ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

Домашняя работа №4 по дисциплине “Проектирование архитектуры программных систем”

Проектирование
Структура и поведение системы

Выполнили студенты группы БПИ223
Абдуллаев А.Ш.
Жалилов А.
Курманова А.

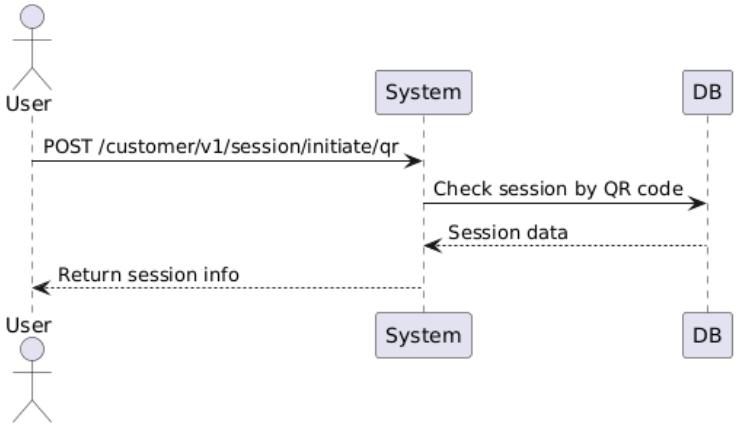
Москва 2025

Содержание

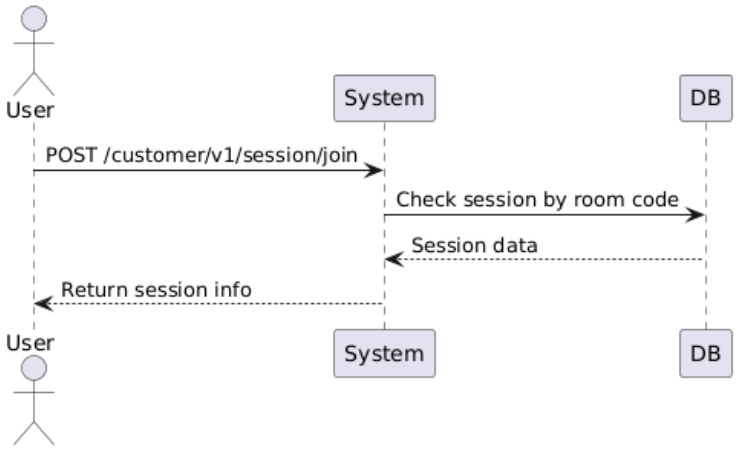
Диаграмма последовательности

Telegram Mini-App

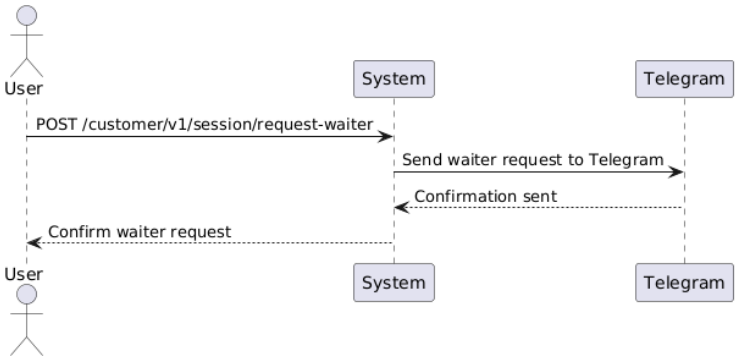
1. Инициализация сессии (по QR коду)



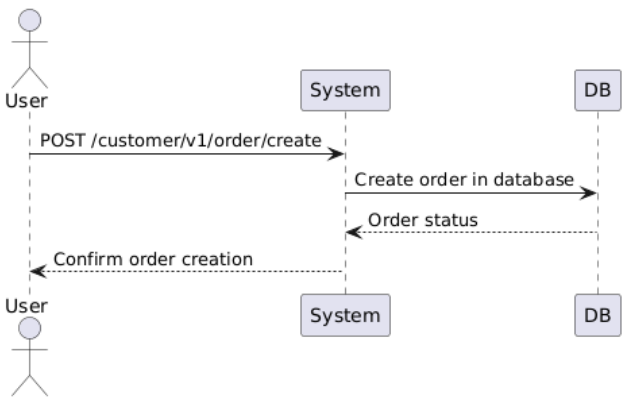
2. Присоединение к сессии. Ввод кода сессии.



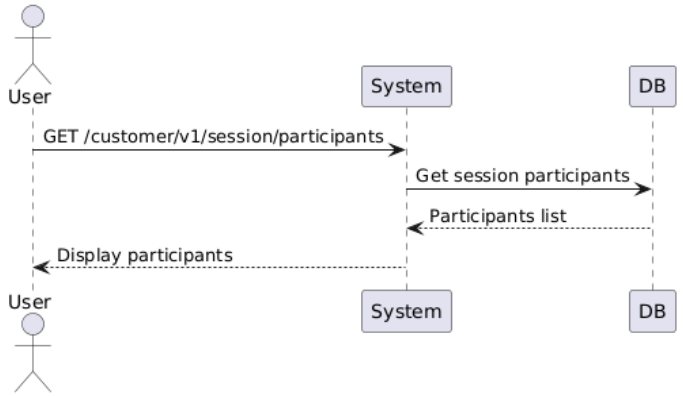
3. Вызов официанта



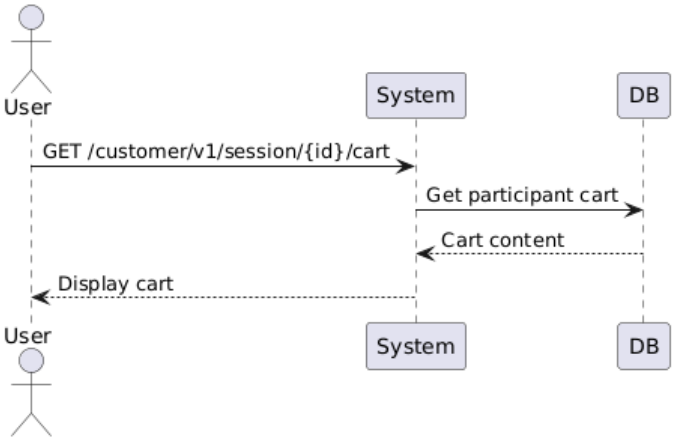
4. Оформление заказа



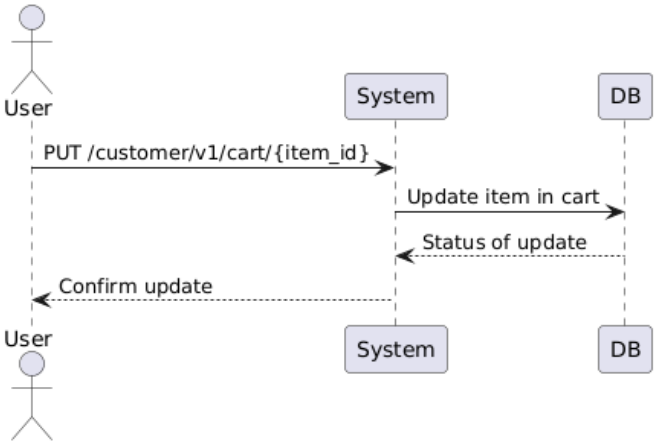
5. Просмотр каждого участника сессии



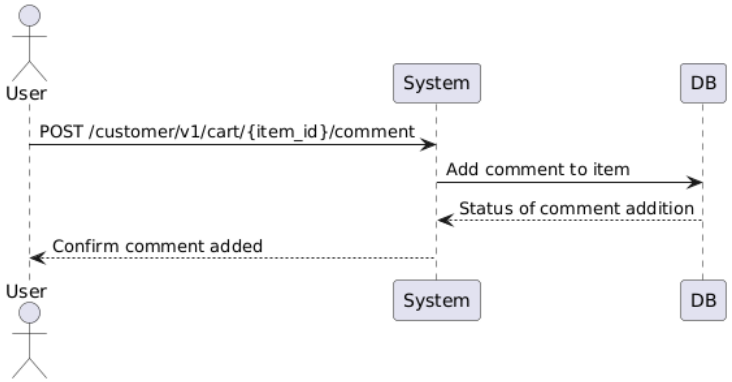
6. Просмотр корзины каждого участника



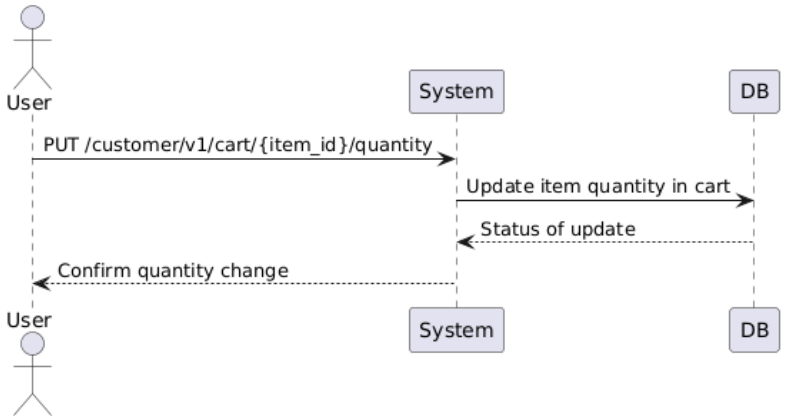
7. Редактирование позиции в меню внутри корзины



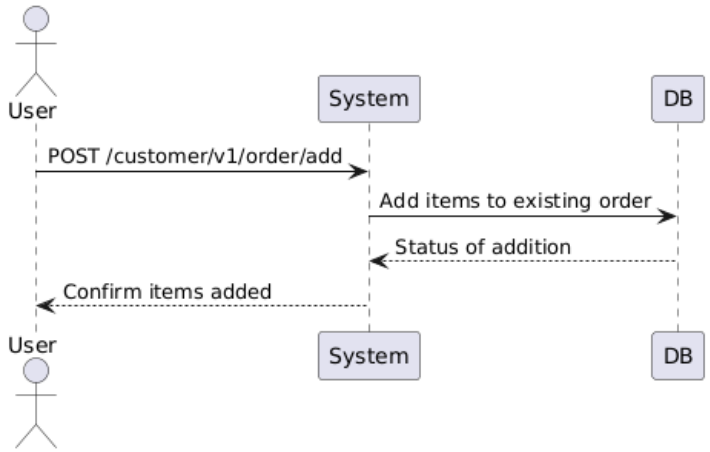
8. CRUD комментария к позиции меню



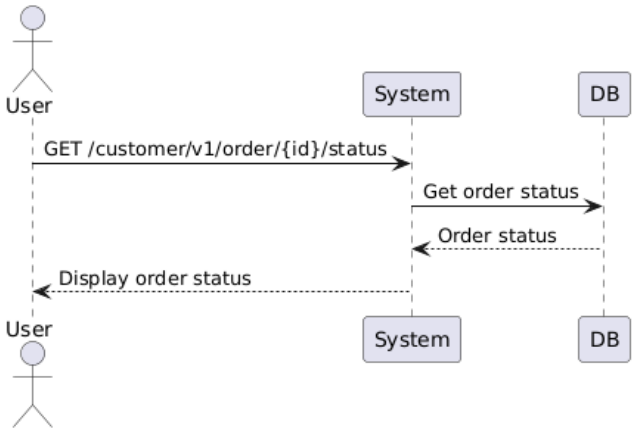
9. Редактирование количества позиций меню внутри корзины



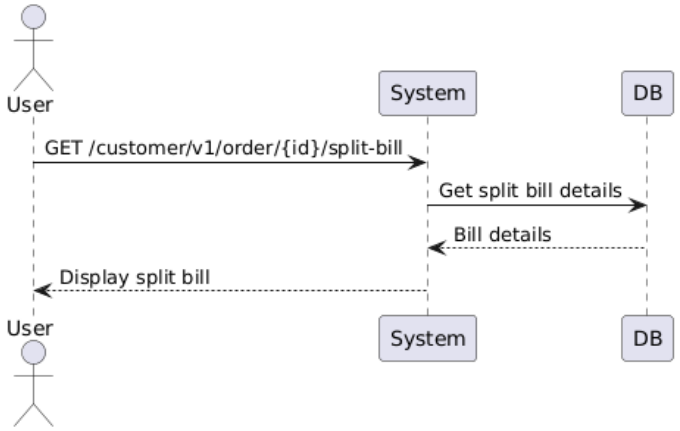
10. Дозаказ



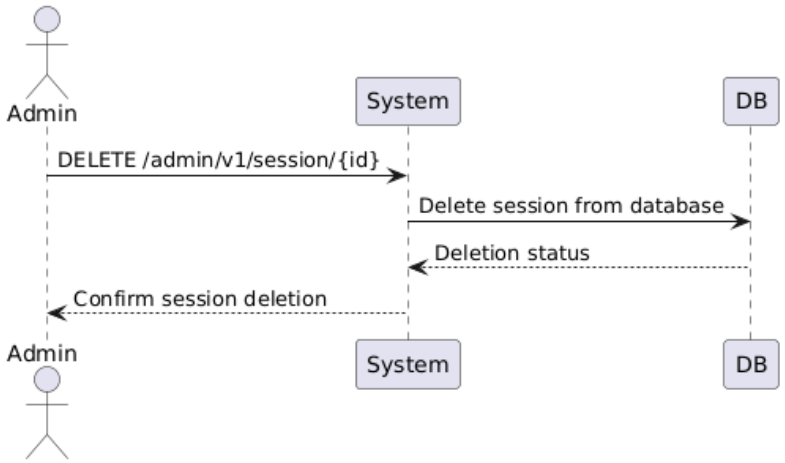
11. Просмотр статуса заказа



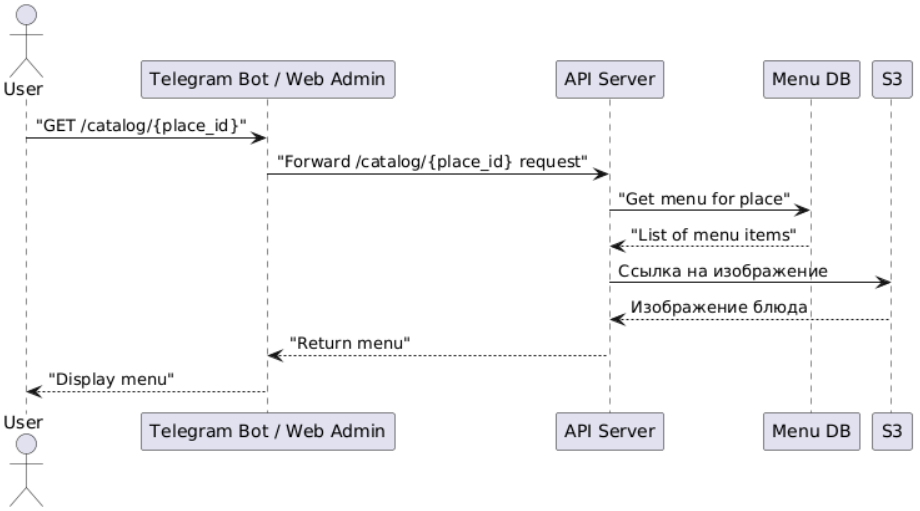
12. Просмотр разделенного общего чека



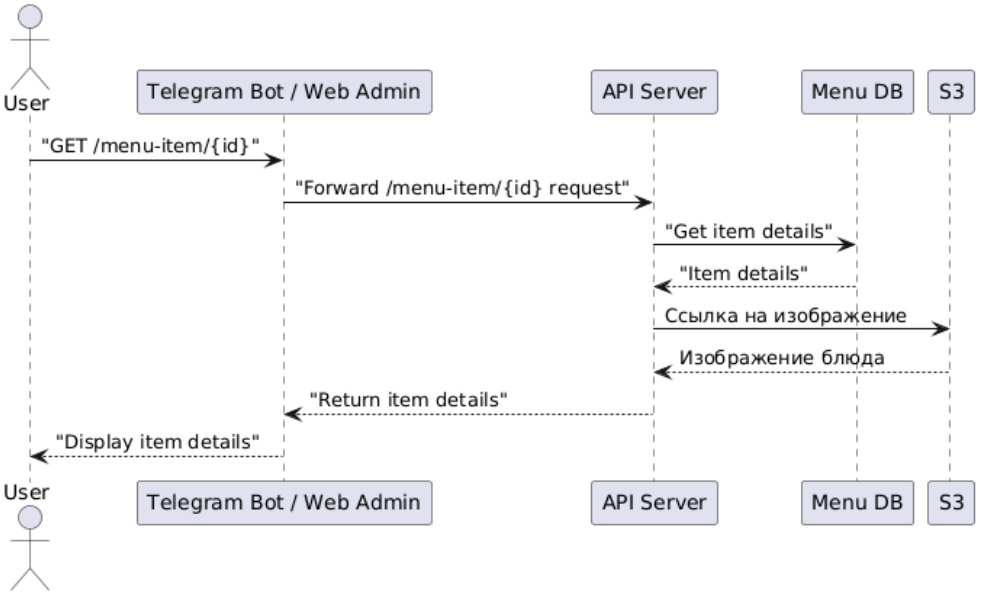
13. Удаление сессии



14. Просмотр каталога плейса

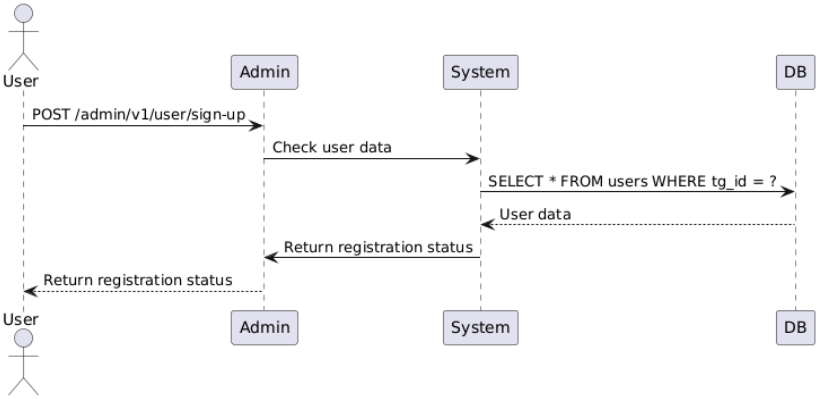


15. Просмотр детальной карточки позиции в меню

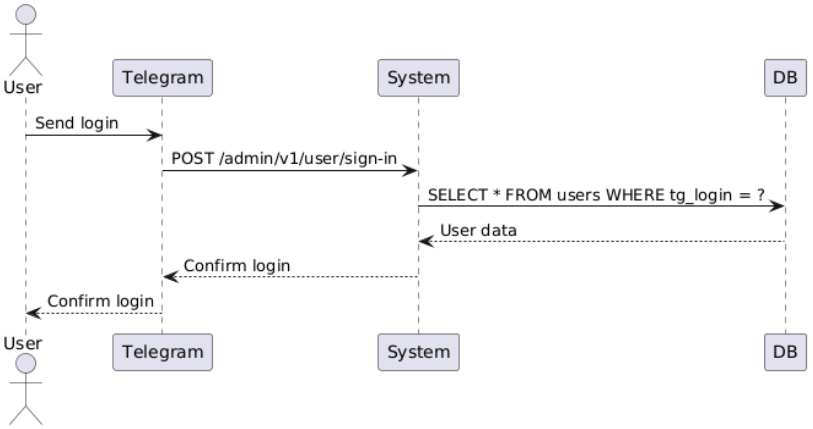


Административная панель

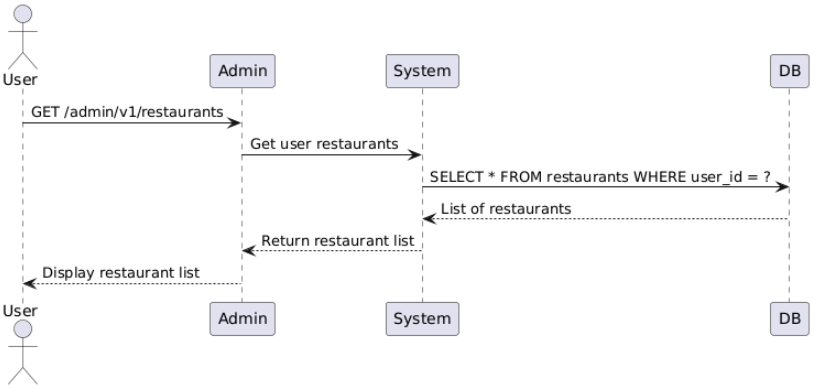
1. Регистрация/Авторизация пользователя



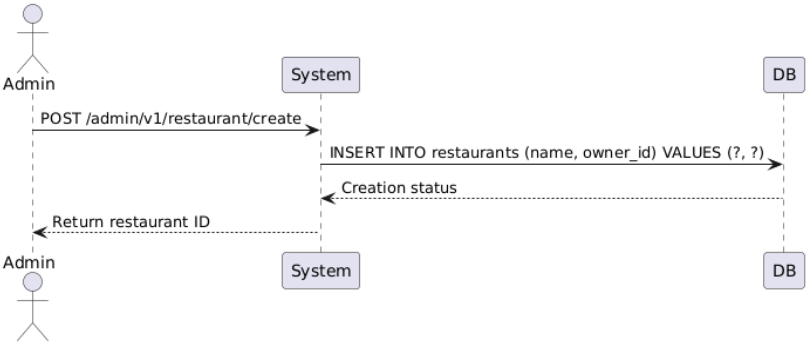
2. Подтверждение телеграмм-логина



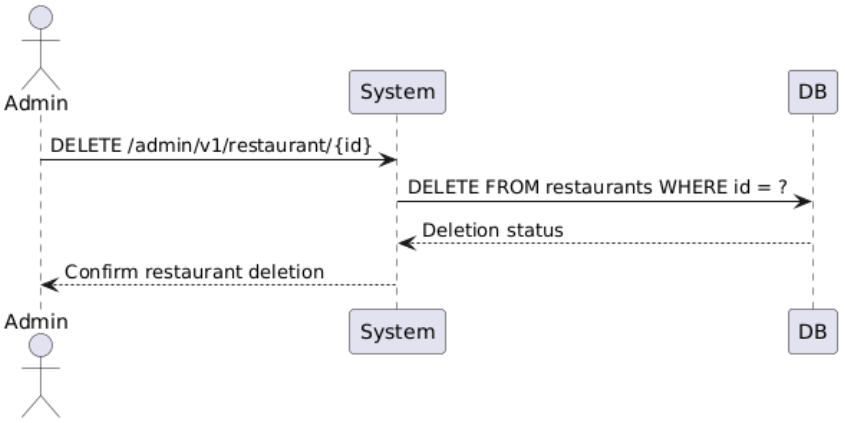
3. Отображение списка ресторанов пользователя



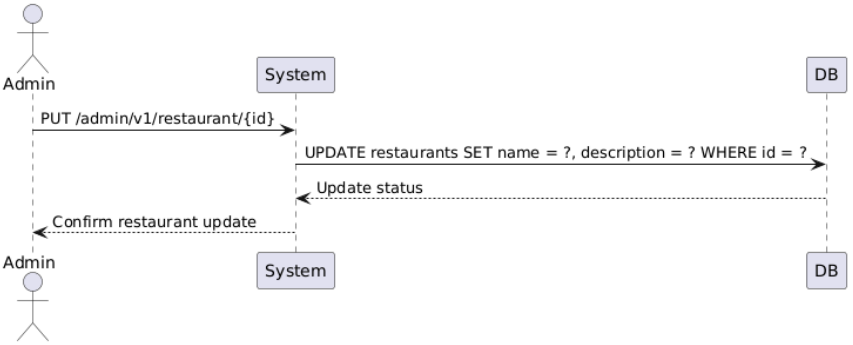
4. Создание ресторана



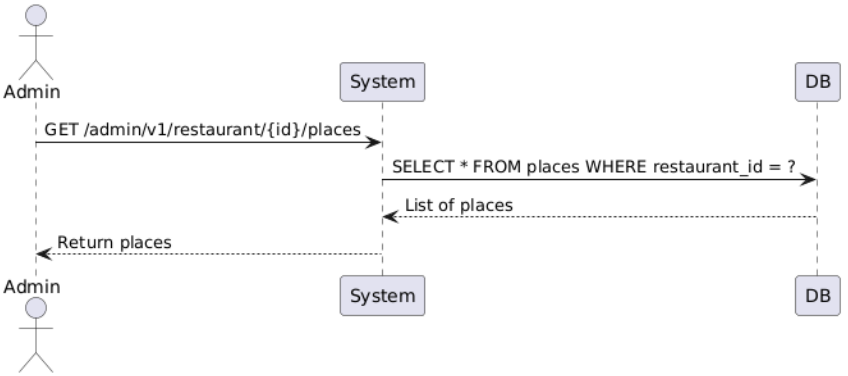
5. Удаление ресторана



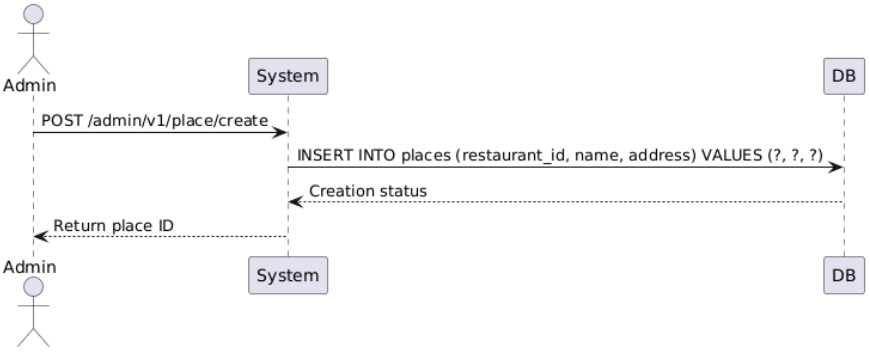
6. Редактирование ресторана



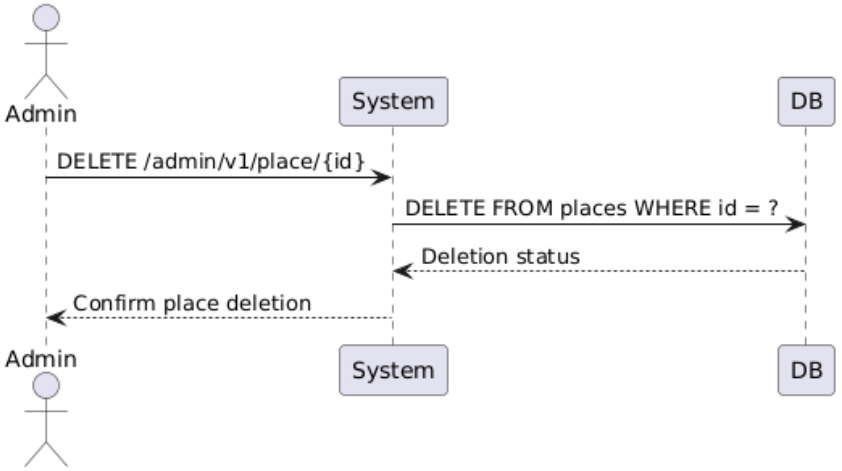
7. Отображение списка плейсов ресторана



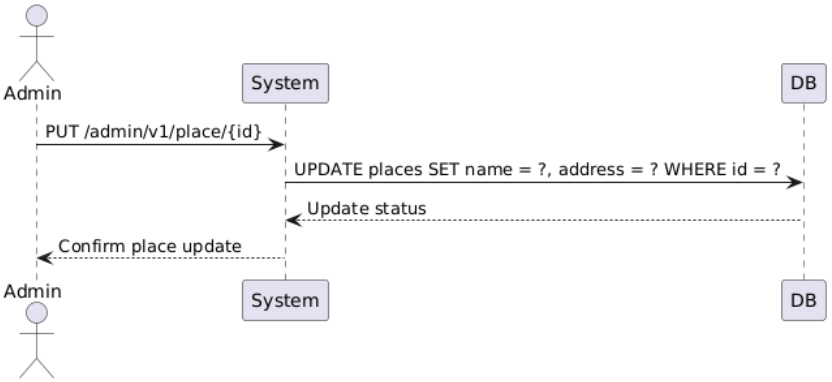
8. Создание плейса



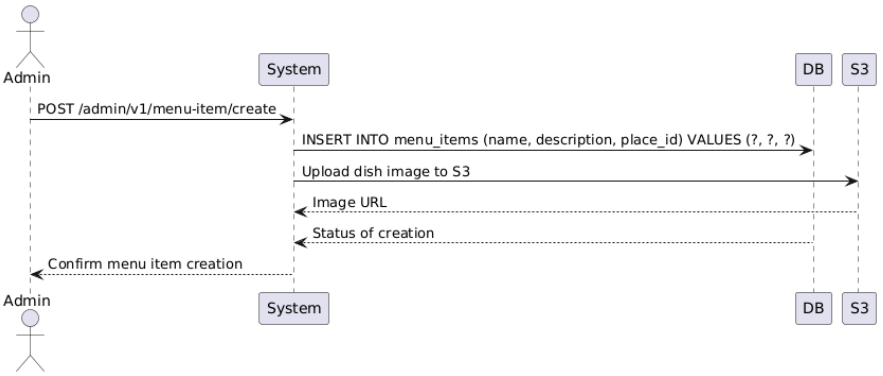
9. Удаление плейса



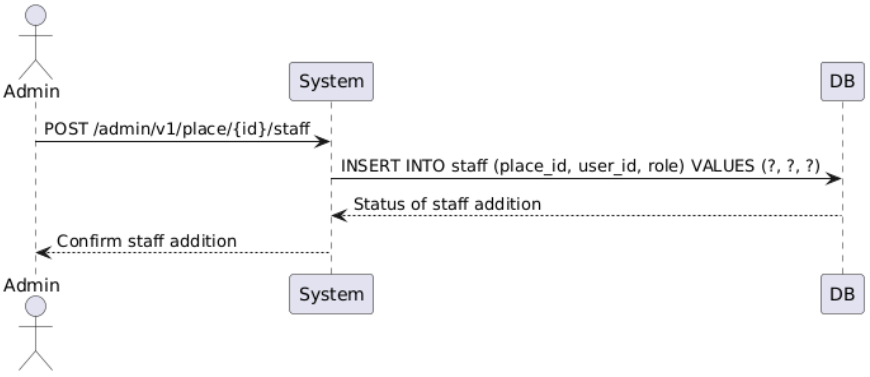
10. Редактирование плейса



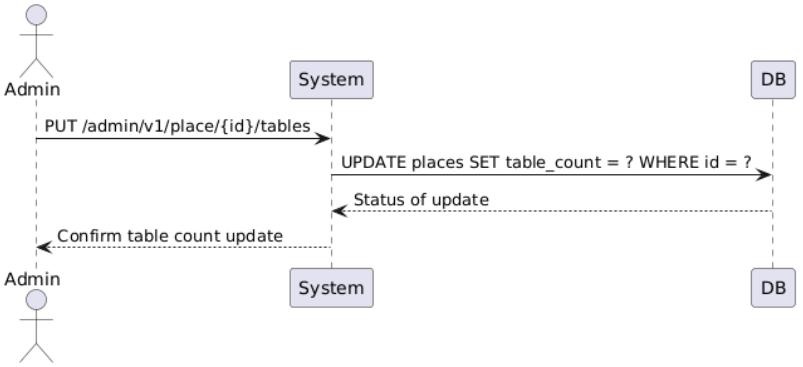
11. CRUD позиции меню плейса



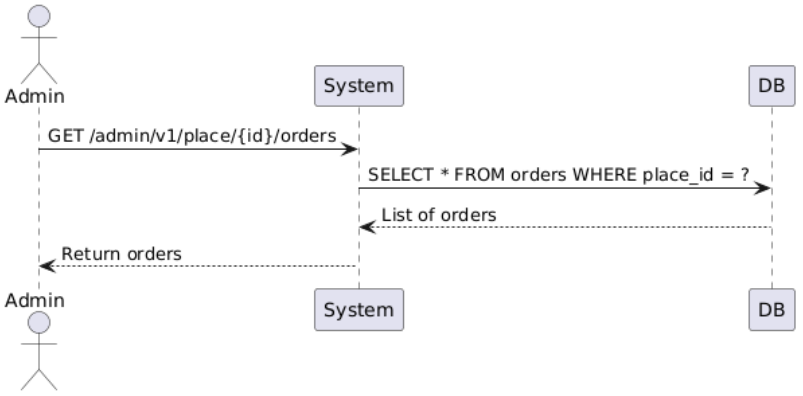
12. CRUD состава сотрудников



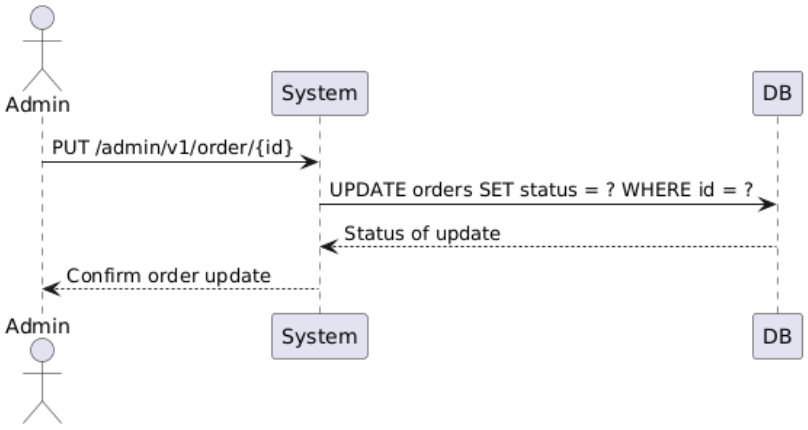
13. Редактирование количества столов



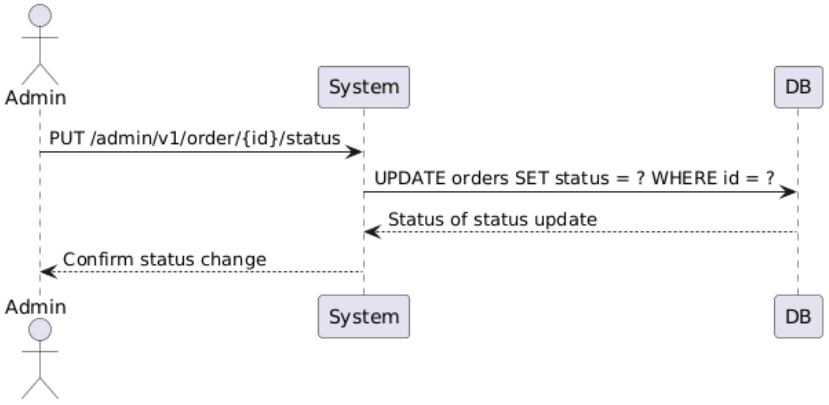
14. Просмотр текущих заказов плейса



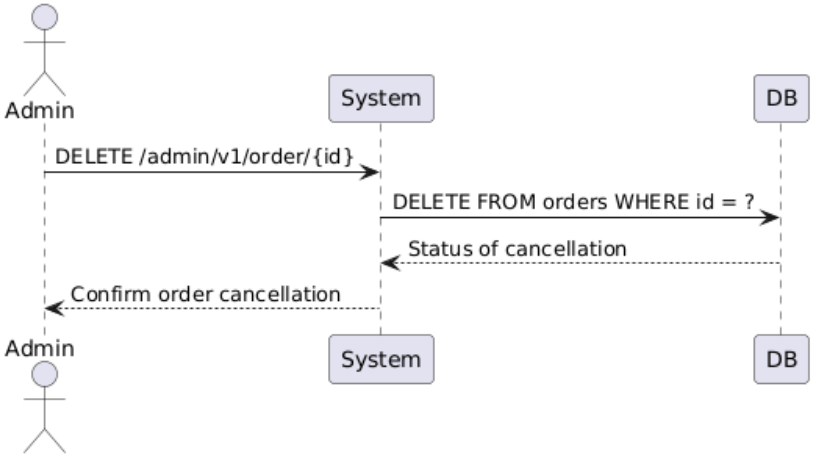
15. Редактирование текущих заказов плейса



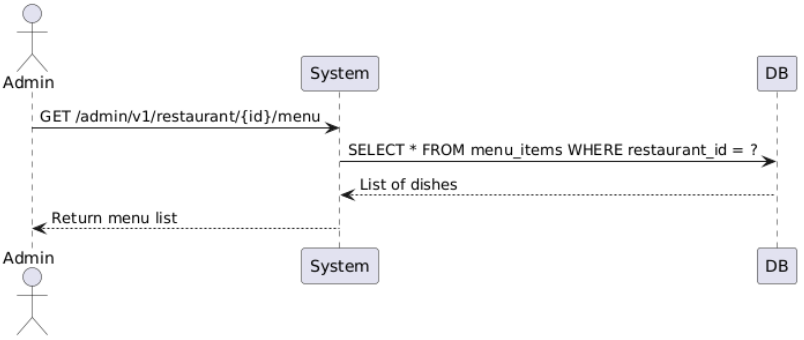
16. Редактирование статуса заказа



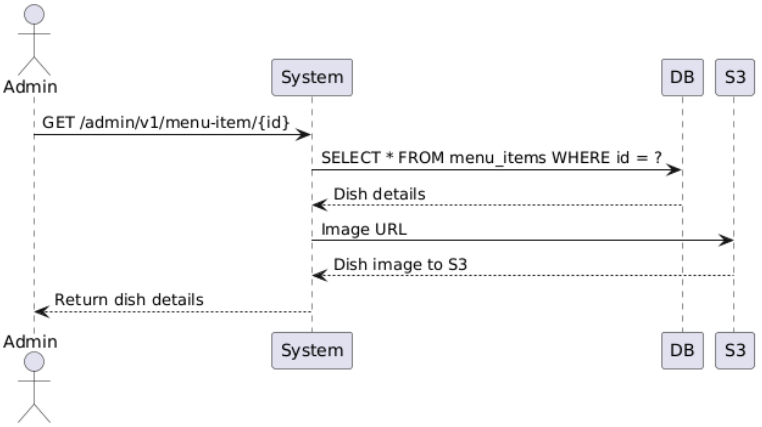
17. Отмена заказа



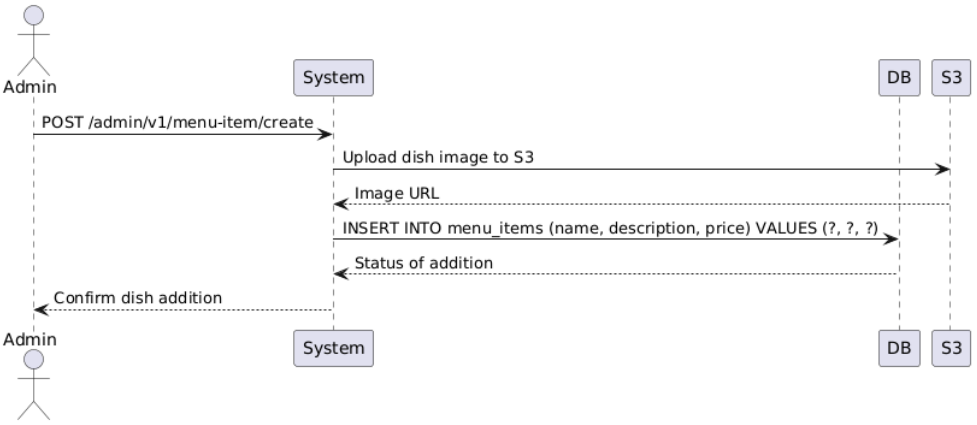
18. Просмотр общего списка блюд ресторана



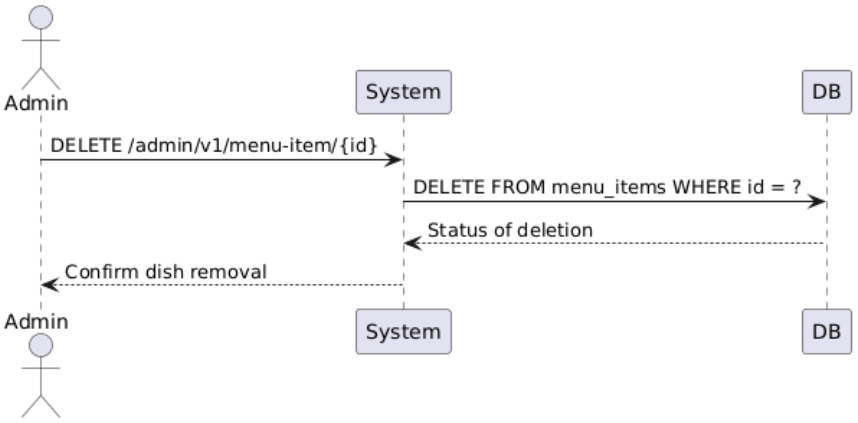
19. Просмотр блюда



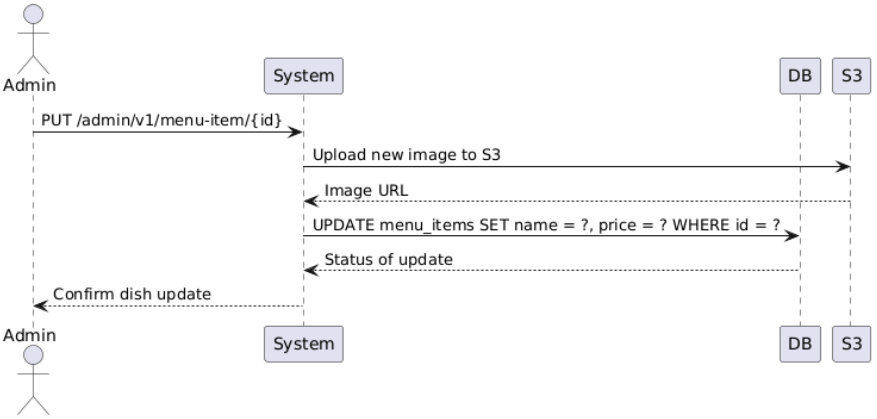
20. Добавление блюда



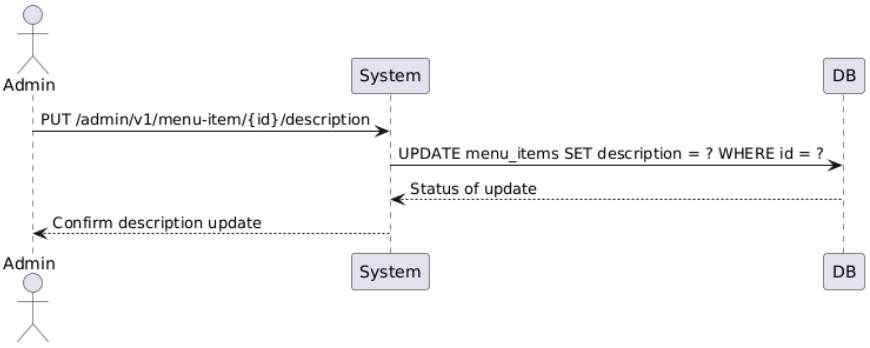
21. Удаление блюда



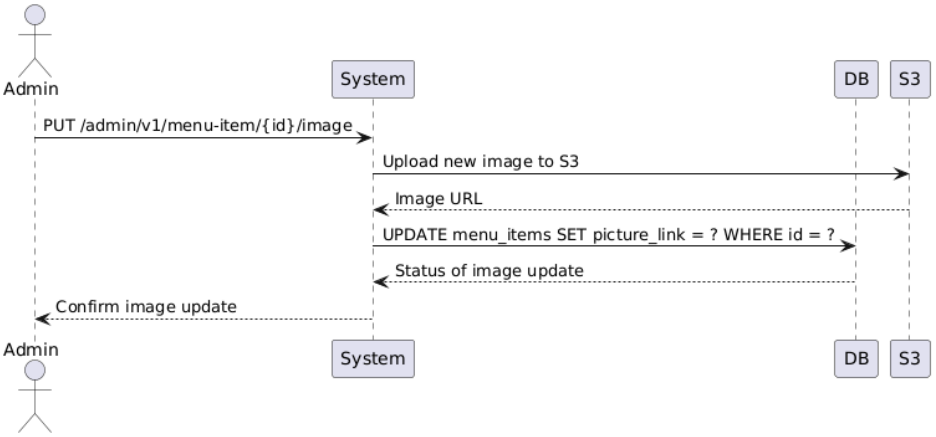
22. Редактирование блюда



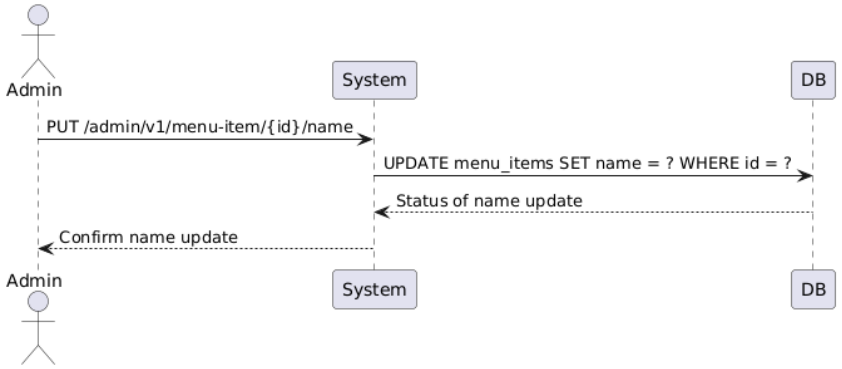
23. Редактирование описания



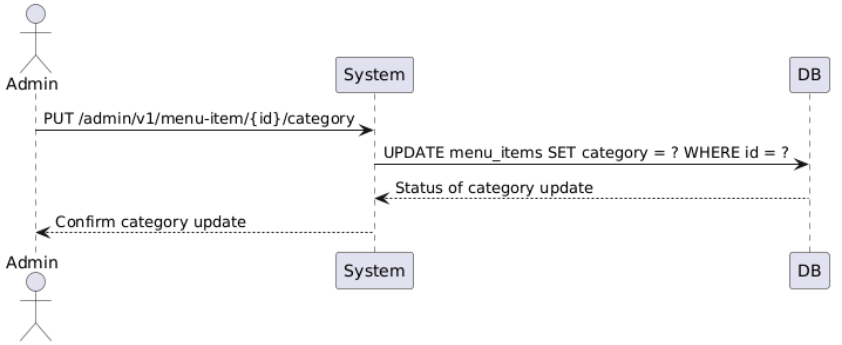
24. Редактирование изображения



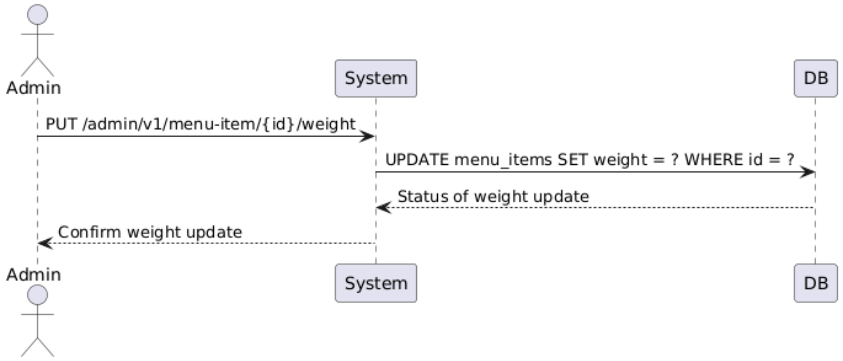
25. Редактирование названия



26. Редактирование категории



27. Редактирование веса



ЗАР по используемым в проекте коннекторам

1. **Авторы:** Абдуллаев Аюбхон, Жалилов Актан, Курманова Амира
2. **Дата создания:** 21.04.2025
3. **Контекст решения:**

Проект SmartTable разрабатывает систему для упрощения процесса заказа и обслуживания в ресторанах, с включением админки для ресторанов и Telegram-бота для пользователей. Система использует монолитную архитектуру, где все компоненты и бизнес-логика сосредоточены в одном сервисе. Взаимодействие между компонентами происходит через внутренние вызовы и API (интерфейсных классов).

Для организации бизнес-логики используется DDD (Domain-Driven Design), что позволяет четко разделить домены и обеспечивать масштабируемость и гибкость. Взаимодействие с Telegram-ботом осуществляется через цепочку обязанностей (Chain of Responsibility) для обработки различных типов сообщений. Для улучшения тестируемости и масштабируемости системы используется Dependency Injection (DI). Аутентификация и управление доступом реализованы с помощью JWT (JSON Web Tokens), что обеспечивает безопасность и простоту работы с токенами.

Система имеет следующие основные цели:

- Упрощение процесса заказа для посетителей.
- Повышение удобства для ресторанов через централизованное управление заказами и меню.
- Использование Telegram-бота для упрощенного взаимодействия с клиентами.
- Управление заказами, меню и пользователями через административную панель.

Все компоненты системы интегрированы внутри одного сервиса, что упрощает архитектуру и снижает сложность взаимодействия.

4. **Решения**

4.1. Используемые паттерны:

4.1.1. Монолитная архитектура

Вся система разрабатывается как единственный сервис, где взаимодействие между компонентами происходит через внутренние API вызовы.

4.1.1.1. Причины использования:

- Простота в разработке и развертывании на старте проекта.
- Быстрая разработка функционала без необходимости интеграции микросервисов.
- Все данные централизованы в одном месте, что упрощает управление.

4.1.1.2. Плюсы:

- Простота реализации и тестирования.
- Легкость в поддержке, все внутри одного приложения.

4.1.1.3. Минусы:

- Возможные проблемы с масштабированием по мере роста нагрузки.

- Потенциально высокая зависимость между компонентами, что усложняет обновления.

4.1.2. Repository Pattern (Паттерн репозитория)

Использование репозитория для изоляции бизнес-логики от реализации доступа к данным в базе данных.

4.1.2.1. **Причины использования**:

- Упрощение тестирования бизнес-логики.
- Легкость в изменении способа хранения данных без затрагивания основной логики.

4.1.2.2. **Плюсы**:

- Чистая и модульная архитектура.
- Легкость в добавлении новых типов данных.

4.1.2.3. **Минусы**:

- Необходимость поддерживать дополнительные абстракции, что усложняет код.

4.1.3. Facade Pattern (Паттерн фасада)

Использование фасадов для предоставления простого интерфейса для взаимодействия с внешними сервисами (например, Telegram API и S3).

4.1.3.1. **Причины использования**:

- Изоляция сложных взаимодействий с внешними сервисами.
- Упрощение взаимодействия с внешними API.

4.1.3.2. **Плюсы**:

- Упрощает интеграцию внешних сервисов.
- Обеспечивает простоту изменения внешних API.

4.1.3.3. **Минусы**:

- Может привести к дублированию кода при добавлении новых внешних сервисов.

4.1.4. Chain of Responsibility Pattern (Паттерн цепочки обязанностей)

Для обработки сообщений от пользователей через **Telegram-бота** используется **цепочка обязанностей**. Каждое сообщение, поступающее от пользователя, обрабатывается в зависимости от типа запроса.

4.1.4.1. **Причины использования**:

- Упрощает добавление новых типов сообщений или команд, не затрагивая основную логику.
- Позволяет гибко управлять процессом обработки сообщений.

4.1.4.2. **Плюсы**:

- Легкость в расширении и добавлении новых обработчиков.
- Чистота и модульность кода.

4.1.4.3. **Минусы**:

- При сложной логике обработки могут возникнуть проблемы с отладкой и трассировкой.

4.2. Используемые коннекторы

4.2.1. Коннектор для облачного хранилища (S3)

Для хранения изображений блюд используется **Amazon S3**.

4.2.1.1. **Плюсы**:

- Высокая доступность и масштабируемость.
- Надежность и возможность хранения больших объемов медиафайлов.

4.2.1.2. **Минусы**:

- Зависимость от внешнего облачного сервиса.
- Потенциальное увеличение задержек при работе с файлами.

4.2.2. Коннектор для Telegram API

Взаимодействие с **Telegram Bot API** для отправки сообщений и обработки запросов от пользователей через Telegram.

4.2.2.1. **Плюсы**:

- Простота в интеграции и использовании.
- Популярность Telegram как мессенджера среди пользователей.

4.2.2.2. **Минусы**:

- Ограничения на скорость запросов и количество вызовов API.

4.2.3. Dependency Injection (DI)

Используется для инъекции зависимостей в сервисы, что позволяет улучшить тестируемость и масштабируемость.

4.2.3.1. **Плюсы**:

- Улучшает тестируемость и читаемость кода.
- Упрощает поддержку и изменение зависимостей.

4.2.3.2. **Минусы**:

- Может добавить дополнительную сложность при начальной настройке и понимании.

4.2.4. JWT для аутентификации

Для обеспечения безопасности и аутентификации используется **JWT** (JSON Web Tokens). Каждый запрос от пользователей и администраторов проверяется на наличие действующего токена.

4.2.4.1. **Плюсы**:

- Централизованное управление безопасностью.
- Простота в реализации и интеграции с различными клиентами.

4.2.4.2. **Минусы**:

- При неправильной реализации может возникнуть угроза утечек данных.

5. Последствия

5.1. Улучшенная простота разработки:

- Монолитная архитектура упрощает разработку, тестирование и развертывание, особенно на старте проекта.

5.2. **Гибкость в изменении функционала**:

- Все компоненты находятся в одном сервисе, что облегчает изменение функционала и логику взаимодействия.

5.3. **Масштабируемость**:

- Система может столкнуться с проблемами масштабируемости при росте пользователей и ресторанов, так как все компоненты находятся в одном сервисе, и масштабировать их будет сложнее.

5.4. **Централизованное управление безопасностью**:

- Легкость в реализации безопасности и аутентификации через централизованный подход с использованием **JWT**.

5.5. **Обработка сообщений через цепочку обязанностей**:

- Легкость в расширении логики обработки сообщений через **цепочку обязанностей**, что позволяет гибко реагировать на изменения в требованиях к функционалу.

6. Риски

6.1. Масштабируемость монолита:

- При росте пользователей и ресторанов возможны проблемы с производительностью и отказоустойчивостью системы, так как все компоненты находятся в одном сервисе.

6.2. **Сложности при обновлениях**:

- Внесение изменений в одну часть системы может затронуть другие компоненты, что усложняет процесс обновлений и тестирования.

6.3. **Проблемы с отказоустойчивостью**:

- Если один из компонентов системы выйдет из строя, вся система может оказаться под угрозой, так как все компоненты зависят друг от друга.

7. Альтернативные решения

7.1. Использование микросервисной архитектуры:

- Перевод системы на микросервисы позволит улучшить масштабируемость и отказоустойчивость. Например, можно разделить **API** для админки и бота, а также сервисы для обработки заказов и меню.

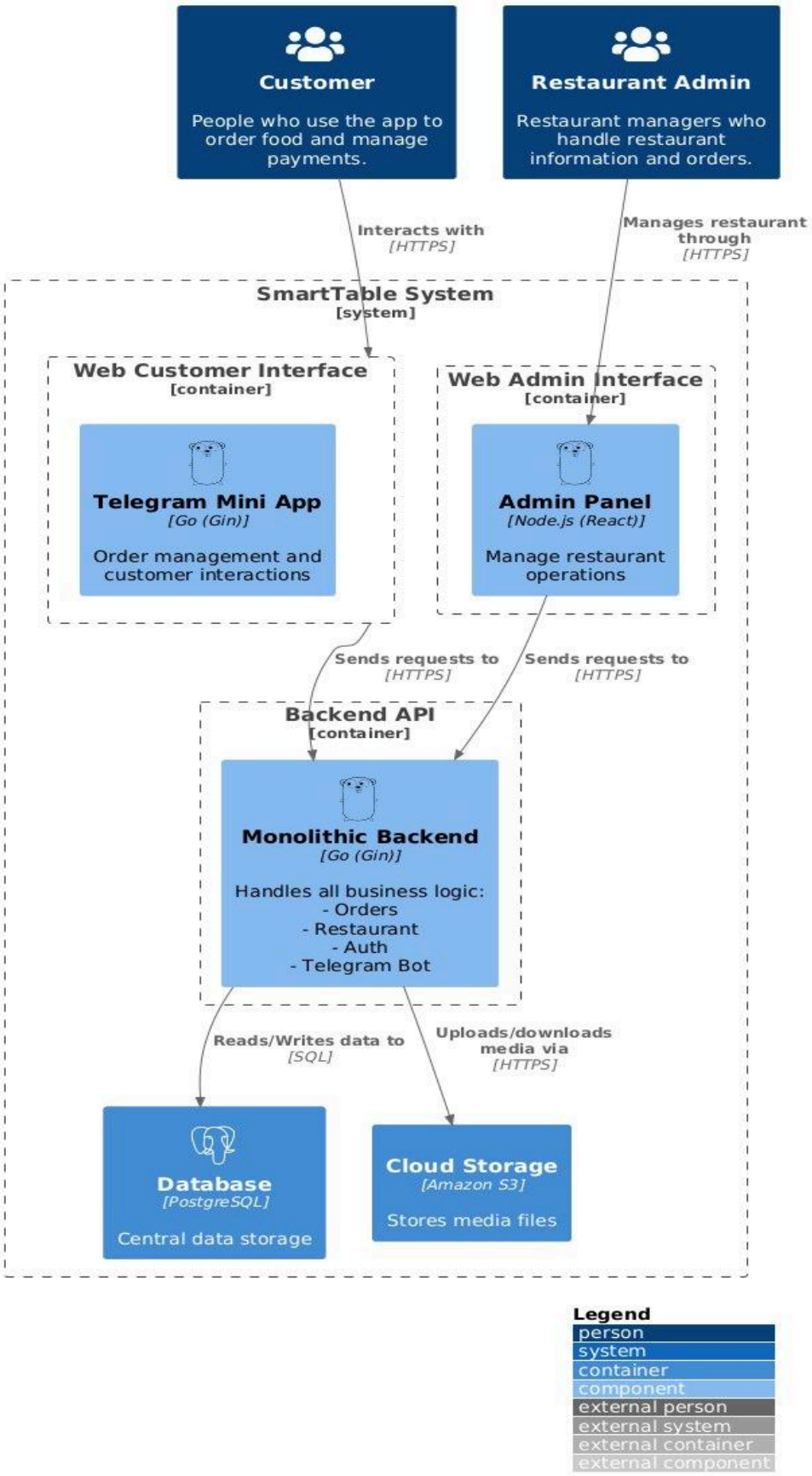
7.2. Использование очередей сообщений для асинхронной обработки:

- Очереди сообщений, такие как **Kafka**, могут помочь разгрузить систему, обеспечив асинхронную обработку запросов и уменьшив нагрузку на сервер.

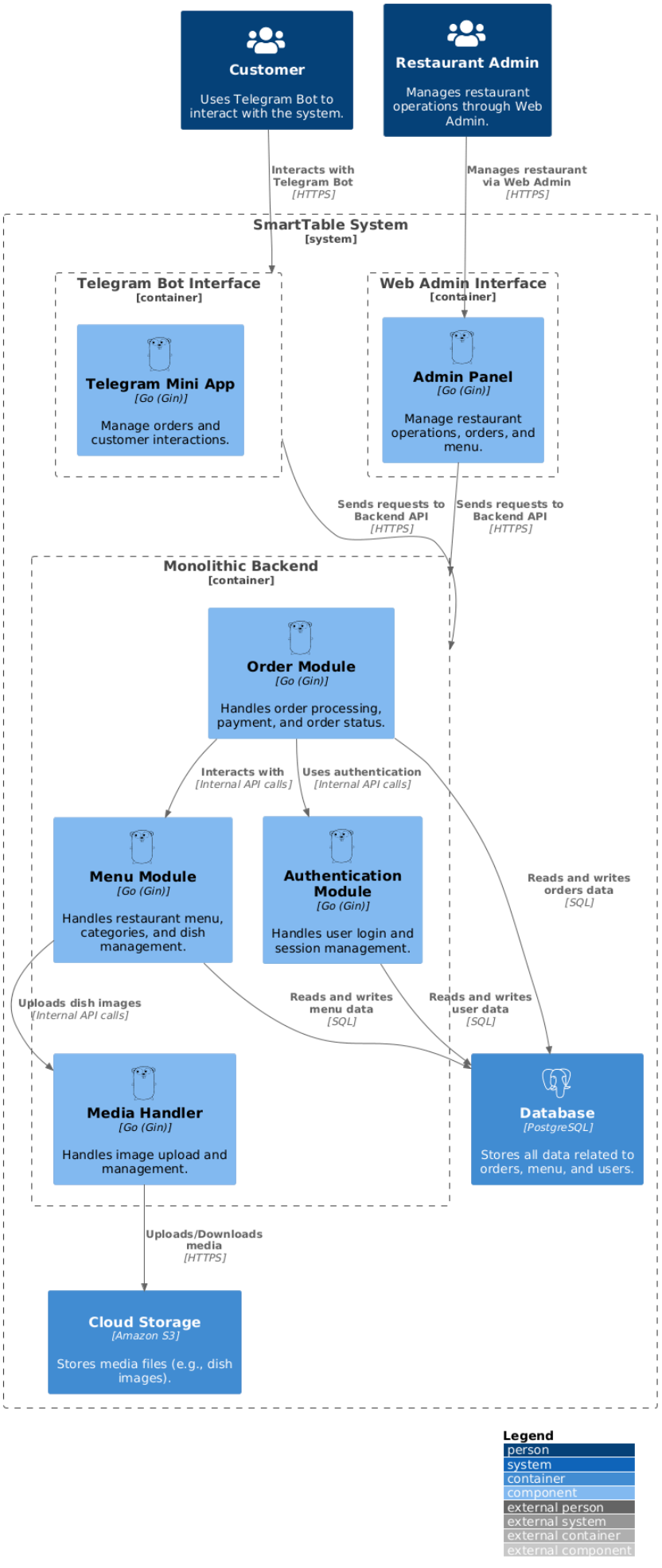
7.3. Использование WebSocket для реального времени:

- Для отправки событий в реальном времени (например, статус заказов, уведомления) можно использовать **WebSocket**.

4К модель: уровень Контейнер



4К модель: уровень Компонент



ER - диаграмма, структура БД

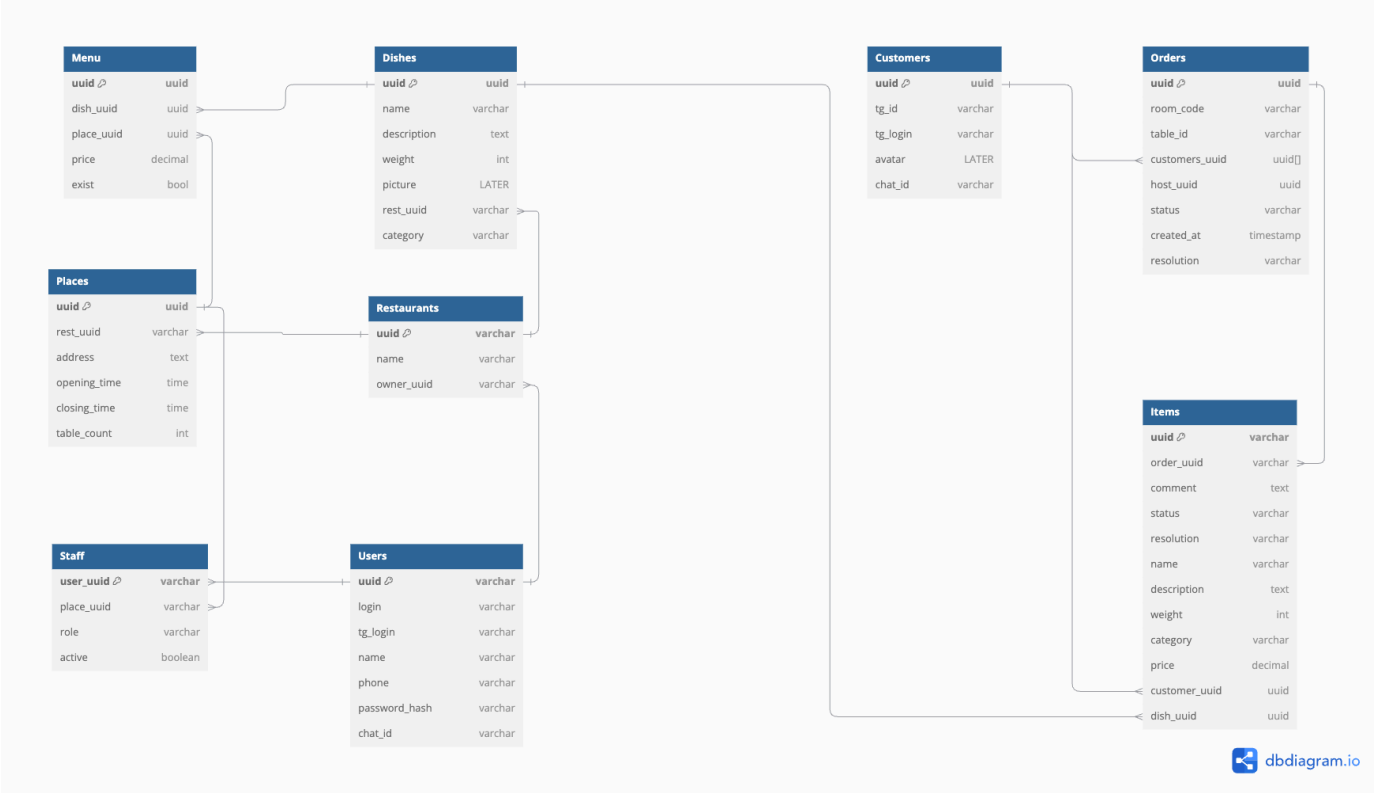
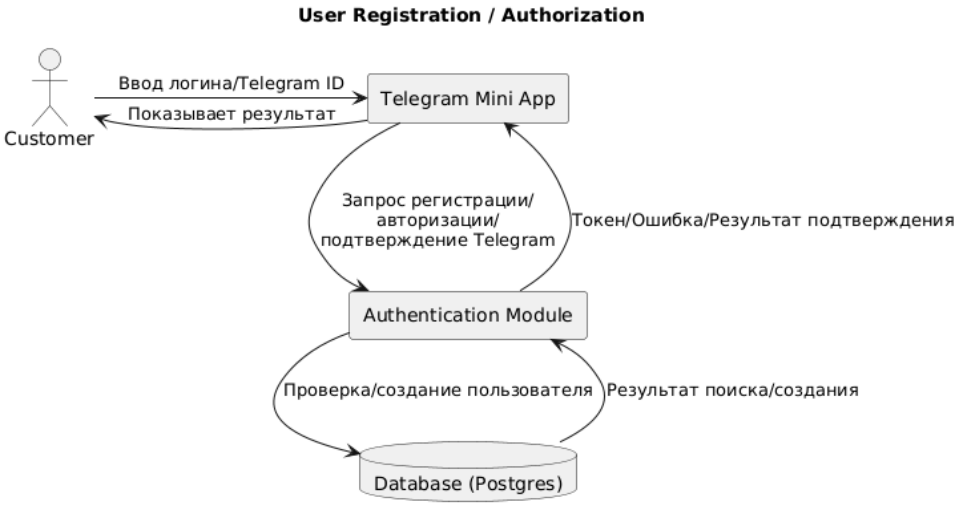
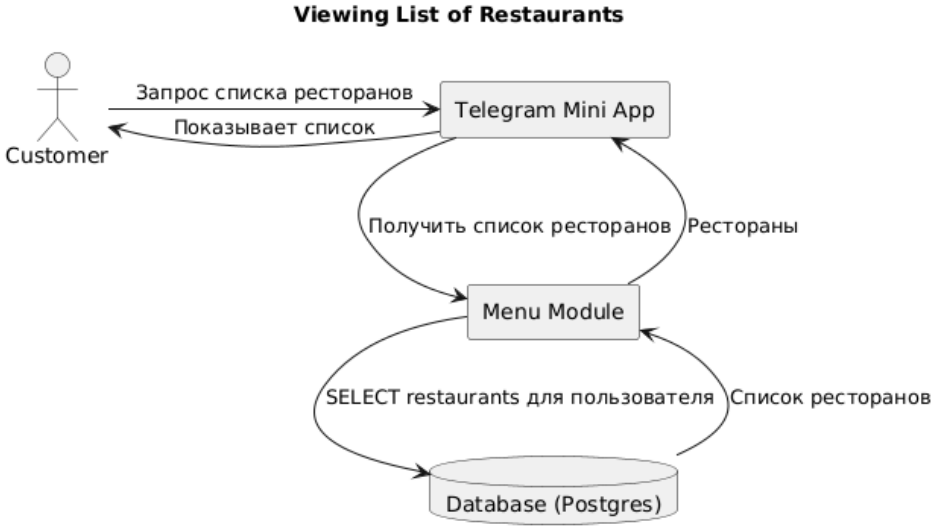


Диаграмма потоков данных

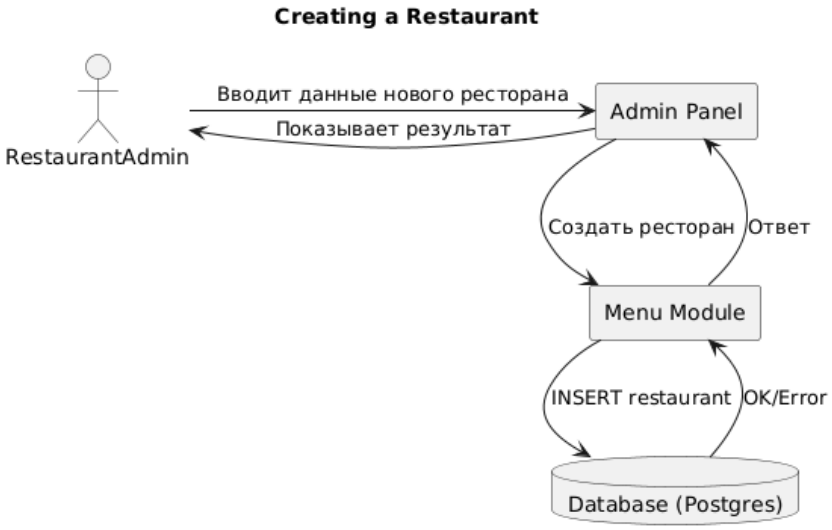
1. Регистрация/Авторизация пользователя и подтверждение логина через Telegram



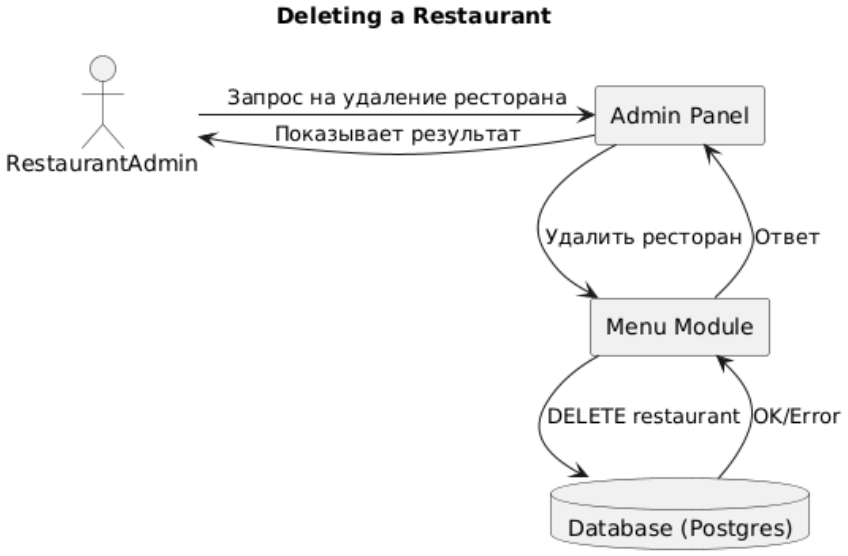
2. Отображение списка ресторанов пользователя



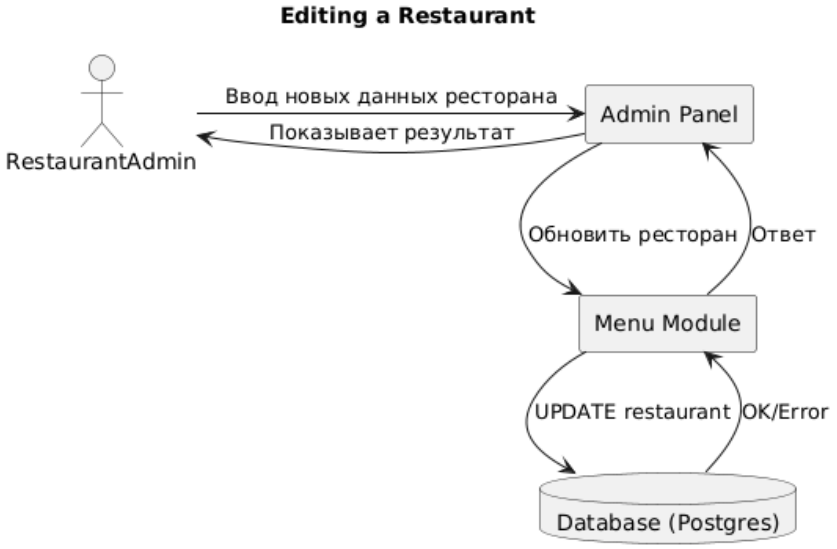
3. Создание ресторана



4. Удаление ресторана

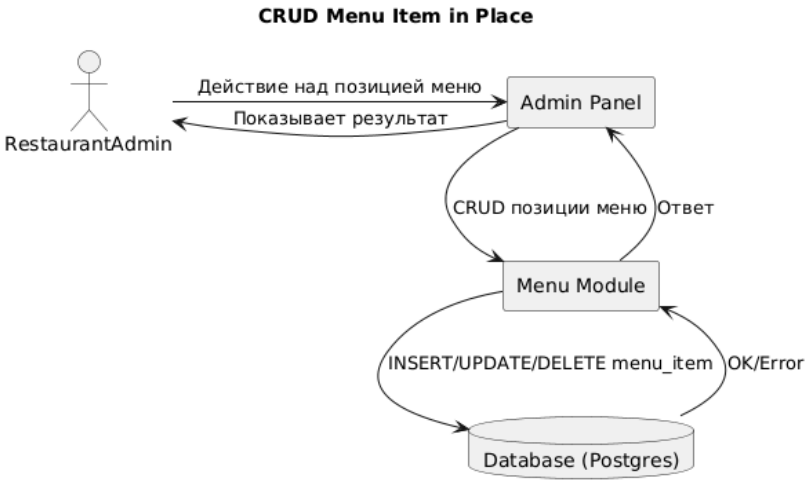


5. Редактирование ресторана

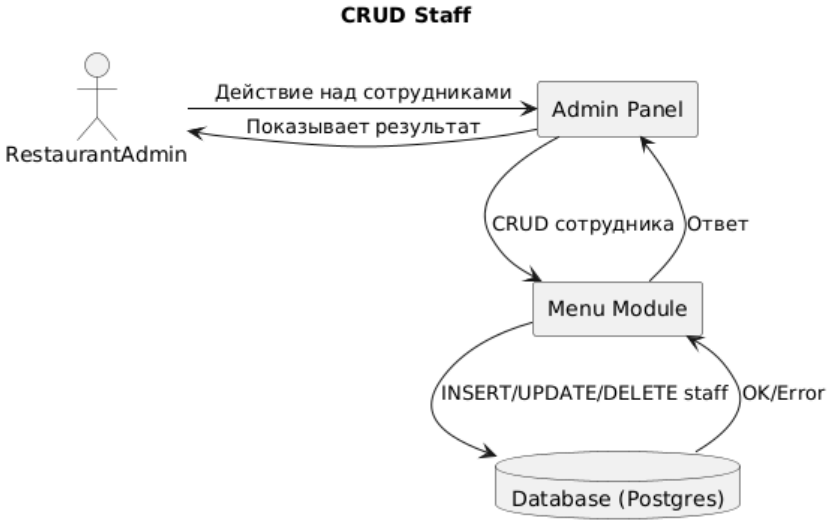


6. CRUD Плейса ресторана

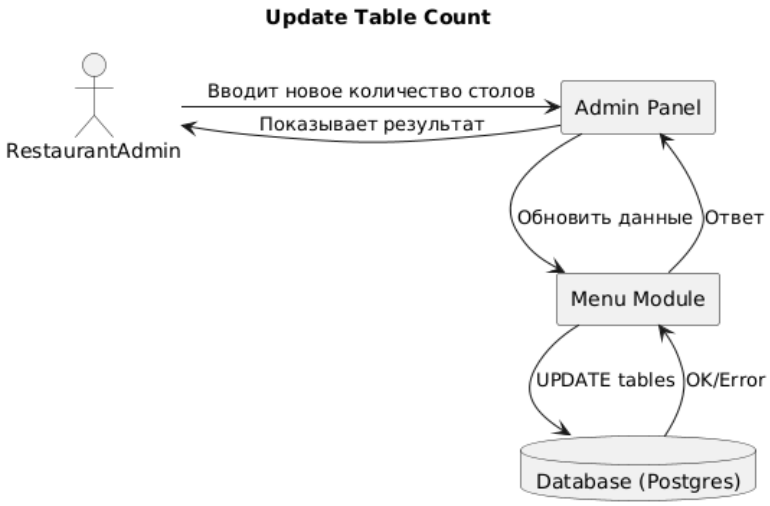
7. CRUD позиции меню плейса



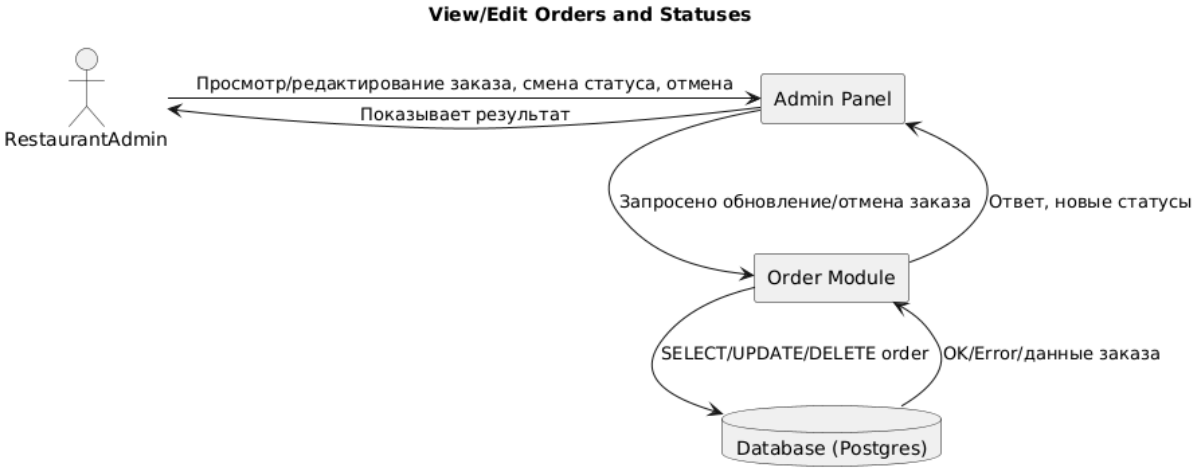
8. CRUD состава сотрудников



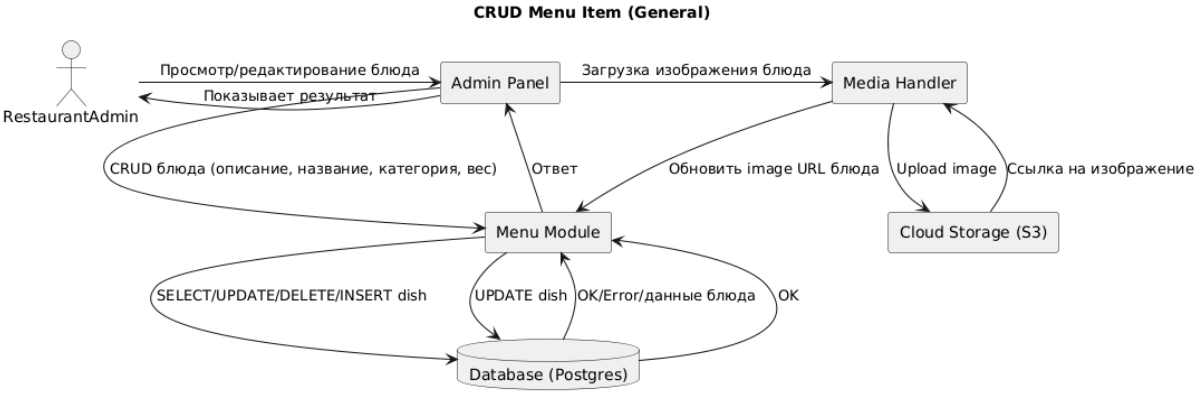
9. Редактирование количества столов



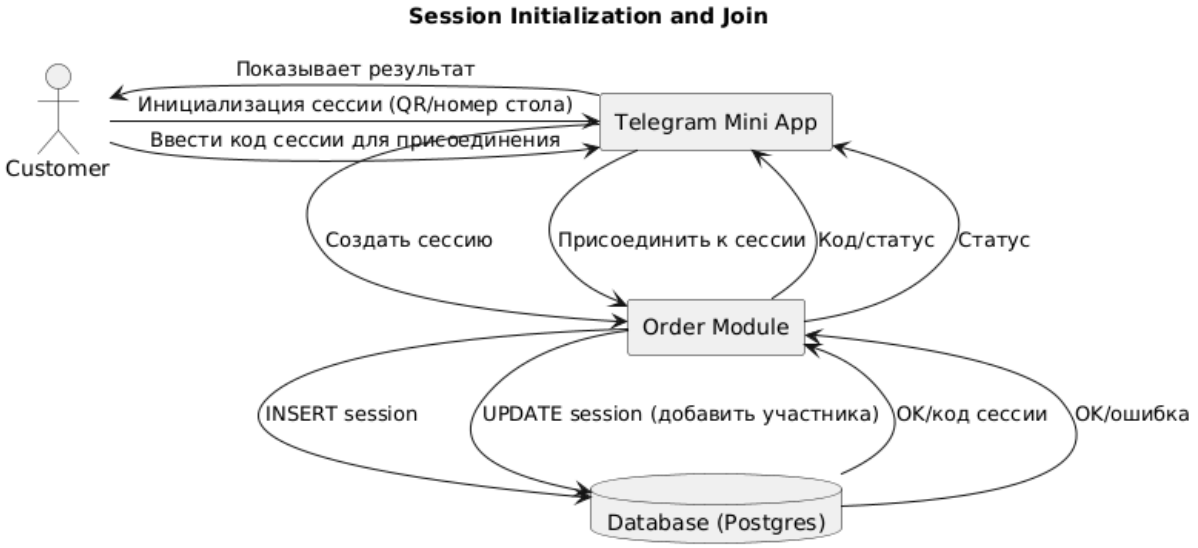
10. Просмотр и редактирование текущих заказов, статусы



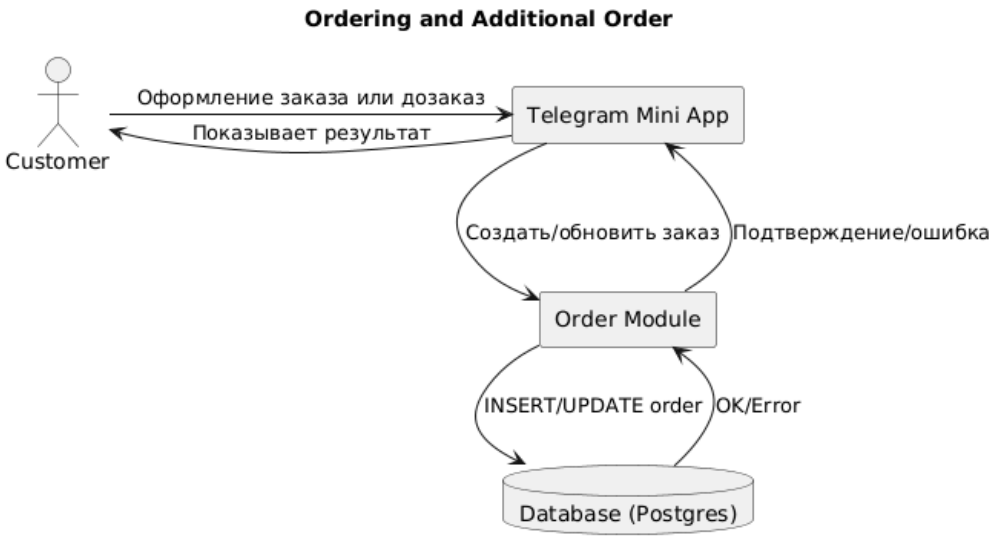
11. Просмотр и редактирование блюд (CRUD блюдо)



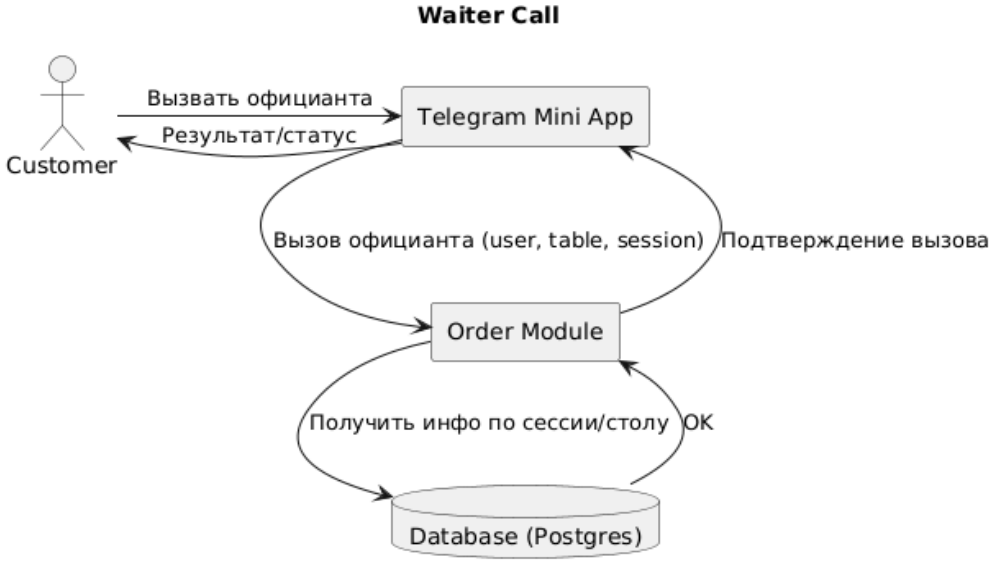
12. Инициализация и присоединение к сессии



13. Оформление заказа и дозаказ



14. Вызов официанта (ещё раз, для completeness)



15. Просмотр каталога плейса и детальных карточек