This is a significant architectural pivot from a personal finance tracker to a sophisticated, institutional-grade **Wealth Management System (WMS)**. You are moving from "tracking where money went" to "optimizing where money grows."

As your polymath engineer partner, I have designed the architecture for this system. It strictly adheres to your requirement for high accuracy, security, and specific asset class handling (Indian context: G-Secs, SDLs, etc.).

We will build the **"Fortress Wealth Engine"**.

System Architecture Overview

To handle ₹1 Cr+ portfolios with high precision, we cannot rely on simple rules. We need a **Hybrid-AI Core**:

1. **Deterministic Engine (The Safety):** Hard mathematical rules for Bonds and fixed income (calculating Yield-to-Maturity, Post-Tax Returns).
2. **Probabilistic Engine (The Growth):** AI/ML models for Stocks and Commodities (predicting volatility, sentiment, and trends).

The 5 Core Modules (MVP Definition)

Here are the top 5 essential modules to build this system, prioritized for value and security.

Module 1: The "Wealth Architect" (Allocation Engine)

This is the brain. It takes the user's input (₹1 Cr, 1.5L return/year, 5-year horizon) and determines the "Perfect Split."

- **Logic:** Uses **Modern Portfolio Theory (MPT)** optimized for Indian Inflation. It doesn't just guess; it solves an equation.
- **The 5 Risk Profiles (Hardcoded Logic for MVP):**
    - **Most Secure (The Fortress):** 90% G-Sec/SDL Bonds, 10% Gold. (Goal: Capital Preservation).
    - **Secure:** 70% AAA Corporate/PSU Bonds, 20% Blue-Chip Stocks, 10% REITS.
    - **Moderate:** 40% Bonds, 40% Large/Mid-Cap Stocks, 20% Gold/Commodities.
    - **Risk:** 20% Bonds, 60% Mid/Small Stocks, 20% Forex/Alt-Assets.
    - **High Risk (Alpha Hunter):** 10% Cash, 70% Small-Cap/Momentum Stocks, 20% Crypto/Forex.
- **Tech Stack:** Python (NumPy, SciPy), PyPortfolioOpt library (industry standard for mean-variance optimization).
- **MVP Feature:**
    - Input: "I have ₹1 Cr, need ₹1.5L/year, Keep it Level 1 (Secure)."
    - System Calculation: ₹1.5L/year is 1.5% return. This is trivial. The system effectively finds the safest assets that yield *at least* this + inflation.

    - Output: "Allocating ₹90L to G-Secs (7.2% yield) and ₹10L to Gold. Total Projected Return: ₹7.4L/year (Exceeds target safely)."

Module 2: The "Bond & Fixed Income Scanner" (Safety Layer)

Deeply analyzes the Indian Debt Market as per your request.

- **Data Sources (MVP):** Since real-time bond data is expensive, scrape data from **CCIL (Clearing Corporation of India)** or use RBI public releases for G-Sec yields. For Corporate bonds, scrape trusted financial news aggregators.
- **The Algorithm:**
    - **G-Secs/SDLs:** Calculate Real Yield (Yield - Inflation).
    - **Corporate/PSU:** Check Credit Rating (AAA, AA+) and "spread" over G-Secs. If a Corporate Bond pays less than a G-Sec, the AI rejects it (Reward < Risk).
- **Security Feature: "Default Risk Checker."**
    - If a PSU Bond (e.g., REC) is selected, the system pulls the last 3 years of financial reports (simulated or scraped) and checks the **Debt-to-Equity Ratio**.
- **Tech:** Python pandas for data crunching, BeautifulSoup for scraping yields.

Module 3: The "Equity Intelligence" (Stock Selector)

Upgrading your existing Stock module to handle Cap-size and Industry logic.

- **Step 1: Macro-Filter (Industry):**
    - The AI scans sectors (Auto, Pharma, IT). It uses **Sector Rotation Theory** (e.g., "In high inflation, buy Energy/FMCG").
- **Step 2: Micro-Filter (Stock Picking):**
    - **Blue Chip:** Filters for Market Cap > ₹20,000Cr, Dividend Yield > 1%, Beta < 1.
    - **Mid/Small Cap:** Filters for high Revenue Growth (>15% YoY).
- **Step 3: AI Validation (Sentiment):**
    - Uses your **FinBERT** module [1111] to ensure no negative news exists for the selected stock.
- **MVP Feature:** A "Stock Screener" that accepts the user's allocated equity budget and returns a list of 5-10 stocks fitting their risk profile.

Module 4: The "Deep Horizon" Simulator (Forecasting)

Validates if the user's goal is mathematically possible.

- **Function:** Before the user invests, run a **Monte Carlo Simulation**[2].
- **Scenario Testing:**
    - "What if another COVID happens?" (Crash Market by 30%).
    - "What if Inflation hits 8%?"
- **Output:** A simple "Success Probability" score.
    - *Example:* "With your request for ₹1.5L/year on ₹1 Cr, you have a **99.9% Success Probability** in 'Most Secure' mode."

- **Tech:** Python (NumPy), plotting with Matplotlib or sending JSON data to the Frontend (Recharts/D3.js).

Module 5: The "Secure Vault" Dashboard & Workflow

The User Interface where the magic happens.

- **Workflow:**
    1. **Onboarding:** User selects Risk Level (1-5), Capital, and Horizon.
    2. **Proposal:** System generates a PDF/Web View of the "Wealth Plan" (e.g., "Buy these 3 Bonds, these 5 Stocks").
    3. **Execution (MVP):** Since we cannot auto-trade legally without a license, we provide **"One-Click Instructions."**
        - *Display:* "Login to Zerodha/Upstox. Search 'NHAI N5'. Buy 500 Qty."
    4. **Monitoring:** The dashboard tracks the *net worth* and *income generated* vs the Target.
- **Tech:** Next.js + Tailwind (from your file [3]), PostgreSQL for user data[4].

Technical Deep Dive: High Security & Accuracy

Since this involves money, "Accuracy" and "Security" are the top priority.

1. Data Accuracy Strategy (The "Oracle" Pattern)

We cannot trust a single data source.

- **Triangulation:** For stock prices, fetch data from yfinance [5] AND a secondary free source (e.g., Google Finance scraper). If they differ by >1%, flag the data as "Unreliable" and do not display it.
- **Bond Math:** Do not use APIs for yield calculations. Fetch the *Price* and *Coupon Rate*, then calculate Yield-to-Maturity (YTM) locally in Python using scipy.optimize.newton. This ensures the math is 100% correct regardless of external display errors.

2. Security Architecture

- **Zero-Trust Database:**
    - User financial data (Net Worth, Holdings) is encrypted at rest in PostgreSQL using AES-256.
    - Even you (the admin) cannot read their portfolio value without their private key (in a real production app). For MVP, standard hashing and ENV variable protection is sufficient.
- **Sanitized Inputs:** All inputs (Risk Level, Capital Amount) pass through strict Pydantic models in FastAPI [6] to prevent injection attacks.

MVP Working Flow (User Journey)

1. **Login:** User logs in via secure Auth[7].
2. **The "Interview":**
    - *Q: Capital?* -> A: ₹10,000,000 (1 Cr).
    - *Q: Goal?* -> A: ₹150,000 / year (Passive Income).
    - *Q: Risk Tolerance?* -> A: Level 1 (Most Secure).
    - *Q: Time Horizon?* -> A: 5 Years.
3. **Engine Processing:**
    - System calculates required yield: 1.5%.
    - System scans G-Secs (Current Yield ~7.1%).
    - System Logic: "Target is very low. Risk is minimized. 100% Allocation to High-Safety Bonds suggested to avoid Equity volatility."
4. **The Proposal Page:**
    - **Headline:** "Strategy: The Iron Fortress"
    - **Allocation:**
        - **90%**: GOI Bond 2030 (Yield 7.2%) - Secure sovereign backing.
        - **10%**: SGB (Sovereign Gold Bond) - Hedge against currency devaluation.
    - **Projected Income:** ₹720,000 / year (Far exceeding the ₹1.5L request).
5. **Tracking:** User manually enters "I bought these". The dashboard now tracks daily value using the daily closing price of the bond/gold.

Next Step for You

You have the raw code components from the previous file. To turn this into the Wealth System:

Would you like me to generate the wealth_architect.py (Module 1) code?

I can write the Python logic that takes capital, risk_level, and horizon as inputs and mathematically outputs the specific asset allocation percentages based on Indian Market standards.

This is the "Expansion Pack" for your **Fortress Wealth Engine**.

While the first 5 modules focused on **Building & Allocating** wealth, these next 5 modules focus on **Optimization, Protection, and Withdrawal**. This completes the ecosystem, turning it from a simple "Investment App" into a "Family Office" grade system for HNWIs (High Net Worth Individuals).

Here are Modules 6–10, designed with high accuracy for the Indian financial context.

Module 6: The "Tax Shield" (Tax Harvesting Engine)

The Wealth Multiplier. It ensures the user keeps what they earn.

- **The Problem:** A 15% return becomes 10% if taxes (STCG/LTCG) are mismanaged.
- **Core Logic (Indian Context):**
    - **LTCG Harvesting:** In India, ₹1.25 Lakh of Long Term Capital Gains (LTCG) from equity is **tax-free** every year. The system must track this.
    - **Loss Harvesting:** If the user has a loss in "Stock A", the system suggests selling it to offset profits in "Stock B", reducing total tax liability.
- **Algorithm: FIFO (First-In-First-Out) Logic**.
    - It tracks every single "Buy" transaction date.
    - If (Current Date - Buy Date) > 365 days AND Unrealized Profit < ₹1.25L, it triggers an alert: *"Sell and Re-buy immediately to reset your tax base at zero cost."*
- **Tech Stack:** Python (pandas for date math), PostgreSQL (to store transaction lots).
- **MVP Feature:**
    - **"Tax-Free Meter":** A progress bar showing how much of the ₹1.25L exemption has been used this year.
    - **Action:** One-click report generation for CA filing.

Module 7: The "Realty Commander" (REITs & Land)

Addressing your request for "Land" and Real Estate.

- **The Problem:** Physical land is illiquid and hard to value.
- **The Solution:** A hybrid module managing both Physical Land (manual tracking) and **REITs (Real Estate Investment Trusts)** for liquid exposure.
- **Core Logic:**
    - **Physical Land:** User enters "Bought at ₹50L in 2015". System applies local "Circle Rate" inflation or average City CAGR (e.g., 6%) to estimate current value.
    - **REITs (Embassy, Mindspace, Nexus):** The system treats these as high-yield dividend assets (5-7% yields).
- **Algorithm: Rental Yield vs. Bond Yield Analyzer**.
    - It calculates: (Monthly Rent * 12) / Property Value.
    - If Rental Yield (2.5%) < Risk-Free Bond Rate (7%), it advises: *"Property is underperforming. Consider selling and moving to REITs/Bonds."*
- **Tech Stack:** Simple Calculation Engine (Python), Web Scraper for REIT prices.

Module 8: The "Income Streamer" (SWP Engine)

Addressing your specific request: "He wants 1.5L per year."

- **The Role:** This is the most critical module for your specific user case. It ensures cash flow without eating into the principal capital.
- **Algorithm: The "Bucket Strategy" (3-Layer Waterfall)**
    - **Bucket 1 (Immediate Income):** Holds 1 Year of expenses (₹1.5L) in **Liquid Funds** or High-Yield Savings (Stable, Instant Access).

    - **Bucket 2 (Stability):** Holds 5 Years of expenses in **Short-Duration Bonds/FDs** (Moderate Growth, Low Risk).
    - **Bucket 3 (Growth):** The rest (₹80L+) stays in **Equity/Gold** to beat inflation.
- **Automation:**
    - The system auto-alerts once a year: *"Transfer profits from Bucket 3 to Bucket 1 to refill your yearly allowance."*
- **MVP Feature:**
    - **"The Paycheck" Button:** The user sets "I need ₹12,500/month". The system highlights exactly which asset to sell to get this cash most efficiently (e.g., sell the asset that has the *least* tax impact).

Module 9: The "Global Hedge" (Gold & Commodities)

Addressing the "Commodity Market" requirement.

- **The Role:** Insurance against the Indian Rupee (INR) failing or crashing.
- **Core Logic: Correlation Matrix**.
    - Gold/Silver often move *inversely* to the Stock Market.
    - When the NIFTY index falls, Gold usually rises.
- **Algorithm:**
    - The system monitors the **Portfolio Beta**. If the portfolio is too correlated to the stock market (Risk > High), it forces a recommendation: *"Buy SGB (Sovereign Gold Bond) or Gold ETFs to stabilize."*
    - **SGB Tracking:** Specific logic for SGBs, tracking their 2.5% extra interest payout dates.
- **Tech Stack:** yfinance for Gold prices (XAU/USD converted to INR).

Module 10: The "Sentinel" (Risk Watchtower)

The "High Security" Guardian.

- **The Problem:** A "Secure" Bond can become "Risky" overnight if the company fails (e.g., DHFL crisis).
- **The Solution:** An active monitoring system that scrapes credit ratings.
- **Core Logic:**
    - **Credit Watch:** Scrapes news/rating agency sites (CRISIL, ICRA, CARE) for the user's bond holdings.
    - **Trigger:** If a holding is downgraded from **AAA** to **AA**, the system sends a **Critical Push Notification**: *"Security Alert: REC Bond downgraded. Risk profile increased. Recommend Review."*
- **Stop-Loss Logic:** For "High Risk" equity portfolios, if a stock falls 15% from the buy price, the Sentinel flags it for immediate review to prevent total loss.
- **Tech Stack:** Python Scrapers (BeautifulSoup), FCM (Firebase Cloud Messaging) for Push Notifications.

Summary: The Full 10-Module Ecosystem

Here is how the flow works for your user with **₹1 Cr wanting ₹1.5L/year**:

1. **Wealth Architect (Mod 1):** Allocates the capital. Since the requirement (1.5%) is low, it puts 95% in Safe Bonds/G-Secs.
2. **Bond Scanner (Mod 2):** Selects the specific "Government of India 2034" bonds.
3. **Income Streamer (Mod 8):** Sets up a "Payout Schedule" to release ₹12,500/month to the user's bank account from the interest.
4. **Tax Shield (Mod 6):** Ensures this interest income is structured efficiently.
5. **Sentinel (Mod 10):** Watches the bonds 24/7 to ensure the Government/PSU rating stays stable.