Stock Watch Project: Step-by-Step Development Plan

# Step 1: Define MVP Scope

- Confirm core features: watchlist, price alert on threshold, basic UI  
- Lock in tech stack for MVP:  
 - Backend: FastAPI (or Flask)  
 - Data: yfinance or Finnhub API  
 - UI: Gradio for rapid prototyping  
 - Storage: In-memory or lightweight JSON file (upgrade later)

# Step 2: Build Core Backend

- Create RESTful API endpoints to:  
 - Add/remove stock tickers  
 - Set/change alert thresholds  
 - Fetch current stock data  
- Tech stack:  
 - FastAPI/Flask  
 - Python + yfinance/finnhub  
 - uvicorn for local testing

# Step 3: Implement Alert System

- Add background job to poll stock data at intervals (e.g., every 5 mins)  
- Compare against threshold  
- Trigger alert (for now: console print or Gradio message)  
- Tech stack:  
 - `schedule`, `apscheduler`, or `asyncio` loop in Python

# Step 4: Integrate Simple Gradio UI

- Build forms to:  
 - Enter new stock symbols  
 - Set percentage thresholds  
 - View current prices and alerts  
- Tech stack:  
 - Gradio  
 - Python for state backend

# Step 5: Add AI-Powered News Summarization (Optional Phase 1)

- Pull headlines using external APIs (e.g., NewsAPI, Finnhub news)  
- Use OpenAI or Hugging Face to summarize news related to tracked stocks  
- Display summarized news in UI  
- Tech stack:  
 - OpenAI API (text-davinci or gpt-3.5-turbo)  
 - LangChain (optional)  
 - News API or Web scraping (fallback)

# Step 6: Paper Trading & Performance Logging

- Simulate a trading engine that logs 'buy/sell' actions triggered by AI insights or price changes  
- Track performance over time  
- Tech stack:  
 - Python logging or SQLite DB  
 - Simple trade strategy logic (threshold-based + news signal)

# Step 7: Optional React Frontend

- Replace Gradio with a full frontend  
- Build dashboard with stock charts, alert logs, news summary  
- Tech stack:  
 - React + Tailwind CSS  
 - Chart.js or Recharts  
 - API communication with backend via Axios/Fetch

# Step 8: Deployment & Monitoring

- Deploy backend on Render, Railway, or Heroku  
- Set up cron or background worker for polling  
- Add uptime monitoring (e.g., UptimeRobot)  
- Tech stack:  
 - GitHub Actions for CI/CD  
 - Docker (optional)  
 - Vercel (if React frontend)  
 - Render/Heroku for backend

# Step 9: Future Enhancements

- Real trading integration via brokerage API (Robinhood, Alpaca)  
- Reinforcement learning to tune trading strategies  
- Sentiment analysis from Reddit, Twitter, etc.  
- Mobile app version