MASTER OF COMPUTER APPLICATIONS

(Programme Code: MCA_NEW)

MCSP – 232

A PROJECT REPORT ON

“Portfolio manager and workflow automation by LIKITH B R”

Under Guidance of

SUBMITTED BY

Name: LIKITH B R

Enrolment Number: 2301465692

Study Canter: 1314

Year of Submission:

Project Title: Portfolio manager and workflow automation

Introduction:

In In today's dynamic financial markets, efficient portfolio management is essential for investors to make informed decisions and achieve their financial objectives. To address this need, I have developed a comprehensive portfolio management tool and report generation automation in Excel. This tool is designed to streamline the process of tracking investments, analyzing performance, and generating insightful reports.

Key features include automated data entry and updates, real-time performance tracking, customizable reporting, and integration with financial data sources**.** By automating data entry and updates, the tool eliminates the risk of manual errors and saves time. Real-time performance tracking provides investors with up-to-date information on portfolio returns, risk metrics, and asset allocation. Customizable reporting allows users to generate tailored reports that meet their specific needs, such as performance summaries, risk analysis, and asset allocation breakdowns.

Objectives: -

The primary objective of this is to streamline portfolio management processes, enhance decision-making, improve reporting efficiency, and increase accuracy by automating data entry, tracking, and analysis.

Streamline portfolio management processes: Automate data entry, tracking, and analysis to reduce manual effort and errors. Automate data entry, tracking, and analysis to reduce manual effort and errors. This means the tool will automatically collect and update investment data, track performance metrics, and analyse portfolio trends.

Enhance decision-making: Provide real-time performance data and insights to support informed investment decisions.

Improve reporting efficiency: Generate customizable reports quickly and easily, saving time and effort.

Enhance user experience: Design the tool with a user-friendly interface and intuitive navigation.

Cost Reduction and Resource Optimization: By automating a wide range of tasks, the system will contribute to significant cost savings and better resource utilization, ultimately driving business growth.

Project Category:

FinTech is a broad term that encompasses a wide range of technologies that are used to deliver financial services.
It includes everything from mobile payments and cryptocurrency to robo-advisors and crowdfunding platforms development of an intelligent automation.
Personal Finance Management is a specific subset of FinTech that focuses on tools and services designed to help individuals manage their personal finances. This can include budgeting apps, debt management tools, and investment tracking software.

The key aspects of this project category include:

Leveraging technology: Utilizing various technologies to automate and improve financial processes.

Focus on individuals: Targeting personal finance needs rather than institutional or corporate finance.

Data-driven: Relying on data analysis and insights to inform financial decisions.

User-friendly: Designing tools that are easy to use and accessible to a wide range of users.

Integration with other financial tools: Connecting with other financial software for a comprehensive solution.

Regulatory compliance: Adhering to relevant financial regulations and standards.

Continuous innovation: Staying updated with emerging technologies and adapting to changing market trends.

Hardware/Software Requirement Specification:

Hardware Requirements: -

- Processor:11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz   3.11 GHz
- RAM: 16.0 GB (15.8 GB usable)
- Storage: 512 GB (or 1 TB) SSD.
- Display: 15.6" Full HD Display.
- Network Card: Intel® Pro 10/100/1000 Mbps LAN Card.
- Optical Drive: No CD/DVD drive (modern laptops typically don't include these).

Client Configuration (Minimum):-

- Processor: 1th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz   3.11 GHz
- RAM: 16.0 GB (15.8 GB usable)
- Network Card: Intel® Pro 10/100/1000 Mbps LAN Card.
- Connectivity: Wi-Fi 6 and Ethernet LAN port.

Software Requirements:

- Programming language: Python (Django), HTML, CSS, Jinja, JavaScript.
- Cloud Services: Azure App services, Azure container registry, Azure App registration.
- Tools : VSCode, Docker, Git,Github repository.

Django Framework:

- Django is a high-level Python web framework that follows the Model-View-Template (MVT) architectural pattern. It's designed to help developers build web applications quickly and efficiently.
- Django is a powerful and versatile web framework that can help developers build high-quality web applications efficiently.

Django Libraries:

- django-allauth: Integrates with various social login providers (e.g., Google, Facebook) for user authentication, simplifying user signup.

- openpyxl: Enables interacting with Excel files (XLSX format), possibly for data import/export or report generation in the desired format.

Data:

- pandas: A powerful Python library for data manipulation and analysis, essential for processing large datasets and preparing data for AI models.
- NumPy: A fundamental package for scientific computing with Python, useful for working with arrays and matrices.

Automation and GUI Interaction:

- Django-extensions: Offers a collection of utilities to streamline Django development.

- pyodbc: Facilitates connecting to Microsoft SQL Server databases, providing database access flexibility.

- **psycopg2-binary:** Enables connection and interaction with PostgreSQL databases.

Database Management:

- MySQL or PostgreSQL: Relational database management systems to store and manage the data used and generated by the RPA system.
- SQLite: A lightweight, disk-based database that can be used for smaller-scale projects or when portability is a concern.

Deployment and CI/CD:

- Docker: A platform for developing, shipping, and running applications in containers, ensuring consistency across different environments.
- Jenkins or GitLab CI/CD: Continuous Integration/Continuous Deployment tools

Problem Definition, Requirements Specifications:

Problem Definition

A. Currently an organization maintains the shareholder register manually in excel documents. This process involves manually updating shareholder details, which is time-consuming and prone to human error. Retrieving and analysing information from these documents can also be challenging and inefficient. Every entity is legally obligated to maintain a shareholder register that comprehensively details its shareholders' information. This register must encompass all shareholders who hold stakes in the company and has to be continually updated from the date of investment onwards.

B. Currently, this process is executed manually, using Microsoft excel documents. However, this manual approach is prone to errors and consumes significant time. One company hosts approximately 1000 entities, some of which have more than 100 shareholders each. Managing and updating the register manually in excel for each entity and shareholder is highly inefficient. Automating this process will provide organization with a centralized system to manage the shareholder register. All data will be securely stored on a database, accessible through a user-friendly interface. Users will be able to effortlessly update transaction details without the need for extensive manual input. The automation tool will handle calculations, descriptions, shareholder names, formatting, and checkpoints, ensuring accuracy and consistency throughout the register.

Requirements Specifications:

Functional Requirements:

- **Centralized Database:** Establish a secure, centralized database to store shareholder information.
- **Data Entry:** Allow users to input shareholder details, including names, addresses, contact information, and ownership stakes.
- **Transaction Management:** Track and record all shareholder transactions, such as share purchases, sales, transfers, and dividends.
- **User-Friendly Interface:** Develop an intuitive interface for easy data input, retrieval, and analysis.
- **Calculation Automation:** Automatically calculate shareholder ownership percentages, dividends, and other relevant metrics.
- **Reporting:** Generate various reports, including shareholder lists, ownership breakdowns, dividend statements, and compliance reports.

Non-Functional Requirements:

- **Performance:** The system should be able to handle a large number of shareholders and transactions efficiently.
- **Scalability:** The system should be able to accommodate future growth and expansion.
- **Usability:** The user interface should be intuitive and easy to use for both technical and non-technical users.
- **Cost-Effectiveness:** The system should be cost-effective to implement and operate.

Technical Requirements:

- **Programming Language**: Python will be the primary programming language for implementing the automation scripts, and system integration.

- **Framework**: Django is designed to handle large-scale applications, making it suitable for managing a potentially large number of shareholders.
- **API**: Develop APIs to allow integration with other systems and to fetch user roles from azure.
- **Libraries**: Utilize libraries such as django-allauth, openpyxl, for developing automation model.
- **Database**: Implement a relational database (e.g., MySQL or PostgreSQL) to store process data, logs, and other relevant information.
- **Cloud Integration**: If required, integrate with cloud services Azure for hosting, data storage, and leveraging cloud-based web app services.

Risk Management:

- **Data Backup:** Implement regular backups of the database to prevent data loss.
- **Access Controls:** Implement strong access controls to restrict access to sensitive data.
- **Security Audits:** Conduct regular security audits to identify and address vulnerabilities.
- **Testing and Validation:** Thoroughly test the system before deployment to identify and fix potential issues.
- **Contingency Planning:** Develop a contingency plan to address potential disruptions, such as system failures or data breaches.
- **User Training:** Provide comprehensive training to users to ensure they understand how to use the system effectively.

Scope of the Solution: The shareholder register automation solution will focus on centralized data management, automated data entry and updates, accurate calculations, comprehensive reporting, ensuring security and compliance, and providing a user-friendly interface. It will not include providing investment advice, managing overall financial operations, or handling legal matters related to shareholder disputes.

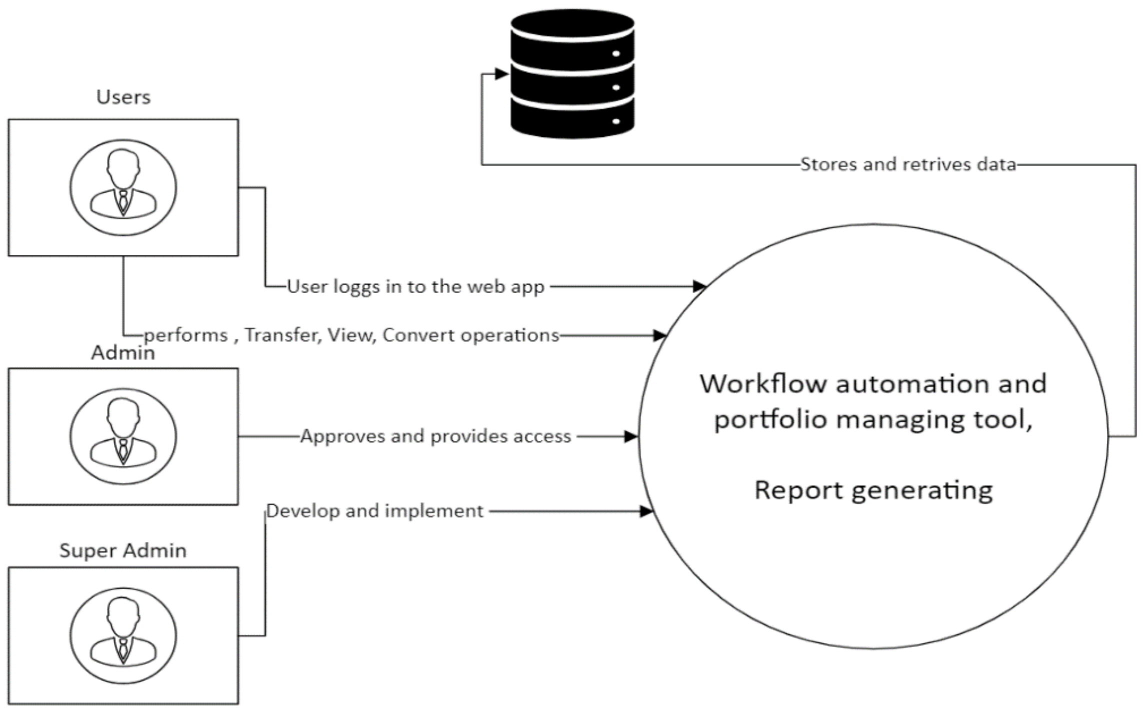
Limitations

- **Dependency on Data Quality:** The accuracy and reliability of the automated system depend on the quality of the data entered. Inaccurate or incomplete data can lead to incorrect calculations and reports**.**
- **Human Oversight:** While the system can automate many tasks, human oversight is still necessary to ensure data accuracy, resolve issues, and make strategic decisions.
- **Complexity of Shareholder Structures:** The system may struggle to handle complex shareholder structures, such as multiple classes of shares or crossholdings, without additional customization.

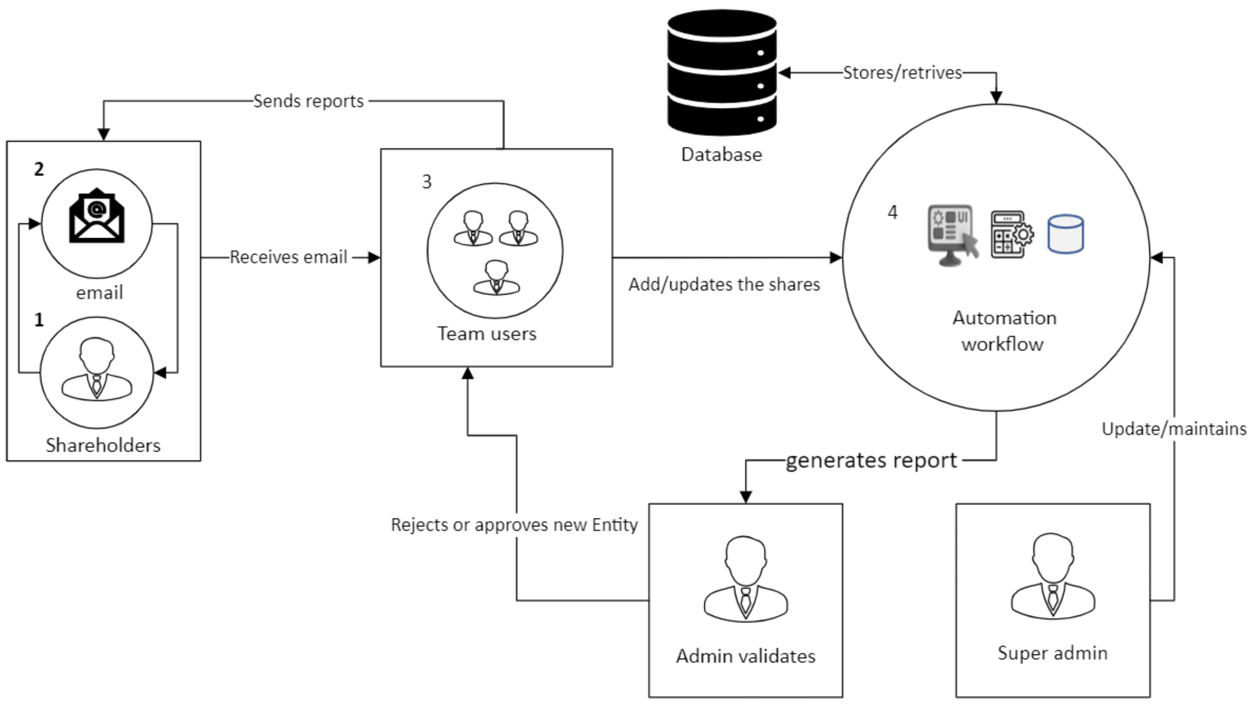
Data Flow Diagrams (DFDs):

DFDs illustrate how data moves through the system, showing the flow of information between processes, data stores, and external entities.

Level 0: DFD Diagram

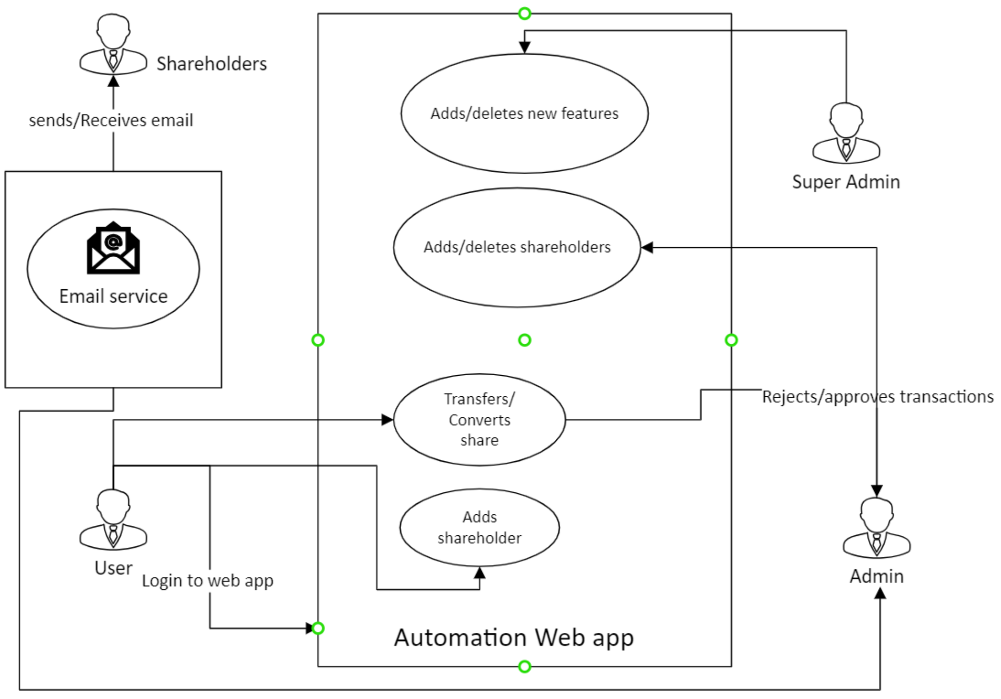

Users
User loggs in to the web app
performs , Transfer, View, Convert operations
Admin
Approves and provides access
Develop and implement
Super Admin
Stores and retrives data
Workflow automation and portfolio managing tool,
Report generating

Level 1:



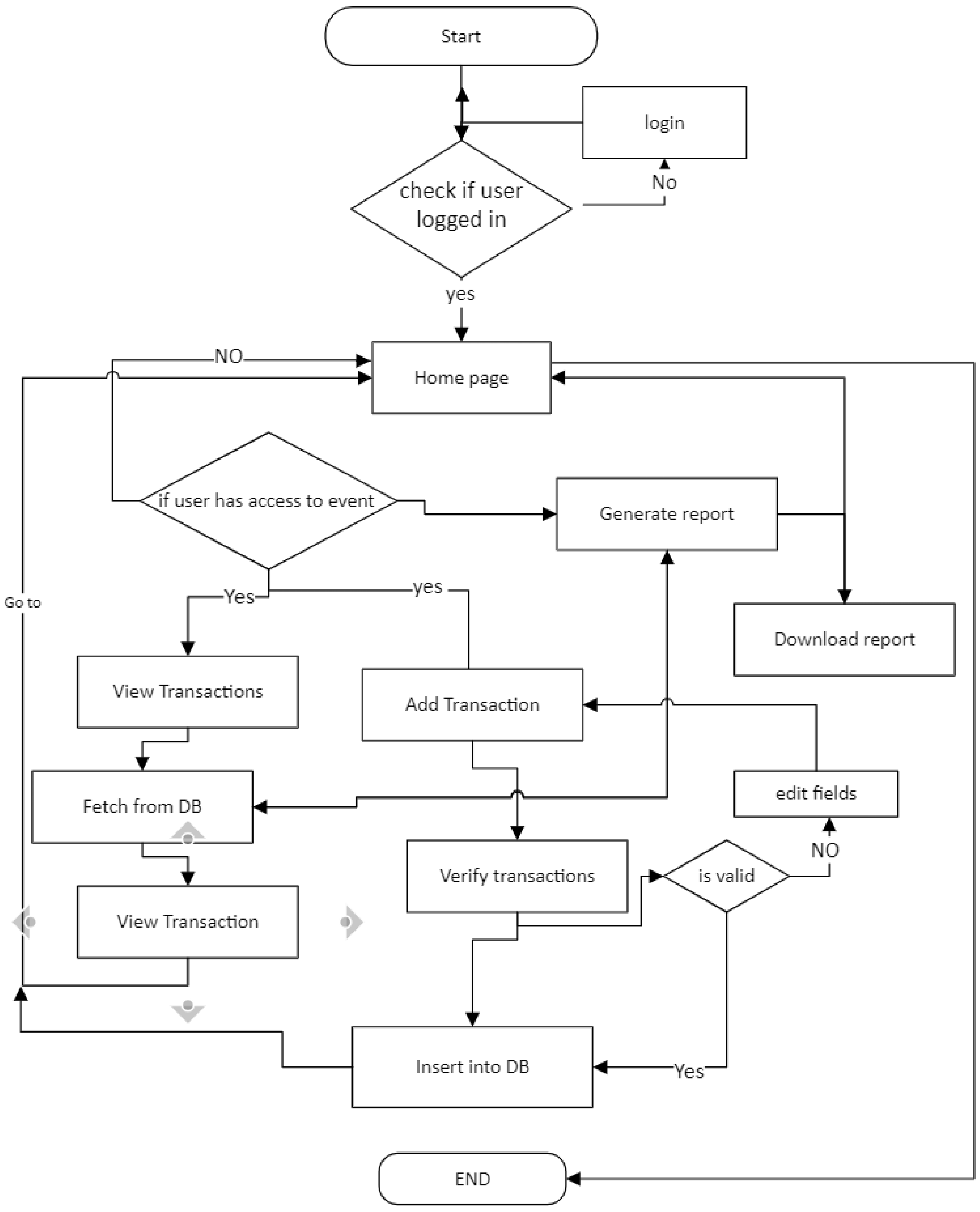
Use Case Diagram:

A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.



Flow Chart :

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.



Class Diagram

If the system is being implemented in an object-oriented manner, a class diagram may be used to represent the system's structure, focusing on classes, their attributes, methods, and the relationships between them.

Classes:

1. **User**:
    - Attributes: UserID, Name, Email, Role
    - Methods: configureTask(), viewReport(),DownlaodReport()
2. **Task**:
    - Attributes: TaskID, Description, TaskType, PerformedBy
    - Methods: extractData(), updateData()
3. **Function**:
    - Attributes: Name, functionType
    - Methods: initiatefunction(), generateReport()
4. **Shareholder**:
    - Attributes: ShareholderID, ShareholderName, Holdings,Shares,
    - Methods: AddShareholder(), Update()
5. **DataSource**:
    - Attributes: SourceID, SourceName, ConnectionDetails
    - Methods: fetchData(), validateData()
6. **Report**:
    - Attributes: ReportID, ReportName, GeneratedDate
    - Methods: generate (), download

7. **Company:**
    - Attributes: CompanyID, CompanyName, Shareholders,
    - Methods: AddShareholder(), Update()

Database Schema Overview

- **Database Name**: Django_db

The database consists of several tables designed to store information related to users, tasks, processes, AI models, data sources, and reports.

Tables and Their Structures

Users

This table stores information about the company.

- **Table Name**: db_core_Company
- **Columns**:
    - CompanyID (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each company.
    - Name (VARCHAR (100)): Name of the company.
    - Address (VARCHAR (100), UNIQUE): address of the company.
    - User_logs (VARCHAR (50)): Company added by.
    - Currency (VARCHAR (255)): Currency of the company

Shareholder

This table stores information shareholders.

- **Table Name**: db_core_shareholder
- **Columns**:
    - Folio_no (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each Shareholder.
    - Name (VARCHAR(100)): Name of theShareholder.
    - Address (VARCHAR(50)):Shareholder address.
    - Legal_form (VARCHAR(50)):Legal form number.
    - CompanyId(INT): Foreign Key from table db_core_company

Shares

This table stores information about the Shares.

- **Table Name**: db_core_Shares
- **Columns**:
    - Share_ID (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each shares.
    - No_of_shares (INT) no of shares holding by the shareholder,
    - Class_of_shares(VARCHAR(100)):Class of the shares belongs to.
    - folio_no_Id(INT): Foreign Key from db_core_shareholder .

Transaction

This table stores information about transactions.

- **Table Name**: db_core_transaction
- **Columns**:
    - transaction_ID (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each transaction.
    - event (VARCHAR) User performing event,
    - No_of_shares (INT) no of shares in the current transaction,
    - Nominal_value(FLOAT):As per the business decision
    - Consideration(FLOAT):Money considered in the current transaction,
    - Event_Date(DATETIME)date and time of the transaction,
    - Folio_no _id(INT): Shareholder Id who is doing transactions,
    - Transfer_details_id(INT):receiver details who is doing transactions,
    - User_log(VARCHAR): User name who is doing transactions
    - Total (INT): Total amount transfer in the transaction,

System Architecture

The architecture of the Portfolio manager is typically a multi-tier, modular structure designed for scalability, flexibility, and ease of integration with existing business systems. The architecture is divided into several layers:

- **Presentation Layer:** The front-end user interface that business users interact with, typically accessed via a web-based portal or a desktop application.
- **Application Layer**: The core logic layer where the business processes, and automation tasks are managed. This layer contains the orchestration Container, Kubernetes services, and process automation modules.
- **Data Layer**: The backend layer where all data is stored, including user data, task logs, process configurations, and Transactions data. It interacts with databases and no external data sources.
- **Integration Layer**: Manages communication between the Web app and databases, and Azure graph APIs. It ensures seamless data flow and process integration.
- **Security Layer**: Cross-cutting layer that ensures data security, access control, and compliance with relevant regulations.

Component Overview:

The key components of the system include:

1. **User Interface (UI)**
    - A web application that allows users to interact with the system.
    - Provides dashboards for performing tasks, adding shares, viewing and downloading reports.
2. **Orchestration Engine**
    - The central controller that manages and sequences automation tasks.
    - Coordinates interactions with graph API to fetch the user roles.
    - Includes models and algorithms used for generating formatted excel sheet.
3. **Process Automation Module**
    - Handles the execution of automated tasks based on predefined business logic.
    - Can manage both simple, rule-based tasks and complex multiple class shares operations.
4. **Data Management System**
    - Comprises databases that hold all relevant data, including user information, task logs, process configuration.
    - Ensures data integrity, consistency, and availability.
5. **Integration APIs**
    - A Graph APIs and Django_all_auth that allow the container system to interact with database.
    - Facilitates data exchange and process integration.
6. **Security Framework**
    - Implements authentication, authorization, and encryption mechanisms.
    - Ensures compliance with data protection regulations like GDPR, HIPAA, etc.

User Interface and Experience:

The User Interface (UI) is designed to be intuitive and user-friendly, enabling non-technical users to interact with the system efficiently. Key features include:

1. **Dashboard**:
    - Provides an overview of all events, tasks, and their status.
    - Displays all the previous transactions done by other users.
2. **Process Configuration**:
    - Allows users to add and modify the shareholder details.
    - Provides options for common business processes to simplify setup.

3. **Reports and Analytics**:
    - Users can generate and view reports on-demand, export data, and view the current holdings,

Key Outcomes:

1. Increased Operational Efficiency

- **Automation of Repetitive Tasks**:
    - **Data Entry:** Manually entering shareholder information, transaction details, and updates into Excel spreadsheets.
    - **Data Validation:** Checking for errors and inconsistencies in the entered data.
    - **Calculations:** Manually calculating ownership percentages, dividends, and other financial metrics.
- **Reduced Process Cycle Time**:
    - Automation eliminates the need for manual data entry and updates, saving time and resources.
    - **Data Updates:** Manually updating shareholder information and transaction details as needed.
    - **Standardized format:** Ensures consistent formatting and layout for all generated reports.
- **Enhanced Accuracy**:
    - **Increased accuracy:** Automated calculations and data validation minimize the risk of human errors.
    - **Efficiency:** Automatically generates reports, saving time and effort.

Conclusion:

The shareholder register automation project represents a significant improvement over manual process. By automating data entry, calculations, and reporting, the system enhances efficiency, accuracy, and compliance. Overall, the shareholder register automation project is a valuable investment that can significantly improve the management of shareholder information and support the organization's long-term success.

Links

https://youtube.com

https://www.geeksforgeeks.com

https://stackoverflow.com

https://learn.microsoft.com