|  |
| --- |
| **Автономное учреждение  профессионального образования**  **Ханты-Мансийского автономного округа – Югры**  **«СУРГУТСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»**  **(АУ «Сургутский политехнический колледж»)** |
|  |
| СТРУКТУРНОЕ ПОДРАЗДЕЛЕНИЕ - 4  (Энергетическое отделение) |

Самостоятельная работа номер 4

**Климентов ИВ**

по теме: "Описание и анализ требований, выбор решений

для разработки приложений"

Цель: Укрепить знания о классификации требований, методах их анализа, а также навыки выбора и применения технологических решений для разработки приложений.

Часть 1: Теория и анализ требований

1. Различие между функциональными и нефункциональными требованиями

Функциональные требования определяют, что система должна делать, то есть описывают функции и возможности, которые пользователь ожидает от программного обеспечения. Эти требования обычно включают спецификации действий, которые система должна выполнять.

Примеры функциональных требований:

· Пользователь должен иметь возможность создать новый аккаунт с указанием имени, электронной почты и пароля.

· Пользователь должен иметь возможность искать свободные столики в ресторанах по дате и времени.

Нефункциональные требования определяют, как система должна выполнять свои функции, включая характеристики, такие как производительность, безопасность, надежность и доступность. Эти требования обеспечивают определенные качества системы.

Примеры нефункциональных требований:

· Система должна загружать страницы не более чем за 2 секунды.

· Приложение должно обеспечивать безопасность данных пользователя в соответствии с стандартами GDPR (General Data Protection Regulation).

2. Важность правильного оформления требований для успешной разработки проекта

Правильное оформление требований критически важно для успешной разработки проекта по следующим причинам:

· Четкость и понимание: Правильно сформулированные требования помогают всей команде (разработчикам, дизайнерам, тестировщикам и менеджерам) понять, что именно необходимо реализовать. Это минимизирует вероятность недопонимания и ошибок, которые могут возникнуть из-за разночтений.

· Планирование и оценка: Корректное оформление требований позволяет точно оценить объем работы и необходимые ресурсы для проекта, что помогает избежать задержек и перерасхода бюджета.

· Контроль за выполнением: Оформленные требования служат основой для проверки и оценки результатов. Команда может сверять реализацию функций с требованиями, что позволяет выявлять и корректировать отклонения на ранних этапах.

· Адаптация к изменениям: В проекте могут возникнуть изменения, и четкие требования позволят гибко реагировать на изменения, адаптировать проект без потерь в качестве.

3. Бизнес-требования для приложения по бронированию столиков в ресторанах

· Приложение должно поддерживать возможность бронирования столиков в ресторанах через мобильное устройство с минимальным временем ожидания в 5 секунд.

· Пользователь должен иметь возможность отменить бронирование столика за 2 часа до назначенного времени без штрафных санкций.

4. User Story для мобильного приложения по бронированию отелей

User Story:

· Кто: Как пользователь приложения для бронирования отелей

· Что делает: я хочу найти и забронировать номер в отеле на нужные даты,

· Зачем это нужно: чтобы обеспечить комфортное проживание во время моей поездки.

Часть 2: Практическая работа с требованиями

Ситуация: Необходимо разработать мобильное приложение для управления расписанием (планировщик задач).

1. Функциональные требования

Для разработки мобильного приложения планировщика задач необходимо учесть следующие функциональные требования:

· Создание задач: Пользователь должен иметь возможность создавать новые задачи с указанием названия, описания, даты и времени выполнения.

· Редактирование задач: Пользователь должен иметь возможность редактировать существующие задачи, изменяя любые поля, включая название, описание, дату и время.

· Удаление задач: Пользователь должен иметь возможность удалять задачи из своего расписания.

· Уведомления: Приложение должно отправлять пользователю push-уведомления о предстоящих задачах за 10 минут до их начала.

· Категоризация задач: Пользователь должен иметь возможность группировать задачи по категориям (например, работа, личные дела) и присваивать цветовые метки каждой категории.

2. Нефункциональные требования

При разработке приложения планировщика задач необходимо учесть следующие нефункциональные требования:

· Время отклика: Время отклика приложения на действия пользователя не должно превышать 1 секунды в 95% случаев. Это обеспечит приятный пользовательский интерфейс и улучшит общий пользовательский опыт.

· Поддержка мобильных платформ: Приложение должно поддерживать iOS и Android, начиная с версий не ниже iOS 12 и Android 8.0. Это обеспечит доступность приложения для широкого круга пользователей.

· Безопасность данных: Приложение должно использовать шифрование данных и безопасное хранение информации о пользователях согласно стандартам GDPR, чтобы обеспечить защиту личной информации пользователей.

3. Бизнес-требования

Для увеличения эффективности и успеха приложения планировщика задач можно выделить следующие бизнес-требования:

· Увеличение числа активных пользователей: Приложение должно достичь увеличения числа активных пользователей на 30% в течение первого квартала после запуска.

· Рост пользовательской базы: Приложение должно обеспечить привлечение не менее 5000 новых пользователей в течение первых трех месяцев после его публикации.

Часть 3: Приоритизация требований и диаграммы

1. Приоритизация требований с использованием метода MOSCOW

Метод MOSCOW позволяет расставить приоритеты для требований по следующим категориям:

Must: абсолютно необходимые требования для минимально жизнеспособного продукта. (должны быть выполнены)

Should: важные, но не критические требования (желательно выполнить)

Could: желательные требования, которые можно было бы выполнить, но которые не являются критичными (могут быть выполнены)

Won't: требования, которые в настоящее время не будут реализованы (но могут быть выполнены в будущем)

Для мобильного приложения по доставке еды расставляем приоритеты следующим образом:

Приоритеты (MOSCOW):

Must:

Просмотр меню и выбор блюд.

Возможность оплаты через интернет.

Should:

Отслеживание доставки.

Поддержка кредитных карт и PayPal.

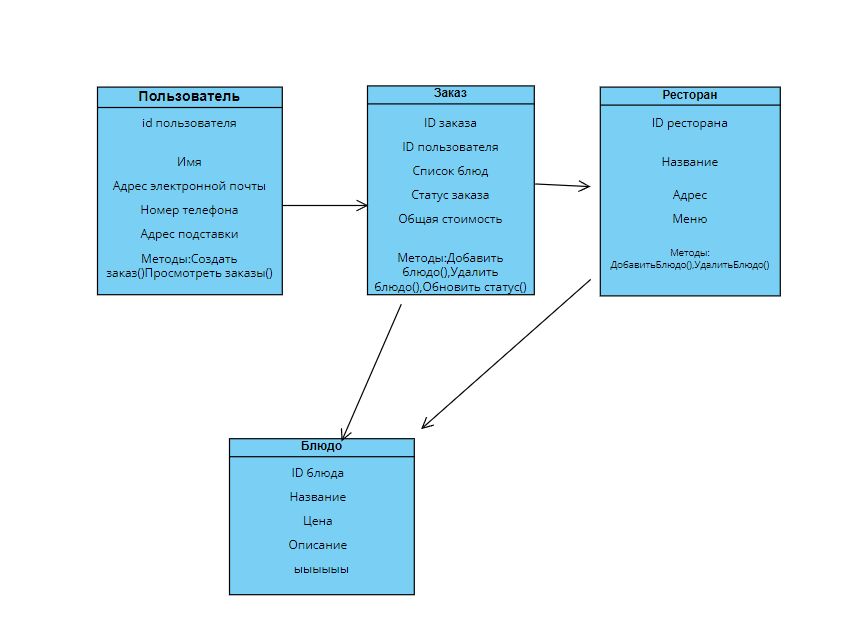
Could:

Уведомления о статусе заказа.

Won't:

Дополнительные функции, такие как интеграция с социальными сетями для обмена блюдами.

2. Создание диаграммы классов для приложения по доставк



Часть 4: Анализ рисков и выбор технологического стека

1. Анализ рисков

При создании приложения для управления расписанием команда разработки может столкнуться с несколькими рисками, которые могут повлиять на успешность проекта. Ниже приведены главные возможные риски и их описание:

Технические риски:

Недостаточная совместимость технологий – использование технологий, которые не интегрируются между собой, может привести к проблемам на этапе разработки и поддержки приложения.

Выбор устаревших библиотек или фреймворков – использование устаревших технологий может замедлить разработку и усложнить поддержку приложения в будущем.

Риски, связанные с требованиями:

Неясные или изменяющиеся требования – если бизнес-требования не будут четко определены, это может привести к путанице и дополнительным временным затратам на исправление после начала разработки.

Избыточные или недостаточные требования – слишком много функций могут перегрузить проект, а недостаток функций может сделать приложение неприемлемым для пользователей.

Риски, связанные с пользователем:

Непонимание конечными пользователями – если приложение будет сложно использовать, это приведет к недовольству пользователей и низкой оценке продукта.

Неудовлетворительное поведение пользователей – если приложение не решает основные проблемы пользователей, оно может не найти своего места на рынке.

Риски, связанные с управлением проектом:

Неэффективное управление проектом – отсутствие четкой структуры и методов управления проектом может привести к задержкам и перерасходу бюджета.

Проблемы в команде – конфликты в команде или недостаточная коммуникация могут снизить мотивацию и эффективность работы команды.

2. Выбор технологического стека

При разработке веб-приложения для управления задачами выбор технологического стека играет критическую роль в обеспечении успешного выполнения проекта. Вот пример рекомендованного технологического стека:

Фронтенд:

HTML5, CSS3, JavaScript – базовые технологии для вёрстки и взаимодействия с пользователем.

React или Vue.js – современные фреймворки для создания динамического пользовательского интерфейса. React предпочтителен благодаря своей популярности и мощной экосистеме.

Бэкенд:

Node.js с Express.js – JavaScript обеспечит возможность использовать один язык между фронтендом и бэкендом, что ускорит разработку.

Python с Django или Flask – альтернатива для более сложных процессов, при этом Python имеет более простой синтаксис и богатую библиотеку для работы с данными.

База данных:

PostgreSQL или MongoDB – в зависимости от структуры данных. PostgreSQL подходит для сложных запросов и реляционной структуры, тогда как MongoDB лучше для гибких и динамических данных.

Разработка API:

RESTful API или GraphQL – для обеспечения взаимодействия между фронтендом и бэкендом приложения.

Развертывание и хостинг:

AWS, Heroku или DigitalOcean – для развертывания приложения в облаке с возможностью масштабирования.

**Вывод :**В ходе работы было исследовано форматирование сообщений, выделены основные принципы и правила. Создана таблица с элементами сообщения и их оформлением, приведены примеры правильного и неправильного форматирования с анализом ошибок. Рекомендации по улучшению коммуникации включают внедрение стандартов и шаблонов оформления. Результаты исследования подтверждают важность структурированного подхода для повышения эффективности общения в организации.