Рассмотрим повышение эффективности ассинхронных API.

Начнем с **Retry**.

Повторная попытка (Retry) - это процесс повторной попытки выполнить неудачную операцию в надежде на успешный результат. Повторные попытки необходимы в ситуациях, когда операция, например, запрос API, может завершиться неудачей из-за временных проблем, таких как проблемы с подключением к сети, перегрузка сервера или другие временные ошибки.

Повторные запросы помогают повысить надежность и устойчивость приложения, предоставляя ему возможность автоматически восстанавливаться после таких временных сбоев. То есть мы не просто отображаем ошибку клиента, а пытаемся автоматически направить запрос еще раз для получения результата. Хотя мы можем говорить о том что retry – это просто повторная попытка, в том числе и от клиента вручную (когда он попробовал руками снова направить запрос).

Как эффективно управлять повторными попытками?

- Предусмотреть механизм повторных запросов в самом API: в некоторых случаях может быть полезно включить встроенный механизм автоматических повторных запросов в API. Это поможет более эффективно управлять повторными попытками и разгрузить часть логики повторных ручных попыток от клиента.
- Используйте задержку между повторами: когда вам необходимо автоматически повторить неудачный запрос, использование задержки между повторами предотвращает перегрузку сервера несколькими повторными запросами подряд. Он предполагает постепенное увеличение времени ожидания между повторными попытками.

Что делать? Начинайте автоматические повторы с короткой задержки и постепенно увеличивайте время ожидания с каждой попыткой повтора (экспоненциально: через 5 секунд повторили, через 25 секунд, через 125…. и т.д.).

- Использование Jitter - это техника, используемая для внесения случайности во время ожидания между повторными попытками при решении проблем, связанных с сетью или сервером. Основная цель джиттера - распределить повторные попытки от нескольких клиентов более равномерно и не дать им перегрузить сервер.

Представьте, что несколько клиентов одновременно пытаются подключиться к серверу. Если все они используют фиксированное время ожидания между повторными попытками, то, скорее всего, они будут отправлять свои запросы одновременно, что приведет к задержке и, возможно, перегрузке сервера. Это может вызвать еще больше проблем и замедлить процесс восстановления как для сервера, так и для клиентов.

Что делать? Рассмотреть добавление джиттера. Таким образом, вы делаете время ожидания между повторными попытками более непредсказуемым. Это означает, что попытки повторных попыток каждого клиента будут немного отличаться от остальных, что поможет более равномерно распределить нагрузку на сервер. В результате сервер с меньшей вероятностью будет перегружен внезапным всплеском запросов, а у клиентов будет больше шансов успешно подключиться.

- Повторять только тогда, когда это необходимо: не все ошибки следует повторять. Например, ошибки клиента (например, 400 Bad Request) обычно указывают на проблему с самим запросом, и повторная попытка не приведет к успешному ответу (или бизнес ошибка, как отсутствие товара или денег).

Что делать? Разрешите автоматический повтор запроса (или если клиент сам повторяет — то разрешите повторную отправку) только тогда, когда возникают ошибки сервера (например, 500 Internal Server Error) или сетевые проблемы.

- Установите максимальный лимит ручных и автоматических повторных попыток: чтобы предотвратить бесконечные повторные попытки, установите максимальный лимит на количество повторных попыток (как для автоматических повторов так и для ручных). После достижения этого предела сервер и клиенты должны прекратить повторные попытки и обработать ошибку в соответствии с конкретным случаем использования. Стандартная практика — попробовать три раза и прекратить попытки (показать клиенту ошибку).

Что делать? Чтобы показать клиентам, когда можно повторить запрос - используйте заголовок retry-after: включите в ответ заголовок "Retry-After", указывающий количество секунд, которое клиент должен подождать перед повторным запросом.

Также используйте общие рекомендации, которые при асинхронных API особенно нужно соблюдать.

- Отслеживайте и регистрируйте повторные попытки: отслеживание повторных попыток может помочь вам выявить проблемы в системе и повысить надежность вашего API.

Что делать? Внедрите системы мониторинга и протоколирования для регистрации повторных попыток, что позволит вам анализировать и выявлять потенциальные проблемы в вашем API.

- Реализуйте комплексную обработку ошибок: разработайте API для обработки различных типов ошибок, включая ошибки клиента, ошибки сервера и сетевые проблемы. Убедитесь, что ваш API отвечает соответствующими кодами состояния и информативными сообщениями об ошибках, чтобы помочь клиентам понять проблему и принять меры по ее устранению.

Что делать? Определите возможные сценарии ошибок и разработайте API для каждого случая. Используйте значимые коды состояния и сообщения об ошибках, чтобы помочь клиентам диагностировать проблему.

Следуя этим рекомендациям, вы сможете разработать асинхронный API, более устойчивый к ошибкам и способный эффективно обрабатывать повторные попытки, обеспечивая более положительный опыт для клиентов, взаимодействующих с вашей системой.