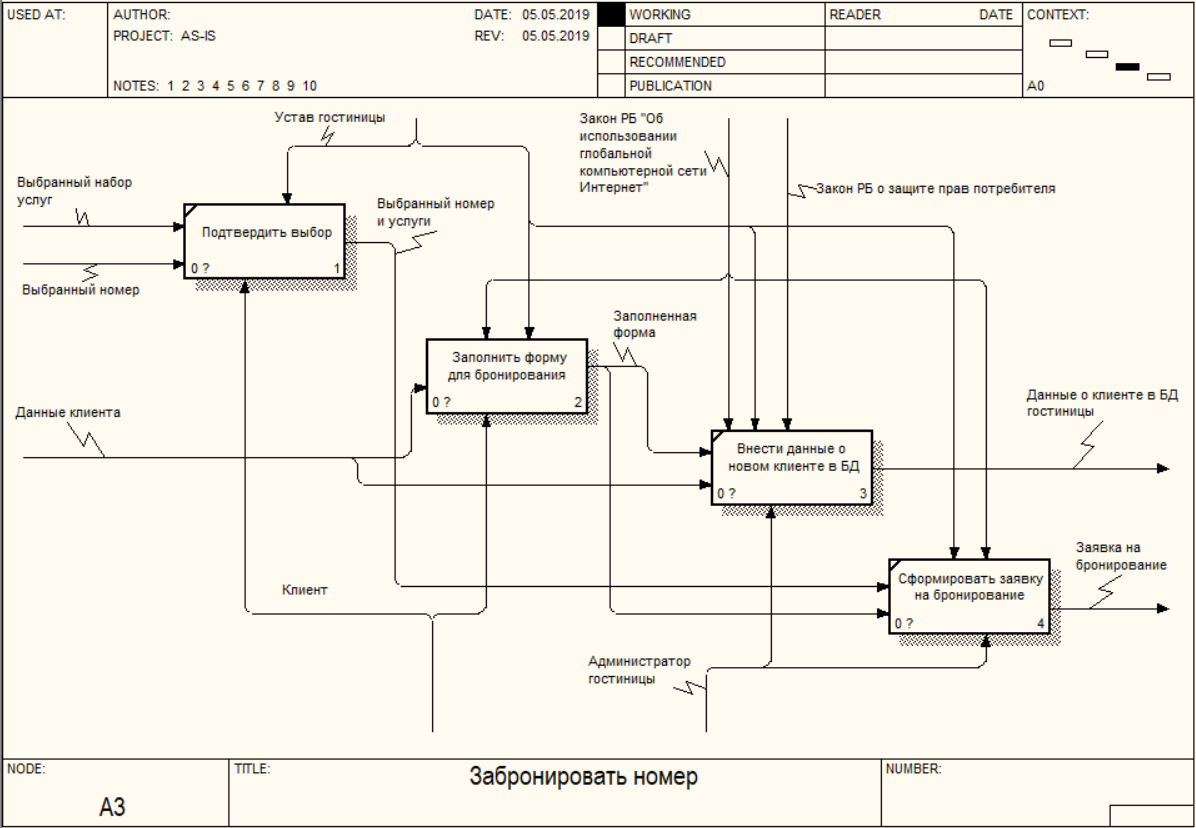


Рисунок 2.7 – Декомпозиция блока «Забронировать номер»

Теперь нет необходимости в непосредственном участии администратора в процессе бронирования. Заявка формируются и отправляется вместе со ссылкой подтверждения на e-mail клиента в автоматическом режиме.

# СПЕЦИФИКАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ ОНЛАЙН СЕРВИСА УПРАВЛЕНИЯ ГОСТИНИЧНЫМ КОМПЛЕКСОМ С ВОЗМОЖНОСТЬЮ БРОНИРОВАНИЯ

При изучении сервисов бронирования была разработана модель, позволяющую изучить функциональные возможности каждого участника, который пользуется данным приложением. Эту модель мы представляем в виде UML диаграммы, а именно диаграммы использования (приложение А рисунок А.1).

Построение данной диаграммы необходимо для описания того, что система в состоянии делать и с кем (или с чем) она будет взаимодействовать. Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с ней с помощью, так называемых вариантов использования [4].

На данной диаграмме видим двух актёров: клиент и администратор. Для входа в систему администратор должен пройти авторизацию, если он ещё не был зарегистрированы в сети. Клиент же может произвести заказ без предварительной регистрации. Регистрация клиента в системе происходит в момент первого заказа.

Клиент может просмотреть как выглядят номера гостиницы, а также прочитать о близлежащих достопримечательностях. Есть возможность подписаться на рассылку новостей гостиницы о новых предложениях и скидках. Также, при необходимости, клиент может напрямую связаться с администрацией гостиницы. При бронировании номера клиентом необходимо выбрать номер из доступных, подобрать дату, выбрать количество посетителей, выбрать пакет услуг, цену, заполнить личные данные и оставить заявку на бронирование. При бронировании так же есть возможность заказать обратный звонок.

Администратор может управлять номерами: например, редактировать цены, описание, а также менять прикрепленную фотографию. Администратору доступен просмотр данных о бронировании, также администратор вправе отклонить заявку на бронирование и отправить сообщение с причиной отказа. Предоставлена возможность просмотра статистики по номерам или бронированию.

# ПОСТРОЕНИЕ ИНФОРМАЦИОННОЙ МОДЕЛИ ГОСТИНИЦЫ

Для обеспечения функционала разрабатываемого сайта была спроектирована информационная модель, представленная на рисунке 4.1.

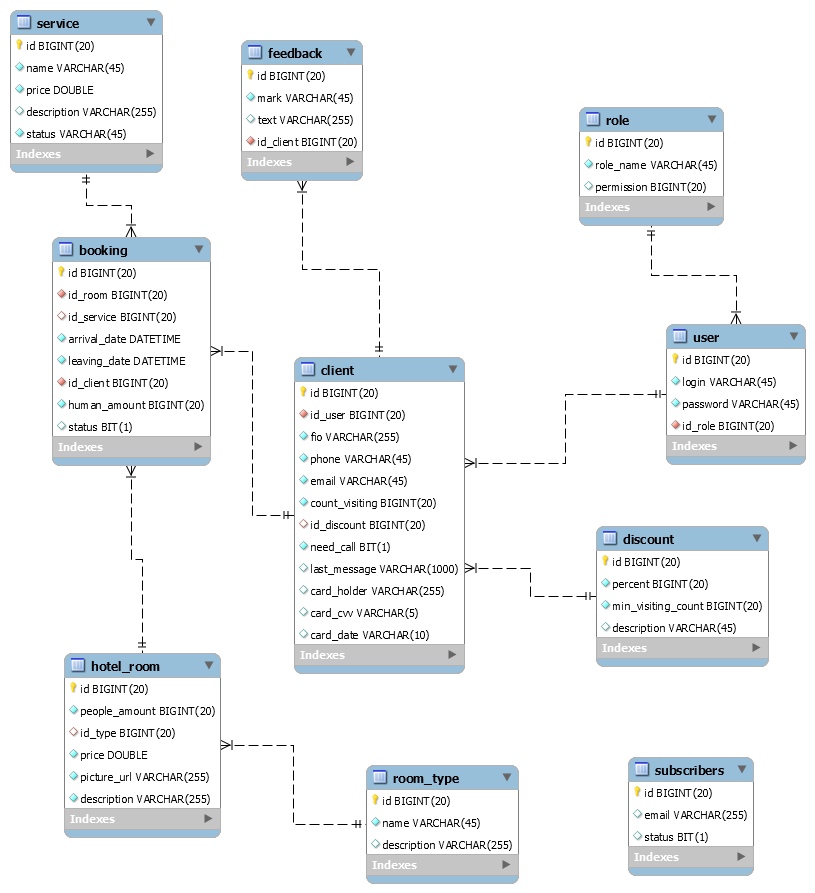


Рисунок 4.1 - Информационная модель программного продукта

В информационной модели были выделены следующие сущности:

- тип комнаты (room\_type);

- услуги (service);

- заявка на бронирование (booking);

- номер в отеле (hotel\_room);

- клиент гостиницы (client);

- отзывы (feedback);

- пользователь (user);

- роль пользователя (role);

- скидки (discount);

Сущность hotel\_room представляет комнату в гостинице, содержит идентификатор (id), фотография номера(picture\_url), вместимость номера (people\_amount), цена за номер (price), идентификатор типа комнаты (id\_type) и описание номера (description)

Сущность room\_type содержит идентификатор (id), название типа (name) и описание (descriptinon).

Сущность client представляет клиента гостиницы. Содержит идентификатор (id), идентификатор пользователя, под которым был выполнен вход на сайт (id\_user), количество посещений гостиницы (count\_visiting), идентификатор полученной скидки (id\_discount) и поля с личными данными: имя (name), фамилия (surname), телефон (phone), электронная почта (email), последнее сообщение пользователя (last\_message), требуется ли обратный звонок(need\_call), данные о карте клиента, если оплата производилась через карту: имя держателя карты (card\_holder), код безопасности(card\_cvv), срок действия(card\_date).

Сущность user необходима для хранения данных о зарегистрированных пользователях в системе. Содержит идентификатор (id), логин (login), пароль (password), идентификатор присвоенной роли (id\_role).

Cущность role с полем идентификатора (id), названием роли (role\_name) и уровнем доступа (permission).

Сущность discount содержит идентификатор (id), процент скидки (percent), минимальное необходимое количество посещений для получения скидки (min\_visiting\_count) и описание условий для получения скидки.

Cущность feedback для хранения отзывов содержит идентификатор (id), оценку (mark), текст отзыва (text) и идентификатор клиента (id\_client), оставившего отзыв.

Сущность service предназначена для хранения данных о предоставляемых сервисах. Содержит идентификатор (id), название сервиса (name), цену (price), описание (description), статус доступности сервера в данный момент (status).

Главная сущность booking необходима для хранения информации о бронировании номера. Содержит идентификатор (id), идентификатор забронированной комнаты (id\_room), идентификатор выбранного сервиса (id\_service), дата приезда (arrival\_date), дата уезда (leaving\_date), идентификатор клиента (id\_client), количество человек планирующих приезд (human\_amount), статус бронирования (status).

Также в проект добавлена Swagger API документация, для описания REST API приложения.

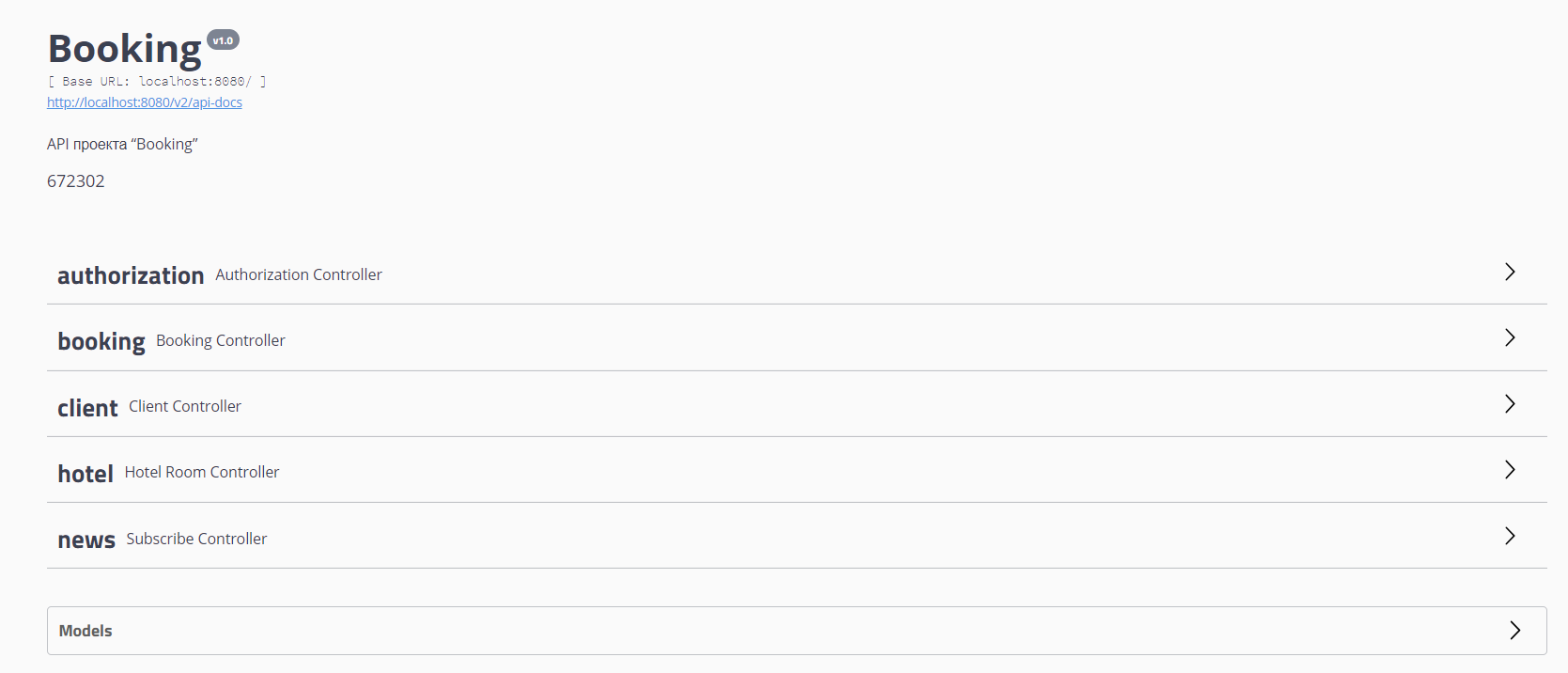


Рисунок 4.2 – Swagger Api в общем виде для проекта

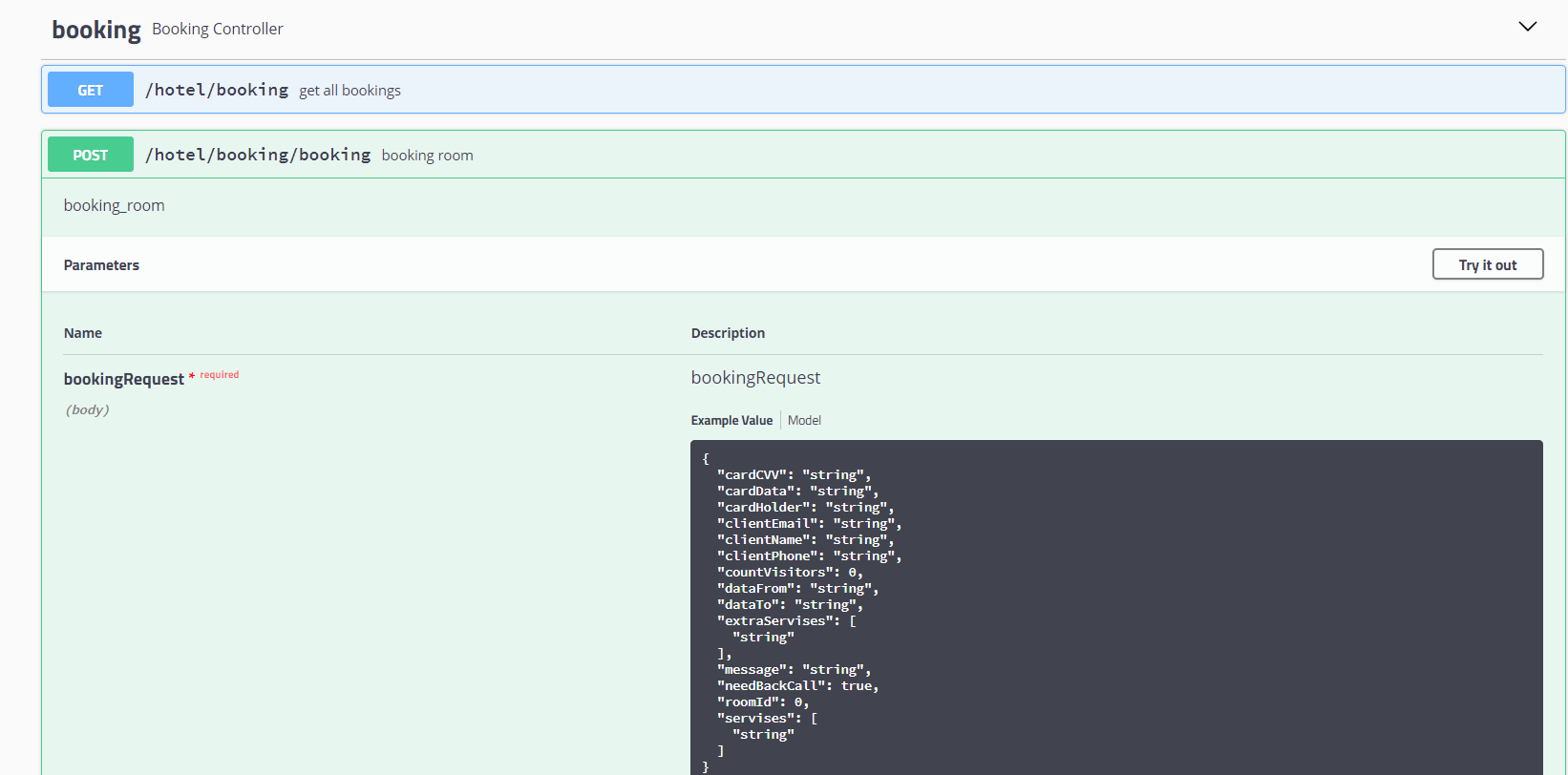


Рисунок 4.3 – Пример Swagger документации BookingController

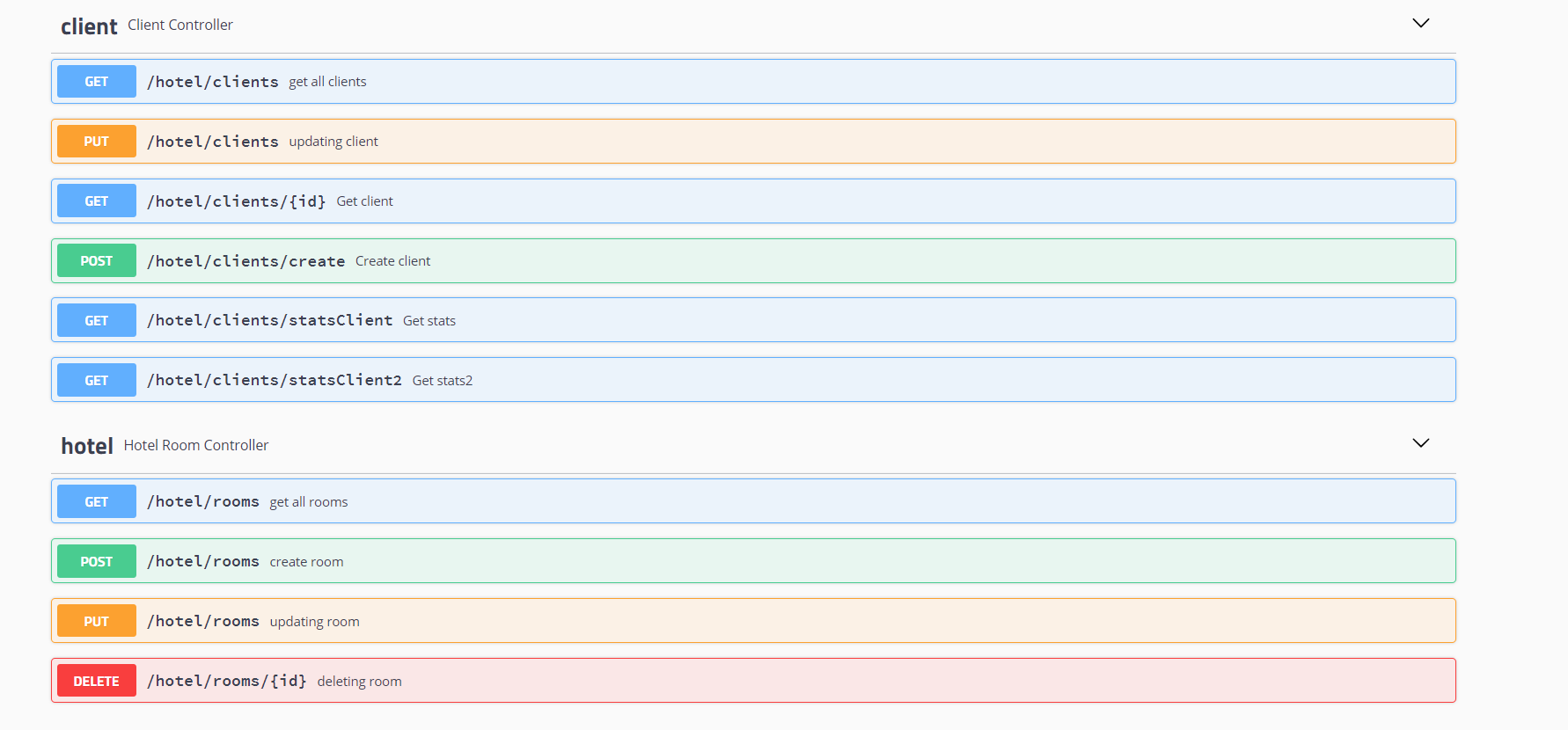


Рисунок 4.4 – Пример Swagger документации ClientController и HotelRoomController

# ОБОСНОВАНИЕ ВЫБОРА КОМПОНЕНТ И ТЕХНОЛОГИЙ ДЛЯ РЕАЛИЗАЦИИ КУРСОВОГО ПРОЕКТА

# Описание выбранных компонентов и технологий

Для разработки данного программного продукта был выбран язык программирования Java. Так как он является самым популярным и востребованным языком программирования. Java является полностью объектно-ориентированным языком. Важно и то, что писать программы без ошибок на Java намного легче. Все дело в том, что разработчиками языка Java из компании Sun был проведен фундаментальный анализ программ на языке С++. Анализировались "узкие места" исходного кода, которые и приводят к появлению трудновыявимых ошибок. Поэтому было принято решение проектировать язык Java с учетом возможности создавать программы, в которых были бы скрыты наиболее распространенные ошибки [4].

Программы на Java транслируются в байт-код, выполняемый виртуальной java-машиной (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор, но с тем отличием, что байтовый код, в отличие от текста, обрабатывается значительно быстрее.

Достоинство подобного способа выполнения программ — в полной независимости байт-кода от ОС и оборудования, что позволяет выполнять Java-приложения на любом устройстве, которое поддерживает виртуальную машину. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание. Это позволяет пользователям загружать программы, написанные на Java, на их компьютеры из неизвестных источников, при этом не опасаясь заражения вирусами и пропажи ценной информации [5].

В курсовом проекте был использован такой фреймворк как Spring Boot который даёт возможность за короткое время создать приложение, в основе которого лежит Spring, а также и его компоненты. Быстрая разработка связана с возможностью использования большого количества сконфигурированных компонентов, что приводит к упрощению работы с зависимостями, а самое главное помогает избежать применения в разработке XML конфигурации, потому что это требует большое количество времени [6].

Использовался фреймворк angularjs, его цель — расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller)-шаблона, а также упрощение тестирования и разработки.Фреймворк работает с [HTML](https://ru.wikipedia.org/wiki/HTML), содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

Apathe Maven – это фреймворк для автоматизации сборки проектов, компиляции, создания jar, создания дистрибутива программы, генерации документации. Maven, обеспечивает декларативную, а не императивную сборку проекта. То есть, в файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды. Все задачи по обработке файлов Maven выполняет через плагины [7].

У maven есть достаточно много преимуществ и вот некоторые из них:

* независимость от OS. Сборка проекта происходит в любой операционной системе;
* управление зависимостями. Редко какие проекты пишут без использования сторонних библиотек;
* хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты, которые собираются с помощью maven;
* декларативное описание проекта.

В данном курсовом проекте были подключены сторонние библиотеки с помощью настроек maven.

Также с помощью плагинов сборки был собран jar файл и сгенерирована структура проекта.

Hibernate - библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (ORM), распространяется свободно [8].

Данный фрэймворк был выбран для соединения с базой данных потому что он имеет ряд преимуществ:

* Hibernate — существенно уменьшает время разработки, на проектах с достаточно большим набором таблиц\сущностей и связей;
* он удаляет множество повторяющегося кода из JDBC API, а, следовательно, его легче читать, писать и поддерживать;
* поддерживает ассоциации, наследование и коллекции, что недопустимо в JDBC API;
* неявно использует управление транзакциями. Так как большинство операций нельзя выполнить вне транзакций;
* поддерживает аннотации JPA, а значит код является переносимым на другие ORM Фреймворки [9].

Данные преимущества являются главной причиной выбора этого фреймворка. Все эти преимущества были использованы в нашей программе. Из информационной модели системы можно заметить большое количество связей между таблицами. Что является положительным фактором для выбора Hibernate.

В реализации запросов к серверу базы данных намного уменьшился код и он стал еще более читабельным и легким к последующему изменению.

Благодаря поддержке коллекций мы можем получать необходимые данные через внешний ключ без использования дополнительных запросов, ведь все данные уже хранятся в коллекциях, и, чтобы получить доступ к данным, необходимо всего лишь обратиться к ним через метод get.

При использовании транзакций повышается безопасность. Так как случает возникновения ошибки при выполнении операции, транзакция будет отменена и данные останутся в сохранности.

Также в данной библиотеке если поддержка аннотаций JPA. Что позволяет использовать классы, которые необходимы для создания моделей, использовать как классы носители информации.

# Диаграмма последовательности гостиницы для варианта использования «Просмотр доступных номеров»

Диаграмма развертывания иллюстрирует, как и на каких устройствах запускается программа.

Данная диаграмма нужна для представления общей конфигурации или топологии распределенной программной системы и включает в себя изображение размещения различных артефактов по отдельным узлам системы. Во время разработки диаграмм развертывания преследуются следующие цели: специфицировать физические узлы, необходимые для размещения на них исполнимых компонентов программной системы, отразить физические связи между узлами реализации системы на этапе ее исполнения. Обнаружить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности. Диаграмма последовательности поиска свободного номера гостиницы представлена на рисунке 5.1.



Рисунок 5.1 – Диаграмма последовательности поиска свободной комнаты

# Диаграмма развертывания гостиницы

Диаграмма развертывания иллюстрирует, как и на каких устройствах запускается программа.

Данная диаграмма нужна для представления общей конфигурации или топологии распределенной программной системы и включает в себя изображение размещения различных артефактов по отдельным узлам системы. Во время разработки диаграмм развертывания преследуются следующие цели: специфицировать физические узлы, необходимые для размещения на них исполнимых компонентов программной системы, отразить физические связи между узлами реализации системы на этапе ее исполнения. Обнаружить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности. Диаграмма развертывания гостиницы представлена в приложении А на рисунке А.2.

# Диаграмма компонентов гостиницы

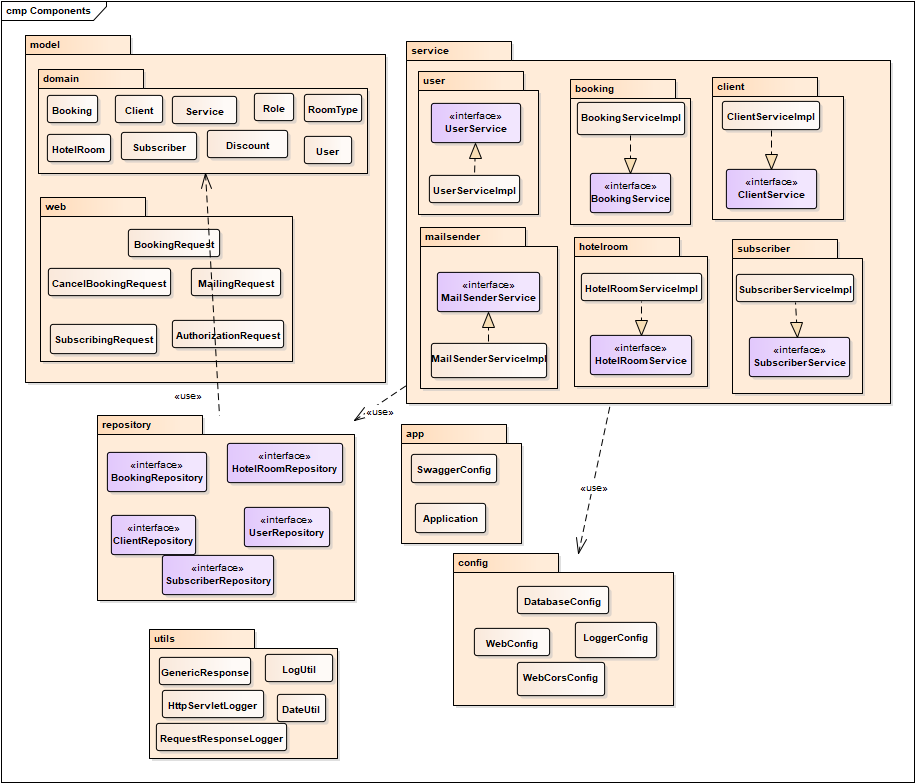


Рисунок 5.2 – Диаграмма компонентов

С помощью диаграммы компонентов мы можем определить архитектуру разрабатываемой системы, с помощью установления зависимости между программными компонентами.

# МОДЕЛИ ПРЕДСТАВЛЕНИЯ ГОСТИНИЦЫ

# Диаграмма состояний подписки на новости гостиницы

Диаграмма состояний представлена на рисунке 6.1.

На данной диаграмме представлены состояния процесса подписки. Изначально система находится в ожидании ввода данных (e-mail). После ввода данные проходят проверку на валидность. Если ввести некорректный e-mail, то система выдаст ошибку и вернется в состояние ожидания ввода. После корректного ввода, данные сравниваются с имеющимися в бд, для исключения повторной подписки. Если данные совпадают их необходимо скорректировать (выбрать другую почту). Если совпадений обнаружено не будет, то данные о новом e-mail добавятся в БД.



Рисунок 6.1 – Диаграмма состояний подписки на новости

# Блок-схема процесса бронирования номера

Блок-схема процесса бронирования номера представлена в приложении Б (рисунок Б.1). Клиент начинает с выбора понравившегося номера, затем выбирает дополнительные сервисы, заполняет форму бронирования. Если в веденных данные есть ошибки или они не корректны, то пользователю предлагается начать процедуру заново. Если данные корректны, то на сервер отправляется соответствующий запрос. Выводится сообщение, призывающее проверить почту. После чего происходит отправка сообщения клиенту на его электронную почту.

# Блок-схема процесса подбора номера

Блок-схема процесса подбора номеров представлена в приложении Б (рисунок Б.2). Для подбора номера клиенту необходимо выбрать диапазон желаемых дат пребывания в гостинице. Указать количество человек. После чего отправляется запрос на сервер, где и производиться подбор комнат. Далее, после приема данных, происходит обработка пришедших данных с сервера и представление их клиенту.

# ОПИСАНИЕ ПРИМЕНЕНИЯ ПАТТЕРНОВ ПРОЕТИРОВАНИ

# Диаграмма классов пакета controller

Диаграмма классов представлена в приложении А на рисунках А.3 – А.6. Пакет repository (рисунок А.5) содержит интерфейсы, которые наследуются от интерфейса JpaRepository. Этот интерфейс имеет различные реализации. Компонент JpaRepository является частью фреймворка Spring Data, предназначенного для произведения различных операций с данными. Этот компонент представляет реализацию для слоя DAO (Data Access Object).

# Используемые паттерны

Шаблон проектирования или паттерн в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

В виду использования различных библиотек появляется необходимость в связывании изначально не совместимых интерфейсов. Адаптер позволяет работать вместе объектам с несовместимыми интерфейсами. Общая схема паттерна Адаптер представлена на рисунке 7.1.

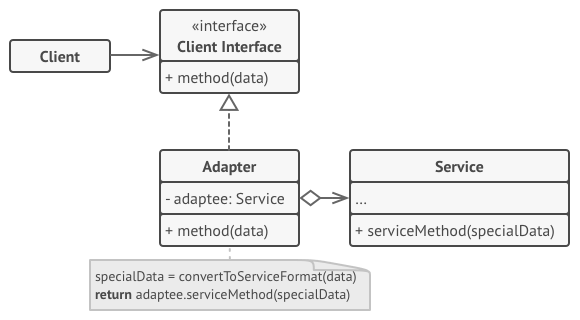


Рисунок 7.1 – Схема паттерна адаптер

Так же был использован паттерн фасад. Фасад — это простой интерфейс для работы со сложной подсистемой, содержащей множество классов. Фасад может иметь урезанный интерфейс, не имеющий 100% функциональности, которой можно достичь, используя сложную подсистему напрямую. Но он предоставляет именно тот функционал, который необходим, и скрывает все остальное. Общая схема паттерна Фасад представлена на рисунке 7.2. Данный паттерн применялся, дабы урезать функционал работы с классами БД JpaRepository.

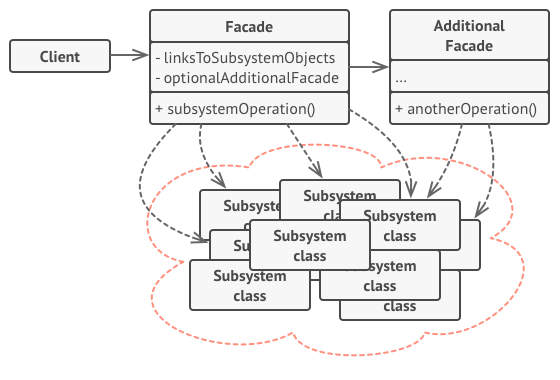


Рисунок 7.2 – Схема паттерна фасад

**Паттерн одиночка был применён для исключения дублирования класса логгера. Данный паттерн гарантирует наличие только одного экземпляра класса и глобальную точку доступа к нему. Общая структура паттерна представлена на рисунке 7.3.**

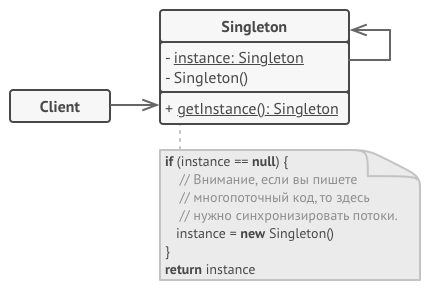


Рисунок 7.4 – Схема паттерна одиночка

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

# Руководство по развертыванию приложения

Для корректной работы сервера приложения необходимо установить JDK 1.8 и установить соединение с интернетом для автоматической подкачки необходимых библиотек. Так же необходимо установить MySql 8.0 и выполнить скрипт из приложения Г. Далее запускаем командную строку, переходим в директорию с файлом server.jar и файлом application.propertises. Выполняем следующую комманду: *java –jar* *server.jar*. При успешном выполнении команды вы увидите схожую картину с рисунком 8.1.



Рисунок 8.1 – Успешный запуск сервера

Для запуска клиентской части Необходим вспомогательный файл *jetty-runner.jar.* Далее выполнить команду *java -jar [путь к файлу jetty-runner.jar] --port 8083 [путь к файлу client.war].* При успешном запуске вы увидите надпись как на рисунке 8.2. После этого в стоке браузера переходим по адресу <http://localhost:63342/mainAdmin.html>.

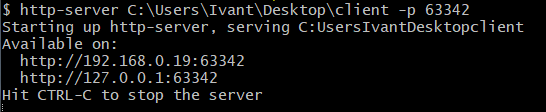


Рисунок 8.2 – Успешный запуск клиента

# Руководство использования приложения

При запуске приложения нас встречает окно приветствия, на котором можно сразу перейти на поиск доступных комнат (рисунок 8.3). Для поиска необходимо выбрать дату и нажать на кнопку проверить. На этой же страницы можно перейти в почтовый клиент при нажатии на e-mail адрес или посмотреть наши страницы в социальных сетях vk или twitter. Так же на этой страницы есть возможность произвести подписку на новостную рассылку нашей гостиницы (рисунок 8.4). Для того, чтобы произвести подписку нужно ввести свой e-mail в соответствующее поле и нажать на кнопку подписаться.

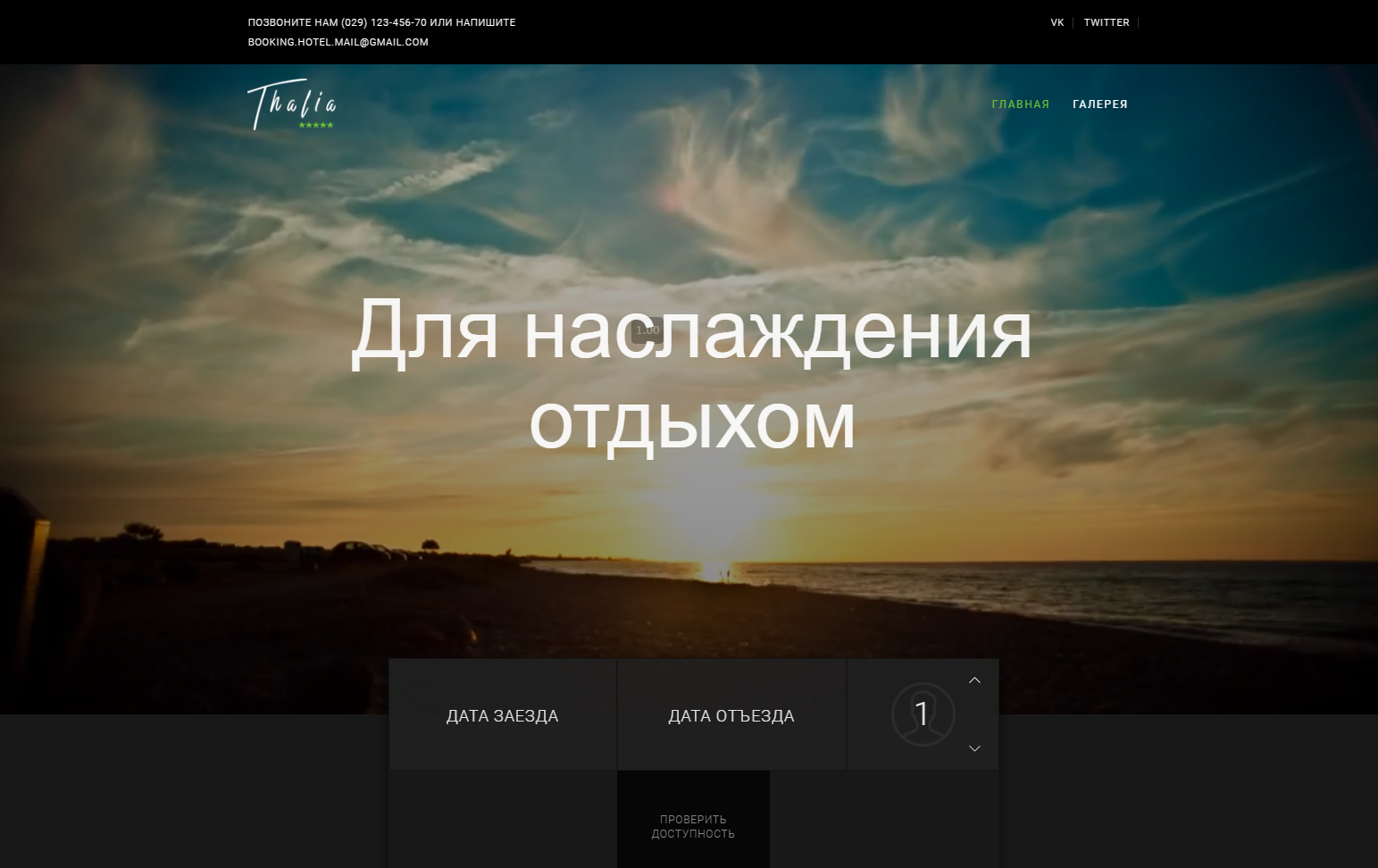


Рисунок 8.3 – Окно приветствия

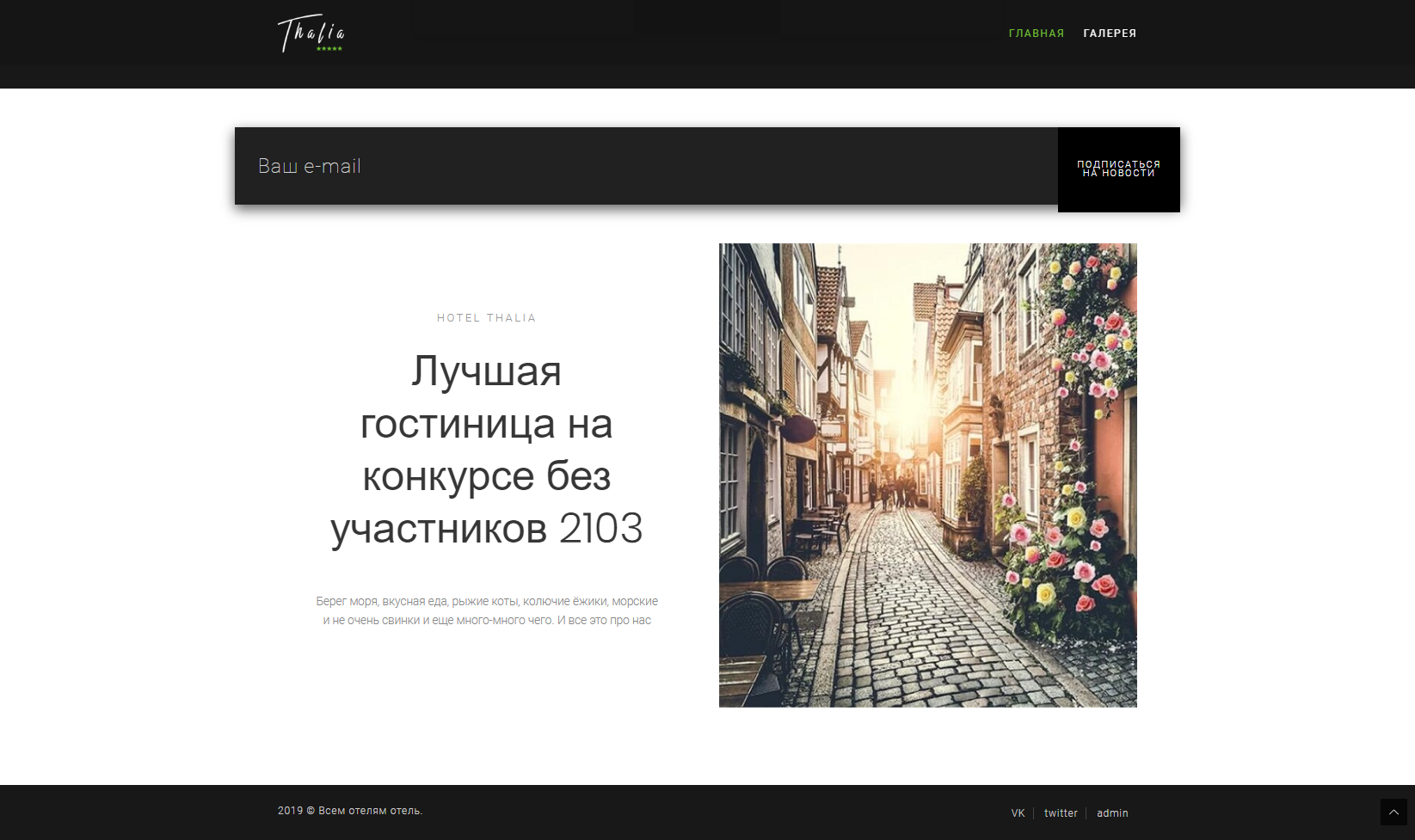


Рисунок 8.4 – Страница с возможностью оформления подписки

На рисунке 8.5 представлен вид нашей галереи, тут же есть возможность просмотреть ролик с нашей презентацией.

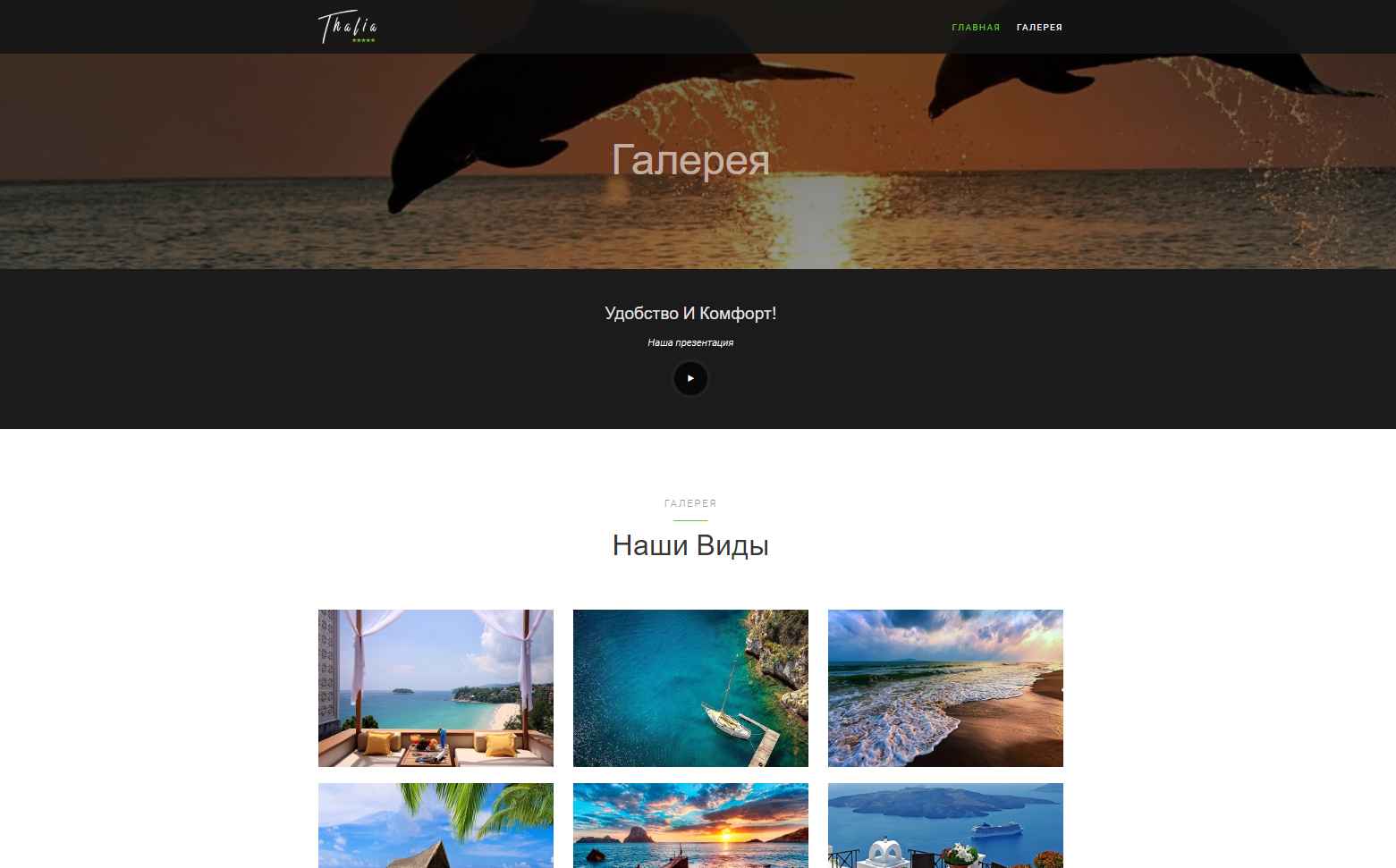


Рисунок 8.5 – Галерея

При подписке на обновления на вашу почту придет письмо следующего содержания (рисунок 8.6).

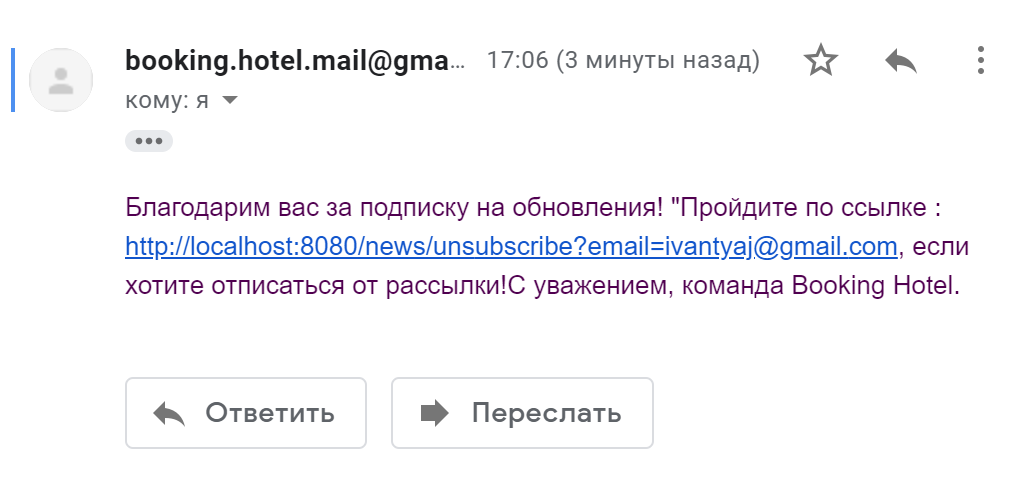


Рисунок 8.6 – Пример письма

Для осуществления бронирования (рисунок 8.7) клиент обязан ввести свое ФИО, e-mail, на который и будут приходить письма и телефон. Так же он может оставить сообщение, которое отобразиться у администратора, который отслеживает заявки. Клиент может выбрать вариант оплаты на месте или ввести данные своей карты сразу. При бронировании можно, при необходимости, заказать обратный звонок, заявка на который также отобразится на странице администратора. Так же вам на почту придет письмо с подтверждением бронирования.

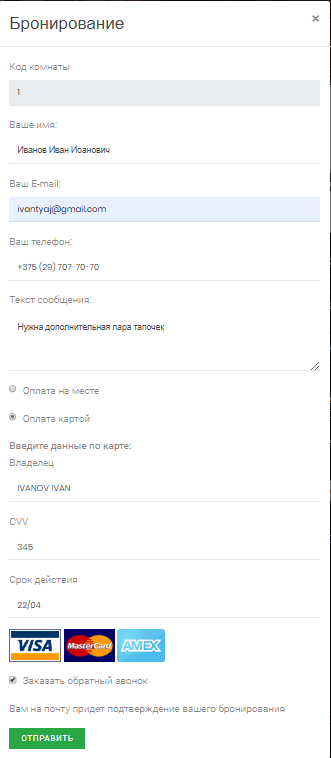


Рисунок 8.7 – Оформление заявки на бронирование

При входе на страницу администратора вам будет необходимо ввести соответствующий пароль (рисунок 8.8).

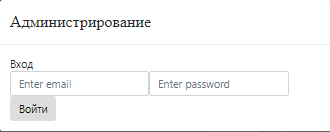


Рисунок 8.8 – Вход на страницу администрирования

На странице администратора можно наблюдать список клиентов, которым необходимо перезвонить (рисунок 8.9). После пометки, что звонок осуществлен, клиент пропадет из данного списка.



Рисунок 8.9 – Клиенты запросившие обратный звонок

Данные статистики (рисунок 8.10), которая отображает статистику цен на номера и статистику заказов по месяцам.

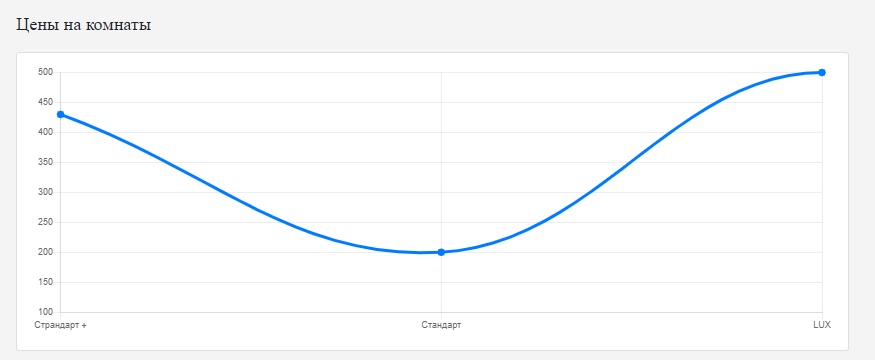


Рисунок 8.10 – Цены на комнаты

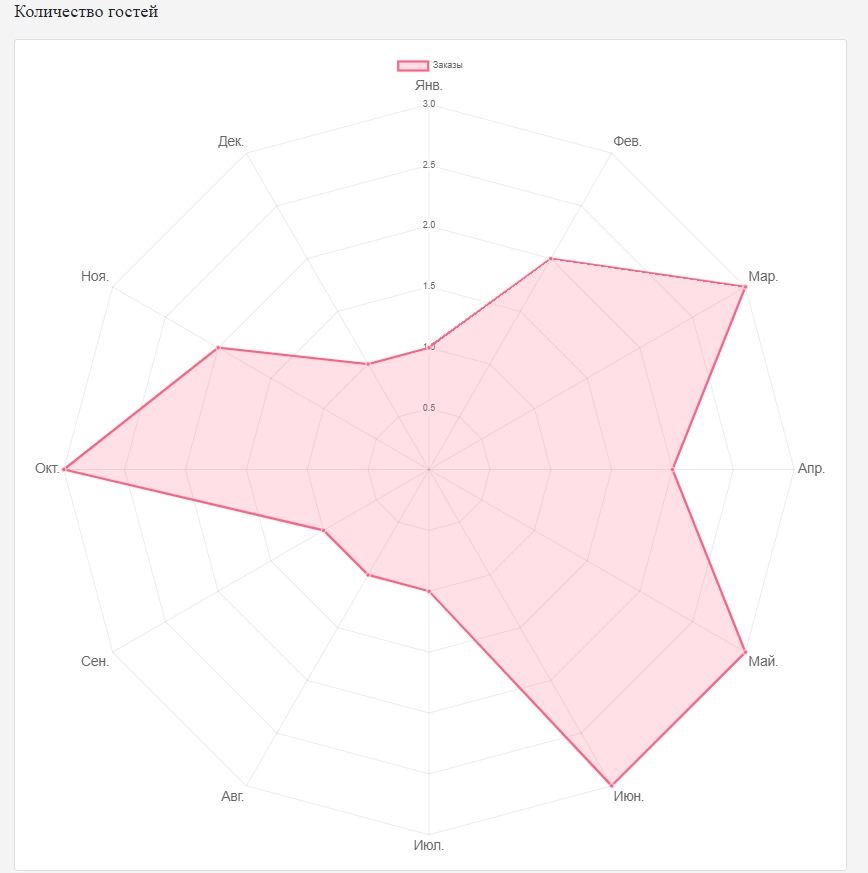


Рисунок 8.11 – Статистика заказов по месяцам

Возможность отмены конкретного запроса на бронирование с возможностью отправки сопутствующего сообщения на e-mail клиента (рисунок 8.11). Для этого необходимо написать причину в соответствующем поле и нажать кнопку отменить бронирование.

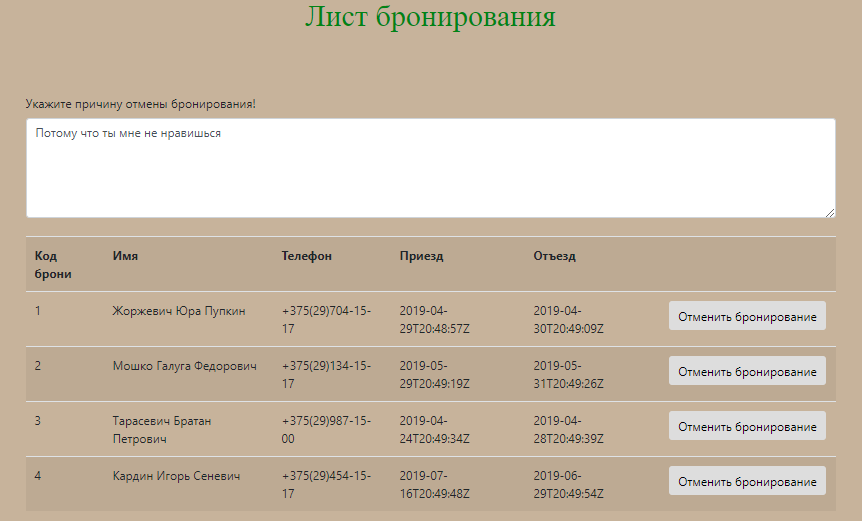


Рисунок 8.12 – Лист бронирования

Возможность изменения данных о комнатах (рисунок 8.12). Возможность менять цену и описание дополняется возможностью менять используемое фото, которое сохраняется в виде ссылки на фотографию в сети интернет.

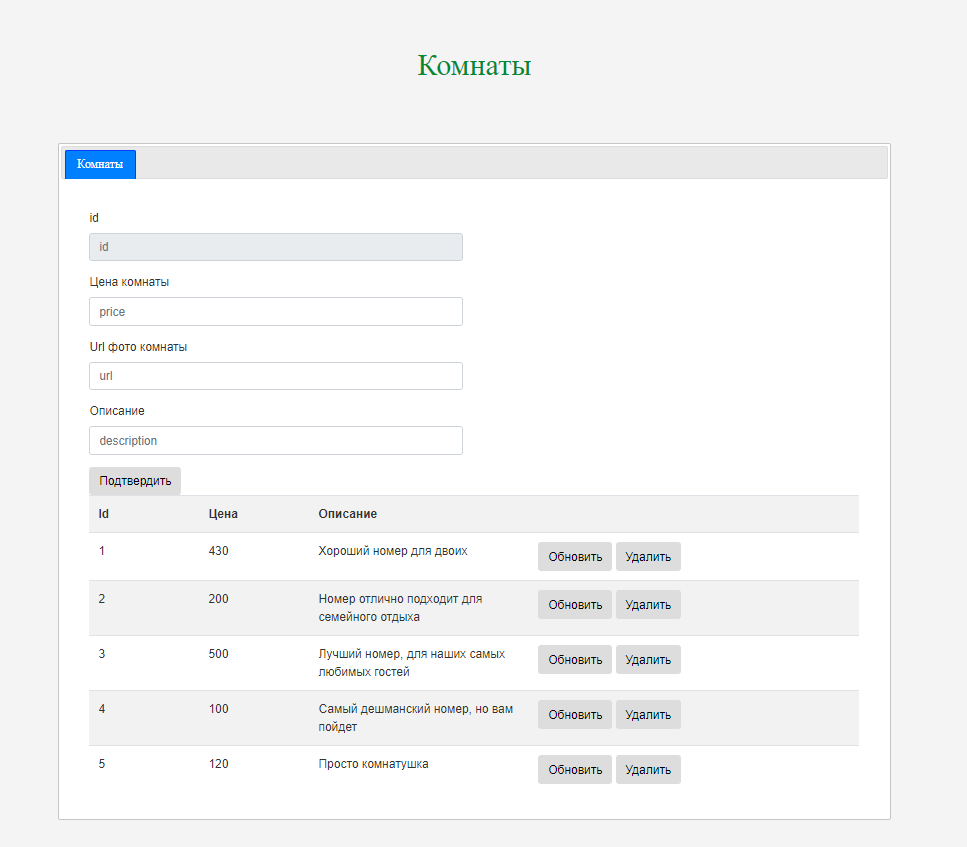


Рисунок 8.13 – Данные о комнатах

Возможность проведения рассылки новостей всем подписавшимся (рисунок 8.13). Данное меню содержит поле для сообщение которое будет разослано и кнопку, при нажатию на которую это сообщение отправится всем, кто оформил подписку на новости.

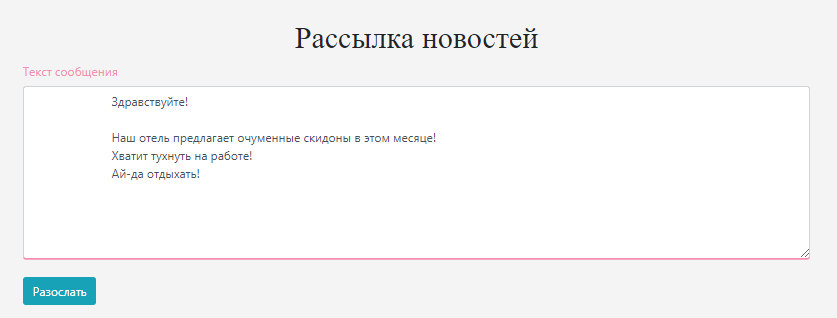


Рисунок 8.14 – Рассылка сообщений

# ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта была создан сайта для управления гостиницей с возможностью бронирования номеров. Была создана база данных для хранения всей необходимой информации гостиницы. Данное web-приложение позволяет клиентам быстро, без лишних действий получить всю актуальную информацию и забронировать номер на определенные даты.

В процессе написания курсового проекта были решены следующие задачи:

* изучены сервисы бронирование гостиницы;
* разработаны объектная и функциональная модели для системы управления гостиницей с возможностью бронирования номеров, спроектированы диаграмма классов и схемы алгоритмов;
* разработана архитектура программы;
* разработан удобный пользовательский интерфейс для работы с сайтом;
* протестирована программа.

Можно сделать вывод, что все цели и задачи данного курсового проекта были выполнены, и программа соответствуют всем требованиям.

Исходя из вышеперечисленного, можно сделать вывод, что разработанная программа является универсальной и актуальной.

Архитектура клиент-сервер позволяет пользователем общаться с одним сервером и общей базой данных, что позволяет в разы ускорить взаимодействие, а также разгрузить клиентскую машину, что снижает общие требования к системе.

Приложение, разработанное в данном курсовом проекте, отвечает главным требованиям к современным продуктам: простота использования не подготовленным пользователем, эффективность и безотказность.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] FB.ru [Электронный ресурс]: Роль Интернета в нашей жизни. – Электронные данные. – Режим доступа: <http://fb.ru/article/372002/rol-interneta-v-nashey-jizni>

[2] Понятия и категории [Электронный ресурс]: Преимущества заказа пиццы с помощью Интернета. – Электронные данные. – Режим доступа: <http://ponjatija.ru/obshepit/preimushchestva-zakaza-piccy-s-pomoshchyu-interneta>

[3] habrahabr [Электронный ресурс]: Паттерны проектирования на javascript. – Электронные данные. – Режим доступа: <https://habrahabr.ru>

[4] Преимущества Java [Электронный ресурс]. – Режим доступа: <https://www.ibm.com/support/knowledgecenter/ru/ssw_aix_61/com.ibm.aix.performance/advantages_java.htm.>

[5] Язык программирования Java [Электронный ресурс]. -

Режим доступа: <https://dic.academic.ru/dic.nsf/ruwiki/>

[6] Разработка приложений со Spring Boot [Электронный ресурс]. -

Режим доступа: <http://pvtjoke.blogspot.com.by/2016/03/spring-boot-1-spring-boot.html>

[7] Что такое Maven? [Электронный ресурс]. – Режим доступа: <https://jsehelper.blogspot.com.by/2016/05/maven-1.html>

[8] Библиотека Hibernate [Электронный ресурс]. – Режим доступа:

<http://java-online.ru/libs-hibernate.xhtml>

[9] Что такое Hibernate [Электронный ресурс]. – Режим доступа:

<https://jsehelper.blogspot.com.by/2016/01/object-relational-mapping-orm-hibernate.html?m=1>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Диаграммы**

****

Рисунок А.1 – Диаграмма вариантов использования

Продолжение приложения А

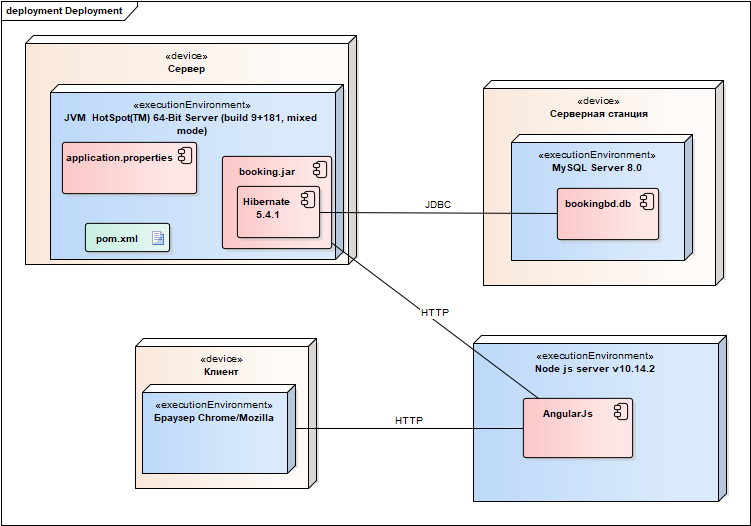


Рисунок А.2 – Диаграмма развертывания

Продолжение приложения А

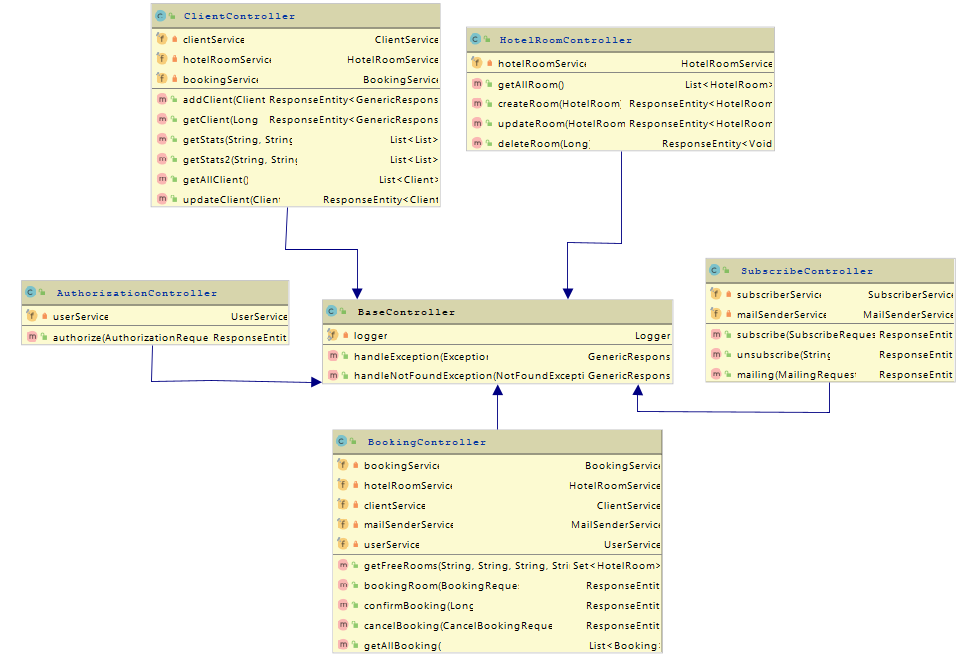


Рисунок А.3 – Диаграмма классов пакета controller

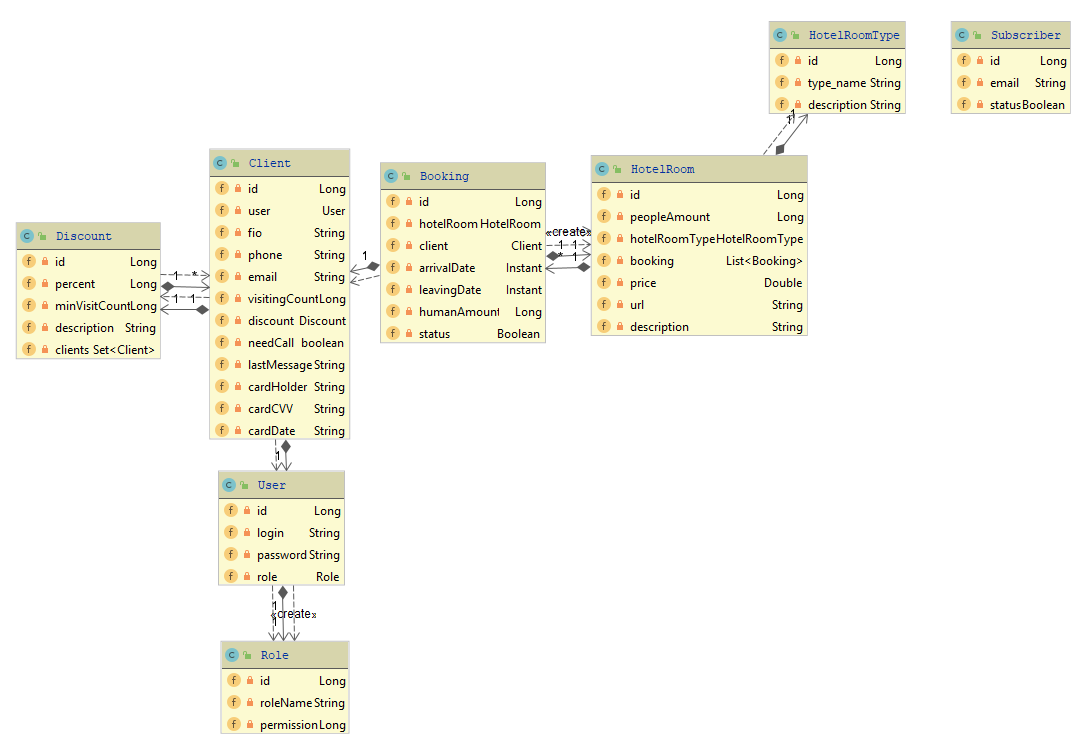


Рисунок А.4 – Диаграмма классов пакета entity

Продолжение приложения А

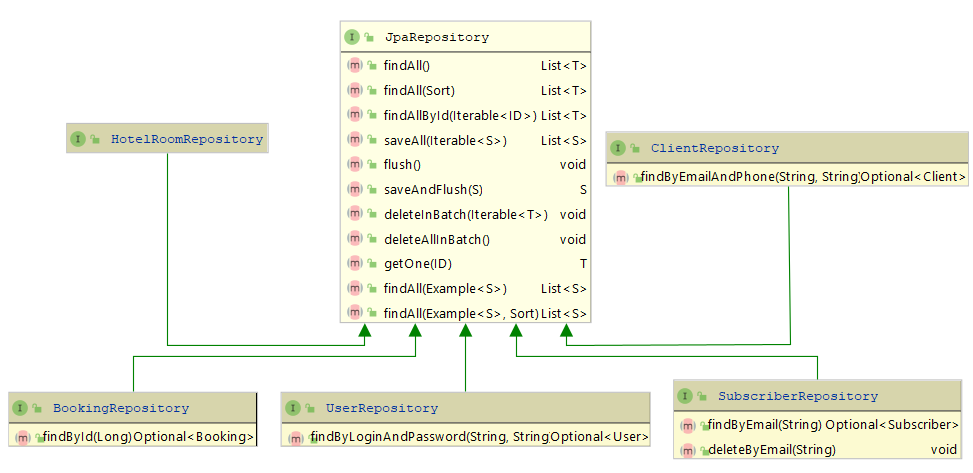


Рисунок А.5 – Диаграмма классов пакета repository

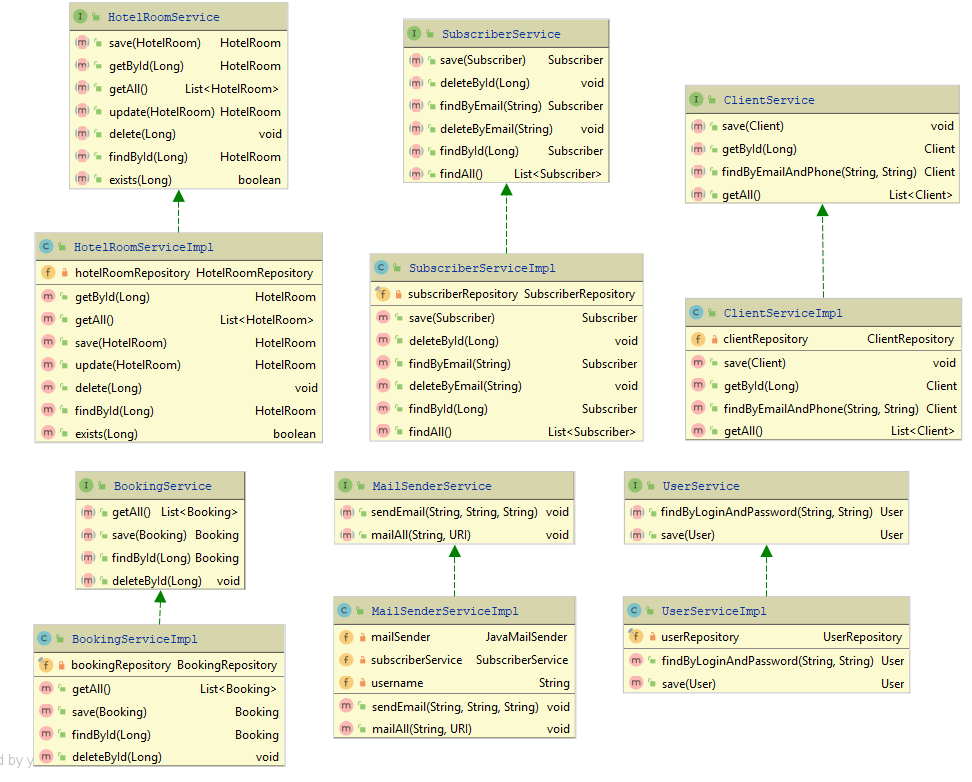


Рисунок А.6 – Диаграмма классов пакета service

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Схемы алгоритмов**

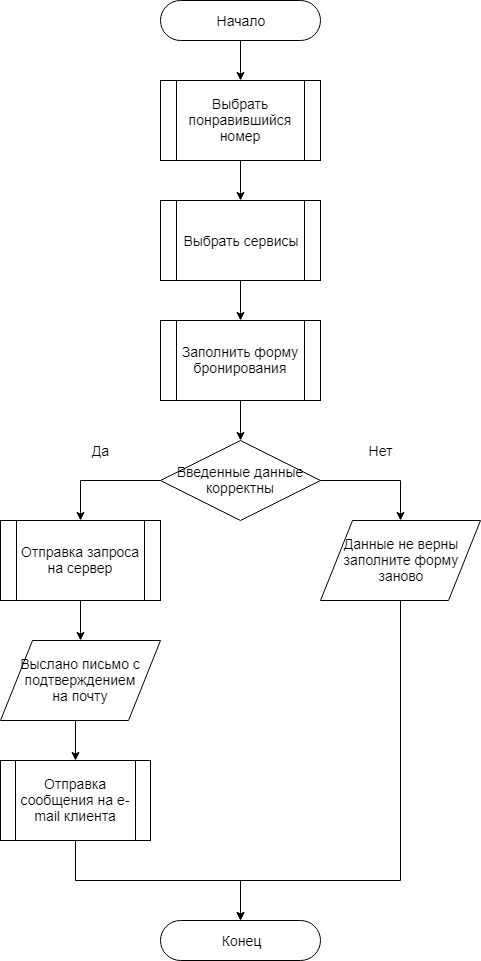


Рисунок Б.1 – Блок-схема процесса бронирования номера

Продолжение приложения Б

Продолжение приложения Б



Рисунок Б.2 – Блок-схема процесса подбора номеров

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Листинг кода**

**package** com.booking\_maiseyenka\_stepovoi.app;  
  
**import** com.booking\_maiseyenka\_stepovoi.config.DatabaseConfig;  
**import** com.booking\_maiseyenka\_stepovoi.utils.logging.LoggerConfig;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.context.annotation.Import;  
  
@SpringBootApplication  
@Import({DatabaseConfig.**class**, LoggerConfig.**class**})  
**public class** Application {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(Application.**class**, args);  
 }  
}

**package** com.booking\_maiseyenka\_stepovoi.config;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.core.env.Environment;  
**import** org.springframework.data.jpa.repository.config.EnableJpaRepositories;  
**import** org.springframework.jdbc.datasource.DriverManagerDataSource;  
**import** org.springframework.orm.jpa.JpaTransactionManager;  
**import** org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;  
**import** org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  
**import** org.springframework.transaction.PlatformTransactionManager;  
**import** org.springframework.transaction.annotation.EnableTransactionManagement;  
  
**import** javax.sql.DataSource;  
**import** java.util.Properties;  
  
@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(basePackages =

Продолжение приложения В

{**"com.booking\_maiseyenka\_stepovoi.model.repository"**})

**public class** DatabaseConfig {  
  
 **private final** Environment **env**;  
  
 @Autowired  
 **public** DatabaseConfig(Environment env) {  
 **this**.**env** = env;  
 }  
  
 @Bean  
 **public** LocalContainerEntityManagerFactoryBean entityManagerFactory() {  
 LocalContainerEntityManagerFactoryBean sessionFactory = **new** LocalContainerEntityManagerFactoryBean();  
 sessionFactory.setJpaVendorAdapter(**new** HibernateJpaVendorAdapter());  
 sessionFactory.setDataSource(dataSource());  
 sessionFactory.setPackagesToScan(**"com.booking\_maiseyenka\_stepovoi.model.entity"**);  
 sessionFactory.setJpaProperties(hibernateProperties());  
 **return** sessionFactory;  
 }  
  
  
 @Bean  
 **public** DataSource dataSource() {  
 String dbDriver = getProperty(**"db.driver"**);  
 String dbUrl = getProperty(**"db.url"**);  
 DriverManagerDataSource dataSource = **new** DriverManagerDataSource();  
 dataSource.setDriverClassName(dbDriver);  
 dataSource.setUrl(dbUrl);  
 dataSource.setUsername(getProperty(**"db.username"**));  
 dataSource.setPassword(getProperty(**"db.password"**));  
 **return** dataSource;  
 }  
  
 @Bean  
 **public** PlatformTransactionManager transactionManager() {  
 **return new** JpaTransactionManager(  
 entityManagerFactory().getObject());  
 }  
  
 **private** Properties hibernateProperties() {  
 Properties hibernateProperties = **new** Properties();  
 hibernateProperties.setProperty(  
 **"hibernate.hbm2ddl.auto"**, getProperty(**"hibernate.hbm2ddl.auto"**));

Продолжение приложения В

hibernateProperties.setProperty(  
 **"hibernate.dialect"**, getProperty(**"hibernate.dialect"**));  
 hibernateProperties.setProperty(  
 **"hibernate.show\_sql"**, **"false"**);  
  
 **return** hibernateProperties;  
 }  
  
 **private** String getProperty(String s) {  
 **return env**.getRequiredProperty(s);  
 }  
}

**package** com.booking\_maiseyenka\_stepovoi.controller;  
  
**import** com.booking\_maiseyenka\_stepovoi.exceptions.NotFoundException;  
**import** com.booking\_maiseyenka\_stepovoi.utils.logging.GenericResponse;  
  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.web.bind.annotation.ExceptionHandler;  
**import** org.springframework.web.bind.annotation.ResponseStatus;  
  
**public class** BaseController {  
  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(BaseController.**class**);  
  
 @ExceptionHandler(Exception.**class**)  
 @ResponseStatus(value = HttpStatus.***INTERNAL\_SERVER\_ERROR***)  
 **public** GenericResponse handleException(Exception e) {  
 ***logger***.error(**"Внутренняя ошибка системы"**, e);  
 **return new** GenericResponse(**"Внутренняя ошибка системы"**,  
 **""**);  
 }  
  
 @ExceptionHandler(NotFoundException.**class**)  
 @ResponseStatus(value = HttpStatus.***BAD\_REQUEST***)  
 **public** GenericResponse handleNotFoundException(NotFoundException e) {  
 **return new** GenericResponse(**"Ошибка"**,  
 e.getMessage());  
 }  
  
  
}

Продолжение приложения В

**package** com.booking\_maiseyenka\_stepovoi.controller;  
  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.\*;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.\*;  
**import** io.swagger.annotations.Api;  
**import** io.swagger.annotations.ApiOperation;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.util.UriComponentsBuilder;  
  
**import** java.net.URI;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
**import** java.util.HashSet;  
**import** java.util.List;  
**import** java.util.Set;  
**import** java.util.stream.Collectors;  
  
@RestController(value = **"BookingController"**)  
@CrossOrigin(exposedHeaders = **"Access-Control-Allow-Origin"**)  
@RequestMapping(value = **"/hotel/booking"**)  
@Api(tags = **"booking"**)  
**public class** BookingController **extends** BaseController {  
  
 **private final** BookingService **bookingService**;  
 **private final** HotelRoomService **hotelRoomService**;  
 **private final** ClientService **clientService**;  
 **private final** MailSenderService **mailSenderService**;  
 **private final** UserService **userService**;  
  
 @Autowired  
 **public** BookingController(BookingService bookingService, HotelRoomService hotelRoomService, ClientService clientService, MailSenderService mailSenderService, UserService userService) {  
 **this**.**bookingService** = bookingService;  
 **this**.**hotelRoomService** = hotelRoomService;  
 **this**.**clientService** = clientService;  
 **this**.**mailSenderService** = mailSenderService;  
 **this**.**userService** = userService;  
 }  
  
 @ApiOperation(value = **"find free"**, response = Set.**class**, notes = **"find\_free"**)  
 @GetMapping(value = **"/searchFree"**)  
 @ResponseBody  
 **public** Set<HotelRoom> getFreeRooms(@RequestParam String

Продолжение приложения В

startDate,  
 @RequestParam String endDate,  
 @RequestParam String clients,  
 @RequestParam String price) **throws** ParseException {  
 System.***err***.println(startDate + **" "** + endDate + **" "** + clients + **" "** + price);  
  
 List<Booking> bookingList = **bookingService**.getAll();  
 List<HotelRoom> hotelRoomList = **hotelRoomService**.getAll();  
  
 Set<HotelRoom> notFreeRooms = **new** HashSet<>();  
 Set<HotelRoom> freeRooms = **new** HashSet<>();  
  
 SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"dd.MM.yyyy"**);  
 Date dateStart = dateFormat.parse(startDate);  
 Date dateEnd = dateFormat.parse(endDate);  
  
 **for** (Booking booking : bookingList) {  
 **if** ((dateStart.after(Date.*from*(booking.getArrivalDate())) && dateStart.before(Date.*from*(booking.getLeavingDate()))) ||  
 (dateEnd.after(Date.*from*(booking.getArrivalDate())) && dateStart.before(Date.*from*(booking.getLeavingDate())))){  
 notFreeRooms.add(booking.getHotelRoom());  
 }  
 }  
  
 **for**(HotelRoom room : hotelRoomList){  
 **if**(!notFreeRooms.contains(room)){  
 freeRooms.add(room);  
 }  
 }  
  
 System.***err***.println(notFreeRooms.stream().map(HotelRoom::getId).collect(Collectors.*toSet*()));  
 System.***err***.println(freeRooms.stream().map(HotelRoom::getId).collect(Collectors.*toSet*()));  
  
 **return** freeRooms.stream()  
 .filter(room -> room.getPeopleAmount() >= Long.*parseLong*(clients))  
 .filter(room -> room.getPrice() <=

Продолжение приложения В

Double.*parseDouble*(price))  
 .collect(Collectors.*toSet*());  
 }  
  
 @ApiOperation(value = **"booking\_maiseyenka\_stepovoi room"**, response = ResponseEntity.**class**, notes = **"booking\_room"**)  
 @PostMapping(value = **"/booking"**, produces = **"application/json"**, consumes = **"application/json"**)  
 **public** ResponseEntity bookingRoom(@RequestBody BookingRequest bookingRequest) **throws** ParseException {  
 *//новый клиент если такого нет* System.***err***.println(bookingRequest);  
 Client client = **clientService**.findByEmailAndPhone(bookingRequest.getClientEmail(),  
 bookingRequest.getClientPhone());  
 System.***err***.println(client);  
 **if** (client.getVisitingCount() == **null**) {  
 client.setVisitingCount(1L);  
 client.setFio(bookingRequest.getClientName());  
 User user = **userService**.save(**new** User(client.getEmail(), client.getEmail(), 2L));  
 client.setUser(user);  
 client.setLastMessage(bookingRequest.getMessage());  
 client.setNeedCall(bookingRequest.isNeedBackCall());  
 client.setCardCVV(bookingRequest.getCardCVV());  
 client.setCardDate(bookingRequest.getCardData());  
 client.setCardHolder(bookingRequest.getCardHolder());  
 **clientService**.save(client);  
 } **else** {  
 client.setCardCVV(bookingRequest.getCardCVV());  
 client.setCardDate(bookingRequest.getCardData());  
 client.setCardHolder(bookingRequest.getCardHolder());  
 **clientService**.save(client);  
 }  
  
 SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"dd.MM.yyyy"**);  
 Date dateFrom = dateFormat.parse(bookingRequest.getDataFrom());  
 Date dateTo = dateFormat.parse(bookingRequest.getDataTo());  
 HotelRoom hotelRoom = **hotelRoomService**.findById(bookingRequest.getRoomId());  
 Booking booking = **new** Booking(bookingRequest.getRoomId(),  
 client,  
 dateFrom.toInstant(),

Продолжение приложения В

dateTo.toInstant(),  
 hotelRoom.getPeopleAmount());  
 System.***err***.println(client);  
 Booking saved = **bookingService**.save(booking);  
 *//ссылка на подтверждение бронирования* URI uri = URI.*create*(**"http://localhost:8080/hotel/booking\_maiseyenka\_stepovoi/confirm"**);  
 URI link = UriComponentsBuilder.*fromUri*(uri)  
 .queryParam(**"id"**, saved.getId()).build().toUri();  
 **mailSenderService**.sendEmail(booking.getClient().getEmail(), **"Confirm booking\_maiseyenka\_stepovoi"**, **"Перейдите по ссылке, для подтверждения бронирования :"** + link);  
  
*// String message = "Благодарим вас за успешное бронирование номера. Информация о бронировании :\n" + bookingRequest + "\n С уважением, команда Booking Hotel.";  
// mailSenderService.sendEmail(client.getEmail(), "Booked Room", message);* **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
  
 @ApiOperation(value = **"confirming booking\_maiseyenka\_stepovoi"**, response = ResponseEntity.**class**, notes = **"confirm\_booking"**)  
 @GetMapping(value = **"/confirm"**)  
 **public** ResponseEntity confirmBooking(@RequestParam(**"id"**) Long id) {  
 Booking booking = **bookingService**.findById(id);  
 booking.setStatus(**true**);  
 **bookingService**.save(booking);  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
 @ApiOperation(value = **"cancel booking\_maiseyenka\_stepovoi"**, response = ResponseEntity.**class**, notes = **"cancel\_booking"**)  
 @PostMapping(value = **"/cancel"**, produces = **"application/json"**, consumes = **"application/json"**)  
 **public** ResponseEntity cancelBooking(@RequestBody CancelBookingRequest request) {  
 Booking booking = **bookingService**.findById(request.getId());  
 **bookingService**.deleteById(request.getId());  
 **mailSenderService**.sendEmail(booking.getClient().getEmail(), **"Cancel booking\_maiseyenka\_stepovoi"**, request.getMessage());  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
 @ApiOperation(value = **"get all bookings"**, response =

Продолжение приложения В

Set.**class**, notes = **"get\_all\_bookings"**)

@GetMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** List<Booking> getAllBooking() {  
 **return bookingService**.getAll();  
 }  
 *//post : id\_booking , message(cause) - когда приходит удалить из бд бронь и прислать клиенту сообщение*}

**package** com.booking\_maiseyenka\_stepovoi.controller;  
  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.Booking;  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.Client;  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.HotelRoom;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.BookingService;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.ClientService;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.HotelRoomService;  
**import** com.booking\_maiseyenka\_stepovoi.utils.logging.GenericResponse;  
**import** io.swagger.annotations.Api;  
**import** io.swagger.annotations.ApiOperation;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** javax.persistence.EntityNotFoundException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
  
@RestController(value = **"ClientController"**)  
@CrossOrigin(exposedHeaders = **"Access-Control-Allow-Origin"**)  
@RequestMapping(value = **"/hotel/clients"**)  
@Api(tags = **"client"**)  
**public class** ClientController **extends** BaseController {  
  
 **private final** ClientService **clientService**;  
 **private final** HotelRoomService **hotelRoomService**;  
 **private final** BookingService **bookingService**;  
  
 @Autowired  
 **public** ClientController(ClientService clientService, HotelRoomService hotelRoomService, BookingService bookingService) {

Продолжение приложения В

**this**.**clientService** = clientService;  
 **this**.**hotelRoomService** = hotelRoomService;  
 **this**.**bookingService** = bookingService;  
 }  
  
 @ApiOperation(value = **"Create client"**, response = ResponseEntity.**class**, notes = **"client\_create"**)  
 @PostMapping(value = **"/create"**)  
 **public** ResponseEntity<GenericResponse> addClient(@RequestBody Client client) {  
 System.***err***.println(client);  
 **clientService**.save(client);  
 **return new** ResponseEntity<>(**new** GenericResponse(**"message"**), HttpStatus.***OK***);  
 }  
  
 @ApiOperation(value = **"Get client"**, response = ResponseEntity.**class**, notes = **"get\_client"**)  
 @GetMapping(value = **"/{id}"**)  
 **public** ResponseEntity<GenericResponse> getClient(@PathVariable(**"id"**) Long id) {  
 Client client = **clientService**.getById(id);  
 **return new** ResponseEntity<>(**new** GenericResponse(**"Client "** + client), HttpStatus.***OK***);  
 }  
  
 @ApiOperation(value = **"Get stats"**, response = List.**class**, notes = **"get\_stats"**)  
 @GetMapping(value = **"/statsClient"**)  
 @ResponseBody  
 **public** List<List> getStats(@RequestParam String startDate,  
 @RequestParam String endDate) {  
  
  
 List<HotelRoom> hotelRoomList = **hotelRoomService**.getAll();  
 List<Double> data1 = hotelRoomList.stream().map(HotelRoom::getPrice).collect(Collectors.*toList*());  
 Set<String> headers = hotelRoomList.stream().map(room -> room.getHotelRoomType().getType\_name()).collect(Collectors.*toSet*());  
*// List<Long> data1 = clientList.stream().map(Client::getVisitingCount).collect(Collectors.toList());  
// List<Long> data2 = Collections.singletonList(clientList.stream().filter(client -> client.getDiscount() != null).count());*

Продолжение приложения В

ArrayList<List> toSend = **new** ArrayList<>();  
 toSend.add(**new** ArrayList<>(headers));  
 toSend.add(data1);  
*// toSend.add(data2);* **return** toSend;  
 }  
  
 @ApiOperation(value = **"Get stats2"**, response = List.**class**, notes = **"get\_stats2"**)  
 @GetMapping(value = **"/statsClient2"**)  
 @ResponseBody  
 **public** List<List> getStats2(@RequestParam String startDate,  
 @RequestParam String endDate) {  
  
 List<Booking> bookingList = **bookingService**.getAll();  
 List<Date> dates = bookingList.stream()  
 .map(booking -> Date.*from*(booking.getArrivalDate()))  
 .sorted().collect(Collectors.*toList*());  
  
 *//dates.forEach(System.err::println);* SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"Mmm"**);  
  
 List<String> monthes = Arrays.*asList*(**"Янв."**, **"Фев."**, **"Мар."**, **"Апр."**, **"Май."**, **"Июн."**, **"Июл."**, **"Авг."**, **"Сен."**, **"Окт."**, **"Ноя."**, **"Дек."**);  
  
 List<Integer> monthesNum = Arrays.*asList*(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11);  
  
 List<Long> counts = **new** LinkedList<>();  
  
 **for** (Integer month : monthesNum) {  
 counts.add(dates.stream().map(date -> date.getMonth() == month).filter(Boolean::booleanValue).count());  
 }  
 System.***out***.println(**"count\n"** + counts);  
  
 List<List> toSend = **new** ArrayList<>();  
 toSend.add(monthes);  
  
 toSend.add(counts);  
  
 **return** toSend;  
 }

Продолжение приложения В

@ApiOperation(value = **"get all clients"**, response = ResponseEntity.**class**, notes = **"get\_all\_clients"**)  
 @GetMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** List<Client> getAllClient() {  
 **return clientService**.getAll();  
 }  
  
 @ApiOperation(value = **"updating client"**, response = ResponseEntity.**class**, notes = **"update\_client"**)  
 @PutMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** ResponseEntity<Client> updateClient(@RequestBody Client client) {  
 System.***out***.println(**"client Hello update"**);  
 **if** (client.getId() == **null**) {  
 **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  
 }  
 **try** {  
 Client clientNew = **clientService**.getById(client.getId());  
 clientNew.setUser(client.getUser());  
 clientNew.setFio(client.getFio());  
 clientNew.setPhone(client.getPhone());  
 clientNew.setEmail(client.getEmail());  
 clientNew.setVisitingCount(client.getVisitingCount());  
 clientNew.setDiscount(client.getDiscount());  
 clientNew.setNeedCall(client.isNeedCall());  
 clientNew.setLastMessage(client.getLastMessage());  
  
 **clientService**.save(clientNew);  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 } **catch** (EntityNotFoundException e) {  
 **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  
 }  
 }  
}

**package** com.booking\_maiseyenka\_stepovoi.controller;  
  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.HotelRoom;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.HotelRoomService;  
**import** io.swagger.annotations.Api;  
**import** io.swagger.annotations.ApiOperation;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.http.ResponseEntity;

Продолжение приложения В

**import** org.springframework.web.bind.annotation.\*;  
**import** javax.persistence.EntityExistsException;  
**import** javax.persistence.EntityNotFoundException;  
**import** java.net.URI;  
**import** java.util.List;  
  
@RestController(value = **"RoomController"**)  
@CrossOrigin(exposedHeaders = **"Access-Control-Allow-Origin"**)  
@RequestMapping(value = **"/hotel/rooms"**)  
@Api(tags = **"hotel"**)  
**public class** HotelRoomController **extends** BaseController {  
  
 **private final** HotelRoomService **hotelRoomService**;  
  
 @Autowired  
 **public** HotelRoomController(HotelRoomService hotelRoomService) {  
 **this**.**hotelRoomService** = hotelRoomService;  
 }  
  
 @ApiOperation(value = **"get all rooms"**, response = ResponseEntity.**class**, notes = **"get\_all\_room"**)  
 @GetMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** List<HotelRoom> getAllRoom() {  
 **return hotelRoomService**.getAll();  
 }  
  
 @ApiOperation(value = **"create room"**, response = ResponseEntity.**class**, notes = **"create\_room"**)  
 @PostMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** ResponseEntity<HotelRoom> createRoom(@RequestBody HotelRoom employee) {  
 **try** {  
 HotelRoom result = **hotelRoomService**.save(employee);  
 URI uri = URI.*create*(**"http://localhost:8080/hotel/rooms"** + result.getId());  
 **return** ResponseEntity.*created*(uri).body(result);  
 } **catch** (EntityExistsException e) {  
 **return new** ResponseEntity<>(HttpStatus.***CONFLICT***);  
 }  
 }  
 @ApiOperation(value = **"updating room"**, response = ResponseEntity.**class**, notes = **"update\_room"**)  
 @PutMapping(value = **""**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** ResponseEntity<HotelRoom> updateRoom(@RequestBody HotelRoom employee) {

Продолжение приложения В

**if** (employee.getId() == **null**) {  
 System.***out***.println(**"ne Hello update"**);  
 **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  
 }  
 **try** {  
 HotelRoom hotelRoom = **hotelRoomService**.findById(employee.getId());  
 hotelRoom.setPrice(employee.getPrice());  
 hotelRoom.setPeopleAmount(employee.getPeopleAmount());  
 hotelRoom.setUrl(employee.getUrl());  
 hotelRoom.setDescription(employee.getDescription());  
 **hotelRoomService**.save(hotelRoom);  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 } **catch** (EntityNotFoundException e) {  
 **return new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);  
 }  
 }  
  
 @ApiOperation(value = **"deleting room"**, response = ResponseEntity.**class**, notes = **"delete\_room"**)  
 @DeleteMapping(value = **"{id}"**, produces = MediaType.***APPLICATION\_JSON\_VALUE***)  
 **public** ResponseEntity<Void> deleteRoom(@PathVariable Long id) {  
 **hotelRoomService**.delete(id);  
 **return** ResponseEntity.*ok*().build();  
 }  
}

**package** com.booking\_maiseyenka\_stepovoi.controller;  
  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.MailingRequest;  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.SubscribeRequest;  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.Subscriber;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.MailSenderService;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.SubscriberService;  
**import** com.booking\_maiseyenka\_stepovoi.utils.logging.GenericResponse;  
**import** io.swagger.annotations.Api;  
**import** io.swagger.annotations.ApiOperation;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.util.UriComponentsBuilder;

Продолжение приложения В

**import** java.net.URI;  
**import** java.util.List;  
  
@RestController(value = **"SubscribeController"**)  
@CrossOrigin(exposedHeaders = **"Access-Control-Allow-Origin"**)  
@RequestMapping(value = **"/news"**)  
@Api(tags = **"news"**)  
**public class** SubscribeController **extends** BaseController {  
  
 **private final** SubscriberService **subscriberService**;  
 **private final** MailSenderService **mailSenderService**;  
  
 @Autowired  
 **public** SubscribeController(SubscriberService subscriberService, MailSenderService mailSenderService) {  
 **this**.**subscriberService** = subscriberService;  
 **this**.**mailSenderService** = mailSenderService;  
  
 }  
  
 @ApiOperation(value = **"subscribe"**, response = ResponseEntity.**class**, notes = **"subscribing"**)  
 @PostMapping(value = **"/subscribe"**, produces = **"application/json"**, consumes = **"application/json"**)  
 **public** ResponseEntity subscribe(@RequestBody SubscribeRequest subscribeRequest) {  
 System.***err***.println(subscribeRequest.getEmail());  
 List<Subscriber> subscriberList = **subscriberService**.findAll();  
 **for** (Subscriber subscriber : subscriberList) {  
 **if** (subscriber.getEmail().equals(subscribeRequest.getEmail())) {  
 **return new** ResponseEntity<>(**new** GenericResponse(**"Такой подписчик уже есть"**), HttpStatus.***BAD\_REQUEST***);  
 }  
 }  
 URI uri = URI.*create*(**"http://localhost:8080/news/unsubscribe"**);  
 URI link = UriComponentsBuilder.*fromUri*(uri)  
 .queryParam(**"email"**, subscribeRequest.getEmail()).build().toUri();  
 String message = **"Благодарим вас за подписку на обновления! Пройдите по ссылке : "** +  
 link.toString() + **", если хотите отписаться от рассылки!С уважением, команда Booking Hotel."**;  
 System.***err***.println(message);  
 **mailSenderService**.sendEmail(subscribeRequest.getEmail(), **"Subscribing"**, message);  
 Subscriber subscriber = **new**

Продолжение приложения В

Subscriber(subscribeRequest.getEmail(), **true**);  
 **subscriberService**.save(subscriber);  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
  
 @ApiOperation(value = **"unsubscribe"**, response = ResponseEntity.**class**, notes = **"unsubscribing"**)  
 @GetMapping(value = **"/unsubscribe"**)  
 **public** ResponseEntity unsubscribe(@RequestParam(**"email"**) String email) {  
 System.***err***.println(email);  
 Subscriber subscriber = **subscriberService**.findByEmail(email);  
 **subscriberService**.deleteById(subscriber.getId());  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
  
 @ApiOperation(value = **"mailing"**, response = ResponseEntity.**class**, notes = **"mailing"**)  
 @PostMapping(value = **"/mailing"**, produces = **"application/json"**, consumes = **"application/json"**)  
 **public** ResponseEntity mailing(@RequestBody MailingRequest mailingRequest) {  
 URI uri = URI.*create*(**"http://localhost:8080/news/unsubscribe"**);  
 **mailSenderService**.mailAll(mailingRequest.getMessage(),uri);  
 **return new** ResponseEntity<>(HttpStatus.***OK***);  
 }  
}

**package** com.booking\_maiseyenka\_stepovoi.service;  
  
**import** com.booking\_maiseyenka\_stepovoi.exceptions.NotFoundException;  
**import** com.booking\_maiseyenka\_stepovoi.model.entity.Booking;  
**import** com.booking\_maiseyenka\_stepovoi.model.repository.BookingRepository;  
**import** com.booking\_maiseyenka\_stepovoi.service.iface.BookingService;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.List;  
  
@Service  
**public class** BookingServiceImpl **implements** BookingService {  
  
 **private final** BookingRepository **bookingRepository**;  
  
 **public** BookingServiceImpl(BookingRepository

Продолжение приложения В

bookingRepository) {  
 **this**.**bookingRepository** = bookingRepository;  
 }  
  
 @Override  
 **public** List<Booking> getAll() {  
 **return bookingRepository**.findAll();  
 }  
  
 @Override  
 **public** Booking save(Booking booking) {  
 **return bookingRepository**.save(booking);  
 }  
  
 @Override  
 **public** Booking findById(Long id) {  
 **return bookingRepository**.findById(id).orElseThrow(() ->  
 **new** NotFoundException(**"Запись о бронировании не найдена"**));  
 }  
  
 @Override  
 **public void** deleteById(Long id) {  
 **bookingRepository**.deleteById(id);  
 }  
  
  
}

**Client:**

**angular.module("crudApp").controller("BookingController", BookingController);**

**BookingController.inject = [ '$scope', 'Booking' ];**

**function BookingController($scope, Booking) {**

**$scope.bookings = Booking.query();**

**console.log($scope.bookings);**

**$scope.booking = {};**

**$scope.buttonText="Submit";**

**$scope.saveBooking = function() {**

**if ($scope.booking.id !== undefined) {**

Продолжение приложения В

**Booking.update($scope.booking, function() {**

**$scope.bookings = Booking.query();**

**$scope.booking = {};**

**$scope.buttonText="Submit";**

**});**

**} else {**

**Booking.save($scope.booking, function() {**

**$scope.bookings = Booking.query();**

**$scope.booking = {};**

**});**

**}**

**}**

**$scope.updateBookingInit = function(booking) {**

**$scope.buttonText="Update";**

**$scope.booking = booking;**

**}**

**$scope.deleteBooking = function(booking) {**

**booking.$delete({id: booking.id}, function() {**

**$scope.bookings = Booking.query();**

**});**

**}**

**$scope.cancelBooking = function (booking) {**

**console.log("cancel booking", booking);**

**let cancelData = {};**

**cancelData.id = booking.id;**

**cancelData.message = $('#messageBooking').val();**

**console.log("cancel data", cancelData);**

**$.ajax({**

**type: 'POST',**

**url: 'http://localhost:8080/hotel/booking/cancel',**

**data: JSON.stringify(cancelData),**

**contentType: 'application/json',**

**dataType: 'json',**

**success: function(data) {**

**console.log("Succc bbouk", data);**

**$scope.bookings = Booking.query();**

**location.reload();**

Продолжение приложения В

**alert("возможно стоит обновить страницу!");**

**},**

**error: function(data){**

**console.log("Err book", data);**

**$scope.bookings = Booking.query();**

**location.reload();**

**alert("возможно стоит обновить страницу!");**

**}**

**});**

**}**

**}**

**$(document).ready(function () {**

**$('.header').height($(window).height());**

**$(".navbar a").click(function () {**

**$("body,html").animate({**

**scrollTop: $("#" + $(this).data('value')).offset().top**

**}, 1000)**

**})**

**$("#tabs").tabs();**

**$("#myModal").modal('show');**

**var startDate = "0";**

**var endDate = "0";**

**console.log(startDate + endDate);**

**updateGraph(startDate, endDate);**

**$("#statsBtn").click(function () {**

**startDate = $("#inputStartDate").val();**

**endDate = $("#inputEndDate").val();**

**if(startDate === "" && endDate === ""){**

**startDate = "0";**

**endDate = "0";**

**}**

Продолжение приложения В

**console.log(startDate);**

**console.log(endDate);**

**updateGraph(startDate, endDate);**

**});**

**//Время статистика**

**$("#dateStatFrom").datetimepicker();**

**$("#dateStatTo").datetimepicker({**

**useCurrent: false**

**});**

**$("#dateStatFrom").on("dp.change", function (e) {**

**$('#dateStatTo').data("DateTimePicker").minDate(e.date);**

**});**

**$("#dateStatTo").on("dp.change", function (e) {**

**$('#dateStatFrom').data("DateTimePicker").maxDate(e.date);**

**});**

**//Время статистика**

**//Форма клиент**

**$('#btnFindClients').click(function(event) {**

**event.preventDefault();**

**$.ajax({**

**type: 'POST', // метод отправки**

**url: 'http://localhost:8080/zzzzzzzzz', // путь к обработчику**

**//data:'',**

**data: {**

**"name": $('#clientName').val(),**

**"email": $('#clientEmail').val(),**

**},**

**dataType: 'json',**

**success: function(data){**

**console.log("syccess " + data);**

**$('#formFind').css("display","");**

**},**

**error: function(data){**

**console.log("errrrrrr" + data);**

**}})**

**});**

**$('#btnSendClient').click(function(event) {**

Продолжение приложения В

**event.preventDefault();**

**$.ajax({**

**type: 'POST', // метод отправки**

**url: 'http://localhost:8080/zzzzzzzzz', // путь к обработчику**

**dataType: 'json',**

**success: function(data){**

**//console.log("syccess " + data);**

**console.log("syccess");**

**},**

**error: function(data){**

**console.log("errrrrrr" + data);**

**}})**

**});**

**$('#spamBtn').click(function () {**

**let messsage = $('#spamArea').val();**

**console.log("area message", messsage,);**

**alert("Рассылка сообщения всем подписчикам выполнена");**

**$.ajax({**

**type: 'POST', // метод отправки**

**url: 'http://localhost:8080/news/mailing',**

**data: JSON.stringify({message: messsage}),**

**contentType: 'application/json',**

**dataType: 'json',**

**success: function(data){**

**console.log("syccess spam ", data);**

**},**

**error: function(data){**

**console.log("errrrrrr spam" + data);**

**}})**

**});**

**})**

**var updateGraph = function (startDate, endDate) {**

**$.ajax({**

**type: 'GET', // метод отправки**

**url: 'http://localhost:8080//hotel/clients/statsClient?' +**

Продолжение приложения В

**"startDate=" + startDate + "&endDate=" + endDate, // путь к обработчику**

**dataType: 'json',**

**success: function(data){**

**console.log("syccess");**

**let headers = data[0];**

**let data1 = data[1];**

**let data2 = data[2];**

**createGraph(headers, data1, data2);**

**},**

**error: function(data){**

**console.log("errrrrrr" + data)**

**}});**

**$.ajax({**

**type: 'GET', // метод отправки**

**url: 'http://localhost:8080//hotel/clients/statsClient2?' + "startDate=" + startDate + "&endDate=" + endDate, // путь к обработчику**

**dataType: 'json',**

**success: function(data){**

**console.log("syccess Rose" , data) ;**

**let headers = data[0];**

**let data1 = data[1];**

**let data2 = data[2];**

**createGraphRose(headers, data1, data2);**

**},**

**error: function(data){**

**console.log("errrrrrr" + data)**

**}});**

**}**

**var createGraph = function (labels, data1, data2) {**

**var colors = ['#007bff', '#28a745', '#333333', '#c3e6cb', '#dc3545', '#6c757d'];**

**/\* large line chart \*/**

**var chLine = document.getElementById("chLine");**

Продолжение приложения В

**var chartData = {**

**//labels: ["S", "M", "T", "W", "T", "F", "S"],**

**labels: labels,**

**datasets: [{**

**data: data1,**

**backgroundColor: 'transparent',**

**borderColor: colors[0],**

**borderWidth: 4,**

**pointBackgroundColor: colors[0]**

**},**

**{**

**data: data2,**

**backgroundColor: colors[3],**

**borderColor: colors[1],**

**borderWidth: 4,**

**pointBackgroundColor: colors[1]**

**}]**

**};**

**if (chLine) {**

**new Chart(chLine, {**

**type: 'line',**

**data: chartData,**

**options: {**

**scales: {**

**yAxes: [{**

**ticks: {**

**beginAtZero: false**

**}**

**}]**

**},**

**legend: {**

**display: false**

**}**

**}**

**});**

**}**

**}**

**var createGraphRose = function (labels, data1, data2){**

**var ctx = document.getElementById('myChart').getContext('2d');**

**var myChart = new Chart(ctx, {**

**type: "radar",**

Продолжение приложения В

**data: {**

**labels: labels,**

**datasets: [{**

**label: "Заказы",**

**data: data1,**

**fill: true,**

**backgroundColor: "rgba(255, 99, 132, 0.2)",**

**borderColor: "rgb(255, 99, 132)",**

**pointBackgroundColor: "rgb(255, 99, 132)",**

**pointBorderColor: "#fff",**

**pointHoverBackgroundColor: "#fff",**

**pointHoverBorderColor: "rgb(255, 99, 132)"**

**}**

**]**

**},**

**options: {elements: {line: {tension: 0, borderWidth: 3}},**

**scale: {**

**ticks: {**

**beginAtZero: true,**

**min: 0,**

**},**

**pointLabels: {**

**fontSize: 18**

**}**

**}**

**}**

**});**

**}**

**var authorisation = function () {**

**var login = $('#email').val();**

**var password = $('#pwd').val();**

**var jsonData = JSON.stringify({login: login, password: password});**

**console.log(login);**

**console.log(password);**

**var obj = {};**

**obj.login = login;**

**obj.password = password;**

**//DELETE удалить заглушку, сделать запрос**

**if(password === "" && login === ""){**

Продолжение приложения В

**$("#myModal").modal('hide');**

**} else {**

**$.ajax({**

**type: 'POST', // метод отправки**

**url: 'http://localhost:8080/auth/login',**

**// data: {**

**// "login": login,**

**// "password": password,**

**// },**

**data: jsonData,**

**contentType: 'application/json',**

**dataType: 'json',**

**success: function(data){**

**$("#myModal").modal('hide');**

**console.log("auth " , data)**

**},**

**error: function(data){**

**if(data.status == 200) {**

**$("#myModal").modal('hide');**

**}**

**else {**

**alert("Данные не верны!");**

**}**

**}})**

**}**

**}**

# ПРИЛОЖЕНИЕ Г

**(обязательное)**

**Листинг скрипта генерации базы данных**

CREATE DATABASE IF NOT EXISTS `bookingbd` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `bookingbd`;

-- MySQL dump 10.13 Distrib 8.0.13, for Win64 (x86\_64)

--

-- Host: localhost Database: bookingbd

-- ------------------------------------------------------

-- Server version 8.0.13

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

SET NAMES utf8 ;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `booking`

--

DROP TABLE IF EXISTS `booking`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `booking` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`id\_room` bigint(20) NOT NULL,

`id\_service` bigint(20) DEFAULT NULL,

`arrival\_date` datetime NOT NULL,

`leaving\_date` datetime NOT NULL,

`id\_client` bigint(20) NOT NULL,

Продолжение приложения Г

`human\_amount` bigint(20) NOT NULL,

`status` bit(1) DEFAULT b'0',

PRIMARY KEY (`id`),

KEY `clients` (`id\_client`),

KEY `idroom` (`id\_room`),

KEY `idservice` (`id\_service`),

CONSTRAINT `clients` FOREIGN KEY (`id\_client`) REFERENCES `client` (`id`),

CONSTRAINT `idroom` FOREIGN KEY (`id\_room`) REFERENCES `hotel\_room` (`id`),

CONSTRAINT `idservice` FOREIGN KEY (`id\_service`) REFERENCES `service` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=30 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `booking`

--

LOCK TABLES `booking` WRITE;

/\*!40000 ALTER TABLE `booking` DISABLE KEYS \*/;

INSERT INTO `booking` VALUES (1,1,NULL,'2019-04-29 20:48:57','2019-04-30 20:49:09',5,3,\_binary '\0'),(2,1,NULL,'2019-05-29 20:49:19','2019-05-31 20:49:26',7,3,\_binary '\0'),(3,2,NULL,'2019-04-24 20:49:34','2019-04-28 20:49:39',6,3,\_binary '\0'),(4,4,NULL,'2019-05-10 20:49:48','2019-05-12 20:49:54',8,3,\_binary '\0'),(10,4,NULL,'2019-12-25 21:00:00','2019-12-30 21:00:00',9,5,\_binary '\0'),(11,3,NULL,'2019-04-30 21:00:00','2019-05-07 21:00:00',9,5,\_binary '\0'),(12,4,NULL,'2019-10-11 21:00:00','2019-10-19 21:00:00',9,5,\_binary '\0'),(13,3,NULL,'2019-06-02 21:00:00','2019-06-08 21:00:00',9,5,\_binary '\0'),(14,4,NULL,'2019-11-12 21:00:00','2019-11-14 21:00:00',9,5,\_binary '\0'),(15,4,NULL,'2019-06-09 21:00:00','2019-06-12 21:00:00',9,5,\_binary '\0'),(16,4,NULL,'2019-11-08 21:00:00','2019-11-10 21:00:00',9,5,\_binary '\0'),(17,4,NULL,'2019-10-20 21:00:00','2019-10-25 21:00:00',9,5,\_binary '\0'),(18,4,NULL,'2019-10-30 21:00:00','2019-11-04 21:00:00',9,5,\_binary '\0'),(19,1,NULL,'2019-06-13 21:00:00','2019-06-18 21:00:00',9,5,\_binary '\0'),(20,4,NULL,'2019-07-02 21:00:00','2019-07-08 21:00:00',9,5,\_binary '\0'),(21,3,NULL,'2019-08-02 21:00:00','2019-08-08 21:00:00',9,5,\_binary '\0'),(22,4,NULL,'2019-09-02 21:00:00','2019-09-08 21:00:00',9,5,\_binary '\0'),(23,4,NULL,'2019-01-02 21:00:00','2019-01-08 21:00:00',9,5,\_binary '\0'),(24,5,NULL,'2019-02-09 21:00:00','2019-02-11 21:00:00',9,5,\_binary '\0'),(25,4,NULL,'2019-02-02 21:00:00','2019-02-08 21:00:00',9,5,\_binary '\0'),(26,4,NULL,'2019-03-12 21:00:00','2019-03-14 21:00:00',9,5,\_binary '\0'),(27,2,NULL,'2019-03-09 21:00:00','2019-03-11 21:00:00',9,5,\_binary '\0'),(28,4,NULL,'2019-03-02 21:00:00','2019-03-08 21:00:00',9,5,\_binary '\0'),(29,4,NULL,'2019-05-29 21:00:00','2019-05-29 21:00:00',11,5,\_binary '\0');

/\*!40000 ALTER TABLE `booking` ENABLE KEYS \*/;

Продолжение приложения Г

UNLOCK TABLES;

--

-- Table structure for table `client`

--

DROP TABLE IF EXISTS `client`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `client` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`id\_user` bigint(20) NOT NULL,

`fio` varchar(255) NOT NULL,

`phone` varchar(45) NOT NULL,

`email` varchar(45) NOT NULL,

`count\_visiting` bigint(20) NOT NULL DEFAULT '0',

`id\_discount` bigint(20) DEFAULT NULL,

`need\_call` bit(1) NOT NULL DEFAULT b'0',

`last\_message` varchar(1000) DEFAULT NULL,

`card\_holder` varchar(255) DEFAULT NULL,

`card\_cvv` varchar(5) DEFAULT NULL,

`card\_date` varchar(10) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `iddiscount\_idx` (`id\_discount`),

KEY `iduser\_idx` (`id\_user`),

CONSTRAINT `iddiscount` FOREIGN KEY (`id\_discount`) REFERENCES `discount` (`id`),

CONSTRAINT `iduser` FOREIGN KEY (`id\_user`) REFERENCES `user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=12 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `client`

--

LOCK TABLES `client` WRITE;

/\*!40000 ALTER TABLE `client` DISABLE KEYS \*/;

INSERT INTO `client` VALUES (5,4,'Жоржевич Юра Пупкин','+375(29)704-15-17','mail@mail.ru',3,NULL,\_binary '\0',NULL,NULL,NULL,NULL),(6,5,'Тарасевич Братан Петрович','+375(29)987-15-00','rewq@mail.ru',1,NULL,\_binary '','',NULL,NULL,NULL),(7,6,'Мошко Галуга Федорович','+375(29)134-15-17','zxcv@mail.ru',1,NULL,\_binary '','Принесите чаю',NULL,NULL,NULL),(8,7,'Кардин Игорь Сеневич','+375(29)454-15-17','ivantyaj@gmail.com',9,NULL,\_binary '','Мне

Продолжение приложения Г

скучно',NULL,NULL,NULL),(9,8,'Молоко Булка Плюшкович','+375(33)654-88-44','vladstepovoyvios@gmail.com',2,NULL,\_binary '\0',NULL,'фцафца',NULL,'фцафца'),(10,9,'Gwagawgaw awg aw','awaw f af wa','vladstepovoyvios@gmail.com',1,NULL,\_binary '','awfawfawfwa','',NULL,''),(11,10,'Иван','+375 (29) 704-60-43','ivantyaj@gmail.com',1,NULL,\_binary '\0','11','',NULL,'');

/\*!40000 ALTER TABLE `client` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `discount`

--

DROP TABLE IF EXISTS `discount`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `discount` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`percent` bigint(20) NOT NULL,

`min\_visiting\_count` bigint(20) NOT NULL DEFAULT '1',

`description` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `discount`

--

LOCK TABLES `discount` WRITE;

/\*!40000 ALTER TABLE `discount` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `discount` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `feedback`

--

DROP TABLE IF EXISTS `feedback`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `feedback` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

Продолжение приложения Г

`mark` varchar(45) NOT NULL,

`text` varchar(255) DEFAULT NULL,

`id\_client` bigint(20) NOT NULL,

PRIMARY KEY (`id`),

KEY `idclients\_idx` (`id\_client`),

CONSTRAINT `idclients` FOREIGN KEY (`id\_client`) REFERENCES `client` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `feedback`

--

LOCK TABLES `feedback` WRITE;

/\*!40000 ALTER TABLE `feedback` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `feedback` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `hotel\_room`

--

DROP TABLE IF EXISTS `hotel\_room`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `hotel\_room` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`people\_amount` bigint(20) NOT NULL DEFAULT '1',

`id\_type` bigint(20) DEFAULT NULL,

`price` double NOT NULL,

`picture\_url` varchar(255) NOT NULL DEFAULT 'https://vk.cc/9kOXbA',

`description` varchar(255) NOT NULL DEFAULT ' Пока нет описания ',

PRIMARY KEY (`id`),

KEY `idtype\_idx` (`id\_type`),

CONSTRAINT `idtype` FOREIGN KEY (`id\_type`) REFERENCES `room\_type` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `hotel\_room`

--

LOCK TABLES `hotel\_room` WRITE;

Продолжение приложения Г

/\*!40000 ALTER TABLE `hotel\_room` DISABLE KEYS \*/;

INSERT INTO `hotel\_room` VALUES (1,2,1,430,'https://images.wallpaperscraft.com/image/bedroom\_furniture\_tenderness\_romance\_comfort\_70085\_3840x2160.jpg','Хороший номер для двоих'),(2,4,3,200,'https://www.exler.ru/blog/upload/images/big/IMG\_2594.JPG','Номер отлично подходит для семейного отдыха'),(3,2,1,500,'http://dekormyhome.ru/wp-content/uploads/2019/12/37-22.jpg','Лучший номер, для наших самых любимых гостей'),(4,5,2,100,'http://www.wallpapers4u.org/wp-content/uploads/bedroom\_antique\_bed\_portrait\_interiors\_39257\_1920x1080.jpg','Самый дешманский номер, но вам пойдет'),(5,1,2,120,'http://www.wallpapers4u.org/wp-content/uploads/bedroom\_antique\_bed\_portrait\_interiors\_39257\_1920x1080.jpg','Просто комнатушка');

/\*!40000 ALTER TABLE `hotel\_room` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `role`

--

DROP TABLE IF EXISTS `role`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `role` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`role\_name` varchar(45) NOT NULL,

`permission` bigint(20) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `role`

--

LOCK TABLES `role` WRITE;

/\*!40000 ALTER TABLE `role` DISABLE KEYS \*/;

INSERT INTO `role` VALUES (1,'admin',1),(2,'user',0);

/\*!40000 ALTER TABLE `role` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `room\_type`

--

Продолжение приложения Г

DROP TABLE IF EXISTS `room\_type`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `room\_type` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`description` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `room\_type`

--

LOCK TABLES `room\_type` WRITE;

/\*!40000 ALTER TABLE `room\_type` DISABLE KEYS \*/;

INSERT INTO `room\_type` VALUES (1,'LUX','Номер для самых изысанных предпочтений'),(2,'Стандарт','Обычный номер для крестьян'),(3,'Страндарт +','Обычный номер, но чуть дороже');

/\*!40000 ALTER TABLE `room\_type` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `service`

--

DROP TABLE IF EXISTS `service`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `service` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`price` double NOT NULL DEFAULT '0',

`description` varchar(255) DEFAULT NULL,

`status` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `service`

--

Продолжение приложения Г

LOCK TABLES `service` WRITE;

/\*!40000 ALTER TABLE `service` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `service` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `subscribers`

--

DROP TABLE IF EXISTS `subscribers`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `subscribers` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`email` varchar(255) DEFAULT NULL,

`status` bit(1) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `subscribers`

--

LOCK TABLES `subscribers` WRITE;

/\*!40000 ALTER TABLE `subscribers` DISABLE KEYS \*/;

INSERT INTO `subscribers` VALUES (1,'ivantyaj@gmail.com',\_binary ''),(2,'ivantyaj@gmail.com',\_binary '');

/\*!40000 ALTER TABLE `subscribers` ENABLE KEYS \*/;

UNLOCK TABLES;

--

-- Table structure for table `user`

--

DROP TABLE IF EXISTS `user`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

SET character\_set\_client = utf8mb4 ;

CREATE TABLE `user` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`login` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

`id\_role` bigint(20) NOT NULL,

Продолжение приложения Г

PRIMARY KEY (`id`),

KEY `idrole\_idx` (`id\_role`),

CONSTRAINT `idrole` FOREIGN KEY (`id\_role`) REFERENCES `role` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `user`

--

LOCK TABLES `user` WRITE;

/\*!40000 ALTER TABLE `user` DISABLE KEYS \*/;

INSERT INTO `user` VALUES (3,'admin','admin',1),(4,'1','1',2),(5,'2','2',2),(6,'3','3',2),(7,'4','4',2),(8,'5','5',2),(9,'vladstepovoyvios@gmail.com','vladstepovoyvios@gmail.com',2),(10,'ivantyaj@gmail.com','ivantyaj@gmail.com',2);

/\*!40000 ALTER TABLE `user` ENABLE KEYS \*/;

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

-- Dump completed on 2019-05-05 19:38:25