**Phase 1 Implementation Checklist - Trading Platform Foundation**

**Environment Setup & Database Foundation**

**Development Environment**

* [ ] Set up version control (Git repository)
* [ ] Create project directory structure
* [ ] Set up C++ development environment and build system (CMake)
* [ ] Initialize Python virtual environment for FastAPI
* [ ] Set up React project with TypeScript template
* [ ] Configure development containers (Docker) for consistent environments

**Database Setup**

* [ ] Install PostgreSQL locally or set up cloud instance
* [ ] Install TimescaleDB extension for time-series optimization
* [ ] Create database schema for:
  + [ ] Stock metadata table (symbol, name, sector, exchange)
  + [ ] Historical price data table (OHLCV with timestamps)
  + [ ] Trading sessions table
  + [ ] Trades log table
* [ ] Set up Redis for caching layer
* [ ] Create database connection utilities in Python

**Data Pipeline & Storage**

**Data Source Integration**

* [ ] Register for API access (Polygon.io/Alpaca/Yahoo Finance)
* [ ] Create Python scripts for data fetching
* [ ] Implement rate limiting to respect API constraints
* [ ] Build data validation functions (check for missing data, outliers)

**Historical Data Loading**

* [ ] Download sample dataset (start with 10-20 stocks, 5 years history)
* [ ] Implement data cleaning routines:
  + [ ] Handle stock splits and dividends
  + [ ] Adjust for missing trading days
  + [ ] Normalize price data
* [ ] Create batch import process to PostgreSQL
* [ ] Verify data integrity with test queries
* [ ] Set up automated daily data updates

**Core Simulation Engine (C++)**

**Basic Engine Architecture**

* [ ] Design core classes:
  + [ ] Portfolio (track positions and cash)
  + [ ] Position (individual stock holdings)
  + [ ] Order (buy/sell instructions)
  + [ ] Market (price data access interface)
* [ ] Implement time-stepping mechanism for simulation
* [ ] Create order execution logic (simple market orders first)

**Account Management**

* [ ] Implement cash tracking and updates
* [ ] Create position tracking (shares owned, average price)
* [ ] Add basic transaction cost calculation (fixed commission)
* [ ] Implement simple portfolio value calculation

**Data Access Layer**

* [ ] Create C++ PostgreSQL connection interface
* [ ] Build efficient price data retrieval methods
* [ ] Implement "current date" restriction (no future data access)
* [ ] Add caching layer for frequently accessed data

**FastAPI Bridge Development**

**API Structure**

* [ ] Set up FastAPI project structure
* [ ] Create C++ binding using pybind11 or ctypes
* [ ] Design RESTful endpoints:
  + [ ] POST /simulation/start
  + [ ] GET /simulation/{id}/status
  + [ ] GET /simulation/{id}/results
  + [ ] GET /stocks/available

**Core Endpoints Implementation**

* [ ] Implement simulation initialization endpoint
* [ ] Create simulation execution wrapper
* [ ] Build results retrieval endpoint
* [ ] Add error handling and validation
* [ ] Create API documentation with FastAPI's auto-docs

**Moving Average Crossover Algorithm**

**Algorithm Implementation (C++)**

* [ ] Create TradingStrategy base class/interface
* [ ] Implement Moving Average calculation functions
* [ ] Build MA Crossover strategy:
  + [ ] Configure short/long period parameters
  + [ ] Implement buy signal (short MA crosses above long MA)
  + [ ] Implement sell signal (short MA crosses below long MA)
  + [ ] Add position sizing logic (fixed percentage)

**Integration with Engine**

* [ ] Connect strategy to simulation engine
* [ ] Implement strategy initialization
* [ ] Add daily strategy evaluation loop
* [ ] Create order generation from signals
* [ ] Test with sample data

**Basic React Frontend**

**Project Setup**

* [ ] Configure React Router for navigation
* [ ] Set up Axios for API communication
* [ ] Install charting library (Chart.js or Recharts)
* [ ] Configure Tailwind CSS
* [ ] Set up basic component structure

**Core Components**

* [ ] Create simulation setup form:
  + [ ] Starting capital input
  + [ ] Date range selector
  + [ ] Stock selection (multi-select)
  + [ ] Strategy parameters (MA periods)
* [ ] Build simulation progress indicator
* [ ] Implement results display page:
  + [ ] Final portfolio value
  + [ ] Total return percentage
  + [ ] Basic equity curve chart

**API Integration**

* [ ] Connect form submission to FastAPI
* [ ] Implement simulation status polling
* [ ] Handle and display API errors
* [ ] Create results fetching and parsing

**Integration & Testing**

**End-to-End Testing**

* [ ] Run complete simulation flow test
* [ ] Verify data accuracy at each step
* [ ] Test edge cases (market crashes, no trades)
* [ ] Performance test with larger datasets

**Bug Fixes & Refinements**

* [ ] Fix identified issues from testing
* [ ] Optimize slow database queries
* [ ] Improve error messages
* [ ] Add logging throughout system

**Documentation**

* [ ] Write setup instructions
* [ ] Document API endpoints
* [ ] Create simple user guide
* [ ] Add code comments for complex logic

**Deliverables Checklist**

**Must Have for Phase 1 Completion**

* [ ] Working database with 5+ years of historical data for 10+ stocks
* [ ] C++ simulation engine executing basic trades
* [ ] FastAPI successfully bridging frontend to backend
* [ ] React UI allowing simulation setup and results viewing
* [ ] One working strategy (MA Crossover) with configurable parameters
* [ ] Basic performance metrics (total return, final value)

**Nice to Have (if time permits)**

* [ ] Docker compose for easy deployment
* [ ] **Comprehensive Testing**: Unit tests, integration tests, performance benchmarks
* [ ] **Documentation**: API docs, architecture diagrams, user guides
* [ ] Simulation history/saved results
* [ ] Export results to CSV
* [ ] Loading states and progress bars

**Success Criteria**

* [ ] Can run a 1-year backtest on 5 stocks in under 10 seconds
* [ ] Results are reproducible (same inputs = same outputs)
* [ ] No crashes during normal operation
* [ ] UI is intuitive enough for basic use without documentation