

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Программные средства для управления
гостевым сервисом в ресторане

Студент _____ Жигалов Кирилл Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 09.06.2025 № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Жигалов Кирилл Александрович гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Программные средства для управления гостевым сервисом в ресторане

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент, доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: программные средства, предназначенные для управления сервисами приема гостей в ресторане, включая выбор стола и предзаказ блюд из меню*

2) *Среда выполнения: Windows 10.*

Языковые средства разработки: JavaScript(Next.js), Python, SQL, UML.

Инструментальные средства: Visual Studio Code, Docker, PostgreSQL, Draw.io, Visual Paradigm 17.2.

Технологии разработки MVC архитектура, монолит.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) онлайн выбор стола для бронирования пользователем;

в) хранение и систематизация информации о бронях пользователя;

г) отслеживание предзаказов, сформированных пользователем;

д) анализ посещаемости пользователя;

е) визуализация схемы зала ресторана для выбора стола.

5. Перечень подлежащих разработке вопросов:

1) <i>Анализ предметной области и постановка задачи;</i>
2) <i>Проектирование;</i>
3) <i>Реализация;</i>
4) <i>Контроль качества программного обеспечения;</i>
5) <i>Оформление пояснительной записки.</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025– 13.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 173 листа, 30 рисунков, 40 таблиц, 25 использованных источников и 4 приложения.

УПРАВЛЕНИЕ ГОСТЕВЫМ СЕРВИСОМ В РЕСТОРАНЕ, MVC АРХИТЕКТУРА, ВЕБ-ПРИЛОЖЕНИЕ, REST, UML, POSTGRESQL, PYTHON, NEXT.JS, БАЗА ДАННЫХ, REACT.

Объект исследования – программные средства для управления гостевым сервисом в ресторане.

Цель выпускной квалификационной работы – разработать программные средства для управления гостевым сервисом в ресторане, обеспечивающие возможность выбора стола онлайн и предзаказа блюд.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования JavaScript с использованием фреймворка Next.js и библиотеки React в среде разработки Visual Studio Code. Разработка серверной части приложения производилась на языке программирования Python с использованием MVC архитектуры в среде разработки PyCharm. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – Windows 10, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть программных средств для управления гостевым сервисом в ресторане. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР212.07 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства для управления гостевым сервисом в ресторане Пояснительная записка	Лит.	Лист	Листов
Разраб.		Жигалов К.А.					4	173
Пров.		Балашова И.Ю.				Группа 21ВП2		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ программных средств для управления гостевым сервисом в ресторане	9
1.1 Анализ предметной области управления гостевым сервисом в ресторане	9
1.2 Обзор и анализ программных средств для управления гостевым сервисом в ресторане	10
1.2.1 Веб-сайт ресторана «Tom Soy».....	10
1.2.2 Веб-сайт ресторана «NOVIKOV».....	10
1.2.3 Веб-сайт ресторана «IL Patio»	11
1.2.4 Веб-сайт ресторана «Pitnica»	12
1.2.5 Сравнение программных средств для управления гостевым сервисом в ресторане	14
1.3 Постановка задачи на разработку программных средств.....	15
1.4 Анализ функциональных требований	16
1.5 Анализ нефункциональных требований	19
1.6 Планирование разработки и оценка бюджета.....	20
2 Проектирование программных средств для управления гостевым сервисом в ресторане	24
2.1 Архитектура приложения.....	24
2.2 Проектирование структур данных	25
2.2.1 Концептуальная модель базы данных	25
2.2.2 Логическая модель базы данных	31
2.3 Проектирование пользовательского интерфейса.....	32
3 Реализация программных средств для управления гостевым сервисом в ресторане	34
3.1 Реализация базы данных	34
3.2 Реализация серверной части приложения	43
3.3 Реализация клиентской части приложения	45

3.4 Реализация интерфейса приложения	47
4 Контроль качества программного обеспечения.....	60
4.1 Обоснование выбора режима тестирования.....	60
4.2 Обоснование выбора метрик программного кода	62
4.3 Тестирование программных средств для управления гостевым сервисом в ресторане	62
4.3.1 Функциональное тестирование	62
4.3.2 Нефункциональное тестирование	71
4.4 Оценка качества программного кода	74
4.4.1 Метрика Холстеда.....	74
4.4.2 Метрика Чепина	76
Заключение	79
Список использованных источников	81
Приложение А (справочное) Глоссарий	85
Приложение Б (рекомендуемое) Скрипты для создания объектов базы данных на PostgreSQL.....	87
Приложение В (обязательное) Листинг серверной части приложения.....	90
Приложение Г (обязательное) Листинг клиентской части приложения	114

Введение

Качественное управление гостевым сервисом в ресторане является одной из главных составляющих в сфере питания и обслуживания гостей. Число ресторанов растет, а в условиях высокой конкуренции важно иметь высокий уровень обслуживания гостей. Посетителям важен развитый сервис гостевого обслуживания, и значительное удобство для них представляла бы возможность удаленного выбора понравившегося столика и предварительный заказ блюд.

Эффективное управление гостевым сервисом предполагает совершенствование процессов обслуживания, повышение качества обслуживания и создание лучших условий для гостей. Достижению этих целей способствуют программные средства, которые позволяют автоматизировать бизнес-процессы ресторана и улучшить взаимодействие с клиентами.

Разработка программных средств для управления гостевым сервисом в ресторане является актуальной темой на сегодняшний день, так как такие средства помогают упростить и визуализировать процесс бронирования столиков, оформлять предзаказ блюд, выбирать подходящий для гостя зал ресторана, а также анализировать статистику посещаемости и предпочтения гостей.

Целью выпускной квалификационной работы является разработка программных средств для управления гостевым сервисом в ресторане, которые позволят гостю выбирать столик для бронирования онлайн по представленной на сайте схеме зала ресторана, а также оформить предзаказ для созданной брони.

Исходя из цели данной работы, были поставлены следующие задачи:

- проанализировать предметную область и аналоги похожего программного обеспечения, поставить задачу на разработку;
- спроектировать программные средства, а также структуры данных разрабатываемого приложения;
- разработать модули программных средств;
- проконтролировать качество программного обеспечения.

Кроме того, стоит отметить, что разработка программных средств для управления гостевым сервисом в ресторане планируется для двух ролей: пользователь и администратор. Для гостя важно добиться простого и понятного интерфейса для взаимодействия с веб-сайтом ресторана, а также предоставить современные методы обслуживания клиентов. Администратору, помимо удобного интерфейса, необходимо реализовать конструктор для создания визуальной схемы зала с возможностью ее изменения и удаления. Кроме того, он будет иметь панель управления для взаимодействия с разделами «Меню» и «Бронирования». В качестве элементов статистики за основу будут рассматриваться такие данные как популярные блюда, средний чек, выручка, загруженность ресторана, распределение гостей по столам.

Успешное решение поставленных задач позволит создать эффективный инструмент для управления гостевым сервисом в ресторане.

1 Анализ программных средств для управления гостевым сервисом в ресторане

1.1 Анализ предметной области управления гостевым сервисом в ресторане

Программные средства для управления гостевым сервисом в ресторане – это инструменты, которые позволяют автоматизировать и упростить процесс взаимодействия ресторана с клиентами [1]. Такие системы дают возможность самостоятельно забронировать подходящий для клиента столик, выбрать заранее желаемые блюда, просматривать меню и информацию о заведении .

Деятельность ресторана охватывает различные направления обслуживания, поэтому удобнее всего рассматривать ее в контексте разных функциональных областей. Например, можно выделить такую область как управление бронированием, анализ статистики посещений и заказов и меню с возможностью его просмотра и оформления предзаказа. В рамках данной работы будет использоваться именно такое структурное разделение процессов управления гостевым сервисом ресторана.

Главной особенностью разрабатываемой системы является создание схемы зала со стороны администратора, ориентируясь на которую пользователь сможет выбрать желаемый на основе своих предпочтений столик. Гость после бронирования стола будет иметь возможность сформировать предзаказ из блюд, представленных на веб-сайте ресторана. Реализация данных функций позволит улучшить гостевой сервис обслуживания клиентов ресторана.

Программные средства для управления гостевым сервисом ресторана разрабатываются для различных категорий пользователей, таких как офисные работники, студенты, обычные клиенты, которые желают получить удобный, понятный и быстрый доступ к услугам, предоставляемые рестораном [2].

Для полного доступа ко всем возможностям ресторана обязательным условием является регистрация пользователя в системе для сохранения и систематизации информации о его бронях и предзаказах.

1.2 Обзор и анализ программных средств для управления гостевым сервисом в ресторане

1.2.1 Веб-сайт ресторана «Tom Soy»

Ресторана «Tom Soy» является типичным примером сайта ресторана. Интерфейс приложения приведен на рисунке 1.

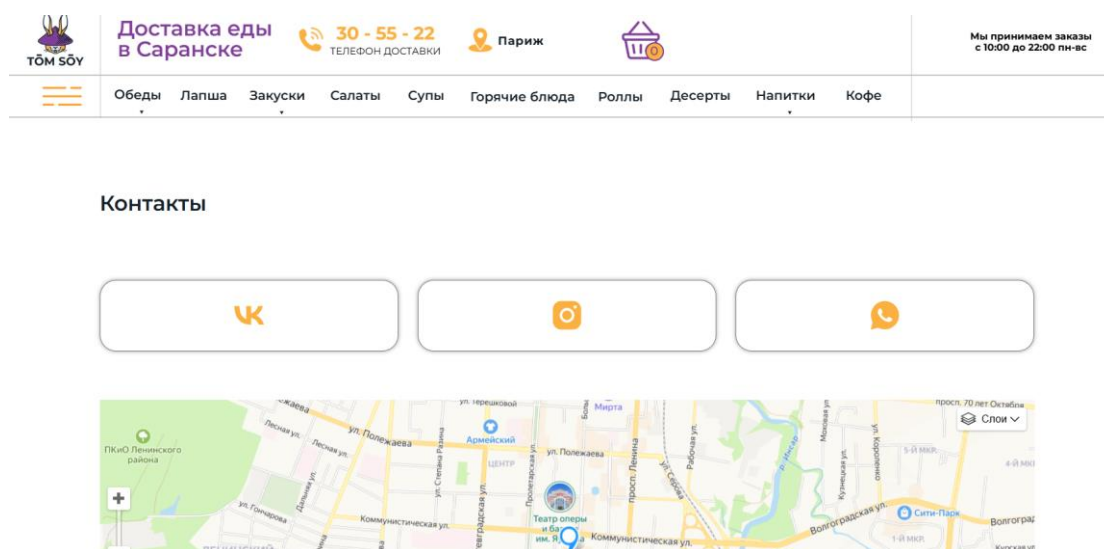


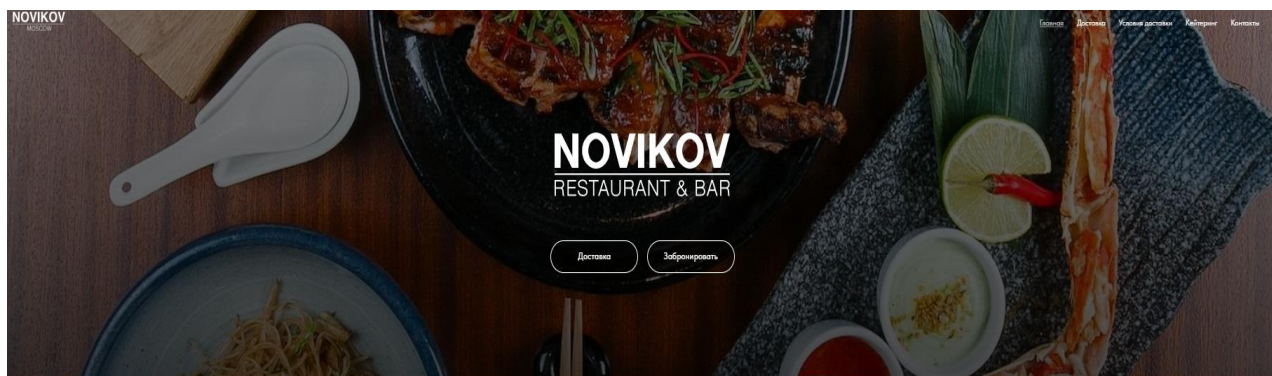
Рисунок 1 – Интерфейс сайта ресторана «Tom Soy»

Среди достоинств данного приложения можно выделить то, что оно предоставляет пользователю простой и интуитивно понятный интерфейс, возможность просмотра меню, заказ блюд на доставку, ссылки на различные месенджеры для связи с рестораном.

Главным недостатком, с точки зрения пользователя, можно считать невозможность бронирования стола на желаемую дату и время, отсутствие предзаказов из блюд, представленных в меню, низкое развитие сервиса бронирования.

1.2.2 Веб-сайт ресторана «NOVIKOV»

Ресторан «NOVIKOV» – это популярный ресторан Москвы, который имеет яркое оформление веб-сайта и огромный список различных блюд, что очень нравится многим пользователям. Интерфейс приложения представлен на рисунке 2.



Интерьер



Рисунок 2 – Интерфейс сайта ресторана «NOVIKOV»

Среди достоинств этого сайта можно выделить то, что оно предоставляет пользователю современный и приятный интерфейс, а также огромный выбор блюд. Одной из главных особенностей является возможность просмотра интерьера заведения с помощью фотогалереи. Пользователь может изучить обстановку и атмосферу ресторана, пролистывая фотографии, и на их основе выбрать желаемый столик в зале, ориентируясь на личные предпочтения.

Недостатком этого приложения является отсутствие состава блюд, их веса и отсутствие предзаказа в ресторане. Также сайт ресторана «NOVIKOV» не предоставляет гибкой системы бронирования столика.

1.2.3 Веб-сайт ресторана «IL Patio»

Хорошим примером для подбора ресторана на основании предпочтений клиента является сайт ресторана «IL Patio». Пользователь указывает необходимые для него требования и на интерактивной карте может выбрать один из возможных вариантов ресторана. Интерфейс веб-сайта представлен на рисунке 3.

[← Главная · Рестораны](#)

Рестораны iL Патио в Москве

Введите название улицы, метро или города

- ☐ Wi Fi
 ☐ Парковка
 ☐ TV
 ☐ Детская комната
 ☐ Ланч
 ☐ Заказ столика
 ☐ Летняя веранда
 ☐ Работает 24 часа
☐ Банкетный зал

iL Патио

ТЦ Мега Белая Дача

Котельники, 1-й Покровский проезд, 1

iL Патио

Москва, 1-я Тверская-Ямская улица, 2с1

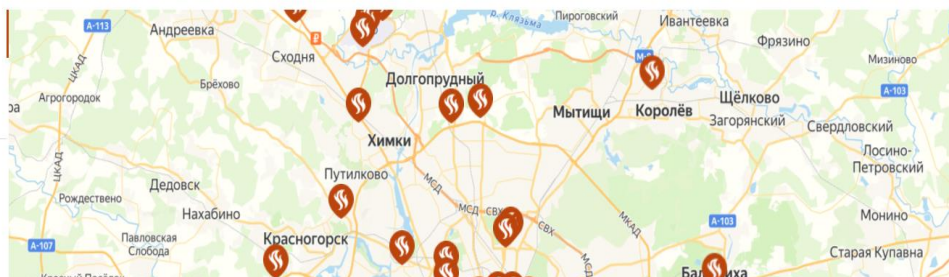
 Маяковская, Пушкинская, Тверская


Рисунок 3 – Интерфейс сайта ресторана «iL Patio»

Главным достоинством данного сайта сети ресторанов «iL Patio» является возможность выбора ресторана на основе предпочтений пользователя. Клиент выбирает из представленного набора услуг те, что ему необходимы, и сайт отображает рестораны с выбранным клиентом набором услуг. После выбора ресторана пользователю отображается режим работы, номер телефона для бронирования, а также кнопка для перехода в меню. Меню ресторана в формате pdf выглядит так, как оно и будет представлено в заведении, что позволяет постоянно отображать клиенту актуальную информацию о блюдах и ценах.

Основными недостатками является то, что нет возможности выбрать конкретный зал ресторана прямо на сайте. Это возможно лишь позвонив по номеру телефона для бронирования, что является не совсем удобным способом.

1.2.4 Веб-сайт ресторана «Pitnica»

Приложение ресторана «Pitnica» — это сеть ресторанов с веб-сайтом, который предоставляет пользователю удобный способ бронирования столика с возможностью указания предпочтений. Интерфейс сайта ресторана «Pitnica» представлен на рисунке 4.



ФРАНШИЗА

МЕНЮ РЕСТОРАНА

МЕНЮ ДОСТАВКИ

ЗАБРОНИРОВАТЬ СТОЛ

АКЦИИ

ПРОГРАММА ЛОЯЛЬНОСТИ

ОСТАВИТЬ ОТЗЫВ

КОНТАКТЫ

Город: Пенза

ул. Московская, 65а

+7 (8412) 30-21-01



РЕЗЕРВ СТОЛА

ЗАБРОНИРОВАТЬ

Имя *

Телефон *

E-mail *

Кол-во человек *

1

Дата *

Время *

06.05.2025

--:--

Комментарий

☒ Согласен на обработку персональных данных

Рисунок 4 – Интерфейс приложения «Pitnica»

Ресторан «Pitnica» позволяет пользователям забронировать столик онлайн, заполнив небольшую анкету. Клиент указывает свое имя, телефон, адрес почты, число гостей, дату и время бронирования, а также, если нужно, свои предпочтения указывает в комментарии к брони. После создания брони пользователь может ее просмотреть в личном кабинете и изменить данные брони, если это необходимо. Это очень удобно, например, когда нет возможности связаться с рестораном по телефону.

К недостаткам можно отнести отсутствие возможности выбора конкретного столика по схеме зала, а также выбора блюд из меню к созданной брони. Это ограничивает гибкость сервиса и уменьшает удобство для гостей, которым важна визуализация зала ресторана и возможность заранее составить заказ. Таким образом, данный аналог обладает различными возможностями, но не имеет реализацию всех необходимых функций.

1.2.5 Сравнение программных средств для управления гостевым сервисом в ресторане

Проведем сравнительный анализ рассмотренных выше программных средств управления гостевым сервисом ресторана, таких как «Tom Soy», «NOVIKOV», «IL Patio», «Pitnica», по критериям, напрямую влияющим на функциональность с точки зрения гостя ресторана [3].

Для оценки приложений будет использоваться трехбалльная шкала соответствия. В таблице 1 представлены характеристики оценки.

Таблица 1 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) полностью отсутствует
1	существует частичная реализация функции (качества)
2	функция (качество) полностью реализована

Результаты сравнения программных средств для управления гостевым сервисом в ресторане представлены в таблице 2.

Таблица 2 – Сравнение программных средств аналогичного назначения

Критерий \ Приложение	«Tom Soy»	«NOVIKOV»	«IL Patio»	«Pitnica»
1. Возможность выбора столика	0	0	0	0
2. Возможность онлайн бронирования столика	0	0	1	2
3. Просмотр меню на сайте	2	1	2	2
4. Возможность оформления предзаказа блюд к броне	0	0	0	0
5. Возможность изменения данных брони	0	0	0	0
6. Наличие схемы зала	0	1	0	0
7. Наличие контактов на сайте	2	2	2	2
8. Наличие современного и удобного интерфейса	1	2	2	2
Итого	5	6	7	8

Подводя итог обзора программных средств для управления гостевым сервисом в ресторане можно сделать вывод, что ни одно из рассмотренных решений не соответствует выделенным критериям в полном объеме. Среди всех продуктов-аналогов приложение ресторана «Pitnica» выделяется большим количеством реализованных функций, однако данное приложение не обеспечивает пользователям возможность выбора конкретного столика и зала, а также отсутствие предзаказа блюд из меню, что делает гостевой сервис обслуживания гостей далеким от современного и удобного.

В результате данного обзора было принято решение разработать собственное программное средство. В новое программное средство для управления гостевым сервисом будут включены функции выбора конкретного столика и зала, предзаказа блюд, а также визуализация схемы ресторана для удобства выбора стола на основе предпочтений пользователя. Реализация этих функций позволит добиться современного и качественного сервиса обслуживания гостей.

1.3 Постановка задачи на разработку программных средств

Основной задачей на разработку программных средств управления гостевым сервисом в ресторане является создание веб-сайта, который позволит пользователю выбирать столик онлайн, а также оформлять предказ [4].

Для реализации приложения необходимо разработать веб-сайт для двух ролей: администратор и пользователь. Администратор сможет создавать схему зала ресторана, используя конструктор для создания плана ресторана. Также он сможет управлять разделом «Меню» и «Бронирование». Пользователю же будет доступна возможность просмотра схемы зала, выбора столика, его бронирование, а также оформление предзаказа после резервации места. Для хранения и использования набора данных необходимо использовать базу данных. Пользователь и администратор могут изменять детали брони или отменять их. Дополнительно у второй роли имеется возможность смены статуса резервации и подтверждение оплаты заказа. Администратору будет доступен

просмотр статистики: средний чек, популярные блюда, выручка ресторана, брони гостей, загруженность ресторана по дням. В данном разделе необходимо для наглядности изобразить актуальные данные с помощью диаграмм.

Для эффективного взаимодействия пользователя и администратора с графическими элементами управления и навигации сайта, необходимо распределить элементы графического интерфейса по групповым признакам. Для клиентов можно сформировать следующие разделы программы:

- «Главная страница»;
- «Меню ресторана»;
- «Забронировать стол»;
- «Мои заказы».

У администратора будут те же разделы, но добавиться панель управления, которая содержит в себе следующее:

- «План зала»;
- «Меню»;
- «Бронирования».

Таким образом, на основе сформированных требований будут произведены проектирование и разработка программных средств для управления гостевым сервисом в ресторане.

1.4 Анализ функциональных требований

В процессе анализа задачи на разработку программных средств для управления гостевым сервисом в ресторане были выделены следующие функциональные требования:

- регистрация и авторизация пользователя;
- создание, сохранение, очистка и просмотр схемы зала;
- добавление и удаление категории блюда;
- добавление, удаление, редактирование блюда;
- просмотр меню;
- просмотр статистики бронирования за неделю, месяц и год;

- возможность выбора одного из трех типов зала (столик в обычном зале, бронь vip-зала, бронь банкетного зала);
- возможность выбора столика на определенную дату и время;
- создание брони;
- возможность оформить предзаказ к брони;
- просмотр истории заказов;
- редактирование деталей бронирования и предзаказа;
- просмотр главной страницы сайта ресторана.

В соответствии с выявленными требованиями была разработана диаграмма вариантов использования UML, представленная на рисунке 5 [5].

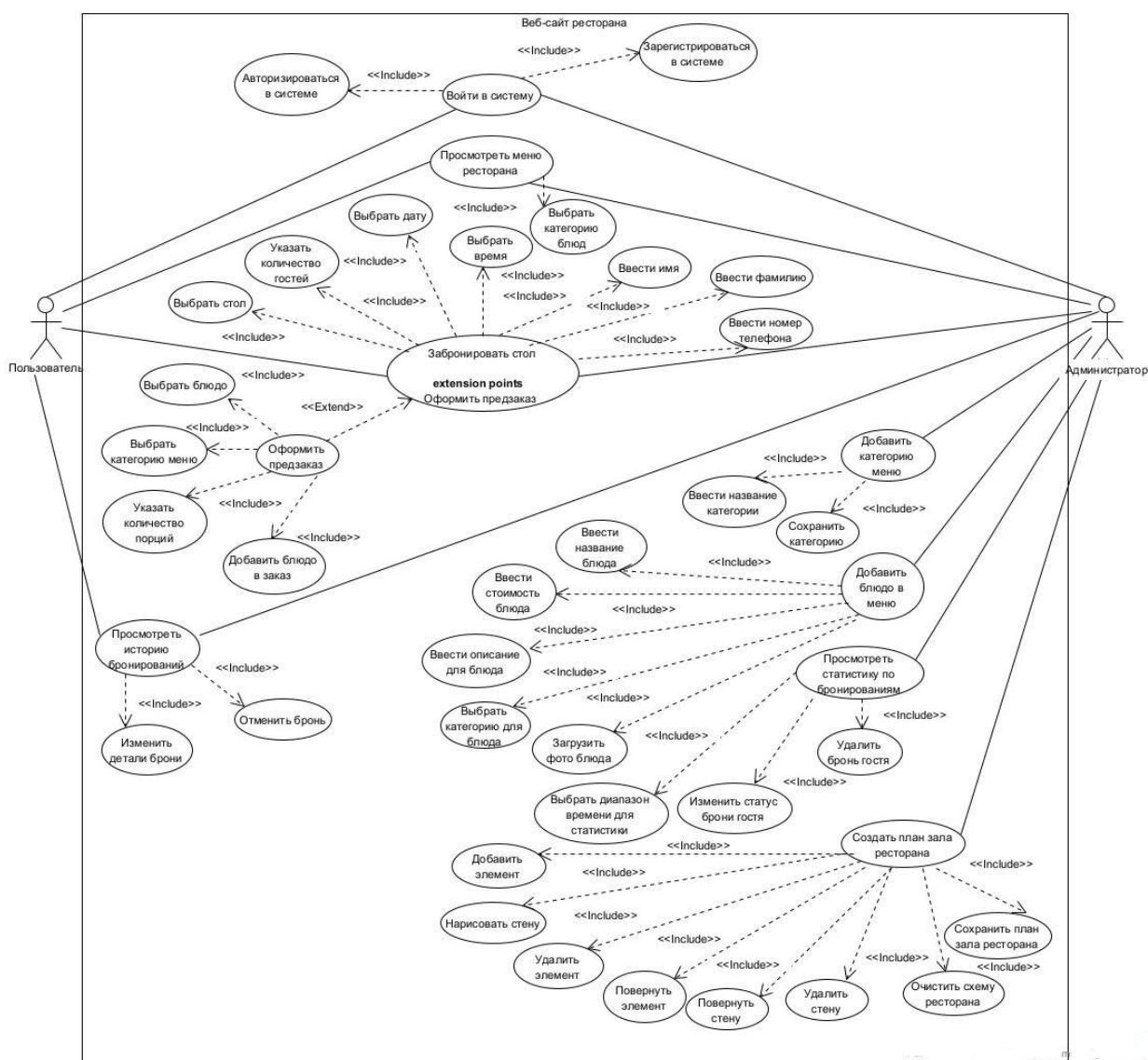


Рисунок 5 – Диаграмма вариантов использования

Ниже приведены спецификации наиболее важных бизнес-процессов.

Сценарий варианта использования «Войти в систему» представлен на рисунке 6.

Препедент: Войти в систему
ID: 1
Краткое описание: Пользователь регистрируется или авторизируется в приложении
Главные актеры: Пользователь
Второстепенные актеры: Администратор
Предусловие: 1. Пользователь открыл приложение; 2. Пользователь не был авторизован или зарегистрирован.
Основной поток: 1. Пользователь переходит на страницу входа в приложение 2. Пользователь вводит логин и пароль для входа в аккаунт 3. Пользователь получает полный доступ к функционалу приложения Альтернативные потоки: 2.1 Пользователь вводит данные для регистрации (электронную почту, пароль, имя, фамилию, номер телефона)
Постусловия: 1. Пользователь вошел в приложение 2. Пользователю доступен функционал приложения

Рисунок 6 – Сценарий варианта использования «Войти в систему»

Сценарий варианта использования «Забронировать стол» на рисунке 7.

Препедент: Забронировать стол
ID: 2
Краткое описание: Пользователь бронирует стол
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: 1. Пользователь открыл приложение; 2. Пользователь был авторизован.
Основной поток: 1. Пользователь переходит на страницу бронирования стола 2. Пользователь выбирает свободный столик 3. Пользователь указывает данные брони (количество гостей, дату, время, имя, фамилию, номер телефона) 4. Пользователь бронирует стол 5. Пользователю предлагается оформить предзаказ к брони 6. Пользователь оформляет или отказывается от предзаказа 7. Пользователь успешно создал бронь в системе
Постусловия: 1. В системе зарегистрирована новая бронь 2. Пользователь просматривает бронь в разделе «Мои заказы»

Рисунок 7 – Сценарий варианта использования «Забронировать стол»

Сценарий варианта использования «Добавить блюдо в меню» представлен на рисунке 8.

Прецедент: Добавить блюдо в меню
ID: 3
Краткое описание: Администратор добавляет новое блюдо в меню
Главные актеры: Пользователь
Второстепенные актеры: Администратор
Предусловие: 1. Пользователь открыл приложение; 2. Пользователь не был авторизован или зарегистрирован.
Основной поток: 1. Пользователь переходит на страницу входа в приложение 2. Пользователь вводит логин и пароль для входа в аккаунт 3. Пользователь получает полный доступ к функционалу приложения Альтернативные потоки: 2.1 Пользователь вводит данные для регистрации (электронную почту, пароль, имя, фамилию, номер телефона)
Постусловия: 1. Пользователь вошел в приложение 2. Пользователю доступен функционал приложения

Рисунок 8 – Сценарий варианта использования «Добавить блюдо в меню»

В результате анализа функциональных требований были выделены варианты использования программы. Для некоторых вариантов использования были представлены их спецификации, позволяющие отобразить последовательность действий пользователя, администратора и системы.

1.5 Анализ нефункциональных требований

В программном средстве для управления гостевым сервисом в ресторане предусматривается разработка базы данных «Ресторан». Разработка базы данных будет проводиться с использованием PostgreSQL на языке SQL [6]. База данных разворачивается в Docker контейнере.

Клиентскую часть приложения необходимо разработать с использованием фреймворка Next.js, который основан на библиотеке React. Разработка будет вестись в интегрированной среде Microsoft Visual Studio Code.

Серверная часть приложения реализуется на языке Python с применением архитектурного паттерна MVC [7].

Для развертывания приложения в продуктивной среде будет использоваться Docker [8]. Каждый компонент приложения (серверная, клиентская часть, база данных) размещен в своем Docker контейнере.

Использование PostgreSQL в качестве базы данных позволит добиться высокой производительности и надежность. Next.js в сочетании с библиотекой React позволит разработать удобный и современный интерфейс для клиентского приложения. Взаимодействие серверной и клиентской части приложения осуществляется через REST API [9]. Использование MVC позволит упростить разработку и сопровождение приложения.

Построение диаграмм осуществляется с помощью языка графического описания UML 2.0 через онлайн-сервис draw.io и программного средства Visual Paradigm 17.2. Оба варианта позволяют создавать различные диаграммы и схемы: диаграммы классов, вариантов использования, деятельности и т. д. [10].

Разрабатываемое веб-приложение должно быть оптимизировано для работы в современных браузерах, таких как Google Chrome (версии 80 и выше), Opera GX (версии 100 и выше), Microsoft Edge (версии 80 и выше).

1.6 Планирование разработки и оценка бюджета

Для оценки временных затрат и наглядного планирования процесса разработки была построена диаграмма Ганта с использованием инструментов Microsoft Office Project Server 2007 [11]. Процесс разработки проекта содержит 4 вехи, каждая из которых включает в себя некоторый набор подзадач.

Для разработки данного приложения можно выделить следующие этапы:

- анализ предметной области и постановка задачи на разработку программных средств для управления гостевым сервисом в ресторане;
- проектирование программных средств для управления гостевым сервисом в ресторане;

- реализация программных средств для управления гостевым сервисом в ресторане;
- контроль качества программного обеспечения;
- оформление отчёта.

Краткое описание задач каждого этапа приведено на рисунке 9.

Название задачи	Длительность	Начало	Окончание
Начало проекта	1 день	Вт 08.04.25	Вт 08.04.25
☒ Анализ предметной области и постановка задачи на разработку программных сре	8 дней	Ср 09.04.25	Пт 18.04.25
Анализ предметной области управления гостевым сервисом в ресторане	2 дней	Ср 09.04.25	Чт 10.04.25
Обзор и анализ программных средств для управления гостевым сервисом в ресторане	1 день	Пт 11.04.25	Пт 11.04.25
Сравнение программных средств для управления гостевым сервисом в ресторане	1 день	Пт 11.04.25	Пт 11.04.25
Постановка задачи на разработку программных средств	1 день	Пн 14.04.25	Пн 14.04.25
Анализ функциональных требований	1 день	Вт 15.04.25	Вт 15.04.25
Анализ нефункциональных требований	1 день	Вт 15.04.25	Вт 15.04.25
Планирование разработки и оценка бюджета	2 дней	Чт 17.04.25	Пт 18.04.25
☒ Проектирование программных средств для управления гостевым сервисом в рест	7 дней	Пн 21.04.25	Вт 29.04.25
Архитектура приложения	1 день	Пн 21.04.25	Пн 21.04.25
Проектирование структур данных	2 дней	Вт 22.04.25	Ср 23.04.25
Разработка концептуальной модели базы данных	1 день	Чт 24.04.25	Чт 24.04.25
Разработка логической модели базы данных	2 дней	Пт 25.04.25	Пн 28.04.25
Проектирование пользовательского интерфейса	1 день	Вт 29.04.25	Вт 29.04.25
☒ Реализация программных средств для управления гостевым сервисом в ресторан	17 дней	Ср 30.04.25	Чт 22.05.25
Реализация базы данных	4 дней	Ср 30.04.25	Пн 05.05.25
Реализация серверной части приложения	5 дней	Вт 06.05.25	Пн 12.05.25
Реализация клиентской части приложения	2 дней	Чт 15.05.25	Пт 16.05.25
Реализация интерфейса приложения	2 дней	Ср 21.05.25	Чт 22.05.25
☒ Контроль качества программного обеспечения	5 дней	Пт 23.05.25	Чт 29.05.25
Обоснование выбора режима тестирования и метрик кода	2 дней	Пт 23.05.25	Пн 26.05.25
Функциональное тестирование	2 дней	Вт 27.05.25	Ср 28.05.25
Нефункциональное тестирование	2 дней	Вт 27.05.25	Ср 28.05.25
Оценка качества программного кода	1 день	Чт 29.05.25	Чт 29.05.25
☒ Оформление отчёта	5 дней	Пт 30.05.25	Чт 05.06.25
Оформление пояснительной записки	5 дней	Пт 30.05.25	Чт 05.06.25
Конец проекта	1 день	Ср 04.06.25	Ср 04.06.25

Рисунок 9 – Сведения об этапах разработки приложения

Также была построена диаграмма Ганта для наглядного представления плана разработки проекта. Данная диаграмма приведена на рисунке 10.

Название ресурса	Тип	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на использ.
Аналитик	Трудовой	100%	200,00 Р/ч	250,00 Р/ч	0,00 Р
Программист	Трудовой	100%	300,00 Р/ч	350,00 Р/ч	0,00 Р
Дизайнер	Трудовой	100%	150,00 Р/ч	250,00 Р/ч	0,00 Р
Тестировщик	Трудовой	100%	250,00 Р/ч	350,00 Р/ч	0,00 Р
Интернет	Затраты				
Компьютер	Материальный		70 000,00 Р		0,00 Р

Рисунок 11 – Стоимость задействованных ресурсов

С помощью средств Microsoft Office Project Server 2007 был автоматически сформирован отчет о затратах для разрабатываемого проекта. Общая оценка финансовых затрат проекта представлена на рисунке 12.

Диаграмма 1 and 2.png

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
18	Реализация серверной части прилож	0,00 Р	Пропорциональное	12 000,00 Р	0,00 Р	12 000,00 Р
17	Реализация базы данных	0,00 Р	Пропорциональное	11 100,00 Р	0,00 Р	11 100,00 Р
27	Оформление пояснительной записки	0,00 Р	Пропорциональное	8 000,00 Р	0,00 Р	8 000,00 Р
19	Реализация клиентской части прилож	0,00 Р	Пропорциональное	7 200,00 Р	0,00 Р	7 200,00 Р
20	Реализация интерфейса приложения	0,00 Р	Пропорциональное	7 200,00 Р	0,00 Р	7 200,00 Р
12	Проектирование структур данных	0,00 Р	Пропорциональное	4 800,00 Р	0,00 Р	4 800,00 Р
22	Обоснование выбора режима тестир	0,00 Р	Пропорциональное	4 000,00 Р	0,00 Р	4 000,00 Р
23	Функциональное тестирование	0,00 Р	Пропорциональное	4 000,00 Р	0,00 Р	4 000,00 Р
24	Нефункциональное тестирование	0,00 Р	Пропорциональное	4 000,00 Р	0,00 Р	4 000,00 Р
3	Анализ предметной области управл	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
9	Планирование разработки и оценка	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
14	Разработка логической модели базы	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
15	Проектирование пользовательского	0,00 Р	Пропорциональное	3 000,00 Р	0,00 Р	3 000,00 Р
11	Архитектура приложения	0,00 Р	Пропорциональное	2 400,00 Р	0,00 Р	2 400,00 Р
25	Оценка качества программного кода	0,00 Р	Пропорциональное	2 000,00 Р	0,00 Р	2 000,00 Р
4	Обзор и анализ программных средств	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
5	Сравнение программных средств дл	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
6	Постановка задачи на разработку пр	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
7	Анализ функциональных требований	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
8	Анализ нефункциональных требован	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
13	Разработка концептуальной модели	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
28	Конец проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
29	Итого:	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		88 900,00 Р	0,00 Р	88 900,00 Р

Рисунок 12 – Общая оценка финансовых затрат

Таким образом, исходя из оценки финансовых и временных затрат, видно, что предполагаемые затраты на разработку составляют 88900 рублей, а общая продолжительность 42 дня.

2 Проектирование программных средств для управления гостевым сервисом в ресторане

2.1 Архитектура приложения

Архитектура программных средств для управления гостевым сервисом в ресторане представляет собой монолитную архитектуру в сочетании с шаблоном проектирования MVC. Модель и контроллеры реализованы на стороне сервера. Они отвечают за обработку и хранение данных, а также обрабатывают действия пользователя и осуществляют взаимодействие с базой данных. Клиентское приложение разработано с помощью фреймворка Next.js и библиотеки React и служит основным интерфейсом взаимодействия пользователя с системой. Все команды, которые инициирует пользователь, передаются на сервер с помощью HTTP-запросов через REST API.

На рисунке 13 изображена диаграмма развертывания, которая отображает архитектуру программных средств для управления гостевым сервисом в ресторане.

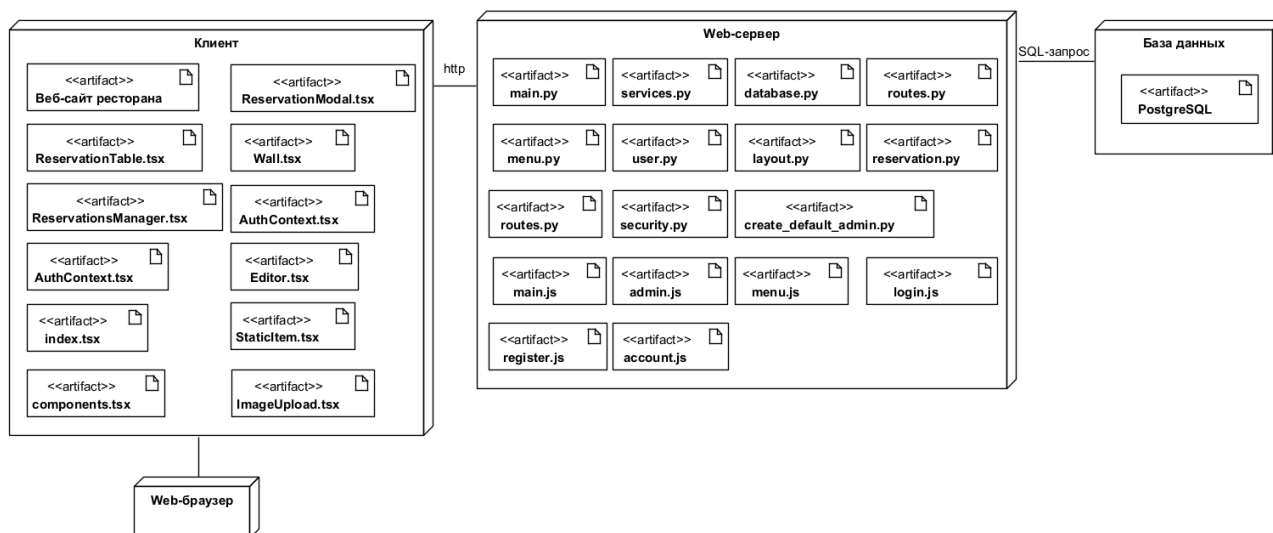


Рисунок 13 – Архитектура программных средств для управления гостевым сервисом в ресторане

Серверная часть приложения реализована в виде разделенного монолита, в рамках которого клиентская и серверная часть приложения разделены, но

функционируют в системе как единое приложение. Серверная часть приложения, разработанная на языке Python, обрабатывает бизнес-логику и взаимодействует с базой данных, а клиентская часть, реализованная с использованием Next.js и библиотеки React, обеспечивает взаимодействие системы с пользователем.

Взаимодействие между контроллерами и моделями осуществляется в соответствии с шаблоном MVC. Контроллеры принимают запросы от клиента, обрабатывают их через модели, которые обращаются к базе данных, и формируют результаты в формате JSON.

Клиент отправляет запросы через REST API и получает от сервера ответ на свой запрос. Такая архитектура обеспечивает надежность, масштабируемость и поддержку проекта.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере [12].

Для разработки концептуальной модели данных необходимо выделить список всех основных сущностей предметной области разрабатываемого приложения. Описание сущностей предметной области представлено в таблице 3.

Таблица 3 – Сущности БД

Название сущности	Идентификатор сущности	Назначение сущности
Пользователь	Users	Содержит сведения о пользователях системы (гости, администраторы)
Категория меню	Categories	Содержит сведения о категориях меню
Блюдо	Menu_items	Содержит сведения о блюдах в меню ресторана
Предзаказ	Order_items	Содержит информацию о предзаказе клиента

Продолжение таблицы 3

Название сущности	Идентификатор сущности	Назначение сущности
Бронь	Reservations	Содержит сведения о брони пользователей
Стол	Tables	Содержит сведения о расположении столов на схеме ресторана
Тип стола	Table_types	Содержит варианты столов (квадратный, круглый и т. д.)
Статичный элемент	Static_items	Содержит сведения о расположении статичных элементов схемы (бар, туалет, кухня и т. д.)
Стена	Walls	Содержит информацию о расположении стены
Комната	Rooms	Содержит схему зала ресторана

Графическое представление концептуальной модели представлено на рисунке 14.

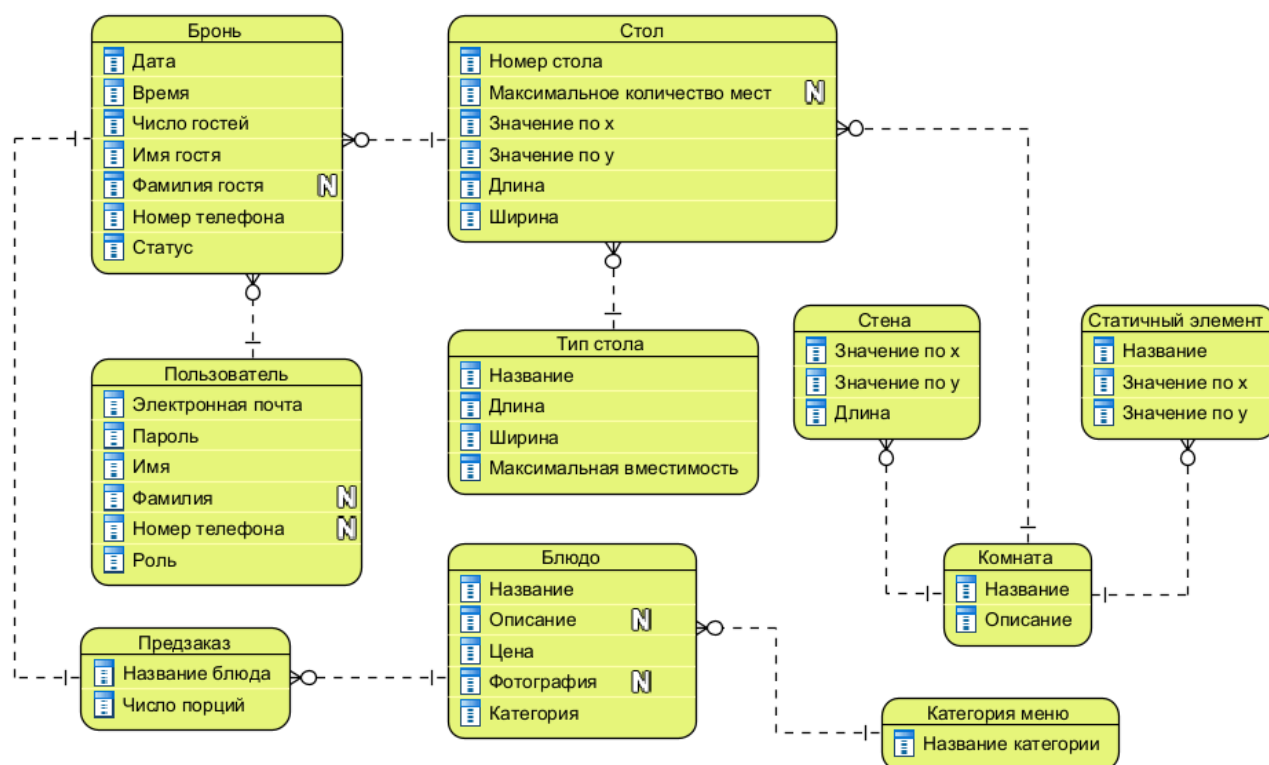


Рисунок 14 – Концептуальная модель базы данных предметной области
«Программные средства управления гостевым сервисом ресторана»

Данная модель построена в среде Visual Paradigm 17.2 в нотации IDEF1X.

По концептуальной модели базы данных ресторана необходимо произвести описание спецификаций атрибутов для каждой сущности.

В таблице 4 описываются атрибуты сущности «Категория меню». Данная сущность предназначена для хранения категорий меню.

Таблица 4 – Атрибуты сущности «Категория меню»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название категории	Название категории блюда	Строковый	50	«А – я»	Нет	Салаты

В таблице 5 описываются атрибуты сущности «Блюдо». Данная сущность предназначена для хранения блюд в меню, которые добавляет администратор.

Таблица 5 – Атрибуты сущности «Блюдо»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название	Название блюда	Строковый	50	«А – я»	Нет	Цезарь
2	Описание	Информация о блюде	Строковый	255	«А – я»	Да	Томаты, листья салата
3	Цена	Стоимость блюда	Числовой	-	≥ 0	Нет	550
4	Фотография	Изображение блюда	Двоичный объект	-		Да	
5	Категория	Категория блюда	Строковый	50	«А – я»	Нет	Салаты

В таблице 6 описываются атрибуты сущности «Предзаказ». Данная сущность предназначена для хранения предзаказа пользователя. Предзаказ не может существовать без брони.

Таблица 6 – Атрибуты сущности «Предзаказ»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название блюда	Название блюда	Строковый	50	«А – я»	Нет	Нисуаз
2	Число порций	Количество порций блюда	Целый		> 0	Нет	2

В таблице 7 описываются атрибуты сущности «Бронь». Данная сущность предназначена для хранения брони гостей. Их можно редактировать или отменять как у пользователя, так и у администратора.

Таблица 7 – Атрибуты сущности «Бронь»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Дата	Дата брони	Дата	-		Нет	15.05.2025
2	Время	Время брони	Время	-		Нет	15:00
3	Число гостей	Кол-во гостей	Целый	-	>=1	Нет	2
4	Имя гостя	Имя забронировавшего	Строковый	50	«А – я»	Нет	Дмитрий
5	Фамилия гостя	Фамилия забронировавшего	Строковый	50	«А – я»	Нет	Федоров
6	Номер телефона	Номер телефона	Строковый	12		Нет	+79278561213

В таблице 8 описываются атрибуты сущности «Пользователь». Данная сущность предназначена для хранения данных о пользователях веб-сайта. Администратор в системе один и его данные уже находятся в базе данных по умолчанию.

Таблица 8 – Атрибуты сущности «Пользователь»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Электронная почта	Email пользователя (логин)	Строковый	100		Нет	admin@example.com
2	Пароль	Пароль от аккаунта	Строковый	100	{0 – 9}, «А – z»	Нет	Password789
3	Имя	Имя пользователя	Строковый	50	«А – я»	Нет	Светлана
4	Фамилия	Фамилия пользователя	Строковый	50	«А – я»	Нет	Петрова
5	Номер телефона	Номер телефона	Строковый	12		Нет	+78564231256
6	Роль	Роль в системе	Строковый	30	Админ, пользователь	Нет	Админ

В таблице 9 описываются атрибуты сущности «Стол». Данная сущность предназначена для хранения информации о параметрах стола. По умолчанию цвет стола зеленый, но, когда его бронируют, он становится красным. У администратора имеется возможность удалить или повернуть его. Каждый из столов относится к определённому типу и имеет своё визуальное отображение на схеме зала.

Таблица 9 – Атрибуты сущности «Стол»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Номер стола	Номер стола	Целый	-		Нет	10
2	Максимальное количество мест	Максимальное число посадочных мест	Целый		>0	Нет	2
3	Значение по X	Координата X	Числовой			Нет	200
4	Значение по Y	Координата Y	Числовой			Нет	400
5	Длина	Длина	Числовой			Нет	350
6	Ширина	Ширина	Числовой			Нет	150

В таблице 10 описываются атрибуты сущности «Стена». Данная сущность предназначена для хранения информации о стенах, которые администратор изображает на схеме зала ресторана.

Таблица 10 – Атрибуты сущности «Стена»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Значение по X	Координата X	Числовой			Нет	200
2	Значение по Y	Координата Y	Числовой			Нет	600
3	Длина	Длина стены	Числовой		>0	Нет	50

В таблице 11 описываются атрибуты сущности «Статичный элемент». Данная сущность предназначена для хранения объектов схемы зала (кухня, туалет, банкетный зал и т. д.).

Таблица 11 – Атрибуты сущности «Статичный элемент»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название	Название элемента	Строковый	100	«А – я»	Нет	Кухня
2	Значение по X	Координата X	Числовой			Нет	350
3	Значение по Y	Координата Y	Числовой			Нет	150

В таблице 12 описываются атрибуты сущности «Комната». Данная сущность предназначена для хранения плана ресторана. Схема зала строится из сущностей «Статичный элемент», «Стол», «Стена».

Таблица 12 – Атрибуты сущности «Комната»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название	Содержит название комнаты	Строковый	50		Нет	Схема зала
2	Описание	Содержание комнаты	Строковый	1000		Нет	Кухня, бар, vip-зал

В таблице 13 описываются атрибуты сущности «Тип стола». Данная сущность предназначена для хранения информации о типе столика. Стол может быть круглый, прямоугольный, с диванами. Каждый из типов имеет свою максимальную вместимость.

Таблица 13 – Атрибуты сущности «Тип стола»

№	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название	Название стола	Строковый	100	«А – я»	Нет	Круглый стол
2	Длина	Длина	Числовой			Нет	350
3	Ширина	Ширина	Числовой			Нет	150
4	Максимальная вместимость	Максимальное число мест	Числовой			Нет	2

Описание связей между сущностями представлено в таблице 14.

Таблица 14 – Спецификации связей между сущностями

№	Тип сущности	Тип сущности	Семантика связи	Мощность	Обязательность
1	Стол	Бронь	Каждый бронь относится к одному столу/ У стола может быть несколько броней на разное время	1:M	Необязательная, т. к. не каждый стол может быть забронирован
2	Тип стола	Стол	Каждый стол относится к определенному типу/ Множество столов может относиться к одному типу	1:M	Обязательная
3	Предзаказ	Бронь	Один предзаказ относится к одной брони и наоборот	1:1	Необязательная, т. к. бронь может существовать без предзаказа

Продолжение таблицы 14

№	Тип сущности	Тип сущности	Семантика связи	Мощность	Обязательность
4	Блюдо	Предзаказ	Предзаказ содержит блюдо/Множество блюд могут быть в предзаказе	1:M	Обязательная
5	Пользователь	Бронь	Пользователь может создать множество броней/ Бронь относится к одному пользователю	1:M	Обязательная
6	Комната	Стол	Комната содержит столы/ множество столов принадлежат комнате	1:M	Обязательная
7	Комната	Статичный элемент	Комната содержит статичные элементы/ Статичные элементы принадлежат комнате	1:M	Обязательная
8	Комната	Стена	Комната содержит стены/ Стены принадлежат комнате	1:M	Обязательная
9	Категория меню	Блюдо	К одной категории относится несколько блюд/ Блюдо относится к определенной категории	1:M	Обязательная

Полученные результаты являются исходными данными для моделирования базы данных и разработки приложения.

2.2.2 Логическая модель базы данных

Логическая модель данных – это расширение концептуальной модели, в которой сущности, атрибуты и связи описываются более подробно. Логическая модель данных служит фундаментом для физической модели, предоставляя

более детальное представление данных [13]. Данная модель представлена на рисунке 15.

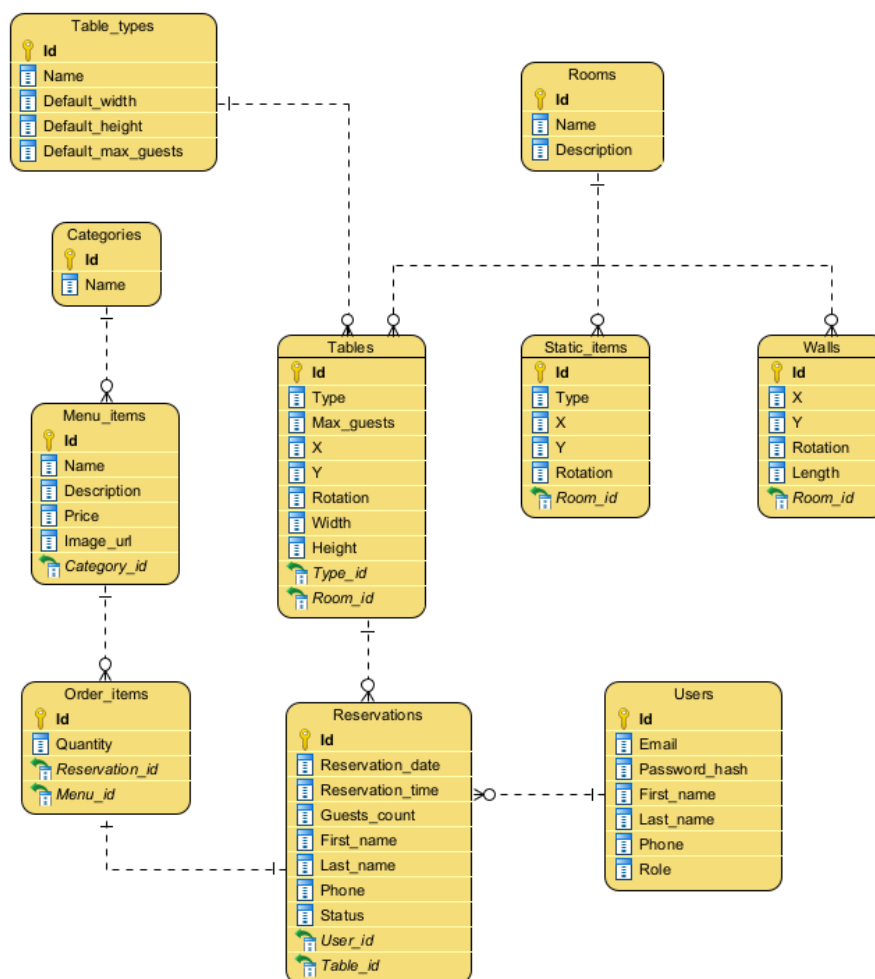


Рисунок 15 – Графическое представление логической модели

Таким образом, спроектированные графические представления организации и хранения данных позволяют выполнить техническую реализацию обращения к базе данных приложения, а также наглядно отразить структурные связи между данными.

2.3 Проектирование пользовательского интерфейса

Интерфейс веб-приложения должен быть интуитивно понятным и удобным для пользователей, а также соответствовать всем функциональным требованиям.

Проектируемый интерфейс включает в себя главную страницу с меню для перехода между вкладками веб-сайта ресторана. Со стартовой страницы

пользователь может перейти на страницу регистрации или авторизации, бронирования, заказов и меню ресторана. У администратора имеется тот же набор страниц, но с главной можно перейти в панель управления, где производится создание схемы ресторана, управление меню и просмотр статистики бронирований.

Кроме того, в системе присутствуют модальные окна: «Бронирование», «Оформление предзаказа», «Изменение деталей брони».

Каждая составляющая интерфейса будет реализована с использованием фреймворка Next.js и библиотеки React, поддерживающих свою функциональность и рабочее состояние. Применение компонентного подхода упрощает поддержку и масштабируемость приложения [14].

3 Реализация программных средств для управления гостевым сервисом в ресторане

3.1 Реализация базы данных

Физическая модель базы данных представляет собой модель базы данных, которая определяет способ и параметры результирующего представления данных, обеспечивающие дальнейшее корректное создание базы данных [15]. Физическая модель базы данных программных средств управления гостевым сервисом в ресторане представлена на рисунке 16.

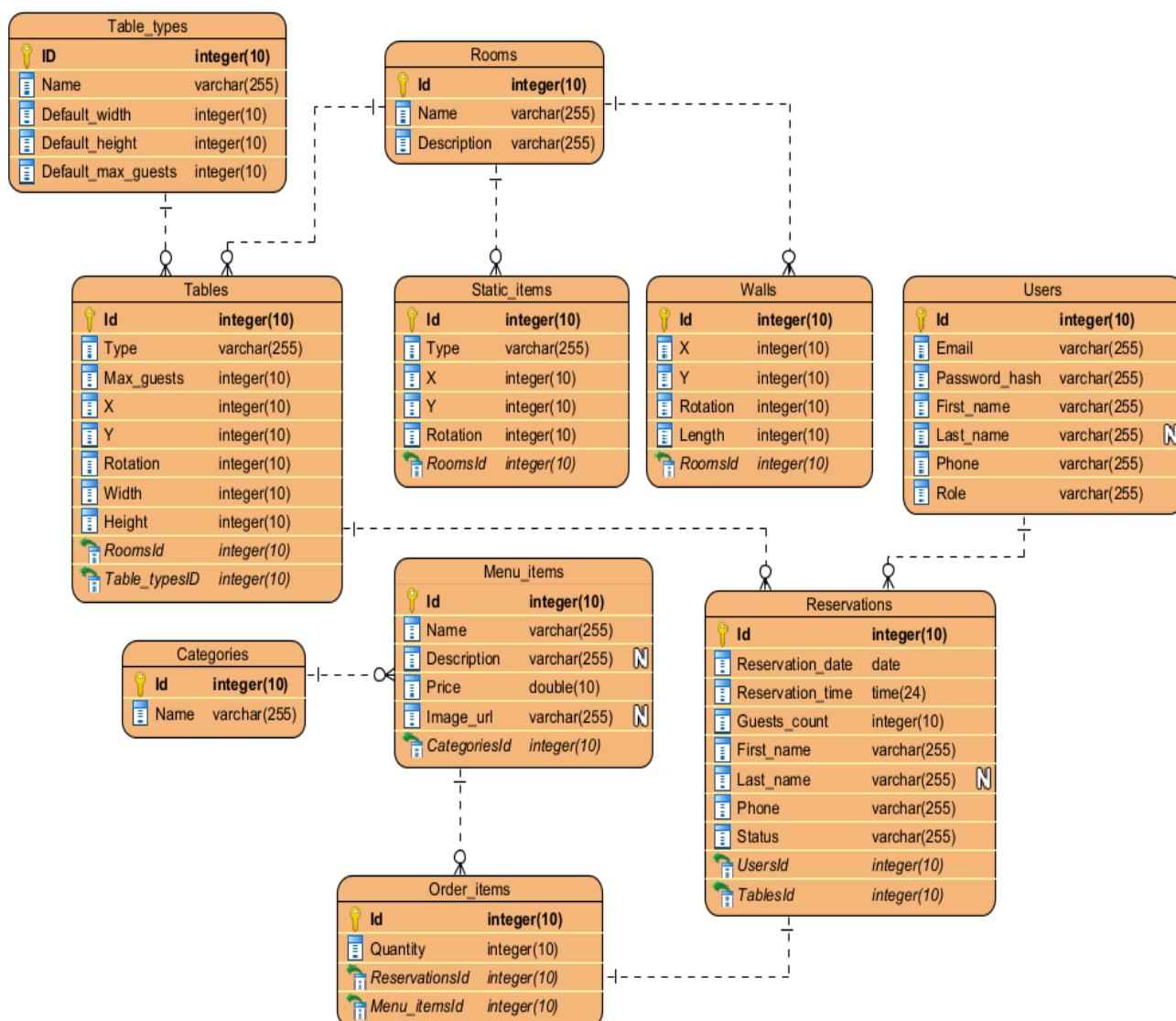


Рисунок 16 – Физическая модель базы данных

Описание отношений представлено в таблице 15.

Таблица 15 – Спецификация отношений физической модели базы данных

№ п/п	Наименование	Описание	Категория
1	Тип стола	Содержит информацию о типе стола (название, длина, ширина, максимальное число гостей)	Справочная Характеристическая
2	Комната	Содержит сведения о комнате	Стержневая
3	Стол	Содержит информацию о столе (номер стола, максимальное число мест, координата по X, координата по Y, угол поворота, длина, ширина)	Стержневая Характеристическая
4	Статический элемент	Содержит сведения о статических элементах схемы зала	Стержневая Характеристическая
5	Стена	Содержит информацию о расположении стен на схеме зала	Стержневая Характеристическая
6	Категория меню	Содержит сведения о категориях меню	Характеристическая Справочная
7	Блюдо	Содержит информацию о блюде из меню (название, описание, цена, изображение блюда)	Характеристическая Справочная
8	Предзаказ	Содержит информацию о предзаказе пользователя	Характеристическая Справочная
9	Бронь	Содержит сведения о брони гостя (дата, время брони, количество гостей, имя, фамилия, телефон пользователя, статус)	Стержневая
10	Пользователь	Содержит информацию о пользователе (электронная почта, пароль от аккаунта, имя, фамилия, номер телефона, роль)	Стержневая

По физической модели базы данных ресторана необходимо произвести описание спецификаций атрибутов для каждой сущности.

В таблице 16 описываются атрибуты сущности «Категория меню».

Таблица 16 – Атрибуты сущности «Категория меню»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор категории	int	10	-	нет	PK	2
Name	Название категории меню	varchar	255	«А – я»	нет		Салаты

К каждой категории меню относится определенный набор блюд. Атрибуты сущности «Блюдо» представлены в таблице 17.

Таблица 17 – Атрибуты сущности «Блюдо»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор блюда	int	10	-	нет	РК	2
Name	Название категории меню	varchar	255	«А – я»	нет		Салаты
Description	Описание	varchar	255	«А – я»	да		Вкусный суп
Price	Цена	double	10	>=0	нет		500
Image_url	Изображение блюда	varchar	255	-	да		-
CategoriesId	Идентификатор категории меню	int	10	1 – 9999	нет	FK	21

Атрибут «CategoriesId» является внешним ключом и обеспечивает связь между таблицами «Категория меню» и «Блюдо». Каждое блюдо относится к конкретной категории (например, нисуаз относится к категории «Салаты»). Множество блюд может соответствовать одной категории. Данная связь позволяет избежать дублирование данных и упрощает управлению меню.

В таблице 18 описываются атрибуты сущности «Комната». Она является схемой зала ресторана и хранится в базе данных в единственном экземпляре.

Таблица 18 – Атрибуты сущности «Комната»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор комнаты	int	10	-	нет	РК	1
Name	Название комнаты	varchar	255		Нет		Схема зала
Description	Описание	varchar	255		Нет		Кухня, бар, wc

В таблице 19 описываются атрибуты сущности «Тип стола». Она предназначена для хранения информации о различных вариантах стола, которой можно разместить на схеме ресторана.

Таблица 19 – Атрибуты сущности «Тип стола»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор типа стола	int	10	-	нет	PK	25
Name	Название типа стола	varchar	255	«А – я»	Нет		Круглый стол с диванами
Default_width	Длина	integer	10		Нет		250
Default_height	Ширина	integer	10		Нет		150
Default_max_guests	Максимальное число мест	integer	10		Нет		10

В таблице 20 представлены атрибуты сущности «Стол».

Таблица 20 – Атрибуты сущности «Стол»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор стола	int	10	-	нет	PK	25
Table_number	Номер стола	int	10	-	нет		2
Max_guests	Максимальное число посадочных мест	int	10	-	нет		5
X	Координата по X	int	10	-	нет		250
Y	Координата по Y	int	10	-	нет		200
Rotation	Угол поворота	int	10	-	нет		90
Width	Длина	int	10	-	нет		450
Height	Ширина	int	10	-	нет		250
RoomsId	Идентификатор комнаты	int	10	-	нет	FK	1
Table_typesId	Идентификатор типа стола	int	10	-	нет	FK	3

Атрибуты «RoomsId» и «Table_typesId» выполняют роль внешних ключей. Каждый стол относится к определенному типу. Благодаря атрибуту «Table_typesId» можно использовать конкретные характеристики для стола (длина, ширина, вместимость). Это позволяет упростить процесс проектирования схемы зала. Сгруппировать столы и привязать их к конкретной комнате можно с помощью атрибута «RoomsId».

Помимо столов на схеме зала изображаются статические элементы и расположение стен. Описание сущности «Статичный элемент» приведено в таблице 21.

Таблица 21 – Атрибуты сущности «Статичный элемент»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор статичного элемента	int	10	-	нет	PK	21
Type	Название элемента	varchar	255	«А – я»	нет		Кухня
X	Координата X	int	10	-	нет		632
Y	Координата Y	int	10	-	нет		256
Rotation	Угол поворота	int	10	-	нет		45
RoomsId	Идентификатор комнаты	int	10	-	нет	FK	1

Описание сущности «Стена» приведено в таблице 22.

Таблица 22 – Атрибуты сущности «Стена»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор стены	int	10	-	нет	PK	21
X	Координата X	int	10	-	нет		632

Продолжение таблицы 22

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Y	Координата Y	int	10	-	нет		256
Rotation	Угол поворота	int	10	-	нет		45
Length	Длина стены	int	10	-	нет		60
RoomsId	Идентификатор комнаты	int	10	-	нет	FK	1

Обе сущности привязаны к конкретной комнате. Множество стен и статичных элементов могут располагаться на схеме зала. Это достигается путем использования атрибута «RoomsId» в роли внешнего ключа.

Получить доступ ко всему функционалу веб-сайта ресторана может только авторизированный пользователь. Описание данной сущности представлено в таблице 23.

Таблица 23 – Атрибуты сущности «Пользователь»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор пользователя	int	10	-	нет	PK	12
Email	Email пользователя (логин)	varchar	255	«A – z», «@», «.», «_», «-», «0 – 9»	нет		malitka@inbox.ru
Password_hash	Пароль от аккаунта	varchar	255	«0 – 9», «A – z»	нет		Qwerty156
First_name	Имя пользователя	varchar	255	«A – я»	нет		Мария
Last_name	Фамилия пользователя	varchar	255	«A – я»	да		Лапшина
Phone	Номер телефона	varchar	255	«0 – 9», «+»	нет		+79275631214
Role	Роль в системе	varchar	255	Админ, пользователь	нет		Пользователь

В таблице 24 описываются атрибуты сущности «Бронь».

Таблица 24 – Атрибуты сущности «Бронь»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор брони	int	10	-	нет	PK	12
Reservation_date	Дата брони	date	10	-	нет		03.06.2025
Reservation_time	Время брони	time	24	12:00 – 00:00	нет		15:00
Guests_count	Число гостей	int	10	«0 – 9»	нет		2
First_name	Имя	varchar	255	«А – я»	нет		Евгений
Last_name	Фамилия	varchar	255	«А – я»	да		Ларингин
Phone	Номер телефона гостя	varchar	255	«0 – 9», «+»	нет		+79995668979
Status	Статус брони	varchar	255	«А – я»	нет		Подтверждено
UserId	Идентификатор пользователя	int	10	-	нет	FK	10
TablesId	Идентификатор стола	int	10	-	нет	FK	2

Атрибут «UserId» связывает бронь с конкретным пользователем. Он выполняет роль внешнего ключа и необходим для удобства заполнения формы брони. Благодаря этому атрибуту данные аккаунта пользователя подставляются в поля бронирования столика. Однако они могут быть изменены, если бронь нужно оформить на другого человека или указать новый номер телефона.

Атрибут «TablesId» также является внешним ключом. Он обеспечивает связь брони с конкретным столом, что позволяет корректно отобразить схему зала с забронированными местами.

Необязательным элементом является сущность «Предзаказ». Пользователю

после оформления брони предлагается выбрать блюда, которые будут для него готовы к назначенному времени. Если гость не желает выполнять эту операцию, то стол бронируется без предзаказа. Описание данной сущности представлено в таблице 25.

Таблица 25 – Атрибуты сущности «Предзаказ»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Уникальный идентификатор предзаказа	int	10	-	нет	PK	12
Quantity	Количество выбранных блюд	int	10	≥ 0	нет		2
ReservationsId	Идентификатор брони	int	10	-	нет	FK	12
Menu_itemsId	Идентификатор блюда	int	10	-	нет	FK	5

Атрибут «ReservationsId» является внешним ключом и связывает предзаказ с конкретной бронью. Это позволяет определить, для какой брони был сделан определенный предзаказ. Атрибут «Menu_itemsId» также является внешним ключом, который указывает на конкретное блюдо из меню, включенное в предзаказ. Данная связь позволяет определить состав предзаказа для каждого гостя и отображать ему актуальный набор блюд, представленных в меню ресторана.

Описание связей между сущностями физического модели базы данных представлено в таблице 26.

Таблица 26 – Спецификации связей между сущностями

№	Тип сущности	Тип сущности	Семантика связи	Мощность	Обязательность
1	Стол	Бронь	Каждый бронь относится к одному столу/ У стола может быть несколько броней на разное время	1:M	Необязательная, т. к. не каждый стол может быть забронирован

Продолжение таблицы 26

№	Тип сущности	Тип сущности	Семантика связи	Мощность	Обязательность
2	Тип стола	Стол	Каждый стол относится к определенному типу/ Множество столов может относиться к одному типу	1:M	Обязательная
3	Предзаказ	Бронь	Один предзаказ относится к одной брони и наоборот	1:1	Необязательная, т. к. бронь может существовать без предзаказа
4	Блюдо	Предзаказ	Предзаказ содержит блюдо/ Множество блюд могут быть в предзаказе	1:M	Обязательная
5	Пользователь	Бронь	Пользователь может создать множество броней/ Бронь относится к одному пользователю	1:M	Обязательная
6	Комната	Стол	Комната содержит столы/ множество столов принадлежат комнате	1:M	Обязательная
7	Комната	Статичный элемент	Комната содержит статичные элементы/ Статичные элементы принадлежат комнате	1:M	Обязательная
8	Комната	Стена	Комната содержит стены/ Стены принадлежат комнате	1:M	Обязательная
9	Категория меню	Блюдо	К одной категории относится несколько блюд/ Блюдо относится к определенной категории	1:M	Обязательная

Также на основе физической модели базы данных был сгенерирован скрипт на PostgreSQL для создания объектов базы данных [16]. Фрагмент

скрипта для создания таблиц «Бронь» и «Стол» приведен в листинге 1. Полный скрипт для создания базы данных содержится в приложении Б.

Листинг 1 – Создание таблиц «Бронь» и «Стол» в базе данных PostgreSQL

```
CREATE TABLE Reservation (
    Id SERIAL NOT NULL,
    Reservation_date date NOT NULL,
    Reservation_time time NOT NULL,
    Guests_count int NOT NULL,
    First_name varchar(255) NOT NULL,
    Last_name varchar(255),
    Phone varchar(255) NOT NULL,
    Status varchar(255) NOT NULL,
    UserId int NOT NULL,
    TablesId int NOT NULL,
    PRIMARY KEY (Id)
);

CREATE TABLE Table (
    Id SERIAL NOT NULL,
    Table_number int NOT NULL,
    Max_guests int NOT NULL,
    X int NOT NULL,
    Y int NOT NULL,
    Rotation int NOT NULL,
    Width int NOT NULL,
    Height int NOT NULL,
    RoomsId int NOT NULL,
    Table_typesId int NOT NULL,
    PRIMARY KEY (Id)
);
```

Каждая сущность имеет поля, которые являются уникальными идентификаторами. Для всех атрибутов устанавливается подходящий тип данных, а также ограничения: возможность NULL значений (при необходимости), максимальная длина строковых данных и наличие первичного ключа. Эти ограничения позволяют обеспечить корректность и целостность данных при их обновлении.

3.2 Реализация серверной части приложения

Серверная часть программных средств для управления гостевым сервисом ресторана разрабатывалась в среде программирования PyCharm, на языке программирования Python. Код серверной части приложения приведен в приложении В.

Серверная часть представляет собой разделенный монолит, где frontend и backend разделены, но функционируют между собой как единое целое. Backend содержит детализированные модели, которые напрямую отражают структуру базы данных и обеспечивают строгую типизацию и целостность данных [17].

Для упрощения обмена между сервером и клиентом используются стерилизаторы, которые преобразуют объекты модели в JSON-представление, пригодное для отправки по протоколу HTTP. Стерилизаторы улучшают и стандартизируют ответы API, устраняя возможные ошибки и упрощая анализ данных на интерфейсе. Кроме того, в серверной части существуют специальные процессы миграции, которые обновляют и поддерживают структуру базы данных в соответствии с требованиями проектов. Миграции позволяют синхронизировать базу данных с текущим состоянием моделей и позволяют очень гибко вносить изменения в схему без потери данных. Это особенно необходимо для упрощения процесса развертывания и обновления и обеспечения безопасной работы приложения при добавлении новых функций.

Разработанные сервисы предоставляют REST API для взаимодействия. Реализованные методы описаны в таблице 27.

Таблица 27 – Методы API программных средств для управления гостевым сервисом ресторана

URL запроса	Http метод	Описание
/auth/login	POST	Авторизация пользователя
/auth/register	POST	Регистрация пользователя
/user/profile	GET	Получение профиля текущего пользователя
/rooms	GET	Получение комнаты
/tables	GET	Получение списка всех столов
/table/{id}	GET	Получение информации о конкретном столе
/menu	GET	Получение списка всех блюд
/menu/{id}	GET	Получение информации о конкретном блюде

Продолжение таблицы 27

URL запроса	Http метод	Описание
/reservation	POST	Создание новой брони
/reservation/{id}	GET	Получение информации о конкретной брони
/reservation/{id}	PUT	Изменение данных брони
/reservation/{id}	DELETE	Удаление брони
/reservation/user/{userId}	GET	Получение всех броней пользователя
/preorder	POST	Создание предзаказа
/preorder/{id}	GET	Получение информации о предзаказе
/stats/average-check	GET	Получение статистики: средний чек
/stats/popular-dishes	GET	Получение статистики: популярные блюда
/stats/revenue	GET	Получение статистики: общая выручка
/stats/bookings	GET	Получение статистики: количество броней

Разработанный разделенный монолит обернут в docker контейнеры. Это сделано для дальнейшего разворачивания приложения на других серверах с платформами, поддерживающих docker. Данный подход обеспечивает независимость от операционной системы [18].

Конфигурация взаимодействия между контейнерами описана в файле `docker-compose.prod.no-nginx.yml`. После сборки запуск приложения осуществляется в изолированной среде Docker Desktop. Сначала поднимается база данных, а затем серверная и клиентская часть приложения.

3.3 Реализация клиентской части приложения

Клиентская часть программных средств для управления гостевым сервисом ресторана разрабатывалась в среде программирования Visual Studio Code, на языке программирования JavaScript с использованием фреймворка Next.js и библиотеки React [19]. Код клиентской части приложения приведен в приложении Г.

На рисунке 17 представлена диаграмма компонентов клиентской части разрабатываемого приложения.

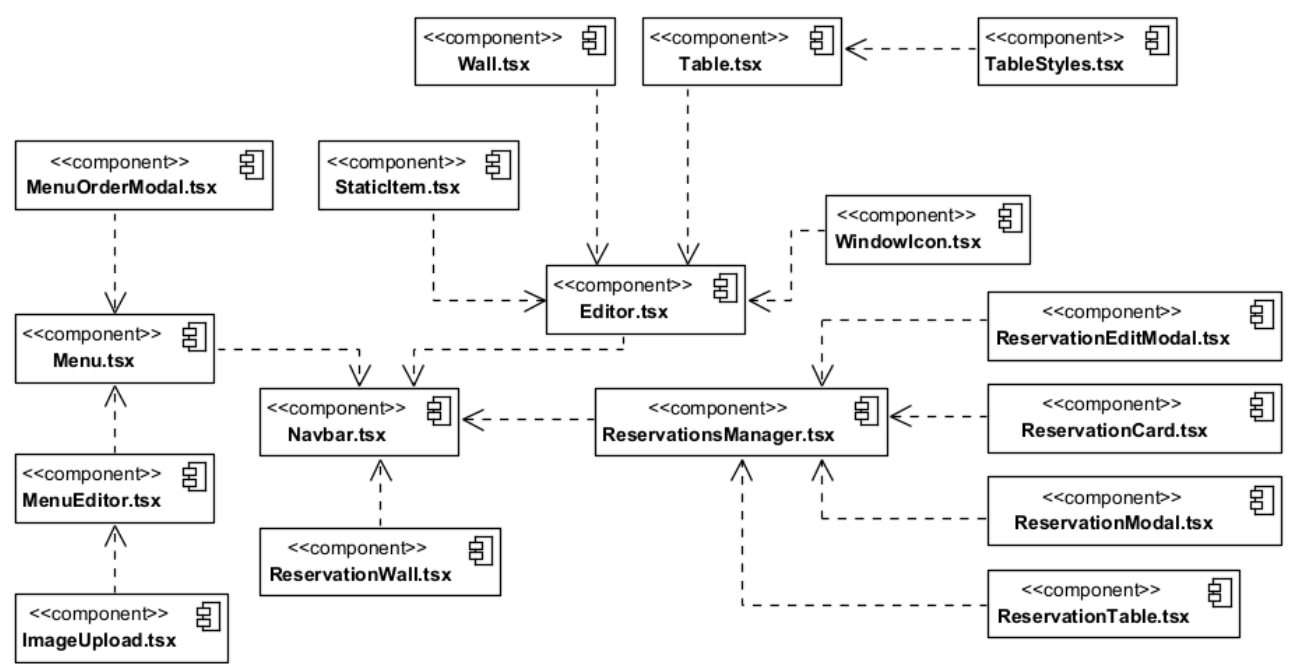


Рисунок 17 – Диаграмма компонентов клиентской части приложения

Описание компонентов клиентской части приложения представлено в таблице 28.

Таблица 28 – Компоненты клиентской части приложения

Компонент	Назначение
Editor.tsx	Компонент для редактирования схемы ресторана.
ImageUpload.tsx	Компонент для загрузки изображений (например, для блюд).
Menu.tsx	Компонент для отображения меню.
MenuEditor.tsx	Компонент для редактирования меню (для блюд).
MenuOrderModal.tsx	Модальное окно для оформления предзаказа блюда.
Navbar.tsx	Компонент навигационной панели.
ReservationCard.tsx	Карточка для отображения информации о брони.
ReservationEditModal.tsx	Модальное окно для редактирования данных брони.
ReservationModal.tsx	Модальное окно для создания новой брони.
ReservationsManager.tsx	Компонент для управления всеми бронированиями (список, фильтрация, поиск).

Продолжение таблицы 28

Компонент	Назначение
ReservationTable.tsx	Таблица для отображения всех бронирований в виде списка.
ReservationWall.tsx	Компонент для отображения бронирований в виде карточек
StaticItem.tsx	Компонент для отображения статических элементов на схеме ресторана (например, декоративных элементов).
Table.tsx	Компонент для отображения информации о столе.
TableStyles.tsx	Файл со стилями для отображения столов.
Wall.tsx	Компонент для отображения стены на схеме ресторана.
WindowIcon.tsx	Компонент для отображения иконки окна или элементов интерфейса окна.

Для обмена данными между клиентом и сервером используется библиотека `axios`. Выбор данной библиотеки обусловлен удобством обработки ответов от сервера, настройки запросов и простотой перехвата исключений.

3.4 Реализация интерфейса приложения

В соответствии с требованиями к пользовательскому интерфейсу, предоставленными в пункте 2.3, были реализованы следующие страницы веб-сайта для управления гостевым сервисом в ресторане:

- страница авторизации;
- страница регистрации;
- страница главная;
- страница меню;
- страница бронирования стола;
- страница брони;
- страница панель управления (у администратора).

При открытии приложения пользователь попадает на главную страницу веб-сайта ресторана, где ему доступен ограниченный функционал без

авторизации в личном аккаунте. Интерфейс главной страницы веб-сайта представлен на рисунке 18.

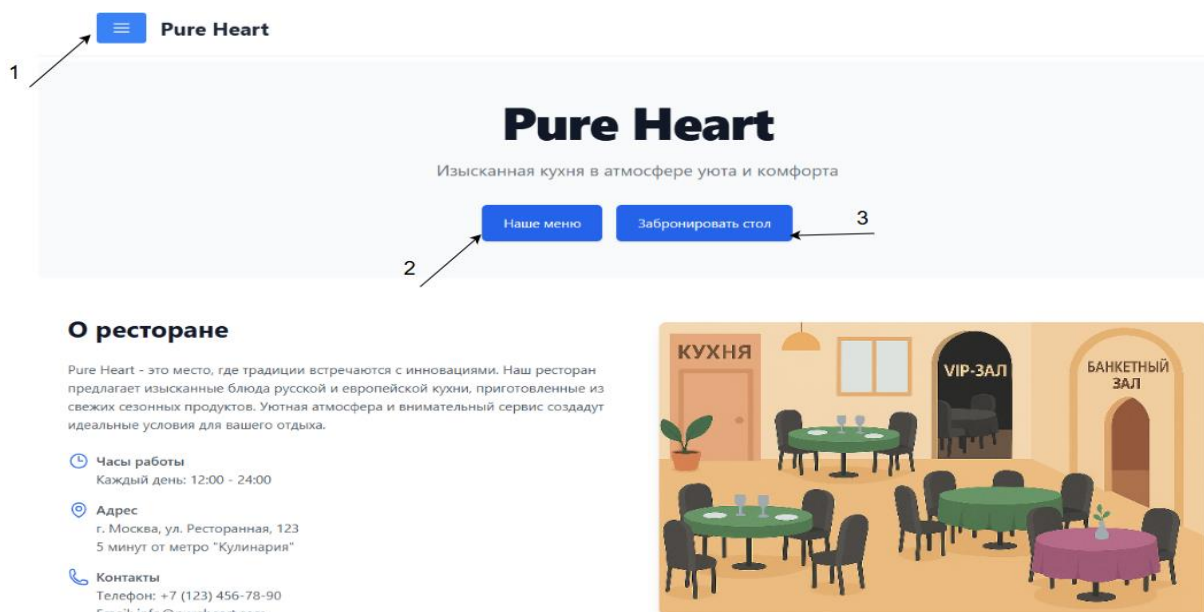


Рисунок 18 – Страница главная

Основные элементы страницы главная:

- 1 – кнопка для перехода в навигационное меню веб-сайта;
- 2 – кнопка «Наше меню»;
- 3 – кнопка «Забронировать стол».

При нажатии на кнопку для перехода в навигационное меню веб-сайта ресторана пользователю и администратору откроется панель, где можно перейти в одну из вкладок («Меню ресторана», «Забронировать стол», «Вход», «Регистрация», «Главная»). Помимо этих вкладок у администратора есть дополнительная. Это «Панель управления», предназначенная для управления меню, схемой зала, бронированиями. Кроме того, обе роли при входе на главную страницу, пролистнув вниз, могут рассмотреть схему зала ресторана и выбрать свободный столик на определённую дату и время. Однако, чтобы забронировать его, необходимо авторизоваться в системе.

На страницу регистрации можно перейти с главной страницы или со страницы авторизации. Форма регистрации представлена на рисунке 19.

The image shows a registration form with the following elements:

- 1**: A link labeled "Войдите" (Log in) for users who already have an account.
- 2**: An input field for "Email".
- 3**: An input field for "Пароль" (Password).
- 4**: An input field for "Имя" (Name).
- 5**: An input field for "Фамилия" (Surname).
- 6**: An input field for "Телефон" (Phone) with a placeholder "+7 (XXX) XXX-XX-XX".
- 7**: A blue button labeled "Зарегистрироваться" (Register).

Рисунок 19 – Страница регистрации

Основные элементы страницы регистрации:

- 1 – гиперссылка для перехода на страницу авторизации;
- 2 – поле ввода электронной почты пользователя;
- 3 – поле ввода пароля от аккаунта пользователя;
- 4 – поле ввода имени пользователя;
- 5 – поле ввода фамилии пользователя;
- 6 – поле ввода номера телефона пользователя;
- 7 – кнопка «Зарегистрироваться».

При нажатии на кнопку «Зарегистрироваться» система проверяет правильность введенных данных и, если ошибок не обнаружено и пользователя с указанными параметрами нет в базе данных, то регистрация проходит успешно. В противном случае будут выведены соответствующие сообщения об ошибках.

Страница авторизации, на которую можно попасть с главной или со страницы регистрации, представлена на рисунке 20.

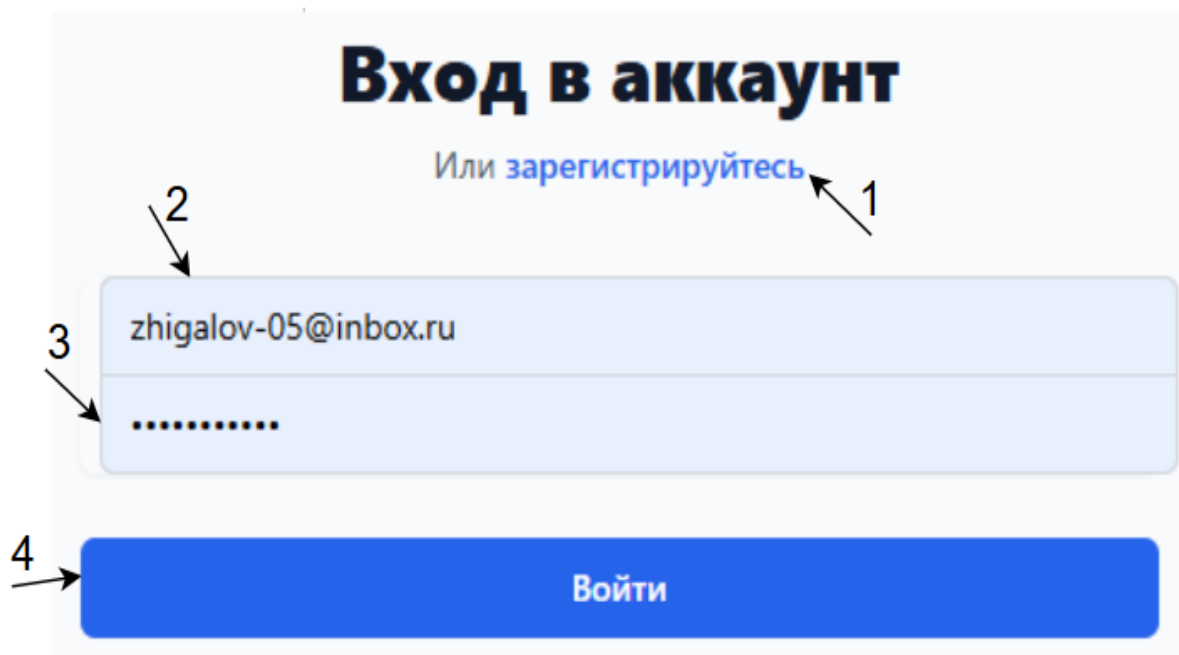


Рисунок 20 – Страница авторизации пользователя

Основные элементы страницы авторизации:

- 1 – гиперссылка для перехода на страницу регистрации;
- 2 – поле ввода электронной почты пользователя;
- 3 – поле ввода пароля от аккаунта пользователя;
- 4 – кнопка «Войти».

При нажатии на кнопку «Войти» система проверяет корректность данных и существование аккаунта в системе. Если аккаунт будет не найден в базе данных или данные некорректны, то вход не удастся и выведутся соответствующее сообщение об ошибках. В противном случае пользователь успешно выполнит вход в свой аккаунт и получит полный доступ к функционалу веб-сайта ресторана. По умолчанию аккаунт администратора нельзя зарегистрировать, он уже есть в базе данных, и в него может войти разработчик приложения или администратор ресторана.

Для формирования предзаказов необходима реализация меню ресторана. Содержимое вкладки меню представлено на рисунке 21.

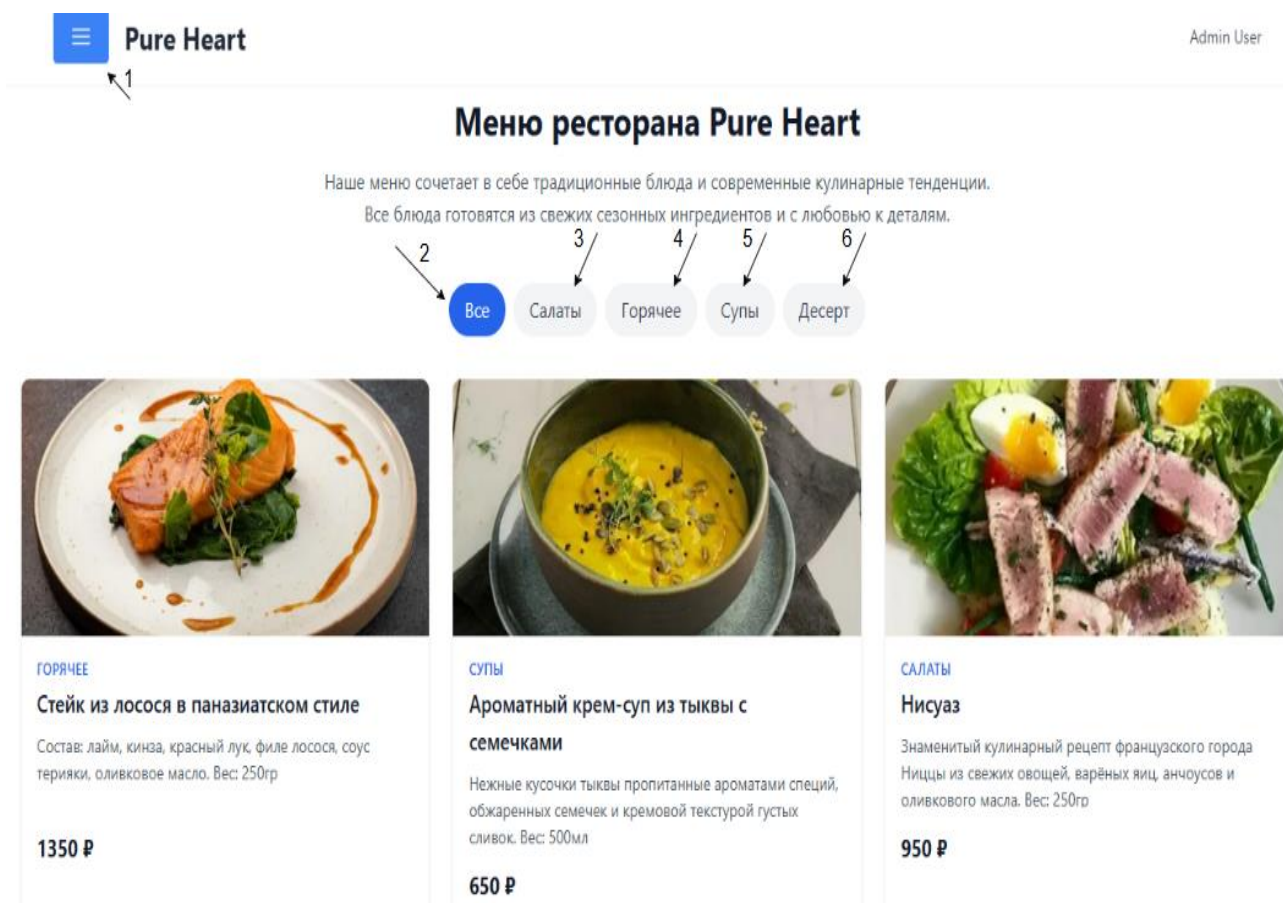


Рисунок 21 – Вкладка с меню

Основные элементы вкладки с настройками системы:

- 1 – кнопка для перехода в навигационное меню веб-сайта;
- 2 – кнопка для просмотра всех блюд из меню;
- 3 – кнопка для просмотра блюд из категории «Салаты»;
- 4 – кнопка для просмотра блюд из категории «Горячее»;
- 5 – кнопка для просмотра блюд из категории «Супы»;
- 6 – кнопка для просмотра блюд из категории «Десерт».

На данной страницы выводится меню ресторана. Пользователь может ознакомиться с внешним видом блюда, его ценой, описанием. Блюда, представленные в меню, можно выбрать в предзаказе после оформления брони стола.

Гость может просмотреть список блюд, которые относятся к конкретной категории, нажимая на кнопку с названием категории блюда.

Модальное окно для бронирования стола представлено на рисунке 22.

Забронировать столик

Количество гостей

2

Дата

01.06.2025

Время

16:00

Продолжительность

1 час

Конец брони: 17:00

Имя

Кирилл

Фамилия

Жигалов

Телефон

+7 (937) 456-85-85

Отмена

Забронировать

Рисунок 22 – Модальное окно для бронирования стола

После нажатия на кнопку «Забронировать» система предложит пользователю оформить предзаказ. Клиент выбирает необходимые блюда из категорий, указывает их количество и добавляет в свой заказ. Если гость не желает оформлять предзаказ, то он может нажать на кнопку «Не сейчас». Форму также можно закрыть, нажав на крестик. Модальное окно для оформления предзаказа к брони представлено на рисунке 23.

Выберите блюда для заказа



Меню

Салаты

Горячее

Супы

Десерт



Стейк из лосося в паназиатском стиле

Состав: лайм, кинза, красный
лук, филе лосося, соус
терияки, оливковое масло.
Вес: 250гр
1350.00 Р

- 1 +

Добавить

Ваш заказ



Нисуаз
1 × 950 Р
950 Р



Стейк из лосося в
паназиатском стиле
1 × 1350 Р
1350 Р



Итого: 2300 Р

Оформить заказ

Не сейчас

Рисунок 23 – Модальное окно оформления предзаказа

Содержимое вкладки «Мои заказы» представлено на рисунке 24.

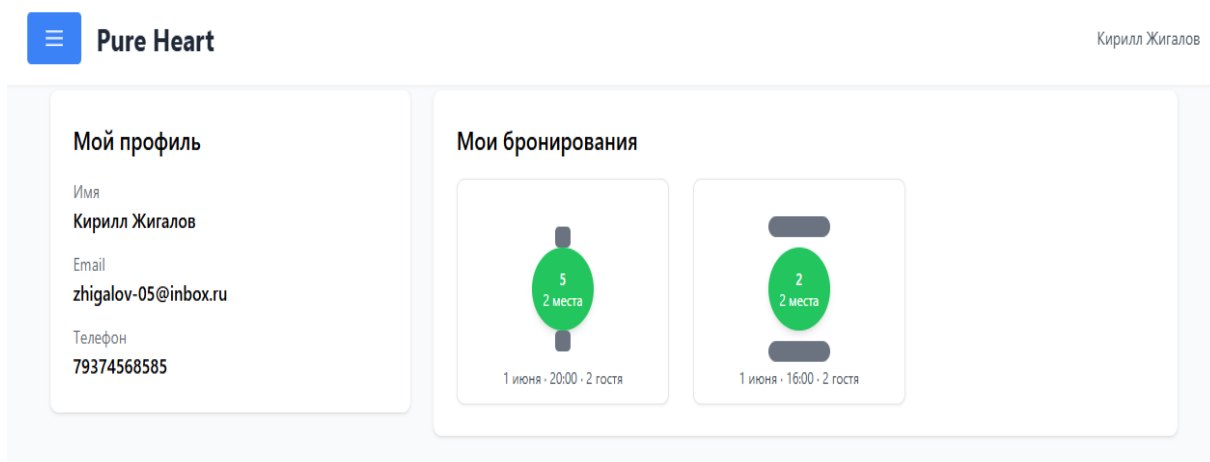


Рисунок 24 – Страница просмотра броней

На данной странице пользователь видит свои личные данные аккаунта и бронирования. При нажатии на столик будет открыто модальное окно с информацией о бронировании, где можно отменить бронь или изменить ее данные. Модальное окно с информацией о брони представлено на рисунке 25.

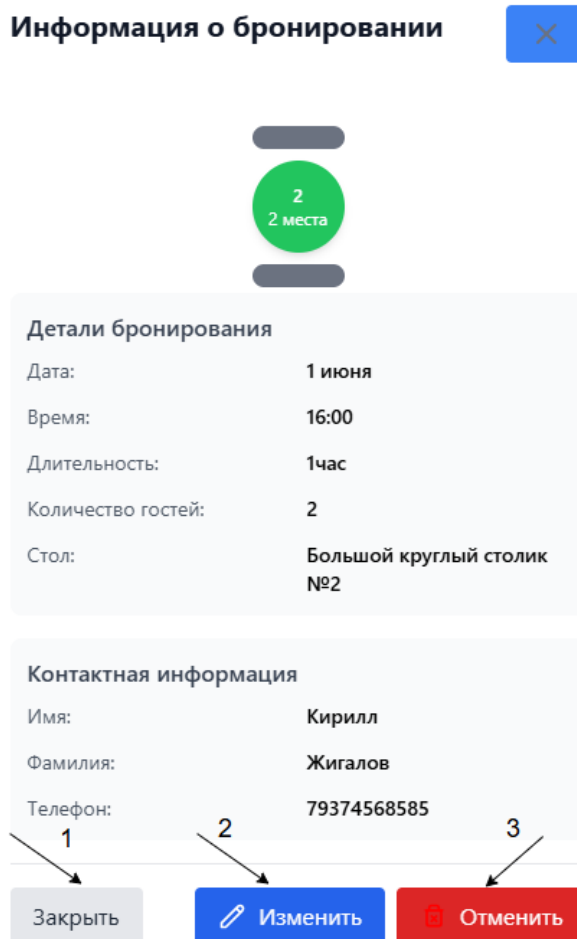


Рисунок 25 – Модальное окно просмотра информации о бронировании

Основные элементы модального окна:

- 1 – кнопка для закрытия модального окна;
- 2 – кнопка для изменения деталей брони;
- 3 – кнопка для отмены брони;
- 4 – кнопка для просмотра блюд из категории «Горячее».

При нажатии на кнопку изменения деталей бронирования открывается модальное окно бронирования стола. Пользователь вносит корректировки в свой заказ и сохраняет новые данные. Кнопка отмены удаляет бронь из базы данных. Оба действия доступны гостю, если разница между текущим временем и временем брони не более 6 часов. По истечению этого срока выполнить операции редактирования и отмены заказа невозможно.

Страница панель управления у администратора содержит три вкладки: план зала, меню и бронирования.

Интерфейс панели управления представлен на рисунке 26.

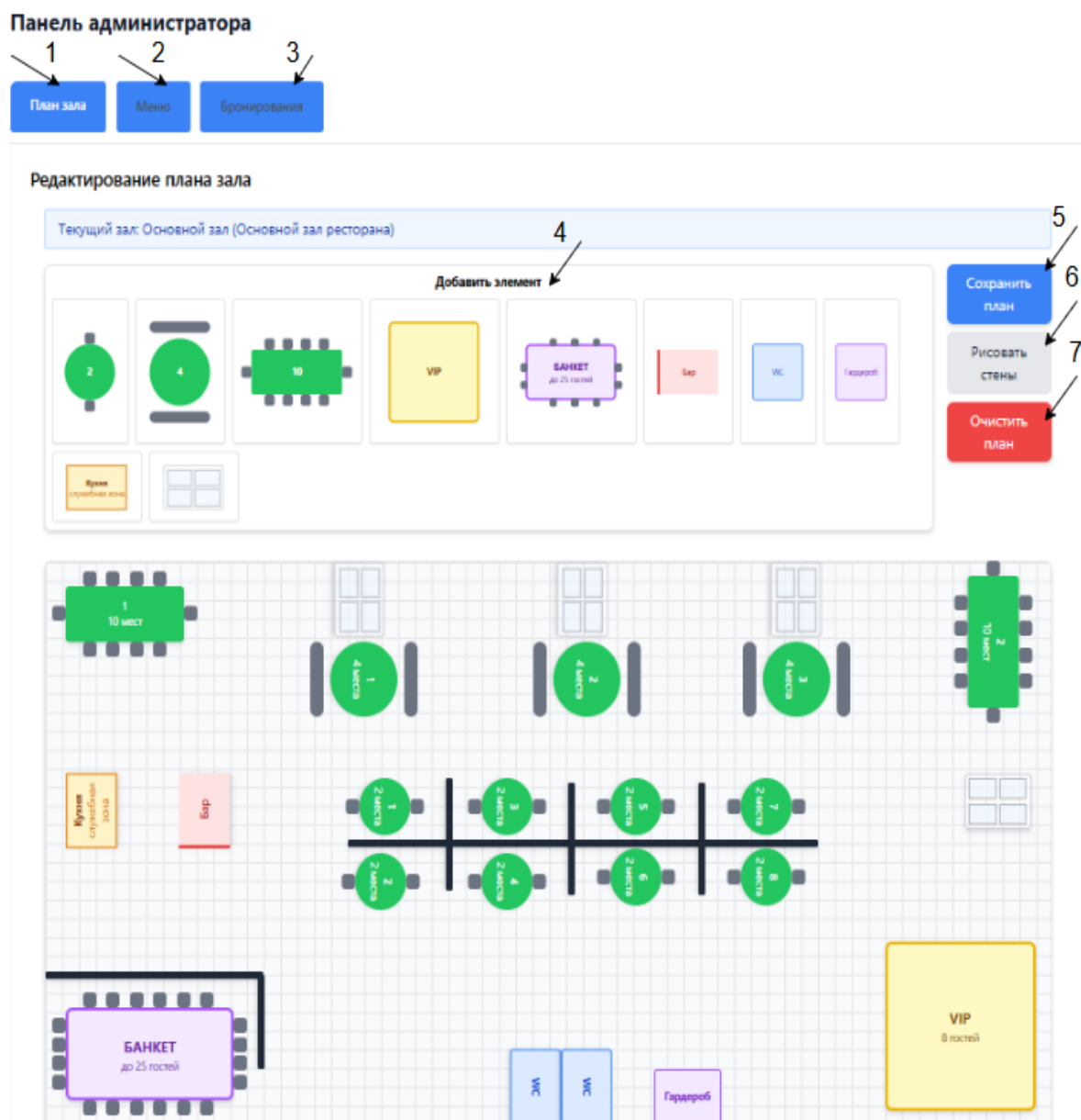


Рисунок 26 – Страница панели управления администратора

Основные элементы панели управления:

- 1 – кнопка для перехода на вкладку «План зала»;
- 2 – кнопка для перехода на вкладку «Меню»;
- 3 – кнопка для перехода на вкладку «Бронирования»;
- 4 – список доступных объектов для добавления на схему зала;
- 5 – кнопка для сохранения плана зала;
- 6 – кнопка для перехода в режим рисования стен;
- 7 – кнопка для очистки схемы зала.

Администратор выбирает из списка любой объект и наносит на схему зала. Элементы можно удалить или повернуть. Аналогичные действия можно делать со стенами.

Нажимая на кнопку «Меню» открывается страница с управлением данного раздела. Интерфейс представлен на рисунке 27.

Панель администратора

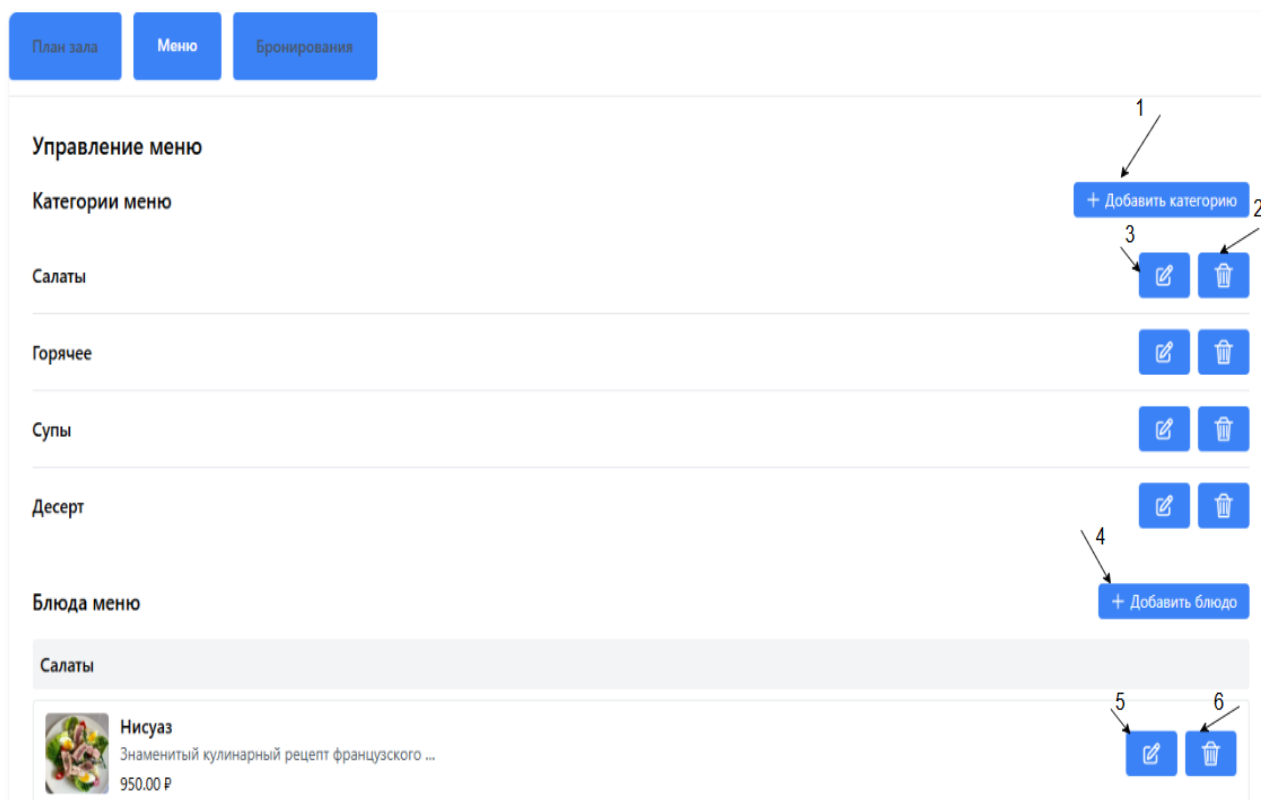


Рисунок 27 – Вкладка управления меню

Основные элементы управления меню:

- 1 – кнопка для добавления категории меню;
- 2 – кнопка для удаления категории;
- 3 – кнопка для изменения названия категории;
- 4 – кнопка для добавления блюда в меню;
- 5 – кнопка для изменения информации о блюде;
- 6 – кнопка для удаления блюда из меню.

Чтобы добавить блюдо в меню, необходимо сначала создать хотя бы одну категорию. После этого администратор нажимает на кнопку «Добавить блюдо»

и открывается модальное окно добавления блюда, интерфейс которого представлен на рисунке 28.

Блюда меню + Добавить блюдо

Новое блюдо

Название блюда Цена (₽)

Описание

Категория

Изображение блюда

Нет изображения

Загрузить изображение

Отмена Сохранить

Рисунок 28 – Модальное окно для добавления блюда в меню

Администратор указывает название блюда, его стоимость, описание, выбирает категорию, к которой оно будет относиться и добавляет изображение. Заполнив все поля, админ нажимает кнопку «Сохранить». Поля цена, описание и изображения могут быть пустыми.

Последним элементом панели управления является раздел «Бронирования». Администратор в данной вкладке управляет бронированиями и просматривает статистику ресторана. Во вкладке представлена информация о популярных блюдах, общей сумме выручки, среднем числе гостей, продолжительности брони, среднем чеке, распределение бронирований по столам и дням. Для наглядности в разделе изображены круговые диаграммы и гистограмма. Они позволяют наглядно отразить показатели важных критериев, которые необходимы для создания удобного и эффективного гостевого сервиса в ресторане. Интерфейс вкладки «Бронирования» представлен на рисунке 29.

Панель администратора

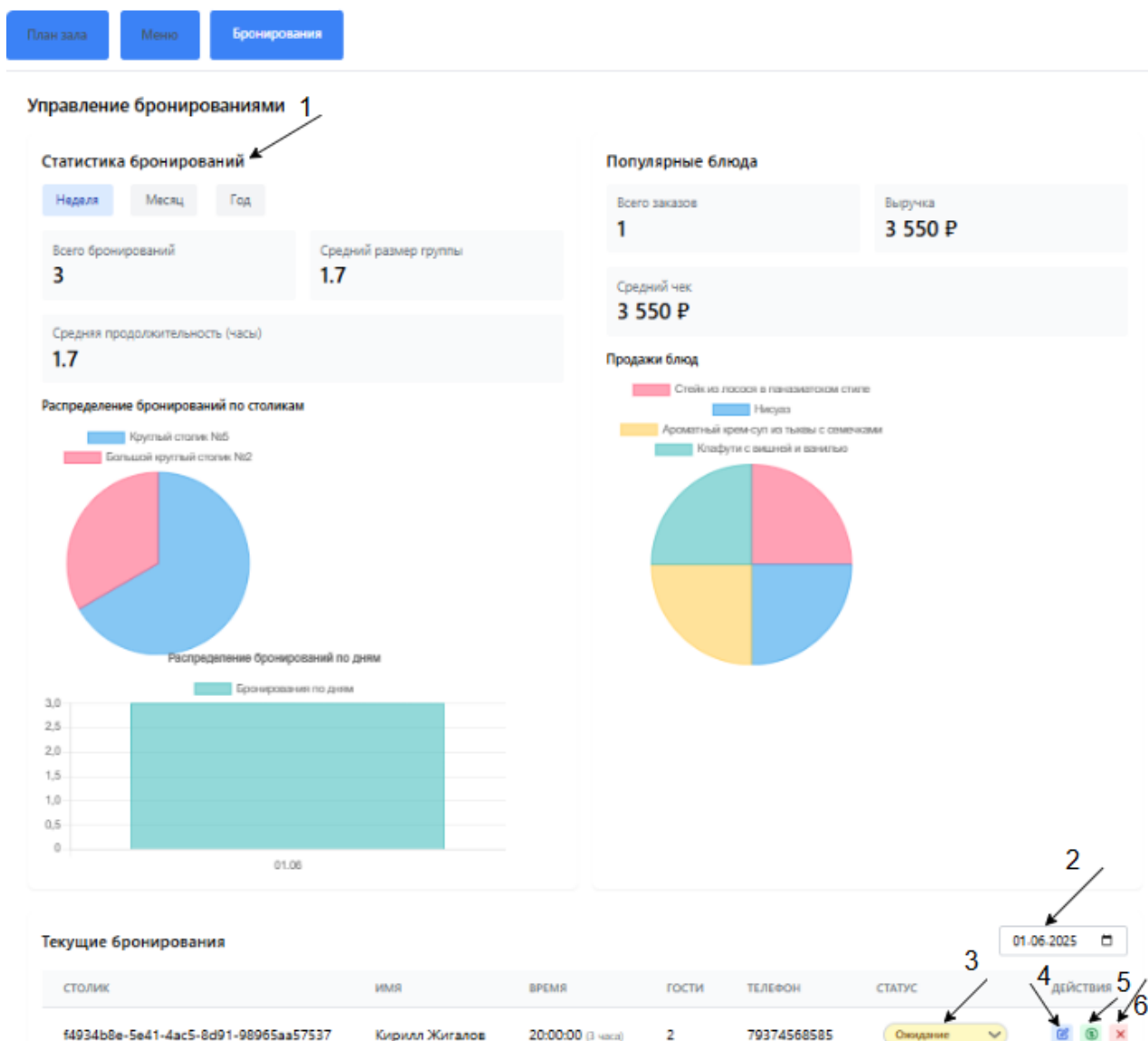


Рисунок 29 – Вкладка «Бронирования»

Основные элементы вкладки бронирования:

- 1 – кнопки для выбора промежутка времени просмотра статистики (неделя, месяц, год);
- 2 – календарь для выбора даты;
- 3 – кнопка для смены статуса брони (ожидание, подтверждено, отменено);
- 4 – кнопка для изменения деталей брони;
- 5 – кнопка для имитации оплаты заказа;
- 6 – кнопка для удаления брони.

В функционал по управлению бронированиями входит: смена статуса, изменение информации, имитация оплаты и удаление брони. Данный раздел позволяет администратору автоматизировать процесс сбора статистики и управления бронированиями.

Графический интерфейс был разработан на основе современных технологий: HTML5 для структурирования содержимого веб-страниц и CSS3 для их оформления, включая дизайн шрифтов, цветовую палитру и другие элементы. В качестве библиотеки для визуализации календаря использовался компонент `react-big-calendar`, а для построения аналитических данных – `chart.js`. Эта библиотека позволяет создавать множество различных диаграмм и графиков, обеспечивая визуализацию данных и простоту их анализа. Благодаря своей гибкости и адаптивности `chart.js` легко интегрируется в интерфейс приложения, упрощая восприятие сложных аналитических показателей [20].

4 Контроль качества программного обеспечения

4.1 Обоснование выбора режима тестирования

Для обеспечения качества программного кода, улучшения его надежности и удобства, необходимо произвести тестирование приложения различными методами. Тестирование – это проверка созданного программного продукта на соответствие заданным требованиям и на отсутствие дефектов [21].

Методы тестирования подразделяют на ручное и автоматическое. При ручном тестировании все тесты вводятся вручную и, в случае с REST API, с использованием специального инструмента для взаимодействия.

Выделяют несколько основных видов тестирования:

- модульное тестирование;
- нефункциональное тестирование;
- тестирование производительности;
- функциональное тестирование;
- интеграционное тестирование.

Модульное тестирование предназначено для тестирования отдельных программных элементов или модулей. Автоматизированное тестирование такого типа позволяет быстро выполнять проверки, используя функциональность серверов для обеспечения непрерывной интеграции.

Тестирование производительности – это метод, который позволяет количественно оценить такие характеристики программного обеспечения, как надежность, производительность и отклик системы.

Функциональное тестирование – это процесс проверки программного продукта на соответствие требованиям, описанным в функциональной документации [22]. Ключевыми преимуществами функционального тестирования являются имитация реальных условий работы системы, раннее выявление программных сбоев и экономия ресурсов за счет исправления ошибок на ранней стадии разработки. Функциональное тестирование состоит

из нескольких этапов: подготовка, тестирование и отчетность. На этапе подготовки проверяется проектная документация, составляется план тестирования, который подписывается с заказчиком. На этапе выполнения проводится ручное тестирование системы на основе заранее подготовленных тестовых сценариев, и все найденные ошибки исправляются. Заключительный этап включает в себя создание и утверждение отчетов об испытаниях, в которых приводится перечень обнаруженных дефектов и даются рекомендации о том, как можно улучшить систему [23].

Нефункциональное тестирование направлено на проверку атрибутов программного обеспечения, не имеющих непосредственного отношения к его функциям. В отличие от функционального тестирования, эта проверка позволяет проверить поведение программы в реальных условиях эксплуатации. Данный вид тестирования важен для обеспечения высокого качества продукта и определения способов взаимодействия системы с пользователем.

Интеграционное тестирование проводится для обеспечения надлежащего взаимодействия между различными компонентами и сервисами в системе. Данный вид тестирования предназначен для проверки корректности взаимодействия между серверной частью приложения и клиентской с помощью API.

Для проведения ручного тестирования серверной части программных средств для управления гостевым сервисом в ресторане было выбрано функциональное тестирование. Преимущества этого метода заключаются в следующем:

- высокая гибкость, позволяющая быстро адаптировать сценарии тестирования к меняющимся потребностям проекта;
- возможность выявлять ошибки даже на начальных этапах разработки, что сводит к минимуму вероятность появления ошибок в конечном продукте;
- множество условий тестирования, которые более разнообразны и точны, чем автоматизированное тестирование.

Также было проведено нефункциональное тестирование, целью которого было проверить такие аспекты, как производительность, стабильность и безопасность системы. Этот тип тестирования позволит определить, насколько стабильно и надежно работает приложение в различных условиях эксплуатации, и выявить возможные узкие места, которые могут повлиять на качество конечного продукта.

4.2 Обоснование выбора метрик программного кода

Метрики кода – это набор оценок программного обеспечения, которые дают разработчикам более глубокое представление о программном коде [24]. Выделяют следующие наиболее распространенные метрики кода:

- метрики устойчивости и абстрактности компонентов приложения;
- комплексный показатель качества кода;
- метрика Чепина – оценка качества внутренней логики модулей программы;
- метрика Холстеда – анализ строк и синтаксических параметров кода программы.

Для оценки контроля качества разработанного серверного приложения были выбраны метрики Холстеда и Чепина [25]. Они позволяют оценить производительность программы с точки зрения количества операндов, операторов и строк программного кода. Их можно использовать для оценки сложности написанного программного кода.

4.3 Тестирование программных средств для управления гостевым сервисом в ресторане

4.3.1 Функциональное тестирование

После завершения разработки программного обеспечения было проведено функциональное тестирование. Тестирование программного обеспечения было проведено с использованием подхода тестирования «белым ящиком». Тестирование «белого ящика» – это способ проверки программного

обеспечения, который концентрируется на внутренней системе и коде программы. В процессе тестирования были проверены все возможные сценарии работы функции «create_reservation», включая различные комбинации входных данных, таких как количество гостей, имя и фамилия, номер телефона, дата и время бронирования. Код представлен в листинге 2.

Листинг 2 – Создание брони стола

```
def create_reservation(reservation_data: ReservationCreate,
user_id: UUID, session: Session):
    # Проверка даты бронирования
    today = date.today()
    max_date = today + timedelta(days=MAX_DAYS_ADVANCE)

    if reservation_data.reservation_date < today:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Нельзя бронировать на прошедшую дату"
        )

    if reservation_data.reservation_date > max_date:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Бронирование возможно только до {max_date}"
        )

    # Проверка продолжительности бронирования
    if reservation_data.duration < 1:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Минимальная продолжительность бронирования - 1
час"
        )

    if reservation_data.duration > 6:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Максимальная продолжительность бронирования -
6 часов"
        )

    # Проверка пересечения с временем закрытия
    if reservation_data.reservation_time.hour +
reservation_data.duration > CLOSING_HOUR + 1:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Время бронирования выходит за пределы рабочих
часов (до {CLOSING_HOUR}:00) "
        )
```

```

)

# Проверка времени бронирования в текущий день
if reservation_data.reservation_date == today:
    now = datetime.now()
    current_hour = now.hour
    if reservation_data.reservation_time.hour <= current_hour:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Бронирование возможно только на время
позже текущего часа"
        )

# Проверка вместимости столика
if reservation_data.guests_count > table.max_guests:
    raise HTTPException(
        status_code=status.HTTP_400_BAD_REQUEST,
        detail=f"Столик вмещает максимум {table.max_guests}
гостей"
    )

# Проверка существования столика
table = session.exec(
    select(Table).where(
        and_(
            Table.id == reservation_data.table_id,
            Table.is_active == True
        )
    )
).first()

if not table:
    raise HTTPException(
        status_code=status.HTTP_404_NOT_FOUND,
        detail="Столик не найден или неактивен"
    )

else:
    existing_reservations = session.exec(
        select(Reservation).where(
            and_(
                Reservation.table_id ==
reservation_data.table_id,
                Reservation.reservation_date ==
reservation_data.reservation_date
            )
        )
    ).all()

```

Схема программы функции «create_reservation» и ее операторы представлены на рисунке 30.

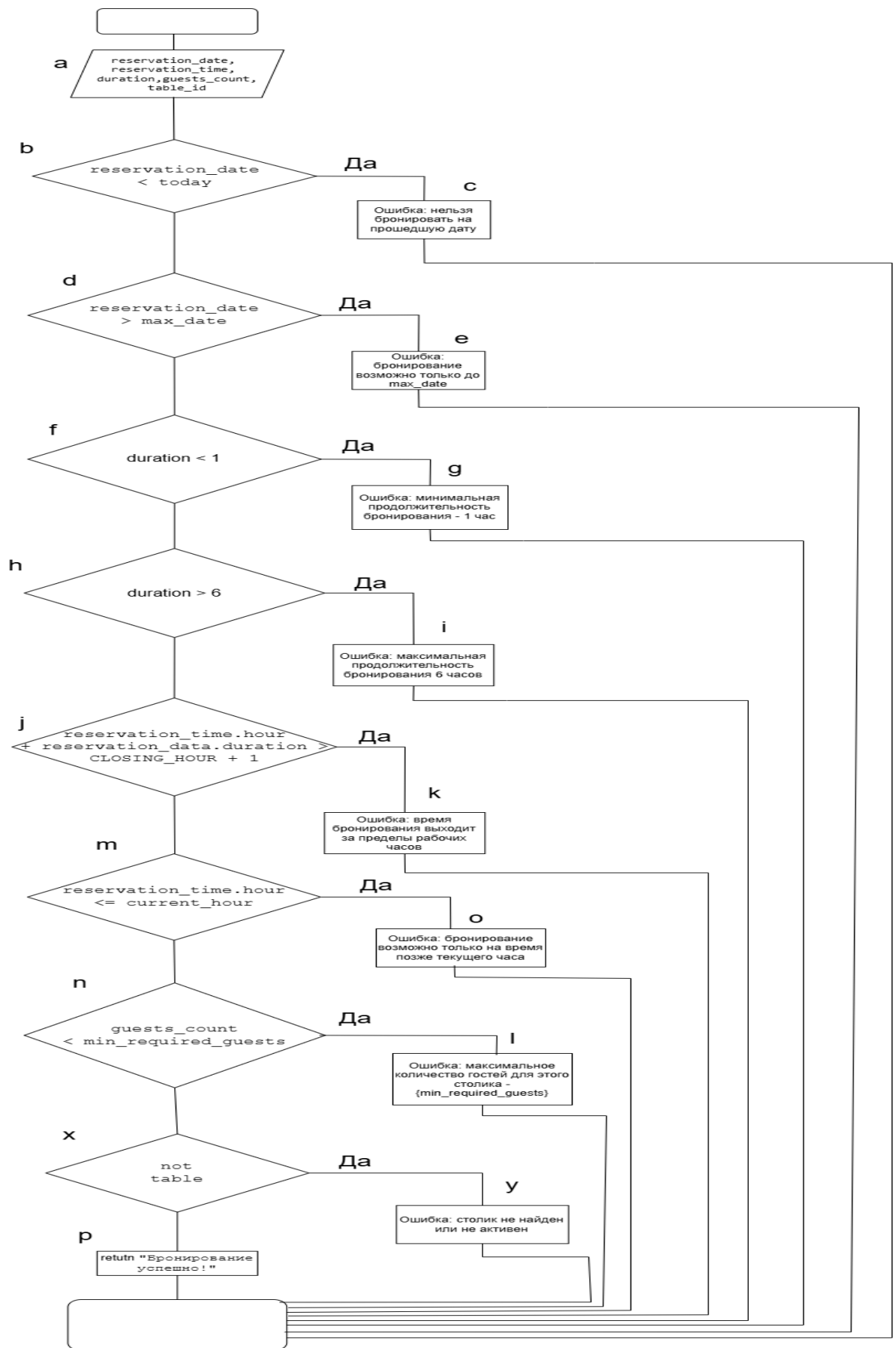


Рисунок 30 – Схема программы функции «create_reservation»

Покрытие операторов для данного кода представлено в таблице 29.

Таблица 29 – Покрытие операторов

№	Операторы	Входные данные	Выходные данные
1	a, b, d, f, h, j, m, n, x, p	reservation_date = 2025 – 06 – 03, reservation_time = 14:00, duration = 2, guests_count = 4, table_id = 1	Бронирование успешно!
2	a, b, c	reservation_date = 2024 – 02 – 03, reservation_time = 18:00, duration = 1, guests_count = 2, table_id = 2	Ошибка: нельзя бронировать на прошедшую дату
3	a, d, e	reservation_date = 2025 – 10 – 03, reservation_time = 12:00, duration = 4, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только до 2025 – 06 – 17
4	a, f, g	reservation_date = 2025 – 06 – 07, reservation_time = 15:00, duration = 0, guests_count = 2, table_id = 1	Ошибка: минимальная продолжительность – 1 час
5	a, h, i	reservation_date = 2025 – 06 – 05, reservation_time = 18:00, duration = 10, guests_count = 1, table_id = 2	Ошибка: максимальная продолжительность – 6 часов
6	a, j, k	reservation_date = 2025 – 06 – 10, reservation_time = 02:00, duration = 2, guests_count = 1, table_id = 1	Ошибка: время бронирования выходит за пределы рабочих часов (до 00:00)
7	a, m, o	reservation_date = 2025 – 06 – 05, reservation_time = 16:00, duration = 1, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только позже текущего часа (текущий час 16:20)
8	a, n, l	reservation_date = 2025 – 06 – 15, reservation_time = 18:00, duration = 5, guests_count = 25, table_id = 5	Ошибка: столик вмещает максимум N гостей (max_guests = 5)
9	a, x, y	reservation_date = 2025 – 06 – 10, reservation_time = 15:00, duration = 2, guests_count = 2, table_id = 10	Ошибка: столик не найден или неактивен

Покрытие решений представлено в таблице 30.

Таблица 30 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – нет, x – нет, p	reservation_date = 2025 – 06 – 15, reservation_time = 14:00, duration = 2, guests_count = 4, table_id = 1	Бронирование успешно!
2	a, b – да, c	reservation_date = 2025 – 03 – 10, reservation_time = 18:00, duration = 1, guests_count = 2, table_id = 2	Ошибка: нельзя бронировать на прошедшую дату

Продолжение таблицы 30

№	Решения	Входные данные	Выходные данные
3	a, b – нет, d – да, e	reservation_date = 2025 – 07 – 28, reservation_time = 12:00, duration = 4, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только до 2025 – 06 – 17
4	a, b – нет, d – нет, f – да, g	reservation_date = 2025 – 06 – 10, reservation_time = 15:00, duration = 0, guests_count = 2, table_id = 1	Ошибка: минимальная продолжительность – 1 час
5	a, b – нет, d – нет, f – нет, h – да, i	reservation_date = 2025 – 06 – 13, reservation_time = 18:00, duration = 10, guests_count = 1, table_id = 2	Ошибка: максимальная продолжительность – 6 часов
6	a, b – нет, d – нет, f – нет, h – нет, j – да, k	reservation_date = 2025 – 06 – 08, reservation_time = 02:00, duration = 2, guests_count = 1, table_id = 1	Ошибка: время бронирования выходит за пределы рабочих часов (до 00:00)
7	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – да, o	reservation_date = 2025 – 06 – 12, reservation_time = 16:00, duration = 1, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только позже текущего часа (текущий час 16:20)
8	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – да, l	reservation_date = 2025 – 06 – 10, reservation_time = 18:00, duration = 5, guests_count = 25, table_id = 5	Ошибка: столик вмещает максимум N гостей (max_guests = 5)
9	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – нет, x – да, y	reservation_date = 2025 – 06 – 10, reservation_time = 15:00, duration = 2, guests_count = 2, table_id = 10	Ошибка: столик не найден или неактивен

Покрывание условий: записывается число тестов достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены по крайней мере один раз.

Однако этот критерий не всегда приводит к выполнению каждого оператора по крайней мере один раз. Поэтому к этому критерию добавляется дополнительное условие, каждый оператор должен быть выполнен хотя бы один раз. В нашем случае каждый оператор выполняется минимум один раз,

поэтому в построении таблиц тестов нет необходимости. Покрытие решений и условий представлено в таблице 31.

Таблица 31 – Покрытие решений и условий

№	Условие	Входные данные	Выходные данные
1	a, b – нет, d, e – нет, f, g – нет, h, i – нет, j, k – нет, m, o – нет, n – нет, x – нет, p	reservation_date = 2025 – 06 – 10, reservation_time = 16:00, duration = 2, guests_count = 4, table_id = 2	Бронирование успешно!
2	a, b – да, c	reservation_date = 2024 – 08 – 10, reservation_time = 18:00, duration = 1, guests_count = 2, table_id = 2	Ошибка: нельзя бронировать на прошедшую дату
3	a, b – нет, d – да, e	reservation_date = 2026 – 06 – 10, reservation_time = 12:00, duration = 4, guests_count = 2, table_id = 5	Ошибка: бронирование возможно только до 2025 – 06 – 18
4	a, b – нет, d – нет, f – да, g	reservation_date = 2025 – 06 – 15, reservation_time = 19:00, duration = 0, guests_count = 2, table_id = 1	Ошибка: минимальная продолжительность – 1 час
5	a, b – нет, d – нет, f – нет, h – да, i	reservation_date = 2025 – 06 – 10, reservation_time = 18:00, duration = 12, guests_count = 5, table_id = 2	Ошибка: максимальная продолжительность – 6 часов
6	a, b – нет, d – нет, f – нет, h – нет, j – да, k	reservation_date = 2025 – 06 – 08, reservation_time = 02:00, duration = 2, guests_count = 1, table_id = 1	Ошибка: время бронирования выходит за пределы рабочих часов (до 00:00)
7	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – да, o	reservation_date = 2025 – 06 – 15, reservation_time = 18:00, duration = 1, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только позже текущего часа (текущий час 18:23)
8	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – да, l	reservation_date = 2025 – 06 – 05, reservation_time = 18:00, duration = 5, guests_count = 125, table_id = 15	Ошибка: столик вмещает максимум N гостей (max_guests = 5)

Продолжение таблицы 31

№	Условие	Входные данные	Выходные данные
9	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – нет, x – да, y	reservation_date = 2025 – 06 – 10, reservation_time = 15:00, duration = 2, guests_count = 2, table_id = 10	Ошибка: столик не найден или неактивен

Комбинаторное покрытие условий: этот критерий требует, чтобы все возможные комбинации результатов условий в каждом решении, а также каждый оператор выполнились по крайней мере один раз. Это позволяет выявить ошибки, возникающие при специфических сочетаниях условий.

Выделены следующие условия алгоритма:

- дата должна быть в пределах 14 дней от текущей, в противном случае возвращается ошибка;
- дата бронирования не должна отличаться от текущей более чем на 14 дней, в противном случае возвращается ошибка;
- продолжительность должна быть больше или равна одному часу, в противном случае возвращается ошибка;
- продолжительность должна быть не более 6 часов, в противном случае возвращается ошибка;
- время брони должно находиться в промежутке рабочего времени (от 12:00 до 00:00), в противном случае возвращается ошибка;
- время брони не должно быть раньше текущего времени, в противном случае возвращается ошибка;
- количество гостей для конкретного стола не должно быть больше максимально допустимого, в противном случае возвращается ошибка;
- стол должен существовать в базе данных и быть активным, если он не занят, в противном случае возвращается ошибка.

Комбинаторные условия:

- дата находится в промежутке 14 дней от текущей даты;

- продолжительность брони находится в промежутке от 1 до 6 часов;
- время брони находится в промежутке от 12:00 до 00:00;
- время должно быть не раньше текущего, если выбрана сегодняшняя дата;
- указано допустимое для стола количество мест;
- стол доступен и свободен.

Комбинаторное покрытие условий представлено в таблице 32.

Таблица 32 – Комбинаторное покрытие условий

№	Условие	Входные данные	Выходные данные
1	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – нет, x – нет, p	reservation_date = 2025 – 06 – 10, reservation_time = 14:00, duration = 2, guests_count = 4, table_id = 1	Бронирование успешно!
2	a, b – да, c	reservation_date = 2025 – 06 – 01, reservation_time = 18:00, duration = 1, guests_count = 2, table_id = 2	Ошибка: нельзя бронировать на прошедшую дату
3	a, b – нет, d – да, e	reservation_date = 2025 – 09 – 10, reservation_time = 12:00, duration = 4, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только до 2025 – 06 – 17
4	a, b – нет, d – нет, f – да, g	reservation_date = 2025 – 06 – 14, reservation_time = 15:00, duration = 0, guests_count = 2, table_id = 1	Ошибка: минимальная продолжительность – 1 час
5	a, b – нет, d – нет, f – нет, h – да, i	reservation_date = 2025 – 06 – 13, reservation_time = 18:00, duration = 10, guests_count = 1, table_id = 2	Ошибка: максимальная продолжительность – 6 часов
6	a, b – нет, d – нет, f – нет, h – нет, j – да, k	reservation_date = 2025 – 06 – 10, reservation_time = 02:00, duration = 2, guests_count = 1, table_id = 1	Ошибка: время бронирования выходит за пределы рабочих часов (до 00:00)
7	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – да, o	reservation_date = 2025 – 06 – 12, reservation_time = 16:00, duration = 1, guests_count = 2, table_id = 3	Ошибка: бронирование возможно только позже текущего часа (текущий час 16:20)
8	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – да, l	reservation_date = 2025 – 06 – 17, reservation_time = 18:00, duration = 5, guests_count = 25, table_id = 5	Ошибка: столик вмещает максимум N гостей (max_guests = 5)

Продолжение таблицы 32

№	Условие	Входные данные	Выходные данные
9	a, b – нет, d – нет, f – нет, h – нет, j – нет, m – нет, n – нет, x – да, y	reservation_date = 2025 – 06 – 09, reservation_time = 15:00, duration = 2, guests_count = 2, table_id = 1500	Ошибка: столик не найден или неактивен

Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является критерий набора тестов, которого: вызывает выполнение всех результатов каждого решения, по крайней мере, один раз и выполняет каждый оператор по крайней мере один раз.

Для программ, содержащих более одного условия минимальный критерий состоит из набора тестов, вызывающих выполнение всех возможных комбинаций результатов условий и выполняющий каждый оператор минимум один раз.

Программа успешно прошла функциональное тестирование на примере тестирования методом «белого ящика», показав правильность работы программных средств и полное соответствие функциональным требованиям.

4.3.2 Нефункциональное тестирование

Нефункциональное тестирование – это процесс проверки программного обеспечения с целью оценки его нефункциональных характеристик. В отличие от функционального тестирования, которое проверяет правильность выполнения функций программы, нефункциональное тестирование оценивает качество работы системы в реальных условиях использования. Оно позволяет выявить узкие места в производительности, слабые места в защите данных, проблемы с адаптацией под разные устройства и неудобства в пользовательском интерфейсе. Нефункциональное тестирование проводится после завершения функционального тестирования, когда основные функции программы работают корректно.

Цели нефункционального тестирования:

- оценить производительность системы под нагрузкой;

- проверить стабильность и надежность программы при длительной работе;
- проверить совместимость с разными устройствами и операционными системами;
- проверить удобство использования и интуитивность интерфейса;
- оценить возможность восстановления системы после сбоев.

Проведем 5 типов тестов, каждый из которых оценивает определенные характеристики разработанной программы.

Первым тестом является тестирование производительности. Его целью является проверка скорости работы приложения и его отклик под нагрузкой. План тестирования представлен в таблице 33.

Таблица 33 – Тестирование производительности

Тест	Условие	Ожидаемый результат	Результат
Время отклика при бронировании	Открыть страницу бронирования, ввести данные, подтвердить	Отклик не более 2 секунд	Тест пройден
Одновременная работа 100 пользователей	Имитация нагрузки от 100 пользователей, пытающихся забронировать столик одновременно	Время отклика не более 5 секунд	Тест пройден
Оформление предзаказа с большим количеством позиций	Оформить заказ на 20+ позиций	Время отклика не более 3 секунд	Тест пройден

Второй тест – это тестирование надежности. Он позволяет проверить стабильность работы приложения при длительном использовании. План тестирования представлен в таблице 34.

Таблица 34 – Тестирование надежности

Тест	Условие	Ожидаемый результат	Результат
Непрерывная работа в течение 24 часов	Приложение активно в течение 24 часов без перезапуска	Отсутствие утечек памяти, и ошибок	Тест пройден
Попытка повторного бронирования на уже занятое время	Забронировать столик на занятое время	Выводится сообщение об ошибке	Тест пройден

Продолжение таблицы 34

Тест	Условие	Ожидаемый результат	Результат
Отключение сервера на 5 минут	Попробовать оформить бронирование во время отключения	Сообщение об ошибке подключения	Тест пройден

Третьим тестом является тестирование совместимости. Данный тест проверяет корректность работы на разных устройствах и в разных браузерах. План тестирования представлен в таблице 35.

Таблица 35 – Тестирование совместимости

Тест	Условие	Ожидаемый результат	Результат
Работа в браузерах (Chrome, Firefox, Edge, Yandex)	Открыть и выполнить бронирование в каждом браузере	Корректное отображение, отсутствие ошибок	Тест пройден
Адаптация под мобильные устройства	Открыть приложение на смартфоне и планшете	Корректное отображение интерфейса	Тест пройден
Разное разрешение экрана (1920x1080, 1366x768, 720x1280)	Открыть приложение на устройствах с разным разрешением	Интерфейс адаптируется корректно	Тест пройден

Четвертый тест позволяет оценить удобство интерфейса и простоту использования. План тестирования представлен в таблице 36.

Таблица 36 – Тестирование удобства использования

Тест	Условие	Ожидаемый результат	Результат
Интуитивность интерфейса	Новый пользователь выполняет бронирование без подсказок	Выполнение задачи менее чем за 1 минуту	Тест пройден
Подсказки при ошибках ввода	Ввод некорректных данных в поле бронирования	Появление понятного сообщения об ошибке	Тест пройден
Удобство навигации	Переходы между страницами	Легкий доступ ко всем страницам	Тест пройден

Последним тестом является тест на восстанавливаемость. Его целью является проверка возможности восстановления работы после сбоя. План тестирования представлен в таблице 37.

Таблица 37 – Тестирование восстанавливаемости

Тест	Условие	Ожидаемый результат	Результат
Восстановление после аварийного завершения работы сервера	Отключить сервер, перезапустить и проверить состояние бронирования	Данные о бронированиях сохраняются	Тест пройден
Восстановление сеанса после перезагрузки страницы	Оформление заказа, затем перезагрузка страницы	Данные бронирования сохраняются в localStorage	Тест пройден

Результаты нефункционального тестирования показывают, что приложение стабильно работает под нагрузкой и в течение длительного времени работы. Удобство интерфейса на высоком уровне. Приложение успешно восстанавливает работу после сбоев.

4.4 Оценка качества программного кода

4.4.1 Метрика Холстеда

Для расчета метрик Холстеда рассматривается функция «create_reservation». Данная функция реализует функционал бронирования стола.

Для вычисления данной метрики требуется посчитать число вхождения уникальных операторов (N1) и операндов (N2), а также общее количество операторов (n1) и операндов (n2).

Операторы функции «create_reservation» и число их вхождений приведены в таблице 38.

Таблица 38 – Операторы функции «create_reservation»

Операторы	Число вхождений
=	12
<	2
>	2

Продолжение таблицы 38

Операторы	Число вхождений
If	7
+	2
Select	2
raise	7
where	2
.	20
and	2
session.exec	2
{}	6
[]	1
()	18
:	7
,	15
else	1
first	1
all	1
n1 = 19	N1 = 110

Операнды функции «create_reservation» и число их вхождений приведены в таблице 39.

Таблица 39 – Операнды функции «create_reservation»

Операнды	Число вхождений
reservation_data	8
user_id	1
session	4
today	2
max_date	2
reservation_date	4
reservation_time	2
duration	3
CLOSING_HOUR	1
current_hour	1
now	1
guests_count	1
Table	3
table_id	2
Table	1
Reservation	1
MAX_DAYS_ADVANCE	1
n2 = 17	N2 = 38

Данные, полученные в ходе обработки программного кода функции «create_reservation» были использованы для расчета характеристик программы по формулам из метрики Холстеда. Результаты вычислений представлены в

таблице 40.

Таблица 40 – Результаты вычислений по метрике Холстеда

Значение	Результат
Длина реализации программы	147
Значение	Результат
Словарь программы	36
Длина программы	148
Объем программы	765.14
Потенциальный объем программы	186.1
Теоретический уровень программы	0.243
Сложность программы	4,12
Работа программирования	3151.77

Все полученные показатели программы находятся в пределах нормы и удовлетворяют требованиям по метрикам программного кода. Длина реализации меньше длины программы, следовательно, несовершенства в программе отсутствуют. Теоретический уровень программы весьма низкий, так как потенциальный объем программы в значительной степени меньше ее реального объема. Возможности языка программирования использованы на достаточном уровне.

4.4.2 Метрика Чепина

Расчет метрики Чепина будет производиться над компонентом, отвечающим за анализ продуктивности пользователя пр.

Для расчета метрики Чепина берется все множество переменных, составляющих список ввода-вывода, а затем разбивается на 4 функциональные группы:

- вводимые переменные для расчетов и для обеспечения вывода (Р);
- модифицируемые внутри программы переменные (М);
- переменные, участвующие в управлении работой программного

модуля (С);

- не используемые в программе переменные (Т).

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе. Метрика Чепина:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T,$$

где a_1 , a_2 , a_3 , a_4 – весовые коэффициенты.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, равный 3, имеет функциональная группа С, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a_1=1$, $a_2=2$, $a_4=0.5$. Весовой коэффициент группы Т не равен 0, поскольку «паразитные» переменные не увеличивают сложность потока данных программы, но иногда затрудняют ее понимание.

В модуле, отвечающем за анализ продуктивности пользователя, было выделено 4 вводимых переменных, 6 модифицируемых внутри, две управляющих переменных и не оказалось «паразитной» переменной. Таким образом, формула подсчета метрики Чепина для данного модуля:

$$Q = 4 + 2 * 6 + 3 * 2 + 0.5 * 0 = 22.$$

Данный результат вычисления метрики Чепина указывает на то, что рассматриваемый компонент имеет относительно небольшой размер и невысокую сложность исходного кода.

Таким образом, было проведено тестирование программных средств управления гостевым сервисом в ресторане, в рамках которого приложение подверглось функциональному и нефункциональному тестированию, а качество кода оценено по метрикам Холстеда и Чепина. Тестирование «белым ящиком» показало, что функциональные компоненты приложения работают стабильно и ошибок при обработке бронирований не возникает. Нефункциональное тестирование показало, что приложение работает стабильно, справляется с

нагрузкой, имеет интуитивно понятный интерфейс, совместимо с различными браузерами и устройствами и является удобным и безопасным для пользователей.

Код разрабатываемых программных средств управления гостевым сервисом в ресторане имеет умеренную сложность, поддерживаемость и читаемость. Все это показала метрика Холстеда. Метрика Чепина указала на умеренные трудозатраты на понимание и развитие кода, что также является положительным моментом для дальнейшего улучшения и работы с программными средствами управления гостевым сервисом в ресторане.

Все проведенные тесты и метрики показали, что приложение работает стабильно и эффективно, не имеет критических ошибок и готово к использованию в реальных условиях.

Заключение

В рамках выпускной квалификационной работы были разработаны и спроектированы программные средства для управления гостевым сервисом в ресторане.

На стадии анализа требований проведен анализ предметной области, сравнительный анализ веб-сайтов популярных ресторанов, выявлены нефункциональные и функциональные требования для разработки нового веб-сайта ресторана. Кроме того, произведено планирование разработки и оценка бюджета программных средств. Затраты на проект составили 88900 рублей. Начало работ 8 апреля, а окончание работ по проекту запланировано на 4 июня текущего года.

На стадии проектирования разработана архитектура программного обеспечения, выявлены требования к интерфейсу и разработаны концептуальная и логическая модели базы данных.

Разработка программных средств для управления гостевым сервисом в ресторане включала в себя реализацию базы данных в PostgreSQL и разделенного монолита, где backend и frontend разделены, но взаимодействуют как единая система. Серверная часть разработана на языке Python в среде PyChar, а клиентская – в среде разработки Visual Studio Code на языке программирования JavaScript с использованием фреймворка Next.js и библиотеки React. Взаимодействие между ними осуществляется с помощью использования Rest API.

Также был проведен контроль качества разработанного веб-приложения с помощью функционального и нефункционального тестирования. Выполнен подсчет метрик Холстеда и Чепина. Функциональное тестирование проводилось методом «белого ящика». В результате тестирования выявлено соответствие функциональным, нефункциональным требованиям и корректность работы программы, подсчет метрик показал достаточный уровень разработанного приложения.

Таким образом, в результате выполнения выпускной квалификационной работы разработаны программные средства для управления гостевым сервисом в ресторане, позволяющие бронировать стол, ориентируясь на схему зала ресторана, оформлять предзаказ к брони, выбирая блюда из меню, представленного на сайте ресторана. Кроме того, веб-приложение разработано как для пользователя, так и для администратора.

Программное обеспечение отвечает требованиям качества, удобства и надежности в управлении гостевым сервисом в ресторане. Разработанные программные средства соответствуют требованиям современного ресторанного бизнеса и способны привлечь внимание пользователей благодаря оригинальному представлению сервисных услуг заведения.

Список использованных источников

1. Васильченко Р. А. Организация обслуживания в организациях общественного питания по выполнению практических работ. // «Издательский дом «Дело» РАНХиГС» 2019. С. 50.
2. Эгертон-Томас К. Ресторанный бизнес. Как открыть и успешно управлять рестораном. //«РосКонсульт» 2015. С. 25.
3. Сравнительный анализ: Что это такое и как его проводить // HR-Portal [Электронный ресурс]. URL: <https://hr-portal.ru/story/sravnitelnyy-analiz-cto-eto-takoe-i-kak-ego-provodit> (дата обращения: 20.04.2025).
4. Росс В. С. Создание сайтов: HTML, CSS, PHP, MySQL. Учебное пособие, ч. 1 // «МГДД(Ю)Т» М 2010. С. 86.
5. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/566218/> (дата обращения: 22.04.2025).
6. Стоунз, Р. Основы PostgreSQL. // «Символ-Плюс» 2002. С. 250 – 285.
7. Буйначев, С. К. Основы программирования на языке Python. Учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 91 с.
8. Иан Милл, Эйдан Хобсон Сейерс Docker на практике / пер. с англ. Д.А. Беликов. – М.: «ДМК Пресс» 2020. С. 256 – 278.
9. REST API // Skillfactory Media. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/rest-api/> (дата обращения: 26.04.2025).
10. Draw.io // Skillfactory Media. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/draw-io/> (дата обращения: 26.04.2025).
11. Зубрицкий, А. А. Эффективное управление проектами с использованием Microsoft Project 2016 / А. А. Зубрицкий. – М.: Наука, 2016. – 123 с.

12. Система управления базами данных: что это такое и зачем она нужна // Skillbox Media [Электронный ресурс]. URL: <https://skillbox.ru/media/code/sistema-upravleniya-bazami-dannykh-chto-eto-takoe-i-zachem-ona-nuzhna/> (дата обращения: 03.05.2025).
13. Рудь В. Концептуальное логическое и физическое моделирование данных // Информационный портал компании «Марк Аврелий» [Электронный ресурс]. URL: https://marcus-aurelius.ru/articles/Data_modeling.html (дата обращения 05.05.2025).
14. Фримен, А. Pro React 16: разрабатывайте современные приложения / А. Фримен, Э. Гартшорн. – СПб. 2019. – 450 с.
15. Физические модели баз данных // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/1001/297/lecture/7415> (дата обращения: 20.05.2025)
16. Рамакришнан, Р. Базы данных от простого к сложному / Р. Рамакришнан, Й. Герк. – М.: Вильямс, 2019. – 672 с.
17. Ричардс Марк, Форд Нил. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы. – СПб. Питер, 2023. – 448 с.
18. Моуэт Э. Использование Docker / пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. – М. // «ДМК Пресс», 2017. – 354 с.: ил.
19. Бэнкс, А. React и Redux: Создание приложений нового поколения на JavaScript / А. Бэнкс, Е. Порселло. – М.: Питер, 2020. – 320 с.
20. ТОП-10 платформ для разработки программного обеспечения // otzyvmarketing. [Электронный ресурс]. URL: <https://otzyvmarketing.ru/articles/platformy-dlya-razrabotki-programmnogo-bespecheniya/> (дата обращения: 26.05.2025).
21. Тестирование и 7 основных этапов его проведения // Neiros [Электронный ресурс]. URL: <https://neiros.ru/blog/code/testirovanie-i-7-osnovnykh-etapov-ego-provedeniya/> (дата обращения: 29.05.2025)
22. Функциональное тестирование: что это, этапы, виды и инструменты использования // Tproger [Электронный ресурс]. URL: <https://tproger.ru/articles/f>

unkcionalnoe-testirovanie-chto-eto-etapy-vidy-i-instrumenty-ispolzovaniya (дата обращения: 30.05.2025).

23. Крауч, Р. Тестирование ПО. Основы, методы и термины / Р. Крауч, В.Д. Влиет. – СПб.: Питер, 2003. – 256 с.

24. Программный код и его метрики // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 01.06.2025).

25. Метрики как средства управления качеством // OSP [Электронный ресурс]. URL: <https://www.osp.ru/os/2009/08/10748698> (дата обращения: 01.06.2025).

Приложение А
(справочное)
Глоссарий

Программные средства (программное обеспечение, ПО) – это набор программ, которые позволяют устройству выполнять определенные задачи.

БД (База данных) – это имеющая название совокупность данных, которая отражает состояние объектов и их отношений в рассматриваемой предметной области.

СУБД (Система управления базами данных) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

PostgreSQL – это открытая и мощная реляционная система управления данными, основанная на объектно-ориентированной парадигме, обеспечивающая высокую производительность, надежность и масштабируемость.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

REST (Representational State Transfer) – это архитектурная концепция, определяющая набор ограничений для построения масштабируемых распределенных систем. Основными принципами REST является ресурсно-ориентированная архитектура, использование стандартных HTTP-методов для манипуляции ресурсами, передача данных без сохранения состояния на сервере, унифицированный интерфейс доступа.

API (Application Programming Interface) – набор готовых классов, функций, процедур, констант и структур, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

Web-браузер – прикладное программное обеспечение для поиска и просмотра web-страниц, а также управления web-приложениями;

Web-приложение – это клиент-серверное программное решение, доступ к которому осуществляется через сетевой браузер посредством передачи данных по протоколу HTTP, размещается на удаленном сервере

JSON – JavaScript Object Notation, легковесный текстовый формат обмена данными, полностью независимый от языка реализации.

Фреймворк – программная платформа, задающая архитектурные принципы и общие правила разработки приложений в определенной предметной области.

MVC – это архитектурный паттерн, предписывающий разделение программного кода на три взаимосвязанных компонента: модель данных, представление для визуализации информации и контроллер для обработки пользовательских запросов.

Приложение Б
(рекомендуемое)

Скрипты для создания объектов базы данных на PostgreSQL

```

CREATE TABLE Table_types (
    Id SERIAL PRIMARY KEY,
    Name VARCHAR(255),
    Default_width INTEGER,
    Default_height INTEGER,
    Default_max_guests INTEGER
);
CREATE TABLE Rooms (
    Id SERIAL PRIMARY KEY,
    Name VARCHAR(255),
    Description VARCHAR(255)
);
CREATE TABLE Tables (
    Id SERIAL PRIMARY KEY,
    Type VARCHAR(255),
    Max_guests INTEGER,
    X INTEGER,
    Y INTEGER,
    Width INTEGER,
    Height INTEGER,
    Rotation INTEGER,
    Roomsid INTEGER REFERENCES Rooms(Id),
    Table_typesId INTEGER REFERENCES Table_types(Id)
);
CREATE TABLE Static_items (
    Id SERIAL PRIMARY KEY,
    Type VARCHAR(255),
    X INTEGER,
    Y INTEGER,
    Width INTEGER,
    Height INTEGER,
    Rotation INTEGER,
    Roomsid INTEGER REFERENCES Rooms(Id)
);
CREATE TABLE Walls (
    Id SERIAL PRIMARY KEY,
    X INTEGER,
    Y INTEGER,
    Length INTEGER,
    Rotation INTEGER,
    Roomsid INTEGER REFERENCES Rooms(Id)
);
CREATE TABLE Users (
    Id SERIAL PRIMARY KEY,
    Email VARCHAR(255) UNIQUE NOT NULL,
    Password_hash VARCHAR(255) NOT NULL,
    First_name VARCHAR(255),
    Last_name VARCHAR(255),
    Phone VARCHAR(255),
    Role VARCHAR(255)
);
CREATE TABLE Categories (
    Id SERIAL PRIMARY KEY,

```

```

        Name VARCHAR(255)
    );

-- Таблица Menu_items
CREATE TABLE Menu_items (
    Id SERIAL PRIMARY KEY,
    Name VARCHAR(255),
    Description VARCHAR(255),
    Price DOUBLE PRECISION,
    Image_url VARCHAR(255),
    CategoriesId INTEGER REFERENCES Categories(Id)
);

-- Таблица Reservations
CREATE TABLE Reservations (
    Id SERIAL PRIMARY KEY,
    Reservation_date DATE,
    Reservation_time TIME,
    Guests_count INTEGER,
    First_name VARCHAR(255),
    Last_name VARCHAR(255),
    Phone VARCHAR(255),
    Email VARCHAR(255),
    UsersId INTEGER REFERENCES Users(Id),
    TablesId INTEGER REFERENCES Tables(Id)
);

-- Таблица Order_items
CREATE TABLE Order_items (
    Id SERIAL PRIMARY KEY,
    Quantity INTEGER,
    ReservationsId INTEGER REFERENCES Reservations(Id),
    Menu_itemsId INTEGER REFERENCES Menu_items(Id)
);

```

Приложение В

(обязательное)

Листинг серверной части приложения

Листинг В.1 – Файл services.py

```

from datetime import timedelta
from sqlmodel import Session, select
from fastapi import HTTPException, status
from db.models import User
from schemas.user import UserRegister, UserLogin, Token
from utils.security import get_password_hash, verify_password,
create_access_token, ACCESS_TOKEN_EXPIRE_MINUTES

def register_user(user_data: UserRegister, session: Session):
    # Check if user with this email already exists
    db_user = session.exec(select(User).where(User.email ==
user_data.email)).first()
    if db_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Этот email уже зарегистрирован"
        )

    # Create new user
    hashed_password = get_password_hash(user_data.password)
    db_user = User(
        email=user_data.email,
        password_hash=hashed_password,
        first_name=user_data.first_name,
        last_name=user_data.last_name,
        phone=user_data.phone
    )

    session.add(db_user)
    session.commit()
    session.refresh(db_user)

    # Generate access token
    access_token_expires =
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": db_user.email, "role": db_user.role},
        expires_delta=access_token_expires
    )

    return {"access_token": access_token, "token_type": "bearer"}

def login_user(user_data: UserLogin, session: Session):
    print(f"Login attempt for: {user_data.email}")

    # Verify user exists
    db_user = session.exec(select(User).where(User.email ==
user_data.email)).first()
    if not db_user:

```

```

        print(f"User not found: {user_data.email}")
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверный email или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )

    print(f"User found: {db_user.email}, role: {db_user.role}")

    # Verify password
    password_valid = verify_password(user_data.password,
db_user.password_hash)
    print(f"Password valid: {password_valid}")

    if not password_valid:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверный email или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )

    # Generate access token
    access_token_expires =
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": db_user.email, "role": db_user.role},
        expires_delta=access_token_expires
    )

    print(f"Login successful for: {db_user.email}")
    return {"access_token": access_token, "token_type": "bearer"}

def get_current_user(user):
    return user

```

Листинг В.2 – Файл models.py

```

from datetime import date, time, datetime
from typing import List, Optional
from uuid import UUID, uuid4
from sqlmodel import Field, SQLModel, Relationship

class Room(SQLModel, table=True):
    __tablename__ = "rooms"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    name: str = Field(index=True)
    description: Optional[str] = None
    created_at: datetime = Field(default_factory=datetime.now)
    updated_at: datetime = Field(default_factory=datetime.now)

    tables: List["Table"] = Relationship(back_populates="room")

```

```

    static_items: List["StaticItem"] =
Relationship(back_populates="room")
    walls: List["Wall"] = Relationship(back_populates="room")

class TableType(SQLModel, table=True):
    __tablename__ = "table_types"

    id: int = Field(primary_key=True)
    name: str = Field(unique=True, index=True) # "circular",
"rectangular", "vip"
    display_name: str # "Круглый столик", "Прямоугольный столик",
"VIP-столик"
    default_width: int
    default_height: int
    default_max_guests: int
    color_code: Optional[str] = None # For frontend styling
    created_at: datetime = Field(default_factory=datetime.now)
    updated_at: datetime = Field(default_factory=datetime.now)

    tables: List["Table"] =
Relationship(back_populates="table_type")

class User(SQLModel, table=True):
    __tablename__ = "users"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    email: str = Field(unique=True, index=True)
    password_hash: str
    first_name: Optional[str] = None
    last_name: Optional[str] = None
    phone: Optional[str] = None
    role: str = Field(default="user")

    reservations: List["Reservation"] =
Relationship(back_populates="user")

class Table(SQLModel, table=True):
    __tablename__ = "tables"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    type_id: int = Field(foreign_key="table_types.id")
    table_number: int # Sequential number for each table type (1
to n)
    max_guests: int
    x: int
    y: int
    rotation: int = Field(default=0)
    width: Optional[int] = None
    height: Optional[int] = None
    room_id: UUID = Field(foreign_key="rooms.id")

```

```

    is_active: bool = Field(default=True) # Instead of deleting,
tables can be marked inactive

    reservations: List["Reservation"] =
Relationship(back_populates="table")
    table_type: TableType = Relationship(back_populates="tables")
    room: Room = Relationship(back_populates="tables")

class StaticItem(SQLModel, table=True):
    __tablename__ = "static_items"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    type: str
    x: int
    y: int
    rotation: int = Field(default=0)
    room_id: UUID = Field(foreign_key="rooms.id")

    room: Room = Relationship(back_populates="static_items")

class Wall(SQLModel, table=True):
    __tablename__ = "walls"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    x: int
    y: int
    rotation: int = Field(default=0)
    length: int
    room_id: UUID = Field(foreign_key="rooms.id")

    room: Room = Relationship(back_populates="walls")

class Reservation(SQLModel, table=True):
    __tablename__ = "reservations"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    user_id: UUID = Field(foreign_key="users.id")
    table_id: UUID = Field(foreign_key="tables.id")
    reservation_date: date
    reservation_time: time
    duration: int = Field(default=1) # Продолжительность в часах,
по умолчанию 1 час
    guests_count: int
    first_name: str
    last_name: str
    phone: str
    status: str = Field(default="pending") # Possible values:
"pending", "confirmed", "cancelled"

    user: User = Relationship(back_populates="reservations")
    table: Table = Relationship(back_populates="reservations")

```

```

    ordered_items: List["OrderItem"] =
Relationship(back_populates="reservation")

class Category(SQLModel, table=True):
    __tablename__ = "categories"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    name: str

    menu_items: List["MenuItem"] =
Relationship(back_populates="category")

class MenuItem(SQLModel, table=True):
    __tablename__ = "menu_items"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    name: str
    description: str
    price: float
    image_url: Optional[str] = None
    category_id: UUID = Field(foreign_key="categories.id")

    category: Category = Relationship(back_populates="menu_items")
    order_items: List["OrderItem"] =
Relationship(back_populates="menu_item")

class OrderItem(SQLModel, table=True):
    __tablename__ = "order_items"

    id: UUID = Field(default_factory=uuid4, primary_key=True)
    reservation_id: UUID = Field(foreign_key="reservations.id")
    menu_item_id: UUID = Field(foreign_key="menu_items.id")
    quantity: int

    reservation: Reservation =
Relationship(back_populates="ordered_items")
    menu_item: MenuItem =
Relationship(back_populates="order_items")

```

Листинг В.3 – Файл create_default_room.py

```

#!/usr/bin/env python3
"""
Script to initialize a default room in the database if none
exists.
"""

import os
import sys
from sqlmodel import Session, select
from dotenv import load_dotenv

# Add the current directory to Python path

```

```

sys.path.append(os.path.dirname(os.path.abspath(os.path.dirname(__
file__))))

# Load environment variables
load_dotenv()

from db.database import engine
from db.models import Room

def create_default_room():
    """Create a default room if no rooms exist in the database"""
    print("=== Checking for default room ===")

    try:
        with Session(engine) as session:
            # Check if any rooms exist
            existing_room = session.exec(select(Room)).first()

            if existing_room:
                print(f"Room already exists: {existing_room.name}
(ID: {existing_room.id})")
                return existing_room

            print("No existing rooms found. Creating default
room...")

            # Create default room
            default_room = Room(
                name="Основной зал",
                description="Основной зал ресторана"
            )

            session.add(default_room)
            session.commit()
            session.refresh(default_room)

            print(f"Default room created successfully:
{default_room.name} (ID: {default_room.id})")
            return default_room

    except Exception as e:
        print(f"Error during room creation: {e}")
        print(f"Error type: {type(e)}")
        raise

if __name__ == "__main__":
    create_default_room()

```

Листинг В.4 – Файл database.py

```

import os
from sqlmodel import create_engine, Session, SQLModel

# Get the DATABASE_URL from environment variable with a fallback

```

```

DATABASE_URL = os.getenv("DATABASE_URL",
"postgresql://postgres:postgres@db:5432/restaurant")

# Create SQLAlchemy engine
engine = create_engine(DATABASE_URL)

# Dependency to get DB session
def get_session():
    with Session(engine) as session:
        yield session

# Create all tables
def create_db_and_tables():
    SQLAlchemyModel.metadata.create_all(engine)

```

Листинг В.5 – Файл routes.py

```

from typing import Optional, List
from uuid import UUID
from fastapi import APIRouter, Depends, Query, HTTPException,
status
from sqlalchemy import Session, select
from db.database import get_session
from utils.security import get_current_user, get_current_admin
from schemas.layout import (
    Layout, EnhancedLayout, LayoutUpdate, TableCreate, TableRead,
    TableFullRead,
    StaticItemCreate, StaticItemRead, WallCreate, WallRead,
    TableTypeCreate, TableTypeRead, RoomCreate, RoomRead
)
from .services import get_layout, save_layout, add_table,
add_static_item, add_wall, clear_layout
from db.models import User, TableType, Room
from datetime import datetime
from db.create_default_room import create_default_room
router = APIRouter(tags=["layout"])
# Room Management Endpoints
@router.get("/rooms", response_model=List[RoomRead])
def get_rooms(session: Session = Depends(get_session)):
    rooms = session.exec(select(Room)).all()
    if not rooms:
        # Create a default room if none exists
        default_room = create_default_room()
        if default_room:
            rooms = [default_room]
    return rooms

@router.get("/rooms/{room_id}", response_model=RoomRead)
def get_room(room_id: UUID, session: Session =
Depends(get_session)):
    room = session.get(Room, room_id)
    if not room:
        # Try to get any room first
        room = session.exec(select(Room)).first()

```

```

        if not room:
            # Create a default room if none exists
            room = create_default_room()
            if not room:
                raise
    HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Room
not found and couldn't create default room")
    return room

@router.post("/rooms", response_model=RoomRead)
def create_room(
    room: RoomCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    existing = session.exec(select(Room).where(Room.name ==
room.name)).first()
    if existing:
        raise
    HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=f"Room with name '{room.name}' already exists")
    new_room = Room(**room.dict())
    session.add(new_room)
    session.commit()
    session.refresh(new_room)
    return new_room

@router.put("/rooms/{room_id}", response_model=RoomRead)
def update_room(
    room_id: UUID,
    room: RoomCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    db_room = session.get(Room, room_id)
    if not db_room:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Room not found")
    if db_room.name != room.name:
        existing = session.exec(select(Room).where(Room.name ==
room.name)).first()
        if existing:
            raise
    HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=f"Room with name '{room.name}' already exists")
    db_room.name = room.name
    db_room.description = room.description
    db_room.updated_at = datetime.now()
    session.add(db_room)
    session.commit()
    session.refresh(db_room)
    return db_room

```

```

@router.get("/", response_model=Layout)
def get_restaurant_layout(room_id: Optional[UUID] = None, session:
Session = Depends(get_session)):
    return get_layout(session, room_id)

@router.get("/enhanced", response_model=EnhancedLayout)
def get_enhanced_restaurant_layout(room_id: Optional[UUID] = None,
session: Session = Depends(get_session)):
    return get_layout(session, room_id, include_types=True)

@router.post("/save", response_model=Layout)
def save_restaurant_layout(
    layout: LayoutUpdate,
    room_id: Optional[UUID] = None,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    print(f"Saving layout with {len(layout.tables)} tables and
{len(layout.static_items)} static items and {len(layout.walls)}
walls")
    print(f"Current admin user: {current_user.email}")
    return save_layout(layout, session, room_id)

@router.post("/tables", response_model=TableRead)
def add_restaurant_table(
    table: TableCreate,
    room_id: Optional[UUID] = None,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    return add_table(table, session, room_id)

@router.post("/static-items", response_model=StaticItemRead)
def add_restaurant_static_item(
    item: StaticItemCreate,
    room_id: Optional[UUID] = None,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    return add_static_item(item, session, room_id)

@router.post("/walls", response_model=WallRead)
def add_restaurant_wall(
    wall: WallCreate,
    room_id: Optional[UUID] = None,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    return add_wall(wall, session, room_id)

@router.post("/clear", response_model=Layout)
def clear_restaurant_layout(
    room_id: Optional[UUID] = None,

```

```

        session: Session = Depends(get_session),
        current_user: User = Depends(get_current_admin)
    ):
        return clear_layout(session, room_id)

@router.get("/table-types", response_model=List[TableTypeRead])
def get_table_types(session: Session = Depends(get_session)):
    return session.exec(select(TableType)).all()

@router.get("/table-types/{type_id}",
response_model=TableTypeRead)
def get_table_type(type_id: UUID, session: Session =
Depends(get_session)):
    table_type = session.get(TableType, type_id)
    if not table_type:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Table type not found")
    return table_type

@router.post("/table-types", response_model=TableTypeRead)
def create_table_type(
    table_type: TableTypeCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    existing = session.exec(select(TableType).where(TableType.name
== table_type.name)).first()
    if existing:
        raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=f"Table type with name '{table_type.name}' already exists")
    new_type = TableType(**table_type.dict())
    session.add(new_type)
    session.commit()
    session.refresh(new_type)
    return new_type

@router.put("/table-types/{type_id}",
response_model=TableTypeRead)
def update_table_type(
    type_id: UUID,
    table_type: TableTypeCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_admin)
):
    db_type = session.get(TableType, type_id)
    if not db_type:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Table type not found")
    if db_type.name != table_type.name:
        existing =
session.exec(select(TableType).where(TableType.name ==
table_type.name)).first()

```

```

        if existing:
            raise
    HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=f"Table type with name '{table_type.name}' already exists")
    for key, value in table_type.dict().items():
        setattr(db_type, key, value)
    session.add(db_type)
    session.commit()
    session.refresh(db_type)
    return db_type

```

Листинг В.6 – Файл menu.py

```

from typing import List, Optional, Dict
from uuid import UUID
from pydantic import BaseModel

class CategoryBase(BaseModel):
    name: str

class CategoryCreate(CategoryBase):
    pass

class CategoryRead(CategoryBase):
    id: UUID

    class Config:
        orm_mode = True

class MenuItemBase(BaseModel):
    name: str
    description: str
    price: float
    image_url: Optional[str] = None

class MenuItemCreate(MenuItemBase):
    category_id: UUID

class MenuItemRead(MenuItemBase):
    id: UUID
    category_id: UUID

    class Config:
        orm_mode = True

class MenuItemWithCategory(MenuItemRead):
    category: CategoryRead

```

```

class Menu(BaseModel):
    categories: List[CategoryRead]
    items: List[MenuItemRead]

    class Config:
        orm_mode = True

class OrderItemCreate(BaseModel):
    menu_item_id: UUID
    quantity: int = 1

class OrderItemRead(BaseModel):
    id: UUID
    menu_item_id: UUID
    reservation_id: UUID
    quantity: int
    menu_item: Optional[MenuItemRead] = None

    class Config:
        orm_mode = True

class OrderStats(BaseModel):
    total_orders: int
    items_sold: Dict[str, int]
    items_by_category: Dict[str, int]

    total_revenue: float

```

Листинг В.7 – Файл reservation.py

```

from typing import List, Optional,
from datetime import date, time
from typing import List, Optional, Dict
from uuid import UUID
from pydantic import BaseModel

class ReservationBase(BaseModel):
    reservation_date: date
    reservation_time: time
    duration: Optional[int] = 1 # Продолжительность в часах, по
    умолчанию 1 час
    guests_count: int
    first_name: str
    last_name: str
    phone: str
    status: Optional[str] = "pending"

class ReservationCreate(ReservationBase):
    table_id: UUID

```

```

class TableInfo(BaseModel):
    id: UUID
    type_id: int
    table_number: int
    width: Optional[int] = None
    height: Optional[int] = None
    table_type: Optional[dict] = None # Will store type info like
name, display_name, etc.

    class Config:
        orm_mode = True

class ReservationRead(ReservationBase):
    id: UUID
    user_id: UUID
    table_id: UUID
    class Config:
        orm_mode = True
class ReservationEnhanced(ReservationRead):
    table: TableInfo

    class Config:
        orm_mode = True
class AvailabilityQuery(BaseModel):
    query_date: date
    query_time: Optional[time] = None

class TableAvailability(BaseModel):
    table_id: UUID
    type_id: int
    table_number: int
    available: bool
    available_times: Optional[List[time]] = None
class ReservationStats(BaseModel):
    total_reservations: int
    reservations_by_date: Dict[str, int]
    reservations_by_table: Dict[str, int]
    average_guests: float
    average_duration: float

class ReservationStatusUpdate(BaseModel):
    status: str

```

Листинг В.8 – Файл user.py

```

from typing import Optional
from uuid import UUID
from pydantic import BaseModel, EmailStr

class UserBase(BaseModel):
    email: EmailStr
    first_name: Optional[str] = None
    last_name: Optional[str] = None
    phone: Optional[str] = None

```

```

class UserRegister(UserBase):
    password: str

class UserLogin(BaseModel):
    email: EmailStr
    password: str

class UserRead(UserBase):
    id: UUID
    role: str
    class Config:
        orm_mode = True

class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    email: Optional[str] = None
    role: Optional[str] = None

```

Листинг В.9 – Файл layout.py

```

from typing import List, Optional
from uuid import UUID
from pydantic import BaseModel
from datetime import datetime

class RoomBase(BaseModel):
    name: str
    description: Optional[str] = None

class RoomCreate(RoomBase):
    pass

class RoomRead(RoomBase):
    id: UUID
    created_at: datetime
    updated_at: datetime

    class Config:
        orm_mode = True

class TableTypeBase(BaseModel):
    name: str
    display_name: str
    default_width: int
    default_height: int
    default_max_guests: int

```

```

        color_code: Optional[str] = None

class TableTypeCreate(TableTypeBase):
    pass

class TableTypeRead(TableTypeBase):
    id: int

    class Config:
        orm_mode = True

class TableBase(BaseModel):
    type_id: int
    table_number: int
    max_guests: int
    x: int
    y: int
    rotation: int = 0
    width: Optional[int] = None
    height: Optional[int] = None

class TableCreate(TableBase):
    id: Optional[UUID] = None
class TableRead(TableBase):
    id: UUID
    room_id: UUID
    is_active: bool = True

    class Config:
        orm_mode = True

class TableFullRead(TableRead):
    table_type: Optional[TableTypeRead] = None

    class Config:
        orm_mode = True

class StaticItemBase(BaseModel):
    type: str
    x: int
    y: int
    rotation: int = 0

class StaticItemCreate(StaticItemBase):
    id: Optional[UUID] = None

class StaticItemRead(StaticItemBase):
    id: UUID
    room_id: UUID

    class Config:
        orm_mode = True

```

```

class WallBase(BaseModel):
    x: int
    y: int
    rotation: int = 0
    length: int

class WallCreate(WallBase):
    id: Optional[UUID] = None

class WallRead(WallBase):
    id: UUID
    room_id: UUID

    class Config:
        orm_mode = True

class LayoutItem(BaseModel):
    id: UUID
    type_id: int
    table_number: int
    x: int
    y: int
    rotation: int
    item_type: str # "table" or "static"
    max_guests: Optional[int] = None
    width: Optional[int] = None
    height: Optional[int] = None

class LayoutUpdate(BaseModel):
    tables: List[TableCreate]
    static_items: List[StaticItemCreate]
    walls: List[WallCreate] = []

class Layout(BaseModel):
    tables: List[TableRead]
    static_items: List[StaticItemRead]
    walls: List[WallRead] = []

    class Config:
        orm_mode = True

class EnhancedLayout(BaseModel):
    tables: List[TableFullRead]
    static_items: List[StaticItemRead]
    walls: List[WallRead] = []

    class Config:
        orm_mode = True

```

Листинг В.10 – Файл security.py

```

from datetime import datetime, timedelta
from typing import Optional
import os

```

```

from jose import JWTError, jwt
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlmodel import Session, select
from db.database import get_session
from db.models import User
from schemas.user import TokenData

# Configuration
SECRET_KEY = os.getenv("SECRET_KEY", "your_jwt_secret_here")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES =
int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "60"))
# Password hashing
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# OAuth2 scheme
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="auth/login/oauth")

def verify_password(plain_password, hashed_password):
    try:
        return pwd_context.verify(plain_password, hashed_password)
    except Exception as e:
        print(f"Password verification error: {e}")
        # Try direct comparison as a fallback for development
        if os.getenv("ENVIRONMENT", "development") ==
"development":
            return plain_password == "adminpassword" # Only for
development!
        return False

def get_password_hash(password):
    return pwd_context.hash(password)

def create_access_token(data: dict, expires_delta:
Optional[timedelta] = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY,
algorithm=ALGORITHM)
    return encoded_jwt

async def get_current_user(token: str = Depends(oauth2_scheme),
session: Session = Depends(get_session)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,

```

```

        detail="Не удалось проверить учетные данные",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY,
            algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        role: str = payload.get("role", "user")
        if email is None:
            raise credentials_exception
        token_data = TokenData(email=email, role=role)
    except JWTError:
        raise credentials_exception

    user = session.exec(select(User).where(User.email ==
        token_data.email)).first()
    if user is None:
        raise credentials_exception
    return user

def get_current_admin(token: str = Depends(oauth2_scheme),
    session: Session = Depends(get_session)):
    """
    Get the current admin user. This is a separate function (not
    using get_current_user)
    to avoid FastAPI issues with nested dependencies.
    """
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Не удалось проверить учетные данные",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY,
            algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        role: str = payload.get("role", "user")
        if email is None or role != "admin":
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Доступ разрешен только администраторам",
            )
        token_data = TokenData(email=email, role=role)
    except JWTError:
        raise credentials_exception

    user = session.exec(select(User).where(User.email ==
        token_data.email)).first()
    if user is None:
        raise credentials_exception
    return user

```

Листинг В.11 – Файл create_default_admin.py

```
#!/usr/bin/env python3
import os
import sys
from sqlmodel import Session, select, SQLModel
from dotenv import load_dotenv

# Add the current directory to Python path
sys.path.append(os.path.dirname(os.path.abspath(__file__)))

# Load environment variables
load_dotenv()

from db.database import engine, create_db_and_tables
from db.models import User
from utils.security import get_password_hash

def create_default_admin():
    """Create a default admin user if no admin exists"""
    # Get admin credentials from environment or use defaults
    admin_email = os.getenv("ADMIN_EMAIL", "admin@restaurant.com")
    admin_password = os.getenv("ADMIN_PASSWORD", "adminpassword")
    admin_first_name = os.getenv("ADMIN_FIRST_NAME", "Admin")
    admin_last_name = os.getenv("ADMIN_LAST_NAME", "User")

    print("=== Creating default admin user ===")
    print(f"Email: {admin_email}")
    print(f>Password length: {len(admin_password)} chars")

    try:
        # Ensure database tables exist
        print("Creating database tables...")
        create_db_and_tables()
        print("Tables created successfully")

        with Session(engine) as session:
            # Check if admin user already exists
            print("Checking for existing admin users...")
            try:
                existing_admin = session.exec(
                    select(User).where(User.role == "admin")
                ).first()

                if existing_admin:
                    print(f"Admin user already exists: {existing_admin.email}")
                    return

            print("No existing admin found. Creating new admin...")

            # Create default admin user
```

```

password_hash = get_password_hash(admin_password)
admin_user = User(
    email=admin_email,
    password_hash=password_hash,
    first_name=admin_first_name,
    last_name=admin_last_name,
    role="admin"
)

session.add(admin_user)
session.commit()
print(f"Default admin user created successfully:
{admin_email}")
print("IMPORTANT: Change the default password
immediately!")

# Verify the admin was created
verify_admin = session.exec(
    select(User).where(User.email == admin_email)
).first()

if verify_admin:
    print(f"Verified admin exists:
{verify_admin.email}, role: {verify_admin.role}")
else:
    print("WARNING: Admin verification failed!")
except Exception as e:
    print(f"Error during admin user query/creation:
{e}")

    print(f"Error type: {type(e)}")
    raise

except Exception as e:
    print(f"Error creating default admin user: {e}")
    print(f"Error type: {type(e)}")

if __name__ == "__main__":
    create_default_admin()

```

Листинг В.12 – Файл main.py

```

import os
import sys
from fastapi import FastAPI, Depends, HTTPException, Form
from fastapi.middleware.cors import CORSMiddleware
from dotenv import load_dotenv
from fastapi.security import OAuth2PasswordBearer,
OAuth2PasswordRequestForm
from fastapi.staticfiles import StaticFiles
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

```

```

# Add the current directory to Python path
sys.path.append(os.path.dirname(os.path.abspath(__file__)))

# Load environment variables
load_dotenv()

# Import database
from db.database import create_db_and_tables
from db.models import User

# Import routers - use relative imports
from auth.services import get_current_user
from layout.routes import router as layout_router
from reservations.routes import router as reservations_router
from menu.routes import router as menu_router
from uploads.routes import router as uploads_router

# Create a router for auth since we don't have routes.py
from fastapi import APIRouter, Depends, HTTPException
from schemas.user import UserRegister, UserLogin, Token, UserRead
from sqlmodel import Session, select
from db.database import get_session
from auth.services import register_user, login_user
from utils.security import get_current_user, oauth2_scheme

from db.init_table_types import init_table_types
from db.create_default_room import create_default_room

auth_router = APIRouter()

@auth_router.post("/register", response_model=Token)
def register(user_data: UserRegister, session: Session =
Depends(get_session)):
    return register_user(user_data, session)

@auth_router.post("/login", response_model=Token)
def login(user_data: UserLogin, session: Session =
Depends(get_session)):
    """Login and get access token (JSON body)."""
    return login_user(user_data, session)

@auth_router.post("/login/oauth", response_model=Token)
async def login_for_access_token(form_data:
OAuth2PasswordRequestForm = Depends(), session: Session =
Depends(get_session)):
    """
    OAuth2 compatible token login, get an access token for future
    requests.

```

This endpoint is used by the Swagger UI for authentication.

- Use your email as the username
- Paste the token from the response into the Authorize button

```

(with Bearer prefix)
    """
    user_data = UserLogin(email=form_data.username,
password=form_data.password)
    return login_user(user_data, session)

@auth_router.get("/me", response_model=UserRead)
async def get_me(token: str = Depends(oauth2_scheme), session:
Session = Depends(get_session)):
    """Get the current logged in user."""
    try:
        user = await get_current_user(token, session)
        return user
    except Exception as e:
        print(f"Error in /auth/me endpoint: {e}")
        raise HTTPException(
            status_code=401,
            detail="Необходима аутентификация",
            headers={"WWW-Authenticate": "Bearer"},
        )

app = FastAPI(
    title="Restaurant Reservation API",
    description="API for restaurant table booking and management",
    version="1.0.0",
    openapi_tags=[
        {"name": "Authentication", "description": "Authentication
operations"},
        {"name": "Layout", "description": "Layout management
operations"},
        {"name": "Reservations", "description": "Reservation
operations"},
        {"name": "Menu", "description": "Menu management
operations"},
        {"name": "Uploads", "description": "File upload
operations"}
    ],
    swagger_ui_parameters={"persistAuthorization": True}
)

# Configure CORS - Fix to allow any origin temporarily
allowed_origins = ["http://localhost:3000",
"http://frontend:3000", "http://0.0.0.0:3000", "*"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=allowed_origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    expose_headers=["*"],
    max_age=600,
)

```

```

# Include routers
app.include_router(auth_router, prefix="/auth",
tags=["Authentication"])
app.include_router(layout_router, prefix="/layout",
tags=["Layout"])
app.include_router(reservations_router, prefix="/reserve",
tags=["Reservations"])
app.include_router(menu_router, prefix="/menu", tags=["Menu"])
app.include_router(uploads_router, prefix="/uploads",
tags=["Uploads"])

# Mount the uploads directory as a static files location
app.mount("/uploads",
StaticFiles(directory=os.path.join("frontend", "public",
"uploads")), name="uploads")

@app.get("/", tags=["Root"])
def read_root():
    return {"message": "Welcome to Restaurant Reservation API"}

# Initialize the database at startup
@app.on_event("startup")
def on_startup():
    logger.info("Initializing database...")
    create_db_and_tables()
    init_table_types()
    create_default_room()

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000,
reload=True)

```

Приложение Г

(обязательное)

Листинг клиентской части приложения

Листинг Г.1 – Файл Editor.tsx

```

import React, { useState, useEffect, useRef, useCallback } from
'react';
import { DndContext, useSensor, useSensors, PointerSensor,
DragEndEvent, DragStartEvent, useDraggable } from '@dnd-kit/core';
import { layoutAPI } from '../lib/api';
import Table from './Table';
import Wall from './Wall';
import StaticItem from './StaticItem';
import { LAYOUT_COMPONENTS, renderComponentPreview,
LayoutComponent, TableComponent } from '../types/components';
import { renderTableByType, renderStaticItem } from
'./TableStyles';

interface TableItem {
  id: string;
  type_id: number; // ID типа стола (1-4)
  type: string;    // Название типа ('circular', 'vip', и т.д.)
  type_name: string; // То же, что и type, для внутреннего
использования
  table_number: number; // Порядковый номер стола
  max_guests: number;
  x: number;
  y: number;
  rotation: number;
  width?: number;
  height?: number;
}

interface StaticItem {
  id: string;
  type: string;
  x: number;
  y: number;
  rotation: number;
  width?: number;
  height?: number;
}

interface WallItem {
  id: string;
  x: number;
  y: number;
  rotation: number;
  length: number;
}

interface Room {
  id: string;
  name: string;
  description?: string;
  created_at: string;
}

```

```

    updated_at: string;
  }

// Simple draggable component
function Draggable({id, children}) {
  const {
    attributes,
    listeners,
    setNodeRef,
    transform,
    isDragging
  } = useDraggable({
    id
  });

  // Only apply transform during active dragging
  const style = isDragging ? {
    transform: `translate3d(${transform?.x || 0}px, ${transform?.y
|| 0}px, 0)` ,
    zIndex: 1000,
    opacity: 0.8,
    position: 'absolute' as const,
    touchAction: 'none' as const
  } : {
    position: 'absolute' as const,
    touchAction: 'none' as const
  };

  return (
    <div
      id={id}
      ref={setNodeRef}
      style={style}
      {...listeners}
      {...attributes}
      data-dragging={isDragging ? "true" : "false"}
    >
      {children}
    </div>
  );
}

const LayoutEditor: React.FC = () => {
  const [tables, setTables] = useState<TableItem[]>([]);
  const [staticItems, setStaticItems] =
  useState<StaticItem[]>([]);
  const [walls, setWalls] = useState<WallItem[]>([]);
  const [selectedItem, setSelectedItem] = useState<any |
  null>(null);
  const [isDrawingWall, setIsDrawingWall] = useState(false);
  const [wallStartPoint, setWallStartPoint] = useState<{x: number,
  y: number} | null>(null);

```

```

    const [currentWallEndPoint, setCurrentWallEndPoint] =
useState<{x: number, y: number} | null>(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState<string | null>(null);
    const [updateCounter, setUpdateCounter] = useState(0);
    const editorRef = useRef<HTMLDivElement>(null);
    const [scale, setScale] = useState(1);

    // Добавляем состояние для отслеживания активной комнаты
    const [activeRoom, setActiveRoom] = useState<Room | null>(null);

    const sensors = useSensors(
      useSensor(PointerSensor, {
        activationConstraint: {
          distance: 5,
        },
      })
    );

    // Инициализация: получаем или создаем дефолтную комнату, затем
загружаем макет
    useEffect(() => {
      initializeRoomAndLayout();
    }, []);

    useEffect(() => {
      const handleResize = () => {
        if (editorRef.current) {
          const containerWidth = editorRef.current.clientWidth;
          // Set scale based on container width
          const newScale = Math.min(1, (containerWidth - 40) / 800);
          setScale(newScale);
        }
      };

      // Initial scale
      handleResize();

      // Add resize listener
      window.addEventListener('resize', handleResize);

      // Cleanup
      return () => {
        window.removeEventListener('resize', handleResize);
      };
    }, []);

    // Новая функция для инициализации комнаты и загрузки макета
    const initializeRoomAndLayout = async () => {
      try {
        setLoading(true);

        // Получаем или создаем дефолтную комнату

```

```

    const defaultRoom = await
layoutAPI.getOrCreateDefaultRoom();
    setActiveRoom(defaultRoom);

    // Загружаем макет для этой комнаты
    await fetchLayout(defaultRoom.id);
  } catch (err) {
    console.error('Failed to initialize room and layout:', err);
    setError('Не удалось инициализировать зал');
  } finally {
    setLoading(false);
  }
};

// Обновляем fetchLayout, чтобы принимать ID комнаты
const fetchLayout = async (roomId?: string) => {
  try {
    setLoading(true);
    const response = await layoutAPI.getLayout(roomId);

    // Map type_id to proper type_name strings
    const mapTypeIdToName = (typeId) => {
      switch (typeId) {
        case 1: return 'circular';
        case 2: return 'circular-large';
        case 3: return 'rectangular';
        case 4: return 'vip';
        case 5: return 'banquet';
        default: return 'circular'; // Default to circular
      }
    };

    // Ensure tables have proper type_name based on type_id
    const tablesWithTypeNames = (response.data.tables ||
[])?.map(table => ({
      ...table,
      type_name: table.type || mapTypeIdToName(table.type_id) //
Use type if available, otherwise map from type_id
    }));

    console.log('Fetched tables with type_name:',
tablesWithTypeNames);

    setTables(tablesWithTypeNames);
    setStaticItems(response.data.static_items || []);
    setWalls(response.data.walls || []);
    setError(null);
  } catch (err) {
    console.error('Failed to fetch layout:', err);
    setError('Не удалось загрузить план зала');
  } finally {
    setLoading(false);
  }
}

```

```

};

const forceUpdate = useCallback(() => {
  setUpdateCounter(prev => prev + 1);
}, []);

const handleDragEnd = (event: DragEndEvent) => {
  const { active, delta } = event;
  const dragId = active.id.toString();

  // Extract the item type and ID - correctly handle UUID format
  let type, itemId;
  if (dragId.startsWith('table-')) {
    type = 'table';
    itemId = dragId.substring(6); // Remove 'table-' prefix to
get the full ID
  } else if (dragId.startsWith('static-')) {
    type = 'static';
    itemId = dragId.substring(7); // Remove 'static-' prefix to
get the full ID
  } else if (dragId.startsWith('wall-')) {
    type = 'wall';
    itemId = dragId.substring(5); // Remove 'wall-' prefix
  } else {
    console.error('Unknown item type in ID:', dragId);
    return;
  }

  // Round delta values to snap to grid
  const roundedDeltaX = Math.round(delta.x / 5) * 5;
  const roundedDeltaY = Math.round(delta.y / 5) * 5;

  console.log("Drag end:", { dragId, type, itemId,
roundedDeltaX, roundedDeltaY });

  // Create completely new arrays to ensure React detects the
change
  if (type === 'table') {
    const updatedTables = [...tables];
    const tableIndex = updatedTables.findIndex(t => t.id ===
itemId);

    if (tableIndex >= 0) {
      const oldX = updatedTables[tableIndex].x;
      const oldY = updatedTables[tableIndex].y;
      const newX = oldX + roundedDeltaX;
      const newY = oldY + roundedDeltaY;

      console.log(`Updating table ${itemId} position from
(${oldX},${oldY}) to (${newX},${newY})`);

      updatedTables[tableIndex] = {
        ...updatedTables[tableIndex],

```

```

        x: newX,
        y: newY
    };

    setTables(updatedTables);

    // Update selected item if needed
    if (selectedItem && selectedItem.itemType === 'table' &&
selectedItem.id === itemId) {
        setSelectedItem({
            ...selectedItem,
            x: newX,
            y: newY
        });
    }
    } else {
        console.error(`Table with ID ${itemId} not found`);
    }
    } else if (type === 'static') {
        const updatedItems = [...staticItems];
        const itemIndex = updatedItems.findIndex(i => i.id ===
itemId);

        if (itemIndex >= 0) {
            const oldX = updatedItems[itemIndex].x;
            const oldY = updatedItems[itemIndex].y;
            const newX = oldX + roundedDeltaX;
            const newY = oldY + roundedDeltaY;

            console.log(`Updating static item ${itemId} position from
(${oldX},${oldY}) to (${newX},${newY})`);

            // Create a new object to ensure React detects the change
            updatedItems[itemIndex] = {
                ...updatedItems[itemIndex],
                x: newX,
                y: newY
            };

            // Set the state with the new array
            setStaticItems(updatedItems);

            // Update selected item if needed
            if (selectedItem && selectedItem.itemType === 'static' &&
selectedItem.id === itemId) {
                setSelectedItem({
                    ...selectedItem,
                    x: newX,
                    y: newY
                });
            }
        } else {
            console.error(`Static item with ID ${itemId} not found`);

```

```

    }
    } else if (type === 'wall') {
      const updatedWalls = [...walls];
      const wallIndex = updatedWalls.findIndex(w => w.id ===
itemId);

      if (wallIndex >= 0) {
        const oldX = updatedWalls[wallIndex].x;
        const oldY = updatedWalls[wallIndex].y;
        const newX = oldX + roundedDeltaX;
        const newY = oldY + roundedDeltaY;

        // Create a new object to ensure React detects the change
        updatedWalls[wallIndex] = {
          ...updatedWalls[wallIndex],
          x: newX,
          y: newY
        };

        // Set the state with the new array
        setWalls(updatedWalls);

        // Update selected item if needed
        if (selectedItem && selectedItem.itemType === 'wall' &&
selectedItem.id === itemId) {
          setSelectedItem({
            ...selectedItem,
            x: newX,
            y: newY
          });
        }
      } else {
        console.error(`Wall with ID ${itemId} not found`);
      }
    }

    // Force update after a small delay to ensure state is updated
    setTimeout(forceUpdate, 50);
  };

const handleItemClick = (id: string) => {
  // Find the clicked item
  const table = tables.find((t) => t.id === id);
  if (table) {
    setSelectedItem({ ...table, itemType: 'table' });
    return;
  }

  const staticItem = staticItems.find((item) => item.id === id);
  if (staticItem) {
    setSelectedItem({ ...staticItem, itemType: 'static' });
    return;
  }
}

```

```

const wall = walls.find((w) => w.id === id);
if (wall) {
  setSelectedItem({ ...wall, itemType: 'wall' });
  return;
}

// If clicked on empty space, deselect
if (!table && !staticItem && !wall) {
  setSelectedItem(null);
}
};

const rotateItem = () => {
  if (!selectedItem) return;

  if (selectedItem.itemType === 'table') {
    const updatedTables = tables.map((table) => {
      if (table.id === selectedItem.id) {
        return {
          ...table,
          rotation: (table.rotation + 90) % 360,
        };
      }
      return table;
    });
    setTables(updatedTables);
    setSelectedItem({ ...selectedItem, rotation:
(selectedItem.rotation + 90) % 360 });
  } else if (selectedItem.itemType === 'static') {
    const updatedItems = staticItems.map((item) => {
      if (item.id === selectedItem.id) {
        return {
          ...item,
          rotation: (item.rotation + 90) % 360,
        };
      }
      return item;
    });
    setStaticItems(updatedItems);
    setSelectedItem({ ...selectedItem, rotation:
(selectedItem.rotation + 90) % 360 });
  } else if (selectedItem.itemType === 'wall') {
    const updatedWalls = walls.map((wall) => {
      if (wall.id === selectedItem.id) {
        return {
          ...wall,
          rotation: (wall.rotation + 90) % 360,
        };
      }
      return wall;
    });
    setWalls(updatedWalls);
  }
};

```

```

        setSelectedItem({ ...selectedItem, rotation:
(selectedItem.rotation + 90) % 360 });
    }
};

const deleteItem = () => {
    if (!selectedItem) return;

    if (selectedItem.itemType === 'table') {
        setTables(tables.filter(table => table.id !==
selectedItem.id));
    } else if (selectedItem.itemType === 'static') {
        setStaticItems(staticItems.filter(item => item.id !==
selectedItem.id));
    } else if (selectedItem.itemType === 'wall') {
        setWalls(walls.filter(wall => wall.id !== selectedItem.id));
    }
    setSelectedItem(null);
};

// Обновляем addTable для использования activeRoom
const addTable = async (type: string) => {
    try {
        if (!activeRoom) {
            setError('Комната не инициализирована. Обновите
страницу.');
```

return;

```
        }

        // Obtain component details for defaults
        const tableComponent = LAYOUT_COMPONENTS.find(comp =>
comp.type === type && comp.isTable) as TableComponent;

        // Determine type_id based on table type
        let type_id: number;
        switch (type) {
            case 'circular': type_id = 1; break; // Круглый стол
            case 'circular-large': type_id = 2; break; // Большой
круглый стол
            case 'rectangular': type_id = 3; break; // Прямоугольный
стол
            case 'vip': type_id = 4; break; // VIP-стол
            case 'banquet': type_id = 5; break; // Банкетный зал
            default: type_id = 1; // По умолчанию круглый стол
        }

        // Count existing tables to determine the next number
        const existingTablesOfType = tables.filter(t => t.type_id
=== type_id).length;
        const nextTableNumber = existingTablesOfType + 1;

        // Set width and height based on table type
        let width = 80;

```

```

let height = 80;

switch (type) {
  case 'circular':
    width = 60;
    height = 60;
    break;
  case 'circular-large':
    width = 80;
    height = 80;
    break;
  case 'rectangular':
    width = 140;
    height = 60;
    break;
  case 'vip':
    width = 180;
    height = 180;
    break;
  case 'banquet':
    width = 200;
    height = 100;
    break;
}

// Create table data with appropriate dimensions
const tableData = {
  type_id: type_id,
  table_number: nextTableNumber,
  max_guests: type === 'banquet' ? 25 : (tableComponent ?
tableComponent.maxGuests : 2),
  x: 150,
  y: 150,
  rotation: 0,
  width: width,
  height: height,
};

console.log("Sending table data to server:", tableData);
const response = await layoutAPI.addTable(tableData,
activeRoom.id);
console.log("Server response:", response.data);

// Add type_name for frontend rendering
const newTable = {
  ...response.data,
  type: type, // Original string type
  type_name: type // For rendering
};

console.log('Added new table:', newTable);
setTables([...tables, newTable]);
setSelectedItem({ ...newTable, itemType: 'table' });

```

```

    } catch (err) {
      console.error('Failed to add table:', err);
      setError('Не удалось добавить стол');
    }
  };

  // Обновляем addStaticItem для использования activeRoom
  const addStaticItem = async (type: string) => {
    try {
      if (!activeRoom) {
        setError('Комната не инициализирована. Обновите
страницу.');
```

return;

```
      }

      const itemData = {
        type,
        x: 150,
        y: 150,
        rotation: 0,
      };

      const response = await layoutAPI.addStaticItem(itemData,
activeRoom.id);
      setStaticItems([...staticItems, response.data]);
      setSelectedItem({ ...response.data, itemType: 'static' });
    } catch (err) {
      console.error('Failed to add static item:', err);
      setError('Не удалось добавить элемент');
    }
  };

  const toggleWallDrawing = () => {
    setIsDrawingWall(!isDrawingWall);
    setWallStartPoint(null);
    setCurrentWallEndPoint(null);
  };

  const handleEditorClick = (e: React.MouseEvent) => {
    // Deselect if not drawing wall
    if (!isDrawingWall) {
      setSelectedItem(null);
    }
  };

  const handleEditorMouseDown = (e: React.MouseEvent) => {
    if (!isDrawingWall || !editorRef.current) return;

    // Get editor element position
    const rect = editorRef.current.getBoundingClientRect();

    // Calculate position and snap to grid (20px grid)
    const x = Math.round((e.clientX - rect.left) / 20) * 20;
```

```

    const y = Math.round((e.clientY - rect.top) / 20) * 20;

    setWallStartPoint({ x, y });
  };

  const handleEditorMouseMove = (e: React.MouseEvent) => {
    if (!isDrawingWall || !wallStartPoint || !editorRef.current)
    return;

    // Get editor element position
    const rect = editorRef.current.getBoundingClientRect();

    // Calculate position and snap to grid (20px grid)
    const rawX = e.clientX - rect.left;
    const rawY = e.clientY - rect.top;

    // Calculate the grid-aligned position
    let x = Math.round(rawX / 20) * 20;
    let y = Math.round(rawY / 20) * 20;

    // For straight lines: if close to straight (horizontal or
    vertical), snap to it
    const dx = Math.abs(x - wallStartPoint.x);
    const dy = Math.abs(y - wallStartPoint.y);

    // Snap to horizontal or vertical if within 10 pixels (adjust
    threshold as needed)
    if (dx > 0 && dy > 0) { // Not already perfectly aligned
      if (dy < 20) { // Close to horizontal
        y = wallStartPoint.y;
      } else if (dx < 20) { // Close to vertical
        x = wallStartPoint.x;
      }
    }

    setCurrentWallEndPoint({ x, y });
  };

  const handleEditorMouseUp = (e: React.MouseEvent) => {
    if (!isDrawingWall || !wallStartPoint || !editorRef.current)
    return;

    // Get editor element position
    const rect = editorRef.current.getBoundingClientRect();

    // Calculate position and snap to grid (20px grid)
    const x = Math.round((e.clientX - rect.left) / 20) * 20;
    const y = Math.round((e.clientY - rect.top) / 20) * 20;

    // Check if points are different
    if (x === wallStartPoint.x && y === wallStartPoint.y) {
      // Just a click, not a drag
      return;
    }
  };

```

```

    }

    // Calculate length and angle
    const dx = x - wallStartPoint.x;
    const dy = y - wallStartPoint.y;

    const length = Math.sqrt(dx * dx + dy * dy);
    if (length < 20) return;

    const angle = Math.atan2(dy, dx) * (180 / Math.PI);

    addWall(wallStartPoint.x, wallStartPoint.y, angle, length);
};

const handleEditorMouseLeave = () => {
  if (isDrawingWall && wallStartPoint) {
    // Reset if mouse leaves the editor during drawing
    setWallStartPoint(null);
    setCurrentWallEndPoint(null);
  }
};

// Обновляем addWall для использования activeRoom
const addWall = async (x: number, y: number, rotation: number,
length: number) => {
  try {
    if (!activeRoom) {
      setError('Комната не инициализирована. Обновите
страницу.');
```

return;

```

    }
    const wallData = {
      x,
      y,
      rotation,
      length,
    };
    const response = await layoutAPI.addWall(wallData,
activeRoom.id);
    setWalls([...walls, response.data]);
    setSelectedItem({ ...response.data, itemType: 'wall' });

    // Exit wall drawing mode after adding a wall
    setIsDrawingWall(false);
    setWallStartPoint(null);
    setCurrentWallEndPoint(null);
  } catch (err) {
    console.error('Failed to add wall:', err);
    setError('Не удалось добавить стену');
  }
};

```

```

// Обновляем clearLayout для использования activeRoom
const clearLayout = async () => {
  if (!activeRoom) {
    setError('Комната не инициализирована. Обновите страницу.');
```

return;

```
  }

  if (window.confirm('Вы уверены, что хотите очистить план зала?
Все элементы будут удалены.')) {
    try {
      await layoutAPI.clearLayout(activeRoom.id);
      setTables([]);
      setStaticItems([]);
      setWalls([]);
      setSelectedItem(null);
      setError(null);
    } catch (err) {
      console.error('Failed to clear layout:', err);
      setError('Не удалось очистить план зала');
```

}

```
  }
};

// Обновляем saveLayout для использования activeRoom
const saveLayout = async () => {
  try {
    if (!activeRoom) {
      setError('Комната не инициализирована. Обновите
страницу.');
```

return;

```
    }

    // Преобразуем таблицы для отправки на сервер
    const transformedTables = tables.map(({ id, ...rest }) => ({
      ...rest,
      type: rest.type_name // Ensure type field is set properly
for backend
    }));

    const layoutData = {
      tables: transformedTables,
      static_items: staticItems.map(({ id, ...rest }) => rest),
      walls: walls.map(({ id, ...rest }) => rest),
    };

    // Отладочный вывод
    console.log('Saving layout:', JSON.stringify(layoutData));

    await layoutAPI.saveLayout(layoutData, activeRoom.id);
    setError(null);
    alert('План зала сохранен успешно');
```

} catch (err) {

```

        console.error('Failed to save layout:', err);
        setError('Не удалось сохранить план зала');
    }
};

// Debug output of current positions
useEffect(() => {
    if (tables.length > 0 || staticItems.length > 0 ||
walls.length > 0) {
        console.log("Current table positions:", tables.map(t => ({
id: t.id, x: t.x, y: t.y })));
        console.log("Current static item positions:",
staticItems.map(i => ({ id: i.id, x: i.x, y: i.y })));
        console.log("Current wall positions:", walls.map(w => ({ id:
w.id, x: w.x, y: w.y, length: w.length })));
    }
}, [tables, staticItems, walls, updateCounter]);

if (loading) {
    return <div className="p-8 text-center">Загрузка...</div>;
}

// Добавляем информацию о текущей комнате
return (
    <div className="flex flex-col max-w-[1400px] mx-auto px-4">
        {/* Error display */}
        {error && (
            <div className="bg-red-100 border border-red-400 text-red-
700 px-4 py-3 rounded mb-4">
                {error}
            </div>
        )}

        {/* Room information */}
        {activeRoom && (
            <div className="bg-blue-50 border border-blue-200 text-
blue-800 px-4 py-2 rounded mb-4">
                Текущий зал: {activeRoom.name} {activeRoom.description ?
`${activeRoom.description}` : ''}
            </div>
        )}

        {/* Layout controls - top toolbar */}
        <div className="flex flex-col md:flex-row md:items-start
md:justify-between gap-4 mb-4">
            {/* Add elements panel */}
            <div className="bg-white p-2 border rounded-lg shadow-md
max-w-full">
                <h3 className="font-bold mb-2 text-center text-
sm">Добавить элемент</h3>
                <div className="flex flex-wrap gap-2">
                    {LAYOUT_COMPONENTS.filter(comp => comp.type !==
'wall').map((component) => (

```

```

        <button
          key={component.type}
          onClick={() => component.isTable ?
addTable(component.type) : addStaticItem(component.type)}
          className="p-1 bg-white border rounded hover:bg-
gray-100 shadow-sm flex items-center justify-center"
          title={component.title}
        >
          <div className={` ${component.scale} transform`} >
            {renderComponentPreview(component)}
          </div>
        </button>
      )}}
    </div>
  </div>

  { /* Buttons - save, draw wall, clear */ }
  <div className="flex flex-col gap-2">
    <button
      onClick={saveLayout}
      className="px-4 py-2 bg-blue-500 text-white rounded-lg
hover:bg-blue-600 shadow-md w-full"
    >
      Сохранить план
    </button>
    <button
      onClick={toggleWallDrawing}
      className={`px-4 py-2 border rounded-lg w-full ${
        isDrawingWall ? 'bg-yellow-500 text-white hover:bg-
yellow-600' : 'bg-gray-200 text-gray-800 hover:bg-gray-300'
      }}
    >
      {isDrawingWall ? 'Закончить рисование стен' :
'Рисовать стены'}
    </button>
    <button
      onClick={clearLayout}
      className="px-4 py-2 bg-red-500 text-white rounded-lg
hover:bg-red-600 shadow-md w-full"
    >
      Очистить план
    </button>
  </div>
</div>

  { /* Drawing instructions */ }
  {isDrawingWall && (
    <div className="bg-yellow-100 border border-yellow-400
text-yellow-800 px-4 py-2 rounded mb-4">
      Нажмите и перетащите мышь, чтобы нарисовать стену. Стены
будут привязаны к сетке.
    </div>
  )}

```

```

    { /* Editor area */ }
    <div className="mt-4 bg-white rounded-lg shadow-md overflow-
auto" ref={editorRef}>
      <div
        className="border relative"
        style={{
          width: '100%',
          height: '80vh',
          margin: '0 auto',
          backgroundColor: '#f9fafb',
          minWidth: '800px',
          minHeight: '600px'
        }}
        onClick={handleEditorClick}
        onMouseDown={handleEditorMouseDown}
        onMouseMove={handleEditorMouseMove}
        onMouseUp={handleEditorMouseUp}
        onMouseLeave={handleEditorMouseLeave}
      >
        { /* Grid background */ }
        <div className="absolute inset-0" style={{
          backgroundImage: 'linear-gradient(to right, #e5e7eb
1px, transparent 1px), linear-gradient(to bottom, #e5e7eb 1px,
transparent 1px)',
          backgroundSize: '20px 20px'
        }}></div>

        { /* Selection controls (SVG icons) */ }
        {selectedItem && (
          <div className="absolute top-2 right-2 flex space-x-2
z-50">
            <button
              onClick={rotateItem}
              className="p-2 bg-white rounded-full shadow-md
hover:bg-gray-100"
              title="Повернуть"
            >
              <svg xmlns="http://www.w3.org/2000/svg" width="20"
height="20" viewBox="0 0 24 24" fill="none" stroke="black"
strokeWidth="2" strokeLinecap="round" strokeLinejoin="round">
                <path d="M21.5 2v6h-6M21.34 15.57a10 10 0 1 1-
.57-8.38"/>
              </svg>
            </button>
            <button
              onClick={deleteItem}
              className="p-2 bg-white rounded-full shadow-md
hover:bg-gray-100"
              title="Удалить"
            >

```

```

        <svg xmlns="http://www.w3.org/2000/svg" width="20"
height="20" viewBox="0 0 24 24" fill="none" stroke="black"
strokeWidth="2" strokeLinecap="round" strokeLinejoin="round">
        <polyline points="3 6 5 6 21 6"></polyline>
        <path d="M19 6v14a2 2 0 0 1-2 2H7a2 2 0 0 1-2-
2V6m3 0V4a2 2 0 0 1 2-2h4a2 2 0 0 1 2 2v2"></path>
        </svg>
    </button>
</div>
))

{/* Wall drawing overlay */}
{isDrawingWall && wallStartPoint && currentWallEndPoint
&& (
    <div
        style={{
            position: 'absolute',
            left: `${Math.min(wallStartPoint.x,
currentWallEndPoint.x)}px`,
            top: `${Math.min(wallStartPoint.y,
currentWallEndPoint.y)}px`,
            width: `${Math.abs(currentWallEndPoint.x -
wallStartPoint.x)}px`,
            height: `${Math.abs(currentWallEndPoint.y -
wallStartPoint.y)}px`,
            border: '2px dashed #000',
            backgroundColor: 'rgba(0, 0, 0, 0.1)',
            zIndex: 100
        }}
    />
)}

{/* Wall start point marker */}
{isDrawingWall && wallStartPoint && (
    <div
        style={{
            position: 'absolute',
            left: `${wallStartPoint.x - 5}px`,
            top: `${wallStartPoint.y - 5}px`,
            width: '10px',
            height: '10px',
            borderRadius: '50%',
            backgroundColor: 'blue',
            zIndex: 101
        }}
    />
)}

{/* Tables, walls, and static items */}
<DndContext
    sensors={sensors}
    onDragEnd={handleDragEnd}
>

```

```

    <div className="h-full w-full relative">
      {/* Walls */}
      {walls.map((wall) => (
        <Draggable key={`wall-${wall.id}`} id={`wall-
${wall.id}`}>
          <div
            key={`wall-container-${wall.id}-
${updateCounter}`}
          >
            <Wall
              id={wall.id}
              x={wall.x}
              y={wall.y}
              rotation={wall.rotation}
              length={wall.length}
              onClick={(id) => {
                handleItemClick(id);
              }}
              isEditing={true}
              isSelected={selectedItem &&
selectedItem.itemType === 'wall' && selectedItem.id === wall.id}
            />
          </div>
        </Draggable>
      )
    )}

    {/* Tables */}
    {tables.map((table) => {
      // Ensure we have valid width and height
      const tableWidth = table.width || (table.type_name
=== 'banquet' ? 200 : 80);
      const tableHeight = table.height ||
(table.type_name === 'banquet' ? 100 : 80);

      console.log(`Rendering table ${table.id}, type:
${table.type_name}, dimensions: ${tableWidth}x${tableHeight}`);

      return (
        <Draggable key={`table-${table.id}`} id={`table-
${table.id}`}>
          <div
            key={`table-container-${table.id}-
${updateCounter}`}
            onClick={(e) => {
              e.stopPropagation();
              handleItemClick(table.id);
            }}
            style={{
              position: 'absolute',
              left: `${table.x}px`,
              top: `${table.y}px`,
              width: `${tableWidth}px`,
              height: `${tableHeight}px`,

```

```

                                transform: `rotate(${table.rotation ||
0}deg)` ,
                                transformOrigin: 'center',
                                zIndex: selectedItem &&
selectedItem.itemType === 'table' && selectedItem.id === table.id
? 20 : 10,
                                }}
                                className={selectedItem &&
selectedItem.itemType === 'table' && selectedItem.id === table.id
? 'ring-2 ring-blue-500' : ''}
                                >
                                <div style={{ width: '100%', height: '100%'
}}>
                                {renderTableByType({
                                type_name: table.type_name ||
'circular',
                                table_number: table.table_number || 1,
                                maxGuests: table.max_guests || 4,
                                isSelected: selectedItem &&
selectedItem.itemType === 'table' && selectedItem.id === table.id,
                                isAdmin: true
                                })}
                                </div>
                                </div>
                                </Draggable>
                                );
                                }}}

                                {/* Static items */}
                                {staticItems.map((item) => (
                                <Draggable key={`static-${item.id}`} id={`static-
${item.id}`}>
                                <div
                                key={`static-item-${item.id}-
${updateCounter}`}
                                onClick={(e) => {
                                e.stopPropagation();
                                handleItemClick(item.id);
                                }}
                                style={{
                                position: 'absolute',
                                left: `${item.x}px`,
                                top: `${item.y}px`,
                                zIndex: selectedItem &&
selectedItem.itemType === 'static' && selectedItem.id === item.id
? 20 : 10,
                                }}
                                className={selectedItem &&
selectedItem.itemType === 'static' && selectedItem.id === item.id
? 'ring-2 ring-blue-500' : ''}
                                >
                                <StaticItem
                                id={item.id}

```

```

        type={item.type}
        x={0}
        y={0}
        width={item.width || 80}
        height={item.height || 60}
        rotation={item.rotation || 0}
        scale={scale}
      />
    </div>
  </Draggable>
  )))
</div>
</DndContext>
</div>
</div>
</div>
);
};

export default LayoutEditor;

```

Листинг Г.2 – Файл ImageUpload.tsx

```

import React, { useState, useRef } from 'react';
import axios from 'axios';

const API_URL = process.env.NEXT_PUBLIC_API_URL ||
'http://localhost:8000';

interface ImageUploadProps {
  currentImage?: string;
  onImageUploaded: (imageUrl: string) => void;
}

const ImageUpload: React.FC<ImageUploadProps> = ({ currentImage,
onImageUploaded }) => {
  const [isUploading, setIsUploading] = useState(false);
  const [uploadError, setUploadError] = useState<string |
null>(null);
  const [previewUrl, setPreviewUrl] = useState<string |
null>(currentImage || null);
  const fileInputRef = useRef<HTMLInputElement>(null);

  const defaultImageUrl = '/images/unknown.png';

  const handleFileChange = async (e:
React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;

    // Check file type
    if (!file.type.match('image.*')) {

```

```

        setUploadError('Пожалуйста, выберите изображение');
        return;
    }

    // Check file size (max 5MB)
    if (file.size > 5 * 1024 * 1024) {
        setUploadError('Размер файла не должен превышать 5MB');
        return;
    }
    // Create local preview
    const reader = new FileReader();
    reader.onload = () => {
        setPreviewUrl(reader.result as string);
    };
    reader.readAsDataURL(file);

    // Upload the file
    const formData = new FormData();
    formData.append('file', file);

    setIsUploading(true);
    setUploadError(null);

    try {
        const response = await
        axios.post(`${API_URL}/uploads/images`, formData, {
            headers: {
                'Content-Type': 'multipart/form-data',
                'Authorization': `Bearer
                ${localStorage.getItem('token')}`
            }
        });

        if (response.data.url) {
            onImageUploaded(response.data.url);
        } else {
            throw new Error('Ошибка загрузки изображения');
        }

    } catch (error) {
        console.error('Image upload failed:', error);
        setUploadError('Не удалось загрузить изображение.
        Пожалуйста, попробуйте снова.');
```

```

    } finally {
        setIsUploading(false);
    }
};

const handleRemoveImage = () => {
    setPreviewUrl(null);
    onImageUploaded('');

    // Reset file input

```

```

    if (fileInputRef.current) {
      fileInputRef.current.value = '';
    }
  };

  const handleImageError = () => {
    if (previewUrl !== defaultImageUrl) {
      setPreviewUrl(defaultImageUrl);
    }
  };

  return (
    <div className="image-upload">
      <div className="mb-2 flex flex-col items-center">
        {previewUrl ? (
          <div className="relative w-40 h-40 mb-2">
            <img
              src={previewUrl}
              alt="Preview"
              className="w-full h-full object-cover rounded-md
shadow-sm"
              onError={handleImageError}
            />
            <button
              type="button"
              onClick={handleRemoveImage}
              className="absolute -top-2 -right-2 w-6 h-6 bg-red-
500 text-white rounded-full flex items-center justify-center"
              title="Удалить изображение"
            >
              ×
            </button>
          </div>
        ) : (
          <div className="w-40 h-40 border-2 border-dashed border-
gray-300 rounded-md flex items-center justify-center mb-2">
            <div className="text-center text-gray-400">
              <div className="text-sm">Нет изображения</div>
            </div>
          </div>
        )}

        <input
          ref={fileInputRef}
          type="file"
          accept="image/*"
          onChange={handleFileChange}
          className="hidden"
          id="image-upload"
        />

        <label
          htmlFor="image-upload"

```

```

        className={`px-4 py-2 bg-blue-500 text-white rounded-md
        cursor-pointer ${isUploading ? 'opacity-50 cursor-not-allowed' :
        'hover:bg-blue-600'}}`
        tabIndex={0}
      >
        {isUploading ? 'Загрузка...' : (previewUrl ? 'Изменить
        изображение' : 'Загрузить изображение'))
      </label>
    </div>

    {uploadError && (
      <div className="text-red-500 text-sm text-
        center">{uploadError}</div>
    )}
  </div>
);
};

export default ImageUpload;

```

Листинг Г.3 – Файл Menu.tsx

```

import React, { useState, useEffect } from 'react';
import { menuAPI } from '../lib/api';
import { Category, MenuItem, OrderItem } from
  '../types/components';

interface MenuProps {
  onAddToOrder: (item: OrderItem) => void;
  reservationId?: string;
  readOnly?: boolean;
}

const Menu: React.FC<MenuProps> = ({ onAddToOrder, reservationId,
  readOnly = false }) => {
  const [menu, setMenu] = useState<{categories: Category[], items:
  MenuItem[]}>({ categories: [], items: [] });
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [selectedCategory, setSelectedCategory] = useState<string
  | null>(null);
  const [quantities, setQuantities] = useState<Record<string,
  number>>({});

  useEffect(() => {
    const fetchMenu = async () => {
      try {
        setLoading(true);
        const response = await menuAPI.getMenu();
        setMenu(response.data);

        if (response.data.categories.length > 0) {
          setSelectedCategory(response.data.categories[0].id);
        }
      } catch {
        setError('Ошибка загрузки меню');
      }
    };
    fetchMenu();
  }, [reservationId]);

```

```

    }

    // Initialize quantities
    const initialQuantities: Record<string, number> = {};
    response.data.items.forEach((item: MenuItem) => {
        initialQuantities[item.id] = 0;
    });
    setQuantities(initialQuantities);

    } catch (err) {
        console.error('Failed to fetch menu:', err);
        setError('Не удалось загрузить меню');
    } finally {
        setLoading(false);
    }
};

fetchMenu();
}, []);

const handleQuantityChange = (itemId: string, change: number) =>
{
    setQuantities((prev) => {
        const newValue = Math.max(0, (prev[itemId] || 0) + change);
        return { ...prev, [itemId]: newValue };
    });
};

const handleAddToOrder = (itemId: string) => {
    if (quantities[itemId] > 0) {
        onAddToOrder({
            menu_item_id: itemId,
            quantity: quantities[itemId]
        });

        // Reset quantity after adding to order
        setQuantities((prev) => ({ ...prev, [itemId]: 0 }));
    }
};

if (loading) return <div className="p-4 text-center">Загрузка
меню...</div>;
if (error) return <div className="p-4 text-red-
500">{error}</div>;
if (menu.categories.length === 0) return <div className="p-4
text-center">Меню пока недоступно</div>;

const filteredItems = selectedCategory
    ? menu.items.filter(item => item.category_id ===
selectedCategory)
    : menu.items;

// Default image path

```

```

const defaultImageUrl = '/images/unknown.png';
return (
  <div className="menu-container">
    <div className="category-tabs mb-4 pb-2 border-b gap-2 flex
overflow-x-auto">
      {menu.categories.map(category => (
        <button
          key={category.id}
          className={`px-4 py-2 font-medium text-sm whitespace-
nowrap ${
            selectedCategory === category.id
              ? 'border-b-2 bg-blue-500 text-white'
              : 'text-gray-500 bg-gray-200 hover:bg-gray-300 '
            }`}
          onClick={() => setSelectedCategory(category.id)}
        >
          {category.name}
        </button>
      ))}
    </div>

    <div className="menu-items space-y-4">
      {filteredItems.map(item => (
        <div
          key={item.id}
          className="menu-item p-4 border rounded-lg flex gap-4
items-center"
        >
          <div className="item-image">
            <img
              src={item.image_url || defaultImageUrl}
              alt={item.name}
              className="w-24 h-24 object-cover rounded"
              onError={(e) => {
                const target = e.target as HTMLImageElement;
                target.onerror = null;
                target.src = defaultImageUrl;
              }}
            />
          </div>
          <div className="item-info flex-1">
            <h3 className="font-medium">{item.name}</h3>
            <p className="text-sm text-gray-
600">{item.description}</p>
            <p className="text-sm font-semibold mt-
1">{item.price.toFixed(2)} ₺</p>
          </div>

          {!readOnly && (
            <div className="item-actions flex items-center">
              <button
                onClick={() => handleQuantityChange(item.id, -
1)}

```

```

        disabled={quantities[item.id] <= 0}
        className="w-8 h-8 rounded-full bg-gray-200 flex
items-center justify-center disabled:opacity-50"
      >
        -
      </button>
      <span className="mx-2 w-6 text-
center">{quantities[item.id] || 0}</span>
      <button
        onClick={() => handleQuantityChange(item.id, 1)}
        className="w-8 h-8 rounded-full bg-gray-200 flex
items-center justify-center"
      >
        +
      </button>
      <button
        onClick={() => handleAddToOrder(item.id)}
        disabled={!quantities[item.id]}
        className="ml-4 px-3 py-1 text-sm bg-blue-500
text-white rounded disabled:opacity-50"
      >
        Добавить
      </button>
    </div>
  )}
</div>
)}}
</div>
</div>
);
};

```

```
export default Menu;
```

Листинг Г.4 – Файл ReservationCard.tsx

```

import React, { useState } from 'react';
import { format, parseISO, addHours, isPast } from 'date-fns';
import { ru } from 'date-fns/locale';
import Image from 'next/image';
import { reservationsAPI } from '../lib/api';
import { renderTableByType } from './TableStyles';

interface ReservationCardProps {
  reservation: {
    id: string;
    reservation_date: string;
    reservation_time: string;
    guests_count: number;
    first_name: string;
    last_name: string;
    phone: string;
    table_id: string;
  };
}

```

```

table?: {
  id: string;
  type_id: number;
  table_number: number;
  width?: number;
  height?: number;
  table_type?: {
    id: number;
    name: string;
    display_name: string;
    color_code?: string;
    default_width: number;
    default_height: number;
    default_max_guests: number;
  };
};
table_name?: string;
total_price?: number;
status?: string;
duration?: number;
};
tableNameMap?: Record<string, string>;
onDelete?: (id: string) => void;
onEdit?: (id: string) => void;
}

const ReservationCard: React.FC<ReservationCardProps> = ({
  reservation,
  tableNameMap = {},
  onDelete,
  onEdit
}) => {
  const [showModal, setShowModal] = useState(false);
  const [isDeleting, setIsDeleting] = useState(false);
  const [error, setError] = useState<string | null>(null);

  // Format the date to a more readable format
  const formattedDate = (() => {
    try {
      const date = parseISO(reservation.reservation_date);
      return format(date, 'd MMMM', { locale: ru });
    } catch (e) {
      return reservation.reservation_date;
    }
  })();

  // Get the table name from the map, table data, or use a default
  const tableName = reservation.table_name ||
    (reservation.table?.table_type?.display_name &&
      `${reservation.table.table_type.display_name}
№${reservation.table.table_number}`) ||
    tableNameMap[reservation.table_id] ||
    `Столик №${reservation.table_id.slice(0, 4)}`;

```

```

    // Get table type for icon rendering
    const tableType = reservation.table?.table_type?.name ||
'circular';
    const tableNumber = reservation.table?.table_number || 1;

    // Check if reservation can be edited or canceled
    const reservationDateTime = (() => {
      try {
        const date = parseISO(reservation.reservation_date);
        const [hours, minutes] =
reservation.reservation_time.split(':');
        const result = new Date(date);
        result.setHours(parseInt(hours, 10));
        result.setMinutes(parseInt(minutes, 10));
        return result;
      } catch (e) {
        return new Date();
      }
    })();

    const canCancel = new Date() < addHours(reservationDateTime, -
6);
    const canEdit = new Date() < addHours(reservationDateTime, -3);
    const isPastReservation = isPast(reservationDateTime);

    // Get the appropriate card width based on table type
    const getCardWidth = () => {
      switch (tableType) {
        case 'rectangular':
          return '260px';
        case 'vip':
        case 'banquet':
          return '280px';
        default:
          return '220px';
      }
    };

    // Get margins for wider cards to prevent overlapping
    const getCardMargin = () => {
      switch (tableType) {
        case 'rectangular':
        case 'vip':
        case 'banquet':
          return 'mx-2'; // Add horizontal margin to wider cards
        default:
          return '';
      }
    };

    const getIconWidth = () => {
      switch (tableType) {

```

```

    case 'rectangular':
      return 'w-24 h-20 mt-6';
    case 'vip':
    case 'banquet':
      return 'w-20 h-20 mt-6';
    default:
      return 'w-16 h-16 mt-10';
  }
};

const handleDelete = async () => {
  if (!canCancel) {
    setError("Бронирование можно отменить не позднее чем за 6 часов до начала");
    return;
  }

  try {
    setIsDeleting(true);
    setError(null);
    await reservationsAPI.deleteReservation(reservation.id);
    setShowModal(false);
    if (onDelete) {
      onDelete(reservation.id);
    }
  } catch (err: any) {
    console.error('Failed to delete reservation:', err);
    setError(err.response?.data?.detail || 'Не удалось отменить бронирование');
  } finally {
    setIsDeleting(false);
  }
};

const handleEdit = () => {
  if (!canEdit) {
    setError("Бронирование можно изменить не позднее чем за 3 часа до начала");
    return;
  }

  if (onEdit) {
    onEdit(reservation.id);
  }
};

return (
  <>
  <div
    className={`border rounded-lg p-3 shadow-sm bg-white
    hover:shadow-md transition-shadow cursor-pointer flex flex-col
    ${getCardMargin()} `}
    onClick={() => setShowModal(true)}
  >

```

```

    style={{ width: getCardWidth() }}
  >
    { /* Table Icon */ }
    <div className="flex justify-center mb-2">
      <div className={getIconWidth()}>
        {renderTableByType({
          type_name: tableType,
          table_number: tableNumber,
          maxGuests: reservation.guests_count,
          isReserved: false
        })}
      </div>
    </div>

    { /* Status badge if needed */ }
    {reservation.status && (
      <div className="absolute top-2 right-2">
        <span className={`text-xs px-2 py-1 rounded ${
          reservation.status === 'confirmed' ? 'bg-green-100
text-green-800' :
          reservation.status === 'pending' ? 'bg-yellow-100
text-yellow-800' :
          'bg-gray-100 text-gray-800'
        }`}>
          {reservation.status === 'confirmed' ? 'Подтверждено'
:
          reservation.status === 'pending' ? 'Ожидание' :
          reservation.status}
        </span>
      </div>
    )}

    { /* Info line - in a single line */ }
    <div className="text-xs text-gray-600 text-center mt-5">
      <div className="truncate">
        {formattedDate} .
        {reservation.reservation_time.substring(0, 5)} .
        {reservation.guests_count} {reservation.guests_count === 1 ?
        'гость' : reservation.guests_count < 5 ? 'гостя' : 'гостей'}
      </div>
    </div>
  </div>

  { /* Reservation Detail Modal */ }
  {showModal && (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center p-4 z-50">
      <div className="bg-white rounded-lg shadow-xl max-w-md
w-full max-h-90vh overflow-y-auto p-6" onClick={(e) =>
e.stopPropagation()}>
        <div className="flex justify-between items-start mb-
4">

```

```

        <h2 className="text-xl font-bold text-gray-
900">Информация о бронировании</h2>
        <button
            className="text-gray-500 hover:text-gray-700"
            onClick={() => setShowModal(false)}
        >
            <svg xmlns="http://www.w3.org/2000/svg"
className="h-6 w-6" fill="none" viewBox="0 0 24 24"
stroke="currentColor">
                <path strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M6 18L18 6M6 6l12 12" />
            </svg>
        </button>
    </div>

    <div className="space-y-4">
        { /* Table visualization */ }
        <div className="flex justify-center p-3">
            <div className={getIconWidth()}>
                {renderTableByType({
                    type_name: tableType,
                    table_number: tableNumber,
                    maxGuests: reservation.guests_count,
                    isReserved: false
                })}
            </div>
        </div>
    </div>

    <div className="bg-gray-50 p-3 rounded-lg">
        <h3 className="font-semibold text-gray-700 mb-
2">Детали бронирования</h3>
        <div className="grid grid-cols-2 gap-3 text-sm">
            <div className="text-gray-600">Дата:</div>
            <div className="font-
medium">{formattedDate}</div>

            <div className="text-gray-600">Время:</div>
            <div className="font-
medium">{reservation.reservation_time.substring(0, 5)}</div>

            <div className="text-gray-
600">Длительность:</div>
            <div className="font-
medium">{reservation.duration}
                {reservation.duration === 1 ? 'час' :
                reservation.duration < 5 ? ' часа' : '
часов'}</div>

            <div className="text-gray-600">Количество
гостей:</div>
            <div className="font-
medium">{reservation.guests_count}</div>

```

```

        <div className="text-gray-600">Стол:</div>
        <div className="font-medium">{tableName}</div>
    </div>
</div>

<div className="bg-gray-50 p-3 rounded-lg">
    <h3 className="font-semibold text-gray-700 mb-2">Контактная информация</h3>
    <div className="grid grid-cols-2 gap-3 text-sm">
        <div className="text-gray-600">Имя:</div>
        <div className="font-medium">{reservation.first_name}</div>

        <div className="text-gray-600">Фамилия:</div>
        <div className="font-medium">{reservation.last_name}</div>

        <div className="text-gray-600">Телефон:</div>
        <div className="font-medium">{reservation.phone}</div>
    </div>
</div>

    {error && (
        <div className="bg-red-50 text-red-700 p-3 rounded-lg text-sm">
            {error}
        </div>
    )}

    <div className="flex justify-between pt-4 border-t border-gray-200">
        <button
            className="px-4 py-2 bg-gray-200 text-gray-800 rounded hover:bg-gray-300 transition-colors"
            onClick={() => setShowModal(false)}
        >
            Закрыть
        </button>

        {!isPastReservation && (
            <div className="flex space-x-2">
                {canEdit && (
                    <button
                        className="flex items-center px-4 py-2 bg-blue-600 text-white rounded hover:bg-blue-700 transition-colors"
                        onClick={handleEdit}
                    >
                        <svg xmlns="http://www.w3.org/2000/svg"
                            className="h-5 w-5 mr-2" fill="none" viewBox="0 0 24 24"
                            stroke="currentColor">
                            <path strokeLinecap="round"
                                strokeLinejoin="round" strokeWidth={2} d="M15.232 5.232 13.536

```

```

3.536m-2.036-5.036a2.5 2.5 0 113.536 3.536L6.5 21.036H3v-
3.572L16.732 3.732z" />
        </svg>
        <span>Изменить</span>
    </button>
  )}

  {canCancel && (
    <button
      className="flex items-center px-4 py-2 bg-
red-600 text-white rounded hover:bg-red-700 transition-colors"
      onClick={handleDelete}
      disabled={isDeleting}
    >
      {isDeleting ? (
        <span>Отмена...</span>
      ) : (
        <>
          <Image
            src="/images/cancel-reservation-
svgrepo-com.svg"
            alt="Cancel"
            width={20}
            height={20}
            className="mr-2"
          />
          <span>Отменить</span>
        </>
      )}
    </button>
  )}
</div>
)}
</div>
</div>
</div>
</div>
)}
</>
);
};

```

```
export default ReservationCard;
```

Листинг Г.5 – Файл Table.tsx

```

import React from 'react';
import { renderTableByType } from '../TableStyles';

interface TableProps {
  id: string;
  type_id: number;
  type_name: string;

```

```

    table_number: number;
    maxGuests: number;
    x: number;
    y: number;
    rotation: number;
    available: boolean;
    onClick?: (id: string) => void;
    isEditing?: boolean;
    isSelected?: boolean;
    scale?: number;
    width?: number;
    height?: number;
  }

const Table: React.FC<TableProps> = ({
  id,
  type_id,
  type_name,
  table_number,
  maxGuests,
  x,
  y,
  rotation,
  available,
  onClick,
  isEditing = false,
  isSelected = false,
  scale = 1,
  width: propWidth,
  height: propHeight,
}) => {
  const handleClick = (e: React.MouseEvent) => {
    e.stopPropagation();
    if (onClick && (available || isEditing)) {
      onClick(id);
    }
  };

  // Функция определения размеров стола
  const getTableDimensions = () => {
    // Use props width/height if provided
    if (propWidth && propHeight) {
      return { width: `${propWidth}px`, height: `${propHeight}px` };
    }

    // Проверяем, что type_name определен
    const tableType = type_name && typeof type_name === 'string'
      ? type_name.toLowerCase()
      : '';

    switch (tableType) {
      case 'circular':

```

```

        return { width: '64px', height: '64px' };
    case 'circular-large':
        return { width: '80px', height: '80px' };
    case 'rectangular':
        return { width: '144px', height: '64px' };
    case 'vip':
        return { width: '192px', height: '192px' };
    case 'banquet':
        return { width: '200px', height: '100px' };
    default:
        return { width: '64px', height: '64px' };
    }
};

const { width, height } = getTableDimensions();

return (
    <div
        style={{
            position: 'absolute',
            left: `${(x - (parseInt(width) / 2) / scale) * scale}px`,
            top: `${(y - (parseInt(height) / 2) / scale) * scale}px`,
            width: width,
            height: height,
            transform: `scale(${scale}) rotate(${rotation}deg)`,
            transformOrigin: 'center',
            cursor: available || isEditing ? 'move' : 'not-allowed',
            zIndex: isSelected ? 20 : 10,
        }}
        onClick={handleClick}
    >
        {renderTableByType({
            type_name,
            table_number,
            maxGuests,
            isReserved: !available,
            isSelected
        })}
    </div>
);
};

export default Table;

```

Листинг Г.6 – Файл TableStyles.tsx

```

import React from 'react';

// Common table rendering functions for consistent style across
the application
export interface TableRenderProps {
    type_name: string;
    table_number: number;

```

```

    maxGuests: number;
    isReserved?: boolean;
    isSelected?: boolean;
    isAdmin?: boolean;
  }

  // Helper function to get the correct word form for guest based on
  count
  const getGuestsWord = (count: number): string => {
    if (count === 1) return 'место';
    if (count >= 2 && count <= 4) return 'места';
    return 'мест';
  };

  // Helper function for "гость" word forms
  const getGuestWord = (count: number): string => {
    if (count === 1) return 'гость';
    if (count >= 2 && count <= 4) return 'гостя';
    return 'гостей';
  };

  // Shared rendering functions for different table types
  export const renderCircularTable = (props: TableRenderProps) => {
    const { table_number, maxGuests, isReserved = false, isSelected
    = false, isAdmin = false } = props;
    const displayText = `${table_number}`;

    const available = !isReserved;
    const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
    lg' : 'shadow-md';
    const tableColor = available ? 'bg-green-500' : 'bg-red-400';
    const textColor = available ? 'text-white' : 'text-white';

    return (
      <div className={`relative w-full h-full rounded-full
      ${tableColor} flex items-center justify-center ${baseClasses}`}>
        { /* Стулья вокруг стола */ }
        <div className="absolute -top-4 left-1/2 -translate-x-1/2 w-
        4 h-4 bg-gray-500 rounded"></div>
        <div className="absolute -bottom-4 left-1/2 -translate-x-1/2
        w-4 h-4 bg-gray-500 rounded"></div>

        <div className="text-center">
          <div className={`text-xs font-medium
          ${textColor}`}>{displayText}</div>
          <div className={`text-xs ${textColor}`}>{maxGuests}
          {getGuestsWord(maxGuests)}</div>
        </div>
      </div>
    );
  };
};

```

```

export const renderCircularLargeTable = (props: TableRenderProps)
=> {
  const { table_number, maxGuests, isReserved = false, isSelected
= false, isAdmin = false } = props;
  const displayText = `${table_number}`;

  const available = !isReserved;
  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';
  const tableColor = available ? 'bg-green-500' : 'bg-red-400';
  const textColor = available ? 'text-white' : 'text-white';

  return (
    <div className={`relative w-full h-full rounded-full
${tableColor} flex items-center justify-center ${baseClasses}`}>
      { /* Большие диваны сверху и снизу */ }
      <div className="absolute -top-6 w-full h-4 bg-gray-500
rounded-lg"></div>
      <div className="absolute -bottom-6 w-full h-4 bg-gray-500
rounded-lg"></div>

      <div className="text-center">
        <div className={`text-xs font-medium
${textColor}`}>{displayText}</div>
        <div className={`text-xs ${textColor}`}>{maxGuests}
{getGuestsWord(maxGuests)}</div>
      </div>
    </div>
  );
};

export const renderRectangularTable = (props: TableRenderProps) =>
{
  const { table_number, maxGuests, isReserved = false, isSelected
= false, isAdmin = false } = props;
  const displayText = `${table_number}`;

  const available = !isReserved;
  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';
  const tableColor = available ? 'bg-green-500' : 'bg-red-400';
  const textColor = available ? 'text-white' : 'text-white';

  return (
    <div className={`relative w-full h-full ${tableColor} rounded
flex items-center justify-center ${baseClasses}`}>
      { /* Стулья сверху */ }
      {[...Array(4)].map((_, i) => (
        <div
          key={`top-${i}`}
          className="absolute w-4 h-4 bg-gray-500 rounded"
          style={{ top: '-1rem', left: `${(i + 1) * 20}%`,
transform: 'translateX(-50%)' }}

```

```

    />
  ))}

  {/* Стулья снизу */}
  [...Array(4)].map((_, i) => (
    <div
      key={`bottom-${i}`}
      className="absolute w-4 h-4 bg-gray-500 rounded"
      style={{ bottom: '-1rem', left: `${(i + 1) * 20}%`,
transform: 'translateX(-50%)' }}
    />
  ))}

  {/* Стул слева */}
  <div
    className="absolute w-4 h-4 bg-gray-500 rounded"
    style={{ left: '-1rem', top: '50%', transform:
'translateY(-50%)' }}
  />

  {/* Стул справа */}
  <div
    className="absolute w-4 h-4 bg-gray-500 rounded"
    style={{ right: '-1rem', top: '50%', transform:
'translateY(-50%)' }}
  />

  <div className="text-center">
    <div className={`text-xs font-medium
${textColor}`}>{displayText}</div>
    <div className={`text-xs ${textColor}`}>{maxGuests}
{getGuestsWord(maxGuests)}</div>
  </div>
</div>
);
};

export const renderVipTable = (props: TableRenderProps) => {
  const { table_number, maxGuests, isReserved = false, isSelected
= false, isAdmin = false } = props;
  const displayText = `${table_number}`;

  const available = !isReserved;
  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';

  return (
    <div className={`w-full h-full border-4 ${available ? 'border-
yellow-500 bg-yellow-100' : 'border-red-500 bg-red-100'} rounded-
lg flex items-center justify-center ${baseClasses}`}>
      <div className="text-center">
        <div className={`text-md font-bold ${available ? 'text-
yellow-800' : 'text-red-800'}`}>VIP</div>

```

```

        <div className={`text-xs ${available ? 'text-yellow-800' :
'text-red-800'}`}>{maxGuests} {getGuestWord(maxGuests)}</div>
      </div>
    </div>
  );
};

export const renderBanquetTable = (props: TableRenderProps) => {
  const { table_number, maxGuests, isReserved = false, isSelected
= false, isAdmin = false } = props;
  const displayText = `${table_number}`;

  const available = !isReserved;
  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';

  return (
    <div className={`border-4 ${available ? 'border-purple-500 bg-
purple-100' : 'border-red-500 bg-red-100'} rounded-lg flex items-
center justify-center ${baseClasses}`} style={{ width: '100%',
height: '100%' }}>
      {/* Multiple chairs on all sides */}
      <div className="absolute top-0 left-0 w-full">
        {[...Array(6)].map((_, i) => (
          <div
            key={`top-${i}`}
            className="absolute w-4 h-4 bg-gray-500 rounded"
            style={{ top: '-1rem', left: `${(i + 1) * 14}%`,
transform: 'translateX(-50%)' }}
          />
        ))}
      </div>

      <div className="absolute bottom-0 left-0 w-full">
        {[...Array(6)].map((_, i) => (
          <div
            key={`bottom-${i}`}
            className="absolute w-4 h-4 bg-gray-500 rounded"
            style={{ bottom: '-1rem', left: `${(i + 1) * 14}%`,
transform: 'translateX(-50%)' }}
          />
        ))}
      </div>

      <div className="absolute left-0 top-0 h-full">
        {[...Array(4)].map((_, i) => (
          <div
            key={`left-${i}`}
            className="absolute w-4 h-4 bg-gray-500 rounded"
            style={{ left: '-1rem', top: `${(i + 1) * 20}%`,
transform: 'translateY(-50%)' }}
          />
        ))}
      </div>
    </div>
  );
};

```

```

</div>

<div className="absolute right-0 top-0 h-full">
  {[...Array(4)].map((_, i) => (
    <div
      key={`right-${i}`}
      className="absolute w-4 h-4 bg-gray-500 rounded"
      style={{ right: '-1rem', top: `${(i + 1) * 20}%`,
transform: 'translateY(-50%)' }}
    />
  )]}
</div>

<div className="text-center">
  <div className={`text-md font-bold ${available ? 'text-
purple-800' : 'text-red-800'}`}>BAHKET</div>
  <div className={`text-xs ${available ? 'text-purple-800' :
'text-red-800'}`}>до {maxGuests} {getGuestWord(maxGuests)}</div>
</div>
</div>
);
};

export const renderDefaultTable = (props: TableRenderProps) => {
  const { table_number, maxGuests, isReserved = false, isSelected
= false, isAdmin = false } = props;
  const displayText = `${table_number}`;

  const available = !isReserved;
  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';
  const tableColor = available ? 'bg-green-500' : 'bg-red-400';
  const textColor = available ? 'text-white' : 'text-white';

  return (
    <div className={`${tableColor} rounded flex items-center
justify-center ${baseClasses}`} style={{ width: '100%', height:
'100%' }}>
      <div className="text-center">
        <div className={`text-xs ${textColor}`}>{maxGuests}
{getGuestsWord(maxGuests)}</div>
      </div>
    </div>
  );
};

// Main rendering function that selects appropriate table type
export const renderTableByType = (props: TableRenderProps) => {
  // Проверяем наличие и тип параметра type_name
  if (!props || !props.type_name || typeof props.type_name !==
'string') {
    return renderDefaultTable({
      type_name: 'default',

```

```

        table_number: props?.table_number || 0,
        maxGuests: props?.maxGuests || 2,
        isReserved: props?.isReserved || false,
        isSelected: props?.isSelected || false
    });
}

const { type_name } = props;
const tableType = type_name.toLowerCase();

switch (tableType) {
    case 'circular':
        return renderCircularTable(props);
    case 'circular-large':
        return renderCircularLargeTable(props);
    case 'rectangular':
        return renderRectangularTable(props);
    case 'vip':
        return renderVipTable(props);
    case 'banquet':
        return renderBanquetTable(props);
    default:
        return renderDefaultTable(props);
}
};

// Компонент для предпросмотра элементов в админ-панели
export const renderPreviewComponent = (component: any):
JSX.Element => {
    switch (component.type) {
        case 'circular':
            return (
                <div className="relative w-16 h-16 rounded-full bg-green-
500 flex items-center justify-center">
                    <div className="absolute -top-3 left-1/2 -translate-x-
1/2 w-3 h-3 bg-gray-500 rounded" />
                    <div className="absolute -bottom-3 left-1/2 -translate-
x-1/2 w-3 h-3 bg-gray-500 rounded" />
                    <div className="absolute -left-3 top-1/2 -translate-y-
1/2 w-3 h-3 bg-gray-500 rounded" />
                    <div className="absolute -right-3 top-1/2 -translate-y-
1/2 w-3 h-3 bg-gray-500 rounded" />
                    <span className="text-white font-bold text-
sm">{component.maxGuests}
{getGuestsWord(component.maxGuests)}</span>
                </div>
            );
        case 'circular-large':
            return (
                <div className="relative w-20 h-20 rounded-full bg-green-
500 flex items-center justify-center">

```

```

        <div className="absolute -top-4 w-full h-3 bg-gray-700
rounded-lg" />
        <div className="absolute -bottom-4 w-full h-3 bg-gray-
700 rounded-lg" />
        <span className="text-white font-bold text-
sm">{component.maxGuests}
{getGuestsWord(component.maxGuests)}</span>
    </div>
    );

    case 'rectangular':
        return (
            <div className="relative w-24 h-12 bg-green-500 rounded
flex items-center justify-center">
                {/* Стулья сверху и снизу */}
                [...Array(3)].map((_, i) => (
                    <React.Fragment key={i}>
                        <div
                            className="absolute w-3 h-3 bg-gray-500 rounded"
                            style={{ top: '-0.75rem', left: `${(i + 1) *
25}%`, transform: 'translateX(-50%)' }}
                        />
                        <div
                            className="absolute w-3 h-3 bg-gray-500 rounded"
                            style={{ bottom: '-0.75rem', left: `${(i + 1) *
25}%`, transform: 'translateX(-50%)' }}
                        />
                    </React.Fragment>
                )))

                {/* Стулья слева и справа */}
                <div
                    className="absolute w-3 h-3 bg-gray-500 rounded"
                    style={{ left: '-0.75rem', top: '50%', transform:
'translateY(-50%)' }}
                />
                <div
                    className="absolute w-3 h-3 bg-gray-500 rounded"
                    style={{ right: '-0.75rem', top: '50%', transform:
'translateY(-50%)' }}
                />

                <span className="text-white font-bold text-
sm">{component.maxGuests}
{getGuestsWord(component.maxGuests)}</span>
            </div>
        );

    case 'vip':
        return (
            <div className="w-24 h-24 border-3 border-yellow-500 bg-
yellow-100 rounded-lg flex items-center justify-center">

```

```

        <span className="text-yellow-800 font-bold text-sm">VIP</span>
      </div>
    );

    case 'bar':
      return (
        <div className="w-full h-full bg-red-100 border-l-4 border-red-600 flex items-center justify-center">
          <span className="text-red-700 font-semibold text-xs">Bap</span>
        </div>
      );

    case 'bathroom':
    case 'wc':
      return (
        <div className="w-16 h-16 bg-blue-100 border-2 border-blue-500 rounded flex items-center justify-center">
          <span className="text-blue-700 text-xs">WC</span>
        </div>
      );

    case 'kitchen':
      return (
        <div className="w-24 h-16 bg-red-100 border-2 border-red-600 flex flex-col items-center justify-center">
          <div className="text-red-800 font-bold text-sm">Кухня</div>
          <div className="text-xs text-red-500">служебная зона</div>
        </div>
      );

    case 'wardrobe':
      return (
        <div className="w-16 h-16 bg-purple-100 border-2 border-purple-500 rounded flex items-center justify-center">
          <span className="text-purple-700 text-xs">Гардероб</span>
        </div>
      );

    default:
      return (
        <div className="w-16 h-16 bg-gray-300 flex items-center justify-center">
          <span className="text-gray-700 text-xs">{component.title || component.label || component.type}</span>
        </div>
      );
  }
};

```

```

// Функция получения отображаемого имени типа стола
export const getDisplayTypeName = (type_name: string): string => {
  if (!type_name || typeof type_name !== 'string') {
    return 'Стол';
  }

  switch (type_name.toLowerCase()) {
    case 'circular': return 'Круглый';
    case 'circular-large': return 'Круглый большой';
    case 'rectangular': return 'Прямоугольный';
    case 'vip': return 'VIP';
    default: return type_name;
  }
};

// Function to properly render static items (not tables)
export const renderStaticItem = (type: string): JSX.Element => {
  switch (type.toLowerCase()) {
    case 'kitchen':
      return (
        <div className="w-full h-full bg-amber-100 border-2
border-amber-600 flex flex-col items-center justify-center">
          <div className="text-amber-800 font-bold text-
xs">Кухня</div>
          <div className="text-xs text-center text-amber-
600">служебная зона</div>
        </div>
      );

    case 'bathroom':
    case 'wc':
      return (
        <div className="w-full h-full bg-blue-100 border-2 border-
blue-500 rounded flex items-center justify-center">
          <span className="text-blue-700 font-bold text-
xs">WC</span>
        </div>
      );

    case 'bar':
      return (
        <div className="w-full h-full bg-red-100 border-1-4
border-red-600 flex items-center justify-center">
          <span className="text-red-700 font-semibold text-
xs">Бар</span>
        </div>
      );

    case 'wardrobe':
      return (
        <div className="w-full h-full bg-purple-100 border-2
border-purple-500 rounded flex items-center justify-center">

```

```

        <span className="text-purple-700 font-bold text-
xs">Гардероб</span>
      </div>
    );

    case 'window':
      return (
        <div className="w-full h-full border border-gray-300
rounded flex items-center justify-center">
          <div className="grid grid-cols-2 gap-1 w-full h-full p-
1">
            <div className="border border-gray-400 bg-blue-50
opacity-70"></div>
            <div className="border border-gray-400 bg-blue-50
opacity-70"></div>

            <div className="border border-gray-400 bg-blue-50
opacity-70"></div>

            <div className="border border-gray-400 bg-blue-50
opacity-70"></div>

          </div>
        </div>
      );

    default:

      return (
        <div className="w-full h-full bg-gray-100 border border-
gray-300 flex items-center justify-center">
          <span className="text-gray-700 text-xs">{type}</span>
        </div>
      );
    }
  };
};

```

Листинг Г.7 – Файл Wall.tsx

```

import React, { useState } from 'react';

interface WallProps {
  id: string;
  x: number;
  y: number;
  rotation: number;
  length: number;
  onClick?: (id: string) => void;
  isEditing?: boolean;
  isSelected?: boolean;
  scale?: number;
}

```

```

const Wall: React.FC<WallProps> = ({
  id,
  x,
  y,
  rotation,
  length,
  onClick,
  isEditing = false,
  isSelected = false,
  scale = 1,
}) => {
  const handleClick = (e: React.MouseEvent) => {
    e.stopPropagation();
    if (onClick) {
      onClick(id);
    }
  };

  const baseClasses = isSelected ? 'ring-2 ring-blue-500 shadow-
lg' : 'shadow-md';

  // Adjust position based on wall orientation to ensure proper
  centering
  let adjustedX = x;
  let adjustedY = y;

  // For vertical walls (90° or -90°)
  if (Math.abs(Math.abs(rotation) - 90) < 0.5) {
    adjustedX = x - 4; // Half of wall thickness to center it on
the point
  }
  // For horizontal walls (0° or 180°)
  else if (Math.abs(rotation) < 0.5 || Math.abs(Math.abs(rotation)
- 180) < 0.5) {
    adjustedY = y - 4; // Half of wall thickness to center it on
the point
  }

  return (
    <div
      style={{
        position: 'absolute',
        left: `${adjustedX * scale}px`,
        top: `${adjustedY * scale}px`,
        transform: `scale(${scale}) rotate(${rotation}deg)`,
        transformOrigin: 'top left',
        width: `${length}px`,
        height: '8px',
        cursor: isEditing ? 'move' : 'pointer',
        zIndex: isSelected ? 20 : 10,
      }}
      onClick={handleClick}
    >

```

```

        className={` ${baseClasses} bg-gray-800 rounded-sm`}
      />
    );
  };

};

export default Wall;

```

Листинг Г.8 – Файл Navbar.tsx

```

import React, { useState } from 'react';
import Link from 'next/link';
import { useRouter } from 'next/router';
import { useAuth } from '../context/AuthContext';
import { Bars3Icon, XMarkIcon } from
 '@heroicons/react/24/outline';

const Navbar: React.FC = () => {
  const { user, isAdmin, logout } = useAuth();
  const router = useRouter();
  const [isSidebarOpen, setIsSidebarOpen] = useState(false);

  const toggleSidebar = () => {
    setIsSidebarOpen(!isSidebarOpen);
  };

  const closeSidebar = () => {
    setIsSidebarOpen(false);
  };

  const handleLogout = () => {
    logout();
    closeSidebar();
  };

  const handleBookingClick = (e) => {
    e.preventDefault();
    if (!user) {
      router.push('/login');
    } else {
      router.push('/booking');
    }
    closeSidebar();
  };

  return (
    <>
      { /* Header */ }
      <header className="bg-white shadow-sm fixed w-full top-0 z-
50">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-
4 flex justify-between items-center">
          <div className="flex items-center">

```

```

        <button
          onClick={toggleSidebar}
          className="mr-4 text-white focus:outline-none"
          aria-label="Toggle menu"
        >
          <Bars3Icon className="h-6 w-6" />
        </button>
        <Link href="/" className="text-2xl font-bold text-
gray-800">
          Pure Heart
        </Link>
      </div>

      {user && (
        <div className="text-sm text-gray-600">
          {user.first_name} {user.last_name}
        </div>
      )}
    </div>
  </header>

  { /* Sidebar */}
  <div
    className={`fixed top-0 left-0 z-50 h-full w-64 bg-white
shadow-lg transform transition-transform duration-300 ease-in-out
${
  isSidebarOpen ? 'translate-x-0' : '-translate-x-full'
}`}
  >
    <div className="p-4">
      <div className="flex justify-between items-center mb-6">
        <h2 className="text-xl font-bold text-gray-800">Pure
Heart</h2>
        <button
          onClick={closeSidebar}
          className="text-white focus:outline-none"
          aria-label="Close menu"
        >
          <XMarkIcon className="h-6 w-6" />
        </button>
      </div>

      <nav className="mt-6">
        <ul className="space-y-4">
          <li>
            <Link
              href="/"
              className={`block py-2 px-4 rounded ${
                router.pathname === '/' ? 'bg-blue-50 text-
blue-600' : 'text-gray-600 hover:bg-gray-100'
              }`}
              onClick={closeSidebar}
            >

```

```

        Главная
      </Link>
    </li>
    <li>
      <Link
        href="/menu"
        className={`block py-2 px-4 rounded ${
          router.pathname === '/menu' ? 'bg-blue-50
text-blue-600' : 'text-gray-600 hover:bg-gray-100'
        }`}
        onClick={closeSidebar}
      >
        Меню ресторана
      </Link>
    </li>
    <li>
      <a
        href="#"
        className={`block py-2 px-4 rounded ${
          router.pathname === '/booking' ? 'bg-blue-50
text-blue-600' : 'text-gray-600 hover:bg-gray-100'
        }`}
        onClick={handleBookingClick}
      >
        Забронировать стол
      </a>
    </li>

    {user && (
      <>
        <li>
          <Link
            href="/account"
            className={`block py-2 px-4 rounded ${
              router.pathname === '/account' ? 'bg-blue-
50 text-blue-600' : 'text-gray-600 hover:bg-gray-100'
            }`}
            onClick={closeSidebar}
          >
            Мои заказы
          </Link>
        </li>
        {isAdmin && (
          <>
            <li>
              <Link
                href="/admin"
                className={`block py-2 px-4 rounded ${
                  router.pathname === '/admin' ? 'bg-
blue-50 text-blue-600' : 'text-gray-600 hover:bg-gray-100'
                }`}
                onClick={closeSidebar}
              >

```

```

                Панель управления
            </Link>
        </li>
    </>
    )}
</>

    )}

    </ul>
</nav>
{user ? (
    <div className="mt-8 pt-4 border-t border-gray-200">
        <button
            onClick={handleLogout}
            className="w-full text-left py-2 px-4 rounded
text-red-700 hover:bg-red-50"
        >
            Выход
        </button>
    </div>
    ) : (
    <div className="mt-8 pt-4 border-t border-gray-200
space-y-4">
        <Link
            href="/login"
            className="block py-2 px-4 rounded text-blue-600
hover:bg-blue-50"
            onClick={closeSidebar}
        >
            Вход
        </Link>
        <Link
            href="/register"
            className="block py-2 px-4 rounded text-blue-600
hover:bg-blue-50"
            onClick={closeSidebar}
        >
            Регистрация
        </Link>
    </div>
    )}
</div>
</div>

{/* Overlay */}
{isSidebarOpen &&
(
    <div
        className="fixed inset-0 bg-black bg-opacity-50 z-40"
        onClick={closeSidebar}
    ></div>
    )}

```

```

    </>

    );

};

export default Navbar;

```

Листинг Г.9 – Файл admin.tsx

```

import React, { useState, useEffect } from 'react';
import Head from 'next/head';
import { useRouter } from 'next/router';
import { useAuth } from '../context/AuthContext';
import LayoutEditor from '../components/Editor';
import MenuEditor from '../components/MenuEditor';
import ReservationsManager from
'../components/ReservationsManager';
import Navbar from '../components/Navbar';

const AdminPage: React.FC = () => {
  const { user, loading, isAdmin } = useAuth();
  const router = useRouter();
  const [activeTab, setActiveTab] = useState<'layout' | 'menu' |
'reservations'>('layout');

  useEffect(() => {
    // Redirect if not admin
    if (!loading && (!user || !isAdmin)) {
      router.push('/');
    }
  }, [user, loading, isAdmin, router]);

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-
center">
        <p className="text-gray-500">Загрузка...</p>
      </div>
    );
  }

  if (!isAdmin) {
    return null;
  }

  return (
    <>
      <Head>

```

```

    <title>Панель администратора - Pure Heart</title>
  </Head>

  <Navbar />

  <main className="pt-20 pb-10 px-4 sm:px-6 lg:px-8">
    <div className="max-w-7xl mx-auto">
      <header className="mb-8 flex justify-between items-
center">
        <h1 className="text-2xl font-bold text-gray-
900">Панель администратора</h1>
      </header>

      <div className="bg-white shadow rounded-lg overflow-
hidden mb-6">
        <nav className="flex border-b border-gray-200">
          <button
            onClick={() => setActiveTab('layout')}
            className={`py-4 px-6 font-medium text-sm mb-3
mr-3 ${
              activeTab === 'layout'
                ? 'border-b-2 border-blue-500 text-white mb-3
mr-3'
                : 'text-gray-600 hover:text-gray-700
hover:border-gray-300'
            }`}
          >
            План зала
          </button>
          <button
            onClick={() => setActiveTab('menu')}
            className={`py-4 px-6 font-medium text-sm mb-3
mr-3 ${
              activeTab === 'menu'
                ? 'border-b-2 border-blue-500 text-white '
                : 'text-gray-600 hover:text-gray-700
hover:border-gray-300'
            }`}
          >
            Меню
          </button>
          <button
            onClick={() => setActiveTab('reservations')}
            className={`py-4 px-6 font-medium text-sm mb-3
mr-3 ${
              activeTab === 'reservations'
                ? 'border-b-2 border-blue-500 text-white'
                : 'text-gray-600 hover:text-gray-700
hover:border-gray-300'
            }`}
          >
            Резервации
          </button>
        </nav>
      </div>
    </div>
  </main>

```

```

        >
        Бронирования
      </button>
    </nav>

    <div className="p-6">
      {activeTab === 'layout' && (
        <div>
          <h2 className="text-xl font-semibold mb-4">Редактирование плана зала</h2>
          <LayoutEditor />
        </div>
      )}

      {activeTab === 'menu' && (
        <div>
          <h2 className="text-xl font-semibold mb-4">Управление меню</h2>
          <MenuEditor />
        </div>
      )}

      {activeTab === 'reservations' && (
        <div>
          <h2 className="text-xl font-semibold mb-4">Управление бронированиями</h2>
          <ReservationsManager />
        </div>
      )}
    </div>
  </div>
</main>
</>
);
};

export default AdminPage;

```

Листинг Г.10 – Файл register.tsx

```

import React, { useState, useEffect } from 'react';
import Head from 'next/head';
import { useRouter } from 'next/router';
import { useAuth } from '../context/AuthContext';
import { formatPhoneNumber, normalizePhoneNumber, isValidPhoneNumber } from '../lib/formatters';
import Navbar from '../components/Navbar';

export default function Register() {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
  });

```

```

    first_name: '',
    last_name: '',
    phone: '',
  });
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [phoneError, setPhoneError] = useState<string |
null>(null);

  const router = useRouter();
  const { user, register } = useAuth();

  useEffect(() => {
    // Redirect if already logged in
    if (user) {
      router.push('/');
    }
  }, [user, router]);

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) =>
{
  const { name, value } = e.target;

  if (name === 'phone') {
    // Format the phone number
    const formattedPhone = formatPhoneNumber(value);
    setFormData((prev) => ({
      ...prev,
      [name]: formattedPhone,
    }));
    if (isValidPhoneNumber(formattedPhone) ||
formattedPhone === '') {
      setPhoneError(null);
    } else if (formattedPhone.length >= 18) {
      // Only show error when user has entered a complete number
      setPhoneError('Введите номер в формате +7 (XXX) XXX-XX-
XX');
    }
  } else {
    setFormData((prev) => ({
      ...prev,
      [name]: value,
    }));
  }
};

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    if (formData.phone && !isValidPhoneNumber(formData.phone)) {
      setPhoneError('Введите корректный номер телефона');
      return;
    }
  }

```

```

try {
  setLoading(true);
  setError(null);

  // Normalize phone number before sending to API
  const dataToSubmit = {
    ...formData,
    phone: normalizePhoneNumber(formData.phone),
  };

  await register(dataToSubmit);
  router.push('/');
} catch (err: any) {
  console.error('Registration failed:', err);
  setError(err.response?.data?.detail || 'Ошибка при
регистрации');
} finally {
  setLoading(false);
}
};

return (
  <>
    <Head>
      <title>Регистрация - Pure Heart</title>
    </Head>

    <Navbar />

    <main className="min-h-screen flex items-center justify-
center bg-gray-50 py-12 px-4 sm:px-6 lg:px-8">
      <div className="max-w-md w-full space-y-8">
        <div>
          <h2 className="mt-6 text-center text-3xl font-
extrabold text-gray-900">
            Регистрация
          </h2>
          <p className="mt-2 text-center text-sm text-gray-600">
            Уже есть аккаунт?{' '}
            <a href="/login" className="font-medium text-blue-
600 hover:text-blue-500">
              Войдите
            </a>
          </p>
        </div>

        {error && (
          <div className="bg-red-100 border border-red-400 text-
red-700 px-4 py-3 rounded relative" role="alert">
            <span className="block sm:inline">{error}</span>
          </div>
        )}
      </div>
    </main>
  </>
);

```

```

        <form className="mt-8 space-y-6"
onSubmit={handleSubmit}>
        <div className="space-y-4">
        <div>
            <label htmlFor="email" className="block text-sm
font-medium text-gray-700">
                Email
            </label>
            <input
                id="email"
                name="email"
                type="email"
                autoComplete="email"
                required
                value={formData.email}
                onChange={handleChange}
                className="mt-1 focus:ring-blue-500
focus:border-blue-500 block w-full shadow-sm sm:text-sm border-
gray-300 rounded-md p-2 border"
            />
        </div>

        <div>
            <label htmlFor="password" className="block text-sm
font-medium text-gray-700">
                Пароль
            </label>
            <input
                id="password"
                name="password"
                type="password"
                autoComplete="new-password"
                required
                value={formData.password}
                onChange={handleChange}
                className="mt-1 focus:ring-blue-500
focus:border-blue-500 block w-full shadow-sm sm:text-sm border-
gray-300 rounded-md p-2 border"
            />
        </div>

        <div>
            <label htmlFor="first_name" className="block text-
sm font-medium text-gray-700">
                Имя
            </label>
            <input
                id="first_name"
                name="first_name"
                type="text"
                autoComplete="given-name"
                value={formData.first_name}

```

```

        onChange={handleChange}
        className="mt-1 focus:ring-blue-500
focus:border-blue-500 block w-full shadow-sm sm:text-sm border-
gray-300 rounded-md p-2 border"
      />
    </div>

    <div>
      <label htmlFor="last_name" className="block text-sm
font-medium text-gray-700">
        Фамилия
      </label>
      <input
        id="last_name"
        name="last_name"
        type="text"
        autoComplete="family-name"
        value={formData.last_name}
        onChange={handleChange}
        className="mt-1 focus:ring-blue-500
focus:border-blue-500 block w-full shadow-sm sm:text-sm border-
gray-300 rounded-md p-2 border"
      />
    </div>

    <div>
      <label htmlFor="phone" className="block text-sm
font-medium text-gray-700">
        Телефон
      </label>
      <input
        id="phone"
        name="phone"
        type="tel"
        autoComplete="tel"
        value={formData.phone}
        onChange={handleChange}
        placeholder="+7 (XXX) XXX-XX-XX"
        className={`mt-1 focus:ring-blue-500
focus:border-blue-500 block w-full shadow-sm sm:text-sm border-
gray-300 rounded-md p-2 border ${phoneError ? 'border-red-500' :
''}}`
      />
      {phoneError && (
        <p className="mt-1 text-sm text-red-
600">{phoneError}</p>
      )}
    </div>

  </div>

  <div>
    <button

```

```

        type="submit"
        disabled={loading || !!phoneError}
        className="group relative w-full flex justify-
center py-2 px-4 border border-transparent text-sm font-medium
rounded-md text-white bg-blue-600 hover:bg-blue-700 focus:outline-
none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500
disabled:bg-blue-300"
      >
        {loading ? 'Регистрация...' :
'Зарегистрироваться'}
      </button>
    </div>
  </form>
</div>
</main>
</>
);
}

```