**I. ФУНКЦИОНАЛЬНЫЕ И НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ**

1. **Функциональные требования** описывают конкретные функции и возможности. Указываются в User Stories, Use Cases, техническом задании. **Примеры:**
2. "Пользователь может авторизоваться по логину и паролю";
3. "Система должна сохранять заказ в базу данных";
4. "Приложение должно отправлять email-уведомления";
5. **Нефункциональные требования** – это характеристики и ограничения, которые описывают, как система выполняет свои функции, а не что она должна делать. Также они задают критерии качества, которым должен соответствовать продукт. **Критерии и примеры:**
6. производительность – "Система должна обрабатывать 1000 RPS";
7. масштабируемость – "Возможность горизонтального масштабирования под нагрузкой";
8. надежность – "Доступность 99.9% (uptime)";
9. безопасность – "Данные передаются только по HTTPS";
10. совместимость – "Поддержка браузеров Chrome, Firefox, Safari".

**II. СОСТАВ АРХИТЕКТУРНОЙ ДОКУМЕНТАЦИИ**

**Архитектурная документация** включает в себя описание структуры и компонентов системы, их взаимодействия и принципы проектирования. Она может содержать диаграммы архитектуры, описания интерфейсов, технологические стеки, а также тестовые подходы и стратегию развертывания. Архитектурная документация помогает всем участникам проекта понять, как система будет организована и как будут реализованы требования.

**III. REST И МИКРОСЕРВИСНАЯ АРХИТЕКТУРА**

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети. Он определяет как компоненты системы (клиент и сервер) обмениваются данными.

Микросервисная архитектура – подход к разработке ПО, при котором приложение разбивается на множество небольших, независимых модулей – микросервисов. Каждый из них выполняет свою уникальную задачу и работает автономно.

**IV. ОСНОВЫ DEVOPS И CI/CD**

**DevOps и CI/CD** – методологии непрерывной разработки ПО, которые объединяют разработчиков и специалистов по IT-обслуживанию для автоматизации процессов сборки, настройки и развёртывания ПО.

**Основы и принципы:**

1. непрерывная интеграция (CI) (автоматическое объединение изменений в коде, проходящих через репозиторий);
2. непрерывная доставка (CD) (автоматическая сборка, тестирование и развёртывание новых версий продукта);
3. автоматизация процессов (сокращение ручного труда за счёт автоматизированных систем и инструментов);
4. мониторинг и обратная связь (постоянный контроль за работой приложений и инфраструктуры);
5. культура сотрудничества (устранение барьеров между командами разработки и эксплуатации).

**V. КОНТЕЙНЕРЫ DOCKER И ИХ ПРЕДНАЗНАЧЕНИЕ**

**Контейнеры Docker** – это изолированные среды, в которых запускаются приложения со всеми зависимостями. Они нужны, чтобы упростить разработку, тестирование и развёртывание приложений, а также для

* утилизации (эффективного использования) ресурсов (на одном сервере можно запустить несколько контейнеров);
* изоляции приложений;
* установки ОС (через использование хостовой ОС);
* настройки окружения для приложения (можно один раз настроить окружение и быстро клонировать его между контейнерами).

**VI. ОСОБЕННОСТИ РОЛИ «ТИМЛИД»**

1. **Организация работы:**

Тимлид планирует и синхронизирует работу команды, распределяя задачи между ее членами, контролирует процессы и результат, а также общается с руководством/заказчиками для уточнения требований и обсуждения сроков и изменений проекта.

1. **Декомпозиция задач:**

Так как тимлид должен давать задачи сотрудникам, он должен уметь декомпозировать крупные задачи, чтобы их можно было распределить между несколькими членами команды. Он также контролирует процесс декомпозиции и проверяет корректность описаний задач для разработки.

1. **Контроль сроков:**

Тимлид контролирует сроки через специальные инструменты для того, чтобы обеспечить успешную реализацию проекта и достижение поставленных целей. Благодаря корректно поставленным срокам команда понимает, сколько времени она может выполнять те или иные задачи.