Zadanie rekrutacyjne – Full-Stack (NestJS + Next.js + Mongo + WS + S3 presign)

Cel

Zaimplementuj mini-moduł „Zamówienia” w architekturze CQRS + projekcje + realtime:

- utworzenie zamówienia (idempotentnie),
- lista zamówień z filtrami,
- powiadomienie w czasie rzeczywistym (WebSocket) o zmianie statusu,
- bezpieczny upload pliku przez pre-signed URL (S3/MinIO lub mock).

Zakres projektowy jest mały, ale zależy nam na jakości: idempotencja, indeksy, prosta projekcja, poprawny WS, sensowny README.

Technologie (preferowane):

- Backend: NestJS (Node 18+), MongoDB (oficjalny driver/Mongoose), WebSocket (Nest Gateway).
- Frontend: Next.js 14+ (app router), SSR/Server Actions; prosty UI w React.
- Pliki: S3/MinIO (albo mock presign – patrz „Uproszczenia”).
- (Opcjonalnie) Redis jako cache listy (mile widziane, ale nie wymagane).

Funkcjonalności (MVP)

1. API – Create (Command)

```
POST /api/orders
```

Request (JSON):

```
{
  "requestId": "6b6e7a9d-6d2a-4c0e-9e7f-001",
  "tenantId": "t-123",
  "buyer": { "email": "alice@example.com", "name": "Alice" },
  "items": [{ "sku": "SKU-1", "qty": 2, "price": 49.99 }],
  "attachment": {
    "filename": "invoice.pdf",
    "contentType": "application/pdf",
    "size": 123456,
    "storageKey": "tenants/t-123/orders/001/invoice.pdf"
  }
}
```

Wymagania:

- Idempotencja po (tenantId, requestId) – drugi taki sam call nie tworzy duplikatu (zwróć ten sam orderId albo kontrolowany 40
- Walidacja pól, limit rozmiaru i typu pliku (na podstawie metadanych).
- Zapisz write model oraz wyemituj zdarzenie OrderCreated (wystarczy zapis do „outbox” lub in-memory event bus).

Response (201):

```
{ "orderId": "ord_abc123" }
```

2. API – List (Query / projekcja)

```
GET /api/orders?status=PENDING&buyerEmail=alice@example.com&from=2025-08-20&to=2025-08-22&page=1&limit=10
```

Wymagania:

```
•       Dane z read modelu (denormalizacja pod listę).
•       Filtry: status, buyerEmail, zakres dat (from, to), page, limit.
•       Indeksy pod filtry (patrz „Indeksy” poniżej).
```

Response (200):

```
{
  "items": [
    {
      "orderId": "ord_abc123",
      "status": "PENDING",
      "createdAt": "2025-08-22T09:00:00Z",
      "buyerEmail": "alice@example.com",
      "total": 99.98,
      "attachment": { "filename": "invoice.pdf", "storageKey": "tenants/t-123/orders/001/invoice.pdf" }
    }
  ],
  "page": 1, "limit": 10, "total": 1
}
```

3. Realtime – WebSocket • Autoryzacja w handshake (JWT w httpOnly cookie lub token przekazany bezpiecznie). • Po zaktualizowaniu projekcji wyślij do właściwego użytkownika/tenant’a event:

```
{
  "type": "order.updated",
  "payload": { "orderId": "ord_abc123", "status": "PAID" }
}
```

4. Upload – pre-signed URL

```
POST /api/uploads/presign
```

Request:

```
{ "tenantId": "t-123", "filename": "invoice.pdf", "contentType": "application/pdf", "size": 123456 }
```

Response:

```
{
  "url": "https://minio.local/…signed…",
  "storageKey": "tenants/t-123/orders/001/invoice.pdf",
  "expiresInSeconds": 120,
  "headers": { "Content-Type": "application/pdf" }
}
```

Flow: klient pobiera URL → PUT pliku bezpośrednio do S3/MinIO → w POST /api/orders podaje storageKey.

Frontend (Next.js)

```
•       Lista (SSR) z filtrami i paginacją, po załadowaniu subskrybuje WS i aktualizuje rekordy bez przeładowania.
•       Formularz tworzenia:
    1.  wywołuje presign,
    2.  robi PUT na url,
    3.  wysyła POST /api/orders,
    4.  pokazuje optimistic row (PENDING), a potem aktualizuje status z WS.
```

Dane / Indeksy / Multi-tenant

- Każda operacja jest w kontekście tenantId.
- Minimalne indeksy (Mongo):
- unique (tenantId, requestId) – idempotencja (write model).
- (tenantId, status, createdAt desc) – lista.
- (tenantId, buyer.email) – lista.
- Read model może być osobną kolekcją (np. orders_read).

Uproszczenia dozwolone (czas 3–5h)

- Broker zdarzeń: jeśli zabraknie czasu na Kafkę – możesz użyć in-memory publish/subscribe i setTimeout(2–5s) do symulacji zmi
- S3/MinIO: jeśli nie używasz dockera, możesz zwrócić mock presign (URL do lokalnego endpointu, który przyjmie PUT i nic nie z
- Auth: uproszczony JWT; nie wymagamy rejestracji/logowania – wystarczy „podszyty” użytkownik i tenantId.

Jak uruchomić (proponowana struktura)

```
/apps
  /api    # NestJS (komendy, projekcje, WS gateway, presign)
  /web    # Next.js (SSR lista, formularz, klient WS)
  /infra  # opcjonalnie: docker-compose (mongo, minio, redis)
README.md
```

W README opisz:

- instalację, komendy (dev, build, start),
- uruchomienie dockera (jeśli używasz),
- przykładowe curl do akceptacji (patrz niżej),
- założone kompromisy i co byś zrobił w v2.

Kryteria akceptacji (sprawdzamy ręcznie)

1. Idempotencja

1. create

```
curl -s -XPOST http://localhost:3000/api/orders -H 'Content-Type: application/json' \
 -d '{"requestId":"r1","tenantId":"t-123","buyer":{"email":"alice@example.com","name":"Alice"},"items":[{"sku":"SKU-1","qty":2,"price
```

2. ponownie z tym samym requestId

```
curl -s -XPOST http://localhost:3000/api/orders -H 'Content-Type: application/json' \
 -d '{"requestId":"r1","tenantId":"t-123","buyer":{"email":"alice@example.com","name":"Alice"},"items":[{"sku":"SKU-1","qty":2,"price
```

oczekujemy: brak duplikatu (ten sam orderId lub 409/200 z jasnym komunikatem)

2. Lista + filtry

```
curl -s 'http://localhost:3000/api/orders?tenantId=t-123&status=PENDING&buyerEmail=alice@example.com&page=1&limit=10'
```

– szybka odpowiedź z read modelu, paginacja i filtry działają.

3. Realtime

- Po utworzeniu zamówienia status zmienia się z PENDING na PAID w 2–5 s; UI aktualizuje się przez WS (bez reloadu).

4. Upload

- POST /api/uploads/presign → PUT pliku (np. curl -T file.pdf <url>), a potem POST /api/orders z storageKey.

Jeśli dołączysz Redis cache listy: przy powtórnym odświeżeniu listy pokaż w logu cache hit, a po order.updated – unieważnienie.

Na co patrzymy (co punktuje)

- Poprawna idempotencja i sensowna obsługa błędów.
- Prosta, ale czytelna projekcja (separacja write/read).
- WS po projekcji, nie „na pałę” po command.
- Min. indeksy pod zadane filtry.
- Presign flow bez proxy dużych plików przez backend (+ walidacje).
- Krótki, konkretny README (jak uruchomić + decyzje + kompromisy).
- Kod, który „się czyta”: podział na moduły, nazewnictwo, typy.

Czas

Szacujemy 3–5 godzin. Nie trzeba robić wszystkiego „na tip-top” – ważniejsze są decyzje i jakość niż wodotryski.

Dostarczenie

- Link do repo (GitHub/GitLab, publiczny lub dostęp dla recenzenta).
- Krótki film (opcjonalnie, 2–3 min) pokazujący flow: create → lista → zmiana statusu → WS → upload.

FAQ

- Czy muszę użyć Kafki? Nie – możesz zasymulować eventy in-memory. Interfejs zdarzeń zaprojektuj tak, by łatwo było potem podm
- Czy muszę użyć MinIO? Nie – ale presign flow powinien być realny (PUT na zwrócony URL, walidacje, TTL).
- Auth? Minimalny JWT, wystarczy stały użytkownik/tenant.
- Testy? Nie wymagamy, ale mile widziane krótkie e2e/smoke (np. supertest).

Załącznik (opcjonalne kontrakty – możesz wkleić na końcu briefu)

```
// Eventy (propozycja)
type OrderCreated = { type: 'orders.created.v1'; tenantId: string; orderId: string; payload: { /* buyer, items, total, attachment */
type OrderStatusChanged = { type: 'orders.status.v1'; tenantId: string; orderId: string; payload: { status: 'PENDING'|'PAID'|'CANCEL

// Read model rekord
type OrderRead = {
  orderId: string;
  tenantId: string;
  buyerEmail: string;
  status: 'PENDING'|'PAID'|'CANCELLED';
  total: number;
  createdAt: string;
  attachment?: { filename: string; storageKey: string };
};
```