МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к проекту

по дисциплине «Спецификация, проектирование и архитектура программных систем»

Студент гр. 1303 _____ Беззубов Д.В.

Студентка гр. 1303 _____ Королева П.А.

Студент гр. 1303 _____ Чубан Д.В.

Студент гр. 1304 _____ Байков Е.С.

Студентка гр. 1304 _____ Чернякова В.А.

Руководитель _____ Романенко С.А.

Санкт-Петербург

2023

ВВЕДЕНИЕ

Работа представляет собой проект по созданию автоматизированной системы для управления гостиницей. Было разработано ТЗ, в соответствии с которым разрабатывалась архитектура системы и описывалась предметная область. Результат работы описан в пояснительной записке.

1.ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Полное наименование системы и её условное обозначение

Полное наименование: Автоматизированная система управления гостиницей «Hotel Hospitality Management».

Условное обозначение: АС «Hotel Hospitality Management».

1.2. Цели, назначение и области использования АС

Система предназначена для автоматизации управления гостиницей, а именно: автоматизированное бронирование комнаты, внесение оплаты клиентом, учет сведений о свободных и занятых комнатах, планирование необходимого количества закупок расходных материалов, создание графика уборки, соответствующего потребностям и предпочтениям жильцов, планирование меню завтраков.

В результате ожидается функционирующая система, обеспечивающая возможность заказчику предоставлять гостям услуги проживания и питания.

2. ОПИСАНИЕ ПРОЦЕССА ДЕЯТЕЛЬНОСТИ

2.1. Описание терминов предметной области

2.1.1. Словарь терминов

- Клиент – клиент гостиницы
- Персонал – сотрудники гостиницы: администратор, повара, горничные, системный администратор
- Регистрация – процесс создания учетной записи пользователя в системе
- Авторизация – процесс входа пользователем в систему
- Валидация – проверка корректности введенных данных
- Логирование – процесс сохранения данных о работе системы в журнал (лог)
- Номер – помещение в гостинице, состоящее из одного или более мест проживания.
- Лист уборки – список помещений, которые необходимо прибрать в гостинице.

2.2. Описание технологических процессов предметной области в нотации IDEF0

2.2.1. Контекстная диаграмма IDEF0

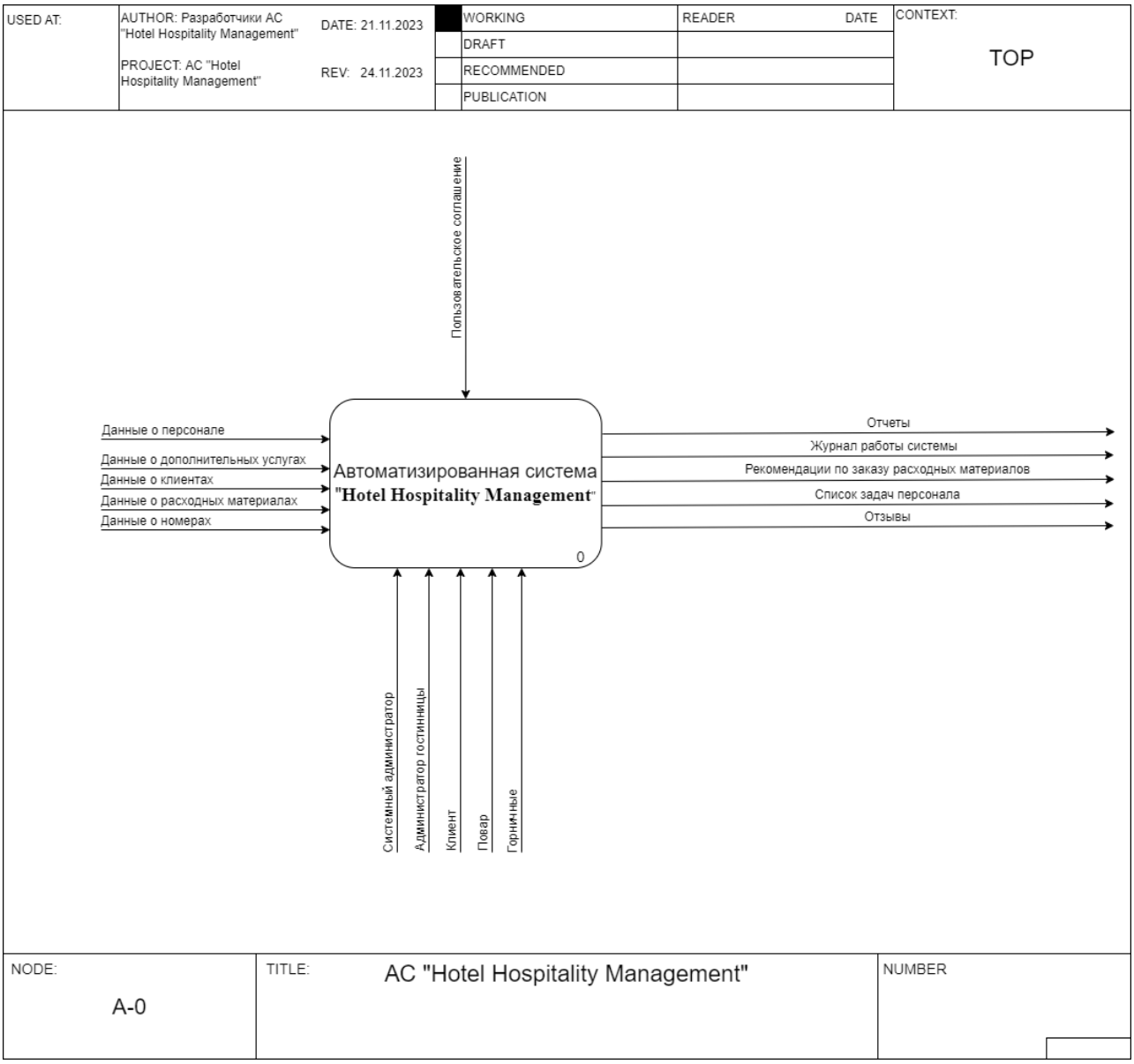

Рисунок 1. Контекстная диаграмма IDEF0

Данная диаграмма определяет границы основной задачи системы: автоматизации процессы в гостинице. Согласно п. 3.2 технического задания имеются 5 категорий пользователей: клиенты, системный администратор, администратор гостиницы, повара и горничные. Для возможности регистрации и авторизации пользователей системы (клиентов и сотрудников гостиницы) предусмотрены входные данные о клиентах и персонале. Для оформления брони в гостинице предусмотрены входные данные о клиентах и номерах. Для

возможности уборки в номерах и приготовления блюд клиентам гостиницы необходимы расходные материалы (продукты, моющие средства и т.д.), поэтому предусмотрены входные данные по расходным материалам, имеющимся на данный момент. На выходе же мы получаем рекомендации по тому какие расходные материалы необходимы, отзывы о работе гостиницы от клиентов, отчеты, журнал работы системы, список задач для персонала.

2.2.2. Диаграмма IDEF0 первого уровня

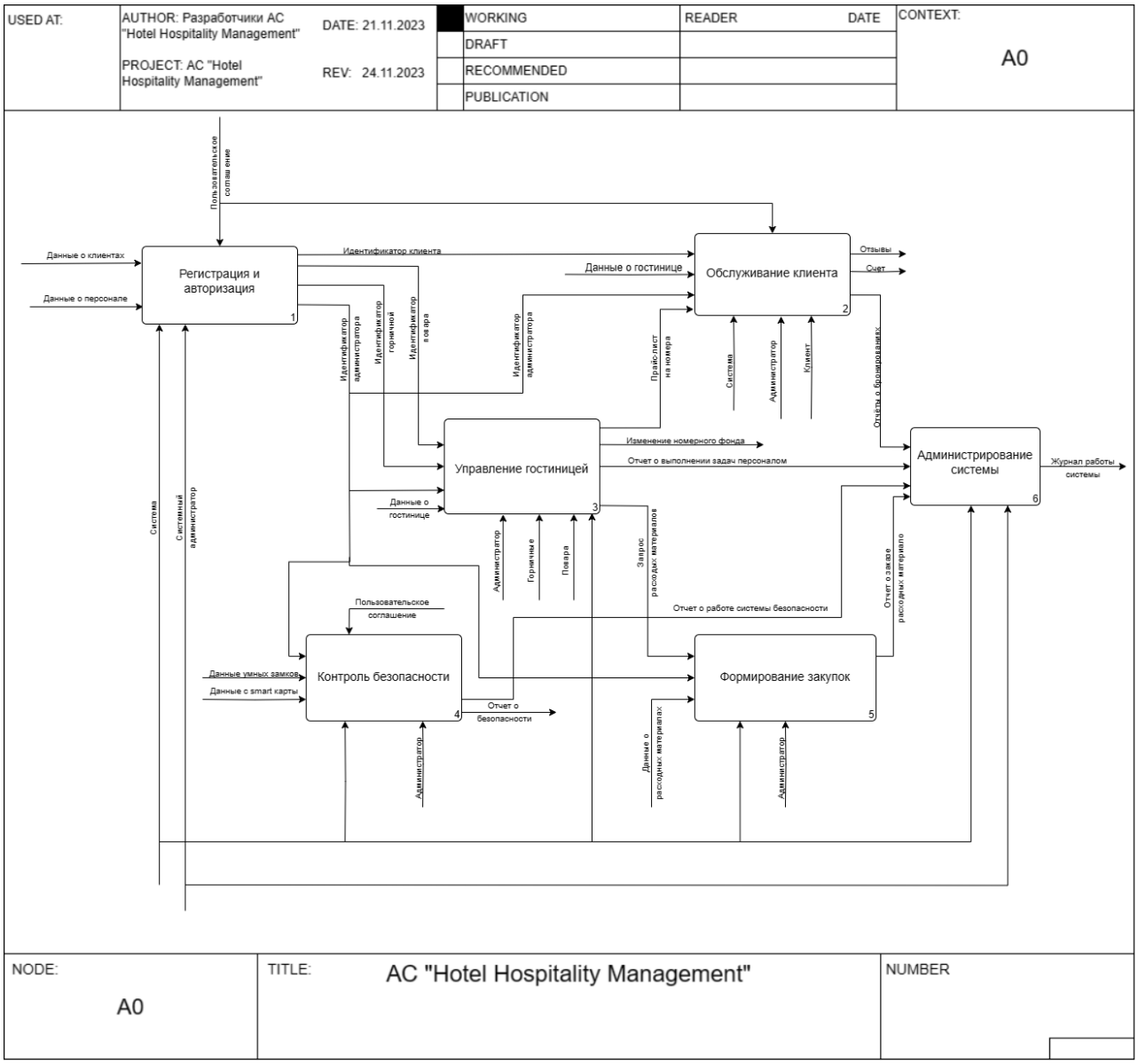

Рисунок 2. Диаграмма IDEF0 первого уровня

С помощью блока «Регистрация и авторизация» клиенты и персонал гостиницы имеют возможность зарегистрироваться и авторизоваться в системе. Для регистрации персонала требуется, чтобы системный администратор выдал

им соответствующие права пользователя. Таким образом, на вход подаются данные о клиенте и данные о персонале, на выходе получаются идентификаторы пользователей.

С помощью блока «Обслуживание клиента» клиенты могут сформировать бронирование, оплатить его, оставить отзыв, а также выбрать дополнительные услуги. При необходимости выбора дополнительных услуг происходит перенаправление в систему дополнительных услуг, после чего данный блок получает набор услуг, выбранных клиентом. При оплате заказа происходит перенаправление на систему оплаты, после чего данный блок получает подтверждение об оплате. Поэтому на вход подается идентификатор клиента, который оформляет бронирование, данные о гостинице, прайс-лист. Также с помощью данного блока администратор может внести бронирование, которое, например, осуществляется с помощью телефонного звонка. Поэтому на вход подается и идентификатор администратора. На выходе получаются отзывы и счет, а также отчеты о бронированиях, с содержанием кто совершил бронь, когда и другие детализации брони.

С помощью блока «Управление гостиницей» администратор формирует список задач по работе гостиницы, то есть распределяет задачи между персоналом, управляет номерным фондом: изменяет доступность номеров и категорий, изменяет цены на номера с учетом акций и скидок. При распределении задач происходит перенаправление в соответствующие системы персонала. В системе горничных формируется график уборки. В системе управления рестораном поварами составляется и корректируется меню. Повара и горничные имеют возможность отметить задачу как выполненную, составить запрос на покупку расходных материалов. Поэтому на вход блока подаются идентификаторы администратора, повара и горничной, а также данные о гостинице, для возможности внесения изменений. На выходе имеется составленный прайс-лист на номера, отчет о выполнении задач персоналом, заказ расходных материалов.

С помощью блока «Контроль безопасности» осуществляется мониторинг и контроль входа и выхода гостей, доступ к номерам и другим местам гостиницы. Администратор управляет выдачей, блокировкой и изменением параметров smart карт. Для этого на вход подается идентификатор администратора, данные smart карты и умных замков. На выходе имеется отчет о безопасности, а также отчет о работе системы.

С помощью блока «Формирование закупок» администратор осуществляет мониторинг заказа расходных материалов. Для этого на вход поступают данные о расходных материалах и сам заказ, а также идентификатор администратора. На выходе имеются отчет о заказе расходных материалов.

С помощью блока «Администрирование системы» происходит запись логов обо всех действиях в системе, а также составление общего журнала. Поэтому на вход подаются данные о бронированиях, список задач персонала, данные о заказе расходных материалов и работе системы безопасности. На выходе имеется журнал работы системы.

2.2.3. Диаграммы IDEF0 второго уровня

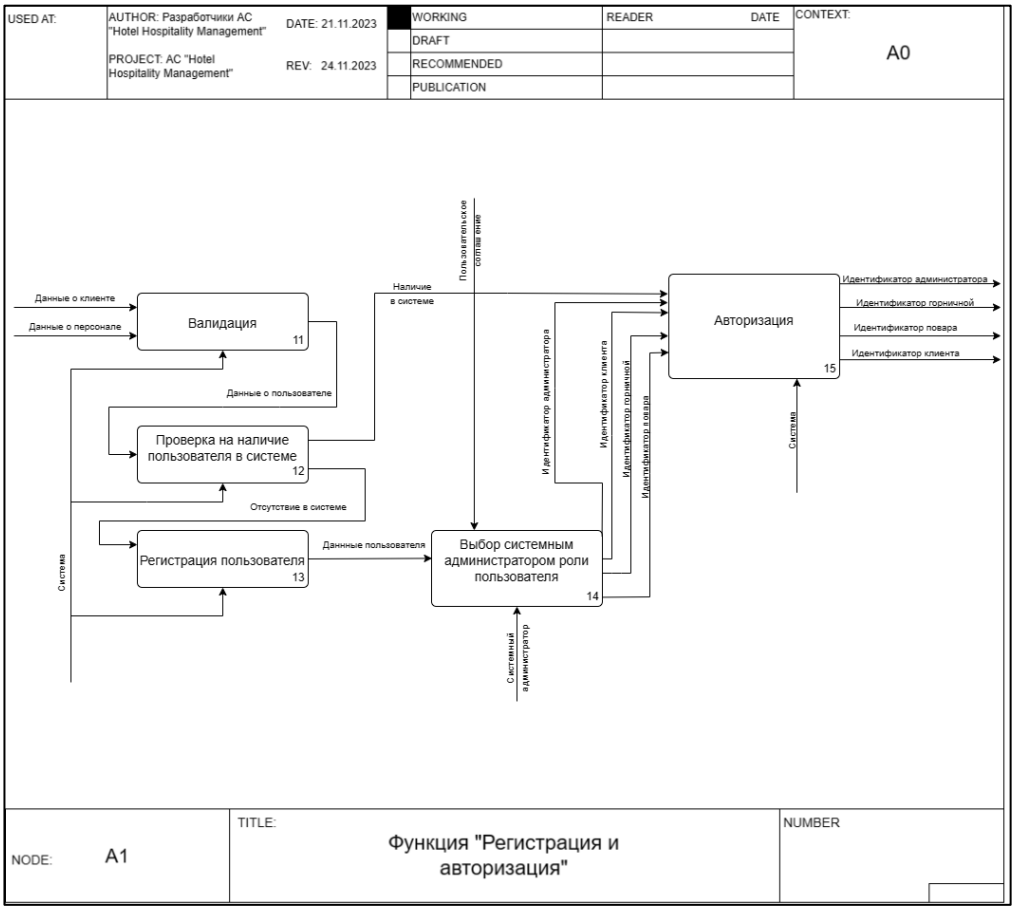

Рисунок 3. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Регистрация и авторизация»

С помощью блока «Валидация» происходит проверка данных клиентов на корректность. Таким образом, на вход подаются непроверенные данные о клиентах и персонале, а на выходе получаются уже проверенные данные.

С помощью блока «Проверка на наличие пользователя в системе» осуществляется поиск пользователя в базе данных системы, в случае если пользователь найден, он перенаправляется на блок «Авторизация», в противном случае — на блок «Регистрация пользователя».

С помощью блока «Регистрация пользователя» происходит регистрация пользователя в системе и сохранение его данных в базу данных системы. Входными и выходными данными являются данные о пользователе.

С помощью блока «Выбор системным администратором роли пользователя» системный администратор выбирает роль новому пользователю. Изначально все пользователи являются клиентами, если необходимо,

администратор может изменить роль на администратора, повара или горничную. На вход блок получает данные пользователя, а на выходе получаются идентификаторы администратора, повара, горничной и клиента.

С помощью блока «Авторизация» происходит выдача клиентам их прав. При этом входными и выходными данными являются идентификаторы клиентов.

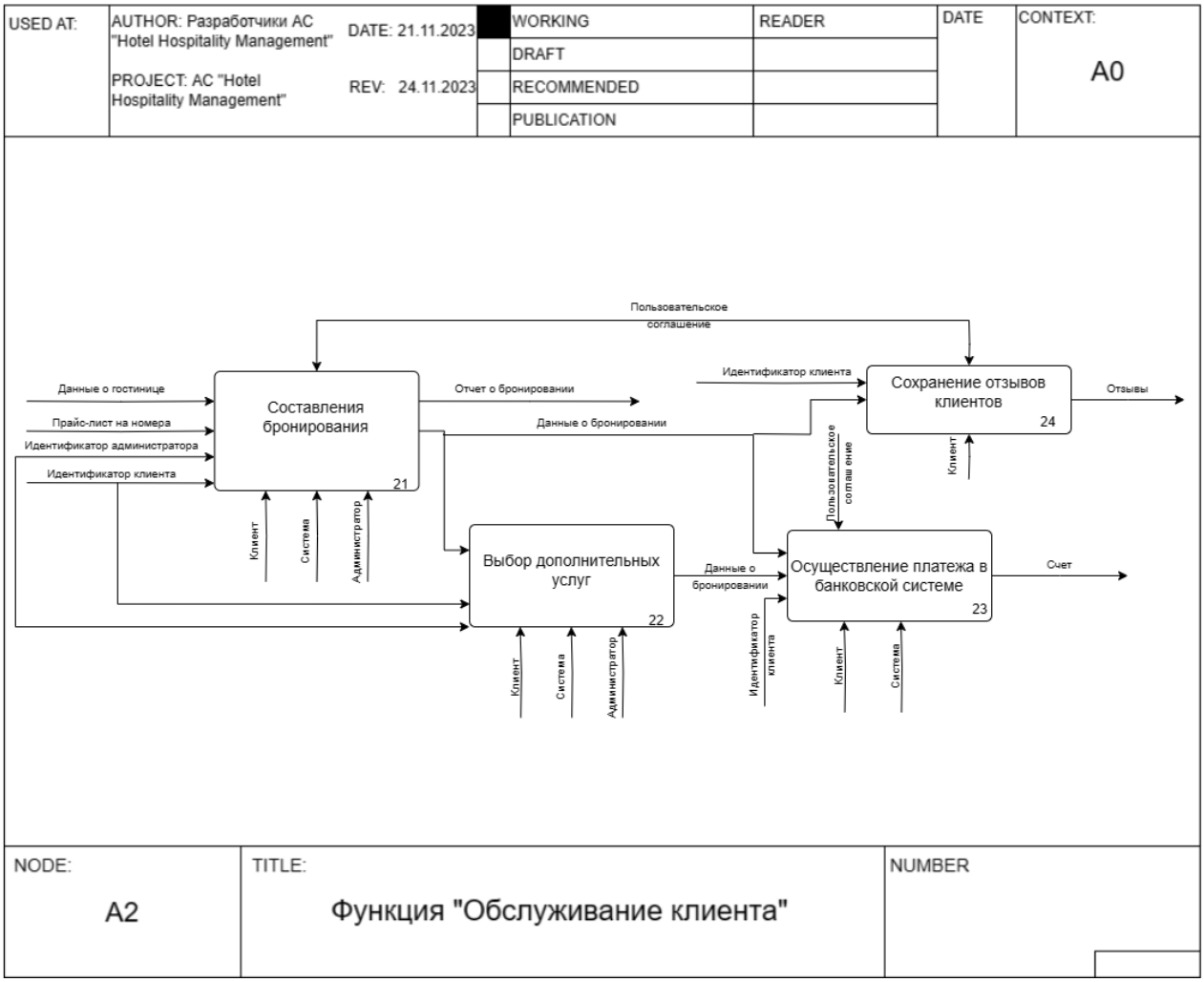

Рисунок 4. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Обслуживание клиента»

С помощью блока «Составление бронирования» клиентом осуществляется выбор номера или номеров на желаемый период пребывания. На вход блок получает идентификатор клиента, данные о гостинице и прайс-листы на номера. Также на вход подается идентификатор администратора, если номер бронируется по телефону. На выход передаются данные о бронированиях. При бронировании используются данные пользователя в соответствии с пользовательским соглашением. Также после осуществления бронирования

формируется отчет, в котором указана дата бронирования, кем совершено и другая детализация брони.

С помощью блока «Выбор дополнительных услуг» осуществляется выбор дополнительных услуг, которые клиент хочет получить. Подается на вход идентификатор администратора и клиента. На выходе также данные о бронировании, но где учитываются дополнительные услуги, в ходе их выбора.

С помощью блока «Подтверждение платежей в банковской системе» происходит оплата бронирования. На вход блок получает идентификатор клиента и данные о бронировании. Если оплата прошла успешно, формируется счет клиента. Данные клиента, которые используются при платеже, используются в соответствии с пользовательским соглашением.

С помощью блока «Сохранение отзывов клиентов» клиенты могут сохранять в системе отзывы о бронированиях и гостинице. На вход блока поступает идентификатор клиента и данные о бронировании. На выходе из блока передаются отзывы пользователей. Сохранение отзыва происходит в соответствии с пользовательским соглашением.

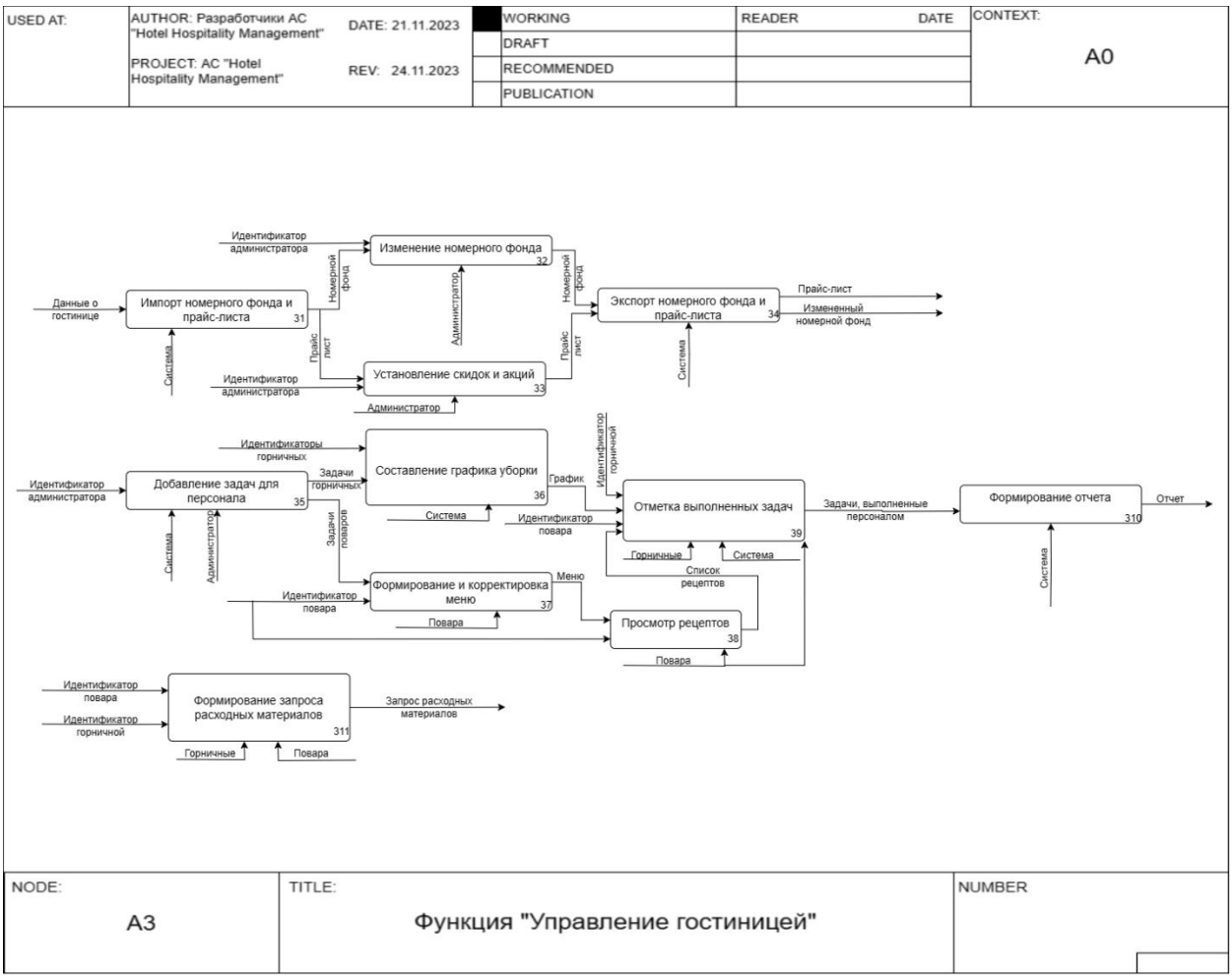

Рисунок 5. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Управление гостиницей»

С помощью блока «Импорт номерного фонда и прайс-листа» импортируются соответствующие данные для внесения изменений. На вход подаются данные гостинице, на выходе получаем номерной фонд и прайс-лист соответственно.

С помощью блоков «Изменение номерного фонда» и «Установление скидок и акций» администратор вносит изменения в соответствующие данные: изменение доступности номеров и категорий, цен на номера с учетом акций и скидок. На вход соответственно поступает идентификатор администратора, а также номерной фонд и прайс-лист. На выходе имеем измененный номерной фонд и прайс-лист.

С помощью блока «Экспорт номерного фонда и прайс-листа» данные экспортируются для возможности отображения. На вход подаются измененные данные, на выходе также получаем их.

С помощью блока «Добавление задач для персонала» администратор добавляет задач, которые должен выполнить персонал, то есть горничные и повара. Система комплектует все задачи вместе, так как в течение дня они обновляются, и передает их персоналу. На вход подается идентификатор администратора, на выходе получаются задачи для горничных и поваров.

С помощью блока «Составление графика уборки» система формирует график, в соответствии с которым горничные должны убирать номера и помещения гостиницы. На вход приходят задачи для горничных и идентификаторы горничных, на выходе получаем готовый график уборки.

С помощью блока «Формирование и корректировка меню» повара на основе полученных задач составляют меню, а в случае необходимости его корректируют. На вход получаем задачи, сформированные для поваров и идентификаторы поваров соответственно. На выходе получаем готовое меню.

С помощью блока «Просмотр рецептов» повара могут просмотреть из чего и как готовить блюда в соответствии с меню. На вход получаем меню, идентификаторы поваров. На выходе имеется готовый список рецептов, то есть перечисление всех блюд и заготовок, которые необходимо сделать.

С помощью блока «Отметка выполненных задач» персонал может отмечать те задачи, которые уже были выполнены. То есть горничные в соответствии с графиком убранные уже номера, а повара в соответствии со списком рецептов приготовленные блюда. Поэтому на вход получаем идентификаторы персонала, а также график уборки и список рецептов. На выходе имеем список задач, которые выполнил персонал.

С помощью блока «Формирование отчета», формируется отчет о работе персонала на основе задач, которые они выполнили. На вход получаем задачи, которые выполнили, на выходе системой формируется отчет.

С помощью блока «Формирование запроса расходных материалов» горничные и повара могут оставить заявку на покупку расходных материалов, которые у них заканчиваются или уже закончились. Поэтому на вход получаем идентификаторы соответствующего персонала, а на выходе запрос расходных материалов, который надо сформировать в заказ.

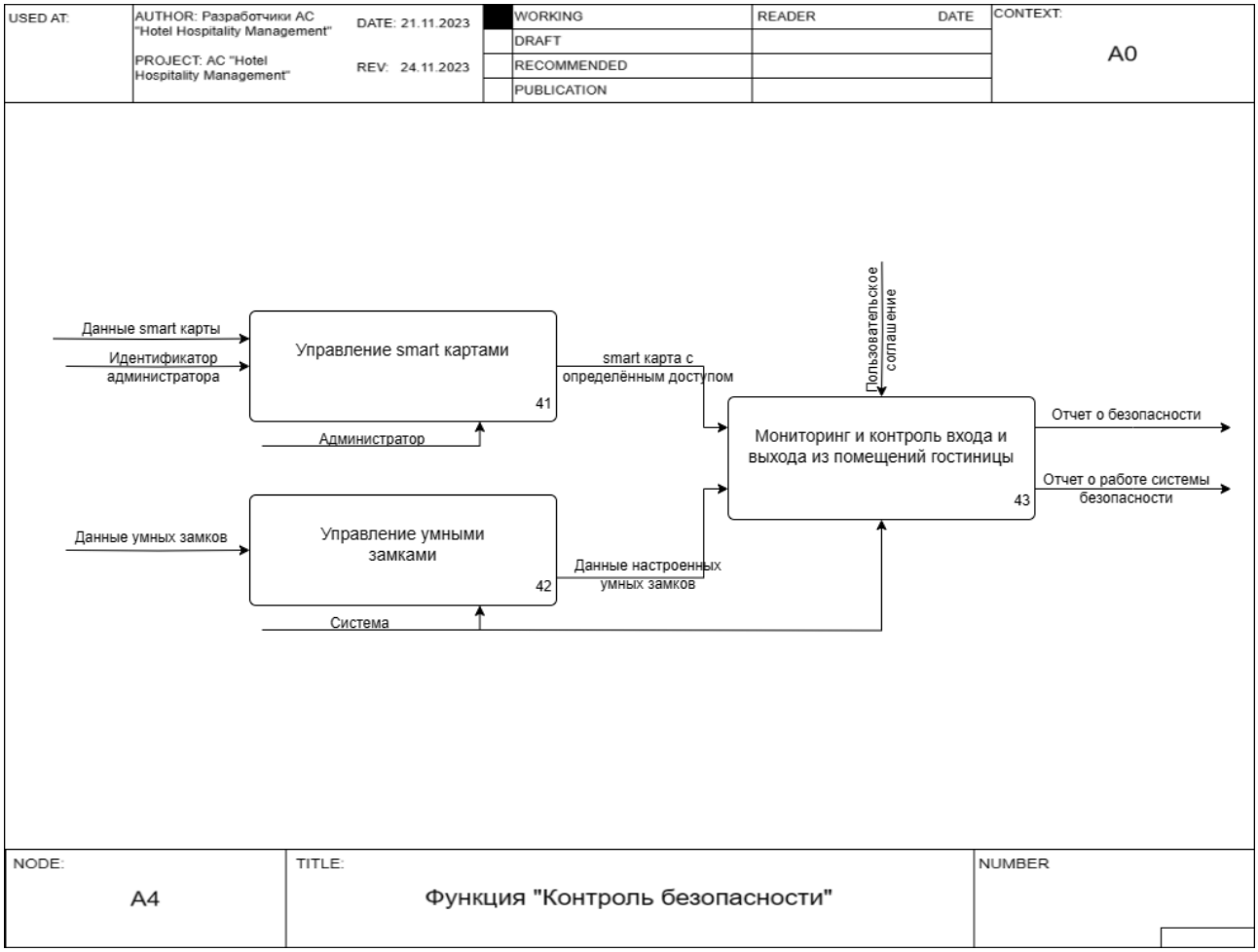

Рисунок 6. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Контроль безопасности»

С помощью блока «Управление smart картами» администратор гостиницы управляет параметрами карт доступа, например, блокирует или привязывает к соответствующему номеру. На вход получаем идентификатор администратора и данные smart карты, а на выходе – smart карту с определенным доступом.

С помощью блока «Управление умными замками» системой настраивается функциональность умных замков: проверяется их работоспособность, меняются параметры. На вход получаем данные умных замков, на выходе – данные настроенных умных замков.

С помощью блока «Мониторинг и контроль входа и выхода из помещений гостиницы» осуществляется контроль за передвижением клиентов и персонала. То есть во сколько клиент пришел, ушел, пытался ли использовать карту для проникновения в чужой номер и тд. Пользовательское соглашение разрешает отслеживание таких перемещений. На вход получаем данные настроенных smart карт и умных замков, а на выход отчет о работе данной системы, а именно кто, где и когда входил или выходил из гостиницы, и отчет по безопасности.

Рисунок 7. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Администрирование системы»

С помощью блока «Формирование заказа» системой на основе данных о расходных материалах (их количество на складах) и запросах от персонала формируется заказ. На вход получаем данные о расходных материалах и запрос о них, на выходе – сформированный заказ, а также отчет о заказе, в котором содержится, кто совершил заказ, что и в каком количестве было заказано.

С помощью блока «Подтверждение заказа» администратор подтверждает заказ, который был сформирован системой и при необходимости вносит в него

изменения. На вход получаем заказ, сформированный системой, и идентификатор администратора, на выход – отредактированный заказ.

С помощью блока «Передача заказа поставщикам» система отправляет уже отредактированный администратором заказ дальше, для его дальнейшего получения. На вход получаемы отредактированный администратором заказ, выход – данные о заказе, который был перенаправлен поставщикам.

Рисунок 8. Диаграмма IDEF0 второго уровня, декомпозирующая функцию «Администрирование системы»

С помощью блока «Журналирование бронирований» происходит запись логов о бронированиях. На вход блок получает отчет о бронированиях, содержание которого описано в описании функции «Управление гостиницы». На выходе получается журнал с соответствующими записями о бронированиях.

С помощью блока «Журналирование выполнения задач поваров и горничных» происходит запись логов о выполнении задач поваров и горничных. На вход блок получает отчет о выполнении задач персоналом, содержание которого описано в описании функции «Управление гостиницы». На выходе

получается журнал с соответствующими записями о задачах поваров и горничных.

С помощью блока «Журналирование заказов расходных материалов» происходит запись логов о заказах расходных материалов. На вход блок получает отчет о заказе расходных материалов, содержание которого описано в описании функции «Формирование закупок». На выходе получается журнал с соответствующими записями о расходных материалах.

С помощью блока «Журналирование работы системы безопасности» происходит запись логов о том, как работает система безопасности. На вход блок получает отчет о работе системы безопасности, содержание которого описано в описании функции «Контроль безопасности». На выходе получается журнал с соответствующими записями о безопасности в гостинице.

С помощью блока «Экспорт системного журнала» происходит формирование общего журнала работы системы. На вход блок получает журналы работ предыдущих четырех блоков и идентификатор системного администратора, и на выходе формирует журнал работы всей системы.

2.2.4. Диаграммы IDEF0 третьего уровня

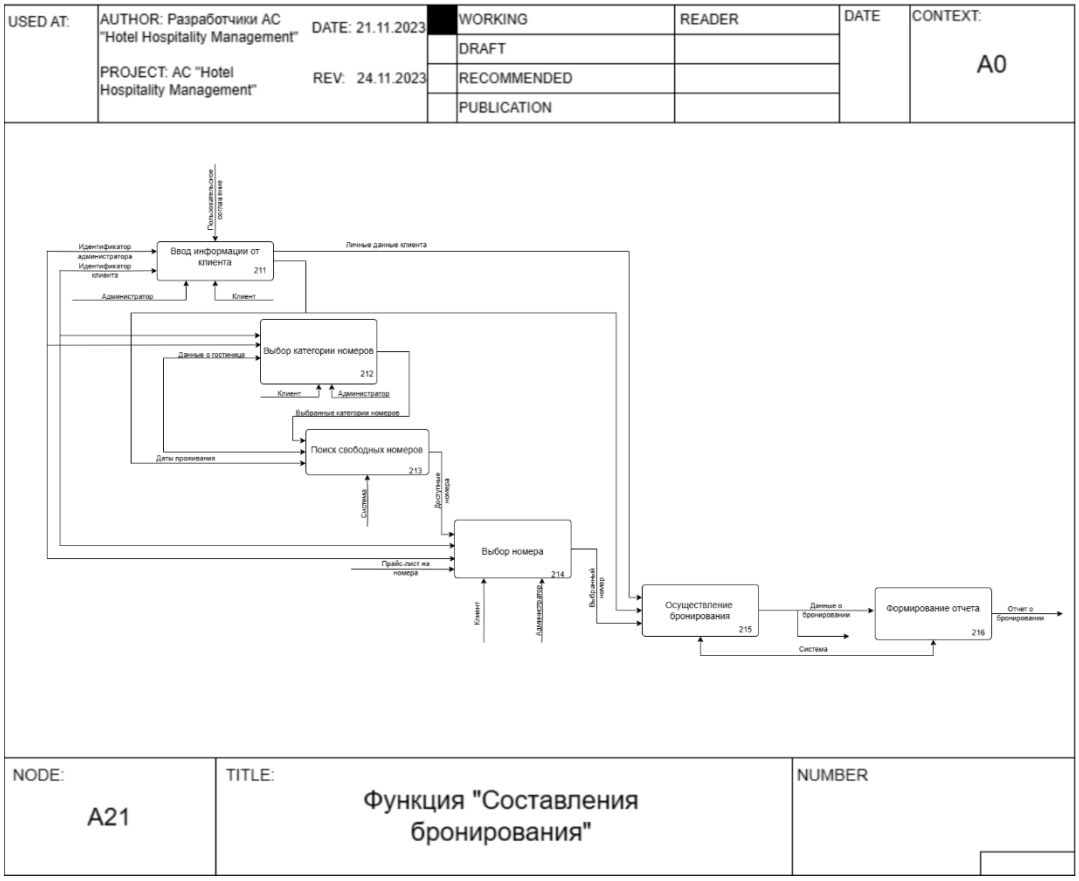

Рисунок 9. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Составления бронирования»

С помощью блока «Ввод информации от клиента» вносятся данные клиента: ФИО, почта, номер телефона, а также даты, когда клиент собирается заехать. Также это может делать администратор в случае, если бронировании идет через звонок в гостиницу напрямую. Входные данные: идентификатора клиента, администратора, на выходе – личные данные клиента и даты проживания. Личные данные клиента сохраняются на сервере в соответствии с пользовательским соглашением.

С помощь блока «Выбор категории номеров» пользователь выбирает желаемые категории: стандарт, полулюкс, люкс, апартаменты. Данные о категориях подгружаются из данных о гостинице, которые подаются на вход. Опять же при бронировании по телефону выбор вносит администратор. На входе также идентификатора клиента и администратора. На выходе – выбранные категории номеров.

С помощью блока «Поиск свободных номеров» системой осуществляется поиск номеров в базе данных гостиницы, которые в даты, когда заезжает клиент, будут свободны. На входе – выбранные категории номеров, данные о гостинице и даты проживания. На выходе – доступные номера.

С помощью блока «Выбор номера» клиент или администратор, по причине описанной выше, на основе прайс-листа и доступных номеров на период в гостинице делает выбор номера для бронирования. Вход – идентификатор клиента, администратора, прайс-лист на номера, доступные номера. На выходе – выбранный номер.

С помощью блока «Осуществления бронирования» за пользователем закрепляется выбранный номер на период проживания, указанный раннее, заносятся его личные данные в базу. Вход – личные данные клиента, даты проживания и выбранный номер, на выходе – данные о бронировании.

С помощью блока «Формирование отчета» системой создается полноценный отчет: кто осуществил бронь, когда и другие ее детали. Вход – данные о бронировании, выход – отчет.

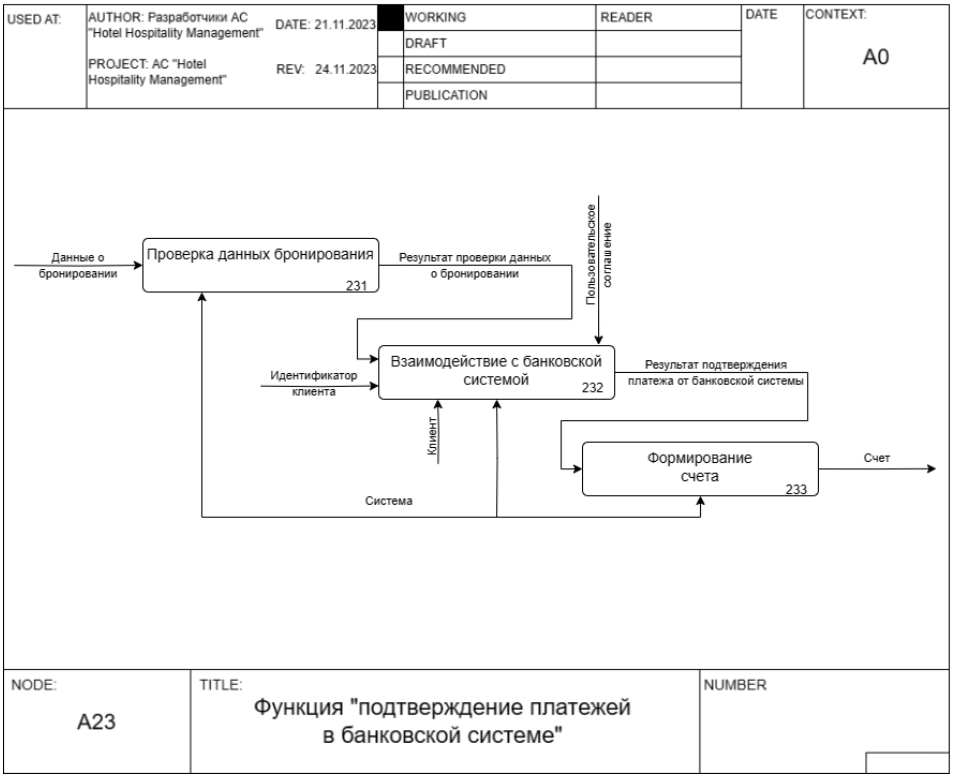

Рисунок 10. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Подтверждение платежей в банковской системе»

С помощью блока «Проверка данных бронирования» системой проверяется корректность и правильность данных, используемых в брони. Вход – данные о бронировании, выход – результат проверки данных о бронировании.

С помощью блока «Взаимодействие с банковской системой» формируется платёж для клиента, в случае успешной проверки брони. Так как клиент вносит свои личные данные для оплаты, применяется пользовательское соглашение. Вход – результат проверки данных о бронировании, идентификатор клиента. Выход – результат подтверждение платежа банковской системы.

С помощью блока «Формирование счета» формируется счет клиента по бронированию. Вход - результат подтверждение платежа банковской системы, выход – счет.

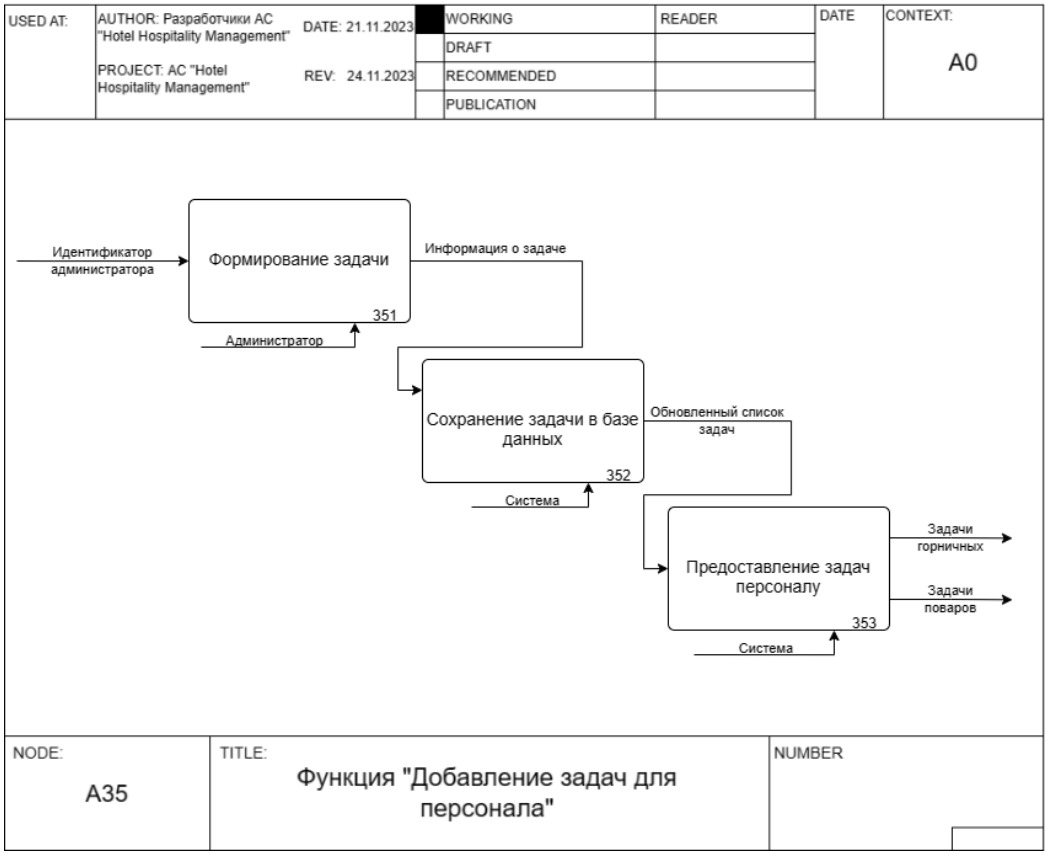

Рисунок 11. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Добавление задач персонала»

С помощью блока «Формирование задач» администратор составляет задачу, которую необходимо выполнить персоналу: что, где и когда сделать. Вход – идентификатор администратора, выход – информация о задаче.

С помощью блока «Сохранение задачи в базе данных» система добавляет задачу в базу к остальным задачам, в зависимости от того, когда ее надо выполнить. Вход – информация о задаче, выход – обновленный список задач.

С помощью блока «Предоставление задач персоналу» систему рассылает всему персоналу конкретный список задач, который был обновлен. Вход – обновленный список задач, выход – задачи горничных, задачи поваров.

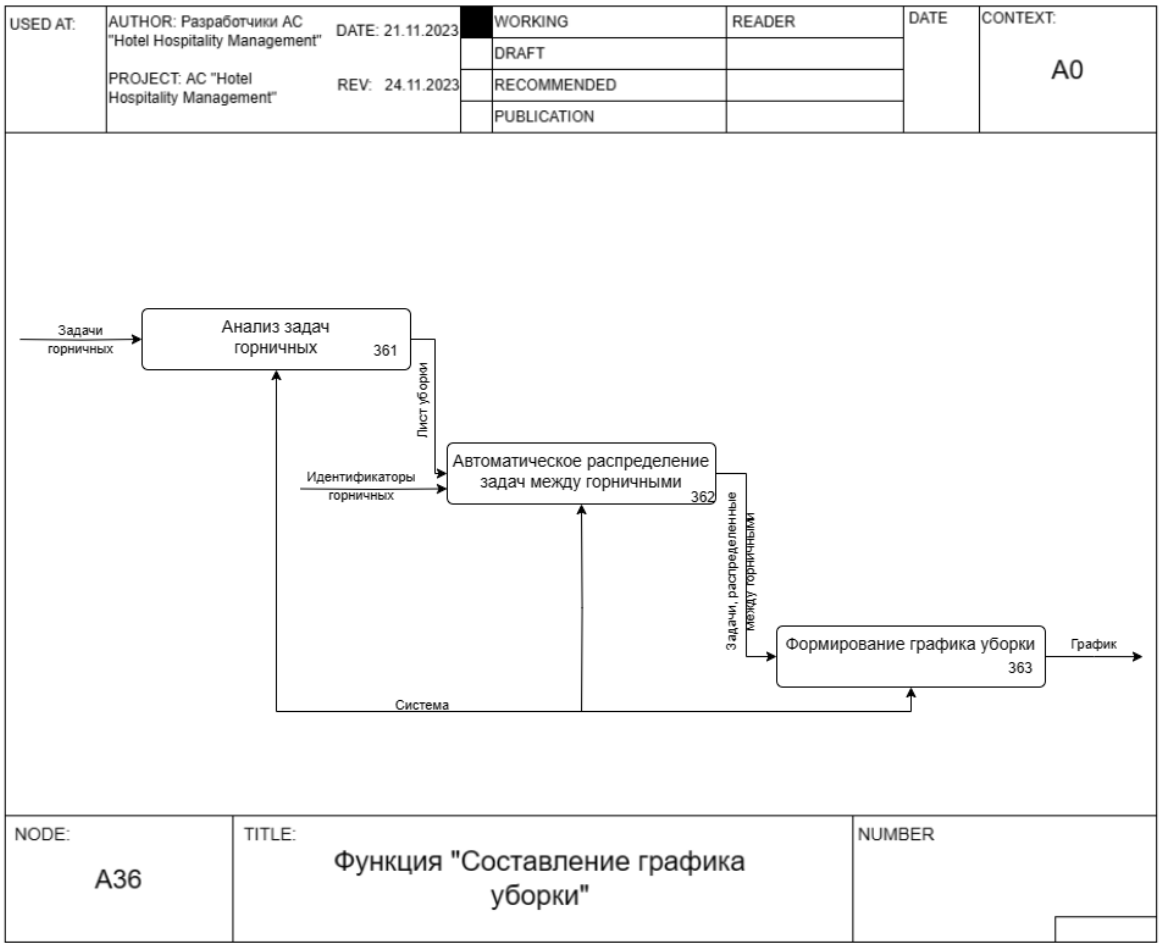

Рисунок 12. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Составление графика уборки»

С помощью блока «Анализ задач горничных» система на основе задач, составленных администратором, формирует список помещений в гостинице, которые необходимо убрать – лист уборки. Вход – задачи горничных, выход – лист уборки.

С помощью блока «Автоматическое распределение задач между горничными» система распределяет пункты из листа уборки между горничными.

Вход – лист уборки, идентификаторы горничных, выход – задачи, распределенные между горничными.

С помощью блока «Формирование графика уборки» системой формируется таблица – график, в которой соотносится время уборки, горничная и ее задачи. Вход – задачи, распределенные между горничными, выход – график.

Рисунок 13. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Управление умными замками»

С помощью блока «Программирование расписания доступа» система программирует замки, чтобы они работали в определенное время, например, ресторан гостиницы может быть открыт только с 8 часов утра до 12, а также настраивается разрешенный доступ к конкретным замкам. Вход – данные умных замков, выход – обновленное расписание доступа.

С помощью блока «Программирование системных настроек» системой программируются системные настройки умных замков: безопасность,

уведомления, автоматическое блокирование. Вход – данные умных замков, выход – обновленные системные настройки.

С помощью блока «Проверка работоспособности» система запускает проверку на правильное реагирование замков с новыми настройками. Вход – обновленное расписание доступа, обновленные системные настройки, выход – результат проверки.

С помощью блока «Взаимодействие с умными замками» система в случае успешной проверки работоспособности передает программированные данные умным замкам, иначе происходит откат изменений. Вход – результат проверки, обновленное расписание доступа, обновленные системные настройки, выход – данные настроенных умных замков.

Рисунок 14. Диаграмма IDEF0 третьего уровня, декомпозирующая функцию «Формирование заказа»

С помощью блока «Обработка запроса расходных материалов» система обрабатывает данные запроса расходных материалов, которые были получены от персонала. Вход – запрос расходных материалов, выход – информация о том, какие расходные материалы нужны.

С помощью блока «Оценка текущего уровня запасов» система проверяет количество расходных материалов на складах. Вход – данные о расходных материалах, выход – данные о текущем уровне запасов.

С помощью блока «Составление списка необходимых расходных материалов» система сопоставляет запрос от персонала и данные со склада, о текущем уровне запасов, чтобы создать корректный заказ. Вход – информация о том, какие расходные материалы нужны, данные о текущем уровне запасов, выход – список необходимых расходных материалов.

С помощью блока «Составление заказа» на основе итоговых данных создается заказ в том виде, который ожидают поставщики. Вход – список необходимых расходных материалов, выход – сформированный заказ.

С помощью блока «Формирование отчета» на основе составленного заказа система создает отчет: когда был составлен заказ, наименования расходных материалов, количество и так далее. Вход – сформированный заказ, выход – отчет.

2.3. Модель предметной области.

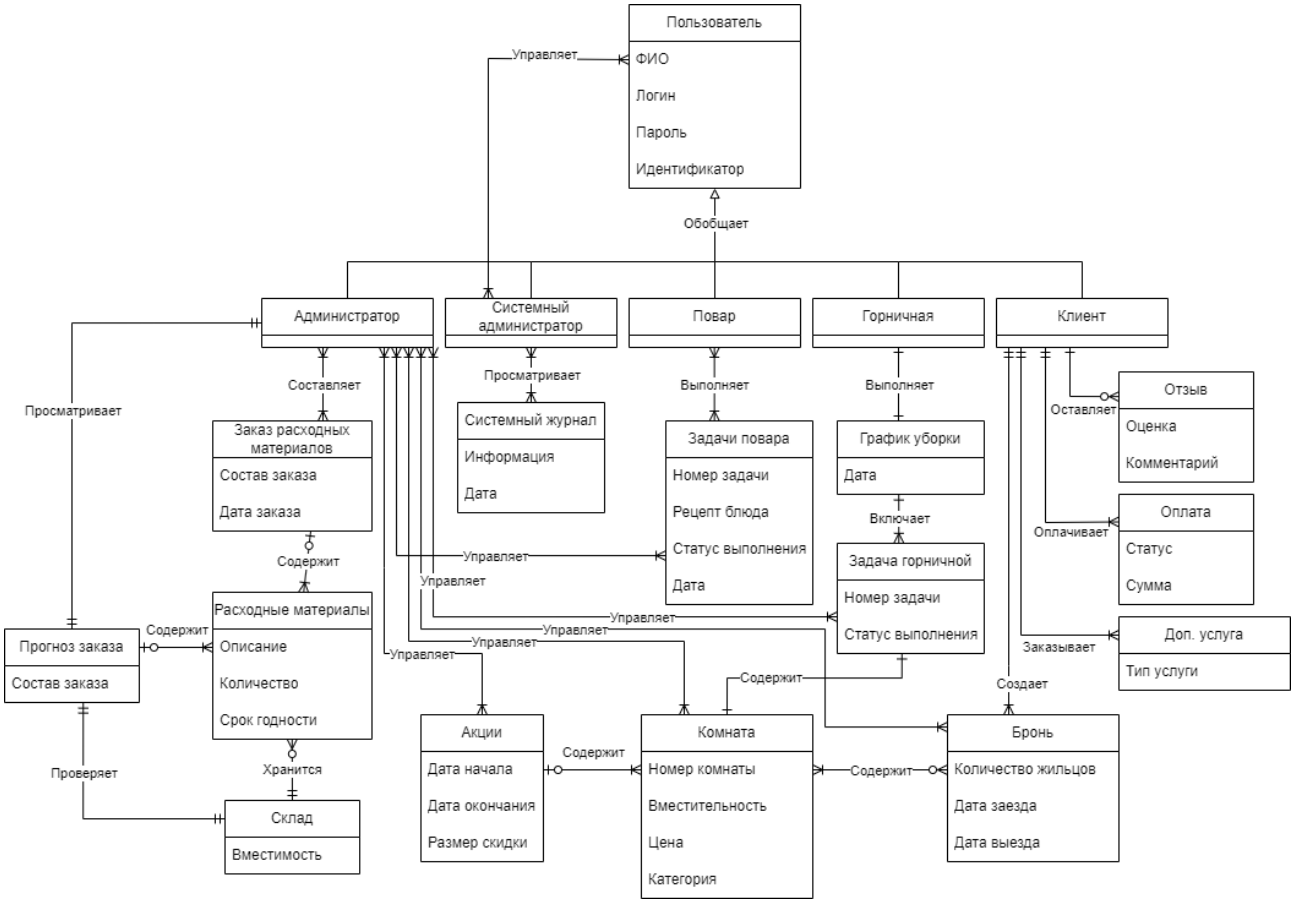

Рисунок 15. Модель предметной области

Сущность «**Пользователь**» является общей: от нее наследуются сущности «Администратор», «Системный администратор», «Повар», «Горничная» и «Клиент». Сущность «Пользователь» определяет ФИО пользователя, роль и данные для входа в систему: логин и пароль. Сущность связана с другими следующим образом:

- «Администратор»: связь многие ко многим (администратор управляет пользователями и назначает им роли).

Сущность «**Администратор**» содержит информацию об администраторе - данные, определенные в сущности «Пользователь». Сущность связана с другими следующим образом:

- «Пользователь»: связь многие ко многим (администраторы управляют пользователями и назначают им роли).
- «Заказ расходных материалов»: связь один ко многим (один заказ может составляться разными администраторами).

- «Комната»: связь многие ко многим (администраторы могут управлять множеством комнат).
- «Бронь»: связь многие ко многим (администраторы управляют бронированием)
- «Задачи пекаря» и «задачи горничной»: связь многие ко многим (администраторы управляют задачами персонала».
- «Прогноз заказа»: связь многие к одному (существует один прогноз заказа расходных материалов и его могут просматривать разные администраторы)
- «Акция»: связь многие ко многим (администраторы могут создавать любое количество акций)

Сущность «**Системный администратор**» содержит информацию о менеджере - данные, определенные в сущности «Пользователь». Сущность связана с другими следующим образом:

- «Системный журнал»: связь многие ко многим (системные администраторы просматривают журналы о работе системы).

Сущность «**Повар**» содержит информацию о пекаре - данные, определенные в сущности «Пользователь». Сущность связана с другими следующим образом:

- «Задача повара»: связь многие ко многим (повар может готовить несколько блюд и приготовлением одного блюда могут заниматься несколько поваров).

Сущность «**Горничная**» содержит информацию о горничной - данные, определенные в сущности «Пользователь». Сущность связана с другими следующим образом:

- «График уборки»: связь один к одному (у каждой горничной свой график).

Сущность «**Клиент**» содержит информацию о клиенте сети пекарен - данные, определенные в сущности “Пользователь”. Сущность связана с другими следующим образом:

- «Отзыв»: связь один ко многим (клиент оставляет отзывы).
- «Оплата»: связь один ко многим (клиент может оплатить несколько заказов).
- «Бронь»: связь один ко многим (клиент может забронировать несколько комнат, но бронь числится за одним клиентом).
- «Доп.услуги»: связь один ко многим (клиент может заказать много доп.услуг)

Сущность «**Заказ расходных материалов**» содержит данные о составе и дате заказа. Сущность связана с другими следующим образом:

- «Расходный материал»: связь один ко многим (в заказе может быть много разных расходных материалов).
- «Администратор»: связь один ко многим (один заказ может составляться разными администраторами).

Сущность «**Задача повара**» содержит информацию о задаче повара: номер задачи, рецепт блюда, которое нужно приготовить, дата и статус выполнения. Сущность связана с другими следующим образом:

- «Повар»: связь многие ко многим (повар может готовить несколько блюд и приготовлением одного блюда могут заниматься несколько поваров).

Сущность «**График уборки**» содержит информацию о графике уборки горничных: дату графика. Содержит задачи, которые берутся из сущности:

- «Задача горничной»: связь многие ко одному (в графике содержится множество задач).

Сущность «**Задача горничной**» содержит информацию о задаче горничной: номер задачи, и статус выполнения. Сущность связана с другими следующим образом:

- «Комната»: связь один к одному (одна задача соответствует одной комнате).
- «График уборки»: связь многие к одному (в графике содержится множество задач).

Сущность «**Отзыв**» содержит информацию об отзыве: оценку и комментарий. Сущность связана с другими следующим образом:

- «Клиент»: связь один ко многим (клиент оставляет отзывы).

Сущность «**Оплата**» содержит информацию об оплате: статус оплаты и сумму. Сущность связана с другими следующим образом:

- «Клиент»: связь один ко многим (клиент может оплатить несколько заказов).

Сущность «**Доп. услуга**» содержит тип заказанной дополнительной услуги. Сущность связана с другими следующим образом:

- «Клиент»: связь один ко многим (клиент может заказать несколько услуг).

Сущность «**Бронь**» содержит информацию о бронировании: количество жильцов, дату заезда и выезда. Сущность связана с другими следующим образом:

- «Клиент»: связь один ко многим (клиент может создать несколько бронирований).
- «Комната»: связь многие ко многим (одна бронь может содержать несколько комнат).

Сущность «**Комната**» содержит информацию о комнате: номер комнаты, вместительность, цену и категорию. Сущность связана с другими следующим образом:

- «Бронь»: связь многие ко многим (комната может содержаться в разных бронях).
- «Администратор»: связь многие ко многим (администраторы могут управлять бронированиями).
- «Задача горничной»: связь один к одному (одна комната содержится в одной задаче горничной).
- «Акция»: связь многие к одному (на стоимость комнаты применяется одна акция).

Сущность «**Расходный материал**» содержит информацию о расходных материалах (продуктах и галантерее) – описание расходного материала, его количество и срок годности. Сущность связана с другими следующим образом:

- «Заказ расходных материалов»: связь один ко многим (в заказе может быть много разных расходных материалов).
- «Склад»: связь один ко многим (на складе может храниться много расходных материалов).
- «Прогноз заказа»: связь многие к одному (в одном заказе указывается множество расходных материалов)

Сущность «**Прогноз заказа**» содержит информацию об составе автоматизированного прогноза заказа расходных материалов. Связи:

- «Расходный материал»: связь один ко многим (в заказе может быть много разных расходных материалов).
- «Склад»: связь один к одному (прогноз составляется на основании запасов одного склада)

Сущность «**Склад**» содержит информацию о складе – вместимость. Сущность связана с другими следующим образом:

- «Расходный материал»: связь один ко многим (на складе может храниться много расходных материалов).
- «Прогноз заказа»: связь один к одному (прогноз составляется на основании запасов одного склада)

Сущность «**Системный журнал**» содержит информацию о работе системы и ее подсистем.

Сущность «**Акция**» содержит информацию об акциях и скидках – даты начала и окончания акции, размер скидки. Сущность связана с другими следующим образом:

- «Администратор»: связь многие ко многим (администраторы могут управлять акциями).
- «Комната»: связь многие к одному (одна акция может распространяться на несколько комнат).

3. АРХИТЕКТУРНЫЕ РЕШЕНИЯ

3.1. Диаграмма Use Case.

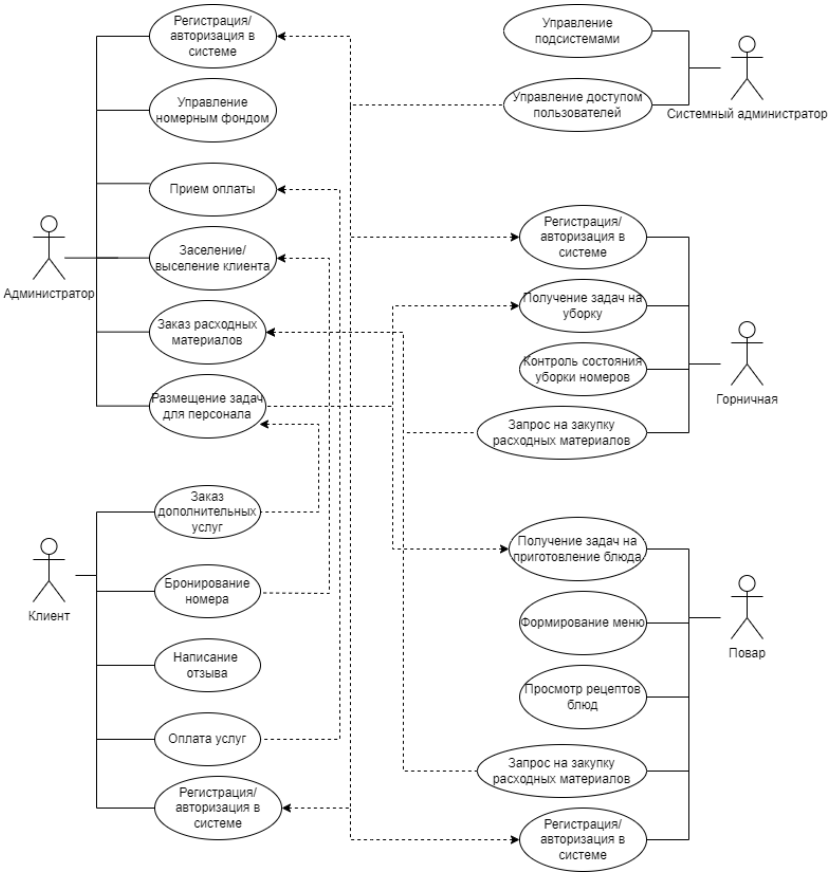

Рисунок 16. Диаграмма Use Case

Рисунок 17. Диаграмма Use Case. Клиент.

Клиент может забронировать номер, оплатить его, написать отзыв и заказать дополнительные услуги. Для выполнения этих действий клиенту необходимо войти в систему. Если клиент ранее не пользовался системой, он должен пройти регистрацию.

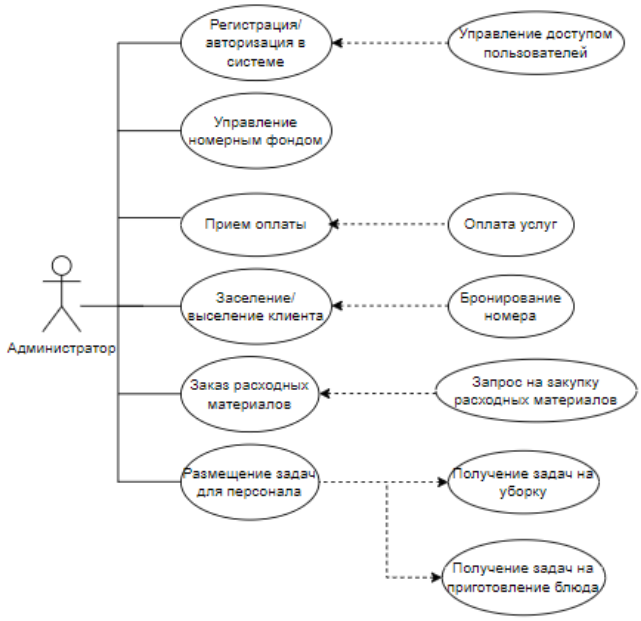

Рисунок 18. Диаграмма Use Case. Администратор.

Администратор напрямую взаимодействует с клиентом. По запросу клиента администратор проводит операции заселения и выселения клиента, принимает от него заказы на дополнительные услуги, принимает оплату от клиента, в зависимости от запросов клиента создает задачи для остального персонала отеля, производит заказ расходных материалов по требованию остального персонала и управляет номерным фондом (изменение доступности номеров и категорий). Для выполнения этих действий администратору необходимо войти в систему.

Рисунок 19. Диаграмма Use Case. Повар.

Повар может получать задачи от администратора, формировать меню ресторана в отеле и создавать запрос администратору на заказ продуктов

(расходные материалы) для обеспечения возможности приготовления блюд из меню. Также повар может посмотреть рецепты блюд из меню. Для выполнения этих действий повару необходимо войти в систему.

Рисунок 20. Диаграмма Use Case. Горничная.

Горничная получает задачи на уборку номеров от администратора, проводит контроль состояния убираемых номеров, а также создает запросы администратору на закупку расходных материалов. Для выполнения этих действий горничной необходимо войти в систему.

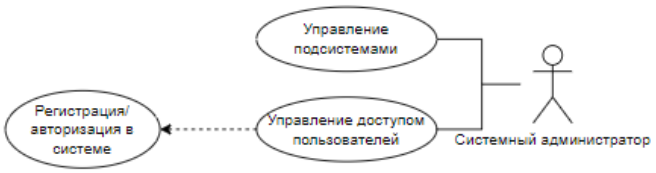

Рисунок 21. Диаграмма Use Case. Горничная.

Системный администратор может управлять подсистемами АС для обслуживания, а также управлять доступом пользователей к различным подсистемам (например добавление новому сотруднику-администратору доступа к подсистеме администратора).

3.2. Диаграмма классов.

На рисунке 22 представлена диаграмма классов проектируемой системы.

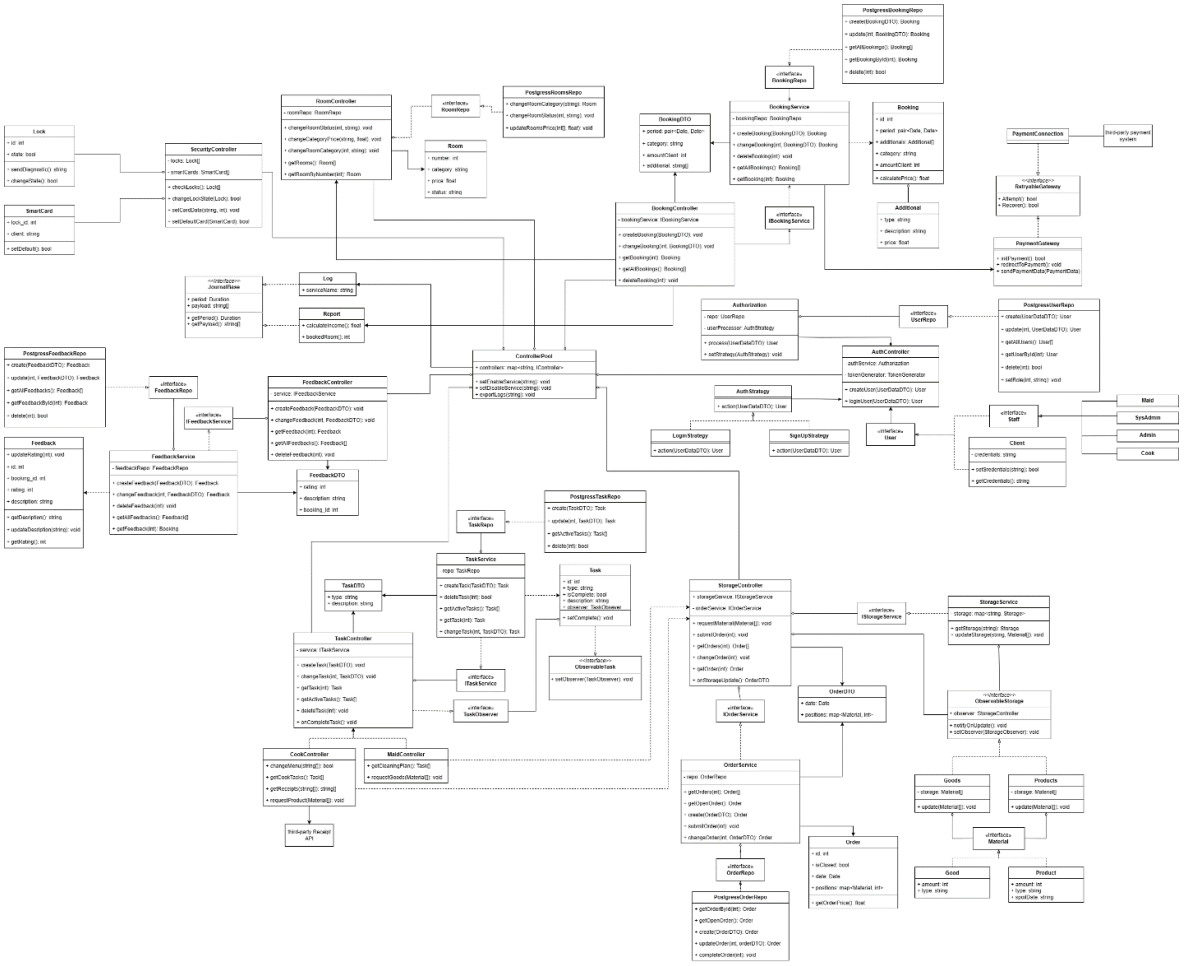

Рисунок 22. Диаграмма классов.

Компонент Авторизации

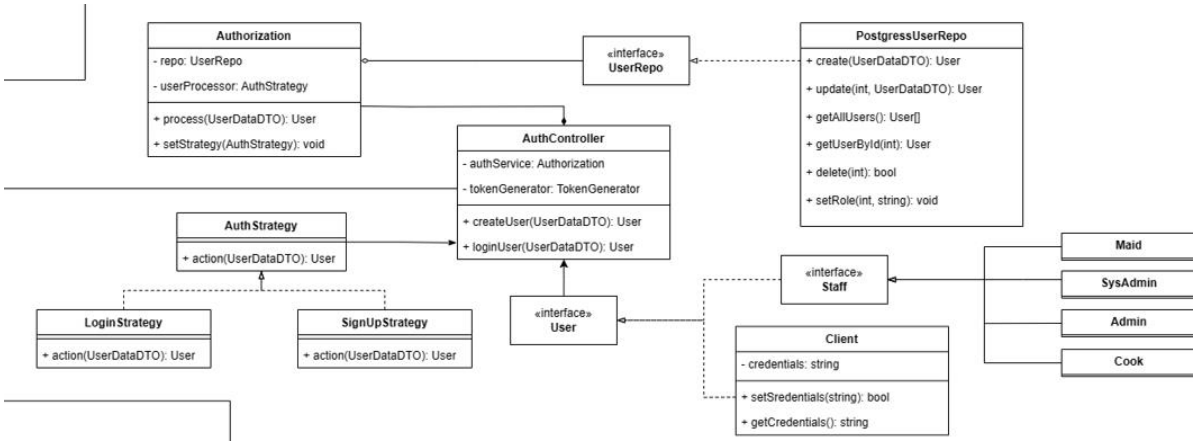

Рисунок 23. Диаграмма классов. Компонента Авторизации.

Описание слоев:

Слой домена

Сущности отражающие объекты бизнес-логики:

User – базовый интерфейс, содержащий необходимые поля, которые наследуются классами-наследниками

Staff – интерфейс для отделения сущностей персонала гостиницы, от сущности клиента гостиницы

Client – сущность, отражающая клиента гостиницы, содержит реквизиты и методы доступа к ним

Слой сценариев:

Authorization – объект, отражающий сценарии обработки данных пользователя при создании аккаунта и авторизации пользователя в системе

Слой контроллеров:

AuthController – объект, осуществляющий управление сервисом авторизации. Реализует получение и отправку данных, передачей данных в сервис.

Слой инфраструктуры:

PostgresUserRepo – реализует интерфейс **UserRepo**, осуществляет взаимодействие с сервером Postgres.

Интерфейс UserRepo служит для инверсии зависимостей в компоненте. Использование данного интерфейса позволяет избежать зависимости от конкретной реализации репозитория.

В данном компоненте используется дизайн-паттерн «Стратегия». Контроллер устанавливает нужную стратегию для обработки пользовательских данных в зависимости от сценария использования.

AuthStrategy – интерфейс объекта стратегии, в нем определен интерфейс взаимодействия с конкретными стратегиями.

LoginStrategy – реализует интерфейс AuthStrategy, данный класс содержит алгоритм обработки данных пользователя в случае входа в систему.

SignUpStrategy – реализует интерфейс AuthStrategy, содержит алгоритм обработки данных в случае создания нового пользователя в системе.

Компонент Задач персонала

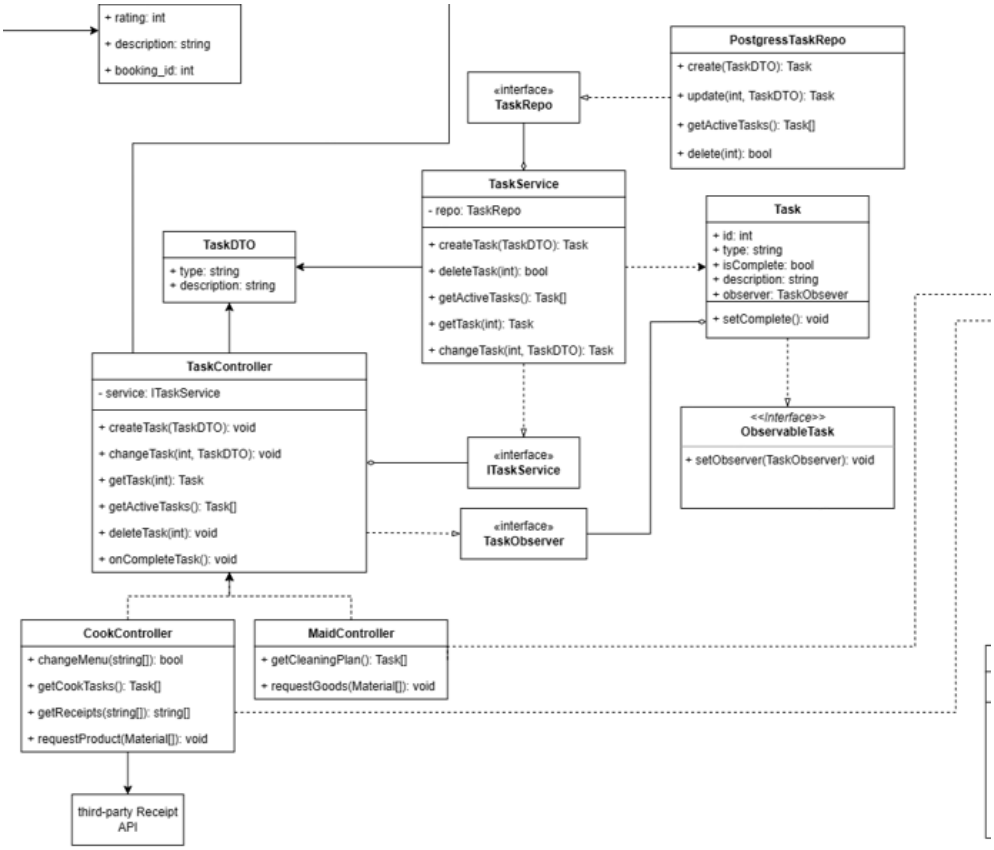

Рисунок 24. Диаграмма классов. Компонента задач персонала.

Слой домена:

Task – класс, отражающий задачи персонала

Слой сценариев использования:

TaskService – класс, содержащий правила бизнес-логики. Осуществляет взаимодействие с базой данных посредством интерфейса TaskRepo. Позволяет создавать, удалять, изменять, получать задачи.

TaskDTO – транспортный объект, служит для передачи данных из одной части компонента в другую.

Слой контроллеров:

TaskController – класс, осуществляющий управление сервисом, получение и отправку данных, передачу данных сервису TaskService.

CookController – класс, зависящий от TaskController, позволяет получать через него задачи для повара, менять меню, запрашивать продукты в компоненте хранилища, получать рецепты через сторонний API.

MaidController – класс, зависящий от TaskController, позволяет получать через него задачи для горничных, запрашивать расходные материалы в компоненте хранилища.

Слой инфраструктуры:

PostgresTaskRepo - реализует интерфейс **TaskRepo**, осуществляет взаимодействие с сервером Postgres.

Интерфейсы:

TaskRepo, **ITaskService** – интерфейсы, определяющие поведение репозитория задач и сервиса соответственно. Служат для инверсии зависимостей в компоненте.

Для отслеживания состояния задач применен паттерн «Наблюдатель».

Для его реализации описаны интерфейсы **ObservableTask** и **TaskObserver**. Интерфейс наблюдателя реализует контроллер задач. Интерфейс наблюдаемого объекта реализует задача. При изменении поля *isComplete*, задача уведомляет контроллер, который в свою очередь обрабатывает ситуацию выполнения задачи.

Компонент склада

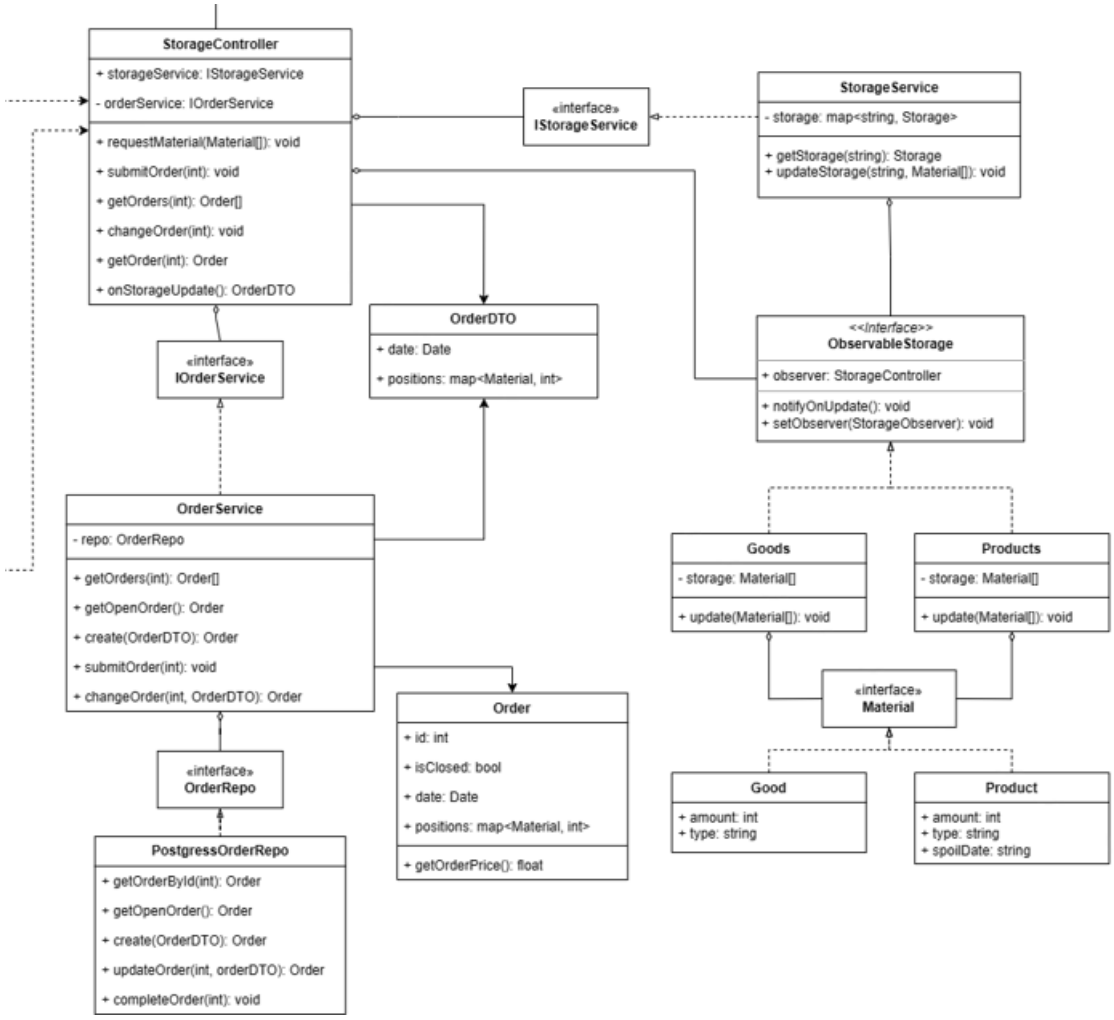

Рисунок 25. Диаграмма классов. Компонента склада.

Слой домена:

Goods – сущность склада хоз.товаров

Products – сущность склада продуктов

Good – сущность, отражающая хоз.товар

Product – сущность, отражающая продукт питания, используемый для приготовления поваром.

Order – сущность, отражающая список материалов для закупки

OrderDTO – транспортная сущность, служит для передачи данных между компонентами системы.

Слой сценариев использования:

StorageService – класс, осуществляющий управление объектами хранилищ, позволяющий получить данные о складах.

OrderService – класс, осуществляющий управление заказами. Позволяет создать, обновить и подтвердить заказ, кроме того, через данный класс происходит взаимодействие с репозиторием заказов.

Слой контроллеров:

StorageController – класс, который осуществляет взаимодействие с сервисами заказов и хранилищ.

Слой инфраструктуры:

PostgresOrderRepo – класс, реализующий интерфейс OrderRepo, осуществляет взаимодействие с сервером Postgres.

Интерфейсы:

OrderRepo, **IOrderService**, **IStorageService** – интерфейсы, определяющие поведение описаных выше сервисов и репозитория. Служат для осуществления инверсии зависимостей в компоненте и снижения связности компонентов.

Material – базовый класс для сущностей Product и Good.

Для отслеживания наличия материалов используется паттерн «Наблюдатель». В качестве наблюдателя используется контроллер, а в качестве наблюдаемых объектов – склады. При изменении количества материалов на складе, и достижении нижней границы, склад уведомляет контроллер, который обрабатывает уведомление и создает заказ, который в дальнейшем корректируется и подтверждается администратором.

Компонент Бронирования

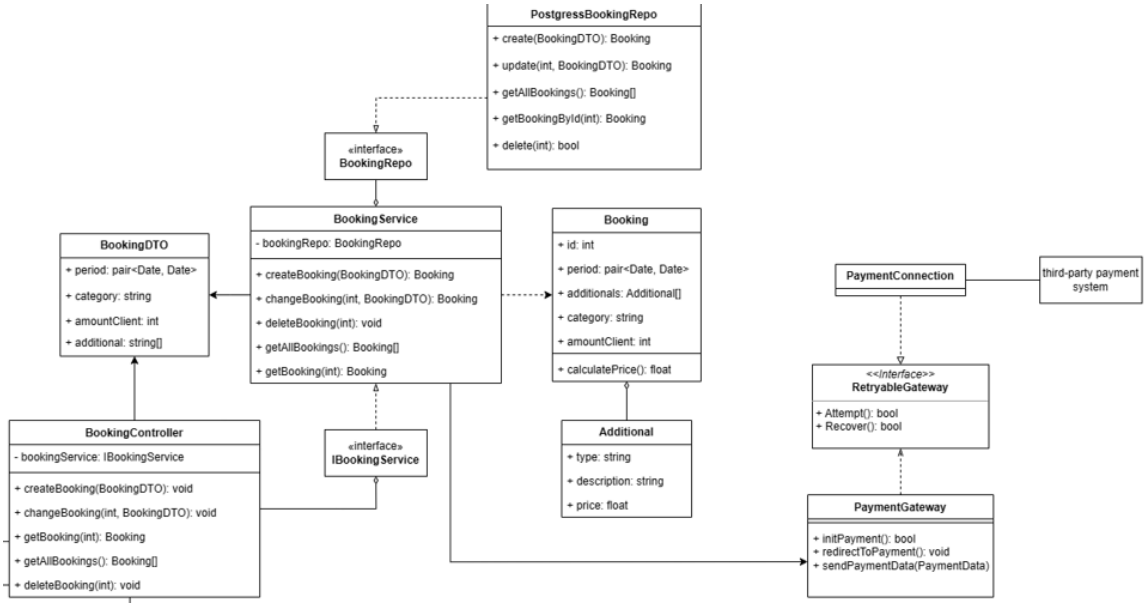

Рисунок 26. Диаграмма классов. Компонента бронирования.

Слой домена:

Booking – сущность, отображающая бронирование. Реализует логику расчета стоимости проживания.

Additional – сущность, отображающая дополнительную услугу.

BookingDTO – транспортная сущность, служит для передачи данных от одного компонента системы к другому.

Слой вариантов использования:

BookingService – реализует правила бизнес-логики. Создает, изменяет, удаляет, получает бронирования. Осуществляет взаимодействие с репозиторием.

Слой контроллеров:

BookingController – класс, реализующий управление сервисом и получение, передачу данных в сервис.

Слой инфраструктуры:

PostgresBookingRepo – реализация интерфейса BookingRepo, осуществляет взаимодействие с сервером Postgres.

Интерфейсы:

BookingRepo – интерфейс, определяющий поведение репозитория бронирований.

IBookingService – интерфейс, определяющий поведение сервиса бронирований.

Служат для инверсии зависимостей в компоненте.

PaymentGateway – класс, реализующий логику инициализации оплаты и установления соединения со сторонним сервисом оплаты

RetriableGateway – интерфейс, определяющий поведение PaymentGateway. Реализует шаблон «**Retriable**», который позволяет повторить операцию, если она завершилась неудачно. Позволяет повысить отказоустойчивость системы при проведении платежных операций.

PaymentConnection – класс, реализующий интерфейс RetriableGateway.

Компонент обработки отзывов

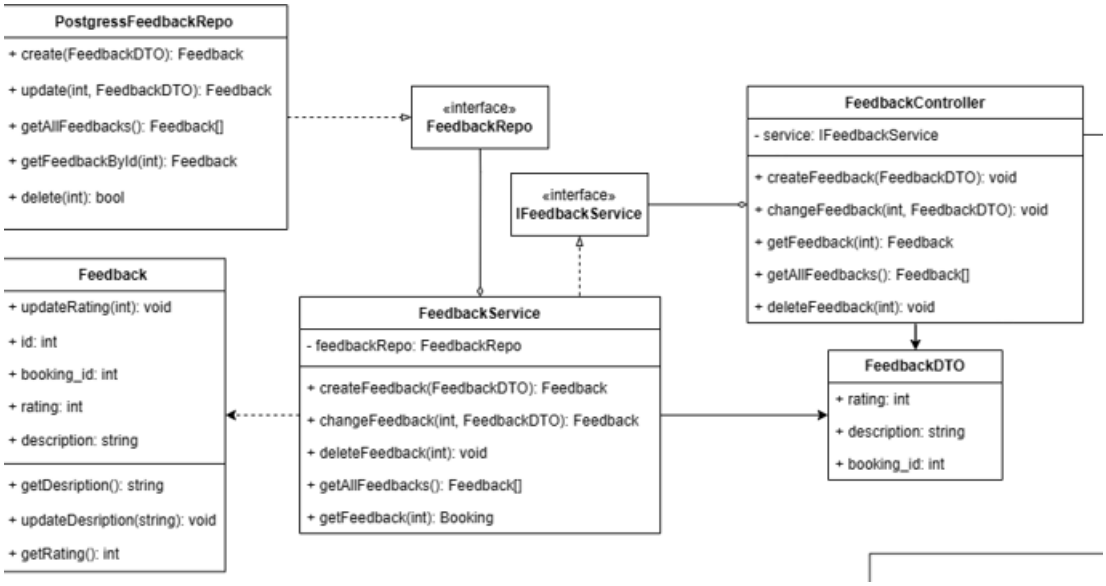

Рисунок 27. Диаграмма классов. Компонента обработки отзывов.

Слой домена:

Feedback – представляет собой отзыв клиента о предоставленных услугах

FeedbackDTO – транспортная сущность для передачи данных об отзывах между контроллером, сервисом и репозиторием.

Слой вариантов использования:

FeedbackService – класс, реализующий взаимодействие с объектами Feedback и репозиторием отзывов.

Слой контроллеров:

FeedbackController – класс, реализующий управление сервисом, передачей данных, полученных с клиента в сервис и в обратном направлении.

Слой инфраструктуры:

PostgresFeedbackRepo – класс, реализующий интерфейс FeedbackRepo. Осуществляет взаимодействие с сервером Postgress.

Интерфейсы:

FeedbackRepo – интерфейс репозитория, определяющий операции взаимодействия с базой данных.

IFeedbackService – интерфейс сервиса, определяющий какие операции должен осуществлять сервис отзывов

Компонент управления номерным фондом

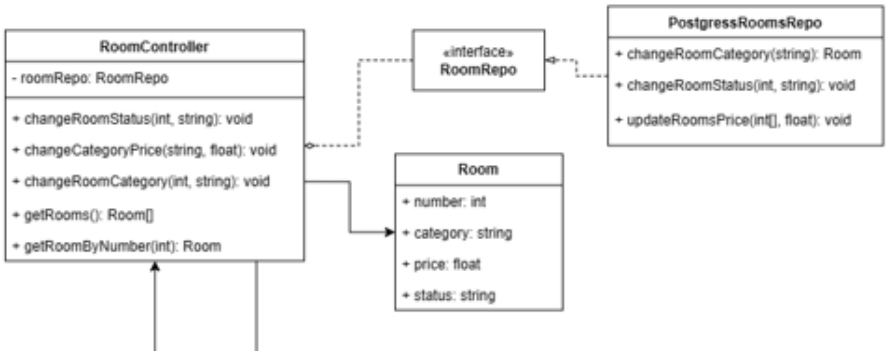

Рисунок 28. Диаграмма классов. Компонента управления номерным фондом.

RoomController – класс, определяющий правила управления номерами. Позволяет изменять стоимость, категорию номера, изменить состояние номера. От данного класса зависит контроллер бронирований, т.к. использует его при создании бронирования.

Использует репозиторий номеров через интерфейс **RoomRepo.**

Room – объект, отражающий сущность номера в отеле.

PostgresRoomsRepo – реализует интерфейс RoomRepo, осуществляет взаимодействие с сервером Postgres.

Компонент управления замками и смарт-картами

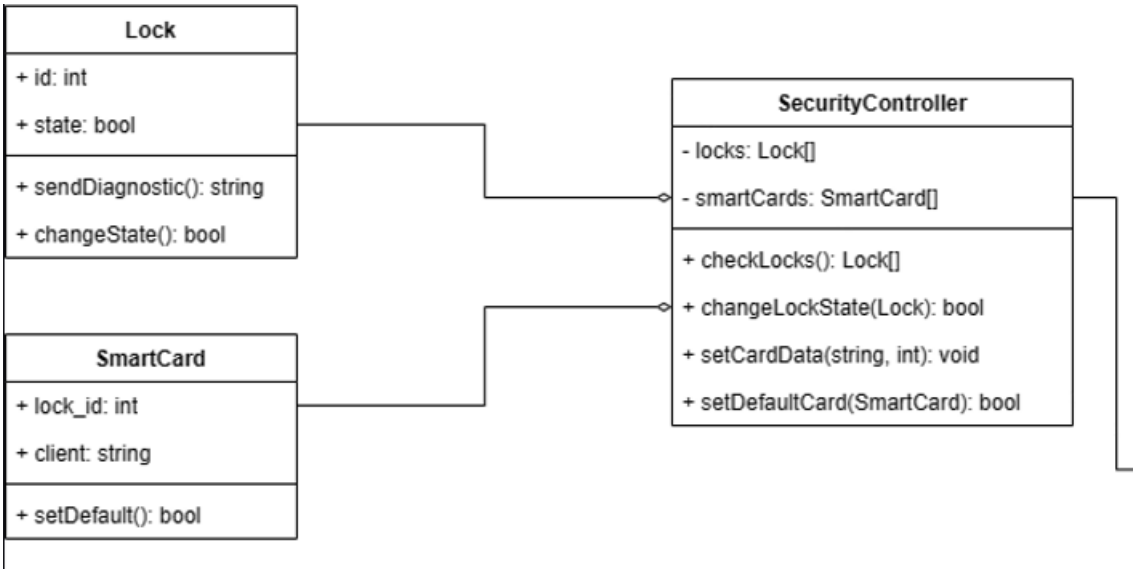

Рисунок 29. Диаграмма классов. Компонента управления замками и смарт-картами.

Lock, **SmartCard** – сущности, которые отражают используемые в гостинице умные замки и смарт-карты соответственно.

SecurityController – класс, реализующий управление картами и замками. Содержит методы проверки замков, смены состояния замков, записи и сброса данных карты.

Сущность ControllerPool

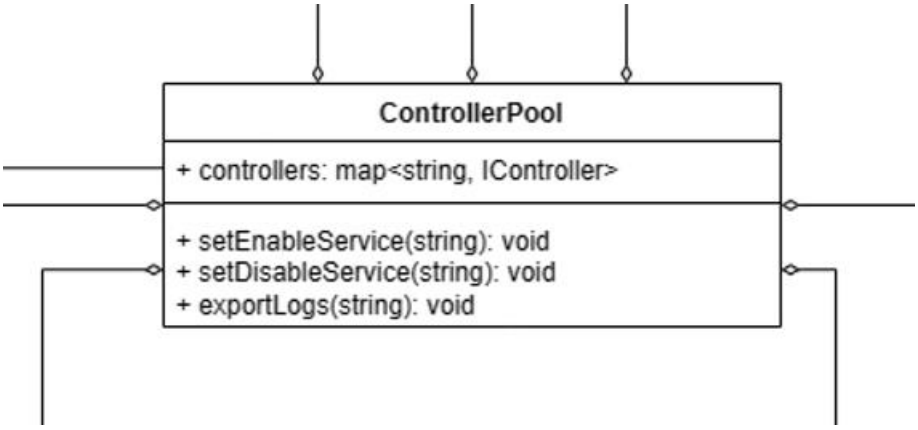

Рисунок 30. Диаграмма классов. Сущность ControllerPool.

ControllerPool – класс, служащий контейнером для всех реализованных контроллеров, позволяет собрать диагностические данные всех компонентов, включить и выключить компоненты.

Сущности отчета и лога

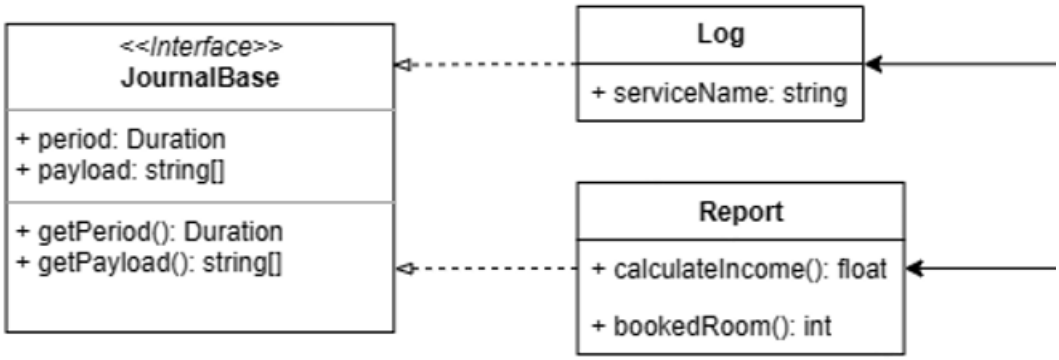

Рисунок 31. Диаграмма классов. Сущности отчета и лога.

Log – класс, содержащий данные о событиях происходящих в компонентах

Report – класс, собирающий данные о бронированиях и подсчитывающий выручку и количество занятых номеров.

Классы наследуются от интерфейса, определяющего общее поведение для Log и Report.

3.3. Диаграмма компонентов.

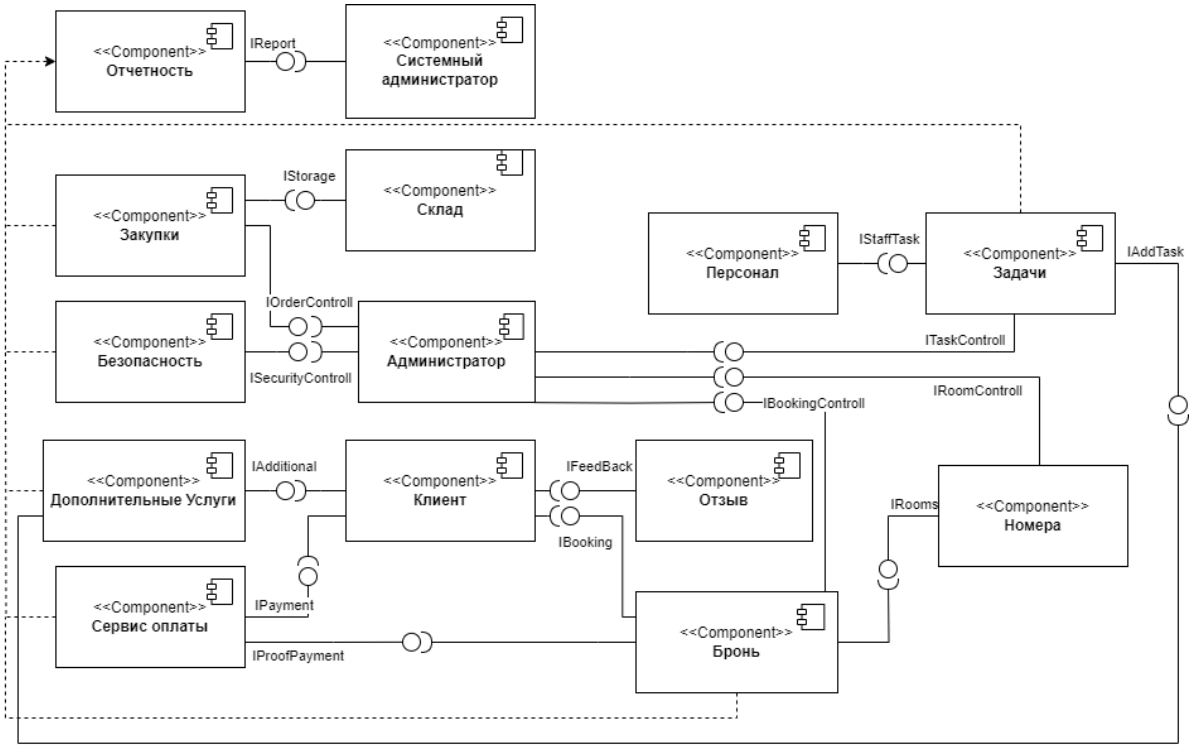

Рисунок 32. Диаграмма компонентов.

На диаграмме представлены следующие компоненты:

1. Отчетность

Отвечает за сбор данных о работе всех компонентов.

Предоставляемые интерфейсы: IReport – служит для предоставления отчетов системному администратору.

Классы: Report, Log.

2. Закупки

Отвечает за закупки расходных материалов.

Предоставляемые интерфейсы: IOrderControll – позволяет администратору управлять закупками.

Классы: Order.

3. Безопасность

Отвечает за модуль безопасности – умные замки и пропуски.

Предоставляемые интерфейсы: ISecurityControll – позволяет администратору управлять системой безопасности.

Классы: Lock, SmartCard, SecurityController

4. Дополнительные услуги

Отвечает за учет дополнительных услуг: ресторана и уборки.

Предоставляемые интерфейсы: IAdditional – позволяет клиенту выбрать доп.услуги.

Классы: Additional.

5. Сервис оплаты

Отвечает за сбор данных о работе со всех компонентов.

Предоставляемые интерфейсы:

IPayment – служит для проведения оплаты клиентом,

IProofPayment – служит для подтверждения системе бронирования факта оплаты.

Классы: PaymentGateway, PaymentConnection.

6. Системный администратор

Отвечает за модуль системного администратора.

Классы: SysAdmin.

7. Персонал

Отвечает за модуль поваров и горничных.

Интерфейсы:

Классы: Maid, Cook.

8. Администратор

Отвечает за модуль администратора.

Классы: Admin.

9. Клиент

Отвечает за модуль клиента.

Классы: User.

10. Бронь

Отвечает за бронирование номеров.

Предоставляемые интерфейсы:

IBooking – позоляет клиенту забронировать номер,

IBookingControll – позволяет администратору управлять бронированиями.

Классы: BookingService, BookingDTO, Booking, BookingController.

11. Задачи

Отвечает за составление и распределение задач между поварами и горничными.

Предоставляемые интерфейсы:

IStaffTask – предоставляет горничным и поварам их задачи,

IAddTask – служит для получения новых задач из компонента доп.услуг,

ITaskControll – позволяет администратору управлять задачами персонала.

Классы: Task, PostgresTaskRepo, TaskService, TaskDTO, TaskController, CoockController, MaidController.

12. Номера

Отвечает за управление номерами.

Предоставляемые интерфейсы:

IRooms – предоставляет компоненту бронирования информацию о комнатах,

IRoomControll – позволяет администратору редактировать информацию о комнатах.

Классы: RoomController, Room, PostgresRoomsRepo

13. Склад

Отвечает за хранение и контроль расходных материалов.

Предоставляемые интерфейсы: IStorage – предоставляет компоненту закупок информацию о материалах, хранящихся на складе.

Классы: StorageController, StorageService, Goods, Products, Good, Product

14. Отзывы

Отвечает за хранение и публикацию отзывов.

Предоставляемые интерфейсы: IFeedBack – позволяет клиенту оставлять и просматривать отзывы.

Классы: Feedback, FeedbackService, PostgresFeedbackRepo, FeedbackController, FeedbackDTO

3.4. Диаграмма пакетов.

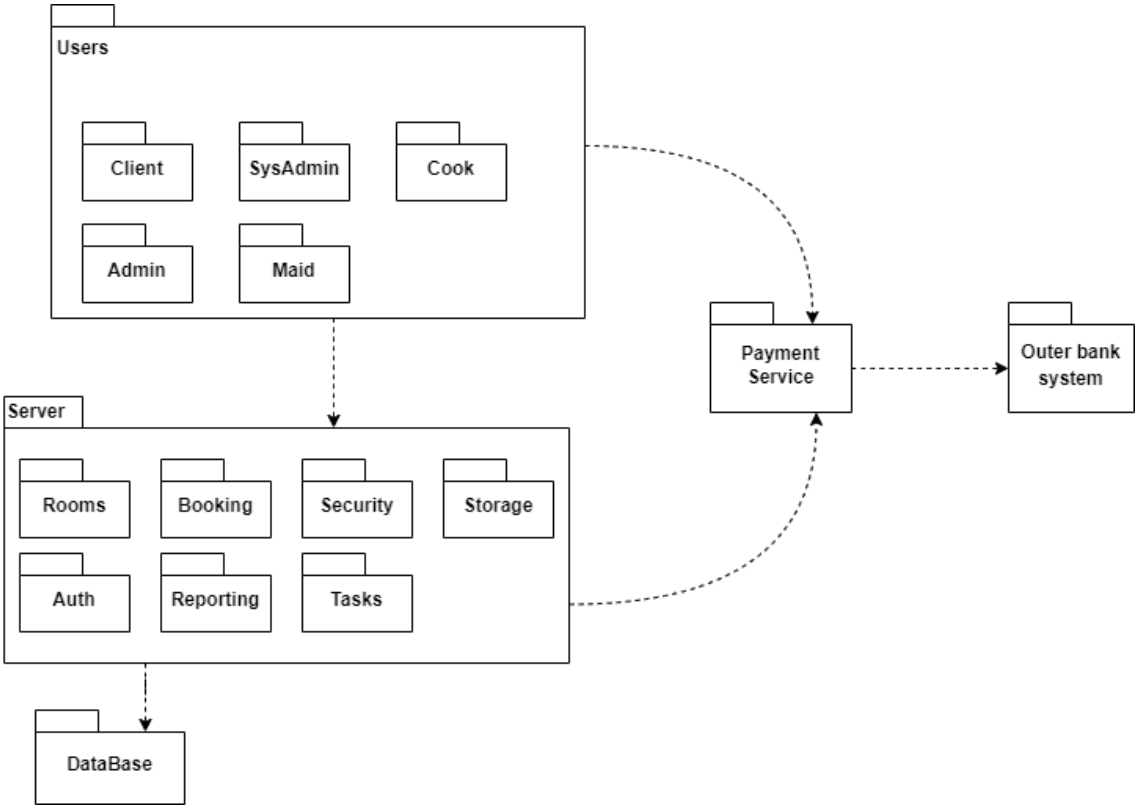

Рисунок 33. Диаграмма пакетов.

Users – пользователи автоматизированной системы гостиницы. Включает

- клиентов
- администраторов
- системных администраторов
- горничных
- поваров.

Server – представляет основной блок работы автоматизированной системы. Здесь располагаются подсистемы:

- авторизации
- управления комнатами
- бронирования
- распределения задач между персоналом
- управления складом
- безопасности
- отчетности.

Payment Service – обращается к внешней банковской системе и проводит транзакцию при оплате услуг и бронирования.

Outer bank system – внешняя банковская система.

Database – база данных, хранящая информацию о системе.

3.5. Диаграммы состояний.

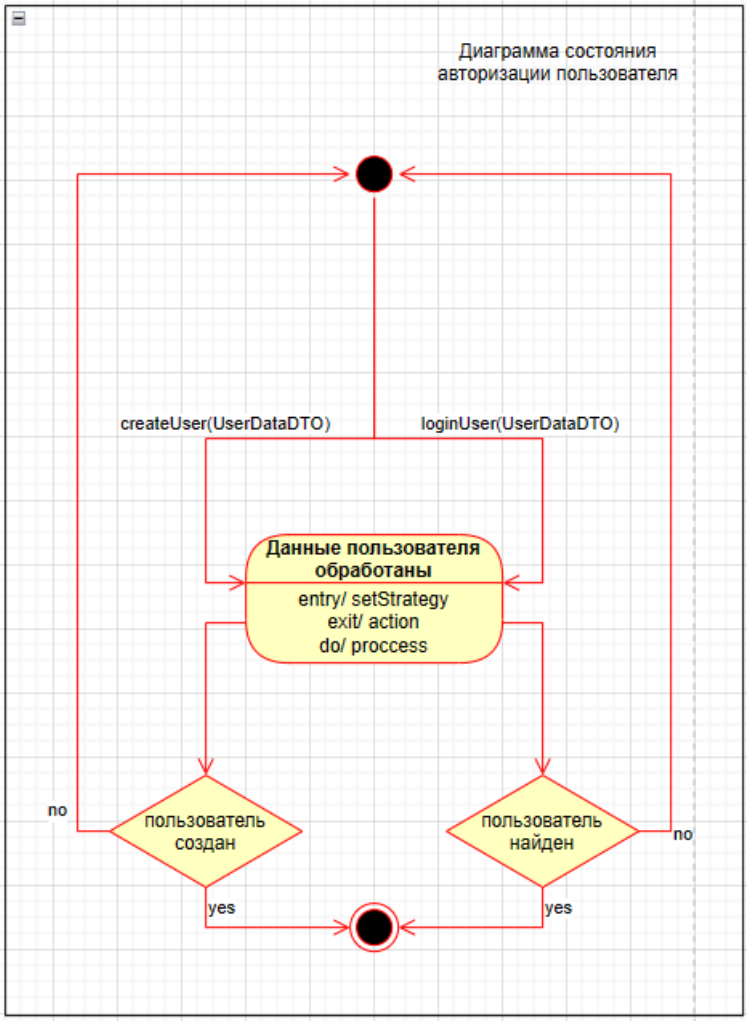

Рисунок 34. Диаграмма состояний авторизации пользователя.

Процесс авторизации пользователя начинается с одного из действий – создания (createUser()) или авторизации (loginUser()) пользователя. Данные действия приводят к простому состоянию с внутренними действиями «Данные пользователя обработаны». Действие при входе setStrategy() – установка соответствующей стратегии обработки данных пользователя. Выходное действие – action() объекта стратегии. Оно реализует алгоритм обработки данных, в случае создания пользователя – валидация данных и создание записи в базе данных, в случае входа – поиск записи в базе данных, сравнение переданных данных с найденными записями. После выхода из этого состояния проверяется успешность действия. Если действие неудачное – система возвращается в начальное состояние. В случае успеха – система переходит в конечное состояние.

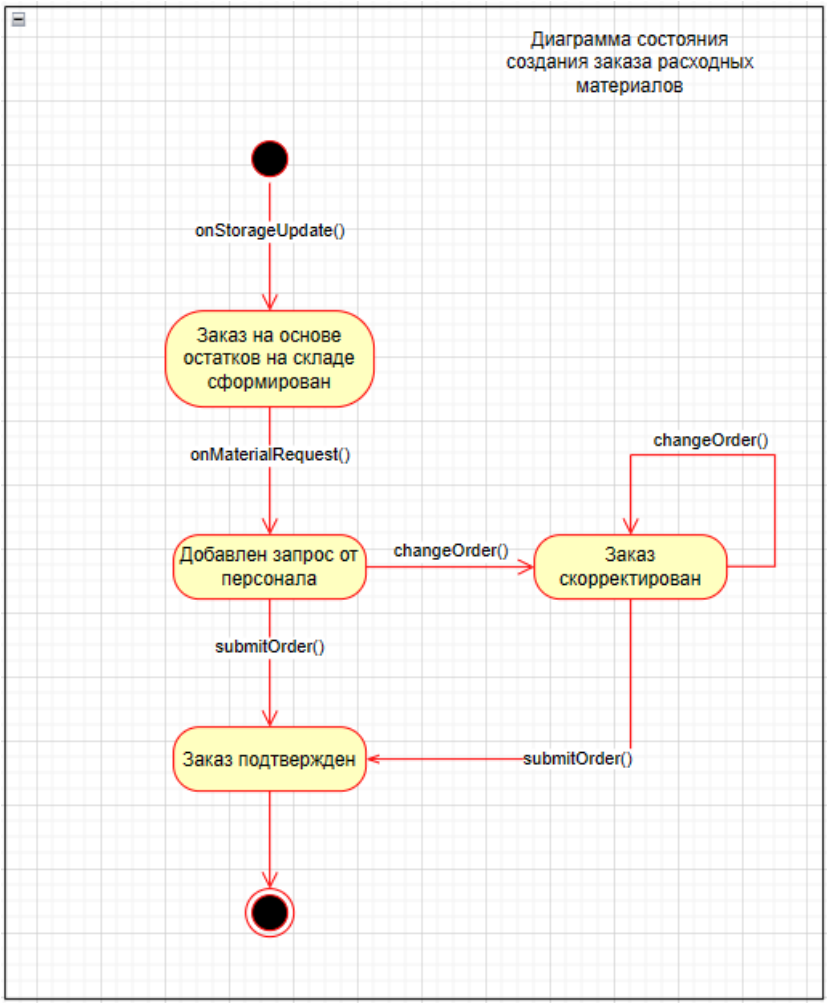

Рисунок 35. Диаграмма состояний заказа расходных материалов.

При совершении действия onStorageUpdate() система переходит из начального состояния в состояние «Заказ на основе остатков на складе сформирован». Затем к заказу добавляются запросы материалов от персонала – система переходит в состояние «Добавлен запрос от персонала». Заказ может быть изменен администратором, при наступлении события changeOrder() система переходит в состояние «Заказ скорректирован». Затем заказ может быть подтвержден администратором, переход в состояние «Заказ подтвержден» происходит при наступлении события submitOrder()», после чего осуществляется переход в заключительное состояние.

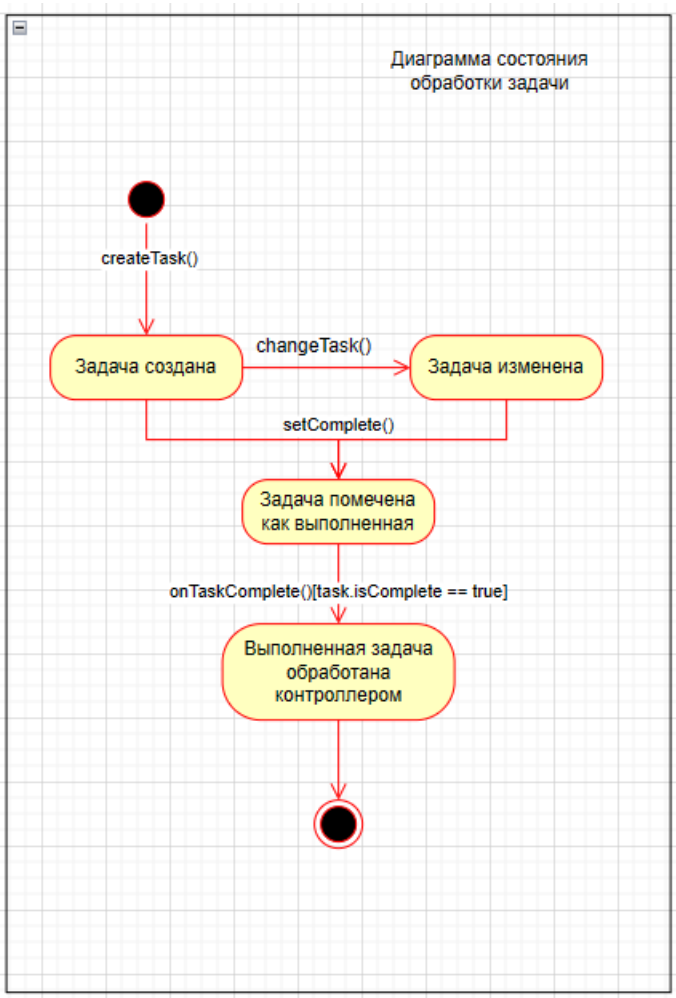

Рисунок 36. Диаграмма состояний задачи персонала.

Задача создается при наступлении события createTask() – система переходит в состояние «Задача создана». Со временем задача может изменяться – при наступлении события changeTask() система переходит в состояние «Задача изменена». Из состояний «Задача создана» и «Задача изменена» при наступлении события setComplete(), которое устанавливает флаг выполнения задачи в true, система переходит в состояние «Задача помечена как выполненная». При переходе в данное состояние автоматически создается события onTaskComplete() с защитным условием, проверяющим, что задача выполнена. При наступлении данного события система переходит в состояние «Выполненная задача обработана контроллером», после чего осуществляется переход в заключительное состояние.

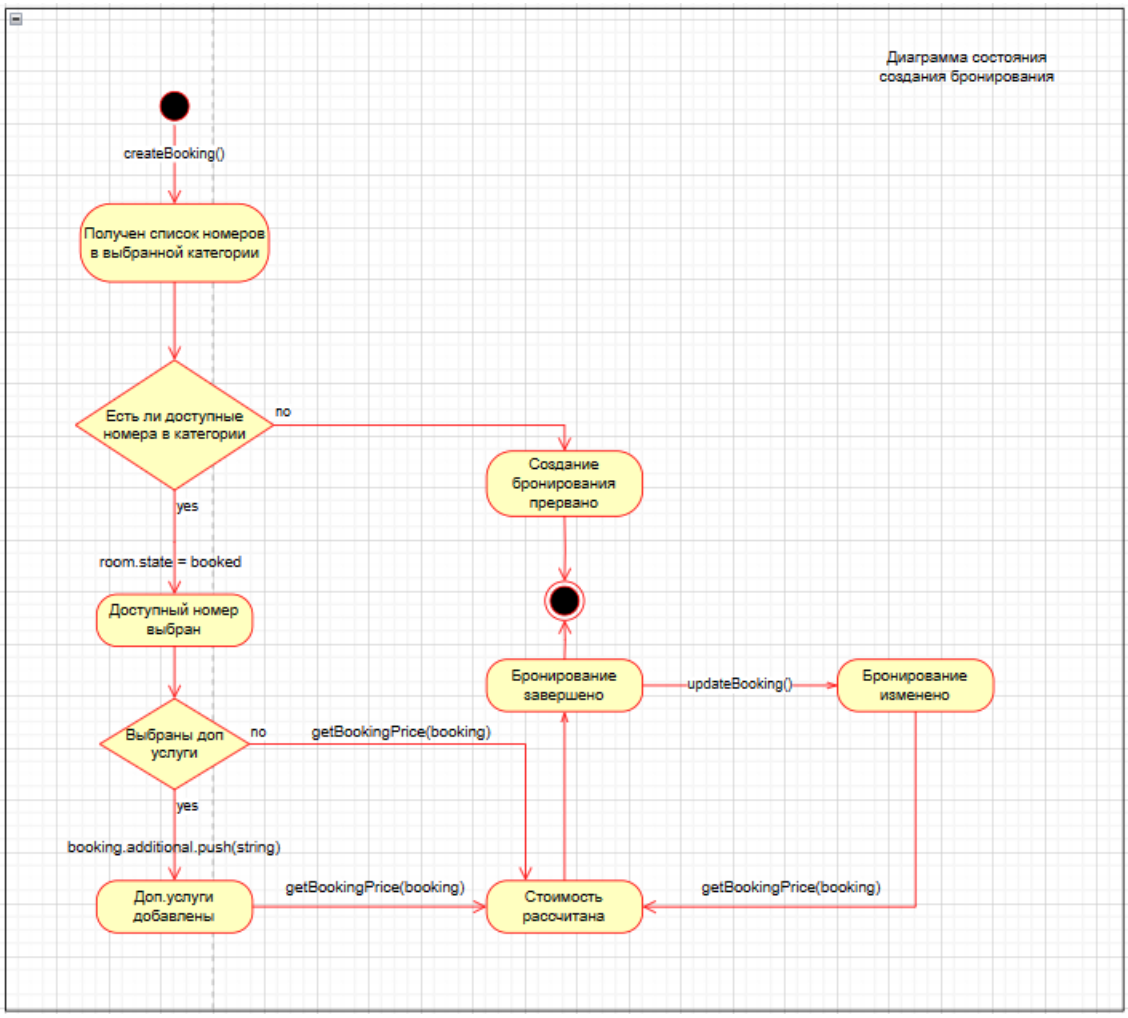

Рисунок 37. Диаграмма состояний создания бронирования.

При наступлении события createBooking() система из начального состояния переходит в состояние «Получен список номеров в выбранной категории». Далее происходит проверка на наличие доступных номеров в категории. В случае отсутствия доступных номеров система переходит в состояние «Создание бронирования прервано», после чего переходит в конечное состояние. Если номера доступны, то у любого доступного номера состояние меняется на «забронирован», после чего система переходит в состояние «Доступный номер выбран». Затем происходит проверка выбранных дополнительных услуг. Если дополнительных услуг нет, то бронирование инициализирует событие calculatePrice(), после чего система последовательно переходит в состояния «Стоимость рассчитана» и «Бронирование завершено». В случае, если дополнительные услуги были выбраны, то они добавляются в бронирование и система переходит в состояние «Доп.услуги добавлены», после

чего при наступлении события calculatePrice() система переходит в состояние «Стоимость рассчитана», а затем и в состояние «Бронирование завершено». Т.к. бронирование может изменяться, то при наступлении события changeBooking(), система переходит в состояние «Бронирование изменено». После этого при наступлении события calculatePrice(), система возвращается в состояние «Стоимость рассчитана». Последующие переходы аналогичны описанным выше. Из состояния «Бронирование завершено» осуществляется переход в конечное состояние.

3.6. Диаграмма деятельности.

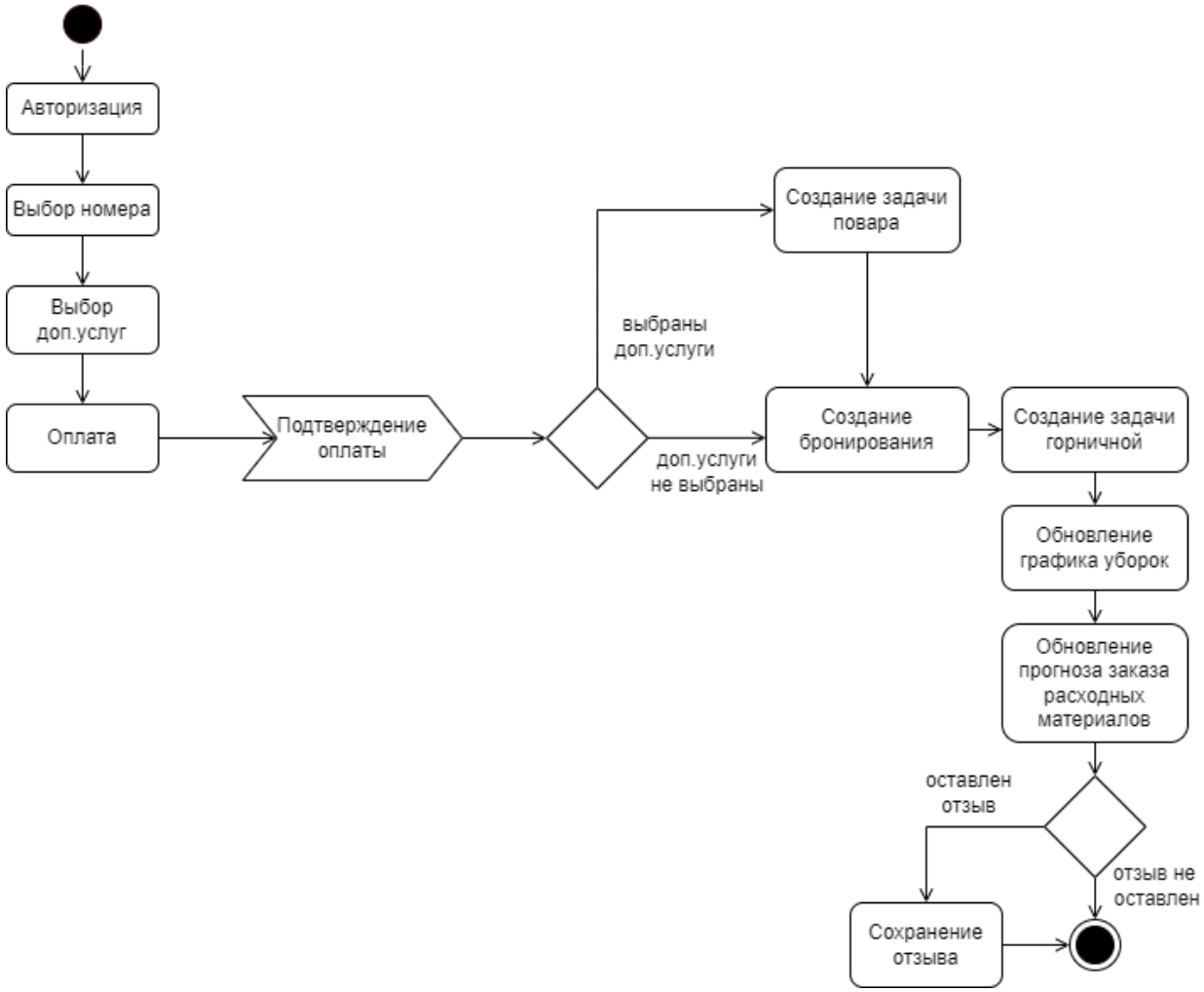

Рисунок 38. Диаграмма деятельности.

На данной диаграмме представлена диаграмма деятельности при бронировании комнаты.

После авторизации клиент выбирает номер и дополнительные услуги. После он производит оплату и система дожидается подтверждения об оплате. Затем создается бронь, а если были выбраны доп.услуги, еще создаются задачи повара.

После создаются задачи для горничных, с обновлением графика уборок, проверяется содержимое склада - хватает ли там расходных материалов для оказания этих услуг и обновляется прогноз заказа.

В конце клиент может оставить отзыв, после осуществляется переход в заключительное состояние.

3.7. Диаграммы деятельности.

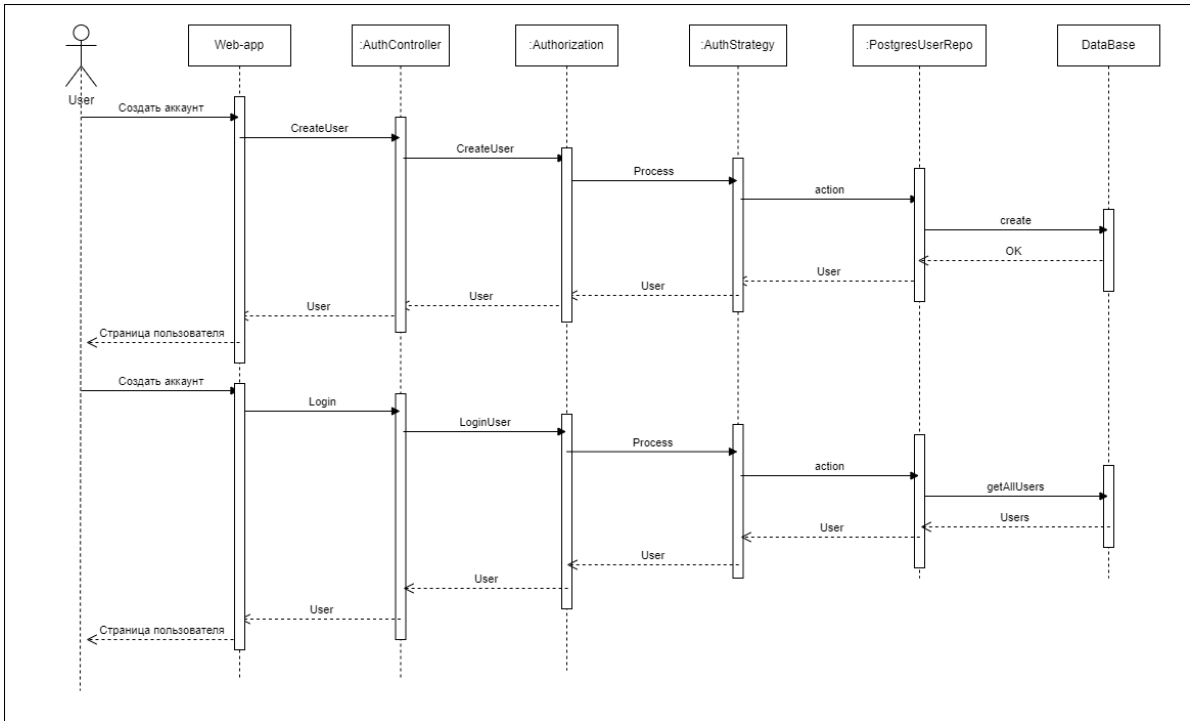

Рисунок 39. Диаграмма последовательности создания аккаунта/входа в приложение.

На данной диаграмме представлена последовательность действ при авторизации пользователя в приложении или при создании аккаунта. При создании аккаунта приложение отправляет запрос к классу AuthController у которого вызывается метод createUser в который передана информация о пользователе, далее будет вызван метод класса Authorization process, который в свою очередь поставит соответствующую стратегию SignUpStrategy у которой вызовет метод action. Стратегия обратиться к PostgresUserRepo вызвав метод create. Данный метод отправляет запрос на создание пользователя в базе данных, база данных соответственно создает в таблицах кортеж с необходимыми значениями и возвращает статус OK. Далее метод создает по данным соответствующий аккаунт вида User и возвращает его в Web-app, где будет отображена страничка пользователя.

Авторизация устроена схоже с созданием аккаунта. Веб-приложение обращается к серверу через контроллер, который запрашивает данные у Authorization, использующего стратегию LoginStrategy. С помощью метода action запрашиваются данные из репозитория, который обращается к БД. После

получения данных они возвращаются в приложение, где отображаются в виде странички пользователя.

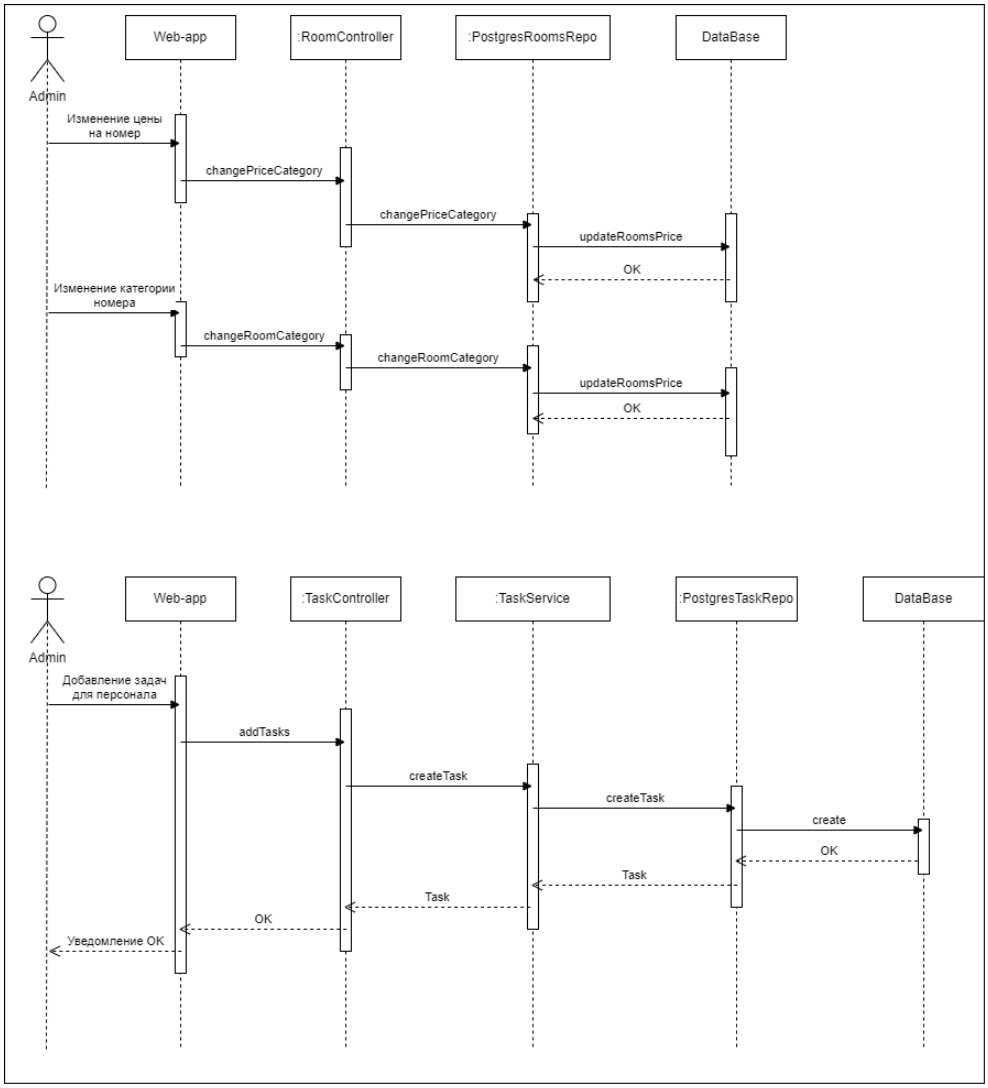

Рисунок 40. Диаграмма последовательности основных обязанностей администратора.

Администратор может добавлять задачи для персонала обращаясь через веб-приложение к TaskController, который дает поручение TaskService, чтобы была создана задача и отправлена в БД через PostgresTaskRepo. В ответ администратору приходит успешное добавление задачи для персонала. Также Администратор может обновлять категории и цены для комнат. Через приложение он обращается к RoomController, который в своих метода вызывает методы PostgresRoomRepo и записывает измененную информацию в БД.

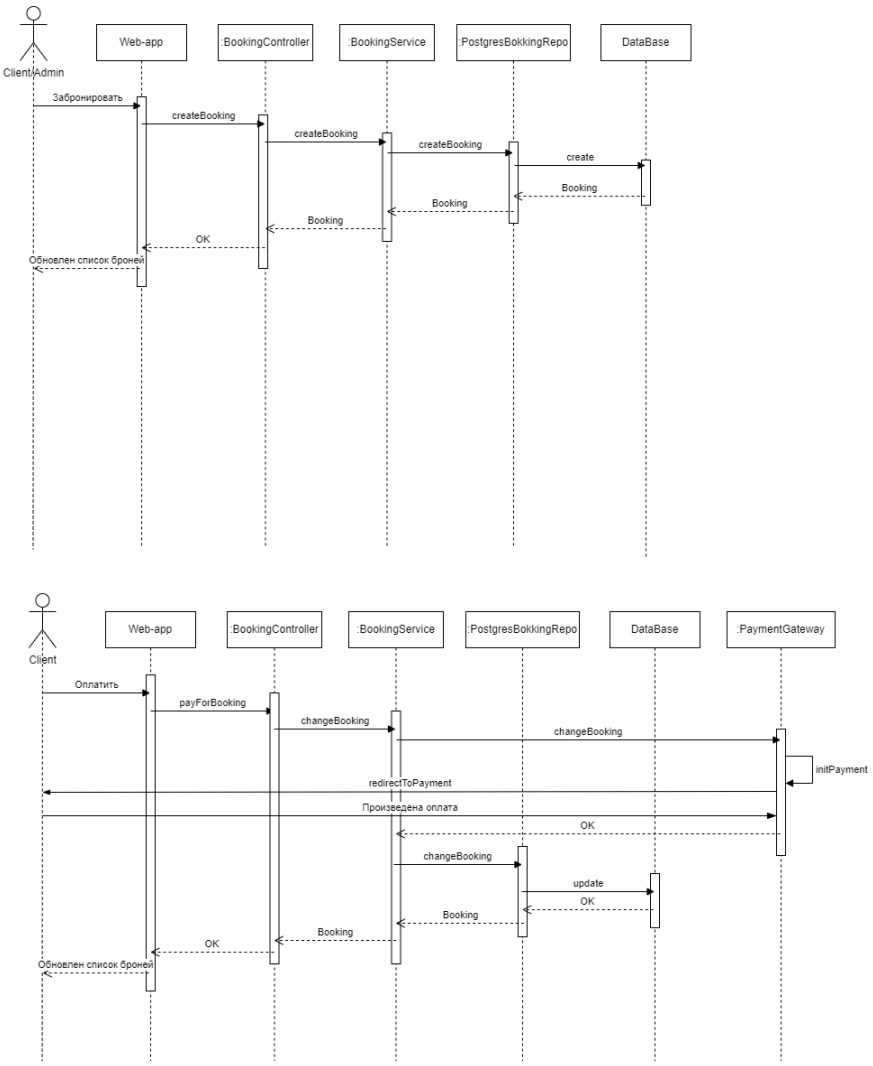

Рисунок 41. Диаграмма последовательности бронирования и оплаты номеров.

Так как бронировать может и администратор, и клиент, то у них соответственно одинаковый функционал. Через приложение вызываются методы BookingController, BookingService на создание бронирования. В PostgresBookingRepo создается информация о бронировании в базе данных. При успешном бронировании пользователь получает уведомление о том, что бронирование успешно добавлено, но пока еще не оплачено.

Клиент сразу же в приложении может оплатить бронирование номера. При этом приложение вызовет changeBooking в BookingController, который вызовет changeBooking в BookingService. Внутри будет инициализирована оплата с помощью метода initPayment объекта PaymentGateway. Данный метод будет переадресовывать пользователя на сторонний сайт оплаты, и после получения уведомления об успешном прохождении оплаты в методе changeBooking будет

вызван метод PostgresBookingRepo update который обновит данные о данном бронировании в БД и вернет соответствующее бронирование в обновленном виде.

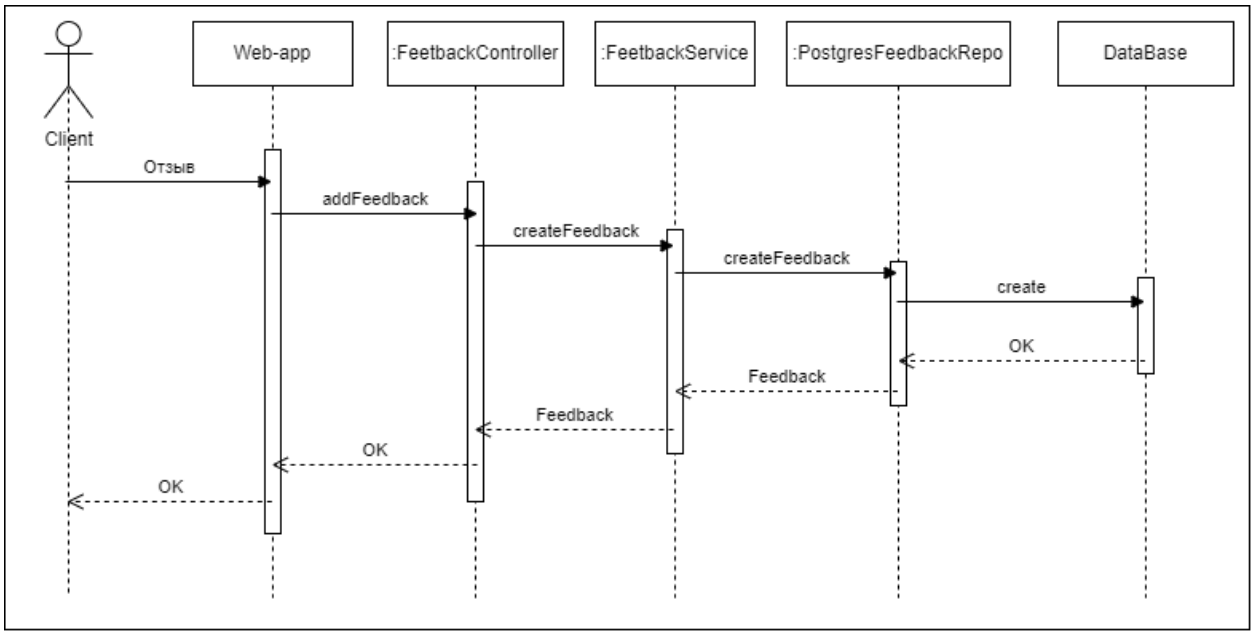

Рисунок 42. Диаграмма последовательности отправки отзывов.

Пользователь может оставить отзыв через приложение. Вызывается метод createFeedback объекта FeedbackController, который вызывает одноименный метод FeedbackService, который обратится к PostgresFeedbackRepo. Репозиторий сохранит информацию в БД, а затем пользователю придет ответ, что отзыв успешно сохранен.

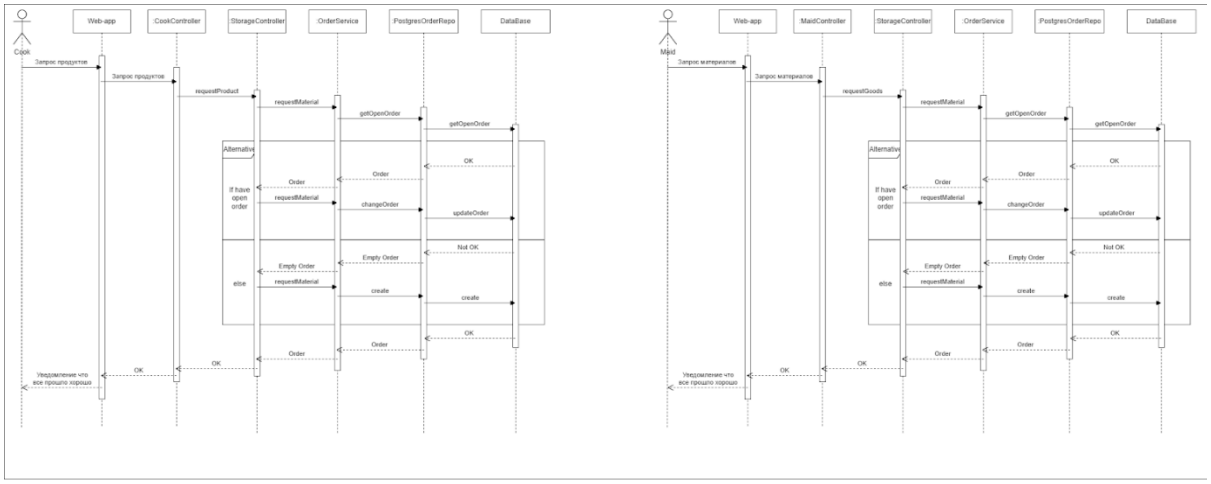

Рисунок 43. Диаграмма последовательности запроса продуктов/расходных материалов для персонала.

Для поваров и горничных алгоритм примерно одинаковый, единственное различие в контроллере, который отвечает на запросы от приложения. Для поваров этом занимается CookController, для горничных этим занимается MaidController. Процесс запроса выглядит следующим образом. Метод requestProduct/requestGoods объекта CookController/MaidController вызывает метод requestMaterial объекта StorageController. Последний в свою очередь обращается к OrderService, который с помощью метода getOpenOrder через репозиторий последний открытый заказ. Если таковой имеется, тогда в requestMaterial будет вызван метод changeOrder объекта OrderService, который изменит текущий заказ через репозиторий и затем отправится уведомление о том, что все прошло успешно отправится работнику. Если же еще нет открытого заказа, то в requestMaterial будет вызван метод create объекта OrderService, который через репозиторий создаст соответственную запись в БД. Работнику вернется информация о том, что все прошло успешно.

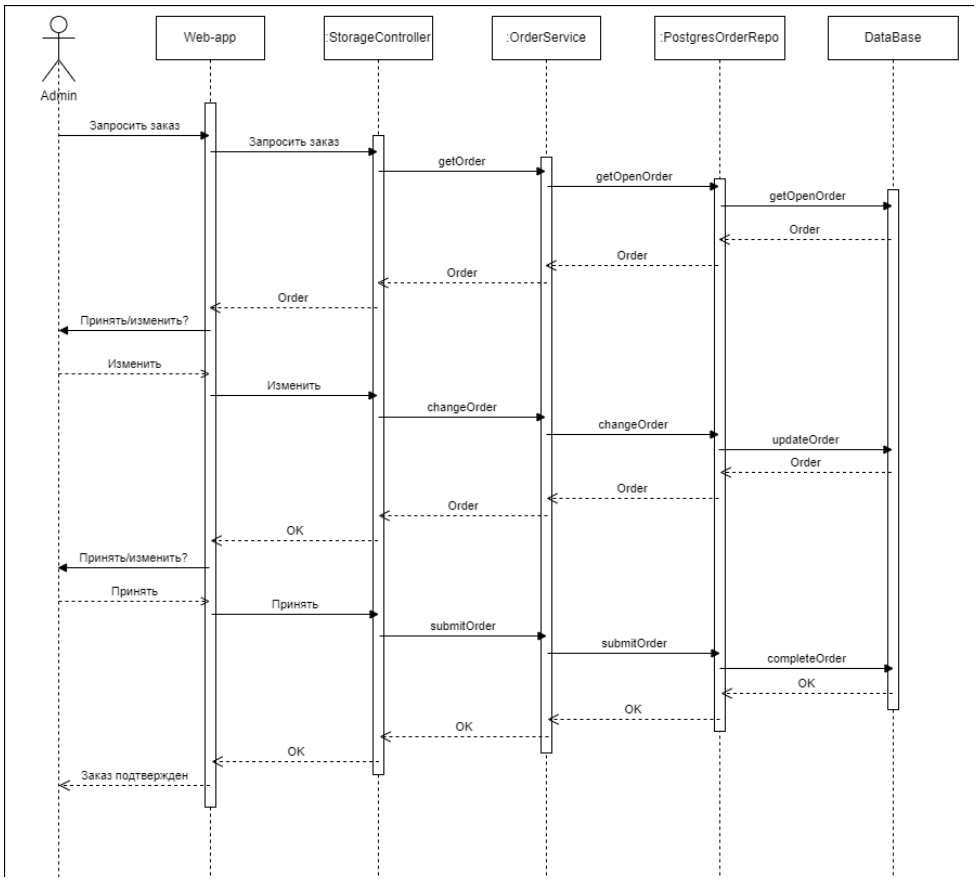

Рисунок 44. Диаграмма последовательности принятия/изменения заказа администратором.

Администратор запрашивает информацию о последнем заказе, через приложение, которое обращается к StorageController, который в свою очередь смотрит последний открытый заказ. При отображении его на странице администратора, последнем у предлагается изменить его или одобрить. При изменении данные отправляются на StorageController и вызывается метод changeOrder. Информация дальше отправляется на OrderService, который с помощью репозитория обновляет данные в базе данных. После обновления данных, веб-приложение показывает их администратору и снова запрашивает, одобрения или изменения.

Как только администратор будет уверен в правильности данных, он отправляет запрос на подтверждение заказа. Внутри приложения отправляется запрос на то, что можно закрыть данный заказ. Соответственно StorageController вызывает в методе submitOrder одноименный метод OrderService, который проси репозиторий подтвердить заказ. После подтверждения заказа администратору отправляется подтверждение об успешном прохождении данной операции и скачивание заказа в формате PDF.

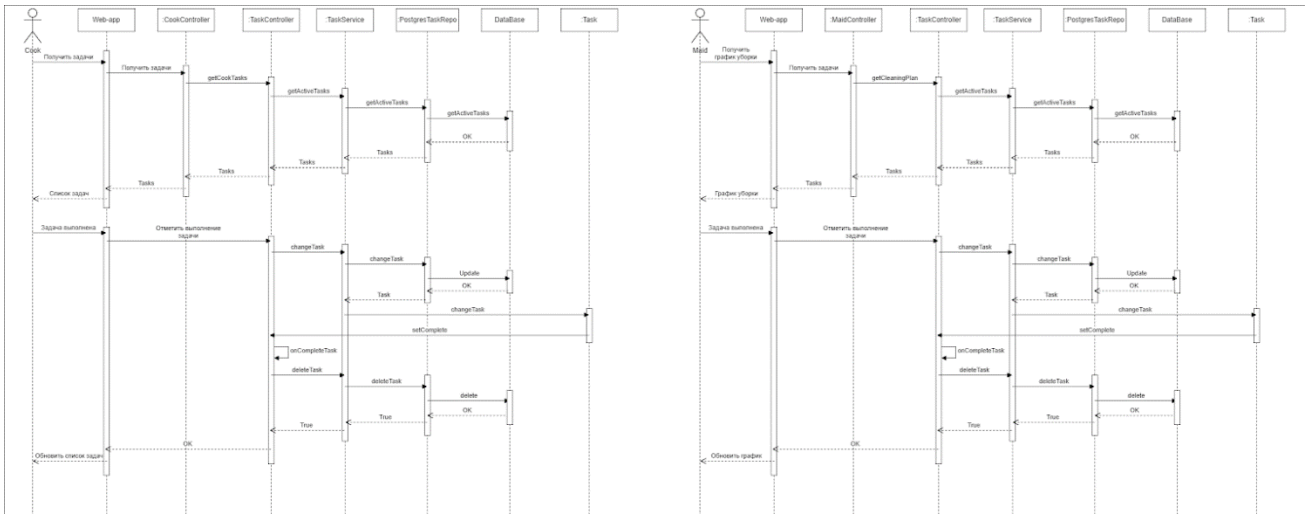

Рисунок 45. Диаграмма последовательности задач персонала и отслеживание их выполнения.

Персонал гостиницы может получить информацию о своих задачах через приложение обратившись к MaidController и CookController для горничных и поваров соответственно. Данные объекты отправлят запрос в TaskController с просьбой выслать задачи. TaskController обращается к TaskService, который

через репозиторий получает доступные задачи. Они возвращаются на страницу персонала.

После получения задач персонал может отмечать их выполнение. При отметке информация отправляется в TaskController и у поваров, и у горничных. Состояние задачи обновляется сначала в базе данных через методы changeTask у TaskController и TaskService. Последний вызвает обновление задачи в репозитории. Затем TaskService обновляет объект Task. При изменении состояния Task TaskController вызывет onCompleteTask, в котором вызывается метод удаления задачи из БД через TaskService. После успешного удаления Задачи персоналу приходит информация о том, что задача выполнена и список задач обновляется.

3.8. Диаграммы коммуникаций.

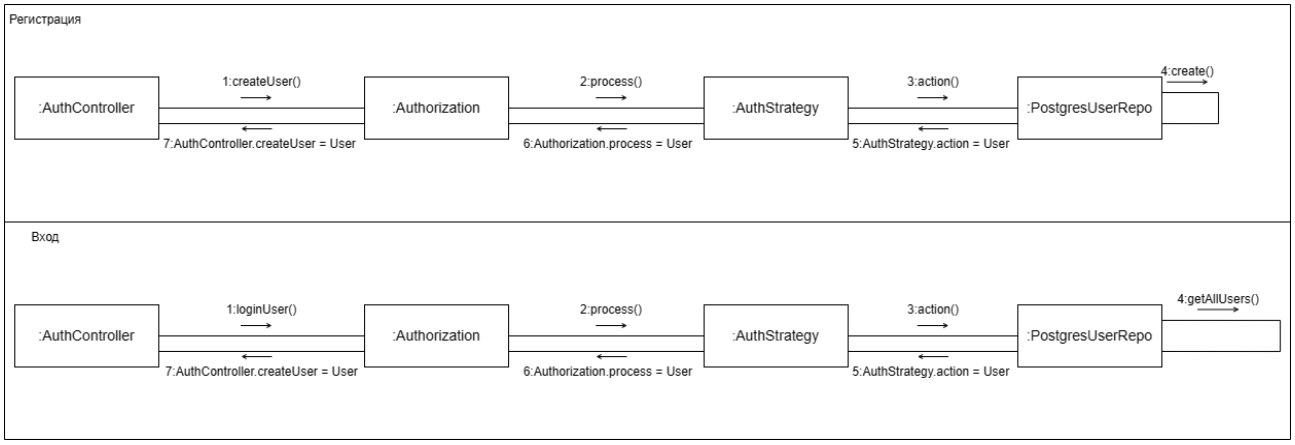

Рисунок 46. Диаграмма коммуникаций создания аккаунта/входа.

На данных диаграммах представлена последовательность действий при авторизации.

Последовательность действий при создании аккаунта. Создание аккаунта реализовано методом createUser() экземпляра класса AuthController. Запрос отсылается в экземпляр класса Authorization, который в свою очередь запускает процесс авторизации методом process(). Далее запрос направляется в AuthStrategy, который реализует метод action(), в зависимости от типа авторизации, в данном случае создание аккаунта. Далее запрос уходит в экземпляр класса PostgresUserRepo, внутри которого методом create() аккаунт создается и сохраняется в базе данных. Далее в обратном порядке в каждый из использованных методов возвращается переменная-пользователь User (реализация интерфейса User) для конечного отображения в приложении/на сайте.

Последовательность действий при входе в аккаунт. Вход в аккаунт реализован методом loginUser() экземпляра класса AuthController. Запрос отсылается в экземпляр класса Authorization, который в свою очередь запускает процесс авторизации методом process(). Далее запрос направляется в AuthStrategy, который реализует метод action(), в зависимости от типа авторизации, в данном случае вход. Далее запрос уходит в экземпляр класса PostgresUserRepo, внутри которого методом getAllUsers() просматриваются все аккаунты в базе данных и с помощью фильтрации возвращается нужный. Далее

в обратном порядке в каждый из использованных методов возвращается переменная-пользователь User (реализация интерфейса User) для конечного отображения в приложении/на сайте.

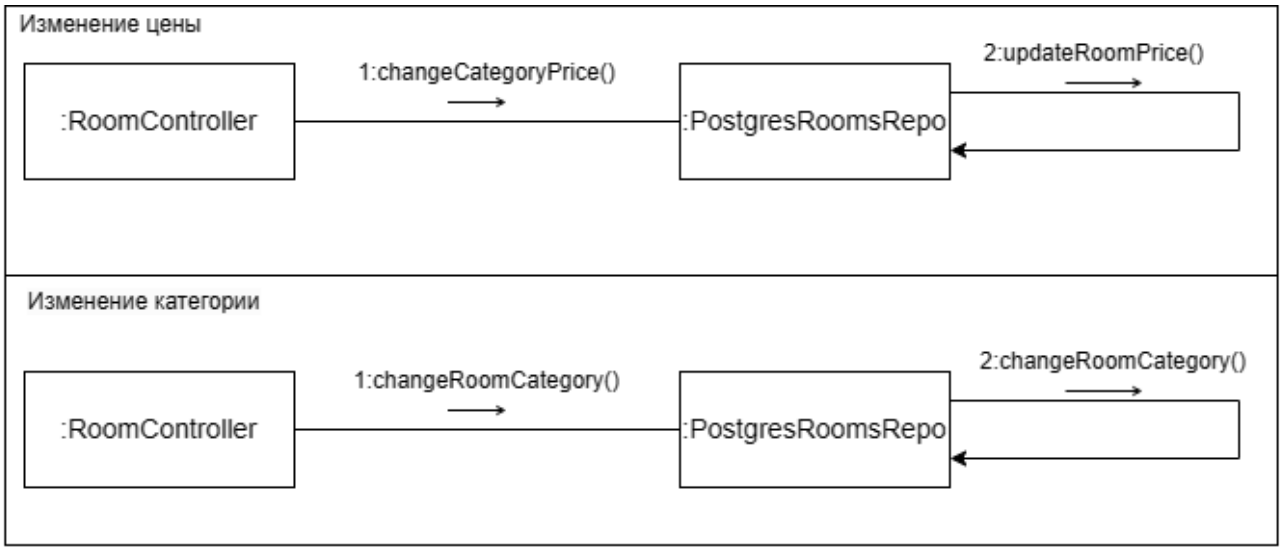

Рисунок 47. Диаграмма коммуникаций изменения цен и категорий номеров.

На данных диаграммах представлена последовательность действий при изменении цен на категории номеров и изменении категорий номеров.

Последовательность действие при изменении цен на категории номеров. Изменение цены номера происходит методом changeCategoryPrice() экземпляра класса RoomController. Далее запрос уходит в экземпляр класса PostgresRoomsRepo, внутри которого методом updateRoomPrice() цена категории номеров изменяется и сохраняется в базе данных.

Последовательность действие при изменении категории номера. Изменение категории номера происходит методом changeRoomCategory() экземпляра класса RoomController. Далее запрос уходит в экземпляр класса PostgresRoomsRepo, внутри которого методом changeRoomCategory() категория номера изменяется и сохраняется в базе данных.

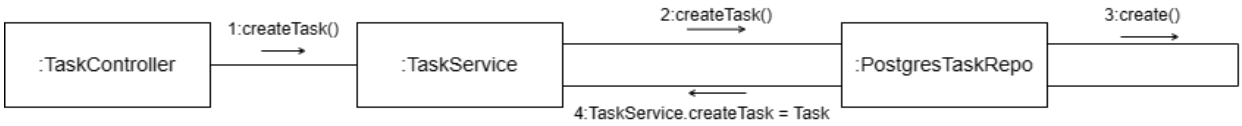

Рисунок 48. Диаграмма коммуникаций добавления задач персоналу.

На данной диаграмме представлена последовательность действий при добавлении администратором задач для персонала. Добавление задачи реализовано методом createTask() экземпляра класса TaskController. Запрос отсылается в экземпляр класса TaskService, который в свою очередь запускает сам процесс создания добавляемой задачи createTask(). Далее запрос уходит в экземпляр класса PostgresTaskRepo, внутри которого методом create() задача создается и сохраняется в базе данных. Далее в экземпляр класса TaskService в использованный метод возвращается переменная-задача Task (реализация класса Task) для успешного добавления задачи.

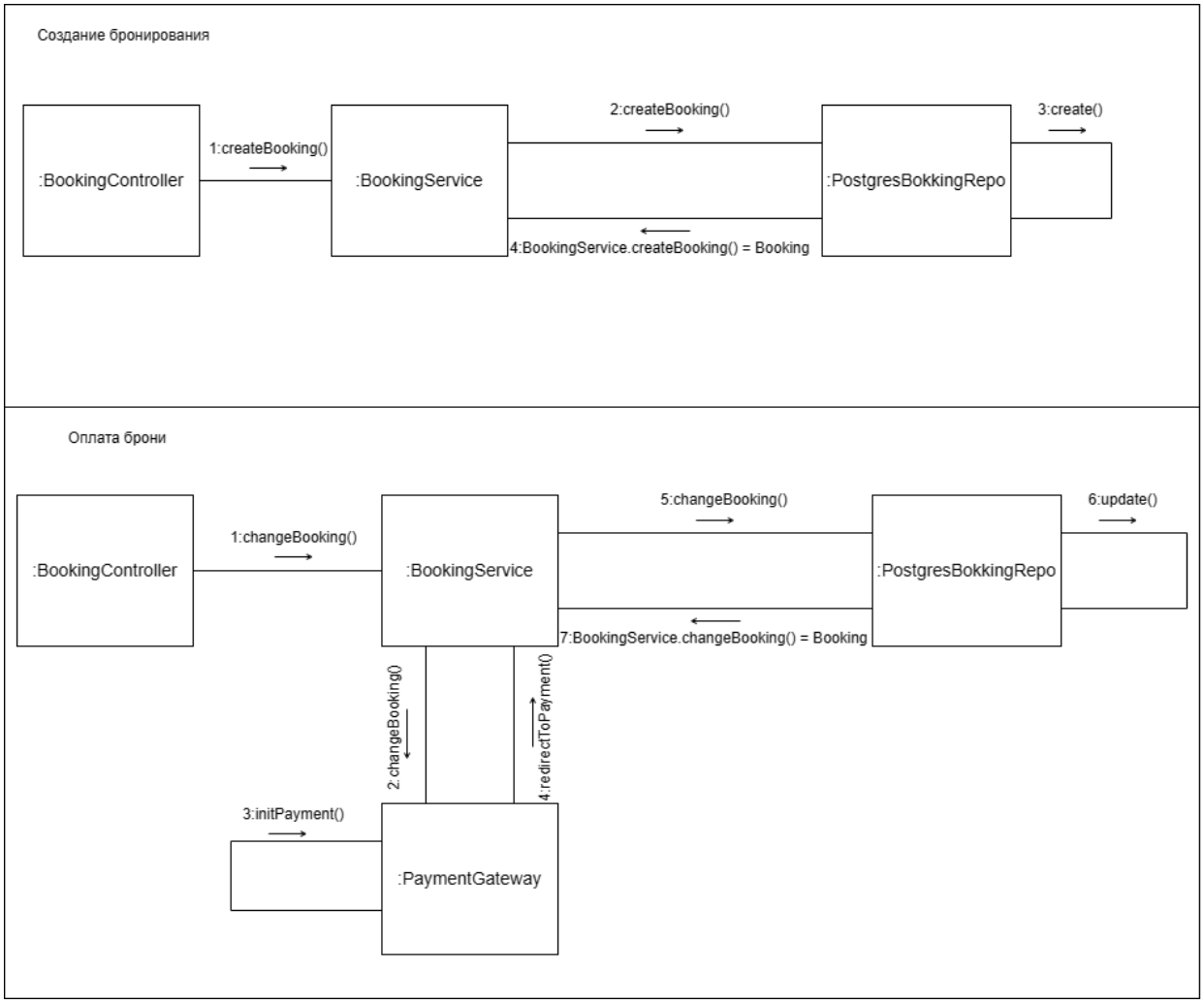

Рисунок 49. Диаграмма коммуникаций создания и оплаты бронирования.

На данных диаграммах представлена последовательность действий при создании бронирования и его оплате.

Последовательность действий при создании бронирования. Создание бронирования происходит методом createBooking() экземпляра класса BookingController. Для выполнения запрос передается в экземпляр класса BookingService, где вызывается createBooking() для синхронизации с базой данных. Далее запрос поступает в экземпляр класса PostgresBookingRepo, где create() создает новую бронь и добавляет ее в базу данных. Обратно отправляется переменная-бронь Booking экземпляру BookingService. Это подтверждает, что бронь была создана.

Последовательность действий при оплате брони. Для оплаты бронирования необходимо внести изменение в само бронирование, то есть внести информацию, что оно оплачено. Методом changeBooking() экземпляра класса BookingController посылается запрос на изменении. Для выполнения запрос передается в экземпляр класса BookingService, где вызывается changeBooking() для совершения операции оплаты. В ходе запроса в экземпляре класса PaymentGateway происходит инициализация оплаты initPayment() и обратно отправляется запрос в BookingService с информацией о том, что был совершен переход на страницу оплаты и она была совершена методом redirectToPayment(). Для внесения информации об оплате в базу, в BookingService вызывается changeBooking(). Далее запрос поступает в экземпляр класса PostgresBookingRepo, где update() создает новую бронь и добавляет ее в базу данных. Обратно отправляется переменная-бронь Booking экземпляру BookingService. Это подтверждает, что бронь была изменена, а именно внесена оплата.

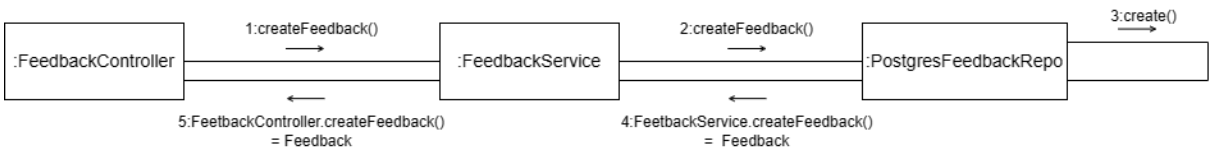

Рисунок 50. Диаграмма коммуникаций отправки отзывов.

На данной диаграмме представлена последовательность действий при составлении клиентом отзыва. Создание отзыва реализовано методом createFeedback() экземпляра класса FeedbackController. Запрос отсылается в экземпляр класса FeedbackService, который в свою очередь запускает сам процесс создания отзыва createFeedback(). Далее запрос уходит в экземпляр класса PostgresFeedbackRepo, внутри которого методом create() отзыв создается и сохраняется в базе данных. Далее в обратном порядке в каждый из использованных методов возвращается переменная-отзыв Feedback (реализация класса Feedback) для успешного добавления отзыва.

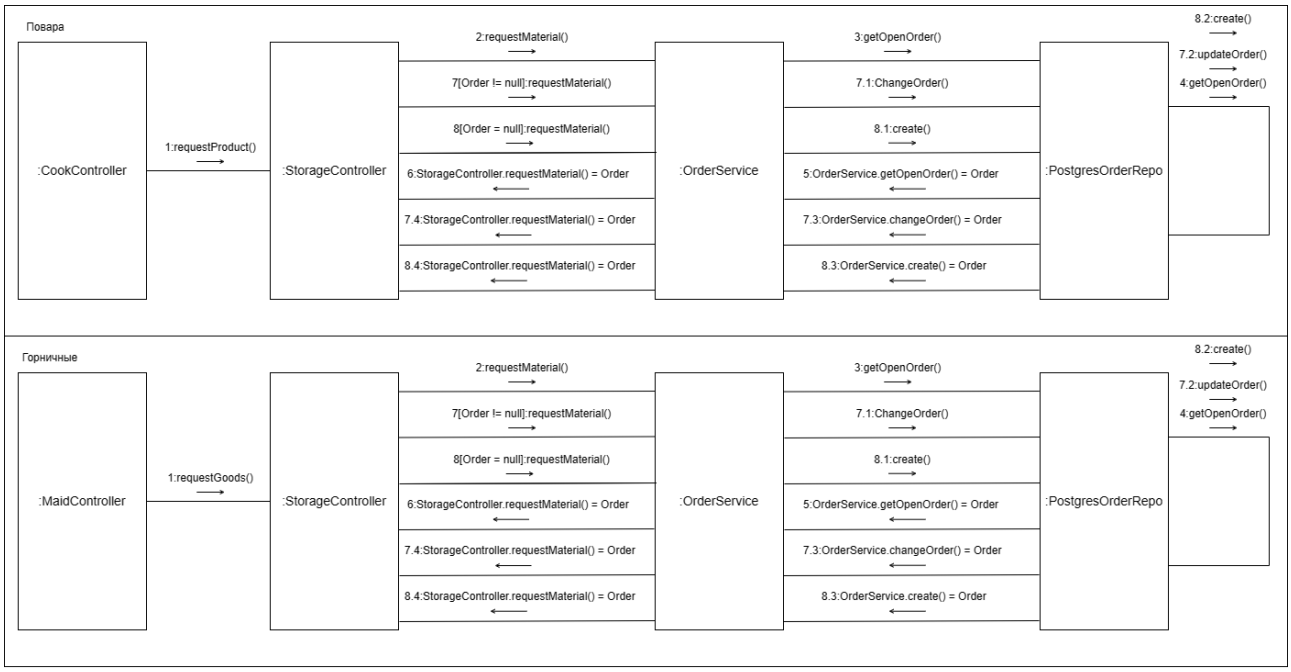

Рисунок 51. Диаграмма коммуникаций запроса расходных материалов персоналом.

На данных диаграммах представлена последовательность действий при запросе расходных материалов от персонала.

Формирование запроса от поваров. Для формирования запроса от поваров в экземпляре класса CookController вызывается метод requestProduct(). Запрос переходит в класс StorageController, который вызывает метод requestMaterial(), чтобы сформировать запрос. При данном запросе в начале проверяется наличие

открытого заказа, для этого метод getOpenOrder() класса OrderService отправляет запрос в базу данных. В реализации базы данных PostgresOrderRepo методом getOpenOrder получаем последний закрытый заказ, если такой существует. Обратно в предыдущие два запроса передается переменная-заказ Order. В StorageController два варианта, которые реализуются после получения Order. Если Order не null, то посылается опять запрос на создание запроса расходных материалов, далее методом ChangeOrder класса OrderService будет отправлен запрос на изменение последнего открытого заказа в базе данных, метод updateOrder реализованный в PostgresOrderRepo добавляет в базе данных к последнему заказу сформированный запрос. Для подтверждения сохранения изменений в обратном порядке посылаются запросы, передающие текущий заказ Order. Если Order null, то посылается опять запрос на создание запроса расходных материалов, далее методом create класса OrderService будет отправлен запрос на создание нового заказа в базе данных, метод create реализованный в PostgresOrderRepo создает новый заказ в базе данных. Для подтверждения создания нового заказа в обратном порядке посылаются запросы, передающие текущий заказ Order.

Формирование запроса от горничных выглядит аналогично, только первый запрос посылает экземпляр сервиса MaidController с помощью requestGoods().

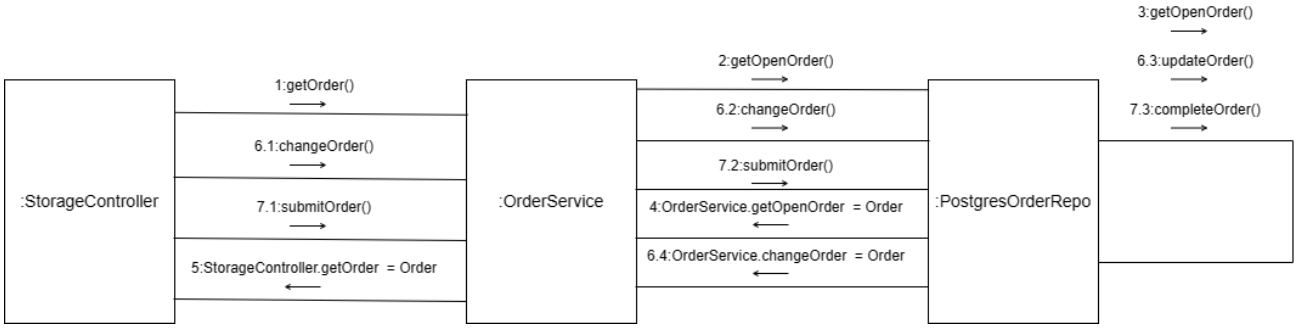

Рисунок 52. Диаграмма коммуникаций принятия/изменения заказа администратором.

На данной диаграмме представлена последовательность действий при изменении/утверждении заказа администратором.

Для начала необходимо получить какой-либо заказ. С помощью метода getOrder() класса StorageController посылается запрос на получение заказа.

Запрос обрабатывается в экземпляре класса OrderService, который методом getOpenOrder() посылает запрос в класс, связанный с базой данных, для получения открытых заказов. В PostgresOrderRepo заказ получаем из базы данных методом getOpenOrder(), затем в обратном порядке экземплярам классов посылается запрос с переменной, содержащий Order – заказ, для дальнейшей обработки.

Изменение заказа. С помощью метода changeOrder() класса StorageController посылается запрос редактирования заказа. Запрос обрабатывается в экземпляре класса OrderService, который методом changeOrder() посылает запрос в класс, связанный с базой данных, для возможности редактирования заказа в базе данных. В PostgresOrderRepo заказ редактируется в базе данных методом getOpenOrder(), затем в обратном порядке экземплярам классов посылается запрос с переменной, содержащий Order – заказ, для подтверждения того, что заказ был отредактирован.

Подтверждение заказа. С помощью метода submitOrder() класса StorageController посылается запрос подтверждения заказа. Запрос обрабатывается в экземпляре класса OrderService, который методом submitOrder() посылает запрос в класс, связанный с базой данных, для возможности отметить заказ как подтвержденный в базе данных. В PostgresOrderRepo заказ становится подтвержденным в базе данных методом completeOrder().

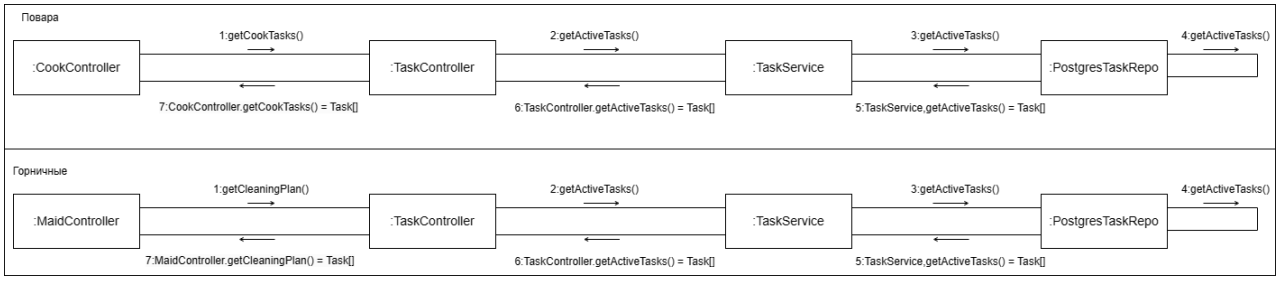

Рисунок 53. Диаграмма коммуникаций получения задач персоналом.

На данных диаграммах представлена последовательность получения задач персоналом.

Последовательность действий при получении задач поварами. Получение зада реализовано методом реализовано методом getCookTasks() экземпляра класса CookController. Запрос отсылается в экземпляр класса TaskController, который в свою очередь запускает процесс получения актуальных задач для поваров getActiveTasks(). Далее запрос направляется в TaskService, который реализует метод getActiveTasks() для отправки запроса в базу данных. Далее запрос уходит в экземпляр класса PostgresTaskRepo, внутри которого методом getActiveTasks() из базы данных подтягиваются актуальные задачи поваров. Далее в обратном порядке в каждый из использованных методов возвращается переменная-набор задач Task[] (реализация класса Task) для их итогового получения.

Последовательность действие при просмотре графика уборки (тех же самых задач) горничными последовательность аналогичная. Только последовательность действий начинается с вызова метода getCleaningPlan() экземпляра класса MaidController, который посылает запроса далее по той же цепочке, описанной ранее.

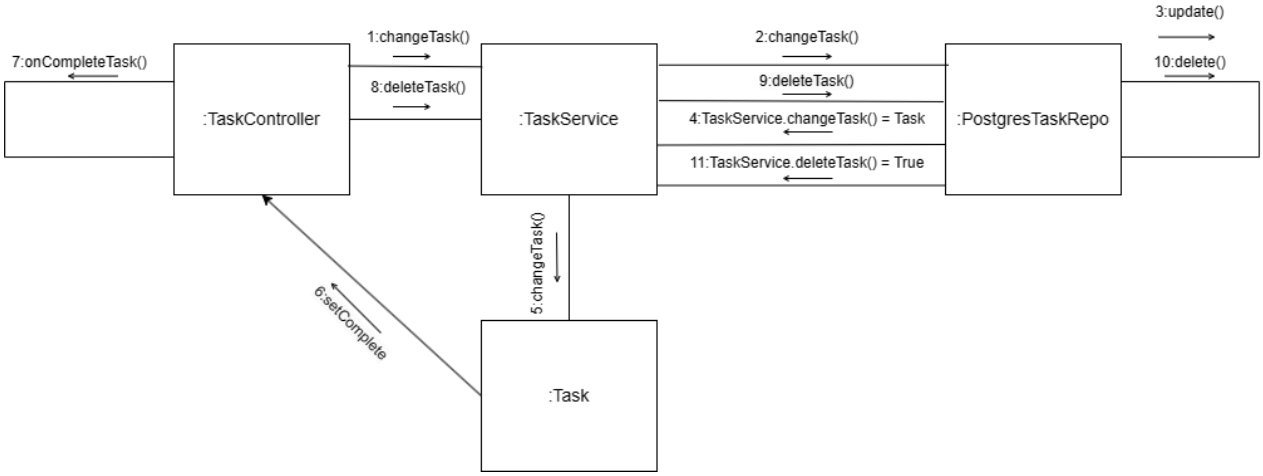

Рисунок 54. Диаграмма коммуникаций отметки выполненных задач персоналом.

На данной диаграмме представлена последовательность действий при отметке персоналом выполненных задач. Чтобы сделать задачу выполненной, в начале изменяется ее статус, поэтому у экземпляра класса TaskController вызывается метод changeTask(). Данный запрос отправляется в экземпляр класса TaskService, который с помощью метода changeTask() отправляет запрос в базу

данных. В экземпляре класса PostgresTaskRepo методом update() в базе данных у задачи изменяется статус - она теперь выполнена. Данная задача возвращается в виде результата работы функции TaskService.changeTask() = Task. Получи выполненную задачу данный класс отправляет запрос changeTask(). Данный запрос обрабатывается в экземпляре класса Task с помощью метода setComplete. Следующий запрос посылается в TaskController, который при получении и обработке данного запроса должен удалть выполненную задачу. Поэтому экземпляр класса TaskController запускает метод deleteTask(). В классе TaskService обрабатывается такой вид запроса, посылая новый в базу данных для удаления - deleteTask(). В экземпляре класса PostgresTaskRepo в бд задача удаляется методом delete() и сервису посылается флаг true, сигнализирующий, что задача была успешно была удалена после выполнения.

3.9. Диаграммы развертывания.

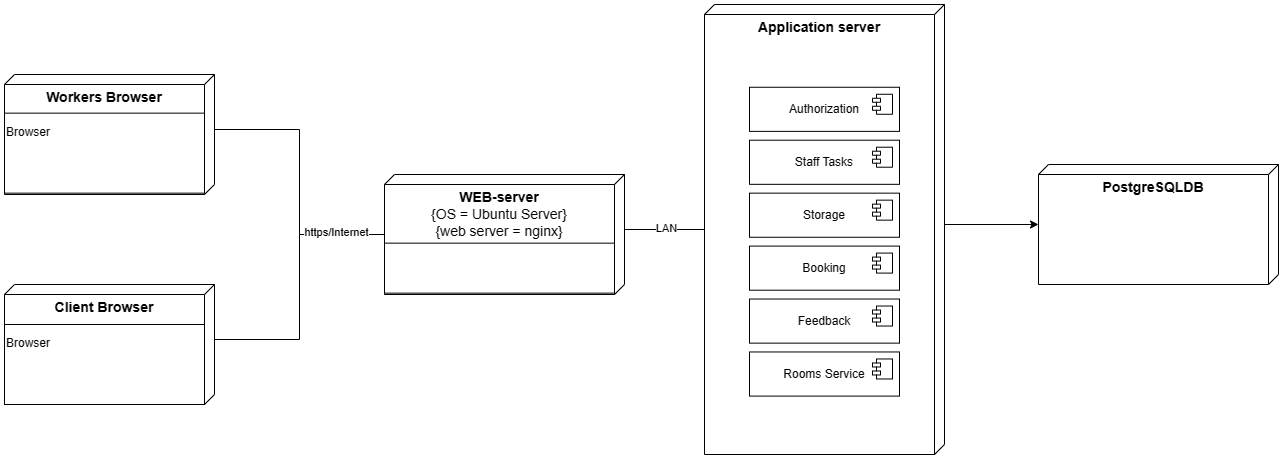

Рисунок 55. Диаграмма развертывания.

Workers browser и client browser – интерфейсы взаимодействия с системой для работников и клиентов соответственно. Пользователи (работники и клиенты) взаимодействуют с Web Server через интернет по протоколу https, в качестве самого сервера используется сервер на платформе Ubuntu Server на основе nginx из-за повышенной отказоустойчивости и недорогой эксплуатации. Web Server поддерживает интерфейсы программы для связи с основной логикой в Application Server. На самом Application Server развернуты модули системы такие как: авторизация, сервис склада, сервис для задач персоналу, сервис бронирования, сервис отзывов и сервис для работы с номерным фондом. Так как модулям необходимо хранить информацию то Application Server имеет доступ к серверу на котором развернута СУБД.