OVERVIEW
FINTECH

TrueCEX: Building a Centralized Crypto Exchange

Lev Ryskildin
l.ryskildin@innopolis.unversity

01 OBJECTIVES

01 Build a real-life CEX platform from scratch

02 Deliver features crucial for trading: multi-user support, authenticated actions, reliable order matching, and live market data.

03 Demonstrate professional code architecture with clear separation between frontend, backend, and data layers.

04 Scope is focused on core trading (matching engine, order book, user authentication, responsive design), intentionally deferring wallets, deposits, and compliance until later phases

02

ARCHITECTURE OVERVIEW

FRONTEND

React SPA (Single-Page Application), built with Vite, ensures a smooth user experience and real-time data via polling.

BACKEND

FastAPI asynchronous web server processes API requests, performs matching logic, and manages security/auth.

DATABASE

Uses SQLite for local/dev testing; easily upgraded to PostgreSQL for high concurrency and reliability in production.

03 ORDER MATCHING ENGINE

FIFO

TrueCEX employs a FIFO (First-In-First-Out) price-time priority matching algorithm

- Orders are sorted first by best price, then by earliest entry if prices are equal.
- Matching is triggered by every new order; fills are instant, and the system ensures fair execution for all users in the queue.
- The engine is built into the backend with clean async logic for speed and extensibility.

```python
async def get_orderbook(symbol: str, db: Session = Depends(get_db)):
    try:
        for order in open_orders:
            elif order.side == "sell":
                asks.append(order_level)

        # Sort bids DESCENDING by price (highest first)
        bids.sort(key=lambda x: x.price, reverse=True)

        # Sort asks ASCENDING by price (lowest first)
        asks.sort(key=lambda x: x.price)

        # Calculate spread = best_ask - best_bid
        best_bid = bids[0].price if bids else None
        best_ask = asks[0].price if asks else None

        spread = 0.0
        if best_bid is not None and best_ask is not None:
            spread = best_ask - best_bid

        return OrderbookResponse(
            symbol=symbol,
            bids=bids,
            asks=asks,
            spread=spread
        )
```

04 BACKEND

IMPLIMENTATION

- Built on FastAPI for high performance and async support.
- SQLAlchemy ORM manages three core models: User, Order, Trade, ensuring data integrity and transactional operations.

SECURITY FOCUSED

- User passwords hashed with bcrypt.
- Endpoints protected via JWT token authentication (stateless), expiring every 24h.
- API docs auto-generated for easy integration/testing

Authentication

POST /api/auth/register Register

POST /api/auth/login Login

Trading

POST /api/trading/orders Create Order

GET /api/trading/orders Get Orders

GET /api/trading/orders/{order_id} Get Order

DELETE /api/trading/orders/{order_id} Cancel Order

Wallet

GET /api/wallet/balances Get Balances

Market

GET /api/market/orderbook/{symbol} Get Orderbook

GET /api/market/ticker/{symbol} Get Ticker

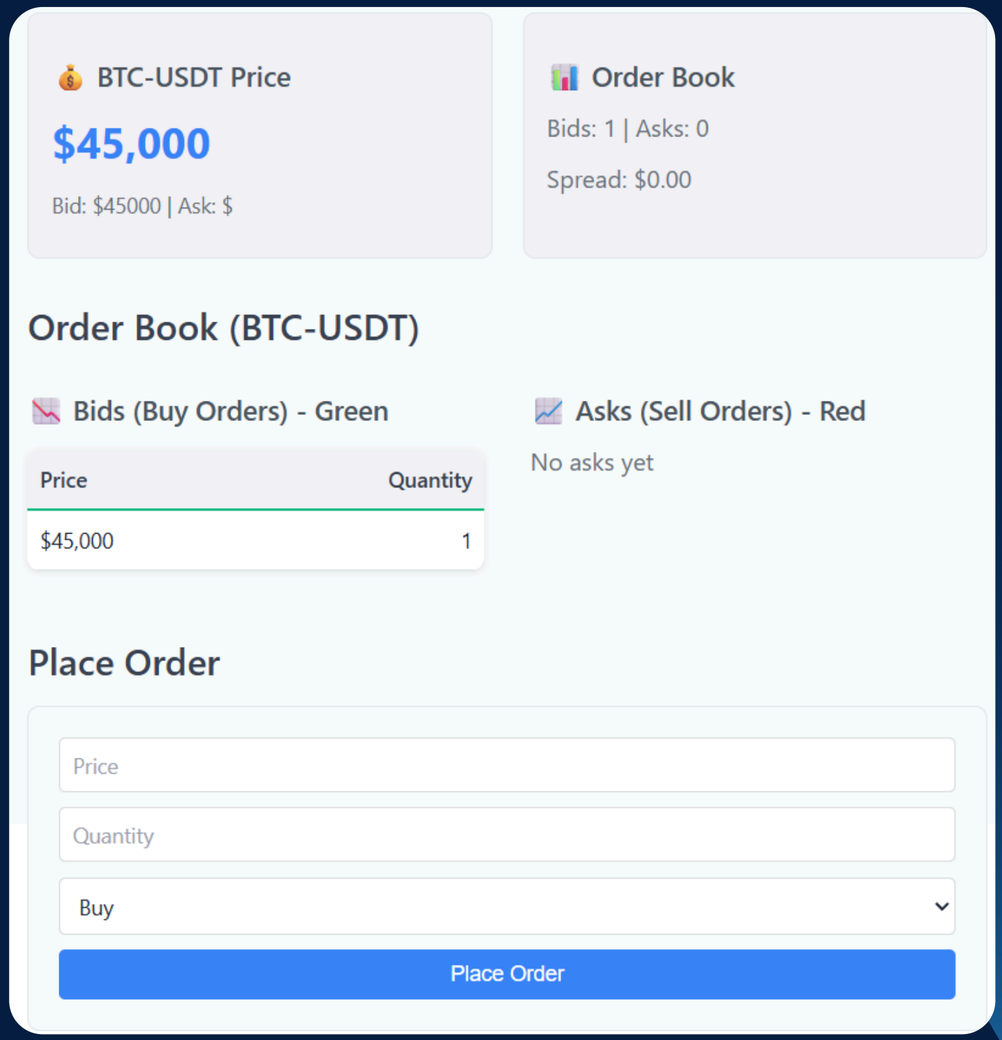
default

GET / Root

05 FRONTEND

Details

- Developed with React; all trading actions, order history, and live market data are presented in a clean, responsive UI.
- Data is fetched from the backend via REST endpoints, updating every 3 seconds to simulate real-time.
- State/feedback logic manages registration, authentication, trading, and error scenarios with user-friendly messaging.



FUTURE IMPROVEMENTS

- WALLET/BALANCE MANAGEMENT, FULL DEPOSIT/WITHDRAWAL FLOWS
- KYC/AML ONBOARDING AND COMPLIANCE FEATURES
- MOVE TO WEBSOCKET-BASED REAL-TIME DATA FOR <100MS LATENCY
- AUTOMATED LOAD/STRESS TESTING FOR SCALING VALIDATION
- IMPLEMENT DDOS/RATE LIMITING PROTECTIONS