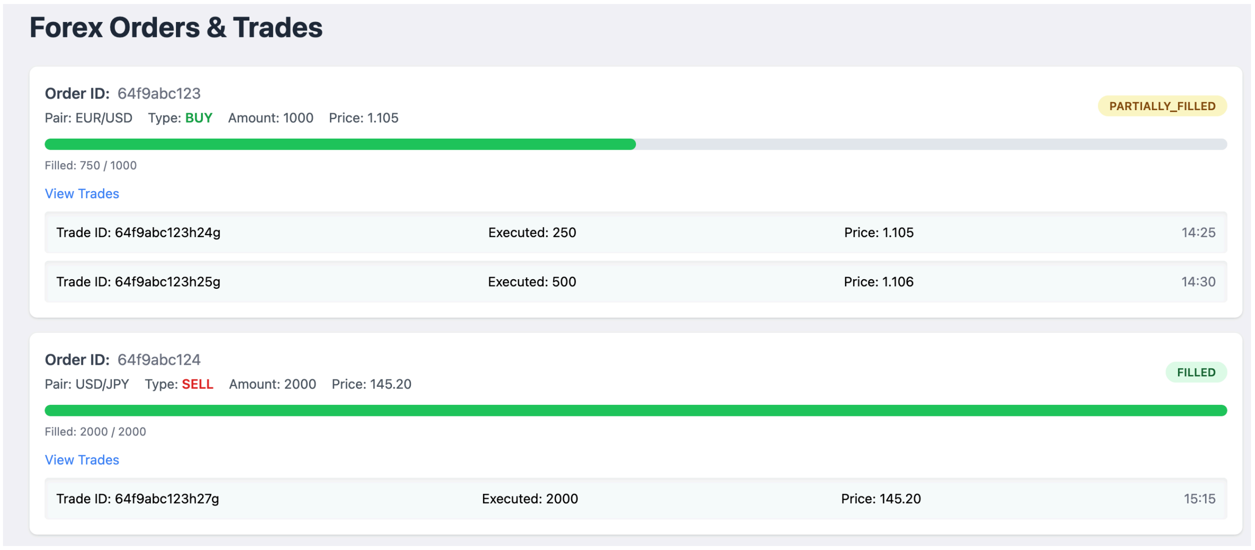
Server Interview – Orders & Trades

This module handles **orders with nested trades** and **real-time updates** for a Forex trading system.

The **screenshot** illustrates how orders and trades should appear use it as the basis for your schema and structure design.



Forex Orders & Trades

Order ID: 64f9abc123 PARTIALLY_FILLED
Pair: EUR/USD Type: BUY Amount: 1000 Price: 1.105
Filled: 750 / 1000
View Trades

Trade ID: 64f9abc123h24g	Executed: 250	Price: 1.105	14:25
Trade ID: 64f9abc123h25g	Executed: 500	Price: 1.106	14:30

Order ID: 64f9abc124 FILLED
Pair: USD/JPY Type: SELL Amount: 2000 Price: 145.20
Filled: 2000 / 2000
View Trades

Trade ID: 64f9abc123h27g	Executed: 2000	Price: 145.20	15:15

Requirements

Orders

- Contain details such as **pair, type, amount, price, status**.
- Have a list of **trades** beneath them.
- Support status progression:
  - `PENDING → PARTIALLY_FILLED → FILLED`

Trades

- Represent **individual executions** (amount, price, timestamp).
- Adding trades updates the order's **filled progress** and **status** automatically.

Business Rules

- The sum of all executed trades must not exceed the order's amount.
- If total filled `< amount` → `PARTIALLY_FILLED`.
- If total filled `= amount` → `FILLED`.
- If adding a trade would overshoot → reject (400).

API Endpoints (JWT required)

Method	Path	Purpose
POST	`/orders`	Create a new order
GET	`/orders`	List orders (filters: `userId`, `pair`, `status`; sort by `createdAt` desc)
GET	`/orders/:id`	Get a single order + its trades
PATCH	`/orders/:id/status`	Update status manually (e.g., set `CANCELLED`)
POST	`/orders/:id/trades`	Add a trade (updates order status automatically)

WebSocket (Real-time Updates)

- Gateway: `/ws/orders`
- Broadcast events whenever:
  - An order's **status changes**, or
  - A new **trade** is added.

Example payload:

```
{
  "orderId": "64f9abc123",
  "event": "TRADE_ADDED",
  "trade": {
    "tradeId": "64f9abc123h25g",
    "executedAmount": 500,
    "executedPrice": 1.106
  },
  "newStatus": "PARTIALLY_FILLED"
}
```

Simulation (Swagger/Postman)

1. **Create an order** (e.g., BUY EUR/USD 1000).
2. **Add trades** (e.g., 250 + 500 executed).
3. Observe **status updates** in API + **real-time events** in WebSocket.

Deliverables

Phase 1: High-Level Design (HLD)

Before implementation, provide a short HLD document covering:

- **Data modeling approach** (how you'll design Orders and nested Trades based on the UI).
- **Database schema strategy** (embedded trades vs separate collection, pros/cons).
- **API design flow** (how endpoints map to backend operations).

- **WebSocket update mechanism** (when and what to broadcast).

Phase 2: Implementation

- Implement Orders + Trades module.
- Add API endpoints + validation.
- Add WebSocket gateway + events.
- Test simulation with Swagger/Postman.