Technical Assignment – Software Engineer (Full-Stack, Real-Time Analytics)

We'd like you to complete a full-stack technical assignment that evaluates real-time systems, state management, and production-ready engineering.

Assignment: Real-Time Payment Analytics Dashboard (Full-Stack)

Build a real-time payment analytics dashboard with WebSocket updates and interactive charts, delivered as an Nx monorepo with:

- Frontend: Next.js (TypeScript) + Redux Toolkit + RTK Query + MUI + Recharts
- Backend: NestJS (TypeScript) + WebSocket Gateway + MongoDB
- Data: MongoDB (schemas, indexes, seed)
- Transport: REST + WebSockets
- Bonus: Redis caching; multi-tenant design (org isolation); containerized dev; CI basics

Existing Contract / APIs to Implement

WebSocket events (server -> client)

```
interface PaymentEvent {
  type: 'payment_received' | 'payment_failed' | 'payment_refunded';
  payment: Payment;
  timestamp: Date;
}
```

Text

REST (server)
GET /api/analytics/metrics
GET /api/analytics/trends?period=day|week|month

WS endpoint
/ws/payments

types/analytics.ts (shared)

```
export interface PaymentMetrics {
  totalVolume: number;
  successRate: number;
  averageAmount: number;
  peakHour: number;
  topPaymentMethod: string;
}

export interface TrendData {
  timestamp: Date;
  amount: number;
  count: number;
  successRate: number;
}
```

Functional Requirements

A) Real-Time Data Management (Backend + Frontend)

Backend (NestJS)
- WebSocket Gateway at /ws/payments broadcasting PaymentEvent.
- REST controllers for /api/analytics/metrics and /api/analytics/trends.
- MongoDB models (Payments, Methods, Tenants) with indexes.
- Streaming aggregation logic.

Frontend (Next.js)
- WebSocket service with reconnection, filtering, and error handling.
- Redux Toolkit middleware for updating metrics and event history.
- RTK Query for REST queries (metrics, trends) with caching and invalidation.

B) Dashboard UI

- MetricsGrid (MUI Grid): total volume, success rate, average amount, active methods, peak hours.
- TrendChart (Recharts): toggle day/week/month; display volume + success rate; highlight anomalies.

- EventsFeed: real-time, color-coded, clickable details; auto-scroll with pause.

C) Performance & Interactivity

- Virtualized events list; memoized chart data; throttled updates.
- Drill-downs; time-range selector; CSV export; full-screen charts.
- Alerts: failure spike detection, volume thresholds, toast notifications.

Bonus (Nice-to-Have)

- Redis caching for hot endpoints with cache invalidation.
- Multi-tenant design via X-Tenant-Id or subdomain parsing.
- Docker Compose for local dev (web, api, mongodb, redis).
- Basic CI (GitHub Actions): type-check, lint, test, build.

Project Structure (Nx Monorepo)

```
apps/
  web/                # Next.js (app router)
  api/                # NestJS
libs/
  shared-types/       # shared TS types
  ui/                 # shared UI components
  utils/              # shared utils
infra/
  docker/             # Dockerfiles
  compose/            # docker-compose.yml
  k8s/                # (optional)
```

Data Model (Guidance)

payments collection: { _id, tenantId, amount, method, status, createdAt, updatedAt }
Index suggestions: tenantId, createdAt, status, method
Seed realistic data and simulate live events.

Deliverables

1. GitHub repository with README, Nx scripts, .env.example, and seed instructions.
2. Short architecture note (WebSocket, state shape, MongoDB schema, Redis caching, Multi-tenancy).
3. Optional: Screenshots or Loom video of dashboard.

Evaluation Criteria

- Correctness & Contracts
- Real-Time & State
- Performance
- Architecture
- Resilience
- DX & Ops
- Bonus Quality (Redis, Multi-tenancy, CI)

Submission

Please reply with:
- GitHub repo link
- Brief setup notes if anything differs from the README