**Вопрос 3. Закон Конвея. Потоки ценности. Стейкхолдеры и RACI. Что это и зачем информация о них нужна архитектору решений?**

**Закон Конвея** говорит, что структура системы повторяет коммуникационную структуру организации.  
**Потоки ценности** помогают видеть, как создаётся и доставляется ценность клиенту, чтобы оптимизировать процесс.  
**Стейкхолдеры** - все, кто влияет на проект или заинтересован в нём; их потребности нужно учитывать.  
**RACI** даёт чёткое распределение ролей и ответственности, чтобы исключить путаницу и задержки.

Все это нужно для того, чтобы архитектор мог:

* **Закон Конвея**: согласовать структуру решения с реальной организацией и коммуникациями.
* **Потоки ценности**: проектировать систему с учётом полного пути создания ценности для клиента.
* **Стейкхолдеры**: учитывать требования и ограничения всех заинтересованных сторон.
* **RACI**: понимать, кто что делает и кто за что отвечает, чтобы избегать конфликтов и задержек.

**Вопрос 5. НФТ: какими бывают? Как узнать все качественные атрибуты? Какие из них значимы для архитектуры, а какие нет? Приведите примеры НФТ значимых и незначимых архитектурно.**

**1. Виды НФТ**

* **Производительность (Performance)**: время отклика, пропускная способность.
* **Надёжность (Reliability)**: частота отказов, время восстановления.
* **Масштабируемость (Scalability)**: возможность наращивать ресурсы без потери качества.
* **Безопасность (Security)**: защита данных и доступов.
* **Удобство сопровождения (Maintainability)**: лёгкость внесения изменений, отладки.
* **Удобство использования (Usability)**: простота обучения и работы для пользователя.
* И др. (портативность, совместимость, интероперабельность и т.п.).

**2. Как узнать все качественные атрибуты**

* **Обсудить со стейкхолдерами** (заказчиками, пользователями, менеджментом), чтобы понять их ожидания и приоритеты.
* **Изучить бизнес-контекст** и **регулятивные требования** (например, финтех, медицина).
* **Изучить предыдущие решения** и аналогичные системы (best practices в индустрии).
* Использовать **стандартизированные списки** (например, ISO/IEC 25010).

**3. Значимость для архитектуры**

* **Значимые** для архитектуры - те атрибуты, которые влияют на ключевые решения по структуре системы, выбору технологий, шаблонов проектирования. Например:
  + **Безопасность**: потребует внедрения криптографии и многослойной защиты.
  + **Масштабируемость**: определит выбор микро-сервисной архитектуры, балансировщиков нагрузки и т.д.
  + **Надёжность**: потребует резервирования, распределённого хранения данных.
* **Незначимые** архитектурно - это атрибуты, которые не затрагивают кардинально структуру или технические решения. Например:
  + Удобство использования (UI/UX) часто считается важным бизнес-требованием, но **архитектуру** (в смысле системной структуры) оно влияет косвенно;
  + Требования к дизайну интерфейсов (цветовая гамма, размер шрифтов) - не меняют внутреннюю архитектуру системы, хотя и важны для конечного продукта.

Таким образом, при выявлении НФТ нужно отделять те, что **меняют технический «скелет»** приложения, от тех, что влияют лишь на уровень представления или процессов, не требующих радикального изменения архитектуры.

**Вопрос 6. Определения и различия между архитектурным стилем и паттерном. Зачем они архитектору решений?**

**Архитектурный стиль** - это высокоуровневый подход к организации системы (например, микросервисная архитектура, «клиент-сервер», многослойная архитектура). Он определяет общие принципы построения системы, её структурные элементы и способы взаимодействия.

**Архитектурный паттерн** - это более конкретное решение типичной задачи в архитектуре (например, «сAGA», «Circuit Breaker», «Layered Architecture Pattern»). Паттерн описывает, как связаны между собой конкретные компоненты и какие ограничения/преимущества возникают.

Зачем они нужны **архитектору решений**?

1. **Стиль** задаёт «скелет» системы и формирует общую логику взаимодействия модулей.
2. **Паттерны** помогают решать часто возникающие проблемы в рамках выбранного стиля, опираясь на проверенные решения и практики.

Использование стилей и паттернов упрощает коммуникацию между участниками проекта (у всех есть общий язык и понимание структуры) и снижает риски, ведь типовые задачи решаются по уже многократно проверенным схемам.

**Вопрос 11. Архитектурные виды (architecture views). Зачем нужны, какими бывают, в каком порядке их представляют? Пример**

**Что такое архитектурные виды?**

Это разные «ракурсы» (представления) системы, позволяющие описать её с учётом интересов различных стейкхолдеров (например, заказчиков, разработчиков, администраторов). Каждый вид фокусируется на своей области: структуре, поведении, развертывании и т.д.

**Зачем нужны архитектурные виды?**

1. **Понимание для разных ролей:** бизнес-заказчику важен общий контекст и процессы, а разработчику - детали компонентов.
2. **Управляемая сложность:** разбиваем архитектуру на несколько описаний, чтобы не перегружать единую диаграмму.
3. **Снижение рисков и ясность коммуникации:** каждая команда видит свою часть и взаимосвязи с остальными.

**Основные виды (пример 4+1)**

В классической «4+1» модели описывают:

1. **Логический (Logical view)**
   * *Что делает система?* Показывает ключевые сущности, модули, их связи.
2. **Процессный (Process view)**
   * *Как система работает?* Описывает взаимодействия процессов, параллелизм и механизмы обмена.
3. **Разработческий (Development view)**
   * *Как устроен код?* Структура репозиториев, модулей, библиотек, управляемость и удобство сборки.
4. **Физический / Развёртывания (Physical / Deployment view)**
   * *Где работает?* Определяет, на каких серверах/контейнерах крутятся компоненты и как они соединены.
5. **Сценарный (Scenario / Use cases / +1)**
   * *Как это используется на практике?* Наглядные сценарии (user stories, use cases), которые иллюстрируют взаимодействие всех предыдущих видов.

**В каком порядке представляют?**

Обычно идут **от общего к частному** (или от высокоуровневого контекста к деталям):

1. **Контекст** (общая схема взаимодействия с внешними системами, стейкхолдерами).
2. **Логический** (или «Container view» в C4), чтобы показать, какие крупные части системы есть.
3. **Процессный / Компонентный** (раскрываем внутреннее устройство, взаимодействие).
4. **Разработческий** (организация кода).
5. **Физический** (среда развёртывания).
6. **Сценарный** (проверяем всё реальными кейсами).

**Пример**

Для **интернет-магазина**:

* **Логический вид:** диаграмма сущностей «Товар», «Корзина», «Заказ», «Оплата», связи между ними.
* **Процессный вид:** как обрабатываются заказы, взаимодействие микросервисов «Каталог», «Оплата», «CRM» в режиме реального времени.
* **Разработческий вид:** структура фронтенда (React-модули), бэкенда (Spring Boot микросервисы), общие библиотеки.
* **Физический вид:** как сервисы развёрнуты на AWS (EC2, Docker, Kubernetes и т.д.).
* **Сценарии:** «Покупатель заходит на сайт, выбирает товар, оформляет заказ, система списывает оплату и формирует накладную».

Таким образом, архитектурные виды помогают «разложить» сложную систему на несколько простых для понимания слоёв и обеспечить ясность для разных стейкхолдеров.

**Вопрос 13. Опишите процесс ревью архитектуры решения. Зачем и когда оно необходимо? Команда, необходимая для арх ревью**

**Что такое ревью архитектуры?**  
Это проверка, соответствует ли архитектура решения всем необходимым требованиям (как бизнес-требованиям, так и техническим вроде надёжности, безопасности и т.д.), и нет ли в ней скрытых проблем.

**Зачем оно нужно?**

1. Чтобы **раньше** заметить возможные ошибки и сэкономить на дорогих переделках.
2. Чтобы **убедиться**, что все заинтересованные стороны (бизнес, безопасность, эксплуатация) понимают и одобряют архитектурные решения.
3. Чтобы **проверить**, что архитектура отвечает стандартам и нормам (например, по безопасности или законодательству).

**Когда проводить ревью?**

* **В начале проекта** (при проектировании) - чтобы не тратить время на реализацию неправильного решения.
* **При крупных изменениях** (добавлении новых модулей или переходе на новые технологии).
* **Регулярно** в больших проектах, где архитектура может меняться со временем.

**Кто участвует в ревью?**

* **Архитектор** (или внешний консультант), который понимает всю концепцию.
* **Ведущий разработчик** (Tech Lead), знакомый с деталями реализации.
* **Эксперт по безопасности**, чтобы учесть все риски.
* **Представитель инфраструктуры / DevOps**, чтобы оценить развёртывание и поддержку.
* **Представитель бизнеса/заказчика**, чтобы убедиться, что всё делается ради нужд проекта.

В итоге все вместе смотрят на схему системы, обсуждают потенциальные слабые места и решают, как улучшить архитектуру до начала (или в процессе) разработки.