Промежуточный отчет по 5 практике

**Название проекта**: Система оптимизации ресторанного бизнеса

**Цель проекта**: Разработка документации для возможности дальнейшей реализации программного продукта.

**Задачи**:

* Определить идею и задачи проекта
* Определить и проанализировать целевую аудиторию
* Составить список основных функций и возможностей мобильного приложения
* Провести технический анализ, сравнить стеки технологий, выбрать подходящий для проекта
* Проанализировать возможные угрозы безопасности и способы защиты данных пользователей
* Разработать макет мобильного приложения
* Изучить виды тестирования и выбрать подходящий вид тестирования
* Составить документацию к мобильному приложению
* Сформулировать вывод о проделанной работе

**Методология:**

В данной работе была выбрана каскадная модель управления проектом. В соответствии с этой концепцией проект выполняется последовательно, и каждый из этапов ведет к следующему. Следующий этап не начинается до того, как будет завершен предыдущий.

Причины выбора данной методологии:

* есть определенные сроки проекта
* низкий риск изменений в проекте
* заранее известно какими инструментами и подходами будет пользоваться команда
* нет взаимодействия с заказчиком
* гибкость на первых этапах работы
* строгий менеджмент
* работа начинается с детального анализа требований заказчика и всех условий реализации проекта.

**Описание идеи:**

Система оптимизации ресторанов – это программа, с помощью которой ресторан может вести учет продуктов, регулировать работу персонала, получать информацию об излишках и недостаче продуктов.

**Анализ целевой аудитории**

Так как наша система предназначена для b2b рынка, ЦА являются владельцы бизнеса, менеджеры, а также пользователи системы (т. е. рядовые работники – в нашем случае официанты, менеджеры закупок, заведующие складом).

Далее представлены типы ресторанов, которые входят в ЦА системы, а также проблемы, с которыми они сталкиваются при работе и их решения с помощью системы:

1. Рестораны престижного класса

Проблема: поддержание высокого уровня обслуживания клиентов, отслеживание правомерности использования продуктов на кухне

Решение: система, позволяющая безошибочно управлять заказами, эффективно помогать гостям с выбором, а также позволяющая менеджерам и владельцам отслеживать правомерность использования продуктов.

2. Повседневные заведения

Проблема: оптимизация процессов управления бизнесом, отслеживание правомерности использования продуктов на кухне

Решение: система, позволяющая уменьшить временные и денежные затраты на ведение бизнеса, а также позволяющая менеджерам и владельцам отслеживать правомерность использования продуктов.

3. Современные рестораны

Проблема: оптимизация процессов управления бизнесом; использование современных технологий в организации работы заведения, в том числе и при общении с посетителями; отслеживание правомерности использования продуктов на кухне

Решение: система, соответствующая современным технологическим трендам, позволяющая безошибочно управлять заказами, эффективно помогать гостям с выбором, а также позволяющая менеджерам и владельцам отслеживать правомерность использования продуктов.

4. Рестораны семейного типа

Проблема: оптимизация процессов управления бизнесом, отслеживание правомерности использования продуктов на кухне

Решение: система, позволяющая уменьшить временные и денежные затраты на ведение бизнеса, а также позволяющая менеджерам и владельцам отслеживать правомерность использования продуктов.

5. Фаст-фуд рестораны

Проблема: осуществление обслуживания посетителей с максимальной скоростью, а также отслеживание правомерности использования продуктов на кухне.

Решение: система, позволяющая ускорить и сделать эффективнее процесс взаимодействия с клиентом, а также позволяющая менеджерам и владельцам отслеживать правомерность использования продуктов.

**Анализ рынка**

1. **R-Keeper**

R-Keeper — это одна из популярных POS[[1]](#footnote-1)-систем в России, которая предлагает широкий набор функций для управления заказами, столами, столами, складом и персоналом, а также имеет возможность интеграции с онлайн-заказами и позволяет создавать аналитические отчеты. Программный комплекс R-Keeper разделен на отдельные программы, которые предназначены для пользователей, представляющих систему общественного питания. Комплекс включает в себя такие модули как: "Официант", "Бармен", "Менеджер" и "Кассир". Каждый модуль соответствует своей направленности и предназначен для четкого выполнения определенных функций. Все действия потребуют от пользователя авторизации. Опознавание сотрудника, работающего с системой, происходит при помощи персональной магнитной карты.

В функциональные возможности модуля “Официант” входит формирование заказа, доставка заказа на кухню, распечатка предварительного счета, отмена определенных блюд и напитков с согласия менеджера, обработка дополнительного заказа и другие действия, которые входят в сферу компетенции официанта. Вся последовательность действий, которые выполняет официант, разработана программой так, что позволяет предотвратить возможные махинации. Случайные ошибки исключены, официантом открывается стол лишь на его имя.

В функциональные возможности модуля “Кассир” входит весь комплекс действий, который выполняет кассир. Это регистрация барменов и официантов в начале работы, оплата счетов, выбор способа оплаты, прием денег, пробитие чека и др. Система работает таким способом, что исключает закрытие рабочего дня в том случае, если в системе находится заказ, по которому не был пробит чек. При завершении операции происходит автоматическое списывание определенного количества продуктов на складе, которое ушло согласно калькуляционной карте на конкретное блюдо.

Бармену приходиться совмещать функции, который выполняют кассир и официант. Он имеет доступ к счетам, кассовому ящику и терминалу заказа. Он может закрыть счет, но в отличие от кассира, он не имеет полного доступа к таблице счетов, и работает только со своими счетами.

Обязанности менеджера различаются по уровню его полномочий. При работе с системой менеджеры (менеджер зала, офиса, администратор) используют один и тот же модуль, но от полномочий менеджера зависит количество доступных ему функций.

Администратор, функции которого выполняются хозяином заведения или управляющим, имеет самые широкие возможности при использовании системы. Под его контролем работа ресторана, кафе. Ему доступно изменение параметров используемой системы. В частности, можно передавать часть своих обязанностей менеджерам с менее широкими полномочиями.

В обязанности этих менеджеров входит заполнение и изменение меню ресторана, списков персонала и другой рабочей документации подобного плана. Ими выполняются отказы, закрывается кассовый день, снимаются отчеты.

Система R-Keeper многофункциональна, она предназначена не только для того, чтобы улучшить обслуживание и упорядочить учетные записи, но и для предотвращения случаев воровства и различных злоупотреблений. Система разработана таким образом, что для каждого сотрудника заданы определенные схемы работы. Чтобы произвести какое-либо обманное действие, персоналу потребуются сложные договоренности между собой, множество манипуляций и расчетов, на практике трудно выполнимых. При этом любой подозрительный фактор можно отследить по поступающим отчетам.

R-Keeper имеет обширную систему отчетности, благодаря которой владельцы заведения могут более полно анализировать ситуацию. Отчеты дают достаточную и наиболее полную информацию для бухгалтеров, менеджеров и непосредственного руководства. Задачей R-Keeper является сбор и систематизация информации, которая впоследствии предоставляется в виде отчетов. Так, с помощью R-Keeper можно просмотреть данные о выручке в общем, а также по блюдам, по столам, по дням или по часам. Также можно просмотреть данные об использовании бонусов и скидок, проверить все подозрительные моменты, отказы и т.д.

Помимо системы отчетов, которые информируют о состоянии дел ресторана после закрытия дня, в системе R-Keeper предусмотрена возможность контроля работы заведения в реальном времени. Для этого достаточно подключить к основной системе приложение "Монитор". Это приложение имеет удаленный доступ к сети интернет, и с его помощью можно просмотреть любую информацию, которая касается работы ресторана.

1. **iiko**

Эта система предлагает функции управления заказами, складом, персоналом, аналитикой и интегрированную CRM-систему для управления клиентской базой данных, акциями и лояльностью. Данный программный комплекс способен представлять данные не только в реальном времени, но и накапливать статистические данные для формирования отчетов. iiko имеет множество специальных решений, которые помогут управлять персоналом, выбирать музыку, получать отчеты на мобильный телефон, оптимизировать процесс ввода накладных и многое другое.

iikoFront — это модуль, который поможет правильно и быстро разобраться с вводом заказов. Он может легко и быстро ввести любой заказ, распределить столы и просмотреть статус каждого столика. Поддерживаются функции экранной клавиатуры, что помогает отправлять спецпожелания прямо на кухню. На экране всего можно видеть информацию о текущих акциях и скидках. Поддерживается функция деления чека на количество гостей. Кроме того, данная система позволяет оценить скорость и качество работы официантов.

iikoHostess — это модуль оптимизации, помогающий приводить банкеты. При помощи данной программы появляется возможность спланировать банкет, учесть все пожелания клиентов относительно приготовления и подачи блюд. Данное приложение поможет спланировать расстановку столов в зале, схема расстановки видна на экране. Реализована функция заказов по предварительной оплате и конечный расчет. Данное приложение способно формировать списки посетителей, резервирующих столы, таким образом, что из списка можно легко выбрать определенного посетителя.

iikoPayment - модуль автоматизации финансовых потоков, позволяет: настроить оплату различными способами: наличный и безналичный расчет, питание сотрудников, VIP; осуществить предоплату и полный расчет посетителей; возвращать и отменять платежи; совмещать работу кассира и проведение платежей; работать с такими платежными системами, как Плас-Тек и Pulsar; осуществлять совместную работу с гостиничными системами; автоматизировать процесс проверки транзакций с целью выявления неточностей; работать совместно с современными фискальными регистраторами.

iikoInventory модуль помогает вести учет товаров и их движения в режиме реального времени. Реализован процесс инвентаризации определенного товара, а также напитков. При этом учет напитков может производить по весу вместе с бутылкой, то есть, нет необходимости переливать напитки при учете.

При помощи iikoProduction модуля работа и учет продуктов, характеристик, заготовок и блюд и процесс создания более сложных блюд станет намного проще. Есть возможность производить групповой анализ себестоимости и распечатывать различные формы техкарт.

iikoManagementAccounting модуль позволит автоматизировать и упростить процесс кассовых операций, расчет заработной платы и расчета налоговых выплат.

iikoTimeManager модуль позволяет автоматизировать расчет заработной платы в зависимости от часов работы, собственной выручки и прочих показателей. Также здесь осуществляется планирование и контроль рабочего времени сотрудников. Приход и уход с работы фиксируются при помощи магнитных карт. За опоздания и неявку на работу штрафы начисляются автоматически.

iikoPersonalReport — это модуль персональных страниц каждого работника компании. Здесь содержится вся информация о рабочем времени, истории посещений, истории поощрений работника и прочая информация. Также можно получить полную информацию о затратах на каждого работника.

Модуль iikoVideoSecurity — это универсальный модуль видеонаблюдения. Здесь наблюдение ведется за каждым официантом. Любое событие может быть восстановлено по номеру чека, имени официанта или гостя или по наименованию операции. Все архивы могут храниться до 90 суток. Все данные системы защищены от постороннего доступа или утери.

1. **1C:Ресторан**

Это программное обеспечение разработано на базе 1C и предлагает широкий набор функциональных возможностей, включая управление заказами, персоналом, складом, отчетностью и бухгалтерией. Бесшовная интеграция позволяет автоматически получать заказы из сервиса.

Для автоматизации процесса обслуживания гостей достаточно ввести информацию о меню, назначить цены и можно приступать к регистрации продаж.

Рабочее место официанта позволяет вносить общий заказ гостей и комментарии по приготовлению/подаче блюд, закреплять заказ за посадочными местами стола для раздельной оплаты счета гостями. Поддерживается вариант с объединением/разделением заказов между столами. Для ускорения обслуживания гостей ресторана очень удобно использовать мобильное приложение для официанта, которое позволяет оперативно принимать и отправлять заказ на кухню и в бар, не отходя от гостя. Всегда актуальное меню и отображение стоп-листа поможет вовремя предложить гостю альтернативу, а уведомление о готовности блюда позволит максимально оперативно обслужить гостя. Индивидуальное создание схемы залов и столиков, а также загрузки собственных изображений, позволит работникам заведения четко визуализировать расположение мест и не даст запутаться в заказах. Метрдотель может не только управлять планом заведения и рассадкой, но и вносить данные о депозите, клиенте и т.д.

Все готовые к оплате заказы от официантов стекаются на рабочее место кассира, которое позволяет кассиру удобно управлять взаиморасчетами с гостями ресторана.

Рабочее место повара поддерживает и классическую схему печати заказов на сервис-принтерах кухни, и работу с дисплеем повара, где отображаются все текущие заказы, их состав и статус приготовления. Количество строк в заказе, колонок может быть настроено в зависимости от размера и разрешения экрана. С помощью дисплея повар может видеть, какие заказы стоят в очереди на приготовление, и изменять статус готовности блюд или заказа целиком, а официант – получать уведомление на мобильное устройство о готовности блюд.

Предусмотрено использование нескольких видов меню с ручным или автоматическим выбором при оформлении/корректировке заказа, доступность меню по дате, времени, дням недели. При этом для одного блюда могут быть установлены разные цены для каждого вида меню. Кроме видов меню поддерживается работа с модификаторами. При этом модификаторами могут быть как товары с ценой или без нее, так и инструкции по приготовлению. Модификаторы можно объединять в наборы. К модификаторам можно применять ограничения по количеству. С использованием механизма модификаторов реализована работа комплексными обедами.

Реализованы управление лояльностью – ручные скидки, дисконтные карты, бонусы, настройка и применение различных автоматических дисконтных схем, различные виды оплат при расчетах с гостем. Для расчета одного заказа можно комбинировать неограниченное количество видов оплат. В зависимости от настроек системы каждый вид оплаты может формировать документы реализации или документы списания.

Блок доставки позволяет полноценно отражать в системе каждый этап работы службы доставки заведения - от принятия заказа до доставки клиенту. После получения заказа менеджер вносит информацию о клиенте, его контактные данные, адресе доставки и самом заказе. Затем заказ перемещается на кухню, приобретая статус “Готовится” и по приготовлению отправляется на сборку и доставку. Каждый из сотрудников может отслеживать на каком этапе находится заказ, кто ответственный и т.д. Рабочее место Call-центр предназначено для приема заказов следующими способами: входящие звонки, получение заказов от агрегаторов доставки, получение заказов с web-сайта, мобильное приложение гостя.

Решение разработано на современной платформе "1С:Предприятие 8.3", которая позволяет: обеспечить высокую надежность, производительность и масштабируемость системы; организовать работу с системой через Интернет, в режиме тонкого клиента или веб-клиент, в том числе в "облачном" режиме; создавать мобильные рабочие места с использованием планшетов и иных мобильных устройств; настраивать интерфейс для конкретного пользователя или группы пользователей с учетом роли пользователя, его прав доступа и индивидуальных настроек.

**Требования к системе**

Для составления списка требований заказчика к системе был проведен анализ похожих на нашу систем: R-Keeper, iiko; а также отзывов к ним.

Список бизнес-требований:

1. Учет продуктов

2. Складской учет

3. Работа с меню

4. Технологическая карта

5. Различные профили системы для разных сотрудников

6. Управление правами доступа для сотрудников

7. Планирование закупки продуктов (и учет излишков и недостач)

8. Мобильное приложение для официантов

9. Безопасность данных

10. Высокая отказоустойчивость системы

**Функциональные требования**

Функционал системы разделен на профили - по категориям персонала, чтобы повысить удобство использования системы и уменьшения количества ошибок. Далее перечислены профили системы и их функционал

1. Официант

1.1. Начать новый заказ

1.2. Выбрать столик

1.3. Просмотреть меню

1.4. Редактировать заказ (добавить/удалить блюдо, настроить ингредиенты, настроить кол-во позиций)

1.5. Написать комментарий к заказу

1.6. Отправить заказ на кухню

1.7. Получать уведомления о готовности заказа

1.8. Сформировать счет

1.9. Отображать статус столиков («ждет обслуживания», «ждет заказ», «ждет расчета», «обслужен»)

1.10. Распечатать счет

2. Заведующий складом

2.1. Сканировать QR-код

2.2. Добавить информацию о продукте

2.3. Изменить информацию о продукте

2.4. Получать уведомления о заканчивающемся сроке годности и местонахождении продукта

2.5. Пометить продукт меткой «закончился»

3. Менеджер-закупщик

3.1. Получить статистику о расходе продуктов

3.2. Фильтровать статистику по дате/ количеству

4. Администратор

4.1. Добавить технологическую карту блюда

4.2. Редактировать технологическую карту блюда

4.3. Удалить технологическую карту блюда

4.4. Зарегистрировать пользователя

4.5. Назначить роли персоналу

4.6. Изменить роли персоналу

4.7. Получать информацию об излишках и недостаче продуктов

4.8. Просмотреть статистику блюд (частота заказов, отказов)

4.9. Фильтровать статистику по дате/ количеству

4.10. Оформить отказ гостя от заказа

4.11. Добавить купон (код купона и размен скидки)

5. Кухня

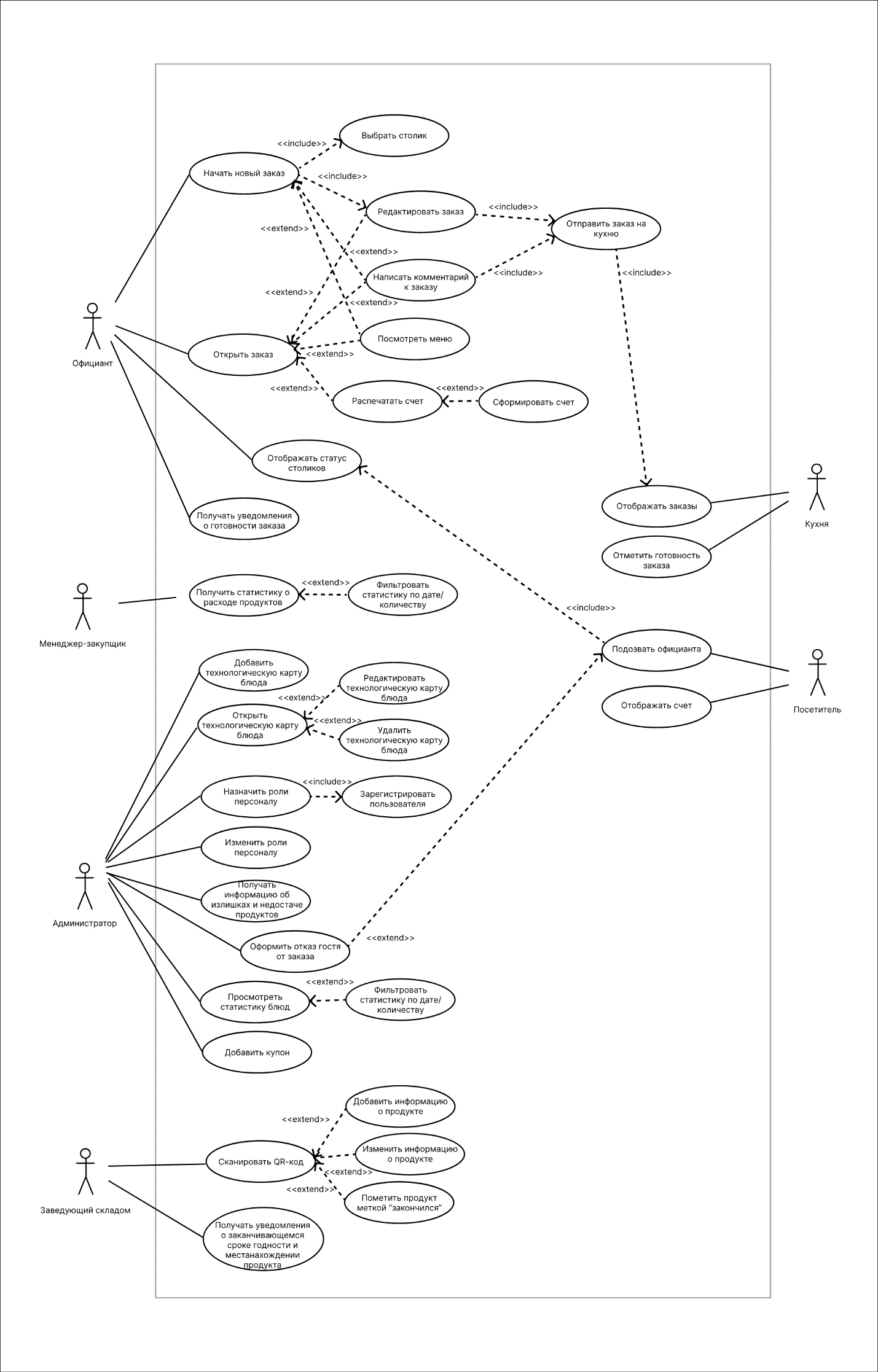
5.1. Отображать заказы

5.2. Отметить готовность заказа

6. Посетитель

6.1. Отображение счета

6.2. Подозвать официанта



Use case диаграмма

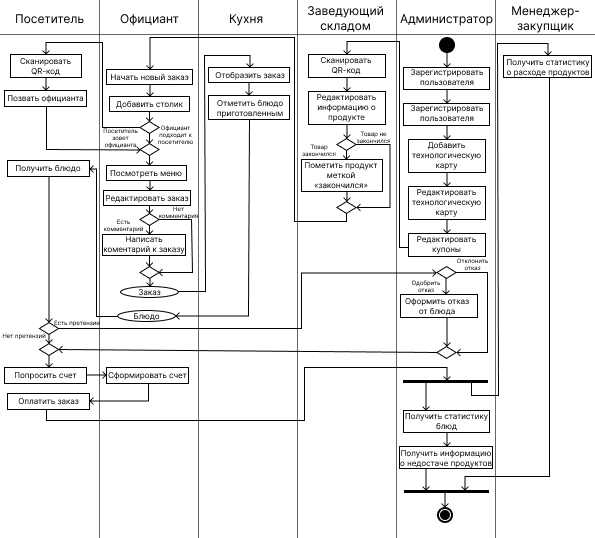


Диаграмма активности

**Внутренняя структура приложения**

1. КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

Клиент-серверная архитектура - это модель организации приложений, в которой задачи распределены между клиентами и серверами. В такой архитектуре клиент, обычно являющийся пользователем или программой, запрашивает услуги или ресурсы у сервера, который отвечает на запросы, предоставляя необходимые данные или функциональность.

В клиент-серверной архитектуре используется три компонента:

* **Клиент** — программа, которую мы используем в интернете. Чаще всего это браузер, но может быть и другая отдельная программа
* **Сервер** — компьютер, на котором хранится сайт или приложение. Когда мы заходим на сайт магазина, мы обращаемся к серверу, на котором находится сайт
* **База данных** — программа, в которой хранятся все данные приложения. Для магазина это будет база клиентов, товаров и заказов

Рассмотрим достоинства и недостатки данной архитектуры.

**Достоинства**:

1. Отсутствие дублирования. Весь сайт или приложение хранится на одном компьютере-сервере. Это позволяет использовать его с разных устройств, будь то компьютер или мобильный телефон
2. Надежность: при использовании клиент-серверной архитектуры, отказ одного компонента системы не влияет на работу других компонентов
3. Безопасность: централизованное управление сервером обеспечивает возможность контроля доступа и защиты данных, что делает клиент-серверную архитектуру более безопасной по сравнению с другими IT-архитектурами
4. Производительность. Серверы обычно производительнее, чем компьютеры пользователей. Это позволяет обрабатывать тысячи запросов от сотни разных пользователей одновременно.
5. Масштабируемость. Клиент-серверная сеть обладает высокой степенью масштабируемости. При необходимости можно добавить больше клиентов и серверов в сеть, что позволит увеличить мощность сервера и обработать больше запросов без остановки работы сети.
6. Поддержка. Клиент-серверная архитектура позволяет взаимодействовать с различными платформами и технологиями, используя открытые стандарты для обмена данными и коммуникаций.

**Недостатки**:

1. Перегрузка сервера. Популярные порталы могут получать большое количество запросов одновременно. Например, при десятке миллионов запросов в секунду сервер может не выдержать и отключиться
2. Зависимость от сервера: клиент не может работать без сервера. Если сервер(а) недоступен или имеет проблемы, все клиенты будут неработоспособны или испытывать проблемы с функциональностью
3. Ограниченность: при использовании клиент-серверной архитектуры возникают ограничения на количество одновременно подключенных клиентов и на пропускную способность сети
4. Высокая стоимость оборудования
5. БЕССЕРВЕРНАЯ АРХИТЕКТУРА

Бессерверная архитектура – это способ создания и запуска приложений и сервисов без необходимости управления инфраструктурой. Приложение по‑прежнему будет работать на серверах, но управление этими серверами облачный провайдер полностью берет на себя. Команде больше не придется заниматься выделением ресурсов, масштабированием и обслуживанием серверов для запуска приложений, баз данных и систем хранения данных.

**Достоинства**:

1. Не нужно управлять серверами и средами исполнения, что сберегает время и силы на разработку.
2. Экономия на ресурсах. Бессерверные функции запускаются только тогда, когда они необходимы, что потенциально позволяет сэкономить значительные ресурсы.
3. Масштабируемость. Модульная организация бессерверных приложений упрощает разработчикам масштабирование приложений. В облаке над новыми функциями можно работать независимо, а затем постепенно добавлять их в работающее приложение.
4. Надежность и доступность. Приложение, разбитое на слабо связанные функции, создает гораздо меньше зависимостей между ними. Поэтому, если одна функция дает сбой, приложение продолжит работу.

**Недостатки**:

1. Ограничения технологического стека. Бессерверная архитектура менее гибкая, чем микросервисы. Эти платформы часто поддерживают только отдельные языки программирования, фреймворки и инструменты
2. Привязка к провайдеру. Полностью бессерверные приложения тесно связаны с платформой поставщика услуг, для которой они были созданы. Но каждая облачная платформа поддерживает разные технологии, а это означает, что перенастройка бессерверных функций для платформы другого провайдера может оказаться трудной задачей.
3. Низкая гибкость: у многих людей возникают проблемы с подключением к одному поставщику бессерверной среды в течение длительного периода.
4. Безопасность. Поставщик облачных служб может одновременно запускать код нескольких своих клиентов на одном сервере. Данные вашего приложения могут быть раскрыты, если общий сервер настроен неправильно.
5. Производительность. Когда функции вызываются после продолжительного периода бездействия, возникают так называемые «холодные» старты (часто встречаются в бессерверных установках), которые приводят к тому, что выполнение кода приостанавливается на большее число секунд.

ПОСТРОЕНИЕ ПРИЛОЖЕНИЯ

1. МОНОЛИТНАЯ АРХИТЕКТУРА

Концепция монолитного программного обеспечения заключается в том, что различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте.

Монолитная архитектура удобна для работы небольших групп, поэтому многие стартапы выбирают этот подход при создании приложения. Компоненты монолитного программного обеспечения взаимосвязаны и взаимозависимы, что помогает программному обеспечению быть самодостаточным.

**Достоинства:**

1. Простой деплоймент. Благодаря монолитному ядру разработчикам не нужно развертывать изменения или обновления по отдельности, поскольку они могут сделать это сразу и сэкономить много времени
2. Простота и скорость разработки.  В монолитной архитектуре разработчик может быстро начать реализовывать бизнес-логику, вместо того чтобы тратить время на размышления о межпроцессном взаимодействие
3. Меньше сквозных проблем. Большинство приложений зависят от множества межкомпонентных задач, таких как контрольные журналы, ведение логов, ограничение скорости и т. д. Монолитные приложения гораздо легче учитывают эти вопросы благодаря своей единой кодовой базе. К этим задачам проще подключать компоненты, когда все работает в одном приложении
4. Производительность. При правильной сборке монолитные приложения обычно более производительны, чем приложения на основе микросервисов благодаря общему коду и памяти.

**Недостатки:**

1. Привязанность к одному стеку технологий. Добавление новых технологий или фреймворков чревато перепроектированием всего приложения. Такое скудное технологическое разнообразие ограничивает ваши возможности и замедляет дальнее развитие приложения
2. Громоздкость. С ростом проекта монолит так же растет, становится все более запутанным и непонятным для других разработчиков.
3. Ограниченная гибкость, так как каждое обновление приложения требует полного повторного развертывания.
4. Масштабируемость. Масштабирование монолитного приложения может стать сложной задачей, поскольку все приложение должно масштабироваться вместе, а не только необходимые части. Это отсутствие гибкости часто увеличивает затраты и снижает эффективность при работе с большими нагрузками.
5. Поддержка. поддержка монолитной кодовой базы становится все более сложной задачей по мере роста сложности и размера приложения. Эта трудность связана с тесной связью компонентов, из-за чего разработчикам сложнее модифицировать или отлаживать приложение, не затрагивая другие части.
6. Производительность. Является как плюсом, так и минусом. Достоинством она может служить при разработке небольших приложений. Однако если проект растет, число пользователей и функций увеличивается, то и производительность сильно падает.
7. Неработоспособность при частичном отказе. Если что-то в монолите работает некорректно или в целом не работает, то и весь монолит не будет работать.
8. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

**Микросервисная архитектура** — один из подходов проектирования, при котором единое приложение строится как набор небольших сервисов. Каждый модуль работает в собственном процессе и взаимодействует с другими модулями. Модули работают независимо и построены вокруг бизнес-потребностей и выполняют определённую функцию. Например, хранят данные пользователей или отвечают за аутентификацию.

**Достоинства**:

1. Масштабируемость. Каждый микросервис можно масштабировать независимо от других, исходя от его конкретных потребностей. Такая детализированная масштабируемость позволяет эффективно распределять ресурсы и обеспечивает оптимальную производительность, особенно при работе с меняющимися требованиями к рабочей нагрузке
2. Технологическое разнообразие. Каждый сервис работает обособленно, так что можно подобрать для него разные технологии и фреймворки – если только их интерфейсы не зависят от технологий. Это открывает целый мир возможностей и позволяет командам выбирать лучшие инструменты для реализации конкретных задач.
3. Простота разработки. Над каждым микросервисом может работать отдельная команда разработчиков, что одновременно и ускоряет работу, и упрощает ее. Команды не обязаны координировать развертывание.
4. Эффективный деплоймент. Более короткое время запуска и возможность развертывания микросервисов независимо друг от друга действительно выгодны для CI / CD. По сравнению с обычным монолитом он намного плавнее.
5. Работоспособность при частичном отказе системы. В микросервисах изолируемые разломы лучше по сравнению с монолитным подходом. Хорошо спроектированная распределенная система переживет сбой одного сервиса.

**Недостатки**:

1. Сложность. Несмотря на простоту разработки (ведь разрабатывать отдельные сервисы намного легче, чем целый монолит), микросервисы сложно координировать. Для координации взаимодействия между сервисами, обеспечения согласованности данных и обработки сбоев необходимо скрупулезное проектирование и внедрение.
2. Тестирование и отладка. Отладка микросервисных приложений может оказаться более сложной задачей, поскольку затронутые микросервисы могут разрабатываться разными людьми и командами. Это означает, что для отладки потребуется координировать тесты, обсуждения и сбор отзывов между многими членами команды, а это дополнительные затраты времени и ресурсов.
3. Транзакции. Транзакции легче проводить в монолите. Решением этой проблемы на микросервисах является Saga Pattern. Хорошее решение, но все же слишком громоздкое для реализации на практике.
4. Оборудование. Помимо вышеперечисленных сложностей, для микросервисов также может потребоваться больше оборудования, чем для традиционных монолитов. Иногда микросервисы могут превзойти один монолит, если есть его части, которые требуют масштабирования до предела.

ТИПЫ ВЕБ-ПРИЛОЖЕНИЯ

1. SPA

SPA или Single Page Application — это одностраничное веб-приложение, которое загружается на одну HTML-страницу. Благодаря динамическому обновлению с помощью JavaScript, во время использования не нужно перезагружать или подгружать дополнительные страницы. На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки нужные элементы просто подгружаются.

**Достоинства**:

1. Высокая скорость. Все ресурсы загружаются за одну сессию, а во время действий на странице данные просто меняются, что очень экономит время.
2. Гибкость и отзывчивость пользовательского интерфейса. За счет того, что веб-страница всего одна, проще построить насыщенный интерфейс, хранить сведения о сеансе, управлять состояниями представлений и анимацией;
3. Кэширование данных. Приложение отправляет всего один запрос, собирает данные, а после этого может функционировать в offline-режиме.
4. Кроссплатформенность. Работает на любых устройствах и операционных системах.
5. Поддержка. Легче мониторить и исправлять ошибки.

**Недостатки**:

1. Безопасность. Высокая уязвимость к XSS-атакам.
2. Не сохраняется история перехода по страницам, поэтому не поддерживаются навигационные кнопки браузера.
3. Дольше загружается при первом посещении.
4. Может возникнуть проблема совместимости с другими браузерами
5. Требуется JavaScript
6. MPA

MPA или Multi Page Application — это многостраничные приложения, которые работают по традиционной схеме. Это означает, что при каждом незначительном изменении данных или загрузке новой информации страница обновляется. Такие приложения тяжелее, чем одностраничные, поэтому их использование целесообразно только в тех случаях, когда нужно отобразить большое количество контента.

**Достоинства**:

1. Безопасность. Меньшая уязвимость к атакам.
2. Простая SEO-оптимизация, так как можно оптимизировать каждую из страниц под нужные запросы.
3. Меньший стек технологий. Как правило для разработки многостраничного приложения требуется меньший стек технологий.
4. Пользовательский опыт. За счет простого интерфейса и навигации многостраничные приложения намного удобнее для пользователей.

**Недостатки**:

1. Тесная связь между бэкендом и фронтендом, поэтому их не получается развивать параллельно
2. Длительные сроки разработки
3. Низкая скорость взаимодействия, так как MPA перезагружают контент, когда пользователь взаимодействует с приложением.
4. Поддерживать и обновлять многостраничные сайты намного дольше и тяжелее, чем одностраничные.

 Стоит также отметить, что SPA часто является частью архитектуры многостраничного вебсайта. К примеру, основная часть ресурса может быть многостраничной, а личный кабинет реализован в виде SPA.

1. ПРОГРЕССИВНЫЕ ВЕБ-ПРИЛОЖЕНИЯ

Progressive Web App (PWA) — это адаптация сайта компании под мобильное устройство в виде приложения. PWA приложения повторяют функционал мобильного приложения. Веб-страницы размещаются на домене, а на устройствах открываются через браузеры. Можно назвать прогрессивные веб-приложения продвинутыми версиями сайта, которые адаптируются под устройство пользователя. При этом специальная разработка под Android и IOS не нужна.

**Достоинства**:

1. Пользовательский опыт. Так как это приложение практически не отличается от мобильного, пользователю удобно и привычно его использовать. Также там приходят знакомые всем пользователям Push-уведомления, что может вовлечь пользователя в приложение и бренд еще больше.
2. Скорость разработки. Для создания PWA приложений существуют специальные конструкторы, что ускоряет и упрощает разработку.
3. Меньший вес, чем у мобильного приложения
4. Быстрая скорость работы (выше, чем у сайта) и возможность перейти в офлайн-режим.
5. Разработка дешевле на 70%, чем у мобильных приложений

**Недостатки**:

1. Ограниченный функционал, так как PWA приложения не могут в полной мере использовать весь функционал устройства.
2. Для добавления PWA нужно посетить первый раз веб-приложение в браузере
3. Большой расход батареи

Какие архитектуры могут быть выбраны для разработки нашего приложения?

В первую очередь, в начале разработки, пока приложение не стало популярным, можно использовать клиент-серверную архитектуру. Однако с распространением приложения и ростом пользователей (а пользователей в каждом ресторане/кафе может быть очень много, около 50 человек) нужно будет приобретать и обеспечивать большее количество серверов, в результате чего можно будет рассмотреть переход на бессерверную архитектуру, так как Она отлично работает для приложений, насыщенных клиентами, и приложений, которые быстро растут и нуждаются в неограниченном масштабе.

Касательно внутреннего устройства приложения, будет разумно использовать микросервисную архитектуру в целях распределения нагрузки между серверами и сервисами.

Так как мы разрабатываем веб-приложение, будет разумно разрабатывать не просто веб-сайт, а прогрессивное веб-приложение, чтобы пользователям было удобно и быстро его использовать.

В совокупности микросервисная архитектура и тип PWA могут обеспечить быстрые сроки разработки, а также упростить ее.

RESTful API:

ОФИЦИАНТ

* GET/menu

Просмотреть меню

* POST/orders

Создать заказ

* PUT/orders/:orderId

Обновить информацию о заказе

* GET/orders/:orderId

Просмотреть информацию о заказе

ЗАВЕДУЮЩИЙ СКЛАДОМ

* PUT/products/:productId

Обновить информацию о продукте, QR код предоставляет Id

* DELETE/products/:productId

Удалить продукт

* POST/products

Добавить новый продукт

* GET/products/:productId

Получить информацию о продукте

* GET/products

Получить информацию о продуктах

* POST/consignment/:consignmentId

Добавить информацию о поставке

КУХНЯ

* GET/orders

Получить информацию о заказах

* GET/orders/:orderId

Получить информацию о заказе

* GET/recipes/:dishId

Получить технологическую карту блюда

* PUT/orders/:orderId

Отметить готовность заказа

ЗАКУПЩИК

* GET/purchases/:period

Получить список закупок

* GET/statistics

Получить высчитанную системой статистику использования

АДМИН

* POST/users/register

Зарегистрировать пользователя

* PUT/users/:employeeId

Обновить информацию о пользователе

* DELETE/users/:employeeId

Удалить пользователя

* GET/statistics

Получить статистику

* POST/recipes

Создать технологическую карту блюда

* PUT/recipes/:dishId

Обновить технологическую карту блюда

* DELETE/recipes/:dishId

Удалить технологическую карту блюда

* GET/products

Получить информацию по продуктам

* POST/purchases

Создать список закупок

* PUT/purchases

Изменить список закупок

* POST/menu

Создать меню

* DELETE/menu/:dishId

Удалить блюдо из меню

* PUT/menu/:dishId

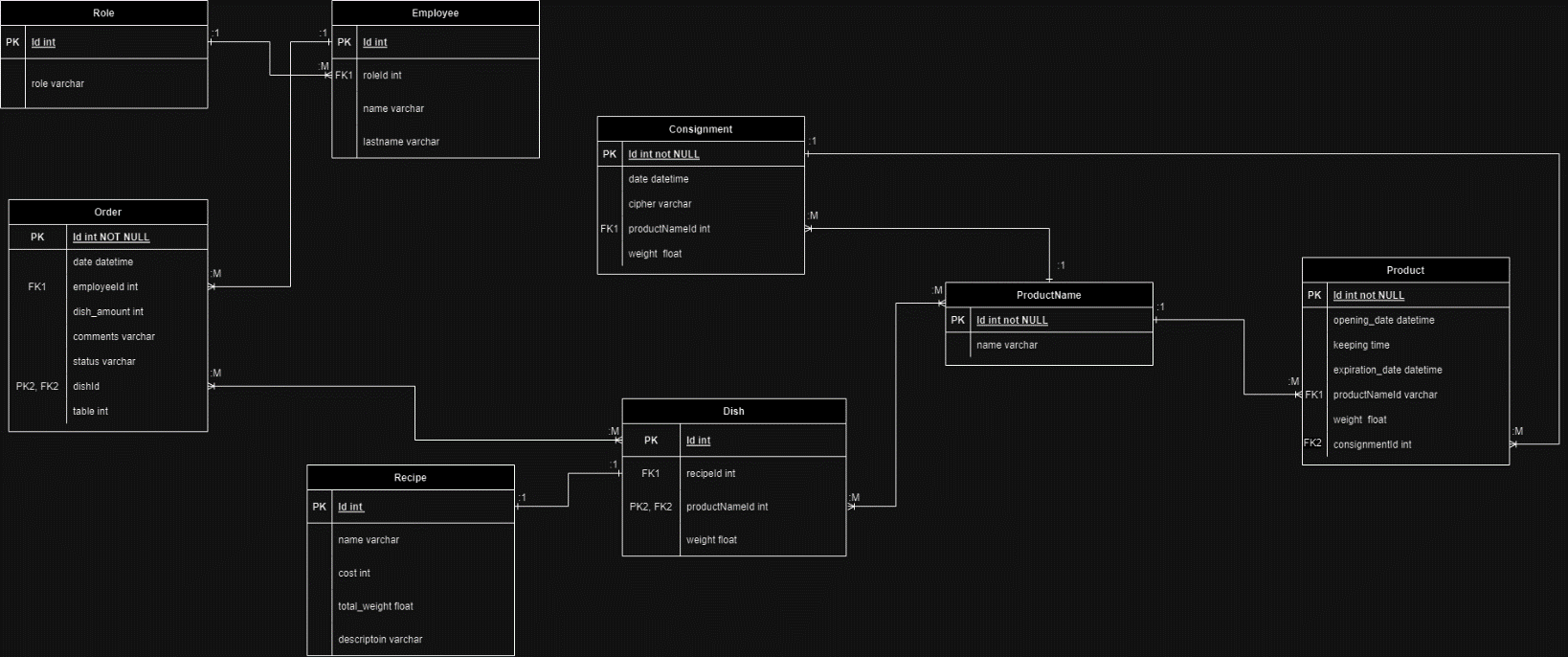
Обновить информацию о блюде

ПОЛЬЗОВАТЕЛЬ ПРИ СКАНИРОВАНИИ QR-КОДА НА СТРАНИЦЕ

* GET/order/:tableId

**Выбор базы данных**

Наша система рассчитана на хранение большого объема данных, поэтому нужно выбрать соответствующую базу данных.



ER диаграмма

1. MySQL

Одна из самых используемых и удобных систем управления базами данных.

1. Это реляционная СУБД, значит данные в ней хранятся в виде таблиц.
2. Поддерживает JSON-формат данных
3. Бесплатная
4. Кроссплатформенная, работает на Linux, Windows, FreeBSD
5. Входит в популярные стеки LAMP и WAMP (Linux/Windows, Apache, MySQL, PHP)
6. Высокая производительность, что отлично подходит для веб-приложений, однако только для малых или средних проектов. Для больших проектов, где могут быть миллионы строк, производительность сильно падет.
7. Не поддерживает полностью ACID
8. Гибкость и поддержка движков, с помощью которых можно обеспечить соответствие требованиям ACID
9. Позволяет добавлять до 50 миллионов строк в таблицы
10. Высокий уровень безопасности
11. Развитое коммьюнити
12. Трудно переходить на другие СУБД за счет того, что MySQL не соответствует требованиям SQL
13. Клиент-серверная архитектура

Таким образом, MySQL подойдет для веб-сайтов и приложений, для корпоративных баз данных начального уровня, проектов маленького и среднего размеров, которые не требуют огромного числа записей.

Ее легко и удобно использовать, она подойдет для проектов, в которых важна безопасность данных.

1. MongoDB
2. Это файловая СУБД, данные могут храниться в форматах JSON, BSON, благодаря чему можно хранить фото, видео и аудиоматериалы и более эффективно хранить данные, например, об одном предмете/продукте/человеке.
3. Кроссплатформенность. Можно использовать с Java, PHP, Python, Perl, C#.
4. Гибкость. За счет того, что это NoSQL СУБД, она подходит и для структурированных, и для неструктурированных данных
5. Плохо обеспечивает соответствие требованиям ACID
6. Применяется для быстрой разработки, потому как в этой СУБД можно менять любые данные в любое время.
7. В 2022-2023 прекращена работа на территории РФ и Беларуси
8. Хорошая отказоустойчивость за счет журналирования и репликации
9. Проблема дублирования данных
10. Высокое использование памяти
11. Лимит файлов – 16МБ

Данная СУБД плохо подходит для проектов, которые требуют высокой безопасности и соответствия требованиям ACID, проектов с бизнес-логикой и транзакциями.

Однако для создания соцсетей, работы с машинным обучением, интернет-магазинов, работы с геоданными эта СУБД отлично подойдет.

Традиционно эту СУБД используют в проектах, где важное значение имеет масштабируемость

1. Oracle

Самая популярная СУБД

1. Объектно-реляционная СУБД, то есть воспринимает данные как объекты со свойствами, методами
2. Клиент-серверная СУБД, связь между клиентом и сервером обеспечивает SQL \*Net. Однако сервер для этой СУБД должен обладать высокой мощностью
3. Высокий уровень безопасности
4. Хорошая отказоустойчивость. Благодаря утилитам, если один из серверов откажет, его работу подхватит другой.
5. Масштабируемость и высокая производительность. За этот и предыдущий пункт отвечают функции Oracle RAC – кластеризация, Oracle Streams – утилита, отвечающая за обмен информацией между бд, отвечает за синхронную работу серверов, Oracle Data Guard – вспомогательные серверы делают бэкапы из главного
6. Запросы пишутся на PL/SQL, с помощью которого запросы можно собрать в одном блоке кода и написать для них логику
7. Дорогая лицензия
8. Развитое коммьюнити
9. Поддержка разных форматов данных (сетевые адреса, данные JSON, координаты геопозиций)
10. Соответствие требования ACID
11. Oracle Machine Learning, представляющая собой машинное обучение с анализом данных в реальном времени
12. Поддерживаются все популярные языки программирования (Node.js, Python, PHP, Ruby, C++, Java)
13. Введены ограничения для РФ с 2022 года

Эта СУБД подойдет для крупных проектов с миллионами записей в базе данных, миллионами пользователей и требующих высокой производительности. Однако высокая стоимость пользования подойдет для успешных и прибыльных проектов, крупных корпораций

1. PostrgreSQL
2. Объектно-реляционная СУБД
3. Открытый исходный код
4. Клиент-серверная архитектура
5. Поддержка популярных языков программирования (Python, Java, Perl, PHP, C++)
6. Кроссплатформенность
7. Поддержка разных форматов данных (сетевые адреса, данные JSON, координаты геопозиций)
8. Размер БД не ограничен, зависит только от свободной памяти в месте хранения
9. Соответствует требованиям ACID
10. Запросы на SQL и на собственном диалекте
11. Репликация
12. Возможность без потерь перенести данные из другой СУБД
13. Одновременная модификация базы, доступ к базе с нескольких устройств
14. Высокая производительность
15. Повышенное потребление ресурсов
16. Может уступать в функциональности аналогам

PostgreSQL лучше подходит для корпоративных приложений с частыми операциями записи и сложными запросами, а также для организаций, которым приходится иметь дела с крупными и потенциально бесконечно масштабируемыми базами данных

В ходе исследования и сравнения различных СУБД была выбрана СУБД PostgreSQL. Наш проект предполагает взаимодействие с большими базами данных, в которых могут храниться миллионы строк. При этом одновременно получать доступ к базам данных должны сотни, а то и больше человек: официанты, повара, кладовщики, закупщики, которых в ресторанах и кафе могут быть десятки.

Также компания, владеющая этой СУБД не вводила ограничения на территории РФ, что является большим плюсом. В начале разработки проекта бюджет не будет большим, а эта СУБД бесплатна и очень производительна. С ростом нашей системы и ростом бюджета можно будет рассмотреть возможность перехода на Oracle DB, а в противном случае приобрести большее количество серверов и продолжить использование PostgreSQL.

Помимо вышеперечисленного, для нашего проекта важно соответствие требованиям ACID – атомарность, согласованность, изолированность, надежность. Важно, чтобы данные не перемешивались друг с другом, данные каждого ресторана были надежно защищены и все транзакции и запросы выполнялись полностью.

**Выбор фреймворка**

Ценность прогрессивных веб-технологий неоспорима, поэтому на рынке появляется все больше фреймворков и инструментов для создания PWA. Выбор фреймворка является решающим первым шагом для любого подобного проекта. Есть несколько определяющих факторов, которые необходимо учитывать, чтобы сделать правильный выбор.

Поскольку каждый фреймворк обладает своими возможностями, ожидаемые наборы функций будущих приложений, их размер, сложность, ожидаемая масштабируемость, а также наличие мультимедийных или интерактивных элементов — все это играет важную роль при выборе решения. Еще одним фактором, который следует учитывать, является команда разработчиков, их опыт, а также то, будут ли они готовы оперативно обслуживать и обновлять приложение при необходимости.

В ходе практической работы был проведен анализ следующих популярных фреймворков для разработки прогрессивного веб-приложения:

**1. React**;

**2. Angular**;

**3. Ionic**;

**4. Vue**;

**5. Polymer**;

**6. PWA Builder**;

**7. Svelte**.

Далее рассмотрим подробнее каждый из них.

**1. React**

React, созданный Facebook в 2013 году, представляет собой JS-библиотеку, которая хорошо известна разработчикам благодаря своему широкому разнообразию инструментов.

Данный фреймворк использует JSX для синхронизации с форматом HTML. Вы также можете воспользоваться бесчисленным  количеством доступных пакетов для дальнейшего расширения вашего проекта.

В настоящее время React — одна из ведущих библиотек для разработки интерфейса любых веб-приложений: от одностраничных (SPA) до прогрессивных (PWA).

**2. Angular**

Angular, созданный Google в 2010 году, является одним из старейших и популярных JavaScript-фреймворков. Известно, что это самый продвинутый и "полнофункциональный" из всех доступных PWA-фреймворков.

Все компоненты Angular являются модульными, то есть совместимыми с другими фреймворками и инструментами для разработки веб-приложений.

**3. Ionic**

Ionic — это веб-фреймворк для приложений, созданных с использованием HTML, CSS и JavaScript. С момента своего создания в 2013 году и до недавнего времени он был связан с элементами AngularJS и зависел от них. Без них он был пригоден только для рудиментарных приложений. Будучи сейчас фреймворком с открытым исходным кодом, он основан на Apache Cordova и Angular.

Ionic содержит огромные библиотеки элементов как для iOS, так и для Android. Инструмент WebView предоставляет веб-страницы в веб-браузере устройства для предпросмотра ваших приложений.

**4. Vue**

Vue имеет некоторые сходства с React, поскольку они оба используют Virtual DOM для эффективного и облегченного представления реального DOM.

Vue использует и улучшает HTML и CSS. Этому PWA-фреймворку удалось упростить код, обеспечив упрощенный процесс разработки и высокоскоростной рендеринг. Благодаря этому данный фреймворк отлично подходит для начинающих фронтенд-разработчиков.

**5. Polymer**

Polymer создан компанией Google и имеет открытый исходный код. Он содержит широкий спектр веб-компонентов, шаблонов и инструментов для разработки, благодаря которым данный фреймворк становится отличным выбором для создания PWA.

Итоговый продукт будет доступным и легко адаптируемым, поскольку компоненты и инструменты Polymer поддерживаются в огромном количестве браузеров. Важно также отметить, что данный PWA-фреймворк использует чистые HTML, CSS и JavaScript: это делает его автономным.

**6. PWA Builder**

PWA Builder, фреймворк с открытым исходным кодом, управляемая сообществом, был основан Microsoft. Он поддерживает разработку приложений как для Android, так и для iOS.

Данный фреймворк был создан для того, чтобы быть лучшим в том, что он делает, а именно превращает сайты в функциональные, привлекательные и быстрые PWA на самых разнообразных платформах.

**7. Svelte**

Svelte, в отличие от обычных фреймворков, использует новый подход, перекладывая большую часть работы с браузера на этап компиляции. Эта уникальная характеристика позволяет Svelte создавать высокооптимизированные, легкие и производительные приложения, которые быстро загружаются и плавно работают.

В результате разработчики могут создавать многофункциональные веб-приложения с использованием части кода из других фреймворков, что приводит к повышению эффективности и простоте обслуживания при разработке PWA.

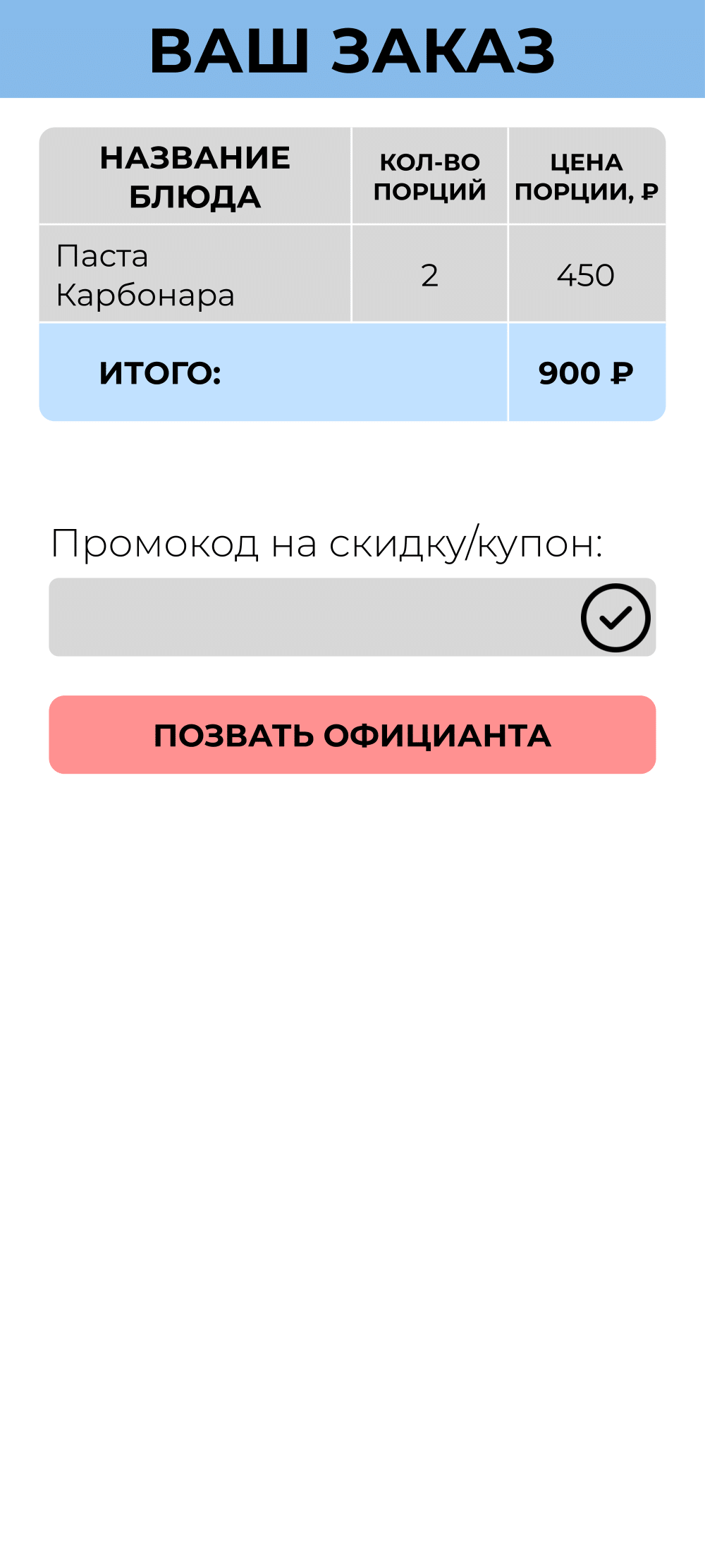
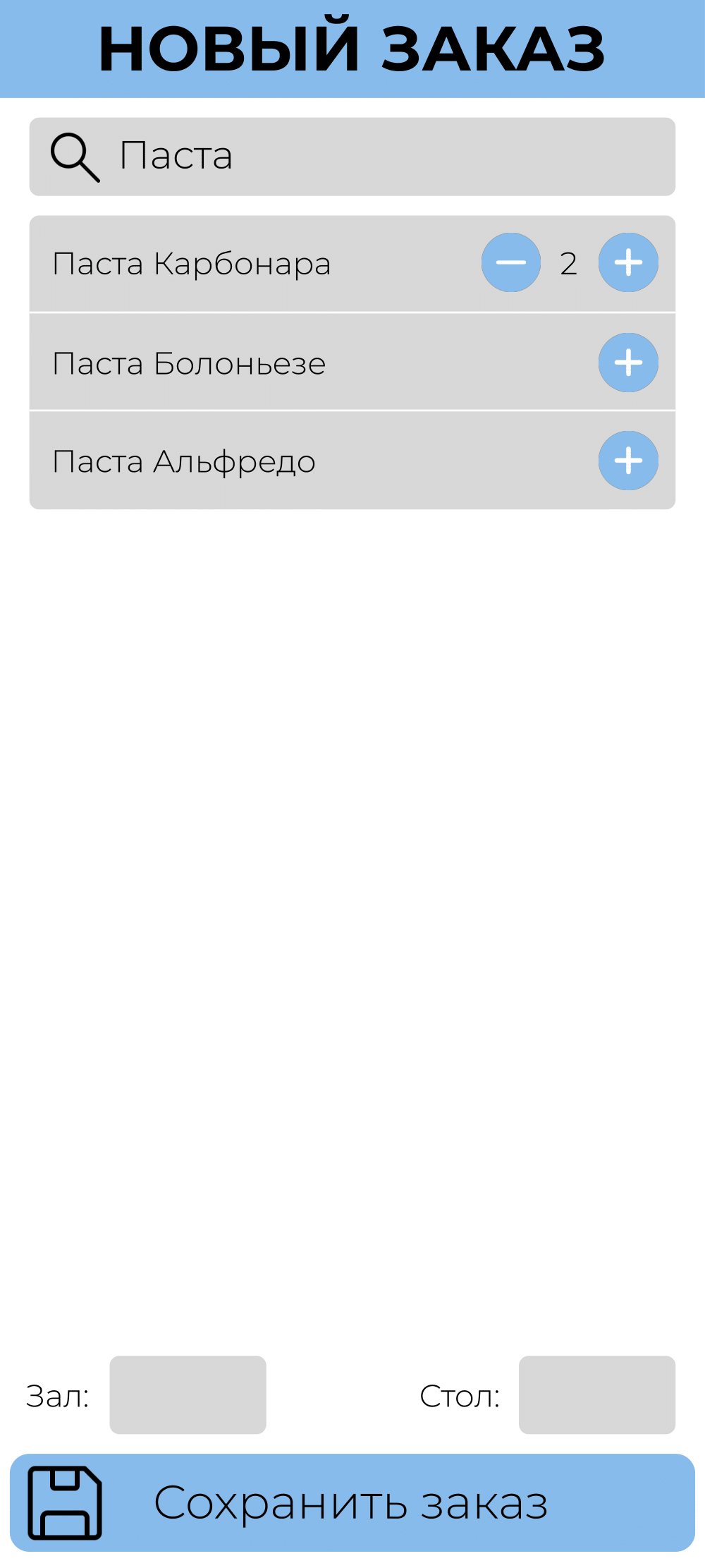
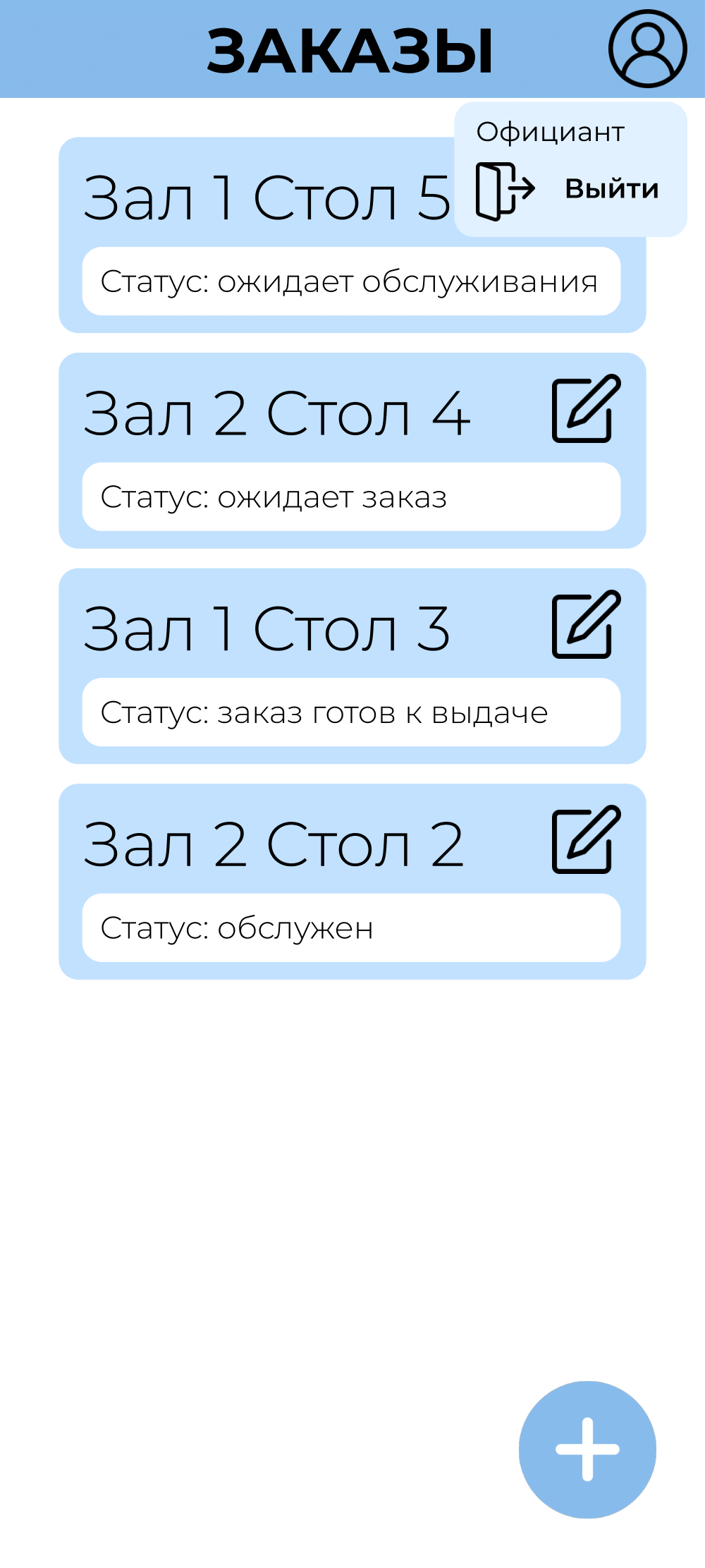
Ниже представлена таблица сравнения описанных выше фреймворков.

Таблица (номер). Сравнение фреймворков.

|  |  |  |
| --- | --- | --- |
| **Фреймворк** | **Плюсы** | **Минусы** |
| **React** | * Обширная экосистема * Имеет большое сообщество * Обеспечивает легкую масштабируемость и гибкость * Имеет разумную кривую обучения * Выполнение происходит быстро с помощью Virtual DOM * Один и тот же код может быть использован для веб-приложений и нативных приложений | * По сравнению с JS, React довольно сложен * Необходимы предварительные знания JSX * Чем больше вы выбираете гибкости, тем больше проблем возникнет |
| **Angular** | * Angular обладает многогранной инфраструктурой, которая подходит для крупномасштабных проектов * Хорошо объясненная методология * Исключительный архив сторонних расширений на выбор * Активное и многочисленное сообщество * Внедрение зависимостей для повышения модульности и эффективности кода * Полностью модульные компоненты * Повышенная безопасность и скорость | * Перед использованием Angular важно изучить Typescript |
| **Ionic** | * Снижает стоимость разработки программного обеспечения, поскольку имеет открытый исходный код * Имеет более короткую кривую обучения * Простое обслуживание с использованием встроенных инструментов браузера и средств отладки * Отлично подходит для гибридной веб-разработки * Большое сообщество активных пользователей * Огромная библиотека плагинов для извлечения API-интерфейсов без каких-либо требований к кодированию | * Периодические обновления требуют внесения изменений в существующее приложение для указания последних изменений |
| **Vue** | * Начинающим веб-разработчикам легко освоить Vue, поскольку он имеет простое кодирование * Хорошая документация * Улучшена производительность рендеринга благодаря Virtual DOM * Может быть объединен с Typescript и JSX для получения наилучшего результата | * Гибкость его конструкции иногда приводит к проблемам * Требует значительного преобразования в устаревшем коде из Angular 1.5 * Имеет небольшую команду поддержки |
| **Polymer** | * Полная поддержка стека веб-приложений, включающая маршрутизацию, адаптивные макеты и макеты уровней данных * Огромная база компонентов, которые работают в разных браузерах * Простой для понимания API * Встроенные в браузер средства построения подтверждают отсутствие требований к отдельным средствам отладки * Отличная документация | * Нет авторизованной IDE * Отсутствие рендеринга на стороне сервера, что приводит к низкой эффективности SEO, увеличению количества HTTP-запросов и большому времени перезагрузки |
| **PWA Builder** | * Комплексный и интуитивно понятный процесс создания * Обширная документация * Поддерживается преданной командой Microsoft | * Автоматизированная процедура, которая может усложнить работу |
| **Svelte** | * Svelte относительно прост в освоении благодаря простому и интуитивно понятному синтаксису * Использует компонентную структуру, способствующую модульности кода и возможности повторного использования | * Отладка может быть немного сложнее по сравнению с другими фреймворками * В то время как Svelte предлагает необходимые инструменты для разработки, экосистема инструментов может быть не такой надежной, как у других фреймворков |

Исходя из возможностей выбранных для анализа фреймворков, а также из анализа баз данных и выбора PostgreSQL, для реализации нашего PWA будет использоваться Vue. Взаимодействие между выбранными СУБД и фреймворком будет происходить через API сервер, который можно создать с помощью Node.js или Exspress.js.

**Создание макетов интерфейса приложения**



Остальное в фигме:

https://www.figma.com/file/AxytF0sIKN8SV69TN77Mpe/СПО-практика-5?type=design&node-id=0-1&mode=design&t=ZxF39XHNSiOXLzXE-0

**Отладка ПО**

Под отладкой понимается процесс, позволяющий получить программу, функционирующую с требующимися характеристиками в заданной области изменения входных данных.

Процесс отладки включает:

* действия, направленные на выявление ошибок (тестирование);
* диагностику и локализацию ошибок (определение характера ошибок и их местонахождение);
* внесение исправлений в программу с целью устранения ошибок.

После того, как программные модули были успешно оттранслированы, размещены по конкретным адресам и связаны между собой, для отладки программы можно воспользоваться любым из следующих методов:

* внутрисхемным эмулятором
* встроенным программным отладчиком
* внешним программным отладчиком
* отлаживаемым устройством с записанным в память программ двоичным кодом программы

**Внутрисхемный эмулятор** с отображением переменных языка программирования на дисплее компьютера оказывает значительную помощь при отладке программ непосредственно на разрабатываемой аппаратуре. Этот метод отладки предоставляет наиболее удобную среду, когда можно непосредственно в отлаживаемом устройстве останавливать программу, контролировать выполнение программы, состояние внешних портов и внутренних переменных, как входящих в состав микросхемы, так и объявленных при написании исходного текста программы.

**Встроенный программный отладчик**, входящий в состав интегрированной среды программирования, не позволяет проконтролировать работу аппаратуры, подключенной к внешним ножкам микроконтроллера, но значительно удешевляет отладочный комплекс, необходимый для написания программ для микроконтроллеров. Сигналы, которые должна подавать на микроконтроллер аппаратура задаются самим программистом. Ручной ввод этих сигналов значительно замедляет процесс отладки программного обеспечения и повышает риск возникновения ошибки.

**Внешний программный отладчик**. В некоторых случаях используется не интегрированная среда программирования, а отдельный транслятор с выбранного языка программирования. В этом случае можно воспользоваться любой программой, эмулирующей выбранный тип микроконтроллера. Так как объектные форматы различных трансляторов несколько отличаются друг от друга, то в качестве входного файла используется загрузочный модуль. В этих форматах отладочная информация полностью отсутствует, поэтому отладку в таких программах можно вести только с помощью встроенного дизассемблера и распечатанного (или открытого в другом окне) листинга программы.

**Программирование микросхемы**. При использовании любого способа отладки программы, готовый загрузочный модуль записывается во внутреннюю память программ микроконтроллера при помощи программатора. После этого ведется тщательное тестирование разработанного устройства с целью обнаружения ошибок в схеме и программе этого устройства. Только после успешного прохождения этого тестирования программа считается полностью написанной и отлаженной.

Для отладки программ обычно применяют три способа:

1. пошаговая отладка программ с заходом в подпрограммы;
2. пошаговая отладка программ с выполнением подпрограммы как одного оператора;
3. выполнение программы до точки останова.

**Принципы и методы тестирования ПО**

Классификация **по целям**:

* Функциональное тестирование (functional testing) рассматривает заранее указанное поведение и основывается на анализе спецификации компонента или системы в целом.
* Нефункциональное тестирование (non-functional testing) — тестирование атрибутов компонента или системы, не относящихся к функциональности.
* Тестирование пользовательского интерфейса (GUI Testing) — проверка интерфейса на соответствие требованиям.
* Тестирование удобства использования (Usability Testing) — определение степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.
* Тестирование безопасности (security testing) — проверка безопасности системы, а также анализ рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.
* Инсталляционное тестирование (installation testing) – проверка успешной установки и настройки, а также обновления или удаления приложения.
* Конфигурационное тестирование (Configuration Testing) — проверка работы ПО при различных конфигурациях системы.
* Тестирование на отказ и восстановление (Failover and Recovery Testing) – проверка обеспечения сохранности и целостности данных, после возможных сбоев, возникших в связи с ошибками ПО, отказами оборудования или проблемами связи.
* Тестирование локализации (localization testing) — проверка адаптации ПО для определенной аудитории в соответствии с ее культурными особенностями.
* Тестирование производительности (performance testing) — определение стабильности и потребления ресурсов в условиях различных сценариев использования и нагрузок.
* Нагрузочное тестирование (load testing) — определение или сбор показателей производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе.
* Тестирование стабильности или надежности (Stability / Reliability Testing) — проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.
* Стрессовое тестирование (Stress Testing) – проверка приложения и системы на работоспособность в условиях стресса (например, повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера) и также оценка способности системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса.
* Объемное тестирование (Volume Testing) — тестирование, которое проводится для получения оценки производительности при увеличении объемов данных в базе данных приложения.
* Тестирование масштабируемости (scalability testing) — тестирование, которое измеряет производительность сети или системы, когда количество пользовательских запросов увеличивается или уменьшается.

Классификация **по уровню тестирования**:

* Модульное (компонентное) тестирование (Unit Testing) проводится самими разработчиками, т.к. предполагает полный доступ к коду, для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде, проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.).
* Интеграционное тестирование (Integration Testing) направлено на проверку корректности взаимодействия нескольких модулей, объединенных в единое целое, т.е. проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.
* Системное тестирование (System Testing) — это проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты и оцениваются характеристики качества системы — ее устойчивость, надежность, безопасность и производительность.
* Операционное тестирование (Release Testing). Даже если система удовлетворяет всем требованиям, важно убедиться в том, что она удовлетворяет нуждам пользователя и выполняет свою роль в среде своей эксплуатации. Поэтому так важно провести операционное тестирование как финальный шаг валидации. Кроме этого, тестирование в среде эксплуатации позволяет выявить и нефункциональные проблемы, такие как: конфликт с другими системами, смежными в области бизнеса или в программных и электронных окружениях и др. Очевидно, что нахождение подобных вещей на стадии внедрения — критичная и дорогостоящая проблема.

Классификация **по исполнению кода**:

* Статическое тестирование — процесс тестирования, который проводится для верификации практически любого артефакта разработки. Целью анализа является раннее выявление ошибок и потенциальных проблем в продукте.
* Динамическое тестирование проводится на работающей системе.

Классификация **по хронологии выполнения**:

* Повторное/подтверждающее тестирование (re-testing/confirmation testing) — тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления этих ошибок.
* Регрессионное тестирование (regression testing) — это тестирование после внесения изменений в код приложения, для подтверждения того факта, что эти изменения не внесли ошибки в областях, которые не подверглись изменениям.
* Приёмочное тестирование проверяет соответствие системы потребностям, требованиям и бизнес-процессам пользователя.

Традиционно тестирование программного обеспечения можно разделить на различные категории по разным параметрам, которые были рассмотрены выше. Однако наиболее популярной классификацией является разделение на ручное и автоматизированное тестирование.

Ручное тестирование — это вид тестирования программного обеспечения, при котором тесты выполняются тестировщиком вручную, без использования каких-либо средств автоматизации и полностью зависит от тестировщика.

Когда используется ручное тестирование:

1. продукт находится на начальном этапе разработки;
2. продукт краткосрочный и небольшой;
3. при тестировании удобства использования продукта;
4. при интуитивном или исследовательском тестировании;
5. при работе с физическими продуктами.

Автоматизированное тестирование — это метод тестирования программного обеспечения, который предполагает использование инструментов и фреймворков автоматизации для выполнения одного и того же набора тест-кейсов снова и снова.

Когда используется автоматизированное тестирование:

1. при выполнении повторяющихся тестов;
2. при использовании тестирования производительности или при нагрузочном тестировании;
3. когда имеется большое количество тест-кейсов;
4. когда необходимо исключить человеческий фактор;
5. при работе с большими объемами данных.

**Тест-кейсы**

# Тест Кейс № 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест авторизации работника | | |
| **Функция:** | Вход в рабочую область | | |
| **Действие** | | **Ожидаемый результат** | **Результат теста:**   * пройден * провален * заблокирован |
| Предусловие: | |  | |
| Откройте приложение | | Приложении открыто и доступно |  |
|  | |  |  |
| Шаги теста: | |  | |
| Заполните поля для ввода данных:  "Логин": username  "Пароль": password | | Введены валидные данные |  |
| Нажмите кнопку "Войти в систему" | | Переход на страницу с рабочей областью |  |
|  | |  |  |
| Постусловие: | |  | |
| Нажмите на стрелочку назад для перехода на страницу в хода | | Страница "Вход в систему" открыта |  |
|  | |  |  |

# Тест Кейс № 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест регистрация пользователя | | |
| **Функция:** | Указание ролей работников, открытие доступа к функционалу | | |
| **Действие** | | **Ожидаемый результат** | **Результат теста:**   * пройден * провален * заблокирован |
| Предусловие: | |  | |
| Откройте приложение | | Приложении открыто и доступно |  |
| У регистрируемого работника еще нет аккаунта в информационной системе | | В БД нет данных о работнике |  |
| Выполнен вход в аккаунт администратора | | Открыт доступ функционала администратора |  |
|  | |  |  |
| Шаги теста: | |  | |
| Нажмите на кнопку «Аккаунты сотрудников» | | Страница аккаунты сотрудников открыта и доступна |  |
| Кликните на иконку «плюс» в нижнем левом углу экрана | | Выполнен переход на страницу «Создание аккаунта сотрудника», страница открыта и доступна |  |
| Заполните поля для ввода данных:  "ФИО сотрудника": Фамилия Имя Отчество  "Должность (роль в системе)": Официант | | Введены валидные данные |  |
| Проверить данные, генерация которых происходит автоматически:  «Логин» (генерация по ФИО сотрудника), «Пароль». | | Данные сгенерированы корректно |  |
| Нажмите кнопку "Сохранить аккаунт" | | Данные сохранены в БД. Переход на страницу «Аккаунт сотрудника», страница открыта и доступна. |  |
|  | |  |  |
| Постусловие: | |  | |
| Нажмите на кнопку «Изменить аккаунт» | | Выполнен переход на страницу «Создание аккаунта сотрудника», страница открыта и доступна. |  |
| Нажмите на кнопку «Удалить аккаунт» | | Отправлен запрос на подтверждения удаления с кнопками «УДАЛИТЬ» и «ОТМЕНА» |  |
| Кликните на иконку стрелочки назад | | Выполнен переход на страницу «Аккаунты сотрудников», страница открыта и доступна. Корректно отображается созданный аккаунт |  |
|  | |  |  |

# Тест Кейс № 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест добавление в информационную систему технологической карты | | |
| **Функция:** | Сохранение данных о блюдах | | |
| **Действие** | | **Ожидаемый результат** | **Результат теста:**   * пройден * провален * заблокирован |
| Предусловие: | |  | |
| Откройте приложение | | Приложении открыто и доступно |  |
| Разработана технологическая карта шеф-поваром | | Разработана документация и отправлен документ техническому специалисту |  |
| Выполнен вход в аккаунт администратора | | Открытый доступ функционала администратора |  |
|  | |  |  |
| Шаги теста: | |  | |
| Нажмите на кнопку «Технологические карты» | | Страница «Технологические карты» открыта и доступна |  |
| Кликните на иконку «плюс» в нижнем левом углу экрана | | Выполнен переход на страницу «Создание технологической карты», страница открыта и доступна |  |
| Заполните поля для ввода данных:  "Название блюда": Пицца Маргарита  "Состав блюда":  «Наименование ингредиента»: Сыр Моцарелла твердый  «Расход Брутто, г»: 95  «Расход Нетто, г»: 91  "Технологический процесс": *Тесто раскатать до диаметра 28-30 см. Раскатанное тесто полить соусом для пиццы, равномерно*  *распределить. Посыпать тертым или нарезанным на слайсере сыром, посыпать специями*  *"Средиземноморские травы" (или смесью специй для пиццы). Сверху положить нарезанные*  *кружочками помидоры. Переместить лопатой в пицца - печь. Выпекать при температуре 300 С до*  *готовности (2,5 - 3 минуты).*  "Требования к оформлению и подаче": *Готовую пиццу посыпать листиками зеленого базилика.* | | Введены валидные данные |  |
| Проверить данные, подсчет которых происходит автоматически:  «Масса порции». | | Данные рассчитаны корректно |  |
| Добавление фото блюда. Кликните на кнопку «Добавить фото готового блюда». Загрузить фото | | Фото загружено и отображается корректно. |  |
| Нажмите кнопку "Сохранить блюдо" | | Данные сохранены в БД. Переход на страницу «Технологические карты», страница открыта и доступна. Отображение созданного блюда корректно. |  |
|  | |  |  |
| Постусловие: | |  | |
| Нажмите на иконку «Изменить блюдо» | | Выполнен переход на страницу «Создание технологической карты», страница открыта и доступна. |  |
| Нажмите на иконку крестика «Удалить блюдо» | | Отправлен запрос на подтверждения удаления с кнопками «УДАЛИТЬ» и «ОТМЕНА» |  |
| Кликните на блюдо | | Выполнен переход на страницу «Технологическая карта блюда», страница открыта и доступна. Корректно отображаются данные о блюде |  |
|  | |  |  |

1. POS или Point Of Sale — система для обработки транзакций [↑](#footnote-ref-1)