**Тестовое задание для Backend-разработчика**

**Обязательная часть:**

Реализовать приложение, которое осуществляет передачу больших объемов бинарных данных через систему обмена сообщениями NATS (<https://nats.io/>). Например, приложение может осуществлять передачу файлов JPEG, PNG или файла с UTF-8 текстом, но бинарным заголовком — при передачи данных должен быть понятен формат данных всем сторонам обмена.

У NATS системы есть ограничения и поэтому нужно передавать чанками. Простой NATS (не JetStream) не хранит сообщения для повторов при ошибках и не обеспечивает внутреннее подтверждение доставки.

Приложение должно состоять из 2-х сервисов Reader-Producer и Writer-Consumer, которые можно запустить на разных серверах (для тестирования можно запускать 2 сервиса на одной машине, главное чтобы использовался выбранный транспорт, в данном случае NATS).

Сервис Reader-Producer должен осуществлять чтение и передачу данных по заранее выбранному транспорту.

Сервис Writer-Consumer должен принимать данные от сервиса Reader-Producer, обрабатывать их и сохранять. Под обработкой понимается некая допустимая модификация бинарных данных в чанке (картинка не разрушается, но не обязательно должна выглядеть корректно после модификации). Бинарные данные могут быть как UTF8 текст и обработка может свестись к добавлению и удалению некоторых символов из текста (не только latin текста).

Сами сервисы нужно реализовать без использования готовых больших фреймворков, только на коровых модулях и минимально необходимых модулях для работы с NATS, с бинарными данными.

Обработать ситуацию, когда сервис Reader-Producer отправляет сервису Writer-Consumer больше данных в чанке, чем тот может успеть обработать, но весь объем просто будет передаваться и обрабатываться дольше (допустим 1кб успевает за 1сек, а 10кб уже не успевает за 1сек, нужно больше времени). Добавить предельное время обработки одного чанка и прерывать передачу по таймауту на стороне Reader-Producer.

Для примера, перед записью в сервисе Writer-Consumer обрабатывайте входящие данные с помощью этого метода: (псевдокод) process\_data(data) { await delay(500ms); return data; } Это будет эмулировать задержку в обработке.

Покрыть тестами в отдельных модулях.

**Дополнительно (по возможности):**

Сервис Reader-Producer может быть построен как долгоживущий сервис и запускать чтение и передачу данных по команде через свой REST API.

Реализовать поддержку различных видов транспорта (например, websockets)

Возможно дополнительно поддержать обработку различных видов данных - JSON, YAML, TEXT — можно использовать крейты для парсинга данных форматов.