00:00:00,000 - 00:00:10,240 - Мы общаемся на «ты», комфортно ли тебе так будет? Да, да, на «ты» комфортно. Угу, хорошо. Борис, у тебя немножечко громковато на фоне.

00:00:10,800 - 00:00:26,920 - Странно, что зум — это всё немножечко убираем. Так, сейчас только найти надо. Хорошо, хорошо. Тогда я думаю, что мы можем постепенно начинать. Серёж, тебе слово передать? Угу, да.

00:00:27,740 - 00:00:46,440 - Так, да, привет ещё раз. Расскажи в целом… Ну, так, объясню, наверное, у нас сейчас будет собес. Вот мы пробежимся по темам некоторым, которые участвуют у нас в грейдах, в точке в целом. Поговорим про инфраструктуру и так далее. Вот. Расскажи в целом про команду, про проекты.

00:00:46,580 - 00:00:54,220 - Какую роль ты занимал в команде? Может, что-то интересное вспомнишь? А, ещё раз всем привет! Да, в основном у меня опыт был в интехе.

00:00:54,280 - 00:01:10,260 - Три-четыре года был WebMoney опыт. Касаемо проектов, самые интересные это были C2C, то есть у нас была внутренняя валюта в рамках компании, и её нужно было переводить, там, к примеру, в доллары, в рубли, то есть некоторая биржевая, так сказать,

00:01:10,300 - 00:01:20,680 - Некоторые биржи. Вот. Этот проект у нас был на Monolith. В основном из стека у нас был ASP.NET Core. Последняя версия — это был на тот момент.NET 6-й.

00:01:20,920 - 00:01:31,580 - Потом мы его тоже переписывали уже на микросервис и переносили. Был опыт с EF-Core, с Dapper. По базам данных был Postgres, SQL-сервер, Docker, то есть все вот эти новомодные штуки.

00:01:31,680 - 00:01:38,560 - Что касается самой команды, у нас она состояла из 10 до 15 человек примерно в этом интервале.

00:01:38,560 - 00:01:44,000 - Были в основном по большей части бэкэнд-разработчики на.NET и Java.

00:01:44,000 - 00:02:01,530 - Был один QI, был архитектор и был Team Lead. Вот. Второй проект — это был СБП, то есть система быстрых платежей. Нужно было как раз, чтобы в нашей системе были переводы по системе СБП. Вот. Этот проект уже был на микросервисах, сразу написан.

00:02:01,530 - 00:02:05,220 - Поэтому тоже был опыт с монолитом и с микросервисами.

00:02:06,240 - 00:02:12,040 - Вот, как ты… Да какую роль в целом в команде занимал? То есть какие-то задачи, может, сам грумил?

00:02:13,820 - 00:02:29,600 - У нас обучение было, да. Сперва я пришёл вообще в компанию стажёром, вот, потом вырос до сеньора. Тоже был менторинг, там два-три человека в рамках своей команды обучал. В основном просто разработчик, но вот когда мы переписывали условно на микросервисы,

00:02:29,600 - 00:02:36,820 - У меня с ними был, так сказать, небольшой опыт, но он был. Вот, там тоже нужно было принимать некоторые решения.

00:02:39,620 - 00:02:41,120 - Хорошо.

00:02:41,180 - 00:02:51,940 - Я думаю, сразу к теме будем переходить, к первой. Давай про базы поговорим. Расскажи, с какими базами работал, опыт, кейсы, может быть.

00:02:51,940 - 00:03:10,240 - Угу. В основном были базы реляционные — это SKL сервер. Потом мы делали миграцию на Postgres. Вот. По новый SKL-им базам данных у нас был Mongo, Clickhouse, если его тоже можно, по сути, назвать. Но там взаимодействия было с ними мало, в основном реляционки. Касаемо

00:03:10,300 - 00:03:21,600 - Каких кейсов? Ну, это оптимизация у меня была с индексами работа, с джойнами, хронимки переписывал на EF Core, потом с ними тоже у меня был опыт. А что ещё?

00:03:21,600 - 00:03:37,920 - Ну вот, прям, полноценный базовый… Сейчас уже, вот, ты, ну, с Postgres-ом, правильно, работаешь? Да, сейчас с Postgres-ом. А ты, вот, насколько глубоко, там, копал, как он устроен, про внутрянку, там, уровни изоляции, вот это вот всё?

00:03:37,920 - 00:03:48,450 - Ну, про уровни изоляции и транзакций разбирал это. Про внутрянку даже был небольшой опыт, прям, с настройкой PG-конфа, вот, но не сказать, чтобы прям как-то детально.

00:03:48,450 - 00:03:59,640 - Но внутрянки, внутренние моменты базы данных копал прям хорошо разбирался, потому что на проекте у нас был прям очень сильный SQL-разработчик и получал от него опыт.

00:04:01,080 - 00:04:21,480 - А вот в целом, вот в Postgre, какие уровни изоляции, например, есть? Самый минимальный. Так, а мне слышно, да? Да, да. Ага. А, ну, по сути, в рамках нескольких баз данных, вот SQL Server у нас четыре, а в рамках Postgres у нас их тоже четыре, но уровень read and Commit там его нет. По умолчанию у нас Read and Commit идет.

00:04:22,620 - 00:04:25,740 - Окей, а ты вот сам как-то пользовался?

00:04:25,820 - 00:04:45,650 - Ну вот, уровнями изоляции, может, какую-то транзакционность разрулила, такую сложную, или не было кейсов? А был опыт, у нас в основном был, опять, это же финтех, очень много финансовых операций, и вот когда были какие-то сложные транзакции по переводу денежных средств, там я использовал уровень «серилайзабл», чтобы была максимальная согласованность данных.

00:04:45,650 - 00:04:49,360 - Так же было использование уровня Read Committed, ну, там

00:04:49,380 - 00:05:01,340 - Для Таски уже, честно, не вспомню, но это вроде бы связано с отчётами. Там не нужна была дикая какая-то согласованность данных, нужна была максимальная производительность. Окей.

00:05:02,280 - 00:05:13,300 - А если бы индексы затронуты? То есть, ты, получается, сам проектировал таблички, да, самостоятельно? Вот расскажи, чем руководствуешься, когда размечаешь индексы?

00:05:13,880 - 00:05:30,290 - А, ну, сперва как, да, сперва, понятно, нужно спросить у, в целом, команды, там, кто пользуется системой, по каким столбцам, допустим, будет очень активный поиск идти. Это, там, сам либо я как-то понимаю, либо спрашиваю, условно, вот, и поактивно, ну.

00:05:30,290 - 00:05:38,580 - По каким столбцам у нас будет идти очень активный поиск? Соответственно, самое такое первое решение, которое приходит в голову, это навесить индексы.

00:05:38,580 - 00:05:42,490 - А второе, когда уже условно таблица сформирована,

00:05:42,490 - 00:05:54,420 - Обычно приходит, условно, задачка, там, что запросы выполняются долго, надо посмотреть, вот. И ты идешь, там, смотреть либо план выполнения запросов, и параллельно раскручиваешь систему, есть ли индексы.

00:05:54,520 - 00:06:02,560 - Если есть, то почему они, условно, не выполняют свою роль, условно, по ускорению запросов? Вот, то есть это в моменте уже разбираешься.

00:06:03,900 - 00:06:06,520 - О, а ты тут про план запросов задел.

00:06:07,380 - 00:06:12,760 - А знаешь, чем отличается вот Explain от Explain Analyze в

00:06:12,860 - 00:06:31,590 - «Explain, explain, analyze» — да, я использовал. Там вроде бы, когда мы используем «explain», мы в принципе собираем статистику, ну то есть как бы чтобы нам статистика отобразилась. А с «explain, analyze» мы вроде бы её как раз прогоняем, то есть делаем загрузку всех данных, и, соответственно, у нас как бы

00:06:31,590 - 00:06:44,320 - При первом запуске запрос выполняется дольше с explain-анализом, а второй раз уже будет быстрее, потому что вся необходимая инфа загрузилась. Ага, всё так. А вот если к индексам вернуться, получается

00:06:44,520 - 00:06:46,960 - А какие вообще минусы у индексов?

00:06:47,280 - 00:07:08,110 - Почему на все колонки, например, их не махнуть? Да, самая главная проблема при использовании индексов — это их размер, то есть они занимают некоторую память, вот. Если мы, в принципе, на все столбцы навешаем индексы, это может привести к тому, что у нас очень быстро разрастаться будет по памяти наша таблица. Вот. И второй

00:07:08,110 - 00:07:14,540 - Вторая проблема — это когда у нас очень часто есть запросы на insert, update, delete, то при

00:07:14,620 - 00:07:32,400 - Вот в этих операциях нам нужно, собственно, перестраивать индекс. Нужно, чтобы полностью его перестроить, и чтобы он построился на новых данных. Ну и третье, еще могу сказать, это, условно, когда у нас табличка маленькая, индексы не всегда могут вообще играть роль. Условно, у нас...

00:07:32,500 - 00:07:52,320 - База данных как бы сама понимает, когда использовать план веб… Ну, использовать индекс или нет. Если, условно, таблица маленькая, мы можем просто сиквью-сканом пробежаться по ней, и этого будет достаточно. Угу, прикольно. А давай такую задачку попробуем решить, вот, например.

00:07:52,860 - 00:08:11,180 - У тебя есть приложение, в которое большой трафик, много запросов летит на вставку и на чтение. Вот у тебя табличка из какой-то колонки, да, на вставку в эту колонку и на чтение этой колонки. У тебя задача, вот, поменять

00:08:11,180 - 00:08:20,380 - в этой колонке тип например там со строки на число ну допустим на или наоборот вот расскажи поэтапно как бы ты

00:08:20,660 - 00:08:29,640 - Ты это делал, чтобы, там, пользователи не пострадали, чтобы сохранить вот этот вот… все данные, которые к нам приходят, и чтобы чтение не нарушать?

00:08:29,920 - 00:08:46,090 - Угу. Ну, если, условно, у нас в моменте это надо сделать, то я бы к этой таблице ничего не применял бы, никаких изменений, потому что вдруг, если что-то там, условно, ну, сбой какой-то произойдёт, то могут пострадать данные. А я бы просто создал, условно, аналог этой же таблицы.

00:08:46,090 - 00:09:00,800 - Добавил бы как раз… Заменил бы вот это поле на новый тип и параллельно, условно, какими-то партициями, пачками вносил бы новые туда изменения. А давай вот поконкретнее. Вот, например, у нас вот первый релиз.

00:09:01,260 - 00:09:06,700 - Ты бы что в рамках него делал? А, то, что у нас «Чёрт» — релиз, правильно?

00:09:08,640 - 00:09:12,720 - Ну, я бы по-любому начал бы создавать просто новую таблицу.

00:09:13,020 - 00:09:30,820 - И, так сказать, с новым типом. Тут, наверное, можно колонку просто создать. Ну вот, а, ну можно новую колонку, кстати, да. И пока, если, условно, такая нагрузка, именно в ту колонку добавлять новые данные. Но надо только понять, как их потом джойнить потом. Вот в этом проблема.

00:09:32,420 - 00:09:43,320 - Можно, кстати, если… Ну, по-любому, нам нужно будет как-то их там между собой связывать и понимать, какой они записи конкретно делают, добавляются. Можно было бы добавить какой-то внешний ключ.

00:09:43,720 - 00:09:58,180 - Чтоб потом с помощью join-а их получить и добавить, условно, в новую следующую таблицу, которую мы будем также создавать. Ну, просто колонки в следующий раз добавим больше новых из первой таблицы.

00:09:58,880 - 00:10:05,300 - Ну то есть, получается, с чтением мы ничего, да, не делаем, получается, читаем из старой тоже.

00:10:06,520 - 00:10:09,280 - В первом релизе. И старый.

00:10:11,480 - 00:10:13,080 - Ну, то есть.

00:10:14,060 - 00:10:26,340 - Блин, я бы всё-таки их перенёс бы как-то. Вот если они прям мегаогромные, ты в моменте их не можешь перенести, то есть это простое какое-то будет. То есть ты нарушишь чтение, если ты начнёшь их переносить.

00:10:26,560 - 00:10:42,860 - Ну, я бы тогда добавлял бы в старую, ну, сделал бы чтение из старой таблицы, к примеру, а потом уже, когда у нас всё наладилось, мы уже могли поэтапно, ну, вот как я говорил, пачками старые данные переводить в новые, условно так.

00:10:43,340 - 00:10:56,980 - А, то есть несколько релизов понадобится. Да, да, да, ну потому что разом, да, большой скоп данных, как бы, я не видел за историей, что вот такое используется. Мы вот именно как-то делили партийцами, условно, и их так внедряли.

00:10:59,320 - 00:11:02,920 - У меня, Борис, есть что докинуть?

00:11:05,140 - 00:11:11,100 - Ну, с этим вопросом просто, там, то, что это всё делается за четыре этапа, вот, просто такое.

00:11:11,260 - 00:11:14,920 - Чтобы разъяснить. То есть, сначала мы добавляем табличку, начинаем

00:11:15,020 - 00:11:32,910 - Добавляем новую колонку, начинаем писать. Две колонки сразу же. Следующий у тебя этап — это перенести данные. То есть, тебе надо данные из… когда у тебя уже все реплики, там, мультиреплики сработали, да, у тебя все реплики уже начали писать в новую колонку, ты переносишь данные.

00:11:32,910 - 00:11:36,040 - После этого ты делаешь чтение из новой?

00:11:36,440 - 00:11:37,800 - Колонки.

00:11:39,320 - 00:11:43,300 - А потом, например, старую можно удалить. Либо потом можно

00:11:44,000 - 00:11:50,080 - Старую, ну, старую можно удалить, и потом сделать то же самое, короче, для…

00:11:50,780 - 00:11:54,600 - Типа, переименовай. Ну, вот, такие всякие моменчики, да.

00:11:55,240 - 00:12:10,720 - — А, ладно. То есть, ну, вот так вот по этапам. То есть, мы сначала создаём, начинаем писать, потом переносим данные, потом начинаем читать из неё. Ну и, собственно, уже так можно всё менять. То же самое делается, если нам надо удалить, например,

00:12:11,080 - 00:12:21,280 - «енум», да, в «Пасгаре», потому что «енум» — такая же тема. Там просто так «енум» не удалить. Ну, значение «енума» из типа «енум».

00:12:23,100 - 00:12:34,980 - Так, вопросики, вопросики, вопросики. Да, что-то было, сейчас вспомню, что-то я пока рассказывал, как забыл.

00:12:36,960 - 00:12:41,220 - Ты можешь объяснить, когда вообще нужно использовать GSON, например?

00:12:42,160 - 00:12:52,940 - Т.е. для чего он нужен, когда его можно и когда его нужно использовать, а когда следует создавать, там, таблички отдельно?

00:12:53,400 - 00:13:00,180 - Да, с JSON-ом я работал. У меня был опыт прикручивания OAuth 2 протокола в нашу систему.

00:13:00,180 - 00:13:02,770 - И там нужно было… Ну, допустим,

00:13:02,770 - 00:13:19,480 - Не знаю, «Одноклассники», да, возьмем в пример, конечно. Нужно было, условно, сделать авторизацию с помощью «Одноклассников», и нужно было у конкретного пользователя, кто делает через этот сервис авторизацию, подтянуть его друзей. Собственно, эта опишка возвращала некоторый JSON-формат.

00:13:19,480 - 00:13:25,010 - И вот эти настройки нужно было как раз у нас хранить именно в JSON.

00:13:25,010 - 00:13:26,430 - вот в виде строки

00:13:26,430 - 00:13:28,250 - Это нужно было

00:13:28,250 - 00:13:34,190 - Ну, хранить их потом при получении, условно, происходит там диссерилизация, всё очень быстро

00:13:34,190 - 00:13:42,460 - маппиться к нужным свойствам и, собственно, могли дальше как-то эти данные обрабатывать. Вот. А когда нужно использовать таблички,

00:13:42,960 - 00:13:47,220 - Пример. За место Джейсона, да, именно, в столбца.

00:13:47,760 - 00:14:07,040 - Ну да, вот у тебя есть выбор. У тебя есть какая-то структура данных, и ты можешь делать либо её, вот, например, «Друзья», да, те же. То есть, что тебе мешает создать табличку «Друзья», да, и «Связь» там один ко многим? То есть, у каждого пользователя, ну, есть какие-то друзья, и они хранятся в табличке «Друзей». То есть, чем… Ну, почему нужно было хранить это дело в JSON?

00:14:08,100 - 00:14:23,540 - Тут, наверное, вопрос, вот почему-то больше меня склоняет именно к нормализации и денормализации. А вот, к примеру, если используем термин «денормализация», иногда может быть такая ситуация, что нам скоупом, вот именно в рамках одного запроса, нужно получить опцию.

00:14:23,540 - 00:14:34,880 - в рамках одного запроса именно JSON-ом. Мы его получаем, и его можем на практике как-то использовать. Если мы говорим про обычную таблицу, то это уже нормализация как бы идет,

00:14:34,880 - 00:14:44,300 - А одним скопом запроса мы как бы не можем подтянуть данные. Там, по-любому, у нас будут джойны идти. Скорее всего, речь про это, я так предполагаю.

00:14:44,940 - 00:14:46,760 - Ну, в целом, да.

00:14:46,760 - 00:15:07,030 - Ну, вообще, здесь, если у тебя глубокая структура какая-то, да, то есть ты используешь JSON, потому что тебе придётся создавать, там, десяток табличек, то есть десяток табличек либо JSON, да, с которым ты будешь и так постоянно, там, что-то работать. Просто вычитывайте, пишите данные. Это не какие-то, там, поиски. Делайте какую-то сложную логику на делать. Ну, а ещё, если у тебя изменяющаяся структура данных,

00:15:07,030 - 00:15:26,040 - Это тоже вполне себе возможно, когда, например, пользователь настраивает, у него есть какой-то конструктор, грубо говоря, где он, там, определяет какие-то поля. Тебе с фронта приходит конкретный JSON-чик, да, там, с нужными парами, которые есть, и ты его тоже сохраняешь, потому что ты не знаешь, какая структура от пользователя тебе придёт.

00:15:26,180 - 00:15:38,320 - Вот это тоже такой вариант. Ну и на самом деле, меньше весит, потому что Json, он, действительно, если ты вот эту структуру, там, разобьёшь на сколько-то табличек, то это всё, меньше весит.

00:15:39,600 - 00:15:44,520 - Ладно, у меня с базами, наверное, особо таких вопросов нету.

00:15:47,280 - 00:16:06,460 - Работал ли ты с локами в базе данных? Какие локи знаешь? Так, ещё раз можно? Вот зависла. Да, работал ли ты с локами в базе данных? И какие локи ты знаешь? Ну, для синхронизации. А примитивы синхронизации касаемо БДшки прям?

00:16:09,180 - 00:16:22,220 - С какими локами? Вот что-то из головы прям вылетело. Ну, я слышал про хинты на практике использовал, там, with no lock. А, скорее всего, локи про прям лок на таблицу и лок конкретно на...

00:16:22,220 - 00:16:37,210 - Какую-то строку в рамках таблицы. Вот про это. Ну, да, да. По сути, вот, да, лог на строку, лог на таблицу. Есть ещё один лог, это называется «advisor log», когда ты можешь кастомный лог просто выставить на какое-то, там, значение, и потом

00:16:37,210 - 00:16:39,800 - По этому значению является что-то типа.

00:16:39,840 - 00:16:50,300 - того, что мы используем, например, какой-нибудь кипер. Да, ну в целом — да. Последний не использовал на практике? Прям такого не было.

00:16:52,040 - 00:16:53,560 - А строки приходилось лачить?

00:16:55,440 - 00:16:58,940 - Вроде бы за практику было, я вот такое вспоминаю.

00:16:59,540 - 00:17:11,460 - Селект фор апдейт. Если это речь про это, это же, по сути, тоже… Селект фор апдейт, когда мы выполняем, это вроде бы лог на конкретную строку. То есть в рамках отдельного, допустим…

00:17:11,460 - 00:17:28,920 - Потока мы можем… Ну, вот это тоже с брокерами в сообщении встречается. Как раз я про outbox-паттерн его использовал. Когда select for update дело для записи, мы в рамках как бы одного потока лодчим, вторая, собственно, ничего сделать не может, пока мы не выполним операцию в первой, ну, транзакции, так назовем.

00:17:30,380 - 00:17:45,300 - О, кстати, вот ты задел select for update. Можешь подсказать, как сделать select for update, если ты используешь Entity Framework? Если F-Core, select for update.

00:17:46,920 - 00:17:48,140 - Блин.

00:17:50,360 - 00:18:02,720 - Ладно, не буду тебя томить, просто девушка сонная. Такой вопрос с подвохом. Я просто не встречал такого впервые, прям сталкивался, но надо будет потом посмотреть, интересно.

00:18:07,540 - 00:18:11,400 - Там нет такой функциональности. Там можно её добавить.

00:18:11,740 - 00:18:28,800 - С кучей кода. С кучей кода. Это если ты, там, будешь разбирать вот эти всякие экспрешены и всё остальное, и сам напишешь, короче, какую-то конспирентацию для сервиса, правда, без этого её, как бы, не делается. И просто поэтому берёшь какую-то кастомную SQL и поехали.

00:18:29,580 - 00:18:35,600 - Спасибо. Ладно, давайте погоним дальше, как раз к вот КРМ перейдём.

00:18:36,200 - 00:18:52,480 - Ты тоже расскажи про УРМ, там, с какими работал, какие нравятся, может, какие-то интересные истории. В основном был опыт с эвкором. По большей части где-то два с половиной года только на нём работал. Что касаемо интересных практик, ну, использовал.

00:18:53,360 - 00:18:57,060 - За загрузки данных, не знаю, можно ли сказать, что это интересно было.

00:18:57,240 - 00:19:14,390 - Я, так сказать, разбирался с лишними инклюдами, которые были. Был опыт написания inner join-а, вот тоже я помню task-у, потому что include — это обычно left join. Потом… А, конфигурационные файлы. У нас всё было в рамках метода onModelCreating. Я вносил практику

00:19:14,390 - 00:19:19,140 - Так сказать, распределение по конфигурационным файлам, чтобы всё было очень удобно.

00:19:19,140 - 00:19:21,080 - Вот, что ещё?

00:19:21,440 - 00:19:41,400 - Про чиндж-трекер, ну, тут не скажу, что сильно сталкивался, просто про него слышал, там, S-Mode трекинг по мере необходимости использовал. Даппер был, даппер, вот, но с даппером, честно, ну, больше нравится F-Core, не знаю, как будто больше нравится именно с ним работать, копать, как он под капотом устроен, вот, но с даппером тоже был опыт.

00:19:41,920 - 00:19:43,700 - Небольшой, но был.

00:19:44,940 - 00:19:46,780 - А вот ты же ты говоришь,

00:19:48,640 - 00:19:59,920 - «Include», говоришь, он «inner join». А вот он всегда «inner join»? Не-не-не, я «I», я сказал. Нет, «include» — он «left join». Когда нам нужно сделать «inner join»?

00:20:01,100 - 00:20:19,360 - Вот, кстати, если... Ну, мы можем сами написать, к примеру, select main и group by, вроде бы комбинация, и тогда у нас получится inner join. Либо использовать методы операторов, конкретно этот, SQL-запрос, там, throw, inner написать, если я не ошибаюсь.

00:20:19,680 - 00:20:28,680 - На самом деле, include — довольно-таки интересная операция. У неё есть такая штука, как «зависит», если смотрел, там, «Акиншина».

00:20:29,020 - 00:20:43,600 - Да, у него зависит от того, какой 4NK. Если 4NK неналабл, тогда он тебе сделает inner join, а если он налабл, то он тебе сделает left join. Вроде как-то так он работает. Я уже сам давно не помню, но что-то я такое разбирался, по-моему, было так.

00:20:44,020 - 00:20:50,920 - Надо почитать. Я просто слышал, что он только left-joined всегда, и поэтому мы select many пишем. Вот, ну, почитаю про это.

00:20:51,940 - 00:21:09,440 - Вот у него тоже я тоже когда-то сталкивался с такими шутками. Я такой, а почему у меня значения не возвращаются? Потом посмотрел, или там много возвращаются. Вот, это такая хитрая штука, да. У них логика сделана на аналогу. По-моему, ну, в шестой версии это было. Возможно, в других версиях это не было.

00:21:09,860 - 00:21:17,060 - Вот. Ещё ты говорил что-то про какие-то… Про загрузку данных.

00:21:17,360 - 00:21:33,940 - Можешь рассказать поподробнее, что за загрузка дана? Да, у нас в рамках Эвкора, чтобы получить, условно, навигационные свойства, есть три подхода. Это ленивая загрузка, когда мы просто пишем этот include, и он прямо в рамках одного запроса подгружает зависимые данные.

00:21:33,940 - 00:21:42,420 - Бывает полезно, когда нам сразу нужны данные, но когда мы их не используем, навигационные, это может только нам ухудшить по производительности, потому что время тратится.

00:21:42,420 - 00:21:50,140 - На join. Вот. Это ленивое. Явное — с помощью обращения к контексту и вызова метод entry.

00:21:50,140 - 00:21:59,910 - Ну, вызываем метод «Entry», прокидываем сущность и в зависимости от дальшей цепочки методов подгружаем либо один объект, либо коллекцию объектов. Вот.

00:21:59,910 - 00:22:14,330 - Что у нас? «Ленивая»? «Ленивая» вроде бы с помощью прокси. Там библиотека отдельно ставится, как я помню. Прокси, как там, «Entity Framework Core», и там она построена на виртуальных свойств. Вся

00:22:14,330 - 00:22:23,160 - Роль в том, что мы, к примеру, когда подгружаем основную сущность, и потом обращаемся к зависимой, она у нас только при обращении подгружается.

00:22:24,080 - 00:22:31,760 - Вот, как-то так, про загрузки.

00:22:34,920 - 00:22:38,740 - А, ну я в целом хотел про Change Tracker узнать.

00:22:39,060 - 00:22:40,400 - Так, сколько?

00:22:40,580 - 00:22:46,070 - А ты в целом, вот, ну, понимаешь примерно, что он там делает под капотом? На что он вообще повли… влиять может?

00:22:46,070 - 00:23:06,500 - Да, я прям в рамках компании делал, ну, курс по EF Core и разбирал эту тему. Там прикол в том, что когда мы, допустим, получаем данные с бдшки, вот с помощью EF Core, у нас как бы под капотом создается слепок изменений. И если мы в моменте что-то редактируем и вызываем этот self-change, у нас проверяются изменения,

00:23:06,960 - 00:23:10,220 - А вот конкретно какая запись изменилась, она

00:23:10,220 - 00:23:24,160 - Выставляется ей определенный статус, там, «Modified», и она будет изменяться. Если вообще никаких изменений не было, то у нас вроде бы статус выставляется «Unchanged», то есть никак не изменено, и…

00:23:24,160 - 00:23:27,350 - Соответственно, сохранений никаких не будет происходить.

00:23:27,350 - 00:23:40,580 - Статусов может быть несколько — «Modified», «Edit», «Deleted», вроде бы, или «Remove» называется. Вот. А этот Change Tracker, он не всегда бывает, так сказать, нужен при, допустим, Get-операциях, когда нам ничего не надо изменять.

00:23:40,580 - 00:23:50,840 - Мы можем его отключить, вызвать метод «As Now Tracking», и Change Tracker будет у нас, ну, disabled, условно, выключен. Как-то так.

00:23:51,020 - 00:23:53,040 - но в объекте, как вы настраивали.

00:23:53,480 - 00:23:57,380 - А может, диагонально его настраивали, вот, относительно запроса?

00:23:57,380 - 00:24:16,840 - Нет, у нас он просто был по мере необходимости запросов. Вот, то есть, в моменте я что-то пишу там, и вот вызывал S-Node Tracking. Да, понятно, что, ну, это, наверное, нехорошая практика. Скорее всего, проще написать какую-то кастомную для Get-операций, которой не надо ничего менять, и вызывать его. Но вот у нас в моменте запроса выписали его, вызывали.

00:24:17,160 - 00:24:34,760 - А там вот глобальные у него настройки есть, и там вот есть «No tracking with identity resolution». Вот туда не копал, не смотрел, что это такое. Вот, кстати, разбирал, да. Там два метода у нас есть, но разницу я уже сейчас не назову, потому что когда я разбирался, у меня голова кипела, сейчас точно не вспомню.

00:24:35,800 - 00:24:54,560 - ну там он в общем трекает только одесшники сущности чтобы не было дублей там при дикартова перемножение вот это вот все такая настройка баланс а в целом а спид куэри вот такие штуки юзал приходилось

00:24:55,040 - 00:24:58,320 - А сплитквери — это вроде бы разбиение…

00:24:59,280 - 00:25:17,080 - Или нет, сплит-квери, сейчас я вспомню. У меня было на практике, когда нужно было получить, допустим, N записей из первой таблицы, поставить между ними тире и подгрузить данные из второй таблицы. Вот это, наверное, про сплит-квери. Что-то такое вспоминаю.

00:25:17,660 - 00:25:23,060 - Ну да, он там все джойны выворачивает в отдельные селекты, то есть

00:25:23,740 - 00:25:27,780 - Это полезно, когда у тебя Декартова перемножение

00:25:28,520 - 00:25:35,960 - Из-за джойнов происходит, вот, взрыв данных. Распухает вот это решение, как вариант.

00:25:36,720 - 00:25:41,380 - Через отдельные. То есть он три запроса просто делает. Ну, примерно, да.

00:25:41,760 - 00:25:43,740 - А миграции как у вас?

00:25:44,120 - 00:26:03,740 - Ты пользовались EF Core, а миграции как были устроены? Тоже через EF? Да, миграции были. Ну, у нас от проекта к проекту были свои приколы, а на проекте, которым я работал, то есть, у нас было всё через миграции. Там, апдейт базы, миграция, удаление, там, таблицы какой-то или колонки — тоже была миграция, собственно.

00:26:04,340 - 00:26:11,440 - А, ну, ещё у нас был прикол через отдельные скейл-файлы. То есть, у нас и была папочка «Migrations», там

00:26:11,600 - 00:26:30,520 - А дате с расширением SQL были свои просто SQL-запросы, которые мы уже выполняли в BD-шке. Ну, вот два подхода было. Ну, то есть, с листами EF, вот, в конечном итоге, да, вы остановились на миграциях через EF, правильно? Да, да, да.

00:26:31,280 - 00:26:35,880 - А вот как вы их менеджили? То есть, как деплоили, например, приложение?

00:26:35,880 - 00:26:55,530 - Вот этот флоу был как устроен? Преинграцию, вы хотите? Да, честно, я прям глубоко не вкопал в них, но видел, что у нас были несколько схем базовых данных, и в зависимости от той схемы, которую мы используем, там, допустим, у нас была паблик, условно, базовая, и какая-то для тестов своя, с определёнными колонками.

00:26:55,530 - 00:27:01,290 - Ну, накатывали, условно указывали схему, и к этой схеме БДшки делали просто...

00:27:01,290 - 00:27:02,450 - Релиз.

00:27:02,450 - 00:27:20,220 - Вот, как конкретно внутренне сильно не копал. Мне просто последний опыт с миграциями был небольшой, поэтому вот что успел посмотреть, то посмотрел. Ага, то есть мне вот просто понять, вот при релизе именно вот миграция в какой момент накатывалась у вас?

00:27:20,380 - 00:27:37,080 - При релизе приложения на прот, например. Ну, мы сперва как бы миграцию загружаем, смотрим, что, типа, никаких поломок не было, и потом у нас уже был релиз самого ПО, ну, приложения обновы. А вы это как-то отдельно руками, да, миграцию накатывали? Да, отдельно руками.

00:27:37,980 - 00:27:48,600 - То есть, ну, пайплайны — у нас такого не было, это как отдельно для миграций. По сути, был просто CICD, ну, там опыта с базами не было вообще никакого, чисто CICD TeamCity-ов.

00:27:50,260 - 00:28:09,700 - Угу, то есть, миграция обратно совместимая, да, её можно заранее, там, выкатить и потом катить приложение, условно, вот, плюс-минус, так, да? Да, но я не знаю, какие могут быть в этом проблемы, потому что, опять, использовал, ну, небольшой опыт с ними был, вот, и вот чисто вот такая практика была.

00:28:11,280 - 00:28:12,580 - Угу, окей.

00:28:13,240 - 00:28:24,120 - А ещё такой вопрос про внутренний пул подключений к базе. Вот, у РЭМ, например, RedBug SQL представляет свой внутренний пул, и вот

00:28:24,420 - 00:28:37,400 - Как он у вас был настроен на проекте? Знаешь ли ты, как его, там, поднастроить можно? Сталкивались, может, с проблемой выжирания коннектов к базе, например, под большой нагрузкой, или что-то в такие моменты?

00:28:37,880 - 00:28:46,050 - Я видел, что у нас вперва использовался на проекте просто AddDbContext, а потом AddDbContectionPool, то есть, да, пул контекстов.

00:28:46,050 - 00:29:04,820 - С саму настройку я прямо в рамках работы не сталкивался, то есть у меня не было таких задач, но, как я помню, там были свойства Max Size Pool, вроде бы максимальное количество полов одновременных на коннекшены, и сам я с ним игрался чисто вот на практике, ну, в своём подпроекте смотрел работу внутреннюю.

00:29:04,860 - 00:29:08,080 - За какие-то более внутренние тонкости

00:29:09,440 - 00:29:15,000 - Блин, ну, честно, вот я помню, Max Size Pool был свойством, максимальное количество подключений.

00:29:15,840 - 00:29:32,880 - Тот в connection строке, да, ты имеешь в виду? Просто db.context.pool же ещё у EF-а есть просто сущности, и db.context.pool их пулит, чтобы не генерить много. Ну да, я, скорее всего, про… Ну вот, когда мы подключаем adb.context, у нас там использовался метод adb.context.pool.

00:29:33,680 - 00:29:42,160 - Вот это именно про сущность, скорее, а тут вот про подключение к БД именно. А, про подключение… То есть, как у нас был выстроен подключение к БДшке?

00:29:43,360 - 00:29:54,780 - Ну, по сути, просто вытаскивание её из Upsetting.json-а и прокидывание в метод, там, usePG… useNPGSQL connection, там, вроде бы так назывался.

00:29:54,900 - 00:30:03,700 - И вот в этой строке настраивается ещё внутренний пул подключения самих. Ты вот… Вы его там не конфигурили, да? То есть он у вас включен, выключен был.

00:30:03,700 - 00:30:17,690 - Честно, не скажу. Не помню этого. Ну окей. А в целом сталкивался, например, с ПГ-баунсером, с балансировщиками, подключением их к базе, вот каких-то аспектов работы с ними?

00:30:17,690 - 00:30:21,080 - А когда, условно, только был джуном?

00:30:21,080 - 00:30:28,020 - То такие задачи у нас были очень часто из-за нагрузки. Но опять, там у нас решали их задачи уже сеньоры.

00:30:28,020 - 00:30:35,540 - До меня прям как-то балансировщик не доходил. Я в курсе, что с помощью него можно распределять нагрузку, междуусловно.

00:30:35,540 - 00:30:50,500 - Серверами, и понимать, что в какой момент там, условно, отвалилось. Но на практике взаимодействия такого не было. Окей, у меня вроде всё, Борис, есть что добавить у тебя?

00:30:51,700 - 00:31:04,920 - Ну, если не сталкивался с… глубоко не работал с ТГ-баунсером, то, наверное, нет. А к ТГ-баунсеру были вопросики. Ну ладно, это не страшно.

00:31:05,520 - 00:31:23,640 - Давайте дальше про брокеры поговорим. Тоже начни, расскажи, с какими брокерами имел дело, очередями. Да, в основном из брокеров у нас был Rabbit и было немножко Kafka. Вот, именно больше опыта было с Rabbit.

00:31:24,020 - 00:31:28,480 - А конкретно рассказать прям, что из себя представляет, или как?

00:31:30,700 - 00:31:50,960 - Ну, если хочешь, можно так, поверхностно. Да, ну, по сути, был опыт с Revit-ом, и задача у меня была, я помню точно, настройка эксченджев, прикручивания к очередям. Это когда только я вникал в Revit. Вот, был немножко… Смотрел на практике, как разворачивается аутбокс-паттерн.

00:31:51,160 - 00:32:02,410 - Это прямо в продакшене у нас было. Пробовал свои примеры посмотреть, тоже, как он реализовывается на своей практике. Что, по сути, в рамках Рэббита у нас есть экчейндж, есть очередь.

00:32:02,410 - 00:32:21,660 - Exchange связывается с очередью по routing.k. Если рассматривать, допустим, Kafka, то там схема работы немножко другая. Есть продюсер, есть, по сути, сама очередь. В рамках очереди есть топики, в рамках топиков могут быть партишены. И, по сути, из очереди сразу сообщение доставляется в продюсер.

00:32:21,940 - 00:32:41,960 - Вот. Как-то так, по сути. Про Outbox транзакшн тоже слышал, что он очень часто встречается для гарантии доставки сообщений. Потом, про гарантию доставки сообщений — это «at most once», то есть хотя бы один раз, как максимум, как минимум один раз,. At least once. И ещё какой там?

00:32:42,360 - 00:32:58,100 - Точно один раз. Вот я помню, что он очень такой… Его сложно реализовать на практике. Exactly once, да-да-да. Вот, что его на практике, ну, очень прям сложно реализовать, и практически невозможно. Ну, вот такие темы слышал про них. А у вас, ну, то есть…

00:32:58,880 - 00:33:03,170 - Как настроена, какая гарантия была поддержана брокером вашим?

00:33:03,170 - 00:33:21,430 - А тут, в зависимости от использования, допустим, у нас очень… Ну, у нас была своя система логов, то есть, вообще, самописный фреймворк. Нужно было собирать аналитику по всяким ивентам, по всяким подключенным сессиям. Здесь, когда у нас использовались логи, то при…

00:33:21,430 - 00:33:28,410 - Этой гарантии, ну, при этом функционале у нас была гарантия как максимум один раз.

00:33:28,410 - 00:33:32,760 - Во, то есть, ну, нам неважно было, потеряется сообщение в моменте или нет.

00:33:33,220 - 00:33:48,020 - Когда это уже были какие-то сложные, допустим, ну, в рамках микросервисов операции, то там уже использовался подход как минимум один раз, потому что потерять какое-то сообщение было очень, ну, так сказать, трудно.

00:33:48,500 - 00:33:50,740 - Плохо, вот, как-то так.

00:33:51,840 - 00:34:06,040 - Окей, а вот если как минимум один раз гарантия доставки, вот как защититься от дублей сообщений на принимающей стороне? Вот если у тебя есть какой-то хендлер, вот как бы ты защищался?

00:34:07,500 - 00:34:08,900 - От дублей.

00:34:10,460 - 00:34:19,720 - Да, в этой как раз проблеме как минимум один раз у нас может быть дубли. Там вообще, я смотрел, четыре ситуации могут быть. Ну вот, одна из них — это дубли. Допустим...

00:34:20,160 - 00:34:24,770 - Ну самый такой банальный механизм — это по какому-нибудь айдишнику

00:34:24,770 - 00:34:27,440 - в сообщении его прокидывать, и чтобы была

00:34:27,440 - 00:34:47,830 - Между ними разница. Там сообщение и конкретно идишник, допустим, пользователя уникальный. Вот. Либо если у нас event-сорсинг-архитектура по, допустим, ивенту. То есть, по ивенту уникальному мы могли в сообщение его прокидывать и его так идентифицировать, по сути, наше сообщение.

00:34:47,830 - 00:34:48,940 - Я бы…

00:34:49,000 - 00:34:55,200 - В Хендлере что происходит? То есть приходит сообщение, и дальше там мы берём айдишникова?

00:34:55,260 - 00:35:03,980 - Да, ну, допустим, мы берём айдишник и, условно, его вместе с общением прокидываем. Там, допустим, структура была бы у нас какая-то. Вот.

00:35:04,240 - 00:35:16,580 - Ну, и по вот этому айдишнику мы бы различали запись. А есть ли уже такая? То есть, была бы уже она обработана или нет? Если нет, то мы её оттолкаем дальше. Если да, то в целом ничего с ней не делаем.

00:35:19,520 - 00:35:26,060 - А как мы понимаем, что она уже была? Как мы можем понять? Ну, допустим…

00:35:27,640 - 00:35:46,460 - Допустим, да, если используя, ну, если в рамках аутбокса, допустим, могу сказать, если мы вот его применяем, там для столбца можем навесить статус, либо in processing, допустим, done, либо там cancelled. Если done, то, по сути, это сообщение с этим id-шником, оно уже было обработано.

00:35:46,780 - 00:35:56,020 - Вот. Ну то есть, словно исключаем Outbox, просто вот делаем такую табличку и храним там айдишники. Вот так, имя ж, веду.

00:35:56,080 - 00:36:10,760 - Ну если исключаем Outbox, то в какой-нибудь табличке, да, но тут тоже должна быть гарантия, что в процессе при записи в таблицу у нас не крашнется там сервер. Ну в рамках транзакции, условно, если это делать, то я думаю проблем сильных не должно быть.

00:36:14,460 - 00:36:23,700 - Окей, да, похоже на правду. А вот если чуть усложнить сценарий, например, у тебя какой-то хендлер принимает

00:36:24,360 - 00:36:40,500 - Сообщение с обновлением, да, апдейтом каким-то. И вот тебе прилетело два разных сообщения с двумя разными апдейтами, там, одного и того же поля, например. Но они по пути в брокере, там, перепутались. И тебе первым пришло

00:36:40,800 - 00:36:59,740 - Новое, а потом пришло устаревшее, и как бы оно перетрёт актуальное значение, то есть, данных. То есть, а стейт нарушится. Вот. Как от этого можно защититься? Так, структура. То есть, у нас два сообщения, и последнее перезатирает первое.

00:37:01,320 - 00:37:18,840 - Сейчас надо подумать, как можно. Ну, мне почему-то на голову приходит, допустим, добавить какое-то свойство по времени, по дате создания, к примеру. Либо по дате обновления. Нет, по дате создания или обновления приходит в голову. Только надо подумать, как их проверять.

00:37:20,640 - 00:37:28,040 - Ну, скорее всего, по времени. То есть, если у нас последнее обновление было, допустим,

00:37:29,200 - 00:37:33,440 - Блин, вопрос. Ещё раз, можно вопрос? Я вот… Есть два сообщения.

00:37:33,460 - 00:37:47,840 - Они, по сути, первая обновляет, и вторая перезатирает значение. А у нас вообще такая струк… Ну, должна быть такая логика, что вот перезатирать? Ну, допустим, вот у нас есть такая логика. Например, у нас там… Ты пришёл на работу, у тебя там

00:37:47,840 - 00:38:06,600 - миллион обработчиков, миллион всяких сообщений, и вот есть проблема, что сообщения могут перепутаться по пути к тебе, и они могут что-то заэффектить этим. То есть, типа, устаревшее сообщение в конце, оно может как-то, там, нарушить стейт. И вот надо эту проблему решить.

00:38:08,000 - 00:38:17,260 - Ну если оно устаревшее, то по дате создания условно как-то можно. Сперва мы берём самые первые, которые шли, обрабатываем их как-то...

00:38:17,260 - 00:38:38,180 - А устаревшие, ну, надо понять, нужно ли им… Смотри, может, не можем брать? Вот к нам приходят сообщения, вот мы же там можем их проверять. Ты вот правильно мыслишь, вот, если вот некритизировать, что конкретно мы можем делать? Да, дату создания, по сути. То есть когда сообщение было было, условно, создано. Да, например, вот, что мы делаем.

00:38:42,040 - 00:38:50,580 - Ну, можно навесить какой-нибудь таймер, допустим, через каждые 30 минут мы делаем обновление нашей записи.

00:38:52,300 - 00:38:56,340 - Не знаю, ну какое-то решение. Ну вот по дате.

00:38:56,880 - 00:39:02,300 - сейчас я хочу всех сообщение есть там какой-то свой там timestamp да когда они были созданы

00:39:02,580 - 00:39:21,030 - Мы айдишники уже проверяем. А, ну, так, ну, можно как? Можно применить здесь реплицирование, допустим, новые записи мы их обрабатываем, а старые мы их, допустим, просто кладём в табличку, что они, условно, были на обновлении, и никак их не трогаем, если они никак не нужны нам.

00:39:21,030 - 00:39:38,980 - Ну, помечаем, условно, статус можно для них добавить, да? Как мы помним, что они старые? А, ну, вот в голову сейчас пришло. Допустим, у вас даты есть и статус. Статус сообщения. По статусу мы, к примеру, можем для сообщения, которое вот сейчас в моменте выполняется, повесить статус «In Processing».

00:39:38,980 - 00:39:58,230 - А когда оно выполнится, мы, допустим, этот статус выдвинем в дан. И если в дан у нас успешное выполнение, то старые сообщения мы как бы, ну, не трогаем. Зачем, если у нас уже произошла запись, апдейт? Ну да, то есть мы просто их отбрасываем, они не нужны, потому что у нас более актуальный апдейт уже применен.

00:39:58,230 - 00:40:09,710 - Ну да, я, кстати, вот читал на Хабре, допустим, старые сообщения, которые уже даже не нужны, их не удаляют, а вроде бы делают репликацию и помещают, условно, в табличку, где хранятся вот…

00:40:09,710 - 00:40:12,700 - Неактуальное сообщение. Ну чтобы вот поддерживать

00:40:12,780 - 00:40:14,660 - информацию, структуру.

00:40:15,100 - 00:40:17,400 - Ну это уже, да, это уже детальки.

00:40:19,280 - 00:40:30,200 - А вот такой вопрос. Это всегда подойдёт, вот такое, чтобы одно сообщение обработали, а другое откинули, и всё, оно пошло по-ичь? Типа, ну, последнее обработали, а первое не обработали.

00:40:31,340 - 00:40:39,420 - Ну если… Нет, это не всегда, наверное. Если у нас есть какая-то зависимость конкретная, то, допустим, между микроси…

00:40:40,140 - 00:40:43,960 - А вот смотри, давай, давай такой пример просто приведу.

00:40:44,500 - 00:40:49,440 - Вот ты работаешь сейчас в финансовой организации, да?

00:40:49,440 - 00:41:06,940 - У тебя приходит сообщение одно — это пополнение баланса, другое — списание баланса, да? То есть, человек пополнил на 1000 рублей и хочет что-то оплатить, какую-то покупку на 1000 рублей. На счету у него сейчас нот. Может ли он сделать покупку на 500 рублей, если он, ну, типа, сообщение о покупке придет перед пополнением?

00:41:07,320 - 00:41:25,980 - Ну нет, здесь не всегда. Здесь, скорее всего, должна быть гарантия согласованности данных, один из принципов ASID. Вот. И такая структура, ну, не всегда пройдет. Тут, скорее всего, именно очередь, важна очередь сообщений, в каком порядке они.

00:41:26,820 - 00:41:46,180 - Вот, я предполагаю, так. Да. То есть, ну, тут вот тот подход подойдёт для некоторых данных, для такого надо будет думать что-то другое. Например, ну, я не знаю, идти, чтобы ты изучил сообщение, как-то типа… Типа, от которого идёт, чтобы он был валидным, чтобы посмотреть состояние на момент какого-то конкретного сообщения.

00:41:46,920 - 00:41:50,060 - Угу. Ну это да, это уже в процессе обдумываешь.

00:41:50,140 - 00:41:57,220 - Даш, да, ну, это такое хитрое тоже момент, но, я думаю, у вас как-то это же реализовано?

00:41:57,220 - 00:42:07,190 - Ну, у нас транзакции в основном были. То есть, в рамках транзакций. И понятно, что

00:42:07,190 - 00:42:26,140 - Просто проверялись балансы, условно. А доставка сообщений, она у нас была… всегда в порядке выполнялась. То есть, Даш, ну, нельзя было сделать так, чтобы, допустим, у пользователя списать деньги, а у него их там нет. Важен, типа, был порядок сообщений, доставки. Да, видишь, тут просто…

00:42:26,900 - 00:42:31,280 - Порядок сообщений — это, конечно, хорошо.

00:42:31,660 - 00:42:41,510 - В монолите он будет соблюдаться, но если ты работаешь с микросервисами, то ты не знаешь, какая реплика у тебя, какое сообщение. Вычитает первое, начнет его первое обрабатывать.

00:42:41,510 - 00:42:55,000 - Одна реплика может вычитать сообщение, например, о списании, а другая — о пополнении, да. И реплика о списании начнёт работать чуть-чуть быстрее, потому что, я не знаю, у неё сервер чуть-чуть… ну, машинка чуть-чуть быстрее, а у той такая не будет.

00:42:55,000 - 00:43:05,120 - В другом «ЦОДе» стоите, там, большой пинк, чтобы она вычитала само сообщение, да. То есть, здесь вот такие моментики бывают тоже, конечно, хитренькие. Ну, это ладно.

00:43:06,060 - 00:43:07,220 - Чао, бэш!

00:43:08,140 - 00:43:12,960 - Погнали, давай про C-sharp в целом, про.NET поговорим.

00:43:13,120 - 00:43:14,740 - Вот расскажи.

00:43:14,840 - 00:43:22,900 - С какими версиями работал.NET-а? Следишь ли за обновлениями? Может, какие-то блоги читаешь, подкасты, чатики профессиональные?

00:43:22,940 - 00:43:36,650 - Да, у меня опыт был с.NET Framework 4.8. Честно, не понравилось. Вот, там было прям сложно. А потом был опыт с.NET 5 и по.NET 9. По сути, все новомодные фичи, которые выходят

00:43:36,650 - 00:43:45,640 - Ну, понятно, что они не все используются, но про них стараюсь читать, как только происходит, там, апдейт в сентябре, в ноябре. Вот, стараюсь читать.

00:43:45,840 - 00:43:59,440 - К примеру, классные штуки, которые у нас очень зашли, это рекорды в.NET 5-ом появились. Потом я слышал, прикольная штука — дом, ну, дерево выражений появилось у нас в.NET-е.

00:43:59,740 - 00:44:19,560 - Frozen Dictionary, я помню, у нас архитектор про это рассказывал. Как под капотом, честно, ну, не расскажу прям детально, но смотрел. Что ещё прикольного? Ну, там, сериализация, то есть, вот, System Tech JSON, сериализация, децерилизация улучшалась по времени.

00:44:19,840 - 00:44:30,400 - А вот, кого смотрю прям тоже, ну, Анг… Джаванович, смотрю его, у него очень классный, вроде так правильно произв… его имя, классный

00:44:30,700 - 00:44:33,940 - Классный контент из русскоязычных

00:44:34,380 - 00:44:50,800 - Ну, «Программысли» бывает тоже, чекаю. Ну, у него в последнее время такие видосы. Потом «Экскалип» смотрю, там. Его собеседования какие-то тоже прикольные. Вот. Ну, и сам, плюс, в рамках компании тоже старался делать обучающий контент.

00:44:52,140 - 00:45:00,200 - А ты вот сказал, тут Net9, он ещё, получается, не вышел. Ты вот превьюшку какую-то использовал? Или ты имел в виду тут Net8?

00:45:00,620 - 00:45:18,920 - По.NET 9 я посмотрел, там приколы вышли с джейсонами, какие-то отступы появились, которые в работе могут пригодиться. Я просто посмотрел там.NET.NEXT.RU, что ли, вроде бы, что-то, подкасты. Вот посмотрел, послушал их.

00:45:19,220 - 00:45:38,960 - Я думал, вы уже использовали, там, тут на 9. Не-не, ну то есть у нас как бы… А когда апдейт случается, мы не сразу на него переходим, потому что могут быть разные последствия. Через год, условно, или полгода, там, когда стабильная выйдет версия. А вы на LTS-версиях сидите, или, там, 7-й, 9-й, может?

00:45:39,140 - 00:45:59,300 - Ну, у нас, к примеру, проекты были даже на.NET 6-ом, то есть мы как бы… Ну, 6-ой LTS всё-таки. Да-да-да, мы как бы не сразу делали апдейты, потому что, ну, не всегда это может сыграть роль. Вот я помню, не знаю, честно, за Дотнет 6-й вроде бы, но...

00:45:59,560 - 00:46:08,100 - Не все проекты мы на него переводили. Могу точно… Ну, могу спутать версию, «Дотент 5-й» или «Дотент 6-й», но какая-то версия была

00:46:08,480 - 00:46:12,500 - Бак, ну, проблемный, и мы не все проекты переводили сразу.

00:46:13,580 - 00:46:30,500 - А вы вот на Дутнет 7-й переезжали в целом? Дутнет 7-й? Нет, у нас 6-й и 8-й. 7-м я уже сам на практике… Ну, стараюсь сам на практике что-то пробовать, смотреть. Там вот в 7-м, вот в EF-е, много всяких прикольных фишек появилось.

00:46:31,500 - 00:46:36,140 - Батчами обновление, батчами удаление. Что давно просили?

00:46:36,140 - 00:46:55,600 - Ага, оно… Вот execute, delete — вот это вот. Да, у нас бачи… У нас была вроде бы самописная даже технология. Да, вроде даже самописная. Мы выполняли, если большой скоуп обновлений, у нас использовался bulk update метод, вот. Под капотом, честно, не могу сказать, как он там работал, не особо вдавался.

00:46:55,700 - 00:47:05,340 - За AFCore смотрел, там прикольный агрегейт, что-то по агрегации методы ускорились, добавились даже какие-то в новой версии. Вот.

00:47:06,220 - 00:47:24,500 - А у вас самописный был bulk update или от EF Extensions, который… классная шучечка. Вот именно что он был самописным, как я вспоминаю. Там прям был самописный отдельно либо побольше. У нас очень много было самописного по NFT Framework Core. Там даже были Expression 3.

00:47:24,660 - 00:47:39,920 - Большая логика выражения, ну, большая логика по выражениям, вот. Но не совсем удалось поработать. А тебе… То есть, ты не работал сам с Expression 3, чтобы что-то доработать, там, ну, делать какой-то свой вывоз какого-то текста, там,

00:47:42,620 - 00:48:00,770 - На практике я посмотрел, то есть, поковырялся немного, понял, что, ну, это довольно непростая тема на самом деле, вот. Просто пару примеров с метанита посмотрел, а в работе, так сказать, мы их не часто прям писали. Они были, но не прям must-have-практика была у нас.

00:48:00,770 - 00:48:12,740 - Ну да, сложная штука на самом деле. Там вот этот паттерн-визитор, и тебе приходится декларативно всё вот это просматривать, а как построить, там, какое-то предложение, там, ещё что-то из него создать.

00:48:12,740 - 00:48:24,610 - Довольно-таки хитрая штука. Согласен. Да, ну она… По оптимизации, можно, наверное, выиграть. Я не спорю, если её добавили. Но она очень, там… Бывали такие методы, что просто посмотришь и застрелишься, условно.

00:48:24,610 - 00:48:40,020 - Да, да. Там построение текста, то есть ты это всё делаешь, как вот эти машиносостояния, всё остальное. То есть ты через него примерно смотришь, как чего куда-нибудь добавлять. Я когда-то делал такую штуку для вывода логов, по-моему, структурных, для

00:48:40,620 - 00:48:51,740 - Забавная практика. На универе редко встали. Так, автопчик небольшой был. Давайте дальше.

00:48:52,500 - 00:49:06,470 - Слушай, вот ты наверняка, там, на SpendCore писал часто какие-то интеграции, больше ничего не делал. Вот, там, в SpendCore есть такая школа, как Http Client Factory. Вот, может, знаешь, зачем её добавили, какие она проблемы решает?

00:49:06,470 - 00:49:24,040 - Пользуйтесь, если вы вообще… Да, знаю несколько проблем. Одна из них — это, ну, тоже критичная. Допустим, когда мы создаём новый HTTP-клайент, у нас может быть закончится свободный пул подключений. При HTTP-клайент, вроде там в факторе даже прописан, он как бы за нас…

00:49:24,880 - 00:49:38,560 - Занас под капотом занимается созданием и, там, условно, удалением, как фредпул, условно, вот этих http-клайентов. Это первая проблема. Вторая проблема, я слышал, связана с ДНС-ами, что

00:49:38,560 - 00:49:51,970 - Честно, сейчас прям детально не расскажу, потому что давно сталкивался, ну, читал про это, но знаю, что может быть проблема с ДНС-ами, когда у нас закончатся полы подключений, и может случиться такая проблема, что собьется ДНС.

00:49:51,970 - 00:49:53,100 - Вот.

00:49:53,880 - 00:50:13,080 - Да, то есть он, скорее, там, не успевает следить за обновлением DNS, если какой-то глобальный Ruby-клиент задан, вот. И когда пул у нас тут к этому толерантнее. То есть у тебя где-то далеко IP-шник поменялся, а он думает, что это ещё старый IP-шник, и, там, ошибки сыпет.

00:50:13,840 - 00:50:16,920 - Ну да, всё правильно. Окей.

00:50:17,880 - 00:50:24,880 - А про строки немного поговорим. Вот тут на эти строки они мутабельны. Какие у тебя там, видишь, плюсы и минусы?

00:50:25,620 - 00:50:45,600 - Ну, минусы в том, что, условно, допустим, в форриче изменять строку — это прям выстрел в ногу, что, допустим, каждое изменение строки приводит к тому, что у нас создается новая строка. А какие минусы… Ой, ну вот это минусы, да. То есть, по производительности, плюсы.

00:50:46,000 - 00:50:47,740 - Плюсы строк.

00:50:48,920 - 00:51:02,420 - Ну, не знаю, можно ли назвать, что, допустим, если у нас есть одна строка с одинаковым названием, то она ссылается на один участок памяти. Насколько это применимо? Ну, вот, не знаю, могу предположить, что это плюс. Плюс строки.

00:51:04,140 - 00:51:14,520 - Ну, я знаю, что строки — это ссылочный тип данных, но их поведение, оно как у значимого типа данных. То есть, если мы её передаём в метод, то у нас получается копирование строки.

00:51:15,260 - 00:51:25,620 - Вот. Что ещё? Ну, стринк-мод, слышал? Ну, в целом, нет, она всё-таки… она всё-таки ссылочный тип. То есть, она… у неё кодирование не будет, просто она снаружи выглядит как…

00:51:27,180 - 00:51:32,240 - По значению, но копирование строки мне будет, простите. Сейчас по ссылке оно передаётся.

00:51:32,620 - 00:51:51,710 - Но ты её поменять не можешь. Соответственно, у тебя значение, оно всегда будет постоянное, в итоге. Вот. Я больше, да, имел в виду про то, что её извне, как бы, изменения мы не затронем. Вот. Что ещё про строки? Ну, про стрингбилдер. Слышал, что это классная штука, когда, условно, нужно делать много, там, контентации со строками. Про…

00:51:51,710 - 00:51:56,320 - Стрингпул слышал тоже. Ну это вот как раз, что у нас, если…

00:51:56,340 - 00:52:04,240 - Да, интернирование, что если у нас есть, допустим, три строки, и у них одинаковые значения, то они будут ссылаться на единый участок памяти.

00:52:04,960 - 00:52:19,280 - То есть всегда, имеешь в виду, любые три одинаковые строки? Ну вот, я на практике пробовал с этим поиграться. Допустим, создал перемену x, y. Если у них значение, там, три однёрки, то у них getHashCode, он будет равный.

00:52:20,140 - 00:52:35,440 - А вот в памяти они на одну область будут ссылаться или нет? А, я разбирался с этим. Ну то есть, у нас при первом создании строки… Хотя, хороший вопрос. Тут, скорее всего, зачем нам и интернирование, стринг-пул?

00:52:36,400 - 00:52:49,620 - Ну вот да, то есть. Да, да. Ну, как я смотрел, то у нас просто будет ссылание на одну ячейку в памяти. Может быть, это сделано с помощью интернирования строк? Под капотом как-то реализовано хитро.

00:52:50,520 - 00:53:03,500 - могу предположить так. Ну, вот эту проблему, скорее, интернирование решает. То есть, она специальная. Если ты будешь интернировать какие-то строки, то да, будет так работать. А знаешь, какие вот по дефолту строки интернируются?

00:53:04,560 - 00:53:13,520 - По дефолту. Интернируется. Ну, стринг-эмпти, не знаю. Просто кавычки там.

00:53:13,960 - 00:53:32,880 - Блин, какие по дефолту, честно, интернируются? Не знаю, такое впервые встречаю. На самом деле, при билде приложения, то всякие константы и так далее, которые компилятор понимает, что они не будут меняться, они интернируются, словом. А, ну, кстати, да, на «Podomyt», «Конст» — да, он же неизменяемый никак у нас.

00:53:33,420 - 00:53:51,960 - Окей, а ты, получается, вот стринг-билдер, да, ты упомянул, вот как, думаю, что будет лучше, если у тебя какой-нибудь метод принимает, там, три короткие строки, там, по три символа, и тебе нужно их, там, сложить, условно, в выдаче, вот что лучше? Конкатенации встроенной, да, их сложить, там, через плюсик или…

00:53:52,300 - 00:53:59,320 - Через интерполяцию. Или, там, объявить стринг-билдер и через него складывать их и возвращать стримы.

00:53:59,320 - 00:54:01,010 - Я смотрел где-то

00:54:01,010 - 00:54:06,210 - Читал пост касаемо вот этих тоже, ну, всяких прикольных моментов со строками.

00:54:06,210 - 00:54:24,370 - Я смотрел и помню, что если, допустим, у нас строка маленькая, и нужно сделать какие-то, ну, небольшие преобразования, то конкат в этом случае будет отрабатывать вроде бы быстрее, чем стрингбилдер, потому что стрингбилдер нам нужно, во-первых, инициализировать, под капотом он там выделяет сразу место,

00:54:24,370 - 00:54:32,800 - достаточно большое. И если у нас строки маленькие, ну то есть нужно с ними небольшие операции сделать, то конкатенация здесь, скорее всего,

00:54:32,800 - 00:54:34,260 - Будет маст… Ну.

00:54:34,280 - 00:54:36,680 - выдаст больше производительности.

00:54:36,900 - 00:54:38,600 - Ну да, всё верно.

00:54:38,880 - 00:54:45,300 - А приходилось тебе вообще какие-нибудь утечки памяти, там, профайлить, анализировать, исправлять?

00:54:45,420 - 00:55:00,080 - Какие-нибудь тузами, может, пользовался. Может, ещё бенчмарк.нетом пользовался. Вот, кстати, да-да, бенчмарк.нет я пользовался, но использовал его в рамках F-Core, в рамках коллекций, когда сам по приколу смотрел, там, условно, работу...

00:55:00,080 - 00:55:08,270 - А, там, Dictionary, и мутабельный… и мутабельного Dictionary. Ну, и мутабельный коллекция, точнее. И Frozen. Вот.

00:55:08,270 - 00:55:09,910 - Памяти.

00:55:09,910 - 00:55:29,000 - Со спаном, с пеном немножко поработал, но если вот сейчас разбирать и вспоминать, как он под капотом работает, то я не вспомню вот этого. Ну, Benchmark.net использовал. В контексте строк, да, было? Да, там, да. Я помню, мы сперва… Как сперва у нас вообще был лекция по спанами?

00:55:29,000 - 00:55:43,600 - В рамках работы. Ну, это было очень давно, я сейчас не вспомню. А так, какой-нибудь дот-мемори, там, приложение анализировал, например, как-то изолировал. Приходилось, не приходилось. С памятью прям нет, я использовал...

00:55:43,600 - 00:55:59,580 - Больше с асинхронностью, когда в стейт-машину влезал, ну, под капотом смотрел, во что разворачивается. По памяти не было такого. Окей. А ты вот про «Продикшен», Эрис, заговорил. У вас какая там…

00:55:59,580 - 00:56:14,560 - Сложность алгоритмическая поиска, например, элементов. А там такая двоякая ситуация, что если у нас, допустим, нет коллизий, то это O1. Если у нас есть коллизия, то у нас получается связанный список, и там…

00:56:15,060 - 00:56:21,880 - ОАТН, потому что нужно перебрать все элементы, которые есть в рамках линкед-листа, ну, связанного списка.

00:56:22,360 - 00:56:25,060 - А Frozen дикшенари вы уже юзали, да?

00:56:25,240 - 00:56:36,930 - Нет, нет, ну у нас таких примеров не было прям, не сразу как бы новые фичи юзаем. Я просто их для себя смотрю на практике, сам изучаю, потому что интересно, и вдруг, если пригодится, я сразу знаю.

00:56:36,930 - 00:56:49,140 - Да, Frozen — это прикольная тема. То есть он анализирует тип ключа и в зависимости от этого подбирает, там, нужный алгоритм. Да, я видел, там, под капотом прям тоже, ну, дерево, по сути, используется.

00:56:49,460 - 00:57:01,300 - В зависимости от ключа. Эта штука, кстати, эта штука, видимо, взята из Крикхауса, потому что Крикхаусы, они у себя тоже делают… Ой, у меня, наверное, шума достал. Не, нормально. Нормально.

00:57:03,400 - 00:57:07,160 - Ну, явно, переиспользую, чего там развиваются технологии.

00:57:07,160 - 00:57:24,840 - Ну, Frozen Dictionary я смотрел. Это прикольная штука, когда, условно, нам нужно загрузить элементы в словарь и больше его никогда не менять. Он в моменте выделяет очень много памяти, а локация происходит, но потом он выигрывает по производительности, как я и смотрел.

00:57:24,840 - 00:57:30,040 - У меня ещё вот такой вопрос. Пользовался ли ты System Channel с каналами?

00:57:30,820 - 00:57:36,040 - Систем Ченнелс? Ченнел. Я вспоминаю. Ченнел.

00:57:36,240 - 00:57:46,270 - Но это вроде бы тема с вип-сокетами связана, правильно? Ну нет, это такие внутренние очереди в дот-нете легковязные. Просто интересный такой в топе.

00:58:06,100 - 00:58:17,620 - Не, не восьмой, а раньше намного. То есть в шестой они точно были? Может, они в пятом появились? Или в шестом? Что-то такое. Вот, сейчас нашёл, да. Посмотрю, спасибо. Я не слышал про неё.

00:58:20,640 - 00:58:23,900 - А, что ещё раз? — «Интерфейсом» — «I observable».

00:58:25,740 - 00:58:41,160 - Раз мы пошли. «I observable»? А, «I observable»! Я помню, мне на собеседовании одном сказали реализовать ивент, который срабатывает при изменении коллекции. И тогда я посмотрел на «Observable Collection», если это про это.

00:58:41,160 - 00:58:55,660 - Там, короче, да, там что-то в этом роде. Это библиотека реактивного программирования — React.js или Rx.js. И вот там тоже можно всякие такие штуки делать. Это на замену в Anthem как раз и Dirt, который есть в Sharp.

00:58:55,700 - 00:59:10,900 - Довольно-таки интересная штука. Она, по сути, по сути, вот это Rx.net, он делает тебе расширение как RINQ, но они идут на события. То есть те события, которые попадают, да, в эту коллекцию, они сразу же обрабатываются, и на выходе что-то происходит.

00:59:11,120 - 00:59:27,740 - На, я вот когда на Нагуляре писал, там это всё обмазано просто этими штуками. Да, и по дотам тоже. Вот это, знаете, это отдельно либо оно, ну, по производительности немного такое себе не очень, поэтому я не пользовался особо. То есть это подход прям, то есть если ты начинаешь ими пользоваться, то там всё обмазываешь как будто такой.

00:59:28,080 - 00:59:39,080 - 1988 там другая там полностью ругая парадигма синхронного программирования хотя синхронность нам гораздо быстрее получается сливать за

00:59:42,720 - 01:00:01,440 - Либо «ак», «нет» и всякая такая фигня. Тоже не на эту штуку. Ладно, это уже в топике. Давайте к чему-нибудь. У меня, наверное, всё по языку, Борис, ну, если хочешь, ты кинь что-нибудь. Ну, у нас это стандартно — это контексты синхрониза… Ой, не контексты синхронизации. Примитивы синхронизации.

01:00:02,340 - 01:00:19,540 - Какие, знаешь? Расскажи, что такое. Да, примитивы синхронизации делятся на два типа, как я знаю, пользовательские, на уровне ядра операционной системы, и гибридные. Там прикол в том, что вот пользовательские — это, допустим, Mutex.

01:00:19,960 - 01:00:33,700 - Монитор, что ещё у нас, семафор, они как бы идут напрямую в ядро операционной системы. А гибридные — это Slim-версии. Вот я как вспоминаю, там, семафор Slim, к примеру, что ещё у нас есть из Slim-версии.

01:00:33,980 - 01:00:44,480 - ReaderWrite, LogSlim вроде бы, что ли. Они как бы крутятся в специальном отдельном спинвейте и нужны для оптимизации, когда очень быстро выполняются у нас

01:00:44,480 - 01:00:53,670 - Блокировки и вот эти моменты. Вот. На практике использовал Mutex. Обычный лог у нас тоже был. По сути, это монитор. Что ещё?

01:00:53,670 - 01:01:08,240 - Симафор и Симафор Слим. Вот. Это именно вот в рамках работы. А сам на практике в своих подпроектах писал ещё интерлокт вроде бы на атомарной операции. Пробовал с ним играть. Вот. Как-то так.

01:01:13,920 - 01:01:26,340 - Да, ну это больше тоже было для обучения в рамках компании, то есть.

01:01:26,400 - 01:01:37,960 - Новеньких, вот. Там я копался прям. Ну, сказать было интересно. Не сказать, но прикольно, что вот его посмотрел. Угу. Так.

01:01:40,020 - 01:01:51,440 - А ты копался в аллокациях памяти, как вообще.NET это делает, и смотрел там новую штуку про «Реджен» и всякая такая? — Про «Реджен» впервые слышу?

01:01:51,740 - 01:02:08,520 - Реджин, нет, скорее, Реджин — это про комментарии, нет? Это, ну, не в этом подтексте. Да, это, короче, там тоже довольно сложная штука. Там раньше при Джамме выделялась большими, а с Дутнэтом, например, 8-го это сам выделяется такими небольшими кусочками. То есть, это пейдж поддерживает

01:02:08,580 - 01:02:15,740 - что поделится на «Альгиона». Ну ладно, это тоже, это прям уже совсем глубоко, вот.

01:02:16,940 - 01:02:31,180 - Ой, про Threadpool, наверное, тоже особо нет смысла. Да нет, нормально. Наверное, всё пододносит. Ну да. И у меня тоже вопросов нет. Давай в целом про инфу поговорим, про докеры

01:02:32,000 - 01:02:35,560 - САД, вот эти вот всё. Да, с докера начнём.

01:02:44,800 - 01:03:02,140 - Сред. Сперва у нас всё в ИСе вообще разворачивалось, но потом пришли к использованию докер, создавал докер-контейнеры, по сути, с докер-файлом работал. Не сказать, что прям все детальные команды условно знаю, в процессе разбираюсь. Ну и плюс, когда свои, там, пэт-проекты пишу,

01:03:02,460 - 01:03:06,740 - Так или иначе, тоже использую докер. Очень удобно разворачивать всё.

01:03:07,780 - 01:03:14,860 - Угу, а вот мультистейдж билды, например, знаешь, что такое? Мультистейдж билды?

01:03:16,140 - 01:03:29,700 - Крустя, ты на базе докера только локально, да, условно, пользовался? Какие-нибудь там CICD с ним не настраивал? Там билд… Нет, CICD у нас был настроен на TeamCity, использовался. Я прям как-то в него не внедрял.

01:03:30,980 - 01:03:36,620 - А как вот, как докер-файл был устроен, словно, вот, для дотнет-проекта?

01:03:37,540 - 01:03:55,780 - А, ну, просто, по сути, стандартные команды, там, run, в зависимости от того, нужно ли использовать копирование некоторых, допустим, файлов, там, upsetting.json, нужно ли было копировать, делать mount, как я знаю, из операции. Базовый образ изначально какой был?

01:03:56,280 - 01:04:00,940 - Базовый образ. Вот, дотнайте из SDK и рантайма, например.

01:04:02,580 - 01:04:07,740 - Базовый образ, ну вроде… Блин, читаю туплю, ну вроде бы рантайм, что ли, шёл.

01:04:09,220 - 01:04:24,200 - Там, на самом деле, сбилдить можно в СДК только, получается, проект. А запускать и там, и там, например, можно. Ну, я запускал только через, именно вот, Докер, через там, PowerShell запускаю, команду выполняю, и всё. Поэтому здесь могу тупить немного.

01:04:24,480 - 01:04:42,680 - но в докер файле как раз таки есть мультистейдж то есть ты сначала from sdk делаешь базовый образ sdk гильдишь в нем проект но он много весит то есть а если ты в нем и запускать будет что у тебя докер образ будет в итоге огромный там большой вот поэтому

01:04:42,960 - 01:04:57,640 - Второй стейдж используют, то есть берут from-рунтайм, и вот эти артефакты из SDK перекидывают в легковесный рунтаймовый образ, и уже там запускают прилагу, чтобы она мало весила, без там лишнего всякого.

01:05:07,780 - 01:05:24,460 - А про кэширование слоев в докере знаешь что-нибудь? Как вот оптимизировать можно? Ну вот мы сейчас поговорили, как, например, можно оптимизировать, там, размер, да, более раковестный, там, использовать базовые образы, например, вот как скорость сборки.

01:05:25,680 - 01:05:46,660 - Скорой сборки, кэширование. То есть у тебя не было особо таких кейсов, чисто локального, да, юзов? Ну окей. Ну нет, ну просто как там, по работе нужно… У нас была документация, условно, в зависимости её выполняешь команды, если что-то не получается, уже разбираешься. Вот прям мультистейдж и кэширование не сталкивался.

01:05:48,860 - 01:05:51,440 - В OPCI-CD, получается,

01:05:51,680 - 01:05:56,840 - у вас это все настроено строена на было кем-то то есть до 2 в санте условно туда

01:05:56,920 - 01:06:10,060 - Не лазил, не приходилось там что менять. Ну, сам настройку не выполнял, потому что всё работало из-под коробки. Условно, залил свою пачку изменений, нажал старт и вливается там на прот или стейдж, условно.

01:06:10,060 - 01:06:17,520 - А с GitLab CI, например, имел дело? Смотрел? А, имел опыт с GitHub-экшенами, вот.

01:06:17,520 - 01:06:31,770 - Там пайплайны выполняются. Можно к ним прикрутить разные правила. Там, допустим, в зависимости от названия ветки мы ставим разрешение, там, мердж или нет. А с Github'ом… У нас Github тоже использовался, но

01:06:31,770 - 01:06:36,520 - Не смотрел пайплайны, вот как они там выполняются, только в рамках гитхаба немного посмотрел.

01:06:38,060 - 01:06:42,200 - Аркестратор у вас какой был? То есть вы куда деплоились?

01:06:42,700 - 01:06:44,520 - Аркистрактор.

01:06:44,760 - 01:06:53,020 - Блин, здесь я тоже не подскажу. Кубер, например, или… А, у нас K8S, но это вроде бы Кубер как раз, Кибернетис.

01:06:53,160 - 01:07:10,180 - Да, там у нас он использовался на… Ну, тоже в рамках локальной какие-то. Создать сессии быстренько для тестирования, вот. Ну, у нас всё есть в подкоробке, я прям в кибернетизм вообще не лазил. Это тоже, я как слышал, тяжёлая… Вот вам, получается, вот этот файлик диплоя-то тоже кто-то описывал?

01:07:10,180 - 01:07:14,800 - дебопс какой-нибудь или как? Вот именно файл релиза в Кубер.

01:07:14,820 - 01:07:29,300 - У нас такового прям отдельной роли для DevOps-а не было. У нас был архитектор, он, по сути, был и мамой, и папой, ну, там, закрывал вот эти бизнес-задачи. Угу. То есть, условно,

01:07:30,900 - 01:07:46,220 - Не приходилось, да, менять какую-то конфигурацию, диплой, вот это вот всё? Нет, я прям… А вот со стороны кода, вот со стороны приложения, ну, были бы какие-то требования, вот какие-нибудь health-чеки, например, написать, там, какие-то обязательные пробы, вот, для кубернетиса?

01:07:46,920 - 01:08:02,440 - А, конкретно для кубернетиса я не писал тоже такого. Хеллчеки я писал и работал даже с библиотекой, там, Hankfire для UI-ки, просмотр. Джоб — это вообще для Джоб вроде связано, да, Hankfire с Джобом связано. Ну, хеллчеки я писал.

01:08:03,160 - 01:08:04,480 - Для Аки.

01:08:04,780 - 01:08:21,890 - У нас был health check самый простой для бдшки. Просто, условно, делаем select, если база выполняется, значит, она жива. Потом health check у нас был для одного из микросервисов важных, который отвечал за, вот, как раз, проект C2C, то есть это просмотр

01:08:21,890 - 01:08:33,880 - Работы нашей биржи. Там прям полностью, ну, там UI-ки не было красивые, там просто огромный текст был, и по нему мы смотрели в самых начальных приложениях, типа, работает или нет.

01:08:33,900 - 01:08:39,120 - Вот. Что ещё? Вот эти хеллчеки, они на что влияли?

01:08:39,180 - 01:08:53,390 - По итогу, вот. Ну, с помощью хелс-чека можно было посмотреть, там, активен или… Ну, работает ли в данный момент наш, условно, наше ПО на сервере, крутится ли оно там.

01:08:53,390 - 01:09:02,200 - То есть оно чисто вот для визуала можно, там, зайти посмотреть. Вот только в этом плане, да, оно применяет? Ну, health-чеки…

01:09:03,540 - 01:09:23,060 - Что произойдёт, если у вас, например, ну вот база данных отворилась, и хелп-чек грохнулся, который проверял базу данных? Да, тут ещё такое… Ну, у нас обычно как было, если что-то падало, нам звонили просто, типа, всё сломалось. А кто вам за этим следит, или это всё-таки автоматически какой-то звонок, например, отвода, там, или…

01:09:23,940 - 01:09:34,860 - У нас просто за этим, типа, следили саппорты, можно их так назвать, типа. И они звонили. А так, я могу предположить, тут, скорее всего, вопрос идёт к метрикам — Графан или Приметеус.

01:09:36,460 - 01:09:50,340 - Да, во, то есть, у нас Графана была, с ней я поработал. Но в основном нам просто звонили, то есть, если что-то ночью падает, звонили, вот так. Ну это алерты как раз, я не знаю, можно ли это назвать. Уведомления у нас приходили тоже.

01:09:50,340 - 01:09:58,360 - А вот вспомнил, когда что-то ломалось, приходили просто в слаке от бота. Но ночью, к примеру, фиг посмотришь, звонки просто тупо шли.

01:09:59,520 - 01:10:05,260 - Ну да, понятно. Ну, собственно. А метрики, вот, какие-то вы отслеживали, вот, в приложении?

01:10:06,140 - 01:10:20,480 - Counter, я помню, точно использовался у нас. Метрики. Ну, у нас очень большая была метрика по нашим ивентам аналитическим, там, с какой скоростью, условно, выполняются запросы. Вот больше для этого

01:10:20,640 - 01:10:27,980 - А что ещё метрики? Какие мы смотрели? Вот эти метрики, они как были реализованы? То есть это какая-то сторонняя либо там была

01:10:28,400 - 01:10:41,740 - Они выставлялись в каком-то формате, там, OpenTelemetry, может, слышал про это. Вот OpenTelemetry как раз у нас и использовался. Я его и на практике тоже с ним повозился сам. Он же обычно идёт OpenTelemetry и плюс там, графана.

01:10:43,100 - 01:10:51,580 - Вот, условно, OpenTelemetric, он, не знаю, с Prometheus, он вроде тоже связан должен быть. Вот, это связка была.

01:10:53,040 - 01:10:56,140 - Угу. А вот логи, ошибки — как это отслеживали?

01:10:56,160 - 01:10:57,320 - А, да, у нас.

01:10:57,320 - 01:11:01,620 - У нас была своя собственная кастомная

01:11:01,620 - 01:11:21,830 - ПО для логов. В основном мы сперва, ну, мы сразу начали использовать неориентированную базу данных Clickhouse, потому что она как раз для сбора данных очень классная штука. Там под капотом движки, и в зависимости от движков, ну, там, в общем, движки на любой вкус, и как раз для логов они идеально подходили.

01:11:21,830 - 01:11:24,340 - и просто в нашей собственной системе

01:11:24,540 - 01:11:41,860 - Они там и показывались. Можно было статус выбрать, по определённым лейблам найти информацию, там, по session id, по дате и так далее. Они пошли чуть дальше, чем Елка.

01:11:43,680 - 01:12:01,100 - Да, а вы какой-нибудь Centric не использовали? Нет, чего-то такое вообще не видел, что у нас она использовалась. Ну, может, использовалась, может, я просто до неё не доходил. У нас проект был огромный, и бывало такое, что не всегда прям со всеми инструментами, ну, можно было поработать.

01:12:02,820 - 01:12:23,040 - ладник у меня на пин-фре особо нетать? Я у тебя просто тоже видел, что…

01:12:23,500 - 01:12:27,940 - Ты там писал тесты, продвигал это всё дело. Можешь вот про это рассказать?

01:12:28,240 - 01:12:30,480 - А, да, у нас… Да, да.

01:12:30,480 - 01:12:44,770 - Я использовал интеграционный тест для микросервисов, а юнит-тесты на.NET и функциональные на Python. Правда, с функциональными было немножко. Вот. У нас на проекте вообще тестов, от слова, было совсем ноль, их просто не было.

01:12:44,770 - 01:12:59,080 - Вот. А, собственно, когда мы выкатывали новые какие-то фичи, то не всегда было такое, что, условно, тестировщик всё проверит, и мы сами могли всё проверить, что всё работает идеально. Поэтому очень часто встречались серьёзные ошибки на продакшене, что

01:12:59,080 - 01:13:10,940 - Ну, приводили, так сказать, к серьезным проблемам. И я, собственно, предложил идею. У нас команда на этом проекте была из пяти человек, чтобы начать писать юнит-тесты. Вот. Я занялся написанием...

01:13:10,940 - 01:13:20,700 - юнитесты для новых фич, а ребята как раз параллельно для старых, условно важных, функциональных моментов писали для них юнитесты.

01:13:23,600 - 01:13:36,560 - А знаешь, разница между юнит-тестами и компонентными тестами? — Компонентными? — Ну да. Ну, просто, возможно, ты писал не совсем юнит-тест, а компонентный тест, потому что они…

01:13:36,700 - 01:13:39,460 - Очень похоже, на самом деле, по своей…

01:13:39,460 - 01:13:43,340 - То есть, ну ладно, давай так, компонентные тесты

01:13:43,340 - 01:14:02,840 - Весь какой-то большой модуль, там, запрос, например, request, там, какой-нибудь медиатор покрывают, там, с обращением в базу данных, ещё что-то туда можно протянуть. А юнит-тесты – это ты мог одного конкретного, там, сервиса, метод какой-то тестировать. То есть, это прям совсем маленький. Вот. Это просто… Да, я просто думал, что…

01:14:02,840 - 01:14:08,630 - А интеграционные в целом большой скоб каких-то компонентов под себя подразумевают и тестируют.

01:14:08,630 - 01:14:26,690 - Но в основном в рамках «Монолита» у нас был чисто МОК одного сервиса. С интеграционными мы уже делали использование разных, там, базы данных, там. А вот как… Ну, это, можно сказать, и был компонентный. Я думал, что это синоним, условно. Да, просто интеграционный тест — это ещё, как бы, отдельный этап.

01:14:26,690 - 01:14:36,490 - По сути, у тебя другое приложение тестирует свое приложение на какие-то… То есть оно уже непосредственно отбирает запросы, к нему реквесты, смотрит, какие ответы ему придут.

01:14:36,490 - 01:14:47,960 - Там ещё всякая такая штука. То есть, это ещё такой, чуть более широкий. Там уже проверяется сама диссерилизация твоего запроса. То есть, запросы, как правило, приходят, там, в результате, ещё что-то. Ну, это так, по этим.

01:14:47,960 - 01:14:50,320 - Так, хорошо.

01:14:50,700 - 01:14:56,980 - Рахажи, вот как вы подымали базу данных для тестов? Вообще, как это происходило?

01:14:57,480 - 01:15:08,460 - То есть, вот, у вас были тесты с базой данных. Это был какая-то просто тестовый слой, или вы подымали под каждый запуск теста какую-то базу данных, или как-то ещё?

01:15:09,240 - 01:15:21,760 - Угу, нет, у нас уже была развернута бд-шка, у нас тоже использовался, по сути, кликхаус для… Ну, туда попадали логи, все наши разные фичи, которые мы тестировали, связанные с какими-то…

01:15:21,760 - 01:15:36,420 - Большим объём данных, вот. У нас в основном использовался клик для этих ситуаций, чтобы по… Ну, из клихауса, как вот я на практике встречал, можно с помощью, как сказать, кавычек, доставать определённые данные там.

01:15:36,460 - 01:15:46,640 - Ну, семейство данных. Вот. У нас она уже была развернута, мы просто обращались к ней и получали набор данных. И всё, в целом. Как-то…

01:15:46,820 - 01:16:02,160 - А на ПГ у вас не было тестов? На ПГ? Да, у нас вот интеграционные часто были именно с Тлихаусом. На ПГ у нас были тесты, но он тоже был развернут в рамках уже отдельного сервера. Мы локально не поднимали их. Понятно. Ага.

01:16:02,840 - 01:16:12,100 - Так, у тебя ещё были функциональные тесты, которые говорил на «Питоне», я там что-то читал. Да, ну там мало, конечно, опыта было, но был немножко.

01:16:13,840 - 01:16:30,850 - А, нет, нам нужно было на функциональных на Пайтоне тестировать какие-то… Я уже не помню, что-то честное я тестировал, это было давно, но вроде бы обращение к очереди, там, с эксченджами. Мы Рэббит и работоспособность Кавки смотрели с помощью Пайтон-тестов.

01:16:30,850 - 01:16:39,020 - Понятно, ладно. Странно, что вы не использовали для этого фиша. Ну ладно. Я вот сам не понял прикола, приходилось изучать Python.

01:16:39,280 - 01:16:54,100 - И это извращение. То есть, у нас тоже бывают такие ситуации, когда нам приходят, такие, у нас вся команда на.NET-е, а нам — где? Пришёл там точерёк, и говорит, я буду писать тесты на Python. И вот один человек пишет на Python, и потом такой спрашивает, «Ребята, как?»

01:16:55,540 - 01:17:00,360 - В этом и проблема, что нужно было всем ребятам, как бы, поддерживать это всё потом.

01:17:00,580 - 01:17:21,080 - Ага, ага. Так, а вот на.NET-е ты писал тесты, и что, вот, что выбрал, выбрал? XUnit, NUnit, может, MS-тест вообще? Да, у нас был XUnit, использовался. XUnit, XUnit. Приходилось в Х-юните делать какие-то

01:17:21,400 - 01:17:28,700 - Совет, какую-то предварительную инициализацию для всех, например, тестов?

01:17:28,700 - 01:17:39,870 - Вспоминаю, у нас использовалась такая либо как автофикстура, что ли. Автофикстура, да. Да, она как бы сразу нам и приводила нужный объём данных.

01:17:39,870 - 01:17:49,680 - реализовывал, и мы просто вот их тестили. Ну, понятно, да. То есть, у вас не было каких-то тестов, где вам нужно, там, первоначально настроить… Ну, вот смотри, нет, ну,

01:17:49,680 - 01:17:51,260 - Тут, наверное, это Москва.

01:17:51,260 - 01:18:02,320 - Вот у тебя есть база данных, да, то есть, у тебя есть, там, DB-контекст, да. То есть, в каждом тесте, получается, DB-контекст создавали из констанции на это счет. То есть…

01:18:02,320 - 01:18:22,380 - В основном те тесты, которые осмотрел, у нас каждый раз создавались собственные, каждый раз новые данные, и мы просто с ними тестили. А, понятно. Ну ладно. Собственно. Хорошо. У меня, в принципе, всё по тестированию. А у нас ещё что-ейк? Даннера нет, можем Таню звать, призывать.

01:18:22,380 - 01:18:23,440 - Ну да.

01:18:23,620 - 01:18:31,260 - Покажи, что так по времени как раз помог. Тоже. Угу. Ой, так, секундочку.

01:18:31,760 - 01:18:41,006 - Да, всё, отлично, отлично. Раз мы закончили с этой частью, то можем переходить к более таким общим вещам. Я ещё несколько вопросов, Андрей, тебе задам.