---

4. API Documentation (RESTful Standards)

You can import the following structure into **Postman** or use it as a reference for **Swagger UI**. All protected routes require a Bearer <JWT_TOKEN> in the header.

Authentication Module

Method	Endpoint	Description	Auth
POST	/api/v1/auth/register	Create a new user (default role: 'user')	Public
POST	/api/v1/auth/login	Returns JWT and user object (id, username, role)	Public

Trade Module

Method	Endpoint	Description	Auth
GET	/api/v1/trades	Fetch all trades (Users see theirs; Admins see all)	JWT
POST	/api/v1/trades	Create a new trade record in MySQL	JWT
DELETE	/api/v1/trades/:id	Remove a specific record by ID	JWT

Market Module

Method	Endpoint	Description	Auth
GET	/api/v1/stocks/price/:symbol	Fetches simulated/live price for a ticker	Public

---

5. Scalability & Architectural Note

To evolve **PrimeTrade** from a monolithic MVP to a high-traffic financial platform, the following scalability patterns should be applied:

1. Microservices Transition

Currently, the Auth and Trade logic share a single Express instance.

- **Strategy**: Decouple the **Market Data Service** (Yahoo API/Simulations) from the **Order Execution Service**.
- **Benefit**: If the market data service lags during high volatility, users can still log in and view their historical records without interruption.

2. Caching Strategy (Redis)

Live stock prices change every second, putting a heavy load on the MySQL database if every "Price Fetch" hits the disk.

- **Strategy**: Implement **Redis** to store stock prices with a short Time-To-Live (TTL) of 1–5 seconds.
- **Benefit**: Reduces MySQL read-query latency from ~50ms to <2ms.

3. Database Scaling

As the trades table grows into millions of rows:

- **Strategy**: Implement **Database Sharding** based on user_id or **Read Replicas**.
- **Benefit**: Ensures that the Dashboard remains snappy even when the global ledger is massive.

4. Load Balancing

- **Strategy**: Use an **Nginx** or **AWS ELB** load balancer to distribute incoming traffic across multiple instances of the Node.js backend.
- **Benefit**: Prevents a single server crash from taking down the entire trading platform.

Project Finalization Checklist

- [x] **Backend**: Express.js with JWT and Role Middleware.
- [x] **Database**: MySQL with relational integrity.
- [x] **Frontend**: Next.js with Role-Based Redirection.
- [x] **Functionality**: Live simulated Market Catalog with Buy/Delete operations.
- [x] **Security**: Redirection rules preventing Admins from User zones and vice-versa.