ИНДИВИДУАЛЬНЫЙ ПРЕДПРИНИМАТЕЛЬ
ПЕРЦЕВ ВЛАДИСЛАВ РОМАНОВИЧ
344033, г. Ростов-на-Дону, пер. Богачева, 16.
Сайт: www.sellkit.cc E-mail: vladpertcev@mail.ru
Тел.: +7 (988) 565-00-38

Sell Kit

ИНН: 614109950071
ОГРНИП: 320619600087432
Расчётный счёт: 40802810552090059516
Банк: ЮГО-ЗАПАДНЫЙ БАНК ПАО СБЕРБАНК
БИК банка: 046015602
Корсчёт: 30101810600000000602

Описание технической архитектуры

программного обеспечения «SellKit»

2026

ВВЕДЕНИЕ

Техническая архитектура программного обеспечения «SellKit» представляет собой совокупность важнейших решений об организации программной системы, включая выбор структурных элементов и их интерфейсов, способы соединения указанных элементов, распределение ответственности между уровнями системы, а также архитектурный стиль, направляющий организацию системы, взаимодействие компонентов, масштабирование и эволюцию. Настоящий документ фиксирует указанные решения в структурированном виде.

СОДЕРЖАНИЕ:

Термины и сокращения……..4

Назначение и область применения……...5

Контекст и границы системы……6

Архитектурный стиль и ключевые решения……...7

Структурные элементы системы и их интерфейсы……....8

Соединение выбранных элементов структуры……..…11

Логическая архитектура (уровни, подсистемы)…….12

Функциональная декомпозиция Back-End……..13

Модель данных и хранилища……...14

Кэширование и производительность……...14

Асинхронная обработка и очереди……..15

Хранение медиа и доставка контента (Storage/CDN)…….16

Поддерживаемые варианты интеграций……..16

Безопасность и управление доступом……..17

Надёжность, отказоустойчивость и восстановление……..17

Масштабируемость и балансировка…….18

Мониторинг, логирование и аудит……...18

Развертывание и эксплуатация (DevOps)…….19

Описание программной реализации…….20

Обоснование технологической независимости и отсутствия иностранного контроля……...21

Соответствие требованиям и нефункциональные характеристики……...21

Приложения (матрица ролей, перечень API, сценарии взаимодействия)…….22

Выводы……25

1. Термины и сокращения

Термин/сокращение	Расшифровка
API	Интерфейс прикладного программирования
RBAC	Role-Based Access Control — управление доступом на основе ролей
TLS	Transport Layer Security — криптографическая защита канала связи
CDN	Content Delivery Network — сеть доставки контента
CI/CD	Непрерывная интеграция и развертывание
SaaS	Software as a Service — программное обеспечение как услуга
ACID	Свойства транзакций в реляционных БД
JWT	JSON Web Token — токен доступа
RPO/RTO	Цели по потере данных/времени восстановления

Термины используются в общеупотребимом в отрасли значении. Приоритет имеет содержание настоящего документа как описания архитектурных решений.

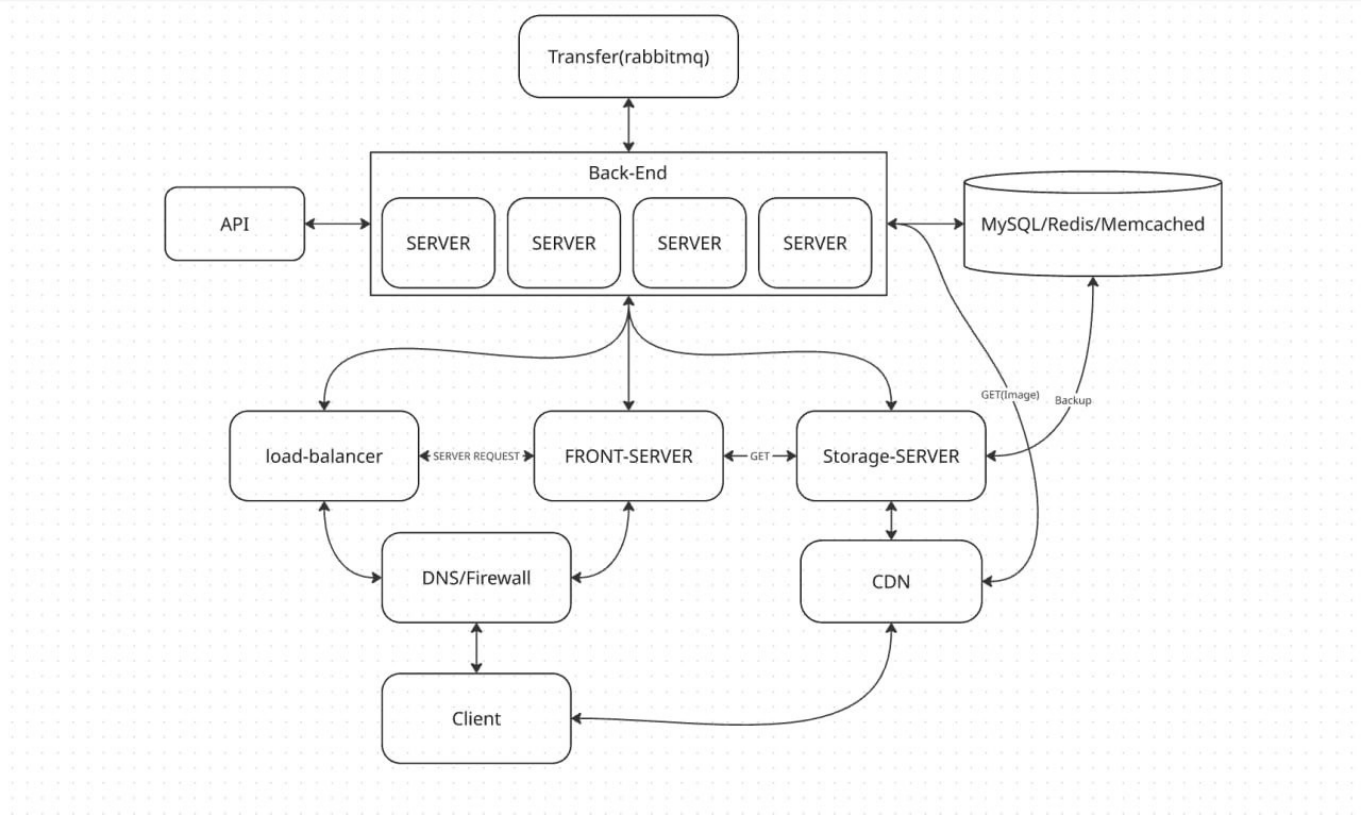
2. Назначение и область применения

ПО «SellKit» — многопользовательская распределённая серверная информационная система класса SaaS, предназначенная для автоматизации процессов ведения цифровых каталогов/меню, приёма и обработки заказов, администрирования пользователей и управления справочными данными организаций сферы услуг (в т.ч. общественное питание, доставка, розничные точки).

Система поддерживает круглосуточную эксплуатацию и работу при переменной нагрузке. Архитектура предусматривает модульное расширение функциональности и масштабирование без остановки сервиса.

Допускаются варианты эксплуатации: (1) в инфраструктуре правообладателя (облачная модель IaaS), (2) в виртуализированной среде заказчика, (3) на выделенных серверах в контуре заказчика (on-premise).

- Тип ПО: серверное, веб-ориентированное, распределённое.
- Режим: многопользовательский, 24/7.
- Клиенты: веб-браузеры и мобильные браузеры (адаптивный интерфейс).



3. Контекст и границы системы

Система взаимодействует с внешними пользователями через публичный веб-интерфейс. Внутренние сервисы и контуры хранения данных размещаются в изолированном сегменте сети и недоступны напрямую из публичной сети.

3.1 Участники и роли

- Гость/посетитель: просмотр меню/каталога, формирование заказа.
- Клиент (авторизованный): доступ к персональным данным (при наличии), история заказов.
- Оператор/менеджер: обработка заказов, управление позициями меню, акции, статусы.
- Администратор организации: управление пользователями организации, ролями, настройками.
- Системный администратор: эксплуатация и мониторинг инфраструктуры (в контуре правообладателя).

3.2 Границы ответственности

ПО «SellKit» обеспечивает хранение, обработку и предоставление данных, а также контроль доступа к ним. Внешние платежные сервисы/кассовые системы/службы доставки при наличии подключаются через интеграционные интерфейсы и не являются частью ядра ПО.

3.3 Контекстная схема (описание)

Клиенты обращаются по HTTPS к Front-Server через балансировщик нагрузки. Front-Server взаимодействует с Back-End (REST) и с подсистемой статического контента (Storage/CDN). Back-End использует MySQL для транзакционных операций, Redis/Memcached для кэширования, RabbitMQ для фоновой обработки, а также сервисы мониторинга и журналирования.

4. Архитектурный стиль и ключевые решения

Архитектура ПО реализована как смешанный стиль: клиент-сервер + многоуровневая (Layered) архитектура с сервисно-ориентированными принципами и событийными элементами (Event-Driven) для асинхронной обработки.

4.1 Архитектурные принципы

- Разделение ответственности (Separation of Concerns)
- Отказоустойчивость
- Масштабируемость
- Кэширование для повышения производительности
- Асинхронная обработка операций
- Минимизация точек отказа

4.2 Обоснование выбора архитектуры

- Масштабируемость: возможность горизонтального увеличения Front/Back без изменения логики клиентов.
- Надёжность: изоляция отказов и контроль доступности узлов посредством health-check.
- Безопасность: разделение публичного и внутреннего контура, RBAC, шифрование каналов связи.
- Расширяемость: модульное развитие прикладного уровня и интеграций без переработки всей системы.

4.3 Ключевые архитектурные решения (совокупность важнейших решений)

1. Использование stateless-взаимодействия на уровне Front-Server для упрощения масштабирования.
2. Выделение Back-End как прикладного уровня, реализующего бизнес-правила и транзакционную обработку.
3. Использование MySQL для согласованного хранения критичных данных и транзакций.
4. Применение Redis/Memcached по паттерну Cache-Aside для ускорения чтения.
5. Асинхронная обработка через RabbitMQ для разгрузки синхронных запросов и устойчивости к пиковым нагрузкам.
6. Разделение хранения медиа (Storage-Server) и доставки контента (CDN).
7. Централизованное логирование и мониторинг для контролируемой эксплуатации.

Архитектура программного обеспечения «SellKit» реализована в смешанном архитектурном стиле:

4.4 Клиент-серверная архитектура

Четкое разделение:
Клиентский уровень
Серверный уровень

4.5 Многоуровневая (Layered Architecture)

Выделены уровни:
Presentation Layer (Client, Front-Server)
Application Layer (Back-End)
Data Layer (MySQL, Redis, Memcached)
Infrastructure Layer (RabbitMQ, CDN, Storage)

4.6 Микросервисные элементы

Back-End может быть масштабирован по сервисам и разделён на логические компоненты.

4.7 Асинхронная архитектура

Использование RabbitMQ обеспечивает:
неблокирующую обработку задач
устойчивость к нагрузкам
масштабируемость

4.8 Горизонтальная масштабируемость

Балансировщик нагрузки
Кластер Back-End серверов
Масштабирование Front-Server

5. Структурные элементы системы и их интерфейсы

5.1 Клиентский уровень (Client)

Назначение:
Веб-браузеры, мобильные устройства и иные клиентские приложения, осуществляющие доступ к системе.

Интерфейсы:
HTTPS (REST API)
HTTP(S) запросы к CDN
DNS-запросы

5.2 DNS / Firewall

Назначение:
Разрешение доменных имен.
Фильтрация и контроль сетевого трафика.
Защита от несанкционированного доступа.

Интерфейсы:
DNS (UDP/TCP)

HTTPS
Сетевые ACL и правила фильтрации

5.3 Балансировщик нагрузки (Load Balancer)

Назначение:
Распределение входящих HTTP(S) запросов между Front-Server экземплярами.
Поддержка отказоустойчивости.
Возможность горизонтального масштабирования.

Интерфейсы:
HTTPS входящий
HTTP(S) к Front-Server
health-check API

5.4 Front-Server

Назначение:
Обработка входящих пользовательских HTTP-запросов.
Взаимодействие с Back-End.
Формирование ссылок на медиа-ресурсы, размещённые в Storage-Server и/или CDN (выдача контента — клиенту напрямую/через CDN).
Формирование ответов клиенту.

Интерфейсы:
REST API к Back-End
HTTP GET к Storage-Server (клиентский уровень напрямую или через CDN по предоставленным URL)
Внутренние RPC/HTTP вызовы
HTTPS к клиенту

5.5 Back-End (кластер серверов)

Состоит из нескольких серверных экземпляров (Server), объединенных в логический кластер.

Назначение:
Бизнес-логика.
Обработка заказов.
Управление пользователями.
Работа с базами данных.
Обработка фоновых задач.

Интерфейсы:
REST API (внутренний)
SQL (MySQL)

Redis protocol
Memcached protocol
AMQP (RabbitMQ)
Внутренние сервисные HTTP-интерфейсы

5.6 Transfer (RabbitMQ)

Назначение:
Асинхронная передача сообщений.
Очереди задач.
Разгрузка Back-End от синхронных операций.

Интерфейс:
AMQP

5.7 Система хранения данных (MySQL / Redis / Memcached)

MySQL
Реляционная база данных.
Хранение структурированных данных.

Redis
In-memory key-value хранилище.
Кэширование.
Быстрое чтение.

Memcached
Кэширование данных для ускорения отклика системы.

Интерфейсы:
SQL
Redis protocol
Memcached protocol

5.8 Storage-Server

Назначение:
Хранение медиафайлов (изображения, файлы меню и др.).
Предоставление файлов по HTTP GET.
Резервное копирование данных в систему хранения.

Интерфейсы:
HTTP GET
API интеграции
Backup interface

5.9 CDN (Content Delivery Network)

Назначение:
Доставка статического контента.
Георасределённое кэширование.
Снижение нагрузки на Storage-Server.
Интерфейсы:
HTTPS
HTTP Cache Protocol

6. Соединение выбранных элементов структуры

Архитектура организована по следующей схеме взаимодействия:
Клиент направляет HTTP(S) запрос.
Запрос проходит через DNS и Firewall.
Балансировщик нагрузки направляет запрос на один из Front-Server.

Front-Server:
Обращается к Back-End для выполнения бизнес-логики.
При необходимости Back-End формирует ссылки на медиа-ресурсы в Storage-Server/CDN; получение медиа осуществляется клиентом напрямую (или через CDN) по предоставленным URL.

Back-End:
Работает с MySQL.
Использует Redis и Memcached для кэширования.
Передаёт асинхронные задачи через RabbitMQ.

Storage-Server:
Предоставляет файлы клиентскому уровню напрямую или через CDN (по URL, сформированным Back-End).
Интегрирован с CDN.

CDN:
Доставляет статический контент клиенту напрямую.
Таким образом реализуется многоуровневое распределение ответственности и минимизация узких мест.

7. Логическая архитектура (уровни, подсистемы)

7.1 Presentation Layer

Presentation Layer включает клиентские приложения (браузеры) и Front-Server. Слой отвечает за обработку HTTP(S) запросов, маршрутизацию, первичную валидацию входных параметров, управление политиками доступа на уровне API-gateway (rate limiting) и формирование ответов.

7.2 Application Layer

Application Layer представлен Back-End и реализует бизнес-логику: операции с каталогами/меню, заказами, пользователями, администрированием, отчётностью и аудитом. Слой формирует единые правила валидации и контроля целостности данных.

7.3 Data Layer

Data Layer включает MySQL (транзакционные данные), Redis/Memcached (кэш), а также хранилище медиа/объектов.

7.4 Infrastructure Layer

Infrastructure Layer включает балансировщик нагрузки, очередь сообщений RabbitMQ, CDN, средства мониторинга и логирования, резервное копирование, а также средства сетевой защиты (Firewall/WAF при наличии).

Слой	Основные компоненты	Ответственность
Presentation	Client, Front-Server	HTTP(S) доступ, валидация, маршрутизация, ответы
Application	Back-End	Бизнес-логика, транзакции, интеграции
Data	MySQL, Redis/Memcached, Storage	Хранение данных, кэширование, медиа
Infrastructure	LB, RabbitMQ, CDN, Monitoring	Надёжность, асинхронность, доставка контента

9. Функциональная декомпозиция Back-End

Back-End реализован как модульный прикладной компонент (модульный монолит/сервисная композиция), в котором модули имеют чёткие границы ответственности и взаимодействуют через документированные внутренние интерфейсы.

9.1 Перечень модулей

- Auth & Identity: регистрация, аутентификация, управление токенами, восстановление доступа.
- RBAC & Organizations: роли, права, организации/точки, разграничение данных по тенанту.
- Catalog/Menu: категории, позиции, модификаторы, доступность, цены, витрины.
- Orders: корзина, оформление, статусы, обработка, история, отмены.
- Admin/Settings: настройки организации, параметры отображения, шаблоны, справочники.
- Reporting: агрегаты, отчёты, выгрузки, KPI.
- Audit & Logging: аудит действий, события безопасности, трассировка операций.
- Integration: адаптеры внешних систем (по мере подключения).

9.2 Контракты модулей (пример)

Модуль	Входные интерфейсы	Выходные интерфейсы/события
Catalog/Menu	REST: /api/v1/menu/*	События: menu.updated
Orders	REST: /api/v1/orders/*	События: order.created, order.status_changed
Auth	REST: /api/v1/auth/*	Токены JWT, события: auth.login
Reporting	REST: /api/v1/reports/*	Фоновые задачи через RabbitMQ

10. Модель данных и хранилища

10.1 Основные сущности (логическая модель)

- Organization (Организация/тенант)
- User (Пользователь)
- Role/Permission (Роль/право)
- Menu, Category, Item, Modifier (Меню и элементы)
- Order, OrderLine, StatusHistory (Заказ и строки заказа)
- MediaObject (Медиафайл)
- AuditEvent (Событие аудита)

10.2 MySQL (транзакционное хранение)

MySQL используется для хранения критичных бизнес-данных и выполнения транзакций. Операции оформления и изменения заказа выполняются в транзакциях с контролем целостности и конкурентного доступа.

10.3 Объектное/файловое хранилище (Storage-Server)

Медиафайлы (изображения, вложения) хранятся отдельно от транзакционных данных. Доступ предоставляется по HTTP GET через Storage-Server, при необходимости — через CDN.

10.4 Версионирование данных и миграции

Изменения схемы данных выполняются посредством миграций, фиксируемых в системе контроля версий. Миграции выполняются контролируемо в составе процедур развертывания (CI/CD).

11. Кэширование и производительность

Для ускорения доступа к часто используемым данным применяется кэширование по паттерну Cache-Aside. Ключевые сценарии: чтение меню/каталога, справочники, параметры отображения.

11.1 Redis

Redis используется для кэширования структурированных данных, хранения короткоживущих токенов/сессий (при необходимости), а также для распределённых блокировок в критичных сценариях.

11.2 Memcached

Memcached применяется для простого кэширования результатов чтения, не требующих сложных структур и операций.

11.3 Инвалидация кэша

- Инвалидация по событиям (menu.updated → сброс ключей меню).
- Инвалидация по TTL для данных со слабой консистентностью.
- Точечная инвалидация при изменении конкретных сущностей.

11.4 Производительность и проектные ориентиры

Показатель	Характеристика / ориентир
P95 latency для чтения меню	Определяется конфигурацией кэша, БД и числом одновременных запросов; целевые значения устанавливаются эксплуатационным регламентом
P95 latency для создания заказа	Определяется составом бизнес-проверок, состоянием очередей и конфигурацией БД; контролируется в промышленной среде
Доступность сервиса (SLA внутр.)	Зависит от выбранной схемы резервирования и условий эксплуатации
Макс. время ответа статического контента через CDN (P95)	Определяется конфигурацией CDN, сети и объёмом статического контента

12. Асинхронная обработка и очереди сообщений

RabbitMQ используется для выполнения фоновых задач и событийной интеграции. Асинхронность позволяет снизить время ответа пользовательских операций и повысить устойчивость к пиковым нагрузкам.

12.1 Типы задач

- Формирование и отправка уведомлений.
- Генерация отчётов и агрегатов.
- Постобработка заказов (например, синхронизация с внешними системами).
- Очереди обработки медиа (миниатюры, оптимизация изображений) — при использовании.

12.2 Гарантии доставки

Очереди настроены с подтверждениями (ack), повторной доставкой при сбоях и контролем количества попыток. Критичные события журналируются для последующего анализа и восстановления.

12.3 Идемпотентность

Для предотвращения повторной обработки применяется идемпотентность на уровне обработчиков задач (идентификаторы событий, контроль статуса обработки).

13. Хранение медиа и доставка контента (Storage/CDN)

Медиафайлы хранятся на Storage-Server и выдаются по HTTP GET. Для ускорения доступа и разгрузки Storage-Server используется CDN с кэшированием контента на краевых узлах.

13.1 Политики хранения

- Логическое разделение по организациям (тенантам).
- Контроль прав доступа к приватным объектам (подписанные ссылки/токены — при необходимости).
- Версионирование объектов для безопасного обновления изображений.

13.2 Резервное копирование медиа

Для медиа используется регулярное резервное копирование в выделенное хранилище. Сроки хранения резервных копий определяются эксплуатационной политикой.

14. Поддерживаемые варианты интеграций

- Экспорт данных меню, заказов и операционной информации (CSV/XLSX или иной согласованный формат) для учёта, аналитики и последующей обработки.
- webhook-уведомления о статусах заказов и иных событиях для внешних сервисов.
- Интеграция с SMS/e-mail шлюзами и иными предусмотренными каналами уведомлений (по настройке).

14.1 Варианты интеграций (примерные, правдоподобные)

- Экспорт данных (CSV/XLSX) для бухгалтерии и учёта.
- webhook-уведомления о статусах заказов для внешних сервисов.
- Интеграция с SMS/e-mail шлюзами для уведомлений (по настройке).

14.2 Контракты внешнего API (пример)

Метод	URI	Назначение	Защита
GET	/api/v1/menu/{org_id}	Получение меню организации	Публично/по ключу
POST	/api/v1/orders	Создание заказа	JWT/Anti-bot
GET	/api/v1/orders/{id}	Просмотр заказа	JWT

15. Безопасность и управление доступом

15.1 Транспортная безопасность

- Использование HTTPS, TLS 1.2+; запрет небезопасных шифров.
- Принудительная переадресация HTTP→HTTPS.
- HSTS (при включении в конфигурации).

15.2 Аутентификация и авторизация

Аутентификация реализована на основе токенов доступа (JWT). Авторизация — на основе ролевой модели (RBAC) с разграничением прав в пределах организации (тенанта).

15.3 Защита от типовых угроз

- Валидация входных данных и строгие схемы запросов.
- Защита от SQL-инъекций за счёт параметризованных запросов/ORM.
- Защита от XSS/CSRF средствами фреймворка и политиками безопасности.
- Ограничение частоты запросов (rate limiting), защита от brute-force.

15.4 Аудит и журналирование безопасности

Фиксируются события входа/выхода, изменения прав, критичные административные действия и операции с заказами. Журналы используются для расследования инцидентов и выполнения требований внутреннего контроля.

16. Надёжность, отказоустойчивость и восстановление

16.1 Отказоустойчивость компонентов

- Балансировщик нагрузки исключает недоступные экземпляры Front-Server по health-check.
- Back-End масштабируется кластером; возможна ротация экземпляров без простоя.
- MySQL — резервирование/репликация (при конфигурации повышенной надёжности).
- RabbitMQ — кластеризация/зеркалирование очередей (при необходимости).

16.2 Резервное копирование

Резервное копирование включает: дампы/снимки MySQL, резервирование конфигураций и медиа-хранилища. Периодичность определяется эксплуатационными требованиями; типовая схема: ежедневные инкрементальные и еженедельные полные копии.

16.3 RPO/RTO (целевые ориентиры)

Параметр	Значение (ориентир)
RPO (макс. потеря данных)	до 24 часов
RTO (время восстановления)	до 8 часов

Фактические показатели зависят от выбранной конфигурации и регламентов эксплуатации у заказчика/правообладателя.

17. Масштабируемость и балансировка

Горизонтальное масштабирование достигается добавлением экземпляров Front-Server и Back-End. Компоненты проектируются stateless либо с выносом состояния в специализированные хранилища.

17.1 Стратегии масштабирования

- Front-Server: масштабирование по количеству экземпляров за балансировщиком.
- Back-End: масштабирование по CPU/RAM и количеству реплик.
- Data Layer: оптимизация индексов, репликация чтения, кэширование.
- CDN: разгрузка выдачи статических объектов.

17.2 Управление конфигурацией

Параметры масштабирования и лимиты регулируются конфигурацией окружения и политиками оркестрации (при контейнерном развертывании).

18. Мониторинг, логирование и аудит

Эксплуатация поддерживается системой мониторинга метрик и централизованным логированием. Контролируются доступность, производительность, ошибки приложений и состояние очередей.

18.1 Метрики (пример)

- HTTP latency (P50/P95/P99), количество запросов, доля ошибок 4xx/5xx.
- Нагрузка CPU/RAM, использование диска, сетевой трафик.
- Состояние RabbitMQ (длина очередей, потребители, retries).
- MySQL (slow queries, репликация, блокировки).
- Cache hit rate (Redis/Memcached).

18.2 Логирование

Логи формируются в структурированном виде (JSON), содержат correlation-id для трассировки цепочек вызовов и собираются в централизованное хранилище логов. Доступ к логам ограничен.

18.3 Аудит действий

События аудита фиксируют изменения сущностей, операции администрирования, входы и попытки доступа. Данные аудита используются для контроля и расследования.

19. Развертывание и эксплуатация (DevOps)

19.1 Среды

- DEV — разработка и тестирование новых функций.
- STAGE — предпрод/интеграционное тестирование.
- PROD — промышленная эксплуатация.

19.2 CI/CD

Сборка и поставка выполняются через CI/CD pipeline: статический анализ, тесты, сборка контейнеров, публикация артефактов, развертывание. Каждое развертывание протоколируется.

19.3 Управление конфигурацией и секретами

Конфигурации хранятся отдельно от исходного кода. Секреты (ключи, пароли, токены доступа, параметры подключений) не размещаются в репозитории исходного кода и передаются в контур исполнения средствами защищённого хранения параметров окружения, принятыми у правообладателя или заказчика.

20. Описание программной реализации

20.1 Технологический стек и программные средства реализации

Компонент	Технология/ПО	Назначение
Web-клиент	JavaScript/TypeScript, React/Next.js или эквивалентный web-стек	Пользовательский и административный web-интерфейс
Серверная часть	PHP	API, бизнес-логика, обработка заказов и административных операций
Сборка web-компонентов	Node.js, npm/yarn	Установка зависимостей и формирование web-артефактов
Клиентское приложение	Flutter SDK, Dart SDK (при наличии в составе поставки)	Отдельный клиентский контур, если используется
DB	MySQL	Транзакционное хранение данных
Cache	Redis, Memcached	Кэширование и ускорение отклика
Queue	RabbitMQ	Асинхронные задачи и обмен сообщениями
Front-Server	Reverse Proxy / web-server	Маршрутизация, публикация статики, проксирование запросов
CI/CD и автоматизация сборки	GitLab CI или эквивалентный инструмент, при использовании	Контролируемая сборка и публикация поставки

В настоящем разделе приводятся программные средства реализации, подтверждённые документами о хранении исходного кода, зависимостях и фактической структуре ПО. Если отдельный компонент не используется в конкретной поставке, это определяется конфигурацией и составом внедряемого решения.

20.2 Организация исходного кода (пример структуры)

Исходный код организован по прикладным модулям и компонентам поставки. Типовая структура проекта может включать следующие каталоги и контуры:

- frontend/ - web-интерфейс пользователя и административная web-часть
- backend/ - серверная часть, API, бизнес-логика и обработка заказов
- mobile/ - клиентское приложение (при наличии в составе поставки)
- migrations/ - миграции и иные изменения структуры базы данных
- integrations/ - модули интеграции с внешними системами и сервисами уведомлений
- storage/ - правила работы с медиафайлами, загрузкой и доступом к объектам
- tests/ - автоматизированные тесты и вспомогательные сценарии проверки
- docs/ - эксплуатационная, пользовательская и техническая документация

20.3 Принципы разработки

- Контракты API документируются и версионируются (v1, v2 и т.п.).
- Покрытие ключевых модулей автоматизированными тестами.
- Линтинг/статический анализ для повышения качества.
- Код-ревью для изменений в критичных частях.

21. Обоснование технологической независимости и отсутствия иностранного контроля

Исходный код ПО «SellKit» разработан правообладателем и хранится в контролируемом репозитории. ПО не содержит встроенных механизмов удалённого управления третьими лицами и не требует обязательного обращения к иностранным облачным сервисам для работоспособности.

Применяемые компоненты инфраструктуры (ОС Linux, MySQL, Redis, RabbitMQ и др.) могут быть развернуты в инфраструктуре, размещенной на территории Российской Федерации, включая российские дата-центры. Архитектура допускает замену инфраструктурных компонентов на функционально эквивалентные решения без изменения логики системы.

21.1 Лицензии и зависимости (общие принципы)

- Используются библиотеки с распространёнными свободными лицензиями (например, MIT/Apache-2.0/BSD).
- Компоненты не накладывают ограничений на использование на территории РФ.
- Зависимости фиксируются в lock-файлах и контролируются в рамках процесса сборки.

22. Соответствие требованиям и нефункциональные характеристики

22.1 Нефункциональные требования (NFR)

Категория	Требование/характеристика
Доступность	Поддержка 24/7; контроль доступности health-check; цель 99,5%+
Производительность	Кэширование, CDN, асинхронные очереди; целевые P95 значения
Безопасность	HTTPS/TLS, RBAC, аудит, защита от типовых веб-угроз
Масштабируемость	Горизонтальное масштабирование Front/Back; stateless подход
Сопровождаемость	Модульная структура, миграции, CI/CD, тестирование
Переносимость	Развертывание в IaaS/on-premise; переносимость обеспечивается разделением компонентов и конфигураций

К системе предъявляются требования по доступности, целостности данных, контролю доступа, масштабируемости, сопровождаемости и переносимости. Конкретные численные показатели устанавливаются эксплуатационными документами и параметрами внедрения, при этом архитектура изначально рассчитана на многопользовательский режим, изоляцию данных по организациям и возможность расширения без переработки основных компонентных связей.

22.2 Документирование и трассируемость
Архитектурные решения, изменения схемы данных и версии API протоколируются. Система журналирования и аудита обеспечивает трассируемость ключевых операций.

23. Приложения

Приложение А. Матрица ролей и прав (пример)

Операция	Гость	Оператор	Администратор
Просмотр меню	Да	Да	Да
Создание заказа	Да	Да	Да
Изменение статуса заказа	Нет	Да	Да

Управление позициями меню	Нет	Да (огр.)	Да
Управление пользователями	Нет	Нет	Да
Просмотр аудита	Нет	Нет	Да

Приложение Б. Сценарии взаимодействия (последовательности)

Сценарий 1: Просмотр меню

8. Client → Front-Server: GET /menu/{org_id}
9. Front-Server → Back-End: GET /api/v1/menu/{org_id}
10. Back-End → Cache: попытка чтения меню
11. При промахе кэша: Back-End → MySQL: выборка данных, формирование ответа, запись в кэш
12. Back-End → Front-Server → Client: ответ JSON + ссылки на медиа через CDN/Storage

Сценарий 2: Создание заказа

13. Client → Front-Server: POST /orders (корзина)
14. Front-Server → Back-End: POST /api/v1/orders (валидация)
15. Back-End: транзакция MySQL — создание заказа и строк, фиксация
16. Back-End → RabbitMQ: событие order.created для фоновой обработки (уведомления/интеграции)
17. Back-End → Front-Server → Client: подтверждение заказа и идентификатор

Приложение В. Минимальный перечень API (версия v1)

- GET /api/v1/menu/{org_id} — получение меню
- GET /api/v1/items/{id} — получение позиции
- POST /api/v1/orders — создание заказа
- GET /api/v1/orders/{id} — просмотр заказа
- POST /api/v1/auth/login — вход
- POST /api/v1/admin/items — создание/изменение позиции меню (RBAC)

Дополнение 1. Расширенные пояснения
Ниже приведены дополнительные пояснения, предназначенные для экспертного рассмотрения архитектурных решений. Текст уточняет границы ответственности компонентов, обоснования применяемых паттернов и допущения по эксплуатации.

Компонентная изоляция и сетевые контуры
Публичный контур ограничен балансировщиком нагрузки и Front-Server. Back-End, базы данных, кэш и очередь сообщений размещаются во внутреннем контуре с ограничением доступа по ACL. Административные интерфейсы изолируются и защищаются политиками доступа.

Обработка ошибок и устойчивость
Ошибки классифицируются по типам (валидация, бизнес-ошибка, инфраструктурная ошибка). Для инфраструктурных ошибок реализуются повторные попытки, экспоненциальная задержка и circuit-breaker (при необходимости).

Требования к сопровождению и расширяемости
Модульная структура позволяет добавлять новые доменные области без изменения существующих контрактов. Версионирование API обеспечивает обратную совместимость и контролируемое обновление клиентов.

• Контролируемые миграции схемы данных и обратная совместимость при обновлениях (rolling update, blue/green — по выбранной схеме).

• Использование correlation-id в логах и трассировке для восстановления цепочек вызовов и анализа инцидентов.

• Ведение архитектурных решений в виде записей (ADR) и протоколирование изменений интерфейсов (версионирование API).

Дополнение 1.3 Уточнения по сопровождению, трассировке и изменениям
• Для фоновых задач — контроль количества попыток, протоколирование ошибок и возможность выделения dead-letter очереди (при конфигурации).

• Применение circuit-breaker для предотвращения каскадных отказов при деградации внешних/внутренних зависимостей.

• Использование тайм-аутов на сетевые вызовы и ограничение количества повторных попыток (retry) с экспоненциальной задержкой (backoff).

Дополнение 1.2 Уточнения по устойчивости, тайм-аутам и повторным попыткам
• Принцип наименьших привилегий (Least Privilege) при выдаче доступов и учётных данных сервисам.

• Ограничение входящих соединений на уровне firewall/ACL: разрешены только необходимые направления (LB→Front-Server, Front-Server→Back-End, Back-End→MySQL/Redis/Memcached/RabbitMQ).

• Разделение сети на DMZ и внутренний контур с запретом прямого доступа из публичной сети к Back-End и Data Layer.

Дополнение 1.1 Дополнительные уточнения по сетевой сегментации и доступу

24. Вывод

Архитектура программного обеспечения «SellKit» представляет собой масштабируемую распределённую систему, построенную на многоуровневой клиент-серверной модели с элементами микросервисного подхода и использованием асинхронных механизмов обмена сообщениями.

Выбранные структурные элементы, их интерфейсы и соединения обеспечивают:

- устойчивость к нагрузке,
- отказоустойчивость,
- масштабируемость,
- гибкость развития,
- возможность интеграции с внешними системами.