BarBooks

JS Full-Stack Developer

Take-Home Test

Jan 6, 2025

Time Limit: 60 minutes

Submission: Create a **new Git repository**, implement your work there, then push and share the repo URL.

Tech Stack:

- **Backend:** Node.js + Express + SQLite
- **Frontend:** React (or Vue) with JavaScript or TypeScript
- **Testing:** Jest (or plain assertions)
- **Configuration:** use a `.env` file for settings

Add a README.md at the repo root.

Task 1: Logic & Unit Testing (15 min)

- **Implement**

```typescript
type Order = { id: number; product: string; qty: number; price: number };

type Summary = {
  totalRevenue: number;
  medianOrderPrice: number;
  topProductByQty: string;
  uniqueProductCount: number;
};

function summarizeOrders(orders: Order[]): Summary { /* … */ }
```

- **Compute:**
  - `totalRevenue` = sum of `qty * price`
  - `medianOrderPrice` = median of all `qty * price` values
  - `topProductByQty` = product with highest total `qty`
  - `uniqueProductCount` = number of distinct products
- **Tests:** write at least two unit tests covering typical and edge cases.

Task 2: Database Schema & Mock Data (5 min)

- **Schema:** create `data.db` with an `orders` table

```SQL
CREATE TABLE orders (
  id      INTEGER PRIMARY KEY AUTOINCREMENT,
  product TEXT    NOT NULL,
  qty     INTEGER NOT NULL,
  price   REAL    NOT NULL
);
```

- **Mock Data:** include a seed script (JS or SQL) that inserts ≥5 diverse orders.

Task 3: API Endpoints (15 min)

- **Config & Middleware:**
    1. Load `DB_PATH` and `PORT` from `.env`
    2. Add simple request-logging middleware and enable CORS
- **Endpoints:**
    1. `GET /api/summary`
        - Read all orders, run `summarizeOrders()`, return JSON.
    2. `GET /api/orders`
        - Support query params:
            - `product` (partial match filter)
            - `limit`, `offset` (pagination)
    3. `POST /api/orders`
        - Accept `{ product, qty, price }`, validate inputs, insert into DB, return new record
- **Error Handling:** return 400 on invalid input; handle DB errors gracefully.

Task 4: Front-End Application (15 min)

- **Set up** a React app (CRA, Vite, etc.).
- **Custom Hook:**

```javascript
function useSummary() {
  // fetch GET /api/summary → { data, loading, error }
}
```

- **UI Requirements:**
    1. Display `totalRevenue`, `medianOrderPrice`, and `topProductByQty`.
    2. List recent orders from `GET /api/orders` (show `product`, `qty`, `price`).
    3. Form to add a new order; on submit, POST to `/api/orders`, then refresh summary & list.
- **Bonus Complexity:** add UI controls to filter by product name and paginate through orders.

Task 5: Optional Integration Test (10 min)

- **Write** one integration test (e.g. with Supertest) for `POST /api/orders` against an in-memory SQLite instance.
- **Verify** a valid order is inserted and returned correctly.

Good luck!