00:00:00,000 - 00:00:13,040 - Ирин, привет! Меня зовут Андрей. Я представляю компанию «Системная аналитика», и я сегодня проведу у тебя техническое интервью. Предлагаю наш диалог построить следующим образом. Ты коротко, буквально, там, пять-десять минут расскажешь о себе.

00:00:13,080 - 00:00:28,560 - Что делало свой предыдущий опыт всё самое интересное? Я позадаю вопросы по этому опыту. Дальше мы с тобой ответим на теоретические вопросы. В конце решим небольшую задачку. И для завершения интервью я готов ответить на все твои вопросы.

00:00:28,700 - 00:00:30,180 - Да, супер.

00:00:30,800 - 00:00:46,670 - Всем, привет! Меня зовут Ирина. Наверное, я начну со своего первого опыта в IT. Начинала я в 2017 году мануальным тестировщиком в финтехе в компании «Диасоф». Я была в отделе отчетности ЦБ, и мы делали продукт

00:00:46,670 - 00:00:53,450 - Это на самом деле были маленькие модули, которые устанавливались на основной продукт Диасофта.

00:00:53,450 - 00:01:13,430 - При запуске этих отчетов у нас формировались отчеты для центрального банка, и моя задача как тестировщика была проверить, что все отчеты формируются корректно. Приходилось очень много работать с данными, потому что мы выверяли каждую колонку отчета, и именно в «Диасофте» я научилась базово работать с базами данных.

00:01:13,430 - 00:01:17,480 - Из баз у нас был Oracle и MS SQL Server.

00:01:17,600 - 00:01:26,730 - В Диасофте я доросла до старшего тестировщика и ушла через год и девять месяцев, потому что поняла, что больше не хочу развиваться в тестировании.

00:01:26,730 - 00:01:41,110 - И решила себя попробовать в системном анализе. Я устроилась работать в местную компанию-интегратора, которая занимается автоматизацией бизнес-процессов на базе CRM-систем и систем складского учёта.

00:01:41,110 - 00:01:51,280 - Моя должность называлась системный аналитик, но по факту я выполняла много разных задач. И, наверное, это больше были задачи проекта, тестировщика.

00:01:51,840 - 00:02:03,260 - Бизнес-аналитика, чем системного аналитика. Мне очень много приходилось работать с клиентами. Я одновременно могла вести до 5-6 проектов.

00:02:03,620 - 00:02:23,660 - Мои задачи были основные — это общение с заказчиками, сбор требований и формализация этих требований для разработки. Из артефактов аналитических, что мы делали? В основном, это юзкейсы, юзер-стори и шаблон, и техническое задание, которое мы писали по определенному шаблону, который был принят в компании.

00:02:24,360 - 00:02:43,940 - В этой компании я проработала три с половиной года, доросла до руководителя направления, но ушла из-за того, что поняла, что я достигла до какого-то потолка в развитии, и мне, наверное, хочется уйти больше в техническую сторону, развивать свои hard skills.

00:02:44,200 - 00:02:56,320 - Хотелось повысить свой уровень заработной платы. И я пошла работать в компанию «Магнит». Сейчас я работаю там, на позиции ведущего системного аналитика. У нас это «Мидл».

00:02:56,560 - 00:03:16,920 - И вообще моя история в «Магните» началась так. Я пришла на один проект, который был связан с маркировкой рекламы, который должен был запуститься. Но так получилось, что через три недели после запуска этот проект заморозили, и нашу всю команду отправили на другой новый проект, связанный с маркировкой.

00:03:17,000 - 00:03:26,980 - Уже зерна. Сейчас давайте кратко расскажу, в чём суть проекта. Есть такой сервис государственный, называется «Вгиззерно».

00:03:26,980 - 00:03:35,940 - И по требованию государства все участники оборота зерна и продуктов переработки зерна должны в этом личном кабинете фиксировать

00:03:35,940 - 00:03:53,570 - всё движение по зерну и продуктам переработки зерна. Там есть различные типы документов, номенклатуры и так далее. А так как магнит является крупным покупателем и поставщиком продуктов переработки зерна и самого зерна, то, конечно же,

00:03:53,570 - 00:04:08,110 - Наши сотрудники активно должны там формировать определенные документы. А сам личный кабинет в ГИС «Зерна» — он достаточно неудобный, потому что, чтобы выполнить одну операцию, сотруднику требуется до четырех минут.

00:04:08,110 - 00:04:21,830 - Это очень долго, особенно когда у тебя в день более тысячи документов пролетает. И поэтому основная цель нашего проекта — это автоматизировать работу сотрудников и сократить

00:04:21,830 - 00:04:35,680 - их время на обработку всех этих документов. Мы автоматически подгружаем эти документы с личного кабинета в «Гиззерно», обрабатываем их по определённой логике и там делаем с ними определённые операции.

00:04:36,500 - 00:04:49,180 - Проект, как я уже сказала, мы запускаем с нуля. То есть практически все процессы в команде и архитектуру приходилось выстраивать с нуля самостоятельно.

00:04:49,180 - 00:05:07,100 - Стек технологий у нас такой. У нас сейчас система — это Monolith. Есть Back, пишется на Python, база Postgres. Front пишется на JavaScript и Vue.js. Front и Back общаются по REST.

00:05:07,160 - 00:05:12,760 - Интеграция нашего сервиса в «ГИС «Зерно» происходит по протоколу СОАБ.

00:05:12,940 - 00:05:16,100 - Вот, очень интересно на самом деле.

00:05:17,240 - 00:05:29,880 - По команде. У нас сейчас в команде семь человек. Из них два аналитика, один тестировщик, остальные — разработчики. Бэк и фронт у нас отдельно. По моим задачам как аналитика. Давай я расскажу.

00:05:30,020 - 00:05:39,120 - Самое первое, конечно, что я делаю, это собираю и анализирую требования. Требования нам приносят бизнес.

00:05:39,120 - 00:05:57,410 - Бизнес как раз хорошо разбирается в законодательстве и в процессах, которые у нас выстроены в компании, и они формируют своё видение в виде концепта. И дальше мы анализируем этот концепт, собираемся с бизнесом, обсуждаем, и моя задача как аналитика — уже спроектировать систему.

00:05:57,410 - 00:06:12,610 - чтоб она, там, соответствовала бизнес-целям. Я пишу функциональные, не функциональные требования и проектирую систему по артефактам, которые у меня выходят. Это, конечно же, описание самого бизнес-процесса. В основном используем BPMN.

00:06:12,610 - 00:06:13,760 - Дальше.

00:06:13,760 - 00:06:29,890 - User Story. На самом деле, User Story используем редко, когда описываем требования к интерфейсу. Use Cases. Обязательно Use Cases. Суперчасто, когда мы описываем логику, вообще, работы нашего продукта. К Use Cases мы прикладываем Activity диаграмму.

00:06:29,890 - 00:06:36,890 - И секвенс-диаграмму, если мы описываем взаимодействие между системами, то есть как они интегрируются.

00:06:36,890 - 00:06:42,340 - Е-ар, диаграмма обязательна, то есть проектирование логической модели базы данных.

00:06:42,560 - 00:06:54,140 - Так, проектируем API-шку. То есть как у нас фронт общается с бэком в REST API. У нас есть специальный шаблон, который мы заполняем, да, что требования к бэку соблюдались.

00:06:54,260 - 00:07:14,600 - И рисуем прототипы в Pygmy, так как это внутренний продукт, поэтому дизайнеров сюда уже не привлекают, приходится справляться со своими силами. И ещё требования, наверное, к самому интерфейсу приписываем. Экранные формы и какие таблички, какие поля, и как они мапятся с эпишкой.

00:07:15,240 - 00:07:35,740 - По инструментам, которые используем в работе, все требования пишем в конфлюенте. Есть специальные у нас шаблоны, которые мы для себя подготовили. И все задачки декомпозируем в жире. Если говорить про диаграммы, то их рисуем в Plan-2-ML, в синт-квинт-диаграммы в основном, остальное в Draw.io.

00:07:36,220 - 00:07:42,820 - Так, «Свагер» и «Постмен», конечно же, для запроса. В принципе, всё.

00:07:43,240 - 00:07:53,470 - Спасибо тебе большое. Очень интересно, очень насыщенно. От тестировщика в аналитике очень большой путь, правда. И сразу не надо дополнительных вопросов задавать.

00:07:53,470 - 00:08:05,900 - в принципе, рассказала всё основное. Давай, исходя из твоего опыта, побольше поговорим про документацию. Тем более, вот, ты упомянула ТЗ, которое ты писала на предыдущем месте работы, юзеркейсы.

00:08:06,040 - 00:08:15,090 - Вернее, юзкейса, юзер-стори. Расскажи, пожалуйста, вот ТЗ ты упомянула, которое было в Диасофте. Что в нём включалось? Это был какой-то шаблон?

00:08:15,090 - 00:08:25,040 - Скажем так, государственный или какой-то общепринятый, или это был внутренний шаблон компании? Если это внутренний шаблон компании, тогда чуть-чуть подробнее, вот, что именно, какие разделы были в документе.

00:08:25,040 - 00:08:33,460 - Угу. Ну, это было не в «Диасофте», потому что «Диасофт» — это там я работала тестировщиком, это во второй компании. Эта команда, «Ф-5», называется. Маленькая компания.

00:08:33,460 - 00:08:48,520 - местная, чебоксарская. Там мы писали тазе по внутреннему шаблону, который был принят в компании. В этом шаблоне в основном у нас содержались цели и задачи проекта, и дальше мы описывали логику самой разработки, что система должна делать.

00:08:49,400 - 00:09:00,220 - Окей, супер. Давай тогда про юзкейс и юзерстори расскажи, что это за формат документирования, и можешь, пожалуйста, по примерчику привести. Давай с юзерстори начнём.

00:09:01,200 - 00:09:10,080 - User Story — это относится, если их определить к пользовательским требованиям. Обычно они пишутся по формату

00:09:10,080 - 00:09:24,840 - Я как такой-то пользователь хочу что-то делать, чтобы…» — то и там пишется уже цель этого пользователя. И обычно к юзер-стори ещё добавляют acceptance criteria, то есть критерий приёмки, чтобы проверить эту юзер-стори.

00:09:25,180 - 00:09:41,360 - Если говорить про пример, давай подумаем, наверное, по текущему проекту, если взять, да. Я как, например, администратор системы хочу видеть актуальные документы, которые пришли за сегодняшний день, чтобы оперативно их погасить.

00:09:41,980 - 00:09:56,460 - Окей, тогда такой вопрос. В целом, достаточно, но вопрос такой. Как нам понять, что юзер-стори составлено корректно? То есть, есть ли какие-то, может быть, критерии или методологии того, как нам понять, что мы сделали всё правильно?

00:09:57,060 - 00:10:10,160 - Ну, можно сказать, что acceptance criteria обязательно должна быть, да, чтобы мы могли проверить нашу юзер-стори. Она супер помогает, например, тестировщикам при тестировании. И разработчикам, кстати, тоже.

00:10:10,420 - 00:10:19,120 - А если говорить про саму юзерстори, то она должна быть с краткой. Там должна описываться какая-то одна

00:10:19,340 - 00:10:34,500 - Задача пользователя. Чисто один маленький функционал, да? И мы должны отобразить цель этого пользователя, зачем он это хочет. Ирина ответила правильно, с логической точки зрения. Но мне бы хотелось услышать о конкретной методологии. Например, «инвест».

00:10:34,500 - 00:10:48,510 - По первым буквам акронима юзер-стори должна быть independent — независимая от других историй, negotiable — обсуждаемая, valuable — ценная, estimatable — оцениваемая, small — маленькая,

00:10:48,510 - 00:10:50,980 - И «тестабл» – тестируемое.

00:10:51,000 - 00:10:52,240 - Окей.

00:10:52,500 - 00:10:54,940 - Хорошо.

00:10:54,940 - 00:11:15,330 - Юзкейсам перейдём. Также расскажи, пожалуйста, что это за формат, и примерчик можешь вот как раз-таки развернуть юзер-стори юзкейсам. Угу. Что такое юзкейсы? Мы их суперчасто, как я уже сказала, используем в своей работе. Практически, да, все процессы мы описываем в виде юз-кейсов.

00:11:15,330 - 00:11:20,000 - Если говорить по шаблону, там описывается, как

00:11:20,540 - 00:11:35,800 - пользователи и элементы системы взаимодействуют друг с другом, что происходит на каждом этапе, и есть ли какие-то альтернативные сценарии. Если строить юзкейсы, то там обязательно должно быть предусловие. То есть мы должны описать

00:11:36,380 - 00:11:46,700 - Какое событие запускает юзкейс. Вот. Дальше мы должны приписать действующих лиц, которые у нас будут в юзкейсе представлены.

00:11:46,700 - 00:12:03,350 - Ну, например, пользователь или система. Можно, кстати, это ещё детальнее декомпозировать, как мы это делаем. Мы пишем пользователь, фронт и бэк. И прям прописываем, что у нас делает пользователь, что у нас показывает фронт, как он взаимодействует с бэком, что отдаёт бэк.

00:12:03,350 - 00:12:04,900 - Вот.

00:12:05,500 - 00:12:06,520 - Е-е-е!

00:12:06,540 - 00:12:08,520 - Дальше пример, да, привести?

00:12:08,520 - 00:12:27,920 - Так, давай тогда возьмем наш уже юзер, юзер-столик, который мы описали. Что у нас с предусловием? Предусловие — у нас пользователь должен быть авторизован в системе, чтобы посмотреть документы. Действующие лица — это администратор.

00:12:28,040 - 00:12:33,500 - Фронт и бэк. Вот сейчас я распишу, как мы это проектируем, как это мы расписываем.

00:12:33,660 - 00:12:42,520 - Первый шаг. Пользователь заходит в раздел, например, «Документы». Дальше. Второй шаг. «Фронт».

00:12:43,100 - 00:12:56,180 - Обращается к Бэку, отправляет такой-то запрос на получение таких-то данных. И там мы прописываем, что это за запрос, и какие там фильтры по умолчанию идут.

00:12:56,620 - 00:13:09,500 - Вот. Дальше. Бэк обрабатывает запрос и отдает на фронт ответ со списком документов. И дальше фронт показывает пользователю. Окей.

00:13:09,820 - 00:13:11,320 - Что-то ещё?

00:13:11,320 - 00:13:31,410 - Также тут можно какие-нибудь альтернативные сценарии добавить, да, например, что будет происходить, если, например, пользов… у пользователя нет права на просмотр этого рееста. Тут, да, уже альтернативный сценарий прописываем, что, там, бэк валидируют, например, токен пользователи, да, понимают, что для этой роли нет доступа к этому

00:13:31,410 - 00:13:49,580 - реестру и уже выдают какую-нибудь ошибку, четырехсот третью, например. И фронт, соответственно, отображает сообщение, что нет доступа. В целом, да, согласен, про альтернативные сценарии не нужно забывать. У меня, наверное, последний вопрос будет по опыту и, соответственно, по документации. Ты упомянула шаблон API.

00:13:49,580 - 00:14:09,990 - Можешь рассказать, вот, что в этом шаблоне было? То есть какие разделы? Ты сама его придумала, или это был какой-то в компании, который ты использовала? И ещё, знаешь, такой вопрос. Вот ты сейчас расскажешь, как бы ты его улучшила. Угу. Ну, на самом деле, этого шаблона раньше в компании не было.

00:14:09,990 - 00:14:12,680 - Разработала я сама.

00:14:13,000 - 00:14:31,840 - На самом деле, несколько вариантов, да, было этого шаблона. Сначала я там просто писала, как бэк должен обрабатывать запросы фронта и что должен в ответ присылать. Потом я поняла, что там не хватает каких-то query-параметров и pass-параметров. Мы же тоже должны как-то понимать, бэк должен понимать, как это вообще фильтровать и так далее. И сейчас у нас шаблон выглядит таким образом.

00:14:31,840 - 00:14:36,560 - Сначала мы описываем, что это за ресурс, потом описываем, вот, на страничке,

00:14:36,560 - 00:14:43,540 - Этот метод и сам endpoint. Описываем, для кого, для какой роли этот endpoint доступен.

00:14:43,540 - 00:14:46,680 - Обязательно описываем

00:14:47,180 - 00:14:51,720 - Дальше. Какие… Если есть пас-параметры, пас-параметры.

00:14:52,260 - 00:15:05,740 - Если есть query-параметры, какие query-параметры у нас есть, какого они типа, какие есть ограничения, какие есть значения по умолчанию. И описываем, с какими полями в базе данных они мапятся.

00:15:05,740 - 00:15:24,560 - Вот. С параметрами разобрались. Дальше переходим к описанию того, как у нас бэк валидирует этот запрос. Ну, например, если, например, запрос у нас неполный или какие-то там обязательные поля не заполнены, то мы какой код ответа с каким дескрипшеном должны возвращать. Или, например, если…

00:15:24,740 - 00:15:39,340 - Токен невалидный. Ну, вот, все эти варианты расписываем, да, ошибочные варианты. И расписываем, что у нас должно происходить, если запрос валиден, всё прошло, какой ответ мы возвращаем. И дальше у нас идут параметры ответа.

00:15:39,340 - 00:15:49,560 - Тут прописываем, да, все параметры. Если там есть какие-то вложенные объекты или массивы, их тоже расписываем, да, прям структуру приводим.

00:15:50,460 - 00:16:00,820 - Что еще? Обязательно прописываем по умолчанию какие типы данных и с какими полями в базе данных они маппятся. То есть как мы рассчитываем эти параметры, что выдаем.

00:16:01,820 - 00:16:10,860 - Из того, что, наверное, можно было улучшить, мы сейчас не прописываем примеры запросов и ответов, наверное, да, наверное, это можно было бы добавить.

00:16:11,580 - 00:16:31,520 - Да, согласен, с примерами запросов было бы, наверное, проще разрабам, но ты сама сказала, куда можно расти дальше. Окей, предлагаю тогда перейти в часть с теоретическими вопросами. Давай начинаем, скажем так, с классического вопроса. Какие виды требований ты знаешь? И приведи, пожалуйста, примеры тех видов требований, которые назовёшь.

00:16:31,600 - 00:16:32,760 - Угу.

00:16:32,780 - 00:16:44,680 - Ну, самое верхнеуровневое требование – это бизнес-требования, которые описывают цели и задачи на уровне бизнеса, на уровне компании. Да, например, мы хотим повысить выручку на столько-то.

00:16:44,680 - 00:16:55,420 - И так далее. Дальше есть пользовательские требования, которые описывают цели и задачи пользователя, да, которые он может решить при использовании системы.

00:16:55,520 - 00:17:00,980 - И третье — функциональные требования. Тут уже описывается, как система.

00:17:01,100 - 00:17:14,760 - Какой функционал должна иметь система, чтобы обеспечить пользовательские требования и бизнес-требования? Также есть нефункциональные требования. Это, например, критерии качества, какие-то свойства системы.

00:17:14,860 - 00:17:25,900 - Например, масштабируемость, требования к безопасности, скорость отклика, доступность, удобство использования и прочее.

00:17:27,660 - 00:17:46,400 - Если говорить про примерчик, например, какого-нибудь бизнес-требования с нефункциональными, понятно, да, там, требования к безопасности, к отказу устойчивости и так далее. Какое-нибудь бизнес-требование, прям конкретное. Уменьшить, например, время…

00:17:47,300 - 00:17:54,100 - Оптимизировать работу сотрудников, уменьшив время обработки документов.

00:17:54,940 - 00:17:56,760 - Окей. К примеру.

00:18:11,440 - 00:18:25,180 - Опытные же узнают про совмещение работ и поиск валютных удаленок. Тематические чаты по всем специализациям с полезными материалами — без флуда. Сходки по всему миру и созвоны сообщества случаются каждую неделю.

00:18:25,180 - 00:18:37,280 - На них мы помогаем друг другу советом и поддержкой. Вступай в сообщество осознанных и меркантильных. Мы тебя ждём!» — Следующий вопрос про требования. Расскажи, пожалуйста, что такое переходные требования.

00:18:40,600 - 00:18:46,960 - Переходные требования, я думаю, может быть, это требования, которые могут быть, могут изменяться.

00:18:47,980 - 00:18:56,640 - Ну, я так полагаю. А можешь расшифровать «изменяться» как? Может быть, это какие-то временные требования?

00:18:57,400 - 00:19:14,220 - А, окей, понял. Так, двигаемся дальше. Давай от требований к техникам выявления требований. Какие способы, техники выявления требований можешь назвать? Какие используешь в работе? Может быть, какие-нибудь фишечки расскажешь, как наиболее эффективно извлекать требования? Угу.

00:19:15,040 - 00:19:33,880 - Так, сейчас самое основное, это, наверное, то, что мы всегда используем, это интервьюирование, да, то есть интервьюирование может быть как в фокус-группе, так и индивидуально с каким-то человеком. Также можно письма, анкеты, опросники.

00:19:34,360 - 00:19:37,820 - Можно идти в поля и смотреть, как

00:19:38,540 - 00:19:42,700 - Люди работают. АБ-тесты, может быть.

00:19:46,180 - 00:19:58,240 - Может, какие-нибудь фишки? Как из практического опыта, что из этого наиболее эффективно? Необязательно из тех, которые ты перечислила. Может, ты можешь рассказать, не знаю, как там наиболее эффективно проводить то же самое интервью?

00:19:59,500 - 00:20:04,440 - Например, к интервью, да, чтобы интервью прошло эффективно, надо обязательно к нему готовиться.

00:20:04,440 - 00:20:22,120 - Нужно обязательно обозначить цель, что я хочу из этого интервью получить, обозначить длительность интервью, да, чтобы его не затягивать. Потому что когда ты чётко не обозначаешь цель и примерную планируемую длительность, может такое случиться, что можно, там, несколько часов болтать и ни к чему не прийти.

00:20:22,300 - 00:20:25,800 - и уже жёстко фасилитировать само интервью.

00:20:27,160 - 00:20:31,380 - Да, согласен, с большими длинными встречами — это прям боль.

00:20:31,680 - 00:20:42,780 - Окей. Такой кейс. Предположим, ты аналитик на проекте, который создает мобильное приложение для банка. У нас нет аналогов, то есть у нас нет там зеленого, желтого, красного банков.

00:20:42,920 - 00:20:53,700 - Тебе поставили такую задачу. Твоя целевая группа — это 100 клиентов, и ты должна с них собрать требования. Вот каким образом ты построишь этот процесс сбора требований?

00:20:54,480 - 00:20:58,780 - А эти 100 клиентов — это целевая группа, которую мы выделили.

00:20:58,780 - 00:21:18,760 - Из других, остального количества людей, которые пользуются нашим приложением. Да, всё верно. Мы их выделили, но при этом ты вот смотришь на них, и у тебя нету в голове какой-то общей картины. То есть я имею в виду, что там есть и молодые, например, да, и пенсионеры, и женщины, работающие, и безработные.

00:21:18,860 - 00:21:23,200 - Как вот ты посмотрел на них, и ничего общего пока найти не можешь.

00:21:24,080 - 00:21:40,960 - Ну, наверное, я бы сначала их как-то классифицировала. Можно дать им заполнить опросник какой-нибудь, да, чтобы как-то ещё их классифицировать по группам, тест какой-нибудь пройти и так далее. Потом выделить, может быть, фокус группы, и уже фокус группах

00:21:41,480 - 00:21:47,120 - Исследования. Можно проводить интервью отдельно с каждым.

00:21:47,180 - 00:22:02,780 - Кстати, еще можно проводить тестирование интерфейсов. Например, если у нас есть какие-то готовые макеты, мы можем с человеком, к примеру, созвониться. Я могу показывать макет, и человек будет уже отвечать на вопросы. Так мы можем понять, насколько у нас приложение удобное.

00:22:03,120 - 00:22:04,680 - Окей, да.

00:22:04,820 - 00:22:20,680 - Также ещё, наверное, можно, если у нас есть соцсети мессенджеры, если мы крупный банк, у меня есть стопудово, мы можем какой-нибудь запрос туда закинуть или какую-нибудь интерактивную игру, да, что пользователи там прошли и помогли тоже нам в исследовании.

00:22:20,680 - 00:22:26,180 - Также на этой выборке людей можно какие-то АБ-тесты запустить, наверное.

00:22:26,720 - 00:22:37,540 - Да, в целом, согласен, интересный подход. Давай, тогда, дальше. Последний вопрос, наверное, про требования. Что такое верификация и валидация требований, и какая разница между ними?

00:22:39,760 - 00:22:57,820 - Разница есть. Сейчас, конечно, в терминологии немножко могу путаться, но, по-моему, верификация требований — это когда мы проверяем, соответствуют ли требования шаблону определенному, который принят, например, соответствует ли ГОСТу.

00:22:58,180 - 00:23:02,840 - И валидация… Так, я верификацию сказала, да?

00:23:03,200 - 00:23:11,320 - Угу. И валидация требований — это когда мы проверяем, соответствуют ли наши требования бизнес-концепту.

00:23:11,680 - 00:23:12,960 - Угу, окей.

00:23:13,220 - 00:23:29,600 - Двигаемся дальше. Давай перейдем в моделирование. Ты упоминала уже нотации, о которых мы сейчас будем говорить. Ты даже диаграмму упоминала, о которых мы будем говорить. Давай начнем с BPMN. Расскажи, пожалуйста, что это за нотация, какие элементы этой нотации ты можешь назвать?

00:23:29,740 - 00:23:44,310 - BPMN — это нотация моделирования бизнес-процессов. Супер широко используется, наверное, практически во многих компаниях, в том числе и у нас. Если говорить про элементы, то там есть пулы и дорожки. Самое первое, да, что мы рисуем.

00:23:44,310 - 00:23:52,220 - Дальше есть разные виды событий. Есть начальное событие, промежуточное событие и конечное событие.

00:23:52,520 - 00:23:54,800 - Есть различные таски.

00:23:54,880 - 00:24:09,120 - Таски бывают разных видов. Системные, пользовательские и так далее. Есть линии. Честно, не помню, как они сейчас называются. Но там есть разные виды линий. Например, если мы внутри

00:24:09,120 - 00:24:20,320 - пола и дорожки, рисуем линии, например, между тасками, да, и событиями, то они, там, пунктирные. Если мы рисуем между различными, например, элементами,

00:24:20,460 - 00:24:40,100 - например, из одной дорожки, из одного пола в другой пол, то там уже будет другая линия. Также есть шлюзы. Шлюзы бывают разных видов, но они соответствуют операторам. Или взаимосвключающие, или после применения которых у нас возникает разветвление в бизнес-процессах.

00:24:40,360 - 00:24:59,980 - Супер, да, да, окей, этого достаточно. Двигаемся тогда дальше в нотацию UML. Расскажи, пожалуйста, что это за нотация, с какими диаграммами приходилось работать, возможно, кроме тех, которые ты уже упомянула. В целом, про каждую из диаграммок, которые ты назовёшь, по паре слов было бы круто услышать.

00:25:00,600 - 00:25:17,480 - На самом деле, диаграмм U-mail — большое количество. Из тех, с которыми мы работаем, это, конечно же, sequence-диаграмма, диаграмма статусов. Кстати, я, по-моему, не упомянула про них, но мы её тоже, как-то, не используем. Что ещё?

00:25:18,300 - 00:25:35,380 - Ну, вот эти два активно используем. Также есть юзки, есть диаграмма, есть диаграмма развертывания, есть диаграмма классов. Я думаю, достаточно будет, да? Ну, наверное, да, это всё, что я вспомнила. Ну, там ещё много их.

00:25:35,380 - 00:25:44,430 - Для начала достаточно нам хватит с тобой пообщаться. Давай начнём с секвенс. Расскажи, пожалуйста, для чего она нужна, что она показывает.

00:25:44,430 - 00:25:50,340 - И, кстати, второй вопрос — как на сиквелинговой диаграмме показать асинхронное взаимодействие между системами?

00:25:51,040 - 00:25:59,640 - Да, секвенция диаграммы у нас показывает взаимодействие между несколькими объектами последовательно, да, по времени.

00:25:59,860 - 00:26:18,040 - Если мы показываем синхронное и синхронное взаимодействие, то тут есть разные виды стрелочек. Например, да, синхронное взаимодействие. У нас стрелочка должна быть… Кончик стрелочки должен быть закрашенный, а в асинхронном кончик стрелочки не закрашенный. Вот. Но есть… Можно ещё показать, например…

00:26:18,220 - 00:26:23,040 - Если мы используем этот паттерн

00:26:24,060 - 00:26:43,510 - То есть обработка и обратный кулбэк. Либо мы можем показать пулинг, когда система сама обрабатывает, и мы… Первая система несколько раз опрашивает вторую систему, пока вторая система не обработает уже первый запрос. Ну, здесь понятно объяснила. Да, в целом понятно.

00:26:43,510 - 00:26:50,860 - Про поллинг даже понятно. Давай про коллбэк. Можешь рассказать, что это такое? Это не относится к UML в целом? Что за технология?

00:26:50,980 - 00:27:10,860 - Это такая технология обратной подписки. Ну, это я так называю. Например, мы отправляем запрос в первую систему, во вторую систему, вторая система это обрабатывает, и дальше, после того, как она обработала, она отправляет запрос на URL-ку, которую мы указали в первую систему.

00:27:11,640 - 00:27:16,220 - Если технология callback… Можно сравнить это с веб-хуками ещё?

00:27:16,920 - 00:27:24,320 - Тогда рассказывай про веб-хуки. Ну в веб-хуке как раз они применяются на технологии коллбека.

00:27:24,940 - 00:27:36,460 - Каким образом они работают? В веб-хоке? Да. То есть, есть система 1, которая подписывается на определённые события от системы 2.

00:27:36,720 - 00:27:44,120 - И в тот момент, когда у нас в системе 2 возникает какое-то событие, она отправляет в систему 1 запрос.

00:27:44,200 - 00:28:02,740 - А тот URL, который система 1 указала. Вот, грубо говоря, вот так вот происходит. Окей, да, достаточно. Такой вопрос, опять возвращаемся с кулбеков и веб-хуков на секвенс-диаграмму. Каким образом показать зацикленность, то есть повторяющиеся действия или там цикл программы, например?

00:28:03,280 - 00:28:20,020 - Можно использовать loop. Ну, если это говорить на синтез-диаграмме, как изображается, то получается такой квадратик. Там пишется loop и вот это вот действие. Если в Plan Q meld писать, то пишем loop, end, само действие и… то есть, start, само действие и end.

00:28:21,100 - 00:28:29,440 - Да, да, согласен. Двигаемся дальше. У нас диаграмма классов, которую ты тоже называла. Расскажи, пожалуйста, для чего она нужна?

00:28:30,040 - 00:28:50,700 - Ой, на самом деле мы диаграмму классов не используем, мы проектируем модель базы данных в ЕР-диаграмме. А если говорить про диаграмму классов, то там, наверное, показываются сами объекты в виде классов, и показываются взаимоотношения между ними, как они связываются. А какие виды связей есть на диаграмме классов?

00:28:52,000 - 00:29:07,160 - Могу назвать, какие есть в ER-диаграмме. Потому что, насколько знаю, там виды связи отличаются. Ну окей, давай ER, тогда, хорошо. Да, в ER-диаграмме там связи один к одному, один к другому.

00:29:07,780 - 00:29:11,880 - Ко многим, много ко многим, ну если вот так перечислить.

00:29:11,880 - 00:29:29,240 - Ирина увела вопрос в сторону и рассказала о связях в ЕАР-диаграмме, но так и не ответила про диаграмму классов. Выделяют следующие связи на диаграмме классов — ассоциация, агрегация, композиция, обобщение и реализация. Ага, тогда вопрос, вот…

00:29:29,240 - 00:29:35,060 - Продолжая с диаграммы классов и AR-диаграммой, в чем разница между этими двумя диаграммами?

00:29:36,560 - 00:29:45,970 - Ну, ЕР-диаграмма, я думаю, используется больше для проектирования логической модели базы данных, да, всё-таки это про базу данных, как мы храним данные в базе.

00:29:45,970 - 00:30:05,880 - А диаграмма классов, она больше, наверное, про объекты. То есть какие у нас объекты существуют и как они связаны между собой в реальном мире. Окей. Так, у меня последний вопрос про диаграммы. В чем разница глобально между нотацией BPMN и нотацией UML, о которых мы с тобой сегодня говорили?

00:30:07,820 - 00:30:23,080 - Это абсолютно две разные нотации. В UML очень много разных видов диаграмм, а BPMN — там только одна диаграмма, которая описывает бизнес-процессы. Ну, у тебя в UML же есть диаграмма активности, которая похожа на BPMN.

00:30:23,820 - 00:30:29,540 - Ну, там разные элементы, во-первых, используются, они по-разному строятся.

00:30:36,920 - 00:30:54,240 - BPMN можно ещё показать время, то есть какой промежуток времени это происходит. А в диаграмме активности, насколько помню, такого специального элемента нет. Окей, окей, ладно. Так, двигаемся с тобой дальше, переходим в раздел с системным анализом.

00:30:54,620 - 00:31:02,360 - Первый вопрос. Давай поговорим с тобой про архитектуру. Соответственно, какие виды архитектуры ПО ты можешь назвать?

00:31:04,160 - 00:31:17,020 - Да. Самые популярные, наверное, это монолиты, микросервисы. Также есть системно-ориентированная архитектура и есть микрофронтенды, как отделение от микросервисов.

00:31:18,400 - 00:31:32,720 - Только, наверное, сервис-ориентированная, не системно-ориентированная. — А, да, сервис-ориентированная, да, говорилось. — Окей, давай сравним каждое из тех, которые ты назвала, между собой. Выделим плюсы и минусы той или иной архитектуры.

00:31:33,380 - 00:31:38,600 - Да, давай попробуем. Начну, наверное, с «Монолита». Что такое «Монолит»?

00:31:39,180 - 00:31:52,050 - Архитектурный шаблон, когда мы приложение делаем в виде одного большого приложения, да, которое развертывается как единое приложение, размещается в одном репозитории.

00:31:52,050 - 00:32:03,100 - Там мы можем использовать только один стек технологий, какой-то один, да, который приняли, и у него всегда одна база данных обычно. Вот. Монолиты — они

00:32:03,700 - 00:32:19,900 - Используются в основном в стартапах, либо когда мы начинаем какой-то проект, потому что на этих этапах их проще всего разрабатывать, их проще развертывать, и меньше требований, да, к системе предъявляются.

00:32:20,260 - 00:32:29,440 - Вот. И плюс. Плюсы, наверное, микросервисов — то, что там поддерживается транзакционность и согласованность данных, потому что база данных-то у нас одна, да, не надо

00:32:29,460 - 00:32:47,060 - Несколько баз данных связывать, как в микросервисах. Из минусов. Когда у нас система разрастается, становится очень много функционала, много бизнес-процессов в нашей системе, или появляется большая нагрузка, то тут уже возникает проблема масштабирования.

00:32:47,360 - 00:32:51,720 - И проблема, может быть, доступности

00:32:52,680 - 00:33:13,660 - Потому что проблема развертывания. Потому что когда мы вносим какое-то маленькое изменение в наш монолит, нам приходится, там, развертывать полностью целый монолит, и это занимает очень много времени. Либо если, там, какую-то ошибку, какую-то… в какой-то части кода мы внесли, да, то у нас падает уже вся система.

00:33:13,900 - 00:33:25,560 - И, конечно же, масштабировать её тяжело, потому что мы ограничены в технологиях, и с каждым разом всё сложнее и сложнее становится этот монолит разрабатывать.

00:33:26,260 - 00:33:40,120 - Так, про микросервисы, да? Микросервисы — это шаблон к проектированию архитектуры, когда наш сервис, наше приложение делится на маленькие микросервисы.

00:33:49,760 - 00:33:51,680 - Вот.

00:33:51,740 - 00:33:54,180 - Плюсы микросервисов.

00:33:54,220 - 00:34:13,460 - Легко их масштабировать, легко вносить изменения, потому что изменения мы вносим в какой-то маленький микросервис, его развертываем, и целая система у нас работает. И даже если какой-нибудь микросервис упадёт, у нас вся система, она будет работать, будет… не будет работать только конкретно упавший микросервис.

00:34:15,060 - 00:34:16,700 - Что ещё?

00:34:17,280 - 00:34:36,560 - Микросервисы могут писать… Каждый микросервис мы можем писать на совсем разных… То есть, каждый микросервис мы можем использовать разные технологии, да, в самом приложении, и микросервис писать на определённом стэке. Плюс, каждый микросервис, он обычно имеет свою базу данных.

00:34:36,600 - 00:34:39,340 - Если говорить про минусы, то

00:34:39,400 - 00:34:44,120 - Обычно микросервисы сложно декомпозировать, потому что

00:34:44,120 - 00:34:56,140 - Очень важно их правильно, да, эти микросервисы выделить, чтобы потом не было проблем у нас, и микросервис не превратился в связанный монолит, точнее, в распределённый монолит.

00:34:56,140 - 00:35:12,940 - Тут больше трудозатрат, особенно на интеграции, потому что мы должны поддерживать согласованность данных между микросервисами. И ещё минус — это отсутствие транзакционности, потому что у нас у каждого микросервиса может быть своя база данных.

00:35:15,080 - 00:35:28,040 - Сервис-ориентированная? Да, сервис-ориентированная архитектура. Она похожа на микросервисы, но там немного другой концепт. Там сами сервисы, они намного крупнее, чем микросервисы.

00:35:28,760 - 00:35:49,460 - СОА всё-таки ориентируются на переиспользование своих сервисов. А микросервисы, наоборот, стремятся к атомарности и малой связанности сервисов. Окей. И в Собычно используются шины данных в USB.

00:35:50,460 - 00:35:58,360 - Спасибо большое. Следующий вопрос. Расскажи, пожалуйста, какие паттерны реализации микросервисной архитектуры ты можешь назвать?

00:35:59,240 - 00:36:19,960 - Так, давай начнём, наверное, с декомпозиции микросервисов, да. Чтобы выделить микросервисы, часто используется domain-driven design, DDD, когда мы выделяем какой-то бизнес-контекст, и на основании этого бизнес-контекста уже выстраиваем наши микросервисы.

00:36:20,980 - 00:36:30,960 - И в ансоусинг есть практика, да, которая как раз применяет domain-driven design, когда мы эти бизнес-

00:36:31,900 - 00:36:40,440 - Элементы выделяем. Так, дальше. Следующий паттерн, наверное, «Api Getaway».

00:36:41,000 - 00:36:59,620 - Это паттерн говорит о том, как общаются фронт и микросервисы. API Gateway — это такой хаб, через который фронт обращается к API Gateway, и API Gateway уже понимает, к какому микросервису можно обратиться, чтобы взять данные.

00:37:01,420 - 00:37:04,260 - Он, кстати, по гитэвей может как…

00:37:05,140 - 00:37:11,700 - Хореограф выступать, если мы говорим про хореографию или оркестрацию. Вот. Это, кстати, тоже паттерны.

00:37:12,140 - 00:37:28,640 - Также есть паттерн BFF. Это когда у нас в системе несколько API Getaway. Он используется, когда у нас несколько разных фронтов. Например, есть мобилка, есть веб, есть, например, телефония, смс и прочее, прочее.

00:37:29,820 - 00:37:46,780 - Оркестрация хореографии. Можем тоже поговорить. По-моему, это называется «Паттерн сага». И он говорит о том, как мы можем управлять последовательностью выполнения бизнес-процесса, когда у нас микросервисы. Если у нас хореография,

00:37:47,940 - 00:38:01,500 - Архистрация, то у нас есть какой-то отдельный сервис, который управляет микросервисами. И именно он последовательно вызывает нужные микросервисы. Вот, это может, например, быть API Gateway, либо какой-то другой микросервис.

00:38:01,940 - 00:38:07,400 - А когда у нас хореография, то микросервисы последовательно вызывают друг друга.

00:38:07,400 - 00:38:26,570 - Ирина ответила верно, но в ответе не хватило структуры. Выделяют различные паттерны, в рамках которых представлены подходы к реализации. Например, паттерн декомпозиции на микросервисы включает следующие подходы. Декомпозиция по бизнес-возможностям или декомпозиция по поддоменам, упомянутые DDD.

00:38:26,570 - 00:38:39,620 - Паттерн управления данными в микросервисах состоит из CQRS, Saga, API-композиции, ивент-сорсинга. Паттерн коммуникации с внешними сервисами включает API Gateway и BFF.

00:38:41,400 - 00:38:55,200 - Саме. — Окей, супер. Супер, да. Спасибо большое. Доступно и понятно объяснила. Двигаемся постепенно в интеграции, то есть чем постоянно работают аналитики. Расскажи, пожалуйста, какие типы интеграции ты знаешь.

00:38:56,240 - 00:39:05,220 - Так, по типам интеграций они абсолютно все разные. Если говорить про старые типы интеграций, например, есть файловый обмен.

00:39:05,500 - 00:39:20,900 - Есть обмен между… используя общую базу данных. Есть RPC API. Туда относятся, например, JSON-RPC, XML-RPC. gRPC тоже относится к этому же семейству, но gRPC — он

00:39:21,660 - 00:39:30,640 - Отмен при помощи REST. Ну, это архитектурный паттерн. Архитектурный шаблон. Что ещё?

00:39:30,900 - 00:39:49,360 - Граф Куэль, брокеры сообщений, вебсокеты. Окей, да, достаточно. Я думаю, нам, опять же, хватит, чтобы поговорить сегодня. Давай начнём с самого распространённого, с Реста. У меня первый вопрос такой. Что такое RESTful принципы? Сможешь ли ты их назвать?

00:39:50,780 - 00:40:00,140 - Ну, как я уже сказала, REST — это не протокол, это архитектурный шаблон проектирования взаимодействия между системами, который опирается на

00:40:00,580 - 00:40:13,420 - определённые правила. Их несколько. Давай попробую их назвать. у нас должна быть обязательно клиент-серверная архитектура. у нас stateless.

00:40:14,140 - 00:40:33,940 - Сервер не должен хранить состояние клиента. Третье — слоистая архитектура. То есть между клиентом и сервером у нас могут находиться разные компоненты, типа роутеры, балансировщики и прочее, прочее. И причём каждый компонент должен знать только про существование следующего.

00:40:34,420 - 00:40:38,880 - Так, что ещё? Кэширование.

00:40:39,760 - 00:40:56,740 - Единообразие интерфейсов. То есть как раз HeyToUs, когда мы в ответе на запрос возвращаем все связанные объекты в виде гиперсылок. И, по-моему, Code on Demand называется, когда...

00:40:56,860 - 00:41:11,560 - Клиент может запускать код, который возвращен серверам. По-моему, так. Окей, да, супер. И вот он как раз необязательный. Первые пять — они обязательны. Чтобы система была RESTful.

00:41:11,920 - 00:41:24,760 - Как раз вопрос про то, обязательно ли их использовать, да, ты уже ответила, что нет. Можешь тогда привести, может, какие-нибудь практические примеры того, при тех ситуациях, когда не нужно использовать RESTful-принципы?

00:41:24,780 - 00:41:33,160 - Ну, например, «hate was», да. И мне кажется, вообще супермало кто это использует, потому что во многих случаях это избыточно.

00:41:34,820 - 00:41:49,700 - То есть передаётся в большой объём данных в виде киперссылок, и мало кому это нужно. Окей, да, почему нет. Погнали дальше, тогда. Давай про HTTP. Какие методы в HTTP ты можешь назвать?

00:41:50,820 - 00:41:56,400 - А что теперь глаголы основные? Это get, post, put, patch, delete, head.

00:41:57,140 - 00:42:01,680 - Можешь коротко, что каждый из них делает? Для чего он нужен?

00:42:01,820 - 00:42:05,980 - Get у нас используется для получения данных.

00:42:06,000 - 00:42:24,870 - Для чтения. Пост используется для записи данных. Пут и патч используются для изменения данных. Делейт понятно для удаления. Хэт он возвращает с заголовок. Угу, окей. Тогда, опять же, вопрос по методам. Какие из них являются этимпатентными? И что такое этимпатентность вообще?

00:42:24,870 - 00:42:40,720 - Идемпатентность — это свойство. Например, когда мы отправляем несколько запросов, нам должен всегда возвращаться один и тот же ответ. Если говорить про идемпатентность методов, то у нас идемпатентными считаются GET,

00:42:41,680 - 00:42:44,580 - Pod the light head.

00:42:45,400 - 00:42:51,060 - Окей, а каким образом нам сделать метод пост- и демпатентным? Можно ли это сделать?

00:42:51,220 - 00:43:08,640 - На самом деле, да, можно. Чтобы это сделать, нам… Мы можем, например, какой-нибудь ключ идемпатентности отправлять в запросе. Да, спасибо. Следующий вопрос. Скажи, пожалуйста, в чём разница между методами PUT и PATCH?

00:43:09,400 - 00:43:17,820 - Да, разница есть. Метод «пут» — он полностью перезаписывает данные, а метод «патч» — он частично их изменяет.

00:43:17,980 - 00:43:28,940 - Почти. Ну, смотри, допустим, вот у нас есть база данных, в ней всего одна запись, да, в этой записи, там, пять столбцов. Айдишка какая-нибудь, фамилия, мёдчество, там, паспортные данные, условно.

00:43:28,960 - 00:43:44,820 - Ты передаешь методом put и методом patch запрос на обновление вот этой записи единственной, но у тебя в баде, то есть в теле запроса, только три параметра. Вот что произойдет с базой, когда ты сделаешь метод put и потом метод патч.

00:43:46,280 - 00:44:00,240 - Да. Если мы делаем метод «пут», то у нас данные перезапишутся полностью, и только три поля будут заполнены, остальные будут нулевые, пустые. А если мы используем метод «патч», то тут

00:44:00,640 - 00:44:12,580 - Эти три поля только изменятся, остальные как и были, так и останутся. Но тут главное, чтобы ещё не было каких-то уникальных ограничений на эти поля, которые мы оставляем нулевые при перезаписи.

00:44:13,360 - 00:44:24,150 - А если будут ограничения, то что тогда произойдёт? Ну, тут, мне кажется, мы опишку должны корректно построить, чтобы в бэк сразу такие моменты

00:44:24,150 - 00:44:35,360 - например, мы можем ограничение уникальности, да, наложить, чтобы вот эти параметры были обязательны, да, при отправке запроса. Тогда будет… фронт сразу получит, там, некорректный запрос.

00:44:35,500 - 00:44:40,540 - Вот. Либо при обработке уже отдавать ошибку.

00:44:40,900 - 00:44:43,960 - Окей, окей. Ну, первый вариант, конечно, наиболее правильный.

00:44:44,180 - 00:45:00,680 - Да, тут я с тобой полностью согласен. Двигаемся дальше. Опять же, про методы. Расскажи, пожалуйста, можем ли мы использовать пост для получения информации, гет для создания нового ресурса? Если можем, то как? Если не можем, то почему?

00:45:00,800 - 00:45:14,760 - Пост для обучения информации. Да, конечно, можем. Например, многие не раз протоколы такие используют. Например, SUAP, там пост используется абсолютно во всех операциях, во всех запросах.

00:45:15,400 - 00:45:29,680 - Г тоже мы можем использовать. То есть никакого технического ограничения нет. Мы можем делать то, что хотим, но тут надо себе задавать вопрос, а как правильно и зачем мы это делаем. Ну, потому что это вообще не соответствует РЕС-концепции.

00:45:30,020 - 00:45:48,180 - Окей, нет, я согласен, что не соответствует, но раз ты говоришь, что технически это можно, может быть, ты можешь привести какой-нибудь бизнес-пример, когда, ну, по-другому просто нельзя. То есть мы можем, скажем так, правила игнорировать и сделать, вот, там, пост, например, для получения информации и гет для, там, создания ресурса.

00:45:49,620 - 00:46:02,730 - Ну, например, насчёт пост могу сказать. Когда мы передаём, например, какие-то чувствительные данные, например, пароль и так далее, GET не рекомендуется использовать, потому что в GET обычно всё же передаётся в query-параметрах.

00:46:02,730 - 00:46:17,280 - А эти пари-параметры… Ну это в URL-ке, получается, да? Это всё у нас сохраняется в браузере, и можно получить доступ к чувствительным данным. Поэтому в таких случаях можно пост использовать. Нужно. Окей.

00:46:17,600 - 00:46:24,360 - Да, хорошо. Вопрос, тогда, следующий. Какие коды ответов в HTTP ты можешь назвать?

00:46:24,480 - 00:46:41,560 - Я, наверное, по группам назову, да. Есть сотые информационные, есть двухсотые, которые говорят, что, вот, всё успешно. Есть трёхсотые — это коды перенаправления, четырёхсотые — клиентские ошибки, и пятисотые — серверные ошибки.

00:46:41,860 - 00:46:59,860 - Окей, тогда такой вопрос. Ты называла по группам, я просто конкретно спрошу, в чём разница между 204-м кодом ответа и 404-м кодом ответа? 204-й — это, по-моему, no content, нет контента. 404-й — это когда ресурс недоступен.

00:47:00,320 - 00:47:04,730 - Bed Getaway – ресурс недоступен, удален, перемещен и т.д.

00:47:04,730 - 00:47:24,510 - 204-ю мы должны использовать, когда, например, мы отправили запрос, этот запрос успешно выполнен, и контента нет. Например, при удалении, да, часто ставят 204-й ответ. Если мы успешно удалили, возвращаем 204-е. А 404-е, когда нет доступа к ресурсу.

00:47:24,510 - 00:47:28,560 - Ну это вот уже ошибочный дед. Это вот конкретно ошибка.

00:47:28,940 - 00:47:31,900 - Да, ошибка на клиенте, всё правильно.

00:47:31,960 - 00:47:40,960 - Такой вопрос у меня, опять же, по API-шкам. Что такое версионирование? Для чего оно нужно? И каким образом можно обеспечить версионирование API?

00:47:46,540 - 00:48:05,000 - Потому что такое часто бывает, что, когда мы обновляем API-шку, на новую версию переходим, то не все успевают подстроиться под нас, да, доработать свой обмен. И чтобы наши потребители могли использовать наш сервис, нашу API-шку, мы делаем версионирование.

00:48:06,340 - 00:48:16,740 - Где обычно указывают версию? Или нет каких-то рекомендаций? В URL. Окей. А можно указывать, там, не знаю, в хейдерсах или, например, в баде?

00:48:18,300 - 00:48:32,840 - В URL-джа проще указывать. Сразу видишь, что за версия. Ну, на самом деле, можно, почему нет, если мы так спроектировали. Но обычно указывают в URL-ке. Окей, согласен. Двигаемся дальше, говорим про свап.

00:48:32,840 - 00:48:42,140 - Расскажи, пожалуйста, что это такое, и назови, пожалуйста, структуру СОАП-сообщения. Угу. Это стандартизированный протокол.

00:48:42,760 - 00:49:03,640 - Окей, согласен. Такой вопрос, получается, у нас в слаб есть.

00:49:04,220 - 00:49:13,910 - Три сущности, скажем так. Это XML, это XSD и это WSDL. Вот что это такое, можешь о этих сущностях рассказать более подробно?

00:49:13,910 - 00:49:16,880 - Да, это абсолютно три разных документа.

00:49:16,880 - 00:49:32,910 - Начну с WSDL — Web Service Definition Language. Там описывается сам наш сервис и все операции, которые в этом сервисе разрешены. И как раз эти операции у нас… Каждая операция расписывается в своём XSD-документе.

00:49:32,910 - 00:49:42,940 - И там описываются, какие элементы могут быть, и атрибуты могут быть, и какого они типа данных, и, может быть, есть какие-то ограничения.

00:49:43,240 - 00:49:54,880 - А сам XML — это XML, которое мы передаем, само сообщение. Там как раз уже используются те операции, которые мы описали в других документах.

00:49:55,660 - 00:50:06,740 - О'кей, да. Хорошо, двигаемся с тобой дальше. Поговорим про асинхронное взаимодействие. Расскажи, пожалуйста, какие способы реализации асинхронного взаимодействия ты можешь назвать.

00:50:07,640 - 00:50:16,160 - Да, асинхронное взаимодействие — это классика брокерс-сообщений. Также REST можно сделать асинхронным, да, если

00:50:16,980 - 00:50:20,980 - Что ещё? WebSocket асинхронные.

00:50:21,420 - 00:50:38,020 - ГРПС может быть с арсинхронным. Граф Койл, мне кажется, тоже может быть асинхронным. Можешь рассказать? Вот ты перечислила достаточное количество технологий. Давай начнём с веб-сокетов и ГРПС. Коротко расскажи, что это за технологии, и

00:50:38,020 - 00:50:42,880 - Можешь сравнить их потом? То есть когда использовать ту или иную технологию?

00:50:43,040 - 00:50:47,140 - Ну, веб-сокеты — это протокол прикладного уровня.

00:50:47,520 - 00:51:05,700 - И как оно работает, когда между клиентом и сервером открывается соединение, оно двунаправленное и работает в режиме реального времени. И данные там передаются тоже в режиме реального времени. Он суперпростой, суперпрост в реализации, и очень часто используется, когда, например, чаты должны сделать или игры.

00:51:06,780 - 00:51:11,580 - Так, и если говорить про gRPC, это уже…

00:51:11,580 - 00:51:32,060 - Фреймворк для реализации удаленного вызова процедур. И там есть требования к формату передачи данных. Там всегда бинарный формат. Есть протокол ProtoBuff, который как раз занимается сериализацией и десерилизацией данных. И как транорт, там всегда используется HTTP 2-й версии.

00:51:32,060 - 00:51:43,980 - Который поддерживает двунаправленные потоки, и, там, несколько поток в одном соединении можно отправлять. И как раз получается у нас стриминговая передача данных.

00:51:44,820 - 00:52:00,100 - Окей. А WebSocket понял, где использовать, там, чаты игры? GFPC где используется? Он используется в основном, я думаю, когда нам нужно связать, передавать данные между микросервисами, когда большой объём данных, и нам нужно быстро их передать.

00:52:00,360 - 00:52:09,950 - Окей, да, хорошо. Погнали дальше, тогда. Ты упомянула брокеры. Расскажи, пожалуйста, принцип работы брокеров. В общем, всё, что ты про них знаешь.

00:52:09,950 - 00:52:29,300 - Угу. Ну, вообще, брокеры можно разделить на две категории по механизму действия. Первое — это пул и пуш. К брокерам, которые реализуют метод пуд, относятся, например, Kafka, а к пуш, например, Rabbit and Co. из популярных.

00:52:29,520 - 00:52:36,380 - Давай про принцип действия каждого расскажу, да, как Кавка у нас работает. Есть

00:52:38,260 - 00:52:55,240 - Паблишер есть субскрайбер, то есть система, которая публикует и которая подписывается, и есть сама Kafka, сервера Kafka. И паблишер публикует в топе Kafka какое-то сообщение, а паблишер его забирает, когда ему это нужно. И тут...

00:52:55,820 - 00:53:14,680 - Особенность Kafka в том, что эти сообщения в топиках, они хранятся, могут даже вечно храниться. Вот на самом деле, там, конечно, есть настройка, через какое время чистить, да, вот эти данные в топиках. Но дело в том, что мы несколько раз можем обратиться и забрать данные из топиков.

00:53:15,320 - 00:53:21,180 - Вот, и «Кавкана» тоже в стивенинговом режиме передаёт данные, то есть супербыстро.

00:53:22,080 - 00:53:28,440 - Если мы говорим про «Rebit&Q», то это стандартный брокер, который реализует

00:53:29,220 - 00:53:43,080 - Обеспечивает гарантию доставки, да, и последовательность доставки. Там есть Producer и Consumer. Producer — это тот, кто отправляет сообщение, а дальше Rabbit&Q берет и доставляет это сообщение

00:53:44,020 - 00:53:45,580 - Консюмеров.

00:53:47,900 - 00:53:51,500 - Окей, да, с этим тоже понял.

00:53:51,560 - 00:54:09,700 - Есть ещё такая, скажем так, сущность, как сервисные шины данных или Enterprise Service Bus. Можешь рассказать про неё? Что это такое? Угу. Ну, сейчас, мне кажется, её редко где используют. Шины данных она больше в SOA используется.

00:54:10,080 - 00:54:22,330 - Тут она тоже используется для передачи данных между сервисами. То есть, первый сервис передаёт данные брокеру… Ой, не брокеру, а ESB. И ESB уже, там, может

00:54:22,330 - 00:54:31,860 - Данные переформатировать можно в определённый формат и передать уже второму сервису. Какой состав шины данных? Что туда входит?

00:54:32,520 - 00:54:37,880 - Насколько я помню, там тоже используется брокер сообщений внутри «Счины».

00:54:38,920 - 00:54:51,400 - Ещё что-нибудь? И там ещё, по-моему, есть компоненты, которые отвечают за преобразование данных. Окей, а для чего они нужны?

00:54:51,920 - 00:55:02,760 - Почему мы можем… Не можем без них, например. Ну, например, да. В чём проблема ESB? В том, что там можно заладить, жить, слишком большую какую-то без… логику.

00:55:04,180 - 00:55:14,040 - Первый сервис, например, работает с одним форматом данных, да, он передает их брокеру, а второй сервис, он работает со вторым форматом данных. Не брокер, а ESB. А ESB уже…

00:55:14,500 - 00:55:30,120 - конвертируя первый формат данных во второй формат данных, чтобы второй сервис мог его принять. То есть какая-то логика обработки уже у нас имеется на самом брокере. И получается сильная связанность сервисов от ESB. То есть постоянно почему-то

00:55:33,600 - 00:55:50,890 - В конце Ирина всё-таки вывела на состав USB, но вопрос раскрыт не до конца. Сервисная шина данных состоит из нескольких компонентов — брокера, о котором мы говорили, комплекта адаптеров, которые служат для преобразования данных, а также инструментов мониторинга и контроля.

00:55:50,890 - 00:55:55,710 - Бизнес-логики на шине быть не должно — она обычно вынесена в отдельный слой.

00:55:55,710 - 00:55:58,060 - Окей, да, ладно, хорошо.

00:55:58,060 - 00:56:13,500 - Двигаемся дальше. У меня последний вопрос про веб-сервисы. Расскажи, пожалуйста, про такие понятия, как идентификация, аутентификация и авторизация. Что это такое и каким образом ее можно обеспечить в веб-сервисах?

00:56:14,680 - 00:56:31,030 - Да, хорошо. Это разные понятия. Идентификация — когда мы идентифицируем нашего пользователя, ну, например, по логину, да, и понимаем, есть ли он в нашей системе. Аутентификация — это проверки его данных, например, логина и пароля, да, и мы проверяем.

00:56:31,030 - 00:56:41,600 - его доступ к нашей системе. И третья авторизация — мы проверяем право пользователя на доступ к ресурсу определённому, если у него права.

00:56:42,180 - 00:57:03,060 - Про методы можно через GVT-токены. Особенно часто сейчас используются в микросервисах, да, потому что это обеспечивает безопасность авторизации, аутентификации. Можно стандартно через логин, пароль, но это достаточно устаревший метод. Если говорить

00:57:03,320 - 00:57:12,760 - Аутентификацию на фронте можно ещё ССО использовать. Single sign-on, например, когда мы авторизуемся через соцсети в каких-нибудь

00:57:13,780 - 00:57:23,940 - Окей, супер. Давай подробнее про JVT. Начнём с него. Как он работает? Да, JVT-токен — это определённый токен, который можно

00:57:24,600 - 00:57:41,520 - Ну, он зашифрованный, получается, да. Если его дешифровать, он, получается, состоит из хедера, пейло, да, с самой полезной нагрузки. Там у нас вся информация содержится, нужная нам, и сигнатура — это уже ключ. Да.

00:57:42,320 - 00:58:00,080 - Говорить про инструменты, которые помогают работать с JVT-токенами, например, Key Clock. Мы Key Clock в нашем проекте используем сейчас для авторизации пользователей. Как он работает? Есть Access Token, есть Refresh Token. И как раз через Access Token у нас происходит аутентификация пользователя.

00:58:00,080 - 00:58:20,640 - Access Token — он живет короткое время. У нас это примерно 5 минут. Ну и вообще рекомендованное время жизни, да, там, 5 минут, не больше. И через 5 минут этот токен протухает. И чтобы его обновить, у нас используется Refresh Token. Через… Используя Refresh Token, мы обращаемся снова к E-clock, и он выдает нам новую связку токенов.

00:58:21,320 - 00:58:27,740 - Окей, да, супер. Двигаемся, тогда, в ССО, которое ты назвала, как работает Single Sign-On.

00:58:27,740 - 00:58:28,850 - Угу.

00:58:28,850 - 00:58:38,790 - Тут у нас уже посредником выступает, например, соцсеть, через которую мы авторизуемся. Да, тоже у нас запрос идет в эту соцсеть.

00:58:38,790 - 00:58:55,600 - Там его… а в окне соцсети мы вводим логин и пароль. Соцсеть нас идентифицирует, проверяет правильность введённых данных. И если всё окей, он уже также отправляет токены приложению. Окей, да, хорошо.

00:58:55,600 - 00:59:04,940 - Может быть, какие-нибудь ещё способы аутентификации, авторизации можешь назвать? Мы с тобой, по сути, разобрали GVT, мы с тобой разобрали ССО. Что-нибудь ещё может быть?

00:59:04,980 - 00:59:07,340 - Вухфакторный.

00:59:07,820 - 00:59:25,900 - Как она работает?

00:59:26,700 - 00:59:47,640 - Окей, давай тогда спрошу ещё так. Есть способ обеспечения аутентификации авторизации с помощью сертификатов? Да, кстати, да. Можешь рассказать, пожалуйста, подробнее, как это работает? Да, кстати, авторизацию по сертификату у нас сейчас на проекте используется, когда мы интегрируемся в «Гизерно».

00:59:47,640 - 00:59:52,020 - А что такое сертификат, да? Он состоит из открытого и закрытого ключа.

00:59:53,280 - 01:00:09,880 - Есть криптопровайдер, и в чем сейчас наши требования, да? Все запросы, которые мы отправляем в «Абгейсерно», мы должны подписать усиленные квалифицированные подписью и должны приложить туда значение нашего сертификата. И как раз по этому сертификату

01:00:10,200 - 01:00:20,970 - гейзерно понимает, что вот это тот самый пользователь, тот самый аккаунт, и дальше он уже проверяет саму подпись. Окей, хорошо.

01:00:20,970 - 01:00:41,220 - Я предлагаю двигаться дальше. У нас, в принципе, подходим к завершению части с теорией. Давай поговорим про базы данных. Расскажи, пожалуйста, в чем разница между SQL и новой SQL-базами данных, когда используют первые, когда вторые, и какие преимущества и недостатки тех или иных видов баз данных можешь назвать.

01:00:41,580 - 01:00:42,680 - Угу.

01:00:43,560 - 01:01:03,560 - Первая разница, самая основная, наверное, — это типи охранения данных. Эскуальные базы данных — это реалиционные базы данных, и хранение данных происходит в виде таблиц, жестких таблиц, нормализованных, структурированных и так далее. И важное свойство реалиционных баз данных — это транзакционность.

01:01:05,320 - 01:01:23,740 - Да, и согласованность данных. Если мы говорим про ноискуэльные базы данных, а это переводится как «not only SQL», то тут уже формат хранения данных может быть абсолютно любой. И ноискуэльные базы данных, они разделяются на несколько видов. Сейчас попробую перечислить основные.

01:01:24,060 - 01:01:35,800 - Например, key-value базы данных. Например, Redis, когда у нас данные хранятся в виде пары ключ-значения. Redis, кстати, используется активно при работе с кэшем.

01:01:36,360 - 01:01:49,992 - Для хранения кэша. Дальше есть документы-ориентированные базы данных, например, MongoDB, где данные хранятся в виде джейсончиков, документа в виде джейсон. Дальше есть