# Задача «Система приема заказов»

## Команда проекта

**Команда Big Int**

* Максим Милованов (Telegram: @Mirovan)
* Арина Свитова (Telegram: @svitova)

Репозиторий решения: <https://github.com/Mirovan/vtb-codenrock-arch-hackaton/tree/master/2.MainProblem>

## Структура

* [Описание задачи](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\1.Problem\README.md)
  + [Бизнес цели](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\1.Problem\1.BusinessGoals.md)
  + [Функциональные требования](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\1.Problem\2.FunctionalRequirements.md)
  + [Нефункциональные требования](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\1.Problem\3.NonFunctionalRequirements.md)
  + [Заинтересованные лица и события](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\1.Problem\4.ActorsAndActions.md)
* [Решение](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\README.md)
  + [Анализ сторонних решений](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\1.ThirdPartySolutions.md)
  + [Выбор архитектуры](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\2.ArchitectureSelection.md)
  + [Компоненты](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\3.Components.md)
  + [Модель C4](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\4.C4-model.md)
  + [Диаграмма развертывания](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\5.DeploymentDiagram.md)
  + [Описание API и модели данных](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\6.API.md)
  + [Стоимость владения](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\7.Cost.md)
* [ADR](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\3.ARD\README.md)
  + [ADR-001 Отслеживание событий изменения статуса заказа](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\3.ADR\ADR-001-order-status-events.md)

# Бизнес цели

Спроектировать «Систему приема заказов» для сети ресторанов

## Описание

Система приема заказов представляет из себя приложение, которое помогает обслуживать гостей в ресторане, а также осуществлять функцонал управления рестораном и вести бухгалтерскую отчетность

## Целевая аудитория

* Сотрудники ресторана (официант/бармен, управляющий рестораном)
* Бухгалтер
* Служба доставки и курьеры
* Клиенты и посетители ресторана
* Администратор системы

## Решаемые проблемы

* Ускорить обслуживание клиентов
* Повысить качество обслуживания
* Повысить прозрачность управления и увеличение финансовых показателей
* Увеличение скорости ведения бухгалтерской отчетности и снижение бухгалтерских ошибок

# Функциональные требования

* Обработка заказов официантами от клиентов ресторана
* Оплата заказа
* Работа с курьерами
* Управление бронью столиков
* Управление расписанием сотрудников
* Формирование меню и промо-программ для клиентов
* Работа с обратной связью
* Формирование бухгалтерской отчетности

## Дополнительные требования, не учтенные в постановке:

* Управление статистикой посещений
* Формирование промоакций и карт лояльности посетителей
* Интеграция со службами доставки
* Ведение склада и управление остатками
* Взаимодействие с поставками продуктов и необходимых материалов

# Нефункциональные требования

* Производительность. Среднее число заказов в сутки в ресторанах достигает 20 000
* Масштабирование. Планируется подключить к системе до 100 ресторанов
* Доступность. Необходимо обеспечить непрерывное время работы системы и её отдельных частей
* Отказоустойчивость. Mission-critical частью системы являются функционал работы обработки заказов. Остальные часть business-critical
* Безопасность. Необходимо обеспечить надежное хранение персональных данных пользователей
* Возможность развития системы с учетом быстро меняющихся требований бизнеса и рынка

# Заинтересованные лица и события

## Cтейкхолдеры и их интересы

Клиент:

* Создание заказа и просмотр статуса
* Оплата заказа

Официант (Администратор заказов для доставки):

* Создание и управление заказом
* Осуществление валидации оплаты
* Отправка заказов в курьерскую службу

Управляющий рестораном:

* Формирование графика работы
* Просмотр статистики работников
* Управление местами и рассадкой в ресторане
* Бронирование столиков
* Настройка промоакций
* Управление меню

Служба доставки и курьеры:

* Создание заказа и просмотр статуса
* Осуществление валидации оплаты

Бухгалтер:

* Формирование бухгалтерской отчетности
* Просмотр финансовой статистики

Администратор системы:

* Управление сетью ресторанов
* Управление конфигураций ресторанов
* Администрирование пользователей

## Сценарии критических бизнес-процессов

Основные события системы:

* Официант создает заказ и сохраняет его. У заказа изменяются статусы
* Клиент выбирает блюда и складывает их в корзину. Формируется заказ, который отправляются в ресторан. Администратор заказов для доставки (официант) отправляет передаёт готовый заказ курьеру
* Курьер принимает заказ, доставляет его. Осуществляет прием оплаты

# Сторонние решения

Основные продукты и решения, которые могут быть применены для реализации системы:

* Fusion POS
* СБИС Престо
* Quick Resto
* R\_keeper
* Iiko
* Yuma

Преимущества систем:

* Гибкие тарифы
* Низкие затраты при начальном использовании
* Множество готовых интеграций с другими продуктами

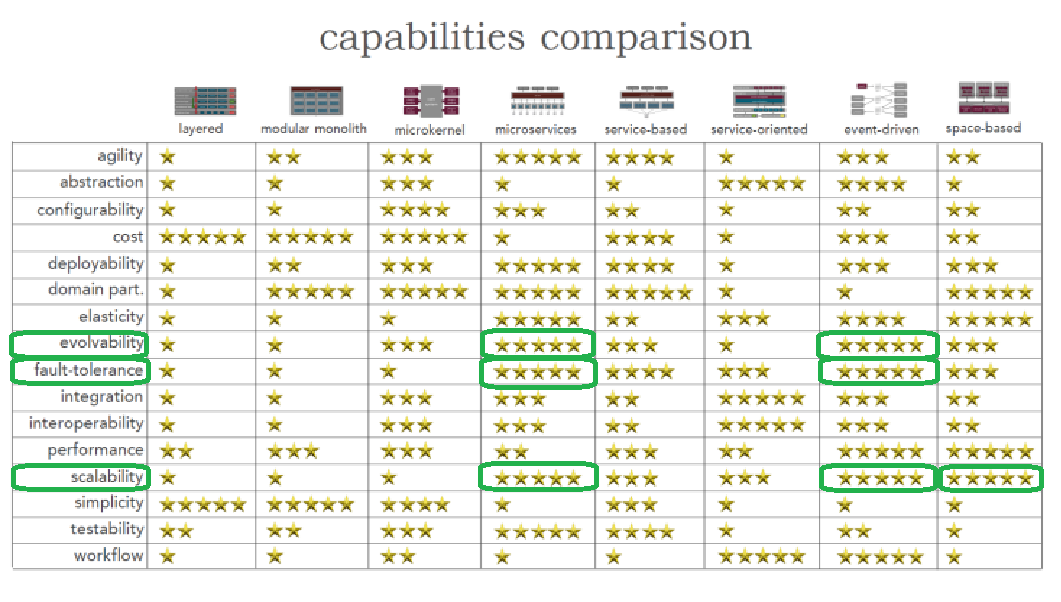
Недостатки:

* Непрогнозируемая стоимость обслуживания с учётом перспектив роста ресторанной сети
* Нет возможности (или достаточна сложная) гибкой настройки ресторанов сети и механизмов эволюционного развития продукта
* Нет гарантий сохранения персональных данных клиентов

# Архитектурные характеристики

Исходя из нефункциональных требований были выбраны 3 главные архитектурные характеристики для реализации системы:

* Отказоустойчивость (fault tolerance) - При возникновении фатальных ошибок другие части системы продолжают функционировать
* Масштабируемость (scalability) - Функция емкости системы и ее роста с течением времени; по мере увеличения количества пользователей или запросов в системе скорость реагирования, производительность и частота ошибок остаются постоянными
* Возможностью развития/Эволюционируемость (evolvability) - Система способна эволюционировать, быстро изменяться под нужды бизнеса. Компоненты системы имеют возможность непрерывной доставки.



Наиболее подходящим решением является **микросервисная архитектура с элементами event-driven архитектуры**.

# Основные компоненты и их связи

Для определения основных компонентов системы применим метод Event Storming, который помог визуализировать рабочие процессы системы, выявить события предметной области и определить взаимодействие между компонентами. В ходе этого процесса определены необходимые компоненты и их взаимосвязи, что послужило основой для проектирования и внедрения системы.

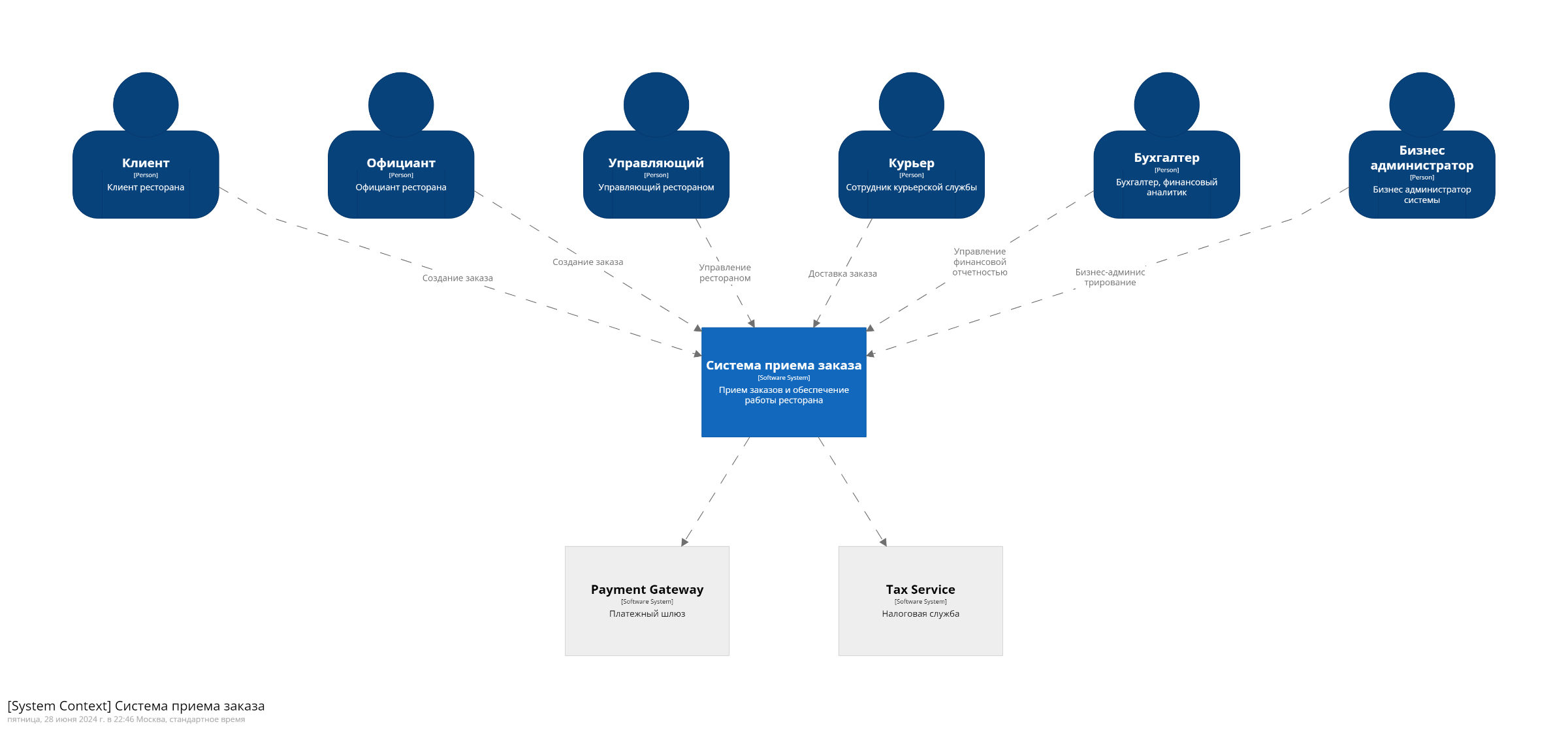
Применяя методологию Event Storming, мы можем эффективно определять соответствующие **ограниченные контексты** в нашей программной архитектуре:

* Заказы - управляет бизнес-процессом создания заказа и обработки его статусов
* Ресторан - осуществляет бизнес-процесс управления одним рестораном
* Доставка - обеспечивает процесс доставки заказов клиентам с помощью курьеров
* Финансы - осуществляет бизнес-процесс управления бухгалтерской отчетностью и финансовыми показателями
* Бизнес-администрирование - осуществляет бизнес-процесс управления сетью ресторанов

# Архитектурные характеристики

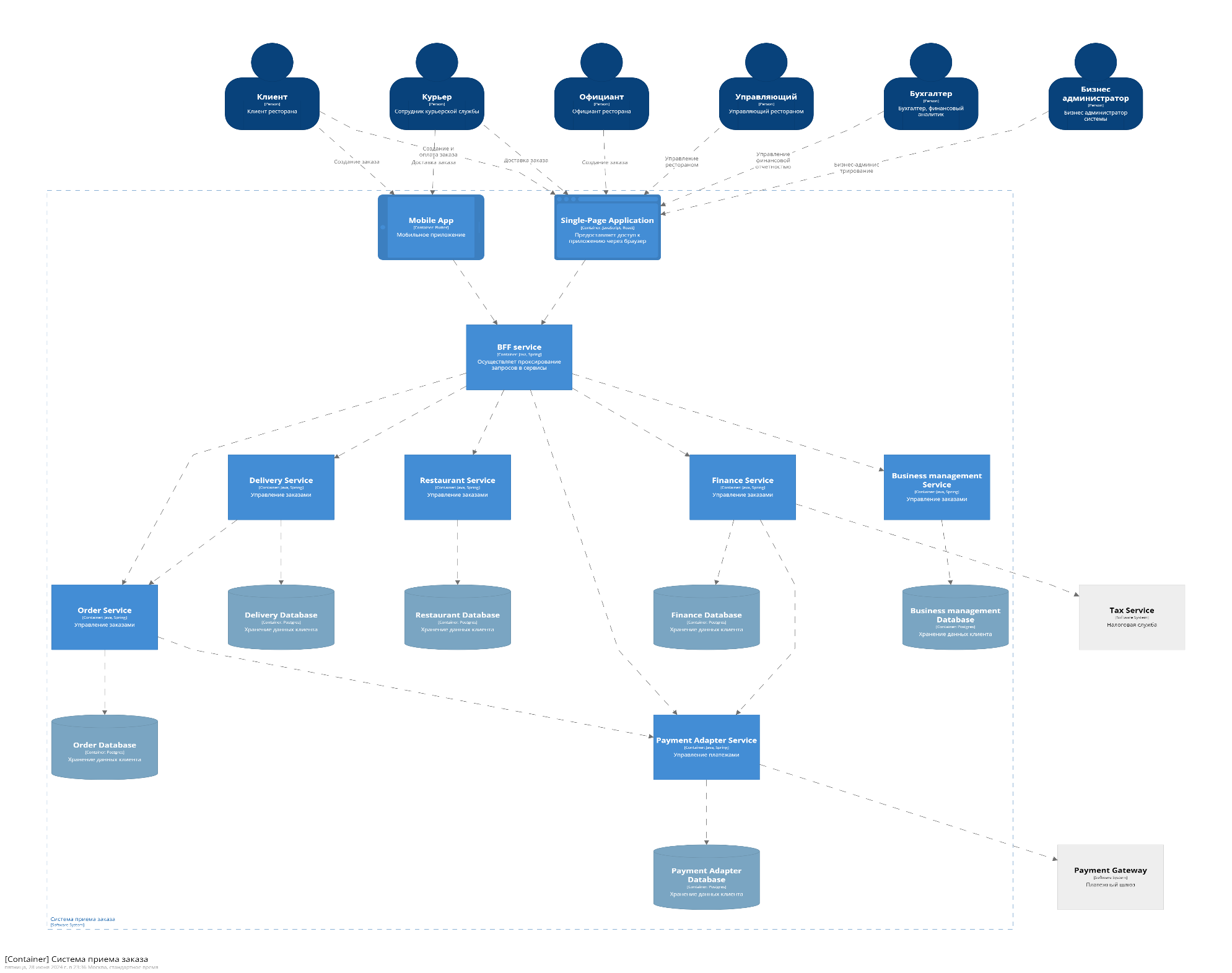
## Контекстная диаграмма модели C4

Уровень C1 (Context view) в модели C4, представляет общий вид системы, ее внешние взаимодействия и зависимости. Диаграмма изображает систему как единое целое, окруженное внешними субъектами, системами.



## Контейнерная диаграмма модели C4

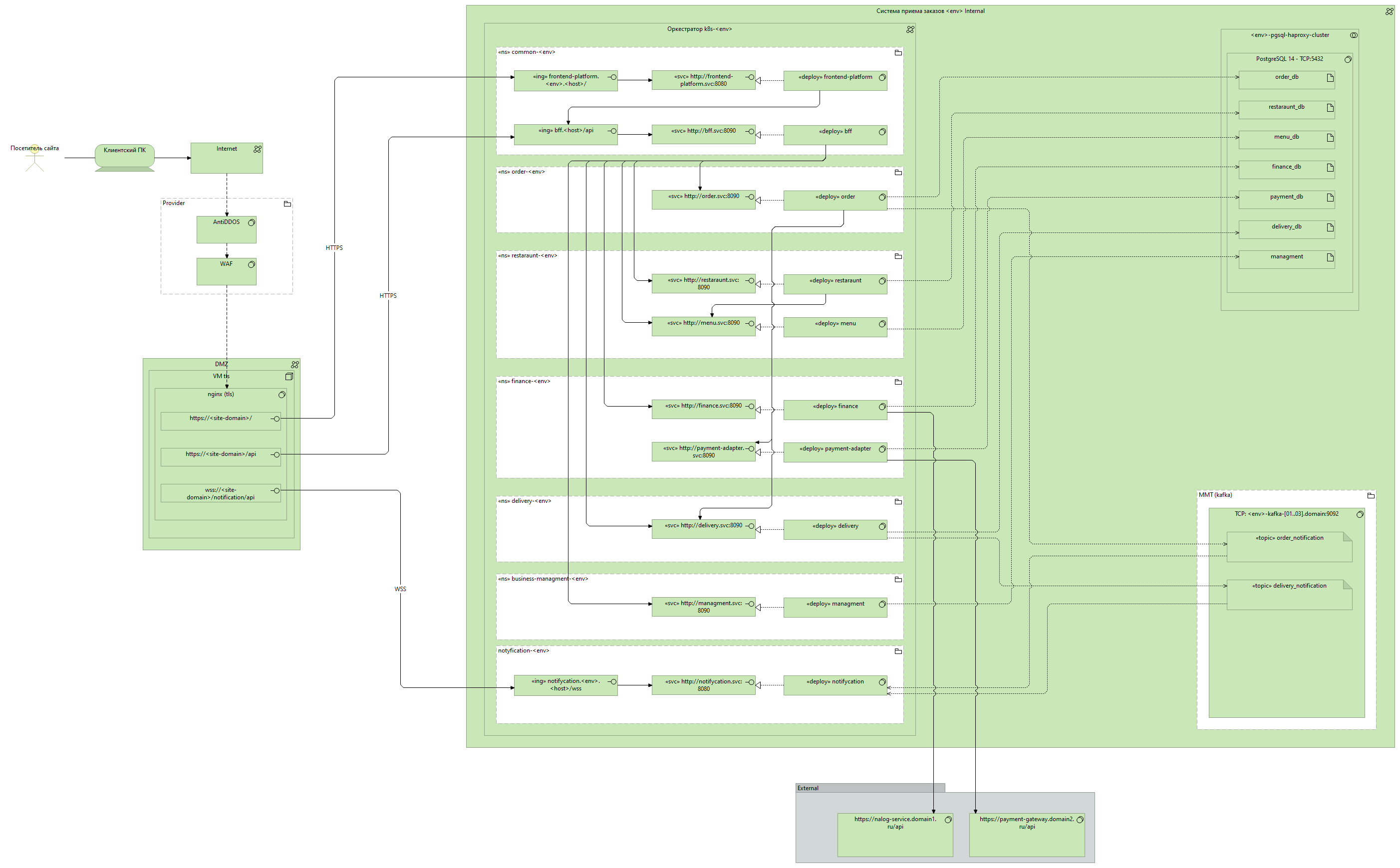
Уровень C2 (Container view) в модели C4, углубляется во внутренние компоненты системы, демонстрируя различные контейнеры или модули приложений и их взаимосвязи. Он предоставляет подробное описание внутренней структуры системы, включая различные типы контейнеров, таких как службы, базы данных, пользовательские и внешние интерфейсы. Такое представление позволяет заинтересованным сторонам понять внутреннюю архитектуру системы и то, как ее компоненты взаимодействуют для выполнения своих функций.



Ресурсы:

* [Исходник диаграммы C4 для Structurizr](file:///D:\IdeaProjects\vtb-codenrock-arch-hackaton\2.MainProblem\2.Solution\resources\c4.dsl)
* [Интерпретатор DSL - Structurizr](https://structurizr.com/dsl)

# Диаграмма развертывания

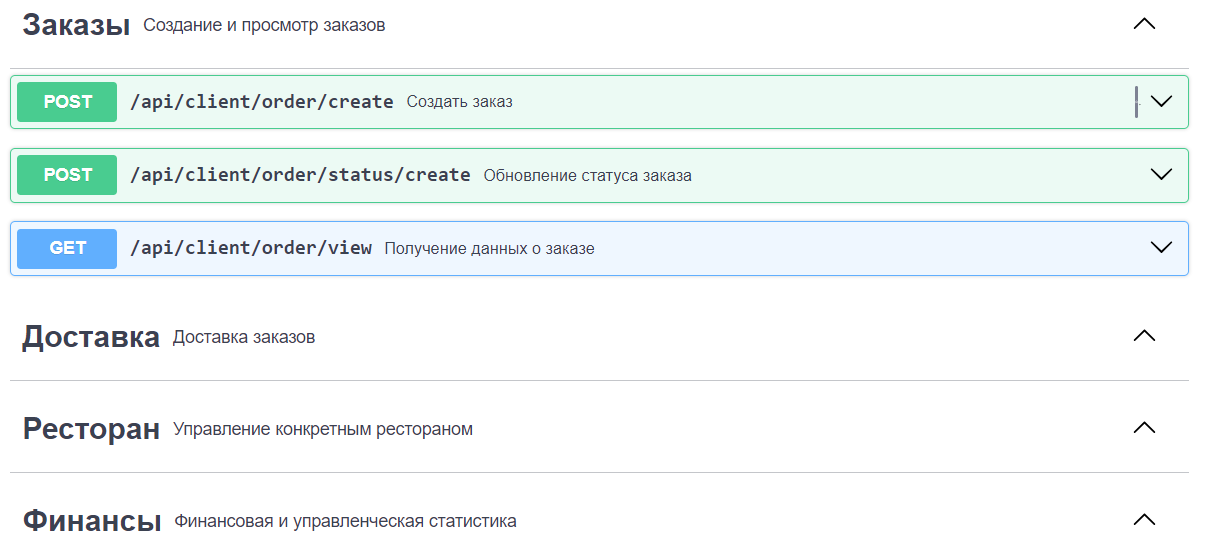


# Модель данных и описание API

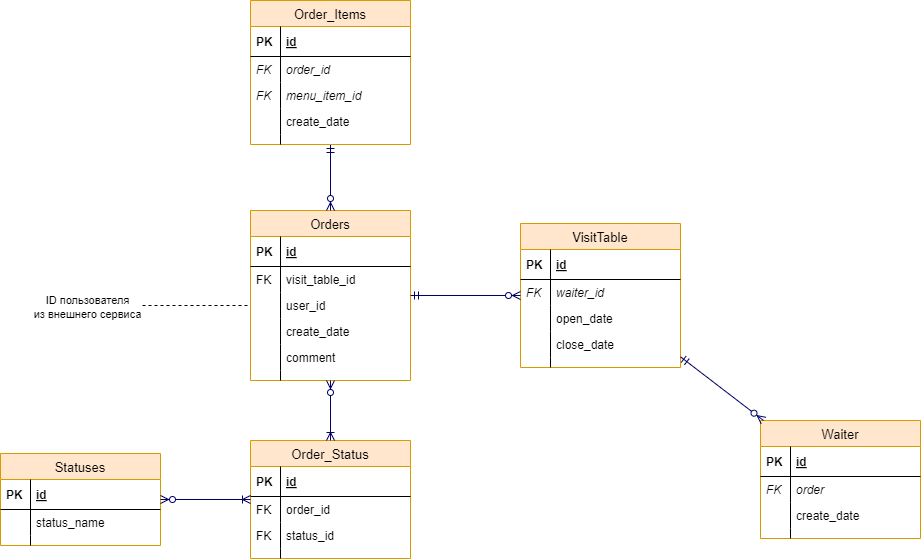
Протоколы для взаимодействия:

* HTTP/REST (взаимодействие пользователей с сайтом, взаимодействие между сервисами)
* WebSocket (оповещение пользователей и сотрудников)
* TCP/Kafka (взаимодействие между сервисами)

## Описание OpenAPI



## ER-модель данных сервиса заказов



# Стоимость владения системой

## Оценка нагрузки на БД

По исходным данным, cреднее число заказов в сутки в ресторанах достигает 20,000 которое продолжает расти.

Примем за рост числа заказов 20% в год, т.е. в течении 5 ближайших лет составит 50,000 заказов в сутки.

Для предложенной архитектуры не требуется больших вычислительных ресурсов. Однако, для хранения всех действий клиентов и статусов, необходимо учесть хранение всех этих событий.

Оценочно число записей различных событий изменений статусов заказов составляет в сутки:

50,000 (заказов в сутки) \* 10 (событий смены статусов) = 500,000 записей в сутки.

или

500,000 (записей) \* 5 (лет) \* 365 (дней) ~= 1 млрд. записей.

В данном случае нагрузка на БД не является существенной. Однако стоит учесть

* сценарии поиска информации по заказам
* деградацию базы данных с увеличением числа заказов

## Оценка количества посетителей

С учетом планируемого числа поключенных ресторанов - 100, рассмотрим вероятное число пользователей в единицу времени. В каждом ресторане работает до 10 человек, который (пусть) может одновременно обслужить 2 столика. Суммарное число обращений к сайту официантами:

100 (ресторанов) \* 10 (официантов) \* (2 столиков) = 2,000 обращений к веб-ресурсу и API в пиковом моменте (RPS).

Для этого целесообразно иметь гибко настроенный кластер Kubernetes с динамическим распределением ресурсов и горизонтальным масштабированием экземпляров приложений.

**Оценка (минимальной) стоимости**

* Сервисы x10 (10 сервисов) - 1Ghz 2CPU, 4Gb RAM, 10Gb HDD
* Nginx - 3Ghz 1CPU, 2Gb RAM, 20Gb HDD
* Kafka - 2Ghz 4CPU, 16Gb RAM, 1Tb HDD
* Database X4 (для каждого сервиса) - 1Ghz 2CPU, 2Gb RAM, 100Gb HDD

Следует также учесть затраты:

* на инфраструктуру: виртуализацию (Docker) и оркестрацию (k8s)
* на мониторинг (ELK, Grafana)
* подсистему хранения файлов (S3 Minio)

В решении (за отсутствием данных) не учтены:

* механизмы репликации и кластеризации данных БД

# ADR-001: Отслеживание событий изменения статуса заказа

## Дата:

29-06-2024

## Статус:

Принято

## Контекст:

Сервис order (обеспечивающий заказы) сохраняет заказ и его статус. Необходимо обеспечить возможность просмотра изменений статуса заказа относительно времени.

## Решение:

Варианты решения:

* Использование Hibernate Envers.
* Реализации механизма хранения статусов самостоятельно применяя подход Event Sourcing. События изменения статуса заказа записываются в отдельную таблицу с указанием времени изменения. При необходимости, в случае изменения статуса заказа происходит оповещение необходимых сервисов через брокер сообщений.

## Последствия:

### Преимущества:

* Возможность отслеживания изменения статусов заказа со временем
* Возможность использования данных для статистики

### Недостатки:

* Необходимость выделения дополнительных таблиц на уровне БД