# Самостоятельная работа

# 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.

## Задания:

**Часть 1: Теория и анализ требований**

1. Прочитайте лекционный материал и выполните следующие задания:

1. - **Функциональные требования**: Эти требования описывают **что должна делать система**. Это поведение, функции и операции, которые должна выполнять система. Пример:
   1. Приложение для интернет-банка должно позволять пользователю проверять баланс счета.
   2. Пользователь должен иметь возможность перевести средства на другой счет.

**Нефункциональные требования**: Они описывают **как** система должна работать, не касаясь её функциональности. Это требования к производительности, безопасности, надёжности и удобству использования системы. Пример:

* 1. Система должна обеспечивать безопасность данных клиентов через шифрование.
  2. Время отклика системы должно быть не более 2 секунд для 90% запросов.

1. - Четкое оформление требований помогает команде избежать ошибок и недопонимания в процессе разработки. Правильно описанные требования служат "дорожной картой" для всей команды.
2. Увеличение числа регистраций в системе на 15% в течение первых шести месяцев после запуска.

Система должна поддерживать оплату через кредитные карты и популярные платёжные сервисы.

1. Как пользователь, я хочу забронировать отель и оформить бронь через мобильное приложение, чтобы быстро забронировать отель на время отдыха.

**Часть 2: Практическая работа с требованиями**   
Ситуация: Необходимо разработать мобильное приложение для управления расписанием (например, планировщик задач).

Функциональные требования для приложения планировщика задач:

1. Создание задач:

- Пользователь должен иметь возможность создавать новые задачи с указанием названия, описания и срока выполнения.

1. Редактирование задач:

- Пользователь должен иметь возможность редактировать существующие задачи, изменяя детали, такие как название, описание и срок.

1. Сортировка и фильтрация:

- Приложение должно предоставлять возможность сортировки задач по дате, приоритету и статусу. Также должна быть возможность фильтрации задач по категориям.

1. Уведомления:

- Пользователь должен получать уведомления о приближающихся сроках выполнения задач через push-уведомления или e-mail.

1. Пользовательская аналитика:

- Приложение должно предоставлять пользователю аналитику по выполненным задачам, включая статистику по времени, затраченному на выполнение задач, и выполнению задач за определённый период.

Нефункциональные требования для приложения планировщика задач:

1. Производительность:

- Приложение должно обеспечивать время отклика не более 2 секунд при загрузке интерфейса и выполнении операций, таких как создание, редактирование или удаление задач. Это необходимо для обеспечения высокой скорости взаимодействия с пользователем.

1. Надежность:

- Приложение должно обеспечивать доступность 99.9% времени, что подразумевает минимальные отключения и сбои. В случае неполадок система должна предоставлять пользователю соответствующее уведомление и возможность восстановления данных.

1. Поддержка мобильных платформ:

- Приложение должно быть доступно как для мобильных устройств (iOS, Android), так и для десктопных платформ (Windows, macOS). Это обеспечит пользователям гибкость в доступе к своим задачам с различных устройств и повысит общий уровень удобства использования.

Бизнес-требования для приложения планировщика задач:

1. Улучшить интерфейс, сделать более удобную навигацию:

- Обновлённый интерфейс заинтересует новых юзеров.

1. внедрение функций напоминаний и персонализированных рекомендаций

- Для увеличение времени удержания пользователей и

для улучшения взаимодействия и вовлеченности пользователей.

**Часть 3: Приоритизация требований и диаграммы**

Приоритизация требований по методу MOSCOW:

- Must have (Обязательно должно быть):

- Просмотр меню и выбор блюд.

- Возможность оплаты через интернет.

- Should have (Должно быть, но можно без этого на старте):

- Отслеживание доставки.

- Could have (Желательно, но необязательно):

- Поддержка кредитных карт и PayPal.

- Won't have (Не будет реализовано на данном этапе):

- Уведомления о статусе заказа.

Задание 2

Вот пример диаграммы классов для приложения по доставке еды:

|  |  |  |  |
| --- | --- | --- | --- |
| Класс: Пользователь | Класс: Заказ | Класс: Ресторан | Класс: Блюдо |
| - Атрибуты:  - userID:int  - имя:string  - email:string  - телефон: string  - Методы:  - зарегистрироваться()  - войти()  - оформитьЗаказ() | - Атрибуты:  - orderID: int  - date: DateTime  - статус: string  - пользователь: Пользователь  - ресторан: Ресторан  - блюда: List<Блюдо>  - Методы:  - обновитьСтатус()  - добавитьБлюдо() | - Атрибуты:  - restaurantID: int  - имя: string  - адрес: string  - меню: List<Блюдо>  - Методы:  - добавитьБлюдо()  - удалитьБлюдо() | - Атрибуты:  - dishID: int  - название: string  - цена: decimal  - описание: string  - Методы:  - изменитьЦену() |

**Часть 4: Анализ рисков и выбор технологического стека:**

При разработке приложения для управления расписанием команда может столкнуться с следующими рисками:

1. Технические риски:

- Сложность интеграции с существующими системами (календарями, почтовыми сервисами).

- Выбор неподходящих технологий, что может увеличить время разработки и затраты.

1. Риски планирования:

- Слишком оптимистичные оценки сроков выполнения задач, что может привести к задержкам.

- Неполное понимание требований, что может вызвать переработки.

1. Командные риски:

- Нехватка необходимых навыков у членов команды, что может затруднить работу.

- Высокая текучесть кадров, что влияет на стабильность проекта.

1. Финансовые риски:

- Превышение бюджета из-за непредвиденных затрат или дополнительных функций.

1. Пользовательские риски:

- Низкий уровень принятия пользователями из-за сложности интерфейса или недостаточной функциональности.

Задание 2

Для разработки веб-приложения для управления задачами можно выбрать следующий технологический стек:

1. Фронтенд:

- HTML/CSS: Базовые технологии для разметки и стилизации страниц.

- JavaScript: Язык программирования для интерактивности.

- React или Vue.js: Популярные библиотеки/фреймворки для создания пользовательских интерфейсов, обеспечивающие быструю разработку и гибкость.

2. Бэкенд: - Node.js + Express: Позволяет легко создавать серверные приложения на JavaScript.

- Python + Django или Flask: Альтернатива для более сложной логики и работы с базами данных.

3. База данных: - MongoDB: NoSQL база данных, хорошо подходящая для хранения гибких данных.

- PostgreSQL: Реляционная база данных с расширенными возможностями.

4. Аутентификация:

- JWT (JSON Web Tokens): Для безопасной аутентификации пользователей.

5. Развертывание и хостинг:

- Docker: Контейнеризация приложения для упрощения развертывания.

- AWS или Heroku: Облачные платформы для хостинга.

6. Инструменты разработки:

- Git: Для контроля версий.

- Webpack или Parcel: Для сборки фронтенд-кода.