**INVEST KARO**

**"The app where learning meets real trading."**

**Table of Contents**

1. Problem Statement
2. Need and Significance
3. System Requirement
4. Software Development Cycle with “Increment Model”
5. Data Flow Diagram
6. Data Dictionary
7. Use Case Approach
8. Sequence Diagram
9. Timeline Chart
10. Function Point Estimation

**Problem Statement** :

These days, stock market trading is not for institutional investors or big investors alone. Individual investors and small investors are actively trading and managing their portfolios too. But most of the available platforms are either too complicated, expensive, or need internet connectivity with live servers. For teaching and training purposes, an easy-to-use, offline simulation system for stock trading is necessary to enable users to learn the process of registration, buying and selling, and portfolio management in a secure environment.

The project addresses this requirement by creating a Tkinter-based desktop application to mimic elementary stock market operations in Python. Rather than being connected to live stock exchanges, it holds and reads data locally from JSON files (users.json and registrations.json). This makes it suitable for students, new users, and instructional use.

**Need of the System**

* To develop a simple and easy-to-use platform for mimicking stock market transactions.
* To enable registration, login, and secure management of accounts by users.
* To enable buying and selling of shares across industries (Automobile, Petroleum, Steel).
* To keep activity records of users in JSON-based databases to make data management convenient.
* To have an Admin Panel to manage and view all registered users.

**Importance of the software**

Educational Value – Enables students to grasp stock market principles without involving actual money.

Data Handling – Exhibits hands-on application of file handling and JSON data manipulation in Python.

GUI Learning – Introduces the students to Tkinter GUI programming.

Security & Validation – Enforces login, registration, and admin authentication functionality.

Scalability – The system can expand to add additional sectors, live APIs, and complex analytics in the future.

**Technologies Used**

**Front-End Technology**

* **Python Tkinter**
  + Used for designing the **Graphical User Interface (GUI)**.
  + Provides windows, labels, buttons, entry fields, and other widgets.
  + Makes the application user-friendly for registration, login, buy/sell, and admin management.
* **PIL (Pillow Library)**
  + Used to load and display images (like logos and icons) inside the GUI.
* **WebBrowser module (optional)**
  + Opens stock information links (Honda, Hyundai, Reliance, ONGC, etc.) directly in a web browser when user clicks on a company.

**Back-End Technology**

* **Python (Core Logic)**
  + Handles business logic like registration, login authentication, buying/selling shares, updating balances, etc.
* **JSON Files** (users.json and registrations.json)
  + Used as the **Database (Data Storage)**.
  + users.json → Stores user details, balance, and share portfolio.
  + registrations.json → Stores registration records for admin viewing.
* **File Handling in Python**
  + Reads and writes JSON files for saving and loading user data.

**Additional Technologies**

* **Operating Environment**: Runs on **Python 3.x** (cross-platform).
* **Libraries Used**:
  + tkinter → GUI
  + PIL (Image, ImageTk) → Image handling
  + json → Data storage and retrieval
  + os → File management
  + webbrowser → Open stock links

**Software Requirements Specification (SRS)**

**Project Name:** Invest Karo – Stock Market Simulator

**1. Introduction**

**1.1 Purpose**

InvestKaro is a desktop application that simulates stock market trading for **learning and fun**.  
Users can register, log in, trade in four sectors (**Automobile, Petroleum, Steel, Gold**), and track their portfolios.  
The app also provides candlestick charts for stock prices and an **admin dashboard** to manage users.

**1.2 Scope**

The system includes the following features:

* **User registration and login**
* **Buy/Sell stocks** across four sectors
* **Portfolio tracking and valuation**
* **Candlestick chart visualization**
* **Admin dashboard** for user monitoring
* **Data storage in JSON files** (users.json, registrations.json)
* **User-friendly interface** built with Tkinter

**1.3 Definitions, Acronyms, Abbreviations**

* **SRS** – Software Requirements Specification
* **GUI** – Graphical User Interface
* **JSON** – Data storage format (used instead of CSV)
* **Tkinter** – Python GUI library

**2. Overall Description**

**2.1 Product Perspective**

Invest Karo is a **standalone desktop application** built with Python and Tkinter.  
It stores data in JSON files and provides separate **user and admin interfaces**.  
The goal is to help users **learn stock trading concepts** safely without real money.

**2.2 User Needs**

* **New Users:** Quick sign-up and starting balance
* **Existing Users:** Simple login, easy access to trading, portfolio, and charts
* **All Users:** Clear insights into investment performance
* **Admin Users:** Tools to view, manage, and export user data

**2.3 Assumptions**

* Users understand basic stock market concepts
* Python and required libraries (Tkinter, PIL, pandas, mplfinance, etc.) are installed
* No real money or brokerage integration is involved

**3. System Features and Requirements**

**3.1 Functional Requirements**

1. **User Registration (Sign Up):** Validate details, save user data, assign starting balance
2. **Authentication (Login):** Verify credentials, notify on success/failure
3. **Portfolio Management:** Track holdings in four sectors with current value
4. **Buy/Sell Transactions:** Update holdings and balance after each trade
5. **Candlestick Charts:** Show OHLC price trends, profits/losses
6. **Admin Dashboard:** View users, balances, and portfolios (admin-only access)
7. **Persistent Data Storage:** Store all data in JSON files with integrity

**3.2 Non-Functional Requirements**

* **Performance:** Responsive under normal load
* **Security:** Protect passwords and admin access
* **Usability:** Simple and intuitive interface
* **Reliability:** Handle errors and invalid operations smoothly
* **Maintainability:** Modular and well-documented code

**4. External Interface Requirements**

**4.1 User Interfaces**

* Tkinter-based GUI with tabs for registration, login, trading, portfolio, and admin dashboard

**4.2 Hardware Interfaces**

* Runs on any desktop/laptop that supports Python and GUI libraries

**4.3 Software Interfaces**

* No external APIs used
* Local file storage (users.json, registrations.json)

**4.4 Communication Interfaces**

* Fully offline, no internet/network needed

**5. Other Requirements**

**5.1 Security**

* Minimum password length required
* Admin access secured with password

**5.2 Risks and Limitations**

* **Simulation only** – not for real investments
* No multi-user or network features

**5.3 Future Enhancements**

* Connect with real stock market APIs
* Add more asset categories
* Export data in CSV/Excel formats

**6. Appendices**

**6.1 Glossary**

* **Buy/Sell:** Simulated trade operations
* **Sector:** Group of assets (Automobile, Petroleum, Steel, Gold)

**6.2 References**

* Standard SRS templates and trading software examples

**6.3 Use Cases**

| **Use Case** | **Actor** | **Description** |
| --- | --- | --- |
| **Sign Up** | User | Creates account and gets starting bonus |
| **Login** | User | Logs in with email and password |
| **Trade Stocks** | User | Buys/Sells shares, updates portfolio |
| **View Portfolio** | User | Displays current holdings and value |
| **Admin Access** | Admin | Views all users and portfolios |

**Software process model pertaining for Invest Karo**

The Incremental Software Process Model was chosen for this project because the application was developed in small functional parts, or increments, rather than as one large system. Each module (Registration, Login, Buy/Sell, Admin Panel, Portfolio Management) was designed, implemented, tested, and integrated step by step.

This approach was most suitable because:

**Step-by-Step Development**

First increment: Basic Registration and Login System.

Second increment: JSON-based database integration.

Third increment: Buying and Selling of shares.

Fourth increment: Portfolio management and balance update.

Fifth increment: Admin Panel and stock links integration.

**NOTE: It is evident from the above discussion that step-by-step function modules will be developed and tested separately, and integrated into a complete system. Therefore, Incremental model is best suited for the proposed project.**

**Flexibility**

This approach allowed for adding features, such as web browser links for stock info or sector-wise companies, without redesigning the entire system.

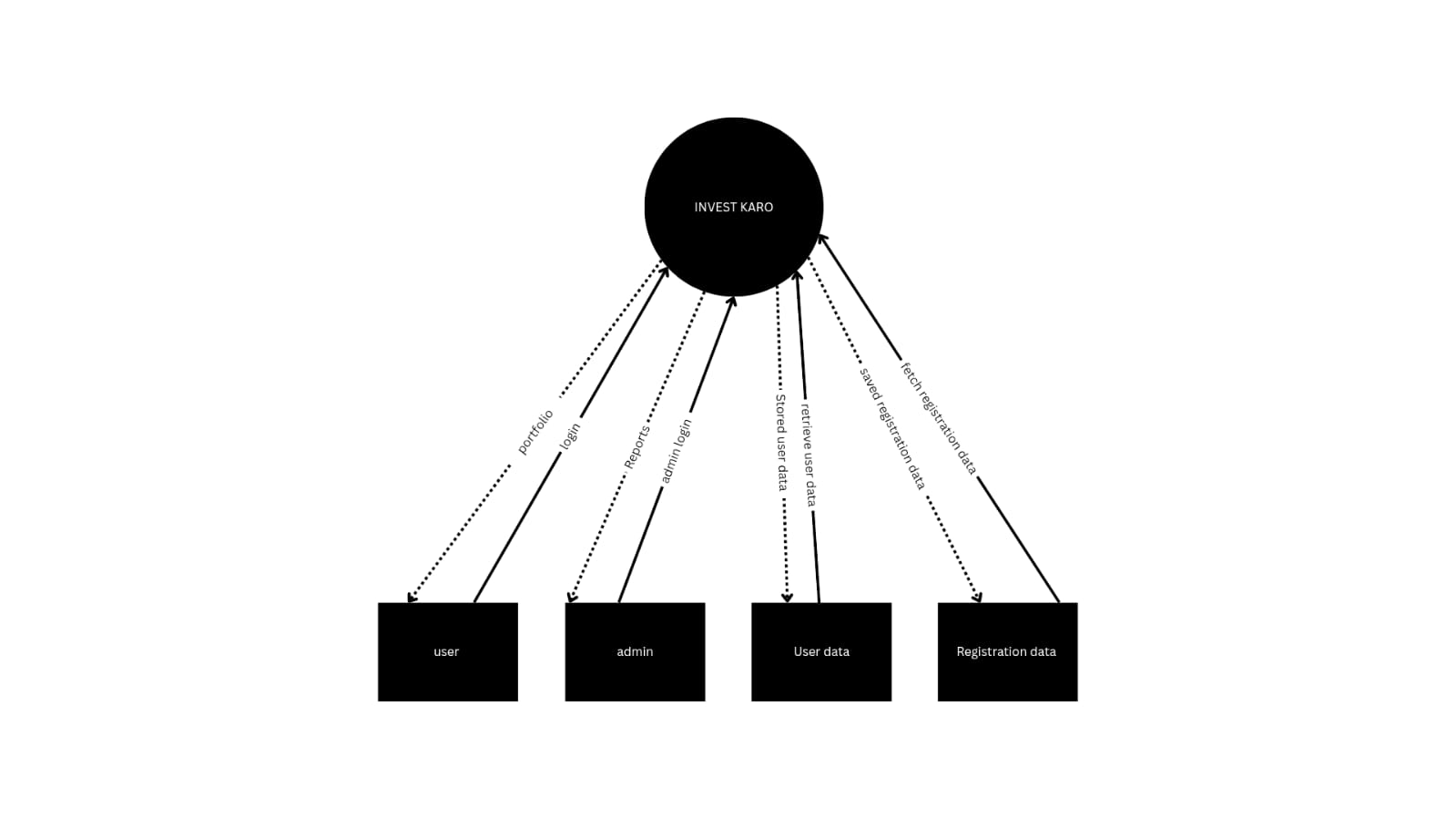
**User Feedback**

It was easy to test and improve at every stage by running small modules, like the login first and then the portfolio.

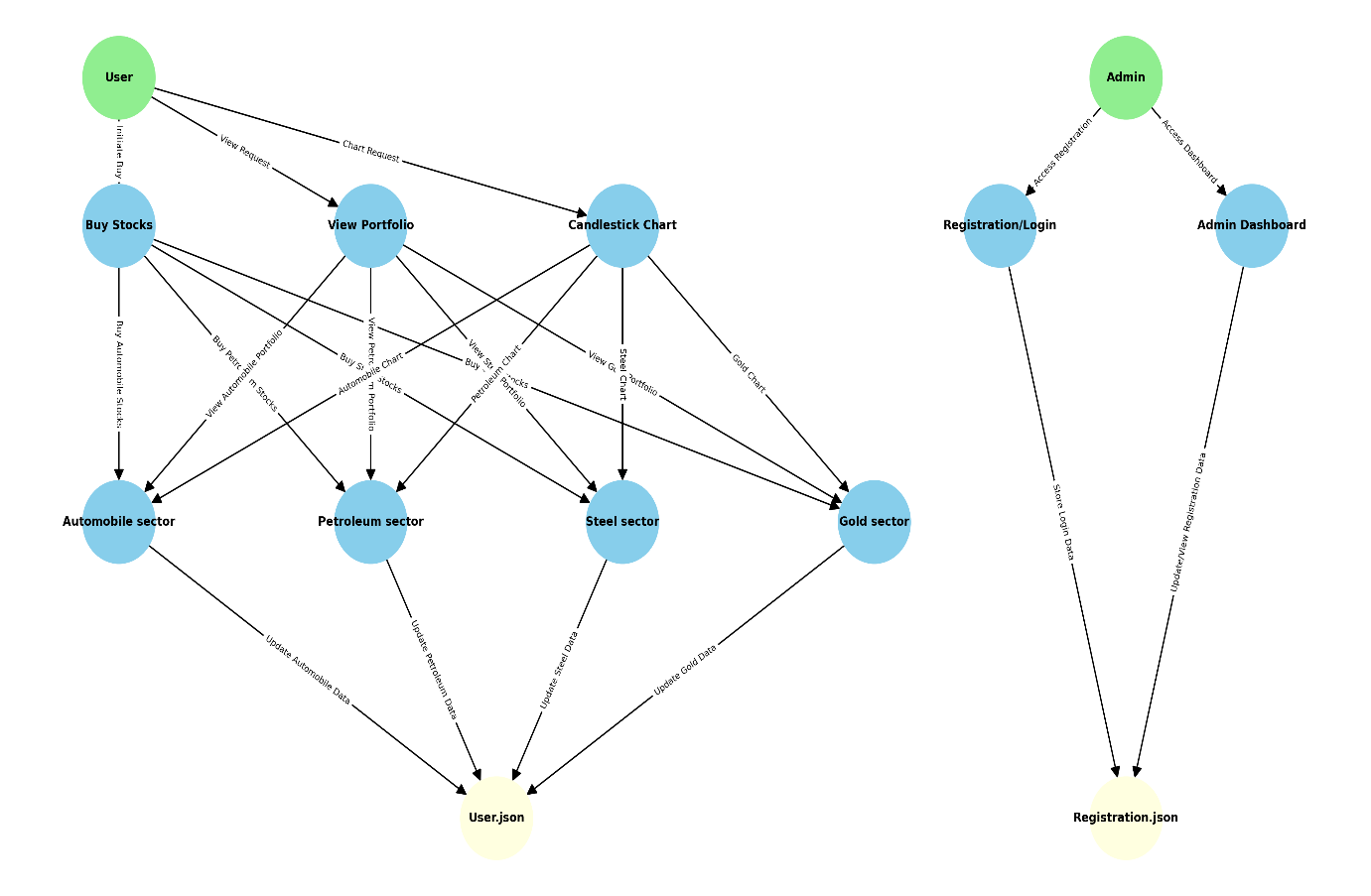
**Low Risk and Easy Maintenance :** Since the project is designed for academic purposes, incremental delivery made sure there was always at least a working prototype available.

**Data Flow Diagram :**

**Level 0 -DFD**



**Level 1 – DFD**



**Data Dictionary**

* + user\_record = Name + Email + Password + Balance + Bonus + Shares (Automobile) + Shares (Petroleum) + Shares (Steel) + Shares (Gold) + Last\_Buy\_Price + Created
  + registration\_record = Name + Email + Bonus
  + company\_record = Company\_Name + Sector + Price + URL
  + portfolio\_record = Company\_Name + Shares\_Owned + Unit\_Price + Total\_Value
  + admin\_record = Admin\_Password

**Use Case Approach**

**Use Case 1: General User Login**

**Primary Actor:**  
General User

**Precondition:**  
User has a registered email and valid password in the system

**Main Success Scenario:**

1. User enters email and password
2. System verifies credentials
3. User is logged in successfully and shown the Trading Menu

**Exception Scenarios:**

* **2a. Invalid email or password**
  + System displays an error message and prompts again
* **2b. User not found**
  + System suggests signup first

**Use Case 2: General User Trading (Buy/Sell)**

**Primary Actor:**  
General User

**Precondition:**  
User is logged in and has sufficient balance or shares

**Main Success Scenario:**

1. User selects sector and company
2. User enters quantity and selects Buy/Sell
3. System verifies balance (for Buy) or shares owned (for Sell)
4. Transaction is processed successfully
5. Balance and portfolio are updated, confirmation shown

**Exception Scenarios:**

* **3a. Insufficient balance for Buy**
  + System displays error and cancels transaction
* **3b. Not enough shares to Sell**
  + System displays error and cancels transaction

**Use Case 3: Admin Login**

**Primary Actor:**  
Admin

**Precondition:**  
Admin knows the system’s master password

**Main Success Scenario:**

1. Admin enters password in Admin Login panel
2. System verifies password
3. Admin is logged in successfully and shown the Admin Dashboard

**Exception Scenarios:**

* **2a. Invalid password**
  + System displays an error message and prompts again

**Use Case 4: View Admin Dashboard**

**Primary Actor:** Admin

**Precondition:**  
Admin is logged in successfully

**Main Success Scenario:**

1. Admin requests to view all registered users
2. System fetches user details from the database
3. System displays a dashboard with:
   * Name, Email, Balance
   * Sector-wise stocks
   * Total portfolio value

**Exception Scenarios:**

* **2a. No user records found**
  + System displays “No users found” message

**Sequence Diagram**

* **Sequence Diagram (General User Login)**

Manager/General User Login Interface Auth Controller User Database

| | | |

|---------Enter creds----------->| | |

| |---Send creds-------------> | |

| | |---Validate-------->|

| | |<--Auth result------|

| |<--Success/Fail-----------> | |

|<--Show menu/error--------->| | |

* **Sequence Diagram (General User Trading – Buy/Sell)**

User Trade Window System (Balance/Portfolio DB)

| | |

|--Select Company--> | |

|--Enter Quantity--> | |

|--Buy/Sell Action-> |---Check Balance/Portfolio---->|

| |<---------Update-------------- |

|<--Success/Fail Msg- | |

* **Sequence Diagram (Admin Login + Dashboard)**

Admin Admin UI Auth Controller User Database

| | | |

|--Enter pwd-->| | |

| |---Verify pwd----->| |

| |<--Success/Fail----| |

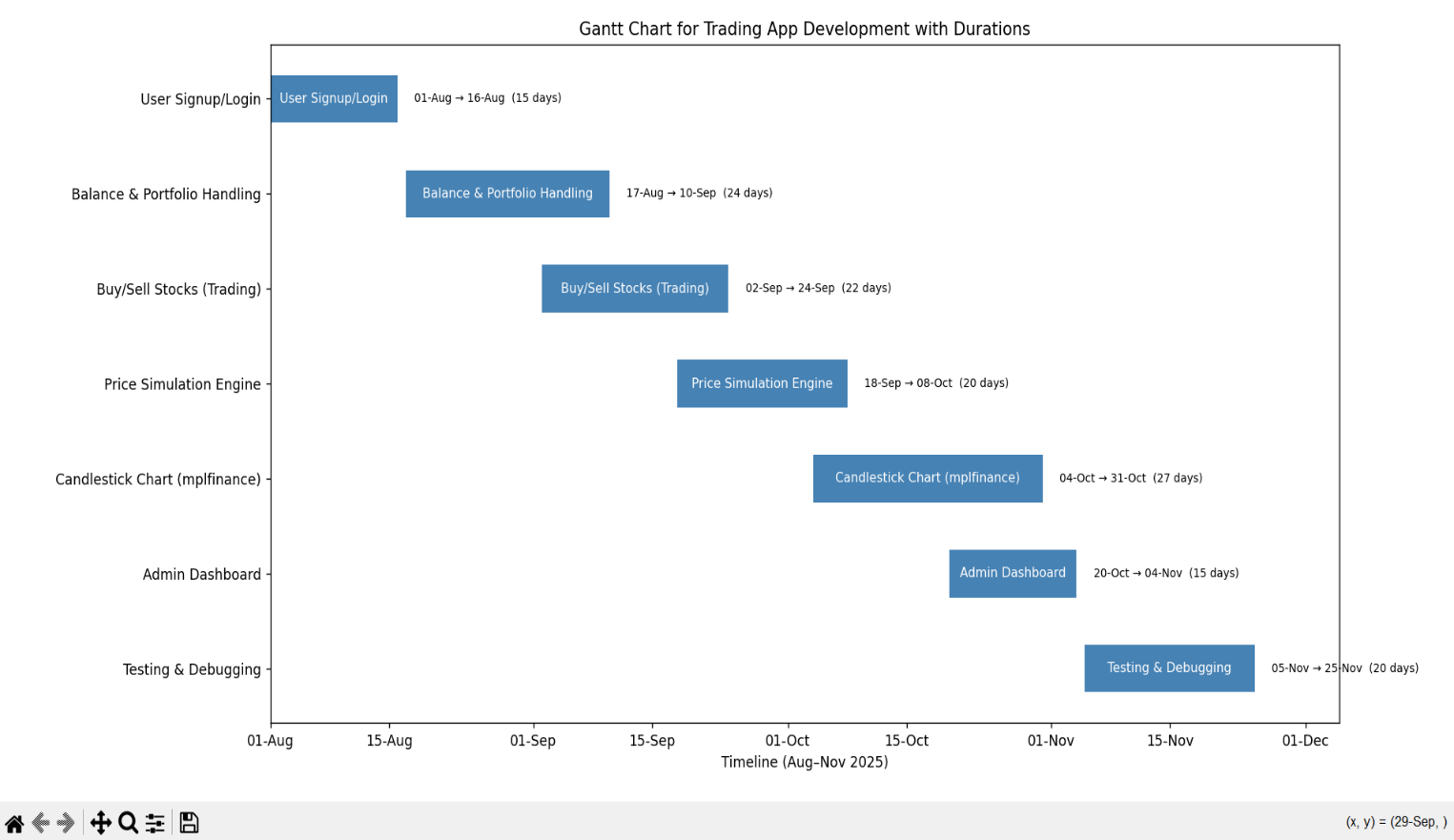
|<--Error/OK---| | |

| |---Fetch all users--------------------->|

| |<---------------User list--------------- |

|<--Show Admin Dashboard with all user data-------|

**Timeline Chart**

****

**Function Point Estimation**

**1. Objective**

The objective of this report is to estimate the size of the project INVESTKARO in terms of Function Points (FP). Function Point Analysis (FPA) is a standardized method to measure the functionality delivered by the system to the user, independent of the programming language, technology, or development environment used.

**2. Project Description**

INVESTKARO is a desktop-based stock market simulator built in Python using Tkinter. The system allows users to register, log in, simulate buying and selling of stocks, track portfolio performance, visualize price trends through candlestick charts, and manage funds virtually. It also includes an admin panel for managing user data and viewing overall system activity.

**3. Identification of Functional Components**

|  |  |  |
| --- | --- | --- |
| Function Type | Description | Examples |
| External Inputs (EI) | User inputs that modify internal data | Signup, Login, Buy, Sell, Manual Price Input, Admin Login |
| External Outputs (EO) | Processed information displayed to the user | Portfolio Display, Admin Dashboard, Candlestick Chart |
| External Inquiries (EQ) | Input-output interaction without internal file update | View Company Link, Load Data, Check Prices |
| Internal Logical Files (ILF) | Data maintained by the system | users.json, registrations.json |
| External Interface Files (EIF) | Data referenced but not maintained internally | External screener.in links, Images |
|  |  |  |

**4. Function Point Counting**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component Type | Count | Complexity | FP Weight | Total FP |
| External Inputs (EI) | 6 | Medium | 4 | 24 |
| External Outputs (EO) | 5 | High | 7 | 35 |
| External Inquiries (EQ) | 4 | Low | 3 | 12 |
| Internal Logical Files (ILF) | 2 | Low | 7 | 14 |
| External Interface Files (EIF) | 1 | Low | 5 | 5 |

Unadjusted Function Points (UFP) = 90

**5. Value Adjustment Factor (VAF)**

There are 14 General System Characteristics (GSCs) rated between 0 (no influence) to 5 (strong influence). For INVESTKARO, an average rating of 3 per characteristic is assumed.  
  
Σ(GSC) = 14 × 3 = 42  
VAF = 0.65 + (0.01 × 42) = 1.07

1. **Adjusted Function Point (AFP) Calculation**

AFP = UFP × VAF = 90 × 1.07 = 96.3 ≈ 96 Function Points

**7. Effort Estimation**

Assuming productivity = 10 Function Points per person-month:  
Effort = 96 / 10 = 9.6 person-months  
  
If 1 person works 160 hours per month:  
Total Effort = 9.6 × 160 = 1536 person-hours

**8. Conclusion**

The INVESTKARO project has an estimated size of approximately 96 Function Points. This estimation provides insight into the development effort, time, and resource requirements for the project. Function Point Analysis helps in early-stage project planning and ensures better cost and schedule estimation.