Absolutely — here’s your **complete, high-level blueprint** for the **Smart Trading & Wallet Management System**, covering **every module, entity, structure, flow, API logic, chatbot integration, technologies**, and architectural details — **without code**, just written concept and system design.

# 📊 ****Smart Trading & Wallet Management System – Full Project Structure****

## 🎯 ****Project Objective****

A desktop-based Java application that allows users to:

* **Manage a digital wallet**
* **Buy and sell stocks or cryptocurrencies**
* **Track their portfolio and transaction history**
* **Use a chatbot for updates and trading help**
* **Fetch live market data via APIs**
* **Enable admins to manage users and monitor activities**

## 🧱 ****High-Level Architecture****

📦 Model (Entity Classes)

 Controller (JavaFX Controllers)

 Service (Business Logic)

 DAO (Database Access Layer)

 Utils (Helpers, Formatters, Parsers)

 View (FXML UI Screens)

 Assets (CSS, Images)

All follow **MVC pattern**:

* **Model** → Defines the data structures
* **View** → FXML + JavaFX UI
* **Controller** → Bridges UI and logic
* **Service + DAO** → Business logic and DB interaction

## 📁 ****Full Project Folder Structure****

SmartTradingWalletSystem/

├── src/

│ ├── model/ // All data classes: User, Wallet, Stock, etc.

│ ├── controller/ // JavaFX controllers

│ ├── dao/ // DB access objects (UserDAO, WalletDAO)

│ ├── service/ // Logic like buying/selling, balance updates

│ ├── utils/ // Utilities: Alerts, API parser, JSON handlers

│ ├── view/ // FXML UI screens

│ ├── config/ // DB connection

│ └── Main.java // App entry point

├── assets/ // Images, CSS files

├── lib/ // External libraries (Gson, MySQL JDBC, etc.)

├── database/ // SQL schema (smartwallet.sql)

├── README.md

└── requirements.pdf

## 📚 ****Entities & Relationships****

### ✅ User

* Attributes: userId, name, email, password, role (enum), wallet, portfolio
* Linked to: Wallet, Portfolio
* Roles: USER, ADMIN

### ✅ Wallet

* Attributes: walletId, userId (FK), balance, transaction list

### ✅ Transaction

* Attributes: transactionId, userId (FK), type (enum), amount, date, asset info

### ✅ Portfolio

* Attributes: portfolioId, userId (FK), List of owned assets with quantity

### ✅ Asset (abstract)

* Attributes: assetId, name, currentPrice, assetType (enum)

### ✅ Stock & CryptoAsset (inherit Asset)

* Stock: +exchange
* CryptoAsset: +blockchain

### ✅ Admin (inherits User)

* Additional controls to manage users and monitor data

## ⚙️ ****Functional Modules****

### 1. ****Authentication****

* Sign up / Login
* Session tracking for logged-in user

### 2. ****Wallet Management****

* Add funds, withdraw
* Transaction recording
* Wallet balance calculation

### 3. ****Trading Engine****

* Buy/Sell assets from list
* Validations: balance check, quantity check
* Record transaction and update portfolio

### 4. ****Portfolio Viewer****

* List of all owned stocks/crypto
* Quantity, average cost, current value
* Gain/Loss overview
* Visualized with **JavaFX charts**

### 5. ****Transaction History****

* Filter by date/type
* View total spent, earned, gains

### 6. ****Market Data Integration****

* Fetch real-time data via **API**
* Live prices, 24h % change, top gainers

## 🌐 ****How APIs Work (Market Data)****

### 🔗 Used APIs:

* **CoinGecko**: For real-time crypto data (free, simple)
* **Alpha Vantage** or **Yahoo Finance**: For stock data

### 🔁 Workflow:

1. User selects an asset (e.g., BTC or AAPL)
2. APIService sends HTTP request to API endpoint
3. JSON response is parsed (via Gson or Jackson)
4. Live price is updated in the UI + stored in Asset object
5. This data is used during trades and charting

## 🤖 ****How the ChatBot Works****

### 🔧 Purpose:

* Provide guidance, updates, and suggestions to the user
* Chat-style interaction with buttons or typed text

### 📦 Structure:

* Class: ChatBot
* Uses: Rule-based keyword matching (or OpenAI API if online)

### 🗣️ What It Can Do:

* "What’s the price of BTC?" → Fetches live API price
* "How much profit did I make?" → Calculates gains from portfolio
* "What can I buy with $100?" → Lists matching assets
* "Show my top asset" → Ranks portfolio by value

### 🔁 Workflow:

1. User asks a question in chat window
2. ChatBot parses question → identifies intent
3. Calls appropriate logic:
   * MarketAPIHandler.getLivePrice("BTC")
   * Portfolio.calculateProfit()
4. Returns a reply → displayed in chat UI

## 🛠 ****Admin Features****

* View all registered users
* View all wallet balances and transactions
* Generate reports (PDF/export optional)
* Freeze/unfreeze accounts (optional)

## 📊 ****UI Screens (FXML + JavaFX)****

| Screen | Purpose |
| --- | --- |
| login.fxml | Login form |
| dashboard.fxml | Main user home screen |
| wallet.fxml | Wallet balance, add/withdraw funds |
| trade.fxml | Buy/sell assets, live market list |
| portfolio.fxml | View owned assets, graphs |
| transactions.fxml | List of all transactions |
| chatbot.fxml | Chat window with smart assistant |
| admin.fxml | Admin dashboard |

## 🔧 ****Tech Stack****

| Layer | Technology Used |
| --- | --- |
| UI | JavaFX, FXML, CSS |
| Logic | Core Java (OOP), MVC |
| DB | MySQL or SQLite + JDBC |
| API | CoinGecko, Alpha Vantage, Yahoo Finance |
| JSON Parser | Gson or Jackson |
| Chatbot | Custom NLP engine (or OpenAI integration) |
| Charts | JavaFX LineChart / BarChart |
| Security | AES encryption (optional) for password/wallet |

## 📈 Key OOP Concepts Used

* **Abstraction**: Abstract Asset class, DAO interface
* **Inheritance**: Stock and CryptoAsset extend Asset; Admin extends User
* **Polymorphism**: Use of List<Asset> to store both types
* **Encapsulation**: Private fields + getter/setter in all entities
* **Composition**: User has Wallet, Portfolio

## 🧠 Optional Enhancements

* 🔐 OTP verification for big transactions
* 📄 Export portfolio report as PDF
* 🔔 Set price alerts with notifications
* 📈 Use sparkline charts for real-time asset price trends
* 🎓 Add a mini quiz/training module for new traders

## ✅ Conclusion

Your project is:

* **Well-structured**
* **Technically rich**
* **OOP-aligned**
* **Innovative with chatbot & real-time data**
* **Highly scalable** if you want to take it beyond the semester

Would you like me to generate:

* A **UML class diagram**
* A **use-case diagram**
* A **requirement doc**  
  to attach with your project files for final submission?