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

Dr B.N SHANKAR GOWDA

SUBMITTED BY

Name: LIKITH B R

Enrolment Number: 2301465692

Study Canter: 1314

Year of Submission:2025

ABSTRACT

The "Portfolio Manager and Workflow Automation" project addresses the complexities of modern financial portfolio management by providing a robust and scalable platform built on Django, Microsoft SQL, Docker Compose, and Azure App Services. This system goes beyond basic data handling and reporting, offering capabilities to manage extensive data for numerous companies and shareholders, record transactions, and enforce role-based access control (RBAC) for global compliance.

The project's current features, including comprehensive data management, report generation, and secure authentication, tackle core industry needs. Future implementations aim to enhance user experience and expand the system's functionality. Key areas of development include interactive dashboards with real-time data visualization, advanced search and filtering, customizable workflows, and mobile accessibility. Integration with external financial platforms, support for additional asset classes, and global expansion are also planned. The project will also explore advanced technologies like AI-powered analytics, blockchain integration, and cloud optimization to provide cutting-edge solutions. Security enhancements, such as multi-factor and biometric authentication, are also a priority. This project seeks to deliver a comprehensive and evolving solution for managing financial portfolios.

PREFACE

In today's dynamic financial landscape, the efficient management of complex portfolios is paramount. This project, "Portfolio Manager and Workflow Automation," aims to create a robust, scalable, and secure solution that addresses the challenges of modern portfolio management. By leveraging a combination of established and cutting-edge technologies, this system seeks to provide a platform that not only meets current industry needs but is also adaptable to future demands.

The application is designed to handle the intricate requirements of managing numerous company portfolios, each with many shareholders, and to accurately record the myriad transactions that occur between them. A key focus is on empowering users with the tools they need to make informed decisions, while ensuring data integrity and security. The project also emphasizes the importance of global compliance by incorporating role-based access control. Through ongoing development and the integration of advanced technologies, this project strives to deliver a comprehensive and evolving solution for financial portfolio management.

The impetus for this project stems from the growing need for sophisticated tools that can navigate the complexities of modern finance. Traditional methods of portfolio management often involve disparate systems, manual processes, and a lack of transparency, leading to inefficiencies and increased risk. This project seeks to address these issues by providing a centralized, automated, and highly secure platform that streamlines workflows, improves accuracy, and facilitates better decision-making.

The development of this system is guided by several key principles. First and foremost is the principle of scalability. The system is designed to handle not only current data volumes but also the anticipated growth in data and transactions in the future. This is achieved through the use of a robust database (Microsoft SQL), efficient containerization (Docker Compose), and a cloud-based deployment environment (Azure App Services).

TABLE OF CONTENT

INDEX

SYNOPSYS

Project Title: Portfolio manager and workflow automation

1.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.

2.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.

3.Project Category:

FinTech (Financial) 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 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.

3.1 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.

4.Hardware/Software Requirement Specification:

4.1 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.

4.2 Client Configuration (Minimum): -

- Processor: 1st 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.

4.3 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.
- Database : Microsoft SQL

5. 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.

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 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

6.Problem Definition, Requirements Specifications:

6.1 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 must 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.

6.2 Requirements Specifications:

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 Limitations

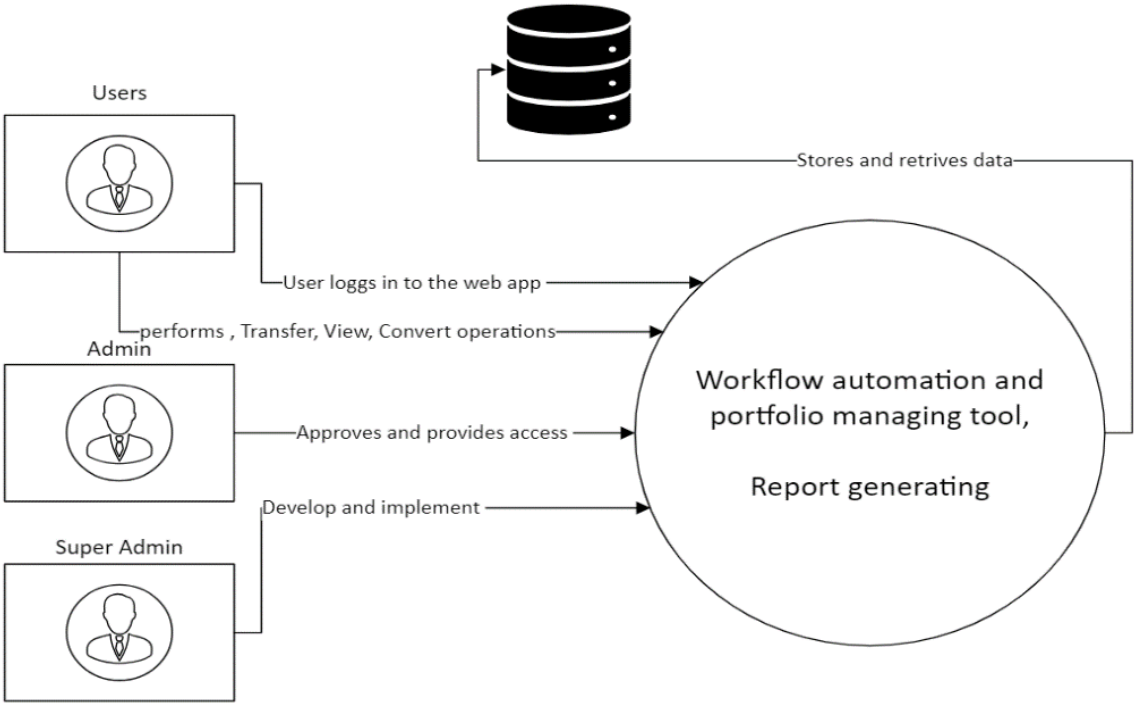
- **Dependency on Data Quality:** The accuracy and reliability of the automated system depend on the quality of the data entered. Inaccurate or incomplete data can lead to incorrect calculations and reports**.**
- **Human Oversight:** While the system can automate many tasks, human oversight is still 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.

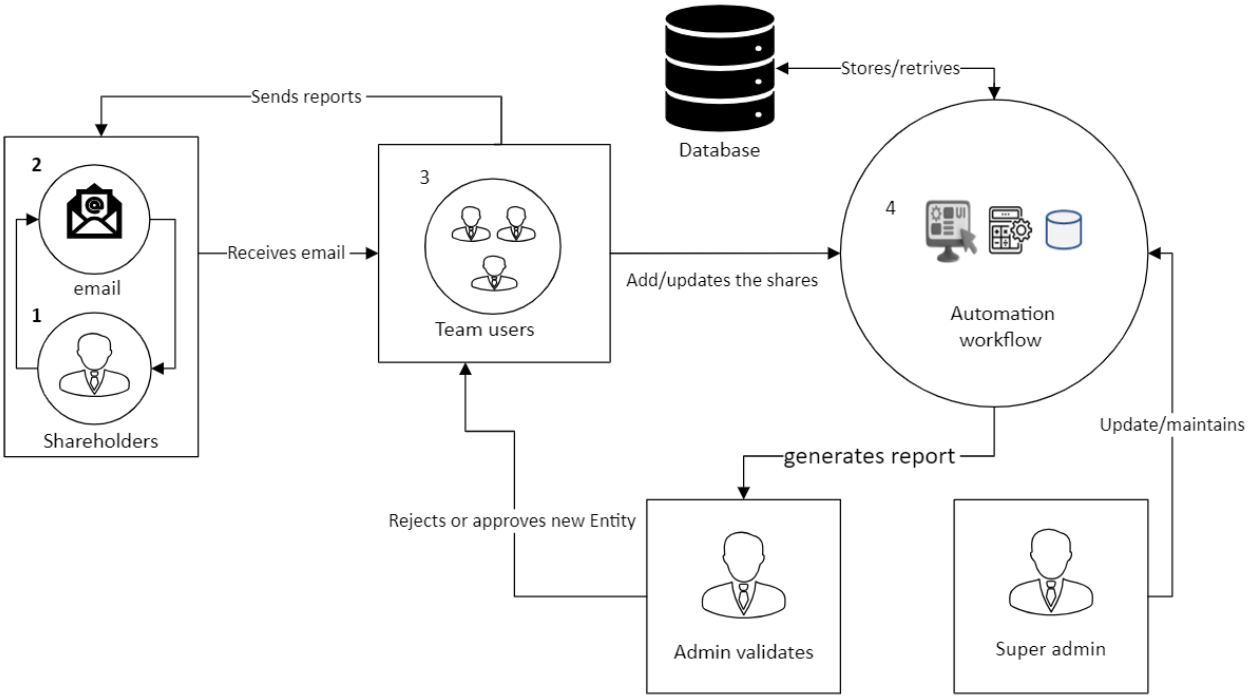
7.Data Flow Diagrams (DFDs):

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

7.1 Level 0: DFD Diagram

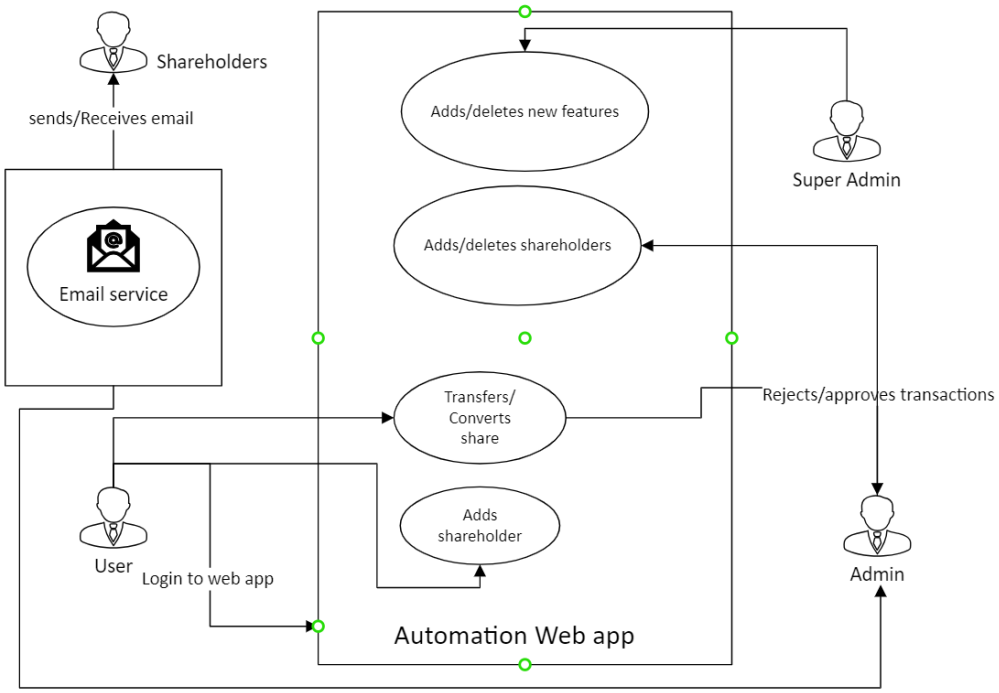


7.2 Level 1:



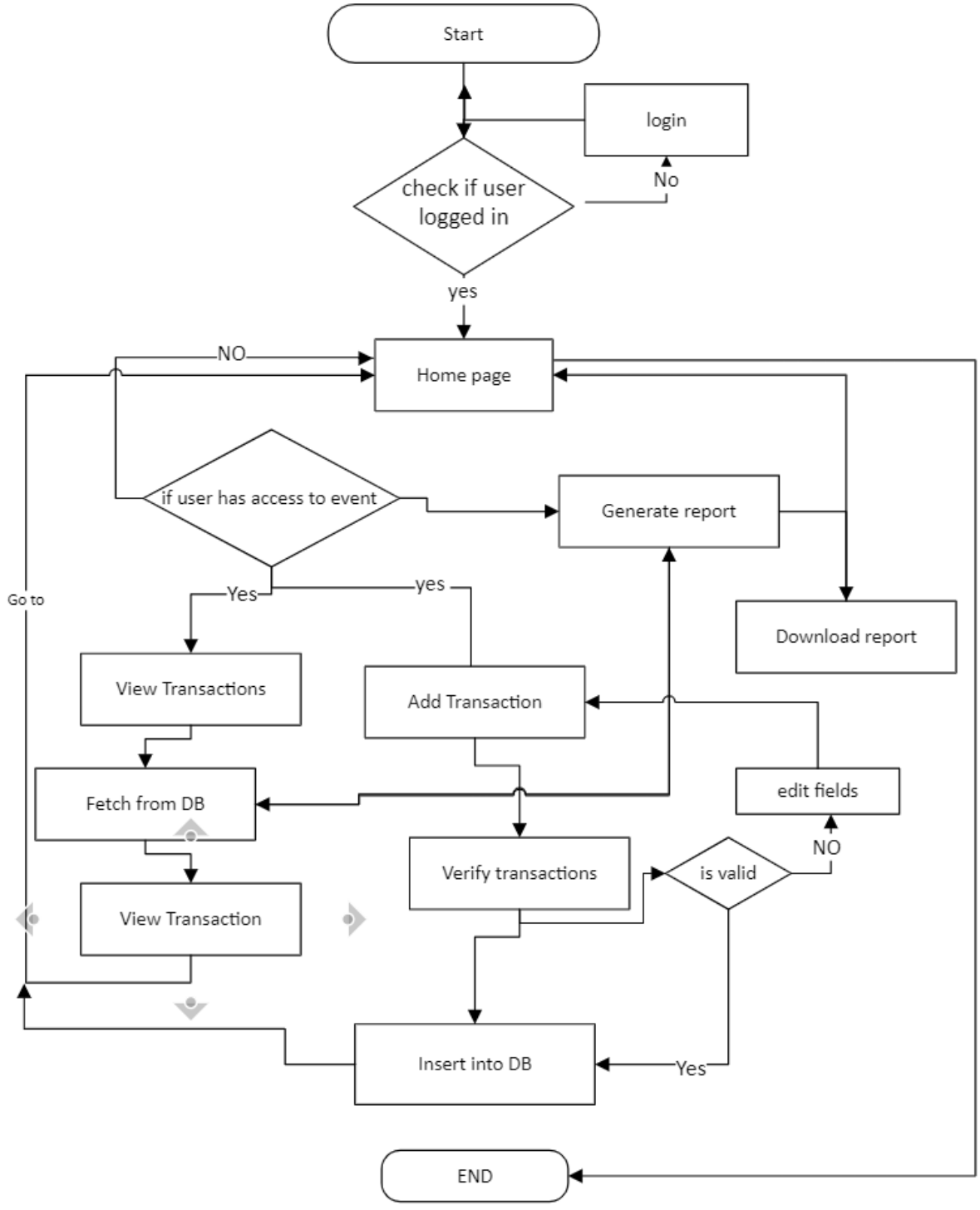
7.3 Use Case Diagram:

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.



7.4 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.



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. **Data Source**:
    - Attributes: SourceID, SourceName, ConnectionDetails
    - Methods: fetchData(), validateData()
6. **Company:**
    - Attributes: CompanyID, CompanyName, Shareholders,
    - Methods: AddShareholder(), Update()

8.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.

8.1 Tables and Their Structures

8.1.1 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

8.1.2 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

8.1.3 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 .

8.1.4 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,

9.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**.**

9.1 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.

9.2 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,

9.3 Key Outcomes:

9.3.1 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.

10. 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

PROJECT DOCUMENT

Chapter 1: Introduction

1.1. Introduction:

The "Portfolio Manager and Workflow Automation" project is designed to streamline and automate the management of extensive company portfolios. This system will efficiently handle data input, database storage, and report generation, addressing the complex needs of businesses with numerous stakeholders. In today's fast-paced business environment, organizations require robust tools to manage their investments, shareholder data, and financial transactions accurately and efficiently. Specifically, it will support the management of 100+ company portfolios, each with 100+ shareholders, and meticulously record all inter-company transactions. The application will feature robust data management capabilities and generate formatted Excel reports, providing clear and concise summaries of portfolio performance and transaction history. This will enable stakeholders to gain valuable insights and make informed decisions.

This solution aims to improve accuracy, reduce manual effort, and enhance decision-making by providing a centralized, automated platform for portfolio management. By automating key processes, the system will minimize the risk of human error, free up valuable time for strategic analysis, and ensure compliance with regulatory requirements.

The "Portfolio Manager and Workflow Automation" project addresses the increasing complexity of financial management in modern enterprises. Companies often struggle with disparate data sources, manual data entry, and the lack of a unified view of their investment portfolios. This can lead to inefficiencies, increased operational costs, and a higher risk of errors. The system will provide a centralized repository for all relevant data, including company information, shareholder details, transaction records, and portfolio valuations. This will ensure that all stakeholders have access to the most up-to-date and accurate information.

Furthermore, the system's workflow automation capabilities will streamline key business processes, such as transaction processing, report generation, and compliance monitoring. By automating these tasks, the system will reduce the need for manual intervention, freeing up staff to focus on higher-value activities. This will not only improve efficiency but also enhance employee satisfaction.

The reporting functionality is a critical component of the project. The system will generate formatted Excel reports that provide comprehensive insights into portfolio performance, financial positions, and transaction histories. These reports can be customized to meet the specific needs of different stakeholders, including senior management, investors, and regulatory bodies. The ability to generate reports in Excel format ensures compatibility with existing systems and facilitates easy data sharing and analysis.

In addition to its core functionality, the "Portfolio Manager and Workflow Automation" project is designed to be scalable and adaptable to future business needs. The system will be built using a modular architecture, allowing for easy expansion and customization. It will also incorporate robust security features to protect sensitive financial data and ensure

compliance with industry regulations. The project will leverage modern technologies and best practices to deliver a high-performance, reliable, and user-friendly solution.

1.2. Objectives: -

The "Portfolio Manager and Workflow Automation" project aims to deliver a robust, scalable, and secure system for managing extensive company portfolios, streamlining workflows, and ensuring global compliance through the implementation of Role-Based Access Control (RBAC). This system will address the complex challenges faced by organizations in managing many company portfolios, shareholders, and transactions, while adhering to increasingly stringent regulatory requirements.

The core objectives of this project are:

- **Centralized Portfolio Data Management:** To create a unified, accurate, and easily accessible repository for all portfolio-related data. This includes detailed information on 100+ companies, their 100+ shareholders each, and all transactions occurring between these entities. This centralization will replace fragmented data sources, reduce redundancy, and improve data integrity, forming a solid foundation for reliable reporting and analysis.

- **Automation of Key Workflows:** To automate essential and repetitive tasks within portfolio management, significantly reducing manual effort and the potential for human error. This encompasses automating transaction processing, report generation (including the creation of formatted Excel reports), and data updates. By streamlining these processes, the system will free up valuable time for personnel to focus on strategic initiatives and higher-level analysis.

- **Enhanced Reporting and Analytics:** To provide comprehensive, customizable, and timely reporting capabilities. The system will generate reports in various formats, with a focus on Excel, to ensure ease of use and compatibility with existing tools. These reports will offer insights into portfolio performance, financial positions, and transaction histories, empowering stakeholders to make informed decisions.

- **Implementation of Role-Based Access Control (RBAC):** To ensure global compliance and data security by implementing a granular Role-Based Access Control (RBAC) system. This will allow the system to define and enforce access policies, restricting data access and functionality based on user roles and responsibilities. RBAC is crucial for adhering to international regulations (like GDPR, CCPA) by providing auditable and controlled access to sensitive financial data. It will also support internal controls, preventing unauthorized or inappropriate actions by management and employees.

- **Scalability and Performance:** To develop a system that can efficiently handle large volumes of data and transactions, and scale to accommodate future growth in the number of companies, shareholders, and transactions. The system architecture will

be designed for optimal performance, ensuring quick response times and efficient data processing.

- **Data Security and Compliance:** To protect sensitive financial data with the highest level of security and ensure compliance with relevant global financial regulations. This includes implementing measures such as data encryption, secure authentication, audit trails, and regular security assessments. The RBAC system will be a cornerstone of this objective, providing fine-grained control over data access.

- **User Experience and Adoption:** To create an intuitive and user-friendly interface that simplifies complex portfolio management tasks and promotes user adoption across different user groups. The system will be designed to be accessible, efficient, and require minimal training.

1.3. Requirements analysis: -

This system aims to streamline the management of company portfolios, shareholder information, and transactions, while ensuring global compliance. The project will be developed using the Django framework, Microsoft SQL database, and Docker Compose for containerization. Azure App Services will be used for authentication.

1.3.1. Functional Requirements

The system must provide the following functionalities:

- **User Management and Authentication:**
  - The system shall use Azure App Services for user authentication.
  - The system shall support user login and logout functionalities.
  - The system shall implement Role-Based Access Control (RBAC) to manage user permissions.
  - The system shall allow administrators to manage user accounts and roles.
- **Company Management:**
  - The system shall allow users to create, read, update, and delete company information.
  - The system shall store company details such as name, address, and contact information.
  - The system shall support the management of 100+ companies.
- **Shareholder Management:**
  - The system shall allow users to add, update, and remove shareholders for each company.
  - The system shall store shareholder details, including name, contact information, and ownership percentage.
  - The system shall support the management of 100+ shareholders per company.
- **Transaction Management:**
  - The system shall record all transactions between companies.
  - The system shall store transaction details, such as date, type (e.g., share purchase, transferred), involved companies, and amount.
  - The system shall maintain a history of all transactions.
- **Portfolio Management:**
  - The system shall calculate portfolio holdings for each company and shareholder.
  - The system shall track changes in portfolio value over time.
  - The system shall provide a consolidated view of all company portfolios.

- **Reporting:**
    - The system shall generate formatted Excel reports.
    - The system shall provide reports on company portfolios, shareholder holdings, and transaction history.
    - The system shall enable users to download generated reports.
- **Data Input and Storage:**
    - The system shall provide user-friendly forms for data input.
    - The system shall validate all user inputs to ensure data accuracy and integrity.
    - The system shall store all data in a Microsoft SQL database.

1.3.2. Non-Functional Requirements

The system must meet the following non-functional requirements:

- **Performance:**
    - The system shall respond to user requests within acceptable timeframes.
    - The system shall efficiently handle a average volume of data and transactions.
- **Scalability:**
    - The system shall be scalable to accommodate future growth in data volume and user traffic.
    - The system shall be able to handle an increasing number of companies, shareholders, and transactions.
- **Security:**
    - The system shall protect sensitive data from unauthorized access.
    - The system shall implement Role-Based Access Control (RBAC) to restrict user access based on their roles.
    - The system shall comply with relevant global data protection regulations, such as GDPR.
    - The system shall use secure authentication mechanisms provided by Azure App Services.
    - The system shall maintain audit logs of all data modifications and access.
- **Usability:**
    - The system shall have an intuitive and user-friendly interface.
    - The system shall provide clear and concise error messages.
    - The system shall be accessible to users with varying levels of technical expertise.
- **Reliability:**
    - The system shall be reliable and available with minimal downtime.

    - The system shall implement proper error handling and data backup mechanisms.
- **Maintainability:**
    - The system shall be designed for easy maintenance and updates.
    - The system shall be well-documented.
- **Technology:**
    - The system shall be developed using the Django framework.
    - The system shall use a Microsoft SQL database for data storage.
    - The system shall be containerized using Docker Compose.
    - The system shall be deployed on Azure App Services.

1.3.3. Role-Based Access Control (RBAC) Requirements

The system shall implement the following RBAC roles and permissions:

- **Super Administrator:**
    - Full access to all system functionalities.
    - Ability to manage users, companies, shareholders, transactions, and reports.
    - Ability to configure system settings (developer access).
- **Employee:**
    - Ability to view company and shareholder information.
    - Ability to input transaction data.
    - Ability to generate standard reports.
- **Auditor (Read only):**
    - Read-only access to all data, including company, shareholder, transaction, and user information.
    - Ability to generate audit logs and reports.

1.3.4. Database Requirements

The Microsoft SQL database shall store the following data:

- **Users:** User authentication and authorization data (managed by Azure App Services, mapping/linking in the Django application).
- **Companies:** Company details (name, address, industry, etc.).
- **Shareholders:** Shareholder details (name, contact information, ownership percentage) linked to companies.
- **Transactions:** Transaction records (date, type, involved companies, amount).
- **Permissions:** Permissions associated with each role.

1.3.5. Deployment Requirements

- The system shall be deployed on Azure App Services.
- The system shall use Docker Compose for containerization of the application and Microsoft SQL database.
- The deployment process shall be automated.

1. MS SQL Database

Option A: Azure SQL Database (PaaS)

- Create a logical SQL server and a SQL database (Standard/Premium based on workload).
- Configure firewall rules to allow access from Azure App Service and container resources.
- Use VNet integration if private connectivity is required.
- Enable auditing, threat detection, and backup policies.

Option B: SQL Server on Azure VM (IaaS)

- Deploy a Windows Server VM and install SQL Server.
- Configure Network Security Group (NSG) rules to allow SQL traffic (port 1433).
- Set up VM backup, patching, and monitoring.
- Secure access with just-in-time VM access and/or VNet peering.

---

2. Containerized Application Deployment

Option A: Azure App Service with Docker Compose

- Create an Azure Web App for Containers.
- Use a docker-compose.yml file to define and deploy all three containers.
- Store container images in Azure Container Registry (ACR) or Docker Hub.
- Enable VNet integration for private backend communication.
- Configure App Settings for environment variables and secrets.

Option B: Azure Kubernetes Service (AKS)

- Set up a Kubernetes cluster with at least 1 node pool.
- Use Helm charts or YAML files to define and deploy the three containers.
- Integrate with Azure AD for access control and RBAC.
- Use Azure Key Vault and Kubernetes secrets for secure configuration.

Option C: Azure Container Apps

- Deploy each container as a separate Container App.

- Configure internal communication via Dapr or service-to-service discovery.
- Scale each container independently based on demand.
- Secure access using VNet, Managed Identity, and secrets integration.

---

3. Azure App Service Authentication

- Enable App Service Authentication (Easy Auth).
- Choose Azure Active Directory as the identity provider.
- Register the application in Azure AD (App registrations).
- Set Redirect URIs and API permissions.
- Use Managed Identity to access Azure resources securely (e.g., Azure SQL).
- Configure access restrictions, authentication flow, and role-based access if required.

---

4. Networking and Security

- Use Azure Virtual Network (VNet) to interconnect App Services, containers, and databases.
- Enable Private Endpoint or Service Endpoint for Azure SQL Database.
- Configure Managed Identity for secure access to Azure resources.
- Store secrets and connection strings in Azure Key Vault.
- Enable diagnostic logging, application insights, and monitoring.
- Apply appropriate role-based access control (RBAC) for all Azure resources.

1.3.6. Analysis

- The Django framework is well-suited for this project due to its robust features, mature ecosystem, and support for rapid development and good community support.
- Microsoft SQL is a reliable and scalable database that can handle the system's data storage requirements.
- Docker Compose simplifies the deployment and management of the application and database containers.
- Azure App Services provides a scalable and reliable platform for hosting the application.
- RBAC is essential for ensuring data security and compliance with global regulations.
- The system's modular design will allow for future expansion and customization.

1.4. Planning and Scheduling: -

4.1. Gantt chart

Task	Start date	End Date	Duration in (Days)
Define Specification and requirement gathering	28-02-2025	02-03-2025	3
Device setup Project Started	05-03-2025	05-03-2025	1
Database schema design	06-03-2025	07-03-2025	2
UI development	11-03-2025	03-03-2025	3
Web app Development	15-03-2025	21-03-2025	7
Modules integration	22-03-2025	25-03-2025	4
Azure integration	26-03-2025	27-03-2025	2
Testing	28-03-2025	02-04-2025	6
User Documentation	10-04-2025	13-04-2025	3

4.1.1 Project Life Cycle Gantt chart

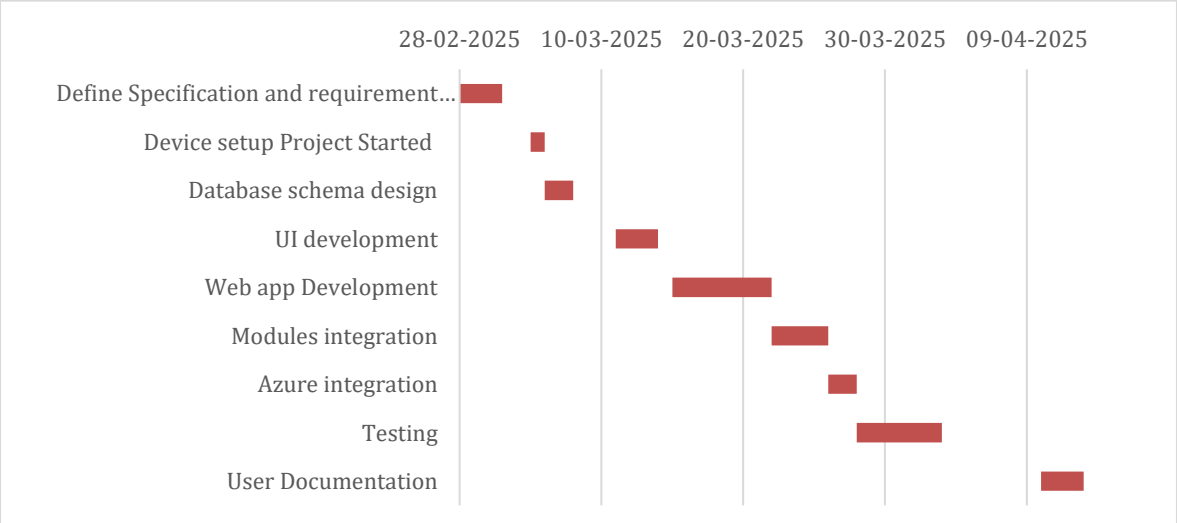


Fig-1: Gantt Chart

4.1.1 Project PERT chart

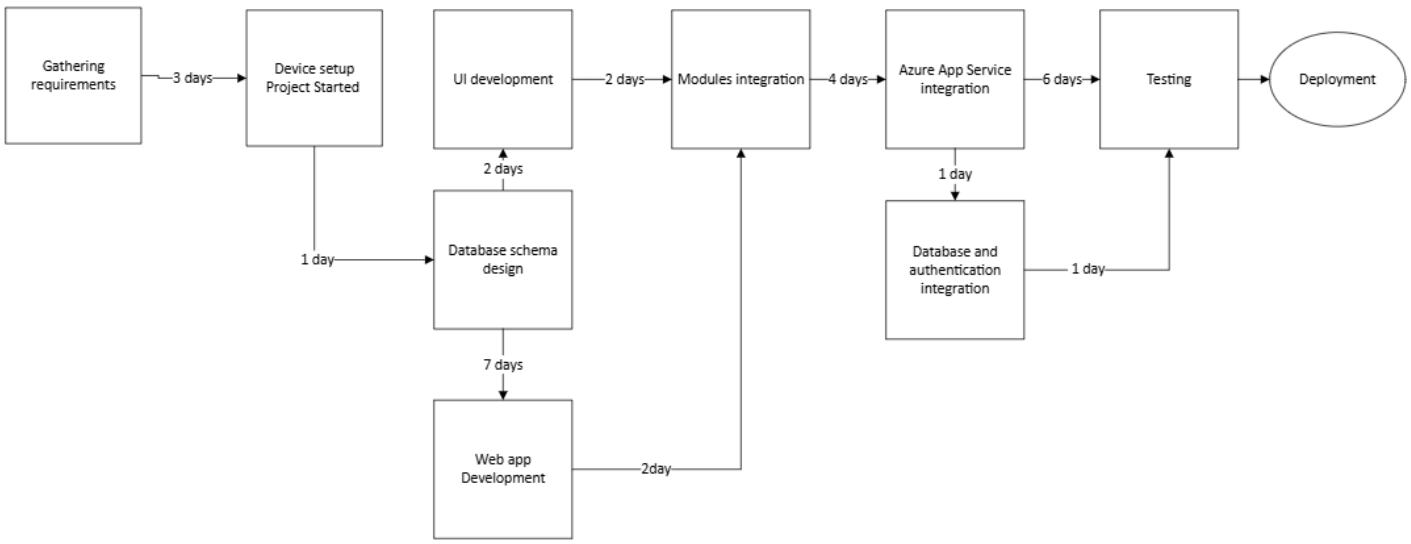


Fig-2: PERT Chart

1.5. ER Diagram: -

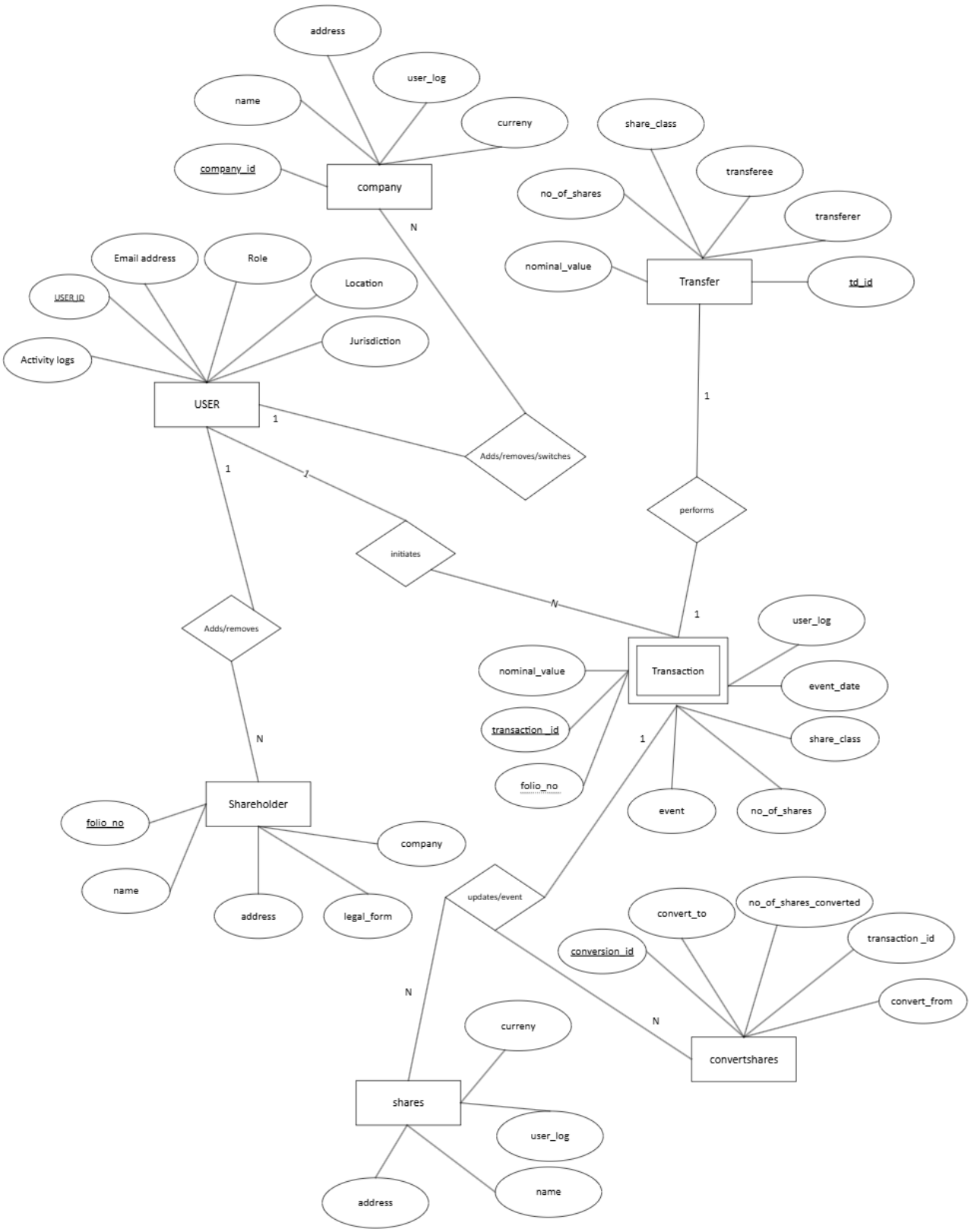


Fig-3: ER Diagram

Chapter 2: Softwares and Tools

2.1. Software

Software is the set of instructions, data, or programs that directs a computer to perform specific tasks, operations, or functions. It is a fundamental component of a computer system, providing the logic and control that enable hardware to be useful. In essence, software comprises the non-tangible elements of a computer, contrasting with the physical hardware components.

Software can be broadly categorized into several types, each serving distinct purposes:

- **System Software:** This type of software manages and controls the computer's hardware and provides a platform for application software to run. Operating systems (like Windows, macOS, and Linux) are prime examples of system software. They handle tasks such as memory management, process scheduling, file system organization, and input/output operations. System software also includes utilities like device drivers, which enable communication between the operating system and hardware devices, and system tools, which aid in tasks like system configuration and maintenance.
- **Application Software:** Application software is designed to perform specific tasks or provide services directly for the user. This category encompasses a wide range of programs, including:
    - Productivity software (e.g., word processors, spreadsheets, presentation software)
    - Communication software (e.g., email clients, web browsers, instant messaging apps)
    - Creative software (e.g., graphic design tools, video editing software, music production software)
    - Business software (e.g., customer relationship management (CRM) systems, enterprise resource planning (ERP) systems)
    - Entertainment software (e.g., video games, media players)

The development of software involves a systematic process that includes designing, coding, testing, and debugging. Various programming languages, such as Python, Java, C++, and JavaScript, are used to write software code. The complexity of software can vary significantly, from small programs with a few lines of code to large, complex systems with millions of lines of code.

2.1.1. Software Requirements Specification

- 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.
- Database: Microsoft SQL

2.1.2. Brief explanation on required software packages

Python (Django):

Python serves as the core programming language, providing the foundation for the Django web framework. Django is a high-level, open-source framework that follows the Model-View-Template (MVT) architectural pattern. It streamlines web development by offering built-in features for URL routing, database interaction through an Object-Relational Mapper (ORM), form handling, and security. Django's emphasis on convention over configuration promotes rapid development and clean, maintainable code. In this project, Django handles the application's backend logic, managing data storage, processing, and retrieval.

To elaborate further, here's a more detailed look at Python and Django:

Python:

- A versatile, interpreted, high-level programming language.
- Known for its clear syntax and readability, which makes it easier to learn and use.
- Supports a wide range of programming paradigms, including imperative, object-oriented, and functional programming.
- Has a large and comprehensive standard library, providing modules for various tasks such as string processing, web services, operating system interfaces, and database connections.
- Benefits from a vast and active community, which contributes to its extensive collection of third-party libraries and frameworks.

Django:

- A powerful web framework built on top of Python.
- Designed to simplify and accelerate the process of building complex web applications.
- Follows the Model-View-Template (MVT) architectural pattern:
    - **Model:** Defines the structure of the data and provides an interface for interacting with the database.
    - **View:** Handles the business logic of the application, processes requests, and interacts with the models to retrieve or modify data.
    - **Template:** Represents the presentation layer, defining how the data is displayed to the user (typically using HTML).
- Key features of Django:
    - **ORM (Object-Relational Mapper):** Allows developers to interact with the database using Python code instead of writing raw SQL queries. This simplifies database operations and makes the application more portable across different database systems.
    - **URL Routing:** Provides a clean and elegant way to map URLs to specific views, defining the application's URL structure.
    - **Forms:** Simplifies the process of creating, validating, and processing HTML forms, including protection against common security vulnerabilities.

- **Security:** Includes built-in protection against common web attacks such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- **Authentication and Authorization:** Provides a robust system for managing user accounts, authentication, and permissions, making it easy to control access to different parts of the application.
- **Admin Interface:** Offers a powerful and customizable admin interface for managing the application's data, making it easy for administrators to create, edit, and delete records.
- **Templating Engine:** Uses a flexible and extensible templating system (like Jinja, which is used in this project) for generating dynamic HTML content.

In the context of the portfolio management application, Django plays a crucial role in:

- Defining the data models for companies, shareholders, and transactions.
- Handling user authentication and authorization through integration with Azure App Services.
- Implementing the application's business logic, such as recording transactions, calculating portfolio holdings, and generating reports.
- Providing a secure and efficient way to interact with the Microsoft SQL database.
- Routing user requests to the appropriate views and generating the HTML responses.

HTML (Hypertext Markup Language):

- The standard markup language for creating web pages.
- Defines the structure and content of a web page (e.g., headings, paragraphs, images, links).
- Used to create the structure of the user interface for the portfolio management application. HTML provides the basic building blocks for the web pages that users will interact with.

  HTML (Hypertext Markup Language) - Detailed Explanation

  HTML is the foundation of all web pages. It's a markup language, not a programming language, which means it uses tags to structure content, rather than using code to perform actions. HTML tells web browsers how to display text, images, and other forms of multimedia on a web page.

  Key Concepts

  - **Elements:** HTML documents are made up of elements. Each element represents a part of the document, such as a paragraph, a heading, or a link.
  - **Tags:** Elements are denoted by tags. Tags are enclosed in angle brackets (< and >). Most elements have an opening tag (e.g.,) and a closing tag (e.g.,). The content of the element goes between the opening and closing tags. Some elements, like images (<img>), are self-closing and only have an opening tag.

- **Attributes:** Attributes provide additional information about an element. They are included in the opening tag and consist of a name and a value (e.g., <img src="image.jpg" alt="My Image">).
- **Structure:** HTML documents have a hierarchical structure. The <html> element is the root element and contains the <head> and <body> elements. The <head> element contains metadata about the document, such as the title, while the <body> element contains the visible content of the page.

Basic HTML Structure

Every HTML document follows this basic structure:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Page Title</title>
  <meta charset="UTF-8">
 </head>
 <body>
  <h1>Heading</h1>
  <p>Paragraph of text.</p>
 </body>
</html>
```

- <!DOCTYPE html>: This declaration specifies the document type and version of HTML being used (HTML5 in this case).
- <html>: The root element of the HTML document.
- <head>: Contains metadata about the document, such as the title, character set, and links to stylesheets and scripts.
- <title>: Specifies the title of the HTML page, which is displayed in the browser's title bar or tab.
- <meta>: Provides metadata about the HTML document. The charset attribute specifies the character encoding for the document (UTF-8 is the most common).
- <body>: Contains the visible content of the HTML page, such as headings, paragraphs, images, links, and forms.

Key HTML Elements

- **Headings (<h1> to <h6>):** Used to define headings of different sizes. <h1> is the largest and most important heading, while <h6> is the smallest.
- **Paragraphs (<p>):** Used to define paragraphs of text.
- **Links (<a>):** Used to create hyperlinks to other web pages or resources. The href attribute specifies the URL of the link.

- **Images (<img>):** Used to embed images in a web page. The src attribute specifies the URL of the image, and the alt attribute provides alternative text for screen readers and search engines.
- **Lists (<ul>, <ol>, <li>):** Used to create unordered lists (<ul>), ordered lists (<ol>), and list items (<li>).
- **Forms (<form>, <input>, <button>, etc.):** Used to create interactive forms for collecting user input.

HTML5 and Beyond

HTML has evolved significantly over the years. HTML5 is the latest major version and introduced many new elements and features, including:

- Semantic elements (<header>, <nav>, <article>, <section>, <footer>): Provide more meaningful structure to web pages, improving accessibility and SEO.
- Multimedia support (<audio>, <video>): Enables embedding audio and video content without relying on third-party plugins.
- Canvas and SVG: For drawing graphics and animations.
- Web storage and application cache: For storing data locally in the user's browser.

HTML is constantly evolving to meet the changing needs of the web. The WHATWG (Web Hypertext Application Technology Working Group) maintains the HTML Living Standard, which is an ongoing effort to update and improve the language.

Jinja:

Jinja is a powerful and flexible templating engine for the Python programming language. It allows developers to embed dynamic content into static HTML or other text-based documents. In the context of web development, Jinja is commonly used with frameworks like Django to generate dynamic HTML web pages.

Core Concepts

At its core, Jinja works by processing templates, which are text files containing a combination of static content and dynamic placeholders. These placeholders are replaced with actual values during the rendering process. Jinja provides a specific syntax to define these placeholders and control how the dynamic content is handled.

Key Features and Syntax

- **Variables:** Variables are used to insert data into the template. They are denoted by double curly braces: {{ variable_name }}. When the template is rendered, variable_name is replaced with its corresponding value from the data passed to the template.
- **Expressions:** Jinja allows the use of expressions within the variable delimiters. These expressions can include arithmetic operations, comparisons, and function calls. For example, {{ 2 + 2 }} would output 4.

- **Control Flow:** Jinja provides tags for controlling the flow of execution within the template:
    - {% if condition %}, {% elif condition %}, {% else %}, {% endif %}: These tags allow for conditional rendering of content based on whether a given condition is true or false.
    - {% for item in sequence %}, {% endfor %}: These tags enable looping over sequences (like lists or tuples) and rendering content for each item in the sequence.
- **Filters:** Filters modify the values of variables before they are displayed. They are applied using the pipe symbol |. Jinja provides a wide range of built-in filters, such as:
    - capitalize: Capitalizes the first character of a string.
    - lower: Converts a string to lowercase.
    - upper: Converts a string to uppercase.
    - title: Converts a string to title case.
    - date: Formats a date object according to a specified format.
    - default: Returns a default value if the variable is undefined.
- **Template Inheritance:** This is a powerful feature that allows you to define a base template with a common layout and structure, and then extend it in other templates. This promotes code reuse and makes it easier to maintain a consistent look and feel across a website.
- **Macros:** Macros are similar to functions in programming languages. They allow you to define reusable snippets of template code that can be called with arguments.
- **Includes:** The {% include 'template_name.html' %} tag allows you to include another template within the current template. This is useful for reusing common elements like headers and footers.

Benefits of Using Jinja

- **Dynamic Content Generation:** Jinja makes it easy to generate dynamic HTML or other text-based content, allowing web applications to display data from databases, user input, and other sources.
- **Code Reusability:** Template inheritance, macros, and includes promote code reuse, reducing development time and improving maintainability.
- **Separation of Concerns:** Jinja helps to separate the presentation logic from the application logic, making the code more organized and easier to understand.
- **Security:** Jinja automatically escapes special characters, which helps to prevent cross-site scripting (XSS) vulnerabilities.
- **Readability:** Jinja's syntax is designed to be readable and intuitive, making templates easier to write and maintain.

Microsoft Azure provides a suite of cloud services that are essential for deploying and managing modern web applications. Here's an explanation of three key services: Azure App Services, Azure Container Registry, and Azure App Registration.

Azure App Services

Azure App Service is a platform-as-a-service (PaaS) that enables you to build and host web apps, mobile app backends, and RESTful APIs in the cloud. It supports various programming languages, including Python, Node.js, Java, .NET, and PHP. App Service simplifies the deployment and management of applications by providing a fully managed environment with built-in infrastructure maintenance, scalability, and security.

Key Features:

- **Multiple Language Support:** Supports a wide variety of programming languages and frameworks.
- **Managed Environment:** Azure handles infrastructure management, including server maintenance, patching, and scaling.
- **Scalability:** Easily scale your application up or out to handle increased traffic.
- **High Availability:** Provides built-in support for high availability and load balancing.
- **Security:** Offers security features such as SSL certificate management, authentication, and authorization.
- **Deployment Options:** Supports various deployment methods, including Git, FTP, continuous integration/continuous deployment (CI/CD), and container deployment.
- **Integration with Azure Services:** Seamlessly integrates with other Azure services, such as Azure Database, Azure Cache, and Azure CDN.
- **Diagnostics and Monitoring:** Provides tools for logging, monitoring, and debugging applications.

How it Works:

1. **Application Development:** You develop your application using your preferred programming language and framework.
2. **Deployment:** You deploy your application code to Azure App Service using one of the supported deployment methods.
3. **Hosting:** Azure App Service hosts your application on its managed infrastructure.
4. **Access:** Users can access your application over the internet via HTTP or HTTPS.

Azure Container Registry (ACR)

Azure Container Registry (ACR) is a managed, private registry service for storing and managing container images. It allows you to build, store, and manage Docker container images and other related artifacts. ACR is a secure and scalable solution for managing containerized applications.

Key Features:

- **Private Registry:** Provides a private and secure registry to store your container images.
- **Docker Compatibility:** Works with standard Docker tools, such as the Docker CLI.
- **Scalability and Performance:** Offers scalable storage and high-performance image retrieval.

- **Security:** Supports role-based access control (RBAC) and Docker Content Trust for enhanced security.
- **Geo-replication:** Enables you to replicate your registry to multiple Azure regions for faster access and high availability.
- **Integration with Azure Services:** Integrates with other Azure services, such as Azure Kubernetes Service (AKS), Azure App Service, and Azure Container Instances.
- **Automation:** Supports automated image builds and updates with ACR Tasks.

How it Works:

1. **Image Building:** You build your container images using Docker.
2. **Registry Creation:** You create an Azure Container Registry in your Azure subscription.
3. **Image Pushing:** You push your container images to your ACR instance.
4. **Image Pulling:** Azure services (like App Service or AKS) or other authorized systems pull the images from ACR to deploy and run your containerized applications.

Azure App Registration

Azure App Registration is the process of registering your application with the Microsoft identity platform. This allows your application to access Azure resources and enable user authentication and authorization. When you register an app, you create an identity for it, which enables it to authenticate with Azure Active Directory (Azure AD) and obtain tokens.

Key Features:

- **Application Identity:** Creates a unique identity for your application in Azure AD.
- **Authentication and Authorization:** Enables your application to authenticate users and access Azure resources.
- **Permissions and Consent:** Allows you to configure the permissions your application needs and manage user consent.
- **Single Sign-On (SSO):** Supports SSO, allowing users to sign in once and access multiple applications.
- **Multi-Factor Authentication (MFA):** Can be used in conjunction with MFA for enhanced security.
- **Conditional Access:** Works with Azure AD Conditional Access policies to enforce access controls.

How it Works:

1. **Registration:** You register your application in the Azure portal or using the Azure CLI or PowerShell.
2. **Authentication:** When a user tries to access application, the application redirects the user to the Microsoft identity platform for authentication.
3. **Authorization:** After successful authentication, the Microsoft identity platform issues an access token to your application.
4. **Resource Access:** Your application uses the access token to authenticate itself when accessing Azure resources.

Graph API and Authentication.

Azure App Service provides a robust and scalable platform for hosting web applications, including those built with the Django framework. When it comes to securing these applications and managing user identities, integrating with Microsoft's identity platform, powered by Azure Active Directory (Azure AD), offers a comprehensive and secure solution. The Microsoft Graph API plays a crucial role in this integration, acting as the gateway to interact with the data and intelligence within the Microsoft 365 ecosystem, including user profiles and organizational information managed by Azure AD.

Leveraging Azure App Service's built-in authentication and authorization features (often referred to as "Easy Auth") in conjunction with the Microsoft Graph API simplifies the process of adding secure authentication to Django web applications. Instead of implementing the complexities of OAuth 2.0 flows and handling token management directly within the Django application, App Service handles much of this heavy lifting. It acts as a reverse proxy, intercepting unauthenticated requests and redirecting users to the Microsoft identity platform for sign-in.

Here's a breakdown of how this process works and how the Microsoft Graph API becomes instrumental:

1. Azure AD as the Identity Provider:

The foundation of this authentication mechanism is Azure AD, Microsoft's cloud-based identity and access management service. It serves as the central directory for managing user accounts, groups, and permissions within an organization. When configuring authentication for an Azure App Service, you'll typically register an application within Azure AD. This registration provides the necessary credentials (client ID and client secret) for the App Service to interact with the Microsoft identity platform.

2. Configuring App Service Authentication:

Within the Azure portal, you can configure the authentication settings for your Django App Service. You'll choose Azure Active Directory as the identity provider and provide the details of your Azure AD application registration. App Service then handles the underlying OAuth 2.0 authentication flow. When an unauthenticated user attempts to access your Django application, App Service automatically redirects them to the Microsoft sign-in page.

3. User Sign-in and Token Issuance:

The user authenticates using their Azure AD credentials. Upon successful authentication, the Microsoft identity platform issues an ID token (containing claims about the user's identity) and an access token (granting limited access to specific resources). App Service intercepts these tokens.

4. App Service Validates and Stores Tokens:

App Service validates the received ID and access tokens against the Microsoft identity platform, ensuring their authenticity and integrity. It then securely stores these tokens, typically in its own session or using a configured token store.

5. Providing User Information to the Django App:

This is where the integration with the Microsoft Graph API becomes significant. While the ID token provides basic user information (like name and email), the access token can be used by the App Service (or your Django application, if configured) to query the Microsoft Graph API for richer user profile details and organizational context.

App Service can be configured to inject user claims from the ID token into the headers of the requests it forwards to your Django application. These headers typically include information like the user's name, email, and unique object ID in Azure AD. This allows your Django application to identify the authenticated user.

6. Accessing More User Information with the Microsoft Graph API:

For scenarios requiring more detailed user information beyond the basic claims in the ID token (e.g., job title, department, profile picture, manager, group memberships), your Django application can use the access token obtained by App Service to call the Microsoft Graph API.

To do this, you would typically:

- **Retrieve the Access Token:** App Service makes the access token available to your application, often through request headers or environment variables.
- **Use an MSAL Library:** Employ a Microsoft Authentication Library (MSAL) for Python within your Django application. MSAL simplifies the process of acquiring and managing access tokens and calling the Graph API.
- **Construct Graph API Requests:** Use the access token to make authenticated HTTP requests to specific Graph API endpoints (e.g., /v1.0/me for the signed-in user's profile, /v1.0/users/{id} for a specific user's details, /v1.0/me/photos/$value for the profile picture).
- **Process the Graph API Response:** Parse the JSON response from the Graph API to extract the desired user information and use it within your Django application (e.g., display the user's job title, personalize the user experience based on group membership).

Benefits of Using Azure App Service and Microsoft Graph API for Django Authentication:

- **Simplified Authentication Implementation:** App Service handles the complexities of the OAuth 2.0 flow, reducing the amount of code you need to write and maintain in your Django application.
- **Centralized Identity Management:** Leverages Azure AD for secure and consistent user management across your organization.

- **Enhanced Security:** Benefits from the security features of Azure AD and App Service, including token validation and secure storage.
- **Rich User Information:** The Microsoft Graph API provides access to a wealth of user profile data and organizational context, enabling personalized and context-aware applications.
- **Scalability and Reliability:** Azure App Service offers a scalable and highly available platform for your Django application and its authentication needs.
- **Integration with Microsoft Ecosystem:** Seamlessly integrates your Django application with other Microsoft services and data.

MSAL

Integrating robust authentication into a Django web application hosted on Azure App Service often involves leveraging the power of Microsoft's identity platform (Azure Active Directory - Azure AD) and the Microsoft Graph API. While Azure App Service's built-in authentication ("Easy Auth") provides a simplified approach, using the Microsoft Authentication Library (MSAL) middleware within your Django application offers greater flexibility and control, especially when needing to interact with the Microsoft Graph API for richer user information and scenarios beyond basic authentication.

Here's a detailed explanation of how Azure App Service, the Microsoft Graph API, and an MSAL middleware (like msal-python integrated into a custom Django middleware) work together to secure and enhance your Django web application:

1. Azure AD as the Central Identity Provider:

As with the "Easy Auth" approach, Azure AD serves as the core identity provider. Your Django application, running on Azure App Service, will be registered as an application within Azure AD. This registration establishes a trust relationship, providing the necessary client ID and secret for authentication and authorization flows.

2. Disabling or Configuring App Service Authentication:

When using MSAL middleware within your Django application, you might choose to either disable the built-in App Service authentication entirely or configure it to act as a first line of defence, potentially simplifying the initial token acquisition. If enabled, App Service can still handle the initial OAuth 2.0 redirect and token exchange, passing validated user information to your Django application via headers. However, for more granular control and direct Graph API interaction, relying primarily on MSAL within Django is often preferred.

3. Implementing MSAL Middleware in Django:

The key to this approach lies in implementing a custom Django middleware that utilizes the msal-python library. This middleware intercepts incoming requests and handles the authentication flow. Here's a conceptual outline of its functionality:

- **Initialization:** The middleware is initialized with your Azure AD application registration details (client ID, client secret, authority URL), the necessary scopes for

accessing user information and the Graph API, and potentially a cache mechanism for storing tokens.

- **Request Interception:** For each incoming request, the middleware checks if the user is already authenticated (e.g., by checking for a valid session or access token).
- **Authentication Challenge:** If the user is not authenticated, the middleware initiates the OAuth 2.0 authorization code flow. This involves redirecting the user to the Microsoft sign-in page.
- **Authorization Code Handling:** After the user successfully authenticates on the Microsoft sign-in page, they are redirected back to your Django application with an authorization code. The middleware intercepts this code.
- **Token Acquisition:** The middleware uses the authorization code to exchange it for an ID token, an access token (for accessing resources like the Graph API), and a refresh token (for obtaining new access tokens without requiring the user to re-authenticate). This exchange happens by communicating with the Microsoft identity platform.
- **Token Storage:** The middleware securely stores these tokens, typically in the user's session or a dedicated token cache.
- **User Information Population:** The middleware extracts user claims from the ID token and makes them available to your Django views (e.g., through the request. User object).

4. Utilizing the Microsoft Graph API with the Access Token:

Once the user is authenticated and an access token has been acquired by the MSAL middleware, your Django application can use this token to interact with the Microsoft Graph API. This allows you to retrieve a wealth of information about the authenticated user and their organization.

- **Acquiring the Access Token:** Your Django views can access the stored access token (managed by the middleware).
- **Using MSAL to Call the Graph API:** The msal-python library provides convenient methods for making authenticated HTTP requests to the Graph API endpoints. You'll need to specify the required scopes when acquiring the initial token to ensure your application has the necessary permissions to access the desired Graph API data. Common endpoints include /v1.0/me (for the signed-in user's profile), /v1.0/users/{id} (for specific user details), /v1.0/me/photos/$value (for the profile picture), and /v1.0/groups (for group memberships).
- **Processing Graph API Responses:** Your Django views will receive JSON responses from the Graph API containing the requested user information. You can then parse this data and use it within your application (e.g., display the user's job title, personalize content based on group membership, show their profile picture).

5. Azure App Service's Role:

While the MSAL middleware handles the core authentication logic within your Django application, Azure App Service still provides a crucial hosting environment. It offers:

- **Scalability and Reliability:** Ensuring your Django application can handle varying levels of traffic and remains highly available.
- **Security Infrastructure:** Providing the underlying security for your application and the storage of sensitive information.
- **Integration with Azure Services:** Facilitating connections to other Azure services, including Azure AD (although the direct interaction is managed by MSAL).
- **HTTPS and SSL Management:** Handling the secure communication with users.

Benefits of Using MSAL Middleware and Graph API in Django on Azure App Service:

- **Greater Control over Authentication Flow:** You have more flexibility to customize the authentication process and handle specific scenarios.
- **Direct Graph API Interaction:** Enables seamless retrieval of rich user information and interaction with other Microsoft 365 services.
- **More Granular Permission Management:** You can request specific scopes for the Graph API, ensuring your application only asks for the necessary permissions.
- **Customizable Token Handling:** You have more control over how tokens are stored and managed within your Django application.
- **Standard Authentication Libraries:** MSAL is a well-maintained and secure library adhering to industry best practices for authentication.

Database

Azure SQL is a family of fully managed, secure, and intelligent SQL cloud database services offered by Microsoft Azure. Built upon the familiar SQL Server engine, it provides a range of deployment options tailored to different application needs and migration strategies. By abstracting away infrastructure management, Azure SQL allows businesses to focus on application development and innovation while benefiting from the scalability, high availability, and security of the Azure cloud.

The Azure SQL family primarily consists of three main offerings: **Azure SQL Database**, **Azure SQL Managed Instance**, and **SQL Server on Azure Virtual Machines**. Each option provides a distinct balance between management overhead, compatibility with on-premises SQL Server, and cost.

Azure SQL Database is a fully managed Platform-as-a-Service (PaaS) offering that provides a logical SQL server to host single databases or elastic pools of databases. It's designed for modern cloud applications and microservices that require a scalable and highly available relational database. Key features include:

- **Serverless Compute:** Automatically scales compute resources based on workload demand and pauses during inactive periods, optimizing costs.
- **Hyperscale:** Offers rapid scaling up to 100 TB, with fast backup and restore capabilities, and scale-out support for read-heavy workloads using up to 30 read replicas.
- **Built-in Intelligence:** Leverages AI to provide automatic performance tuning recommendations, identify potential threats, and improve data protection.

- **High Availability:** Guarantees up to 99.995% availability with built-in redundancy and automated failover.
- **Comprehensive Security:** Includes features like advanced threat protection, data encryption at rest and in transit, dynamic data masking, and integration with Azure Active Directory for authentication.

Azure SQL Database offers two main deployment models:

- **Single Database:** A fully managed, isolated database ideal for cloud-native applications needing a single, reliable data source.
- **Elastic Pool:** A cost-effective solution for managing multiple databases with variable usage patterns by sharing a pool of resources.

Azure SQL Managed Instance is also a fully managed PaaS offering, but it provides near 100% compatibility with on-premises SQL Server, making it an ideal target for lift-and-shift migrations. It offers instance-level features and capabilities, closely mirroring the traditional SQL Server experience, including:

- **Broad SQL Server Compatibility:** Supports features like cross-database queries, linked servers, SQL Server Agent, Common Language Runtime (CLR) integration, and database mail.
- **Native Virtual Network (VNet) Integration:** Provides enhanced security and seamless connectivity with other Azure resources within a private network.
- **Simplified Migration:** Facilitates easier migration of existing SQL Server databases to the cloud with minimal application changes.
- **High Availability and Disaster Recovery:** Includes built-in high availability with automatic failover and options for configuring failover groups for multi-region disaster recovery.
- **Advanced Security:** Offers similar security features as Azure SQL Database, along with VNet integration for network-level isolation.

SQL Server on Azure Virtual Machines (VMs) provides an Infrastructure-as-a-Service (IaaS) option, allowing you to run full versions of SQL Server on Windows or Linux VMs in Azure. This offers the most control over the SQL Server environment but requires managing the operating system, patching, and backups. It's suitable for scenarios requiring OS-level access or when migrating applications with strict dependencies on specific SQL Server features not yet fully supported in Azure SQL Database or Managed Instance.

In summary, Azure SQL offers a comprehensive suite of cloud-based SQL services catering to various needs. Azure SQL Database is ideal for modern cloud applications demanding scalability and cost-efficiency, while Azure SQL Managed Instance provides a smooth migration path for existing SQL Server workloads with high compatibility. SQL Server on Azure VMs offers maximum control but with increased management responsibilities. Choosing the right Azure SQL option depends on factors like application requirements, migration strategy, desired level of management, and cost considerations.

Azure App Service Authentication Process using Django, MSAL, OAuth 2.0, and Microsoft Graph API for RBAC

This section explains the end-to-end authentication process for a Django web application hosted on Azure App Service. The application leverages **OAuth 2.0** protocol using **MSAL (Microsoft Authentication Library)** for sign-in and token management. It also integrates **Microsoft Graph API** to query and enforce **role-based access control (RBAC)**. Authentication is handled via **Azure Active Directory (Azure AD)**.

1. Application Registration in Azure Active Directory

Before any authentication can occur, the Django app must be registered in **Azure AD**:

- Navigate to **Azure Portal > Azure Active Directory > App registrations > New registration**.
- Define:
  - **Name**: e.g., MyDjangoApp.
  - **Redirect URI**: https://<your-app>.azurewebsites.net/auth/redirect.
  - Supported account types: choose based on your needs (e.g., single-tenant or multi-tenant).
- Once registered, note down the following:
  - **Application (client) ID**
  - **Directory (tenant) ID**
  - **Client Secret** (create under Certificates & secrets)

2. MSAL Setup in Django

Install the Python MSAL library:

```
bash
CopyEdit
pip install msal
```

Configure settings in your Django project (e.g., settings.py):

```
python
CopyEdit
AZURE_AD_CLIENT_ID = '<your-client-id>'
AZURE_AD_CLIENT_SECRET = '<your-client-secret>'
AZURE_AD_TENANT_ID = '<your-tenant-id>'
AZURE_AD_AUTHORITY = f"https://login.microsoftonline.com/{AZURE_AD_TENANT_ID}"
AZURE_AD_REDIRECT_URI = 'https://<your-app>.azurewebsites.net/auth/redirect'
AZURE_AD_SCOPE = ['User.Read', 'Directory.Read.All', 'GroupMember.Read.All']
```

Set up the MSAL ConfidentialClientApplication instance in your views:

```
python
CopyEdit
from msal import ConfidentialClientApplication

def _build_msal_app():
   return ConfidentialClientApplication(
```

```
    AZURE_AD_CLIENT_ID,
    authority=AZURE_AD_AUTHORITY,
    client_credential=AZURE_AD_CLIENT_SECRET
  )
```

---

3. OAuth 2.0 Authorization Flow

The Django app initiates an OAuth 2.0 **Authorization Code Grant flow**:

Step-by-step flow:

1. **User Clicks Login**: Redirect to Azure AD authorization endpoint:

```
python
CopyEdit
def login(request):
  auth_app = _build_msal_app()
  auth_url = auth_app.get_authorization_request_url(
    AZURE_AD_SCOPE,
    redirect_uri=AZURE_AD_REDIRECT_URI
  )
  return redirect(auth_url)
```

2. **User Authenticates with Azure AD**:
   - If successful, Azure AD redirects to the app's `redirect_uri` with an **authorization code**.
3. **App Exchanges Code for Access and ID Tokens**:

```
python
CopyEdit
def auth_redirect(request):
  code = request.GET.get('code')
  auth_app = _build_msal_app()
  result = auth_app.acquire_token_by_authorization_code(
    code,
    scopes=AZURE_AD_SCOPE,
    redirect_uri=AZURE_AD_REDIRECT_URI
  )
  # Store tokens and user info in session
  request.session['access_token'] = result.get('access_token')
  request.session['id_token_claims'] = result.get('id_token_claims')
```

4. **Access and Refresh Tokens**:
   - MSAL manages token caching.
   - Refresh tokens are used to obtain new access tokens when the old ones expire.

4. RBAC via Microsoft Graph API

After authentication, the app uses the **access token** to query **Microsoft Graph API** to determine the user's group membership or assigned app roles.

Querying Groups for RBAC:

```
python
CopyEdit
import requests

def get_user_groups(access_token):
    headers = {
        'Authorization': f'Bearer {access_token}'
    }
    response = requests.get(
        'https://graph.microsoft.com/v1.0/me/memberOf',
        headers=headers
    )
    if response.status_code == 200:
        return response.json()['value']
    return []
```

You can now map users to roles in your Django app based on the group IDs or app roles returned.

5. Logging and Diagnostics

Azure App Service offers detailed diagnostic tools:

- Enable **App Service Authentication Logs**:
  - Go to your Web App > Authentication > Enable logs.
  - Choose logging to **Application Insights** for centralized monitoring.
- **Django Logging**:
  - Configure logging in settings.py:

```
python
CopyEdit
LOGGING = {
    'version': 1,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/home/LogFiles/django.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
        },
    },
}
```

- **MSAL Logs**:
    - MSAL provides internal logging. Enable it for debugging authentication/token issues:

```
python
CopyEdit
import logging
logging.basicConfig(level=logging.DEBUG)
```

6. Security Best Practices

- **Use HTTPS-only** App Service configuration.
- Use **Managed Identity** for backend access (e.g., to Azure SQL or Key Vault).
- Store client secrets in **Azure Key Vault**, not in code.
- Limit permissions (Graph API scopes) to only what's required.
- Use **conditional access policies** for added security.

The Django application hosted on Azure App Service authenticates users using Azure AD via OAuth 2.0 with MSAL. After login, it uses access tokens to call Microsoft Graph API and determine the user's group or role for RBAC implementation. MSAL handles secure token management, while logs in both Django and Azure App Service help with auditing and debugging authentication flows. This setup ensures secure, centralized, and scalable authentication aligned with enterprise-grade identity standards.

Azure Active Directory (Azure AD) – A Comprehensive Overview

Azure Active Directory (Azure AD) is Microsoft's cloud-based identity and access management (IAM) service. It is designed to help organizations manage users, groups, and access to resources both in the cloud and on-premises. Azure AD enables employees to sign in and access external resources such as Microsoft 365, the Azure portal, SaaS applications, and internal resources such as apps on your corporate network or intranet, as well as any cloud apps developed by the organization.

Core Concepts

1. Identity

An identity in Azure AD refers to an object that can be authenticated by the directory. This includes users, groups, service principals (applications), and managed identities. Each identity is assigned attributes like name, email, location, roles, etc.

2. Authentication

Azure AD supports modern authentication protocols like **OAuth 2.0**, **OpenID Connect**, **SAML**, and **WS-Federation**. These protocols enable secure authentication for a wide variety of applications, including mobile, web, desktop, and cloud-based systems.

3. Authorization

Once authentication is complete, Azure AD handles authorization by issuing tokens (e.g., ID tokens, access tokens) that define what the user or application can do, typically by defining scopes or roles.

4. Tenants and Directories

A **tenant** in Azure AD represents a dedicated, isolated instance of the directory. Organizations typically have one tenant per company, and within the tenant, they can manage all users, apps, groups, and policies.

Key Features of Azure AD

1. Single Sign-On (SSO)

Azure AD enables **SSO**, which allows users to authenticate once and gain access to multiple apps and services without needing to sign in again. This improves productivity and enhances security by reducing password fatigue.

2. Multi-Factor Authentication (MFA)

Azure AD supports MFA to increase the security of user logins. It requires users to verify their identity using two or more methods (e.g., password + mobile verification).

3. Conditional Access

Conditional Access allows administrators to enforce access policies based on specific conditions such as user location, device compliance, or risk level. For example, you can require MFA if a user logs in from an unknown location.

4. Azure AD B2B and B2C

- **Azure AD B2B** (Business-to-Business): Enables organizations to share access to applications and services with guest users from any other organization.
- **Azure AD B2C** (Business-to-Consumer): Provides identity management for external consumer-facing applications, allowing users to sign up or sign in using local accounts or third-party identities like Google or Facebook.

5. Application Management

Azure AD provides a centralized portal to manage access to applications. Admins can register applications, configure SSO, manage API permissions, and control how users interact with those apps.

6. Directory Synchronization (Azure AD Connect)

Organizations with on-premises Active Directory can use **Azure AD Connect** to synchronize their identities to the cloud. This enables hybrid identity solutions where users can use the same credentials both on-premises and in the cloud.

7. Device Management and Identity Protection

Azure AD integrates with **Microsoft Intune** and **Endpoint Manager** to manage device compliance and automate actions based on risk assessments. **Azure AD Identity Protection** uses machine learning to detect risky behaviors like impossible travel or leaked credentials and can automatically enforce remediation.

How Azure AD Works

1. **User Sign-in**: A user accesses an application protected by Azure AD (e.g., Microsoft 365, a custom app).
2. **Authentication Request**: The app redirects the user to Azure AD for authentication.
3. **Credential Validation**: Azure AD validates the credentials and enforces policies like MFA or Conditional Access.
4. **Token Issuance**: Upon successful authentication, Azure AD issues a security token (e.g., OAuth 2.0 access token).
5. **Access Granted**: The application verifies the token and grants access to the user.

Integration with Microsoft Graph API

Azure AD is accessible through **Microsoft Graph API**, a unified endpoint that provides programmatic access to directory data. Admins and developers can use it to:

- Query users, groups, and roles.
- Automate provisioning and de-provisioning of users.
- Read security alerts and sign-in logs.
- Assign licenses and manage app roles.

For example, you can fetch user information using:
http
CopyEdit
GET https://graph.microsoft.com/v1.0/users
Authorization: Bearer <access_token>

Role-Based Access Control (RBAC)

Azure AD supports **RBAC** to control who can manage what within Azure resources. Roles can be assigned at various scopes (subscription, resource group, or resource level). Common roles include:

- **Owner**: Full access to resources.
- **Contributor**: Can create and manage resources but can't grant access.
- **Reader**: Can view everything but not make changes.

Custom roles can also be created to tailor permissions to specific organizational needs.

Azure AD Editions

Azure AD comes in different editions to suit various business requirements:

1. **Free**: Basic user and group management, SSO for 10 apps, limited reporting.
2. **Office 365 Apps**: Includes features needed for Microsoft 365 subscriptions.
3. **Premium P1**: Adds Conditional Access, self-service password reset, and hybrid identity support.
4. **Premium P2**: Adds Identity Protection, Privileged Identity Management (PIM), and advanced security reporting.

Security and Compliance

Azure AD is built with enterprise-grade security features and is compliant with major standards including:

- ISO 27001
- SOC 1, 2, and 3
- HIPAA

- GDPR

Key security features include:

- **Privileged Identity Management (PIM)**: Just-in-time role activation.
- **Access Reviews**: Periodic access validation.
- **Audit Logs and Sign-in Logs**: Visibility into sign-ins, failures, and suspicious activity.

Real-World Use Cases

- **Workforce Identity**: Manage employee access to Microsoft 365, Salesforce, Dropbox, and custom apps.
- **Customer Identity (B2C)**: Allow customers to sign in to e-commerce sites using Facebook or Google.
- **Partner Collaboration (B2B)**: Grant secure app access to external partners and vendors.

Azure Active Directory is a foundational service for modern IT environments, providing secure identity management and access control in the cloud. It empowers organizations with centralized identity governance, strong authentication, flexible access policies, and seamless integration with thousands of SaaS applications. By adopting Azure AD, businesses can enhance security, improve productivity, and reduce administrative overhead, making it a critical component of a modern cloud strategy.

2.2 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.

Based on the "Production Hardware Requirements" and the general principles of UAT environments, the hardware requirements for the full-scale cloud deployed application.

- **Database Server:**
    - **Processor:** Multi-core Intel Xeon or AMD EPYC processor.
    - **RAM:** 64 GB of ECC DDR4 RAM.
    - **Storage:**
        - Primary Storage: Fast, redundant storage subsystem, such as RAID 10 SSDs, for the database.
        - Secondary Storage: Sufficient storage for database backups.
    - **Network:** bandwidth (100 Mbps or faster) network connectivity.
    - **Operating System:** A robust server operating system (e.g., Linux - Red Hat Enterprise Linux, Ubuntu Server).
- **Application Server(s):**
    - **Processor:** Multi-core Intel Xeon or AMD EPYC processor.
    - **RAM:** 32 GB of ECC DDR4 RAM.
    - **Storage:** Fast SSD storage for the operating system.
    - **Network:** High-bandwidth (100 Mbps or faster) network connectivity.
    - **Operating System:** A robust server operating system (e.g., Linux - Red Hat Enterprise Linux, Ubuntu Server).
    - **Load Balancer:** A load balancer (hardware or software) may be needed if UAT involves testing with a realistic number of concurrent users. If not, a simple reverse proxy may fulfil the requirements.

2.3 DFD (Data Flow Diagram)

A **Data Flow Diagram (DFD)** is a graphical representation of the flow of data through an information system or business process. It provides a visual way to understand:

- **Data Sources and Destinations (External Entities):** Where data comes from and where it goes outside the system.
- **Processes:** The activities that transform data.
- **Data Stores:** Where data is held or stored.
- **Data Flows:** The paths that data takes between different components.

DFDs use a set of standard symbols to represent these components, making complex systems easier to understand. They are a crucial tool in system analysis.

2.3.1 Level 0: DFD

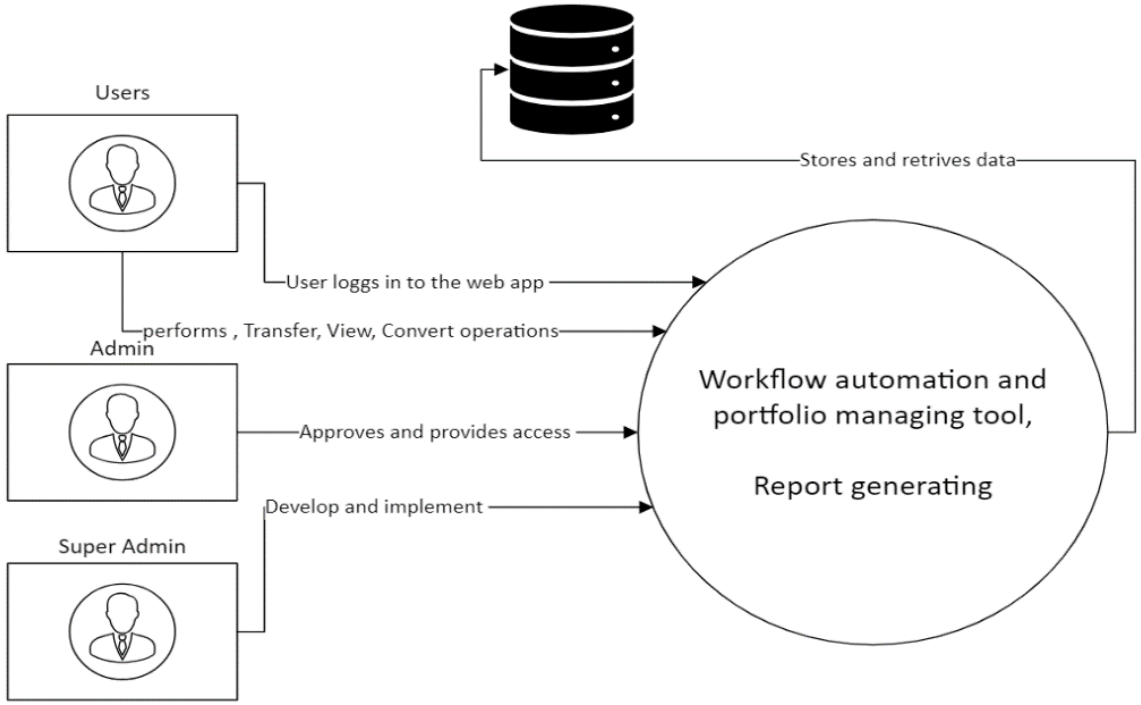


Fig-4: DFD Level 0

2.3.2 Level 1: DFD

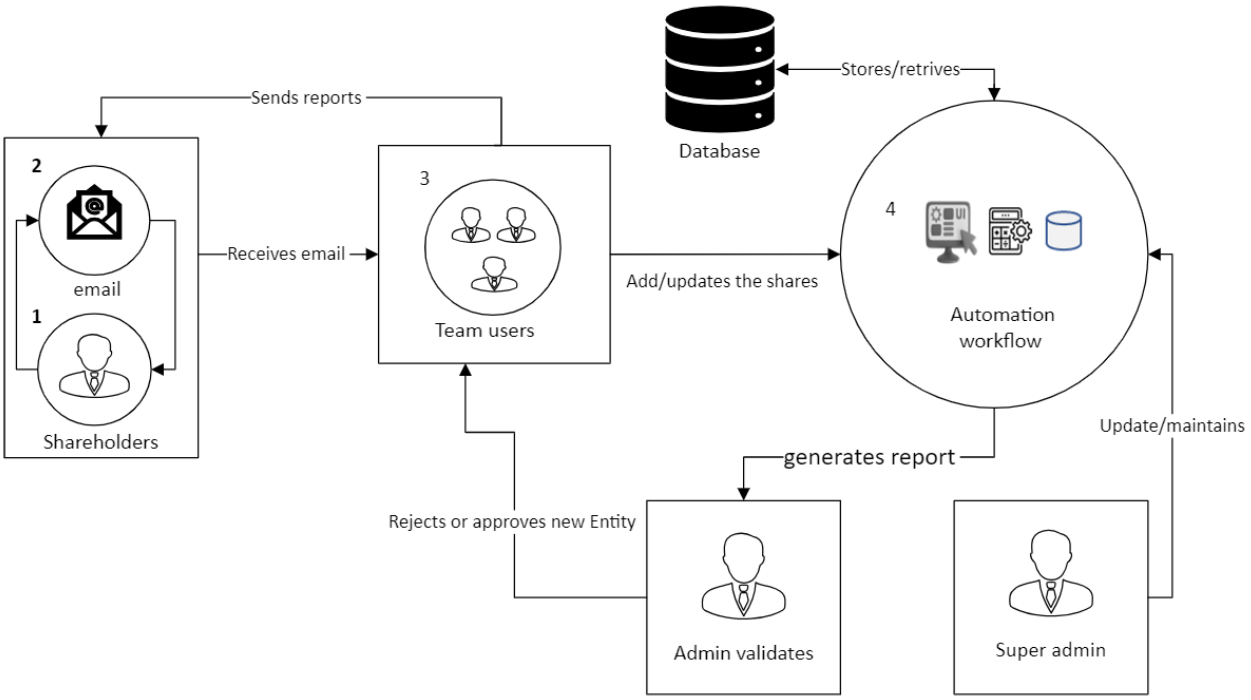


Fig-4: DFD Level 1

2.3.3 Level 2: DFD

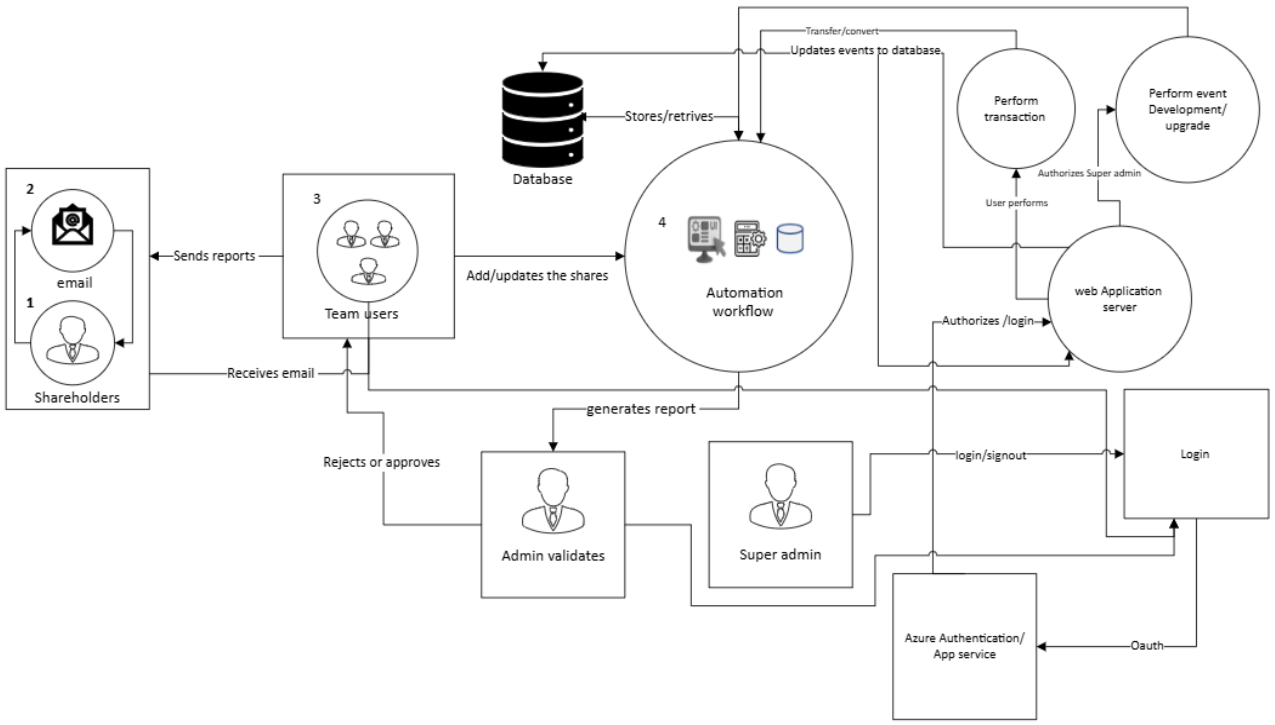


Fig-5: DFD Level 2

2.3.4 Use case diagram

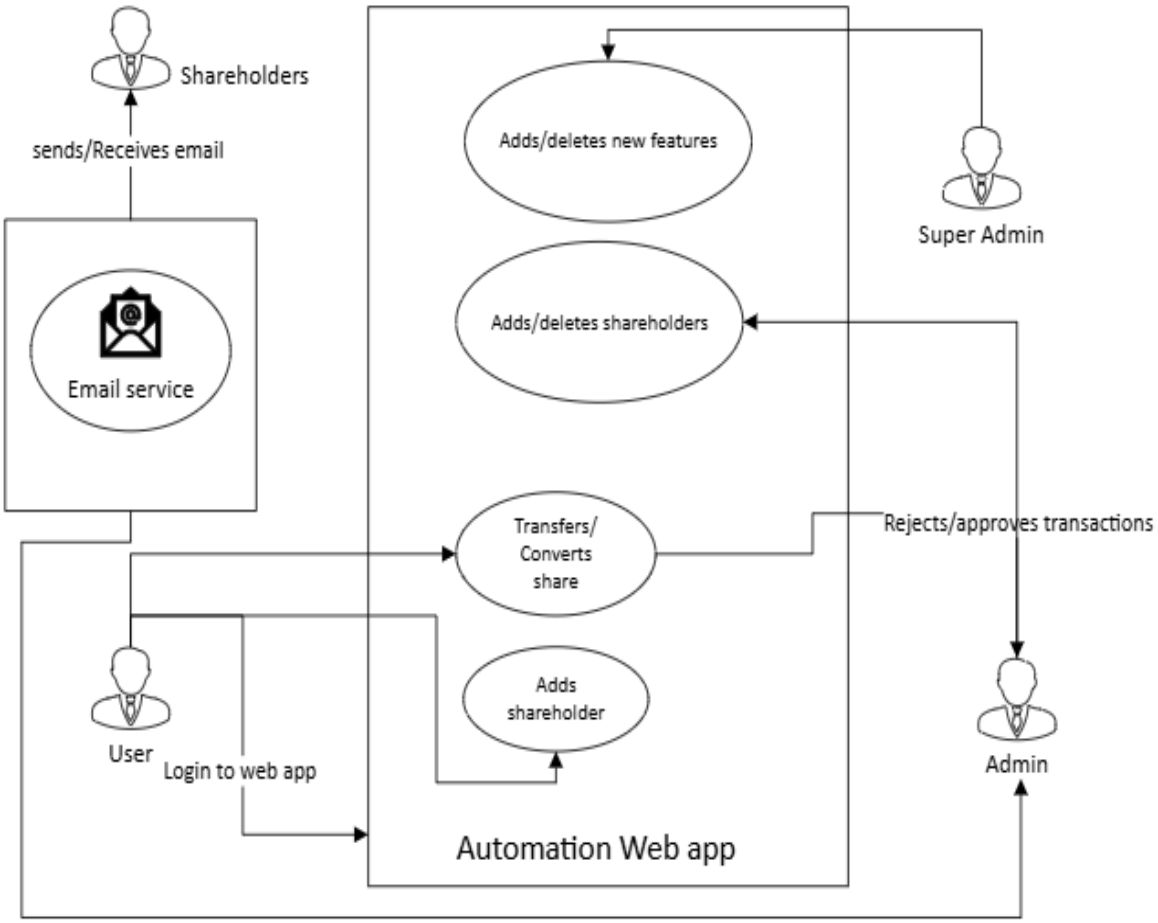


Fig-6: Use case Diagram.

2.3.5 Sequence diagram

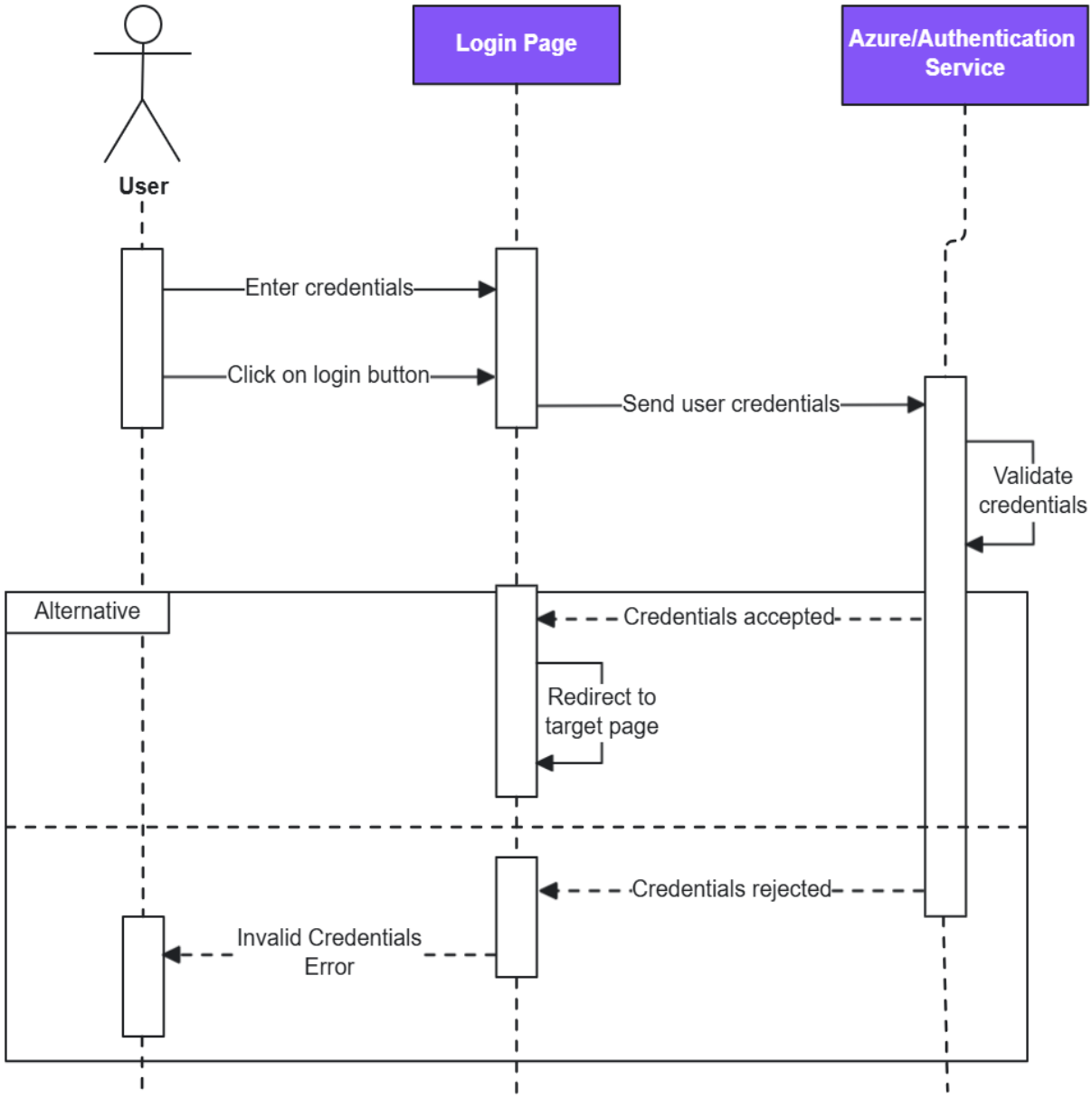


Fig-7: Sequence Diagram

2.3.6 Deployment diagram

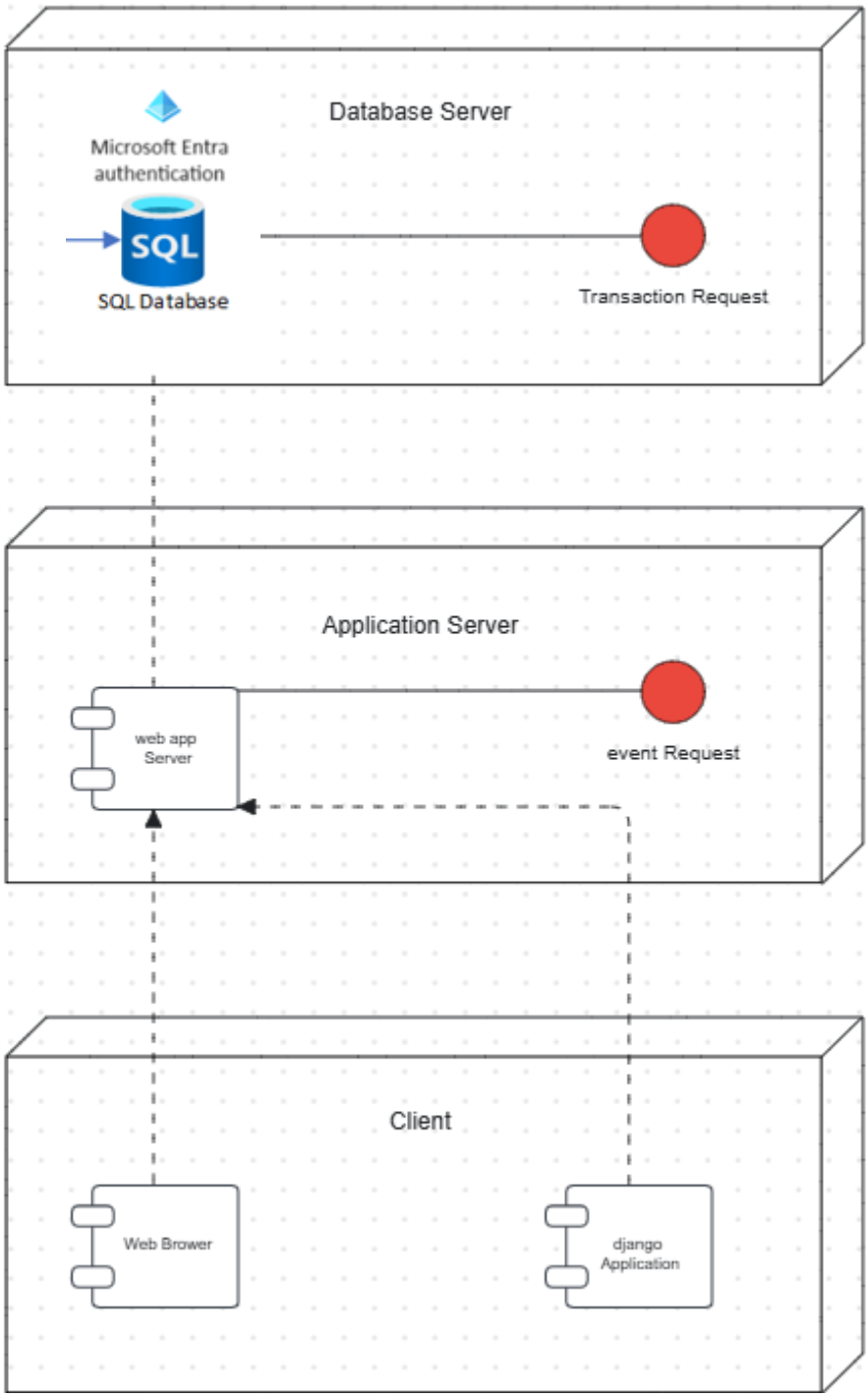


Fig-8: Deployment Diagram

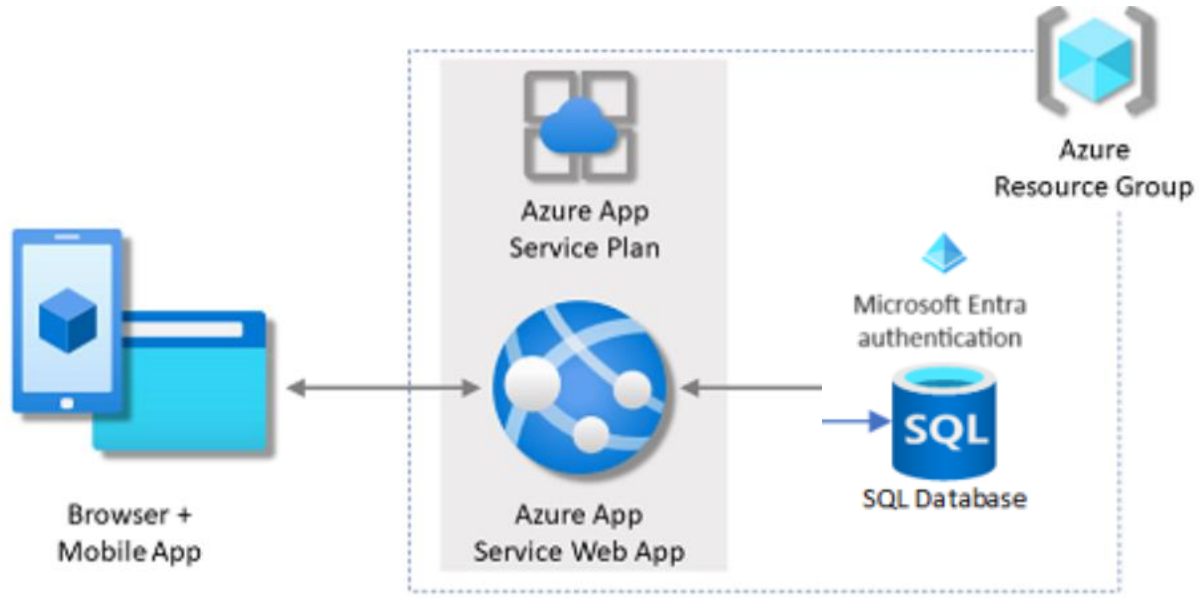


Fig-9: Deployment Diagram

2.3.7 Deployment diagram Microsoft SQL and Azure AAD

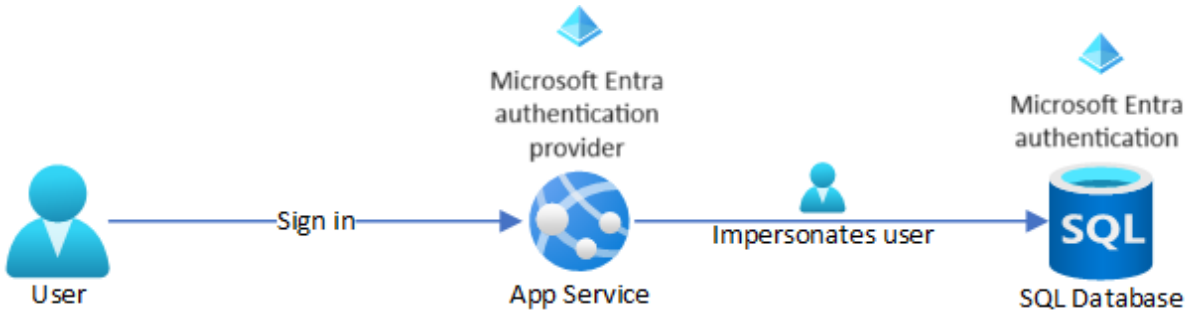


Fig-10: Deployment diagram Microsoft SQL and Azure AAD

2.4 Data Design

2.4.1 Schema Design

Table Company

Field Name	Data Type	Field Size	Description
company_id	BIGINT	8 bytes	Primary key, auto-incrementing
name	VARCHAR	255	
address	TEXT		Can store large text
user_log	VARCHAR	255	Allows NULL values
currency	VARCHAR	255	Allows NULL values

Table-1: Company Table

Table Shareholder

Field Name	Data Type	Field Size	Description
folio_no	BIGINT	8 bytes	Primary key, auto-incrementing
name	VARCHAR	255	
address	TEXT		Can store large text
legal_form	TEXT		Can store large text
company_id	BIGINT	8 bytes	Foreign key to Company table, allows NULL values

Table-2: Shareholder Table

Table Shares

Field Name	Data Type	Field Size	Description
share_id	BIGINT	8 bytes	Primary key, auto-incrementing
no_of_shares	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal
class_of_shares	TEXT		Stores large text
folio_no	BIGINT	8 bytes	Foreign key to Shareholder table

Table-3: Shares Table

Table Transferdetails

Field Name	Data Type	Field Size	Description
td_id	BIGINT	8 bytes	Primary key, auto-incrementing
transferer_id	BIGINT	8 bytes	Foreign key to ShareHolder table (related name: transfered_from)
transferee_id	BIGINT	8 bytes	Foreign key to ShareHolder table (related name: transfered_to)
share_class	VARCHAR	255	
no_of_shares	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal
nominal_value	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal, default value is 0.0

Table-4: Transferdetails Table

Table Transaction

Field Name	Data Type	Field Size	Description
transaction_id	BIGINT	8 bytes	Primary key, auto-incrementing
folio_no	BIGINT	8 bytes	Foreign key to ShareHolder table
event	VARCHAR	255	Stores a string, with a set of choices defined in event_choices, default value is 'A'
no_of_shares	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal
share_class	VARCHAR	255	default value is "Ordinary Shares"
nominal_value	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal, default value is 0.01
consideration	DECIMAL(15,4)		Stores decimal number with 15 total digits, 4 after the decimal
total	DECIMAL(15,2)		Stores decimal number with 15 total digits, 2 after the decimal
status	VARCHAR	255	
created_at	DATETIME		Stores date and time, automatically set on creation
event_date	DATETIME		Stores date and time
transfer_details_id	BIGINT	8 bytes	Foreign Key to TransferDetails, can be null
user_log	VARCHAR	255	Allows NULL values

Table-5: Transaction Table

Table ConvertDetails

Field Name	Data Type	Field Size	Description
conversion_id	BIGINT	8 bytes	Primary key, auto-incrementing
convert_from	VARCHAR	255	
convert_to	VARCHAR	255	
no_of_shares	DECIMAL (15,2)		Stores decimal number with 15 total digits, 2 after the decimal
transaction_id	BIGINT	8 bytes	Foreign key to Transaction table (related name: convert)

Table-6: Convertdetails Table

2.5 App UI

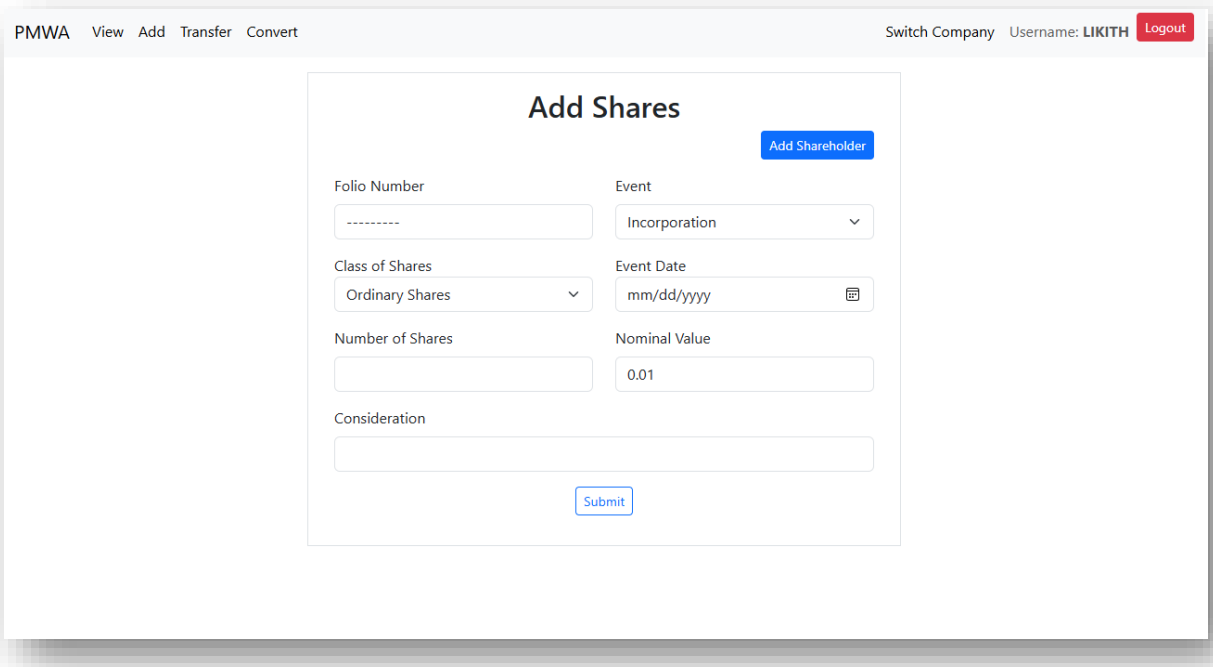


Fig-11: Add shares page.

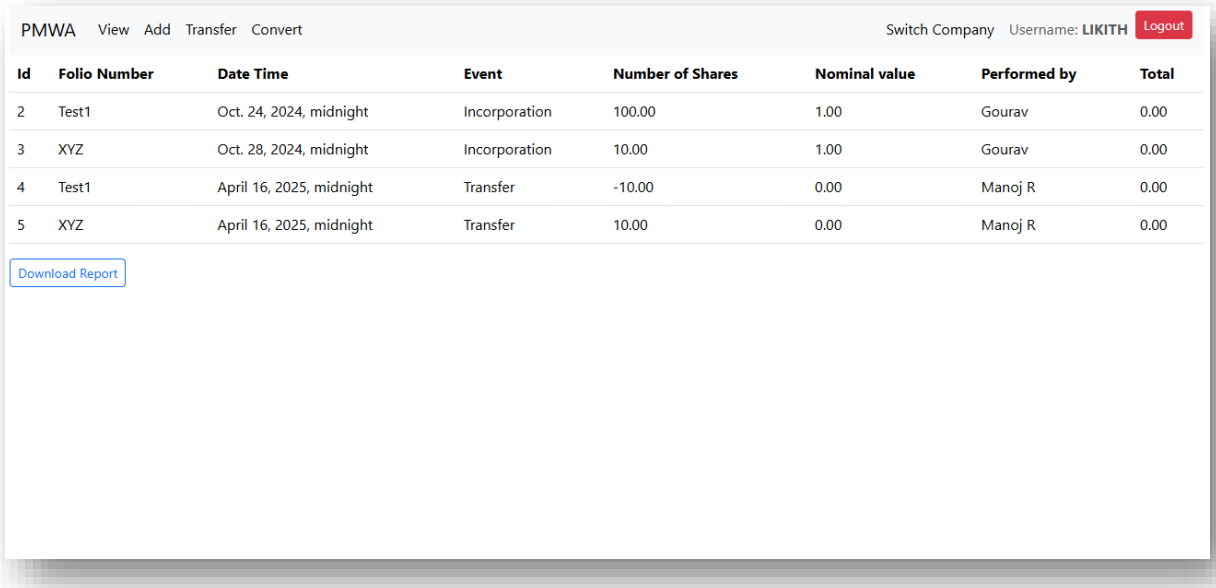


Fig-12: Transaction view page.

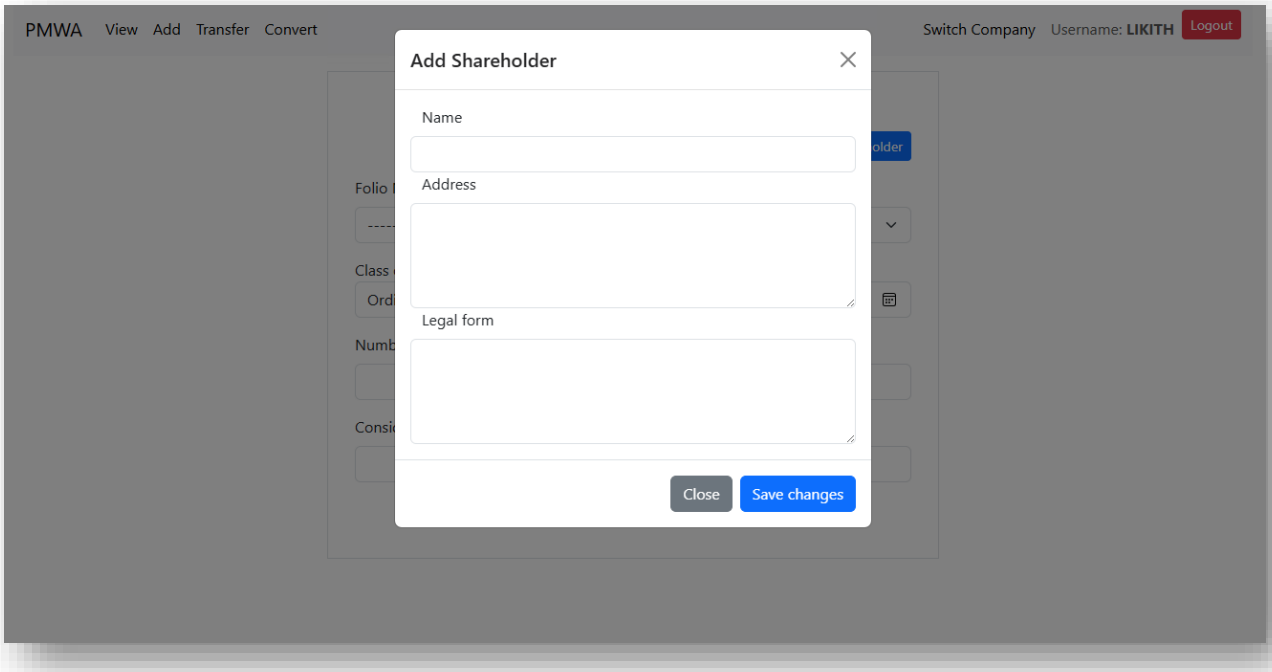


Fig-13: Add Shareholder Page

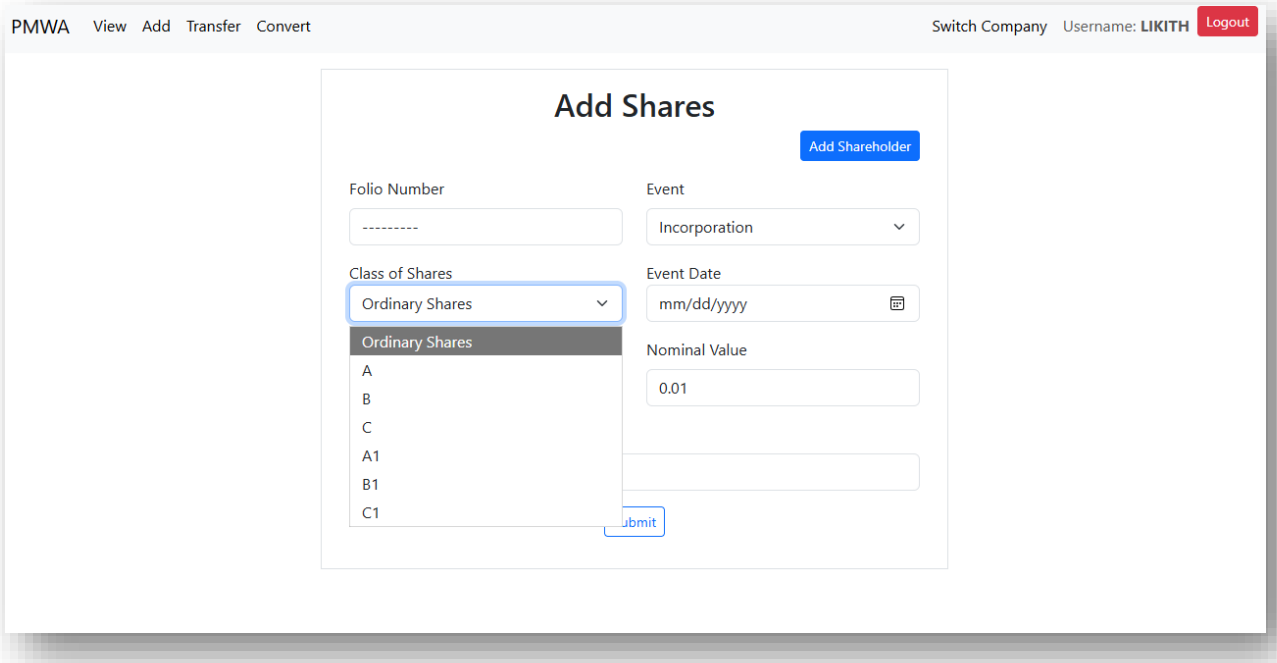


Fig-14: Share category Selection dropdown.

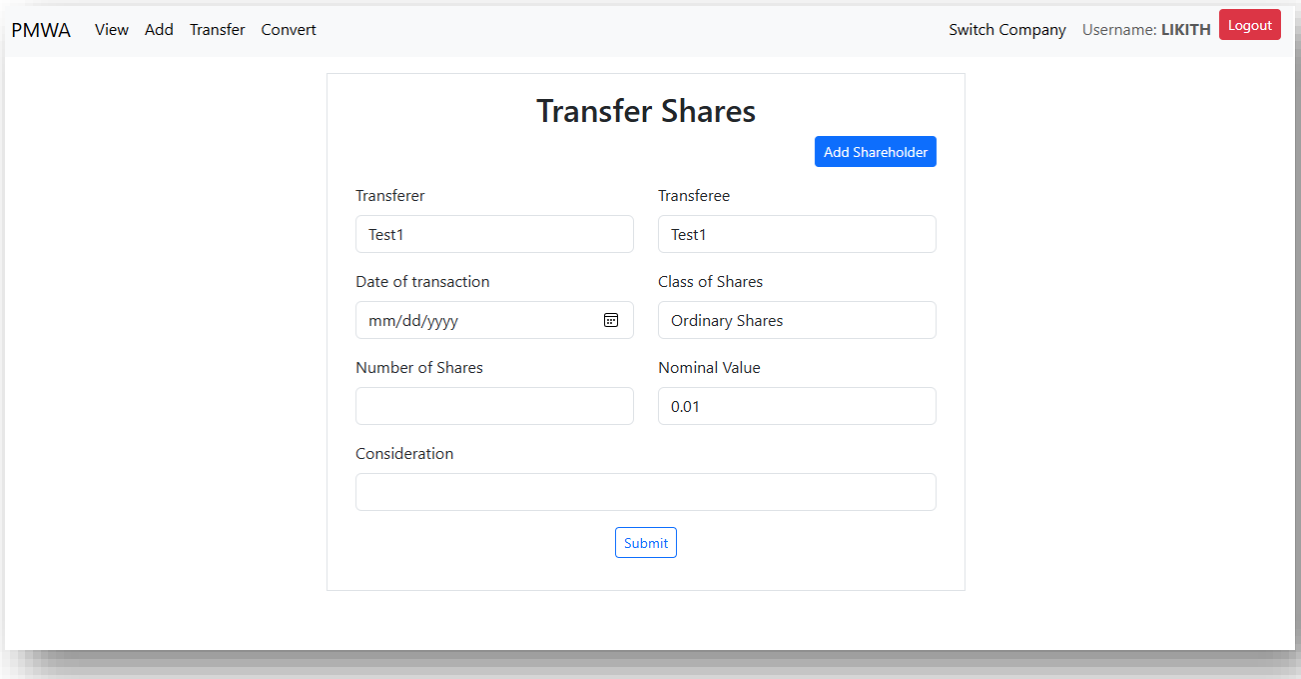


Fig-15: Transfer Share Page

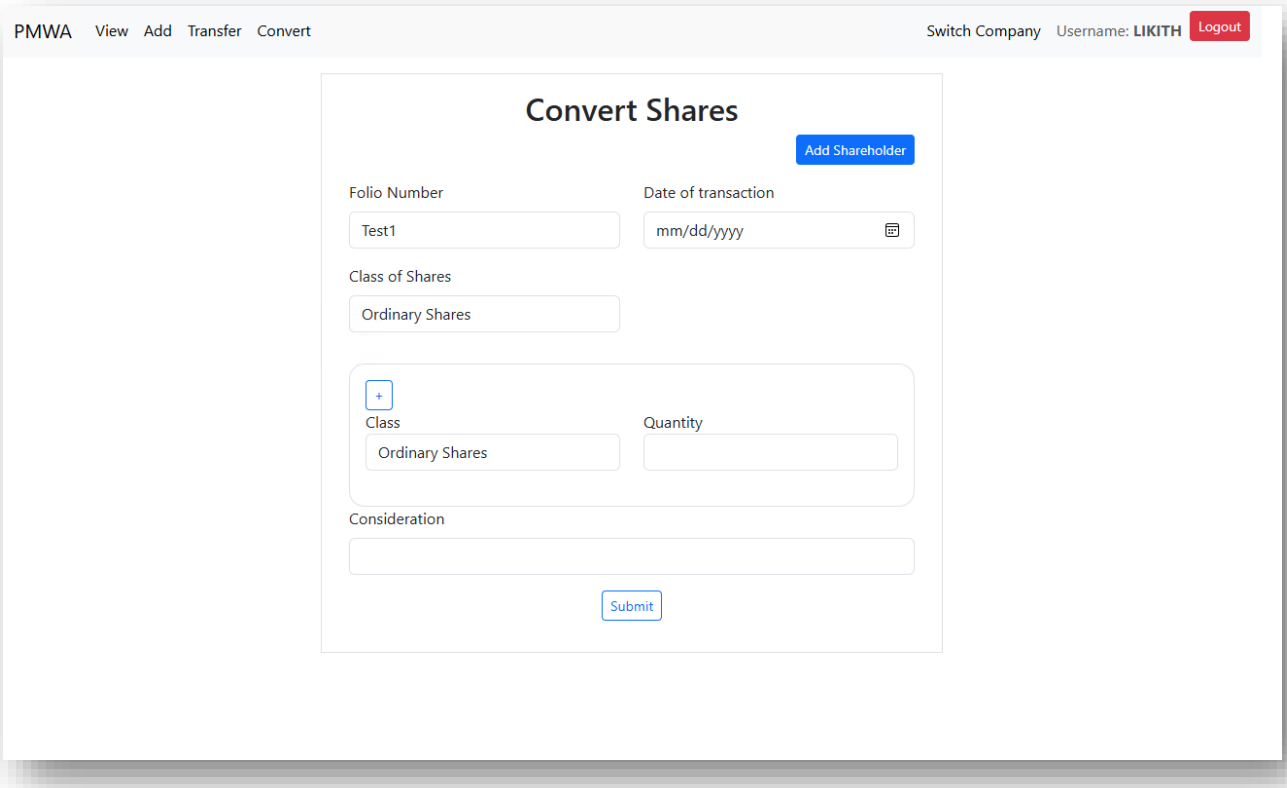


Fig-16: Convert shares Page

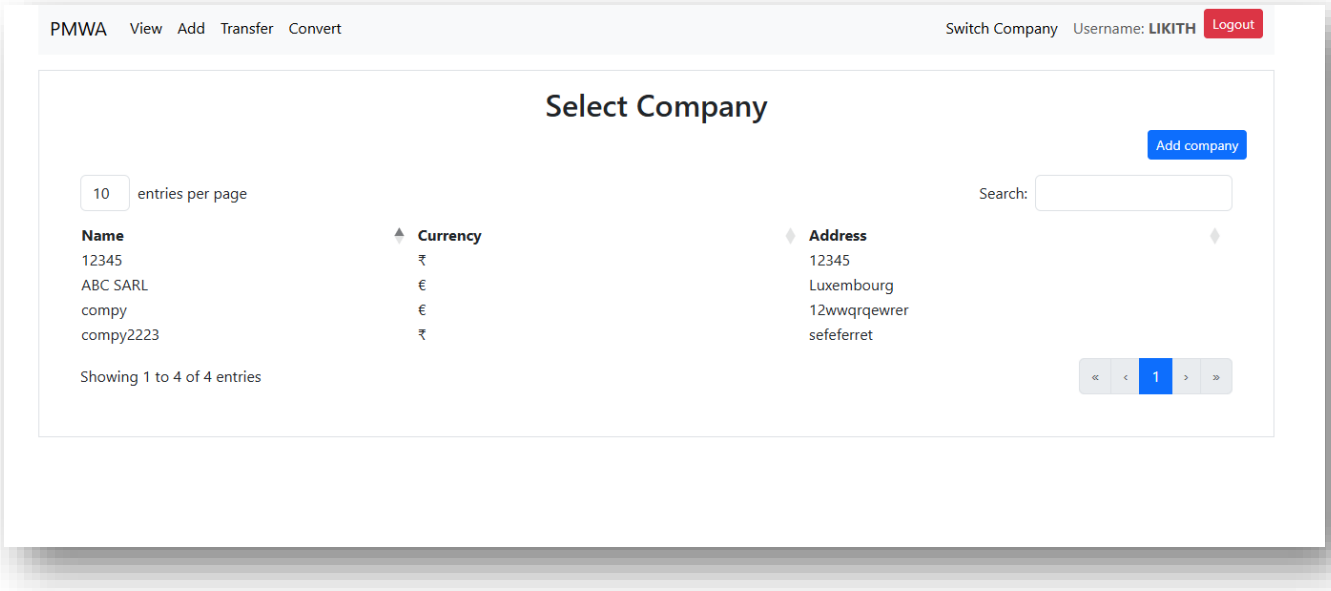


Fig-17: Select Company Page

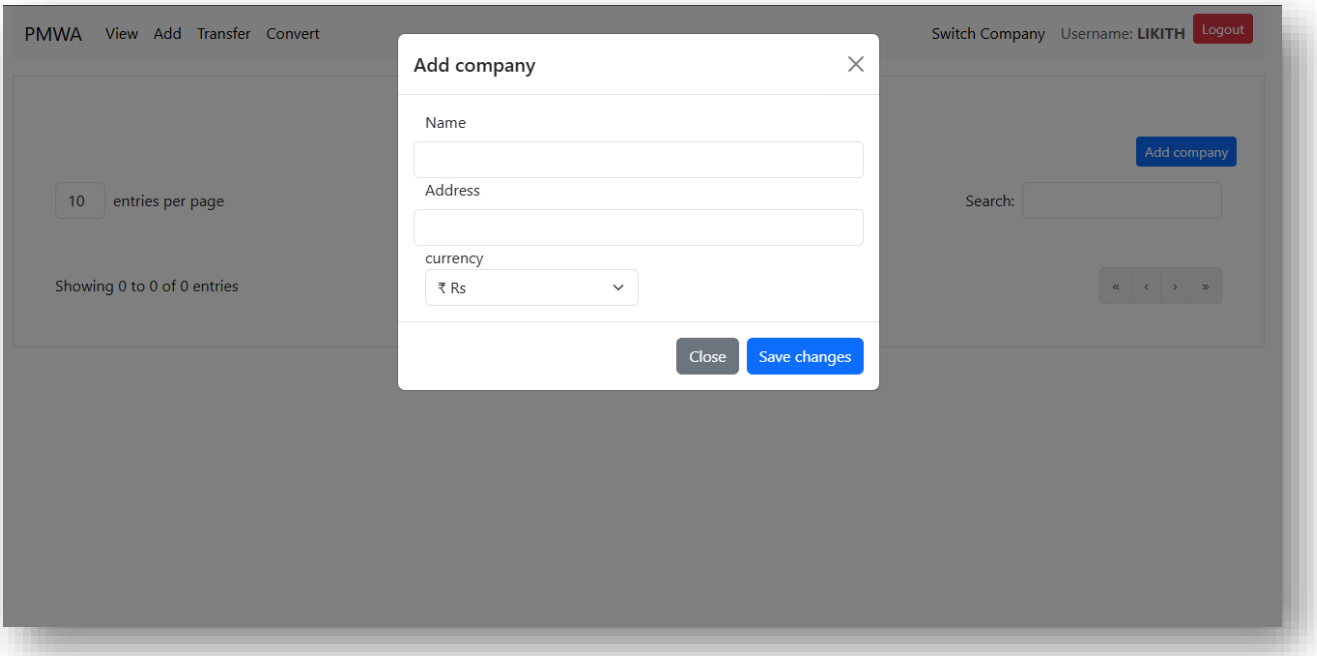


Fig-18: Add Company Page

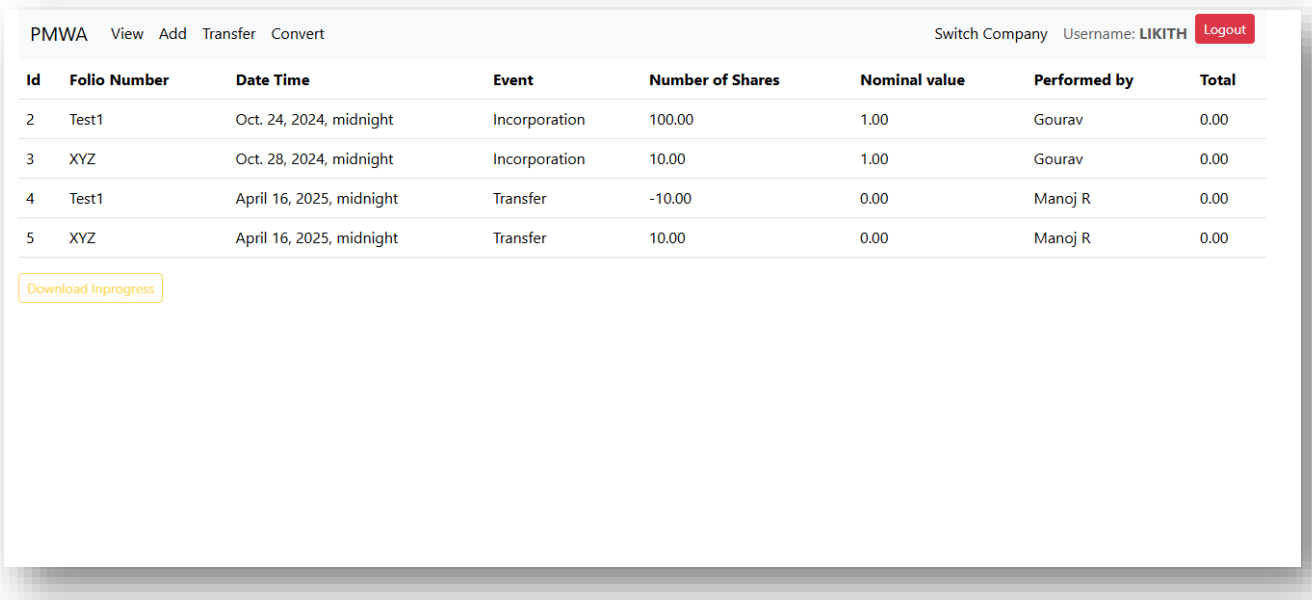


Fig-19: Report downloading event

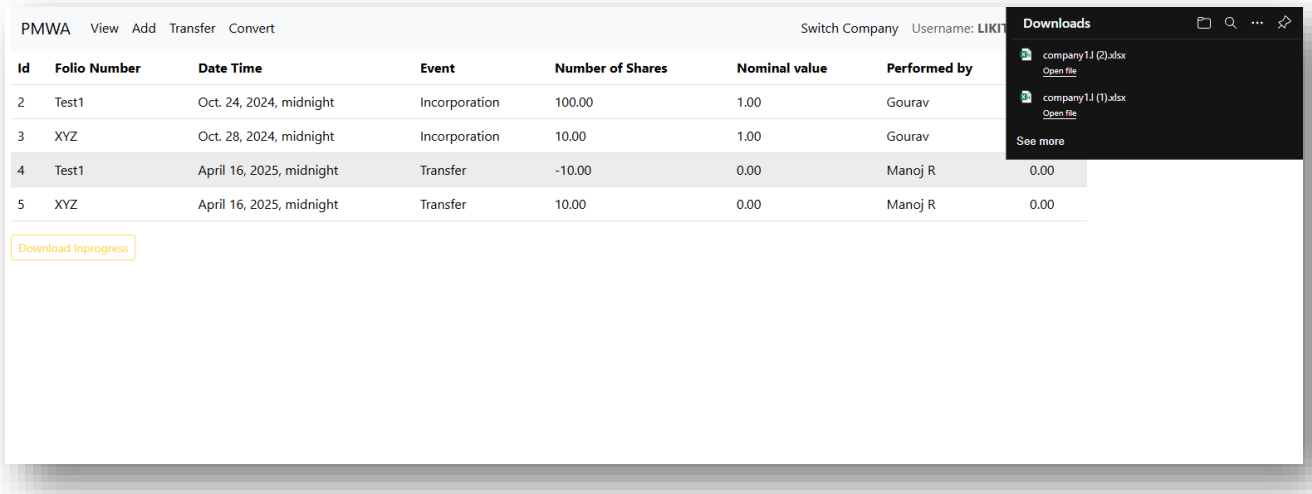


Fig-20: Report Downloaded

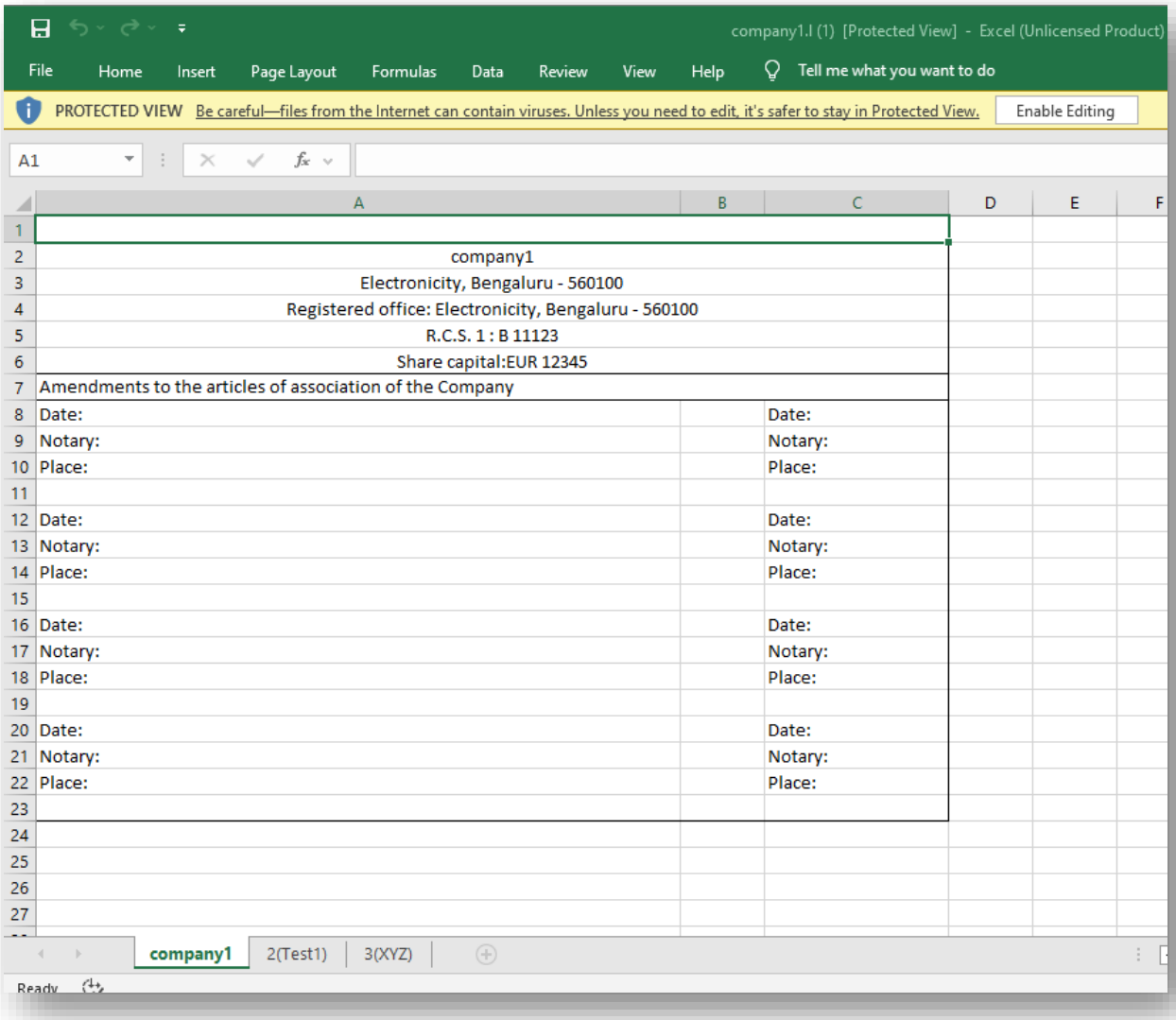


Fig-21: Formatted Excel Sheet

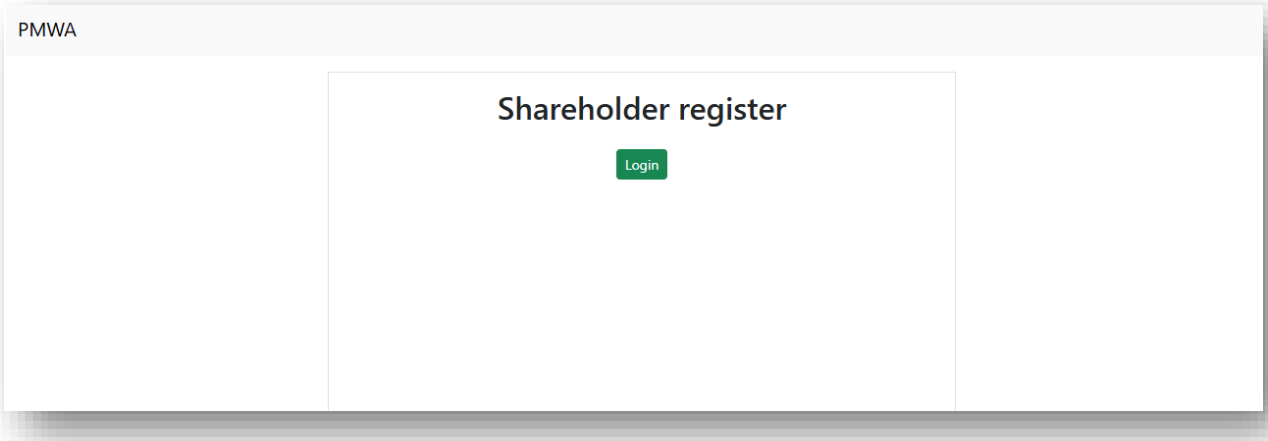


Fig-22: Login Page

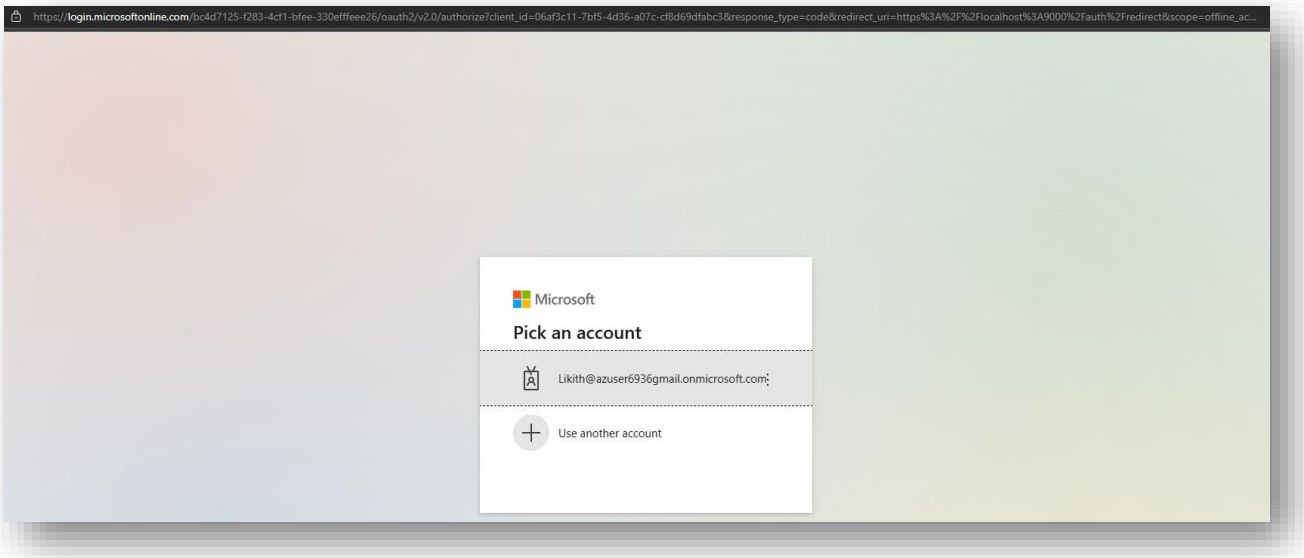


Fig-23: Azure app Service Authentication

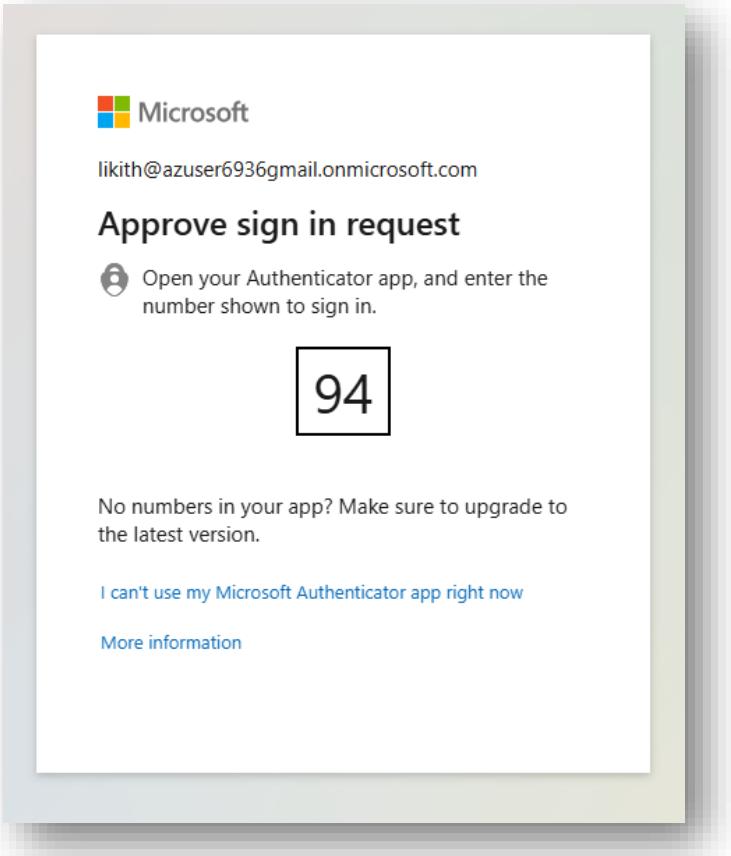


Fig-24: Multifactor Authentication

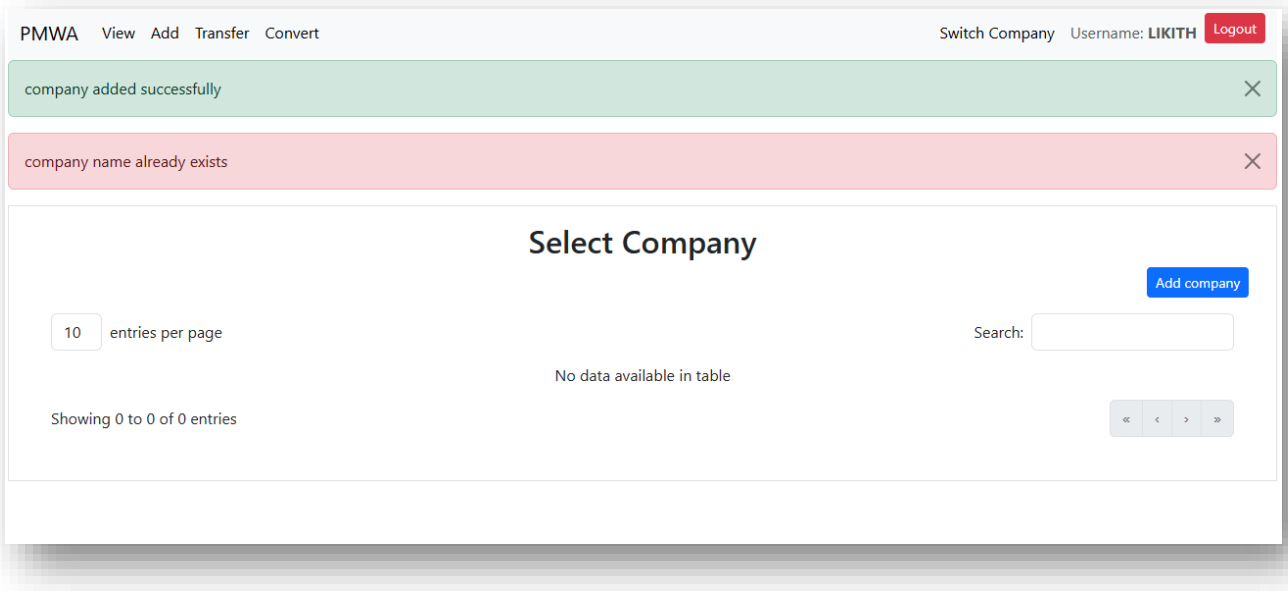


Fig-25: Company addition successful event

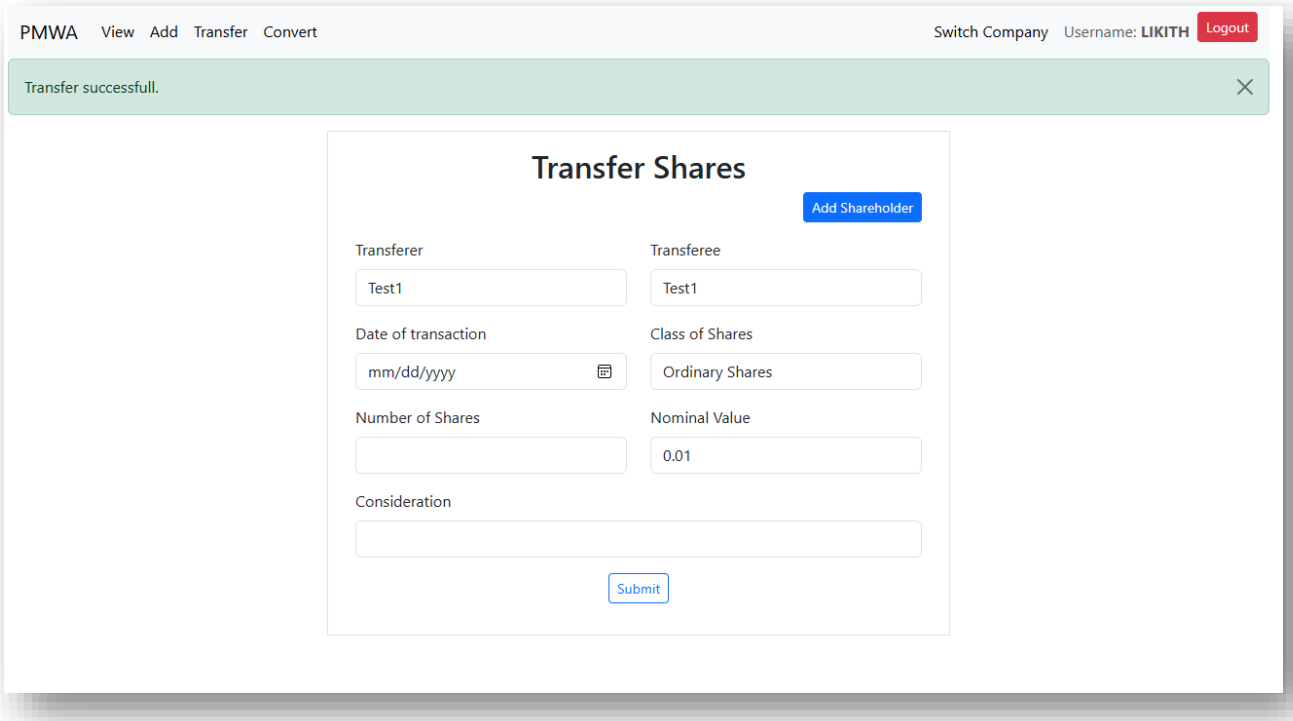


Fig-26: Transfer Successful event

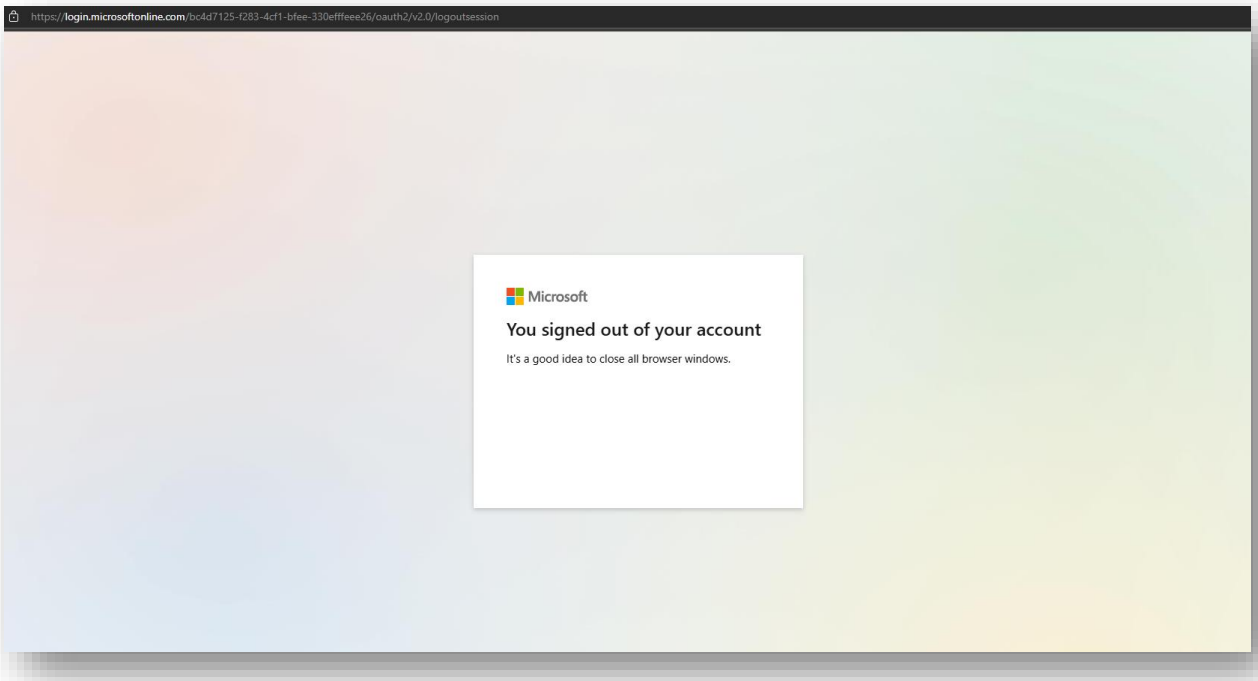


Fig-27: Logout Page

Chapter 3: Coding and Implementation

3.1 Implementation Approach

In software development, an **implementation approach** is the strategy or methodology you choose to convert your design into a working system. It's essentially *how* you go about building the software. This involves a series of decisions and processes, including:

- **Development Methodology:** The overall framework guiding the development process (e.g., Waterfall, Agile, Scrum).
- **Technology Stack:** The specific programming languages, frameworks, databases, and tools used.
- **Development Process:** The steps and activities involved in writing, testing, and integrating the code.
- **Deployment Strategy:** How the software will be released to users (e.g., phased rollout, big bang).

The implementation approach dictates the sequence of tasks, the way developers write and test code, how different parts of the system are integrated, and how the final product is delivered.

Advantages of a Well-Defined Implementation Approach

A well-defined implementation approach provides several advantages:

- **Structured Development:** It provides a clear roadmap for the development team, ensuring a systematic and organized process.
- **Improved Efficiency:** By defining the steps and processes in advance, developers can work more efficiently, reducing wasted effort and rework.
- **Better Project Management:** It facilitates project planning, tracking, and control, allowing project managers to monitor progress and identify potential issues.
- **Enhanced Code Quality:** A good approach often includes coding standards, best practices, and testing procedures, leading to higher-quality code.
- **Reduced Risk:** By addressing potential challenges and risks early on, a well-defined approach can help mitigate them and prevent project failures.
- **Increased Maintainability:** A structured approach often results in code that is easier to understand, modify, and maintain in the long run.
- **Improved Collaboration:** It provides a common understanding of the development process, fostering better communication and collaboration among team members.

The specific advantages depend on the chosen approach. For example:

- **Waterfall:** Provides clear stages and documentation, which is advantageous for projects with well-defined requirements.

- **Agile:** Offers flexibility and adaptability, which is advantageous for projects with evolving requirements.

For "Portfolio Manager and Workflow Automation" tool, the Waterfall implementation approach was chosen to provide a structured development process.

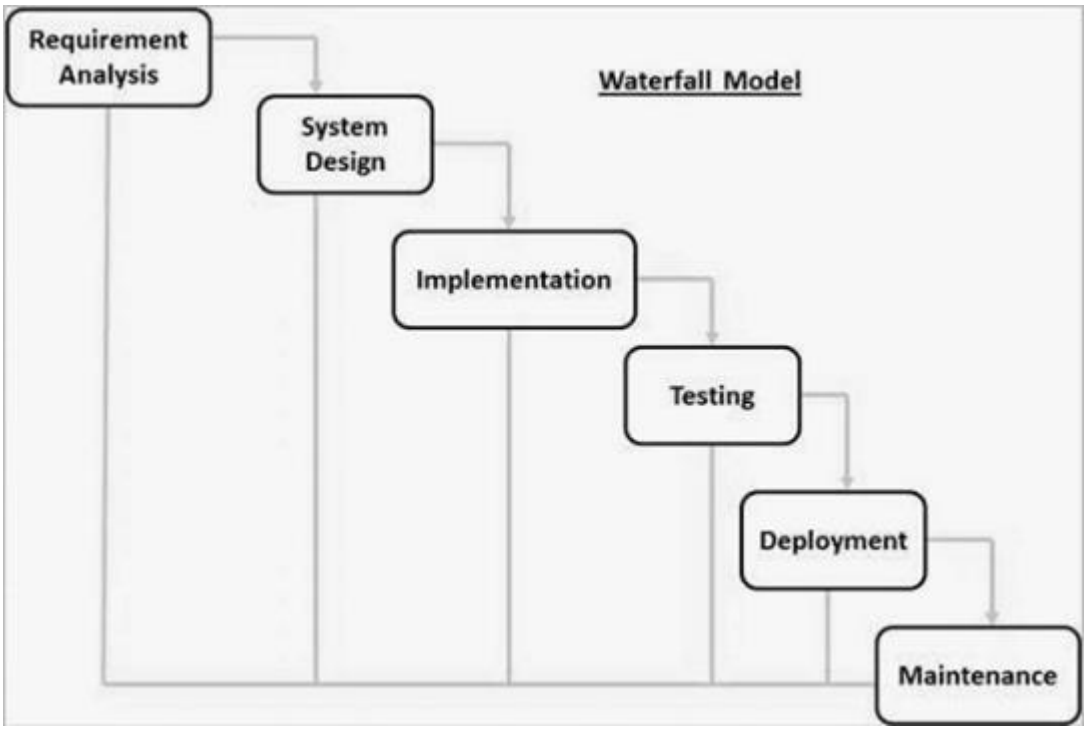


Fig-28: waterfall model

Coding

3.1 Models

```
from django.db import models
from django.utils import timezone

event_choices = (
    ('Incorporation', 'Incorporation'),
    ('Subscription', 'Subscription'),
)

class Company(models.Model):
    company_id = models.BigAutoField(primary_key=True)
    name = models.CharField(max_length=255)
    address = models.TextField(blank=True)
    user_log = models.CharField(max_length=255, null=True)
    currency = models.CharField(max_length=255, null=True)
```

```
class ShareHolder(models.Model):
    folio_no = models.BigAutoField(primary_key=True)
    name = models.CharField(max_length=255)
    address = models.TextField(blank=True)
    legal_form = models.TextField(blank=True)
    company = models.ForeignKey(Company, on_delete=models.CASCADE,
to_field="company_id", null=True)

    def __str__(self):
        return self.name

class Shares(models.Model):
    share_id = models.BigAutoField(primary_key=True)
    no_of_shares = models.DecimalField(max_digits=15, decimal_places=2)
    class_of_shares = models.TextField(max_length=255)
    folio_no = models.ForeignKey(ShareHolder, on_delete=models.DO_NOTHING,
to_field="folio_no")

class TransferDetails(models.Model):
    td_id = models.BigAutoField(primary_key=True)
    transferer = models.ForeignKey(ShareHolder, related_name="transfered_from",
on_delete=models.DO_NOTHING)
    transferee = models.ForeignKey(ShareHolder, related_name="transfered_to",
on_delete=models.DO_NOTHING)
    share_class = models.CharField(max_length=255)
    no_of_shares = models.DecimalField(max_digits=15, decimal_places=2)
    nominal_value = models.DecimalField(max_digits=15, decimal_places=2,
default=0.0)

class Transaction(models.Model):
    transaction_id = models.BigAutoField(primary_key=True)
    folio_no = models.ForeignKey(ShareHolder, on_delete=models.CASCADE,
to_field="folio_no")
    event = models.CharField(max_length=255, choices=event_choices, default='A')
    no_of_shares = models.DecimalField(max_digits=15, decimal_places=2)
    share_class = models.CharField(max_length=255, default="Ordinary Shares")
    nominal_value = models.DecimalField(max_digits=15, decimal_places=2,
default=0.01)
    consideration = models.DecimalField(max_digits=15, decimal_places=4)
    total = models.DecimalField(max_digits=15, decimal_places=2)
    status = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    event_date = models.DateTimeField()
```

```
    transfer_details = models.ForeignKey(TransferDetails, null=True,
on_delete=models.DO_NOTHING)
    user_log = models.CharField(max_length=255, null=True)
    # convert_details = models.ForeignKey(ConvertDetails, null=True,
on_delete=models.DO_NOTHING)

class ConvertDetails(models.Model):
    conversion_id = models.BigAutoField(primary_key=True)
    convert_from = models.CharField(max_length=255)
    convert_to = models.CharField(max_length=255)
    no_of_shares = models.DecimalField(max_digits=15, decimal_places=2)
    transaction_id = models.ForeignKey(Transaction, null=True,
on_delete=models.CASCADE, related_name="convert")
```

3.2 Utils.py

```
import openpyxl
from openpyxl.styles import Alignment
from openpyxl.styles import Border ,Side ,Font
from datetime import timezone

def gen_row_incorporation(data, total_shares):
    # d = []
    # d.append(data.created_at.strftime('%d-%m-%Y'))
    # d.append(data.event)
    # d.append(data.no_of_shares)
    # d.append(total)
    # d.append('signature')
    if data.share_class in total_shares:
        total_shares[data.share_class] += float(data.no_of_shares)
    else:
        total_shares[data.share_class] = float(data.no_of_shares)
    d = []
    d.append(data.event_date.astimezone(timezone.utc).strftime('%B %d, %Y'))
    d.append(f"Incorporation of the company")
    d.append(f"Subscription of {str(int(data.no_of_shares))} shares/{data.share_class}
shares with a nominal value of EUR {str(data.nominal_value)} each")
    # d.append(str(abs(int(data.no_of_shares))) + " shares")
    tot_str = ""
    for key in total_shares:
        tot_str += f"\n{int(total_shares[key])} class {key} shares"
    d.append(tot_str)
    d.append('')
    return d, total_shares
```

```
def gen_row_subscription(data, total_shares):
    if data.share_class in total_shares:
        total_shares[data.share_class] += float(data.no_of_shares)
    else:
        total_shares[data.share_class] = float(data.no_of_shares)
    d = []
    d.append(data.event_date.astimezone(timezone.utc).strftime('%B %d, %Y'))
    d.append(f"Increase of the share capital of the Company")
    d.append(f"Subscription of {str(int(data.no_of_shares))} shares/{data.share_class}
shares with a nominal value of EUR {str(data.nominal_value)} each")
    tot_str = ""
    for key in total_shares:
        tot_str += f"\n{int(total_shares[key])} class {key} shares"
    d.append(tot_str)
    d.append('')
    return d, total_shares

def gen_row_transfer(data, total_shares):
    if data.folio_no == data.transfer_details.transferer:
        if data.share_class in total_shares:
            total_shares[data.share_class] += float(data.no_of_shares)
        else:
            total_shares[data.share_class] = float(data.no_of_shares)
        d = []
        d.append(data.event_date.strftime('%B %d, %Y'))
        d.append(f"Transfer of shares to {data.transfer_details.transferee}")
        d.append(f"Transfer of {str(abs(int(data.no_of_shares)))} {data.share_class}
shares, with a nominal value of EUR {str(data.nominal_value)} each")
        tot_str = ""
        for key in total_shares:
            tot_str += f"\n{total_shares[key]} shares/{key} shares"
        d.append(tot_str)
        d.append('')
        return d, total_shares
    if data.share_class in total_shares:
        total_shares[data.share_class] += float(data.no_of_shares)
    else:
        total_shares[data.share_class] = float(data.no_of_shares)
    d = []
    d.append(data.event_date.astimezone(timezone.utc).strftime('%B %d, %Y'))
    d.append(f"Acquisition of shares from {data.transfer_details.transferer}")
```

```
    d.append(f"Acquisition of {str(int(data.no_of_shares))} {data.share_class} shares,
with a nominal value of EUR {str(data.nominal_value)} each")
    tot_str = ""
    for key in total_shares:
        tot_str += f"\n{int(total_shares[key])} class {key} shares"
    d.append(tot_str)
    d.append('')
    return d, total_shares

def gen_row_conversion(data, total_shares):
    exist_share = []
    new_share = []
    if data.share_class in total_shares:
        exist_share.append(data.share_class)
        total_shares[data.share_class] -= float(data.no_of_shares)
    else:
        new_share.append(data.share_class)
        total_shares[data.share_class] = float(data.no_of_shares)
    temp_str = f"Reclassification of existing {str(int(data.no_of_shares))}
{data.share_class} shares into: \n"
    for row in data.convert.all():
        temp_str += f"{str(int(row.no_of_shares))} class {str(row.convert_to)} shares\n"
        if row.convert_to in total_shares:
            exist_share.append(row.convert_to)
            total_shares[row.convert_to] += float(row.no_of_shares)
        else:
            new_share.append(row.convert_to)
            total_shares[row.convert_to] = float(row.no_of_shares)
    temp_str += "\nwith a nominal value of EUR 0.01 each"
    d = []
    d.append(data.event_date.astimezone(timezone.utc).strftime('%B %d, %Y'))
    B_str = "Extra ordinary general meeting approving:\n"
    if new_share:
        B_str += "\nCreation of new classes of shares into "
        for share in new_share:
            B_str += f"class {share} shares, "
        B_str = B_str[:-2]
    if exist_share:
        B_str += f"\nReclassification of existing {data.share_class} shares"
        # for share in exist_share:
        #     B_str += f"{share} shares\n"
    d.append(B_str)
    d.append(temp_str)
    tot_str = ""
```

```
    for key in total_shares:
        tot_str += f"\n{int(total_shares[key])} class {key} shares"
    d.append(tot_str)
    d.append('')
    return d, total_shares

def generate_header(sh, ws):
    # print(sh.folio_no)
    worksheet = ws
    border = Border(left=Side(style='thin'),
                top=Side(style='thin'),
                right=Side(style='thin'),
                bottom=Side(style='thin'))
    font = Font(bold=True)
    line1 = ["Shareholder:", sh.name]
    line2 = ["Legal form:", sh.legal_form]
    line3 = ["Address:", sh.address]
    line4 = []
    line5 = ["Date","Subscription/acquisition","Shares transferred/ converted/
subscribed","Total number of shares","Signature"]

    merged_range0 = "B1:E1"
    merged_range1 = "B2:E2"
    merged_range2 = "B3:E3"
    merged_range3 = "A4:E4"

    # Change height of row A1
    worksheet.row_dimensions[1].height = 30
    worksheet.row_dimensions[2].height = 30
    worksheet.row_dimensions[3].height = 30
    worksheet.row_dimensions[4].height = 15
    worksheet.row_dimensions[5].height = 35
    # Change width of column B
    worksheet.column_dimensions["A"].width = 16
    worksheet.column_dimensions["B"].width = 50
    worksheet.column_dimensions["C"].width = 30
    worksheet.column_dimensions["D"].width = 25
    worksheet.column_dimensions["E"].width = 25

    # Write data to cells
    worksheet.append(line1)
    worksheet.append(line2)
    worksheet.append(line3)
    worksheet.append(line4)
```

```
    worksheet.append(line5)

    # Merge cells and set alignment
    worksheet.merge_cells(merged_range0)
    worksheet.merge_cells(merged_range1)
    worksheet.merge_cells(merged_range2)
    worksheet.merge_cells(merged_range3)
    worksheet['A1'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['A2'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['A3'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['B1'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['B2'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['B3'].alignment = Alignment(horizontal ='left', vertical='center')
    worksheet['C5'].alignment = Alignment(horizontal ='center',
vertical='center',wrap_text=True)
    worksheet['D5'].alignment = Alignment(horizontal ='center',
vertical='center',wrap_text=True)
    worksheet['E5'].alignment = Alignment(horizontal ='center', vertical='center')
    worksheet['A5'].alignment = Alignment(horizontal ='center', vertical='center')
    worksheet['B5'].alignment = Alignment(horizontal ='center', vertical='center')

    #font
    worksheet['A1'].font = font
    worksheet['A2'].font = font
    worksheet['A3'].font = font
    worksheet['A5'].font = font
    worksheet['B5'].font = font
    worksheet['B1'].font = font
    worksheet['C5'].font = font
    worksheet['D5'].font = font
    worksheet['E5'].font = font
    #border

    worksheet['A5'].border = border
    worksheet['B5'].border = border
    worksheet['C5'].border = border
    worksheet['D5'].border = border
    worksheet['E5'].border = border
    worksheet['A1'].border = border
    worksheet['B1'].border = border
    worksheet['A2'].border = border
    worksheet['B2'].border = border
    worksheet['A3'].border = border
    worksheet['B3'].border = border
```

```
    worksheet['c1'].border = border
    worksheet['c2'].border = border
    worksheet['c3'].border = border
    worksheet['D1'].border = border
    worksheet['D2'].border = border
    worksheet['D3'].border = border
    worksheet['E1'].border = border
    worksheet['E2'].border = border
    worksheet['E3'].border = border

def generate_frontpage(ws):
    worksheet = ws
    data = [
        ['company1',' ',' '],
        ['Electronicity, Bengaluru - 560100',' ',' '],
        ['Registered office: Electronicity, Bengaluru - 560100 ',' ',' '],
        ['R.C.S. 1 : B 11123',' ',' '],
        ['Share capital:EUR 12345',' ',' '],
        ['Amendments to the articles of association of the Company', ' ', ' ', ],
        ['Date: ', ' ', 'Date: ', ],
        ['Notary:', ' ', 'Notary: ', ],
        ['Place:', ' ', 'Place: ', ],
        [' ', ' ', ' ', ],
        ['Date: ', ' ', 'Date: ', ],
        ['Notary:', ' ', 'Notary: ', ],
        ['Place:', ' ', 'Place: ', ],
        [' ', ' ', ' ', ],['Date: ', ' ', 'Date: ', ],
        ['Notary:', ' ', 'Notary: ', ],
        ['Place:', ' ', 'Place: ', ],
        [' ', ' ', ' ', ],['Date: ', ' ', 'Date: ', ],
        ['Notary:', ' ', 'Notary: ', ],
        ['Place:', ' ', 'Place: ', ],
        [' ', ' ', ' ', ],
    ]

    for d in data:
        rwstart = int(worksheet.max_row)+1

        for i in range(1,len(d)+1):
            if rwstart <= 6:  # Check if in the first 6 rows (merged)
                alignment = Alignment(horizontal='center', vertical='center',
wrap_text=True)
            #elif rwstart == 7:
```

```
            #   worksheet.cell(row=rwstart, column=i).border =
Border(left=Side(style='thin'),top=Side(style='thin'),right=Side(style='thin'),bottom=Sid
e(style='thin'))
                #worksheet.cell(row=7, column=1).border =
Border(left=Side(style='thin'),top=Side(style='thin'),right=Side(style='thin'),bottom=Sid
e(style='thin'))
            else:
                alignment = Alignment(horizontal='left', vertical='center')
            worksheet.cell(row=rwstart, column=i).value = d[i-1]
            worksheet.cell(row=rwstart, column=i).alignment = alignment
        rwstart += 1

            # ws.cell(row=rwstart, column=i).alignment = Alignment(vertical='middle')
    worksheet.column_dimensions["A"].width = 70
    worksheet.column_dimensions["C"].width = 20

    for r in range (1 ,7):
        merged_range = f"A{r}:C{r}"
        worksheet.merge_cells(merged_range)

        for n in range (1,23):
            if n == 7:
                worksheet[f"C{n}"].border = Border()
            else:
                worksheet[f"C{n}"].border = Border(right=Side(style='thin'))

        for n in range (1,23):
            worksheet[f"A{n}"].border = Border(left=Side(style='thin'))

        worksheet['A7'].border =
Border(left=Side(style='thin'),top=Side(style='thin'),right=Side(style='thin'),bottom=Sid
e(style='thin'))
        worksheet['B7'].border =
Border(left=Side(style='thin'),top=Side(style='thin'),right=Side(style='thin'),bottom=Sid
e(style='thin'))
        worksheet['C7'].border =
Border(left=Side(style='thin'),top=Side(style='thin'),right=Side(style='thin'),bottom=Sid
e(style='thin'))
        worksheet['A1'].border = Border(top=Side(style='thin'),left=Side(style='thin'))
        worksheet['B1'].border = Border(top=Side(style='thin'))
        worksheet['C1'].border = Border(top=Side(style='thin'),right=Side(style='thin'))
        worksheet['A23'].border = Border( bottom=Side(style='thin'),
left=Side(style='thin'))
        worksheet['B23'].border = Border( bottom=Side(style='thin'))
```

```
        worksheet['C23'].border = Border( bottom=Side(style='thin'),
right=Side(style='thin'))

        mrange = "A7:C7"
        worksheet.merge_cells(mrange)

def generate_row(data, ws, total):
    rwstart = int(ws.max_row) + 1
    print('Start',rwstart)
    rwend = rwstart + 10

    if data.event == 'Incorporation':
        d, total = gen_row_incorporation(data, total)
    elif data.event == 'Subscription':
        d, total = gen_row_subscription(data, total)
    elif data.event == 'Transfer':
        d, total = gen_row_transfer(data, total)
    elif data.event == 'Conversion':
        d, total = gen_row_conversion(data, total)
    else:
        d = []
        d.append(data.created_at.strftime('%d-%m-%Y'))
        d.append(data.event)
        d.append(data.no_of_shares)
        d.append(total)
        d.append('signature')
    for i in range(1, int(ws.max_column)+1):
        # print(i)
        ws.merge_cells(start_row=rwstart, start_column=i, end_row=rwend,
end_column=i)
        ws.cell(row=rwstart, column=i).value = d[i-1]
        ws.cell(row=rwstart, column=i).alignment =
Alignment(horizontal='center',vertical='center', wrapText=True)
        ws.cell(row=rwend, column=i).border = Border(bottom=Side(style='thin'))
        for i in range(1, int(ws.max_column)+1):
            for j in range(rwstart, rwend+1):
                ws.cell(row=j, column=i).border = Border(right=Side(style='thin'),
left=Side(style='thin'), bottom=Side(style='thin'))
    return total
```

3.3 Settings.py

Path /app

```
from pathlib import Path
import os
import pyodbc
from pathlib import Path

from django.core.management.commands.runserver import Command as runserver

runserver.default_port = 9000

from django import VERSION

from ms_identity_web.configuration import AADConfig
from ms_identity_web import IdentityWebPython

AAD_CONFIG = AADConfig.parse_json(file_path='/aadconfig.json')
MS_IDENTITY_WEB = IdentityWebPython(AAD_CONFIG)
ERROR_TEMPLATE = 'auth/{}.html' # for rendering 401 or other errors from
sal_middleware

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-0t=)hc^a&+0!(j7f8!(!9&&z0y%kzmb&(=)-
(@xxu!%w%o1x+='

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['*']

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_extensions',
    'core',
    'transaction',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
MIDDLEWARE.append('ms_identity_web.django.middleware.MsalMiddleware')

ROOT_URLCONF = 'app.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'app.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

# DATABASES = {
#     'default': {
```

```
#         "ENGINE": "mssql",
#         "NAME": "django-rd",
#         "USER": "mdmadmin",
#         "PASSWORD": "TR29=w\)0WH5",
#         "HOST": "tcp:db-we-sql-django-rd.database.windows.net",
#         "PORT": "1433",
#         "OPTIONS": {"driver": "ODBC Driver 18 for SQL Server",
#             'extra_params': 'MARS_Connection=Yes',
#         },
#     }
# }

# DATABASES = {
#     'default': {
#         "ENGINE": "mssql",
#         "NAME": "admin_sql",
#         "USER": "admin_sql",
#         "PASSWORD": "Azureuser@6936",
#         "HOST": "ms-temp.database.windows.net",
#         "PORT": "1433",
#         "OPTIONS": {"driver": "ODBC Driver 18 for SQL Server",
#             'extra_params': 'MARS_Connection=Yes',
#         },
#     }
# }

DATABASES = {
    'default': {
        "ENGINE": "mssql",
        "NAME": "django-rd",
        "USER": "mdmadmin",
        "PASSWORD": "TR29=w\)0WH5",
        "HOST": "tcp:db-we-sql-django-rd.database.windows.net",
        "PORT": "1433",
        "OPTIONS": {"driver": "ODBC Driver 18 for SQL Server",
            'extra_params': 'MARS_Connection=Yes',
        },
    }
}

#Driver={ODBC Driver 18 for SQL Server};Server=tcp:db-we-sql-django-
rd.database.windows.net,1433;Database=django-
rd;Uid=mdmadmin;Pwd={your_password_here};Encrypt=yes;TrustServerCertificate=
no;Connection Timeout=30;
```

```
# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE =  'Asia/Kolkata'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

3.4 Admin.py

Path /core

```
from django.contrib import admin

from core.models import (
    Company,
    ShareHolder,
    Shares,
    TransferDetails,
    Transaction,
    ConvertDetails
)

class CompanyAdmin(admin.ModelAdmin):
    ordering = ['company_id']
    list_display = ['company_id', 'name', 'address']
    readonly_fields = ['company_id']
    add_fields = (
        (None, {
            'classes': ('wide',),
            'fields': (
                'name',
                'address'
            )
        })
    )

class ShareHolderAdmin(admin.ModelAdmin):
    ordering = ['folio_no']
    list_display = ['folio_no', 'name', 'address', 'legal_form', 'company']
    readonly_fields = ['folio_no']
    add_fields = (
        (None, {
            'classes': ('wide', ),
            'fields': (
                'name',
                'address',
                'legal_form',
                'company'
            )
        })
    )
```

```
class SharesAdmin(admin.ModelAdmin):
    ordering = ['share_id']
    list_display = ['share_id', 'no_of_shares', 'class_of_shares', 'folio_no']
    readonly_fields = ['share_id']
    add_fields = (
        (None, {
            'classes': ('wide', ),
            'fields': (
                'no_of_shares',
                'class_of_shares',
                'folio_no'
            )
        })
    )

class TransferDetailsAdmin(admin.ModelAdmin):
    ordering = ['td_id']
    list_display = ['td_id', 'transferer', 'transferee', 'share_class', 'no_of_shares',
'nominal_value']
    readonly_fields = ['td_id']
    add_fields = (
        (None, {
            'classes': ('wide', ),
            'fields': (
                'transferer',
                'transferee',
                'share_class',
                'no_of_shares',
                'nominal_value'
            )
        })
    )

class ConvertDetailsAdmin(admin.ModelAdmin):
    ordering = ['conversion_id']
    list_display = ['conversion_id', 'convert_from', 'convert_to', 'no_of_shares',
'transaction_id']
    readonly_fields = ['conversion_id']
    add_fields = (
        (None, {
            'classes': ('wide', ),
            'fields': (
                'convert_from',
```

```
            'convert_to',
            'no_of_shares',
            'transaction_id'
        )
    })
)

class TransactionAdmin(admin.ModelAdmin):
    ordering = ['transaction_id']
    list_display = ['transaction_id', 'folio_no', 'event', 'no_of_shares', 'share_class',
'nominal_value', 'consideration', 'total', 'status', 'created_at', 'event_date',
'transfer_details']
    readonly_fields = ['transaction_id']
    add_fields = (
        (None, {
            'classes': ('wide', ),
            'fields': (
                'folio_no',
                'event',
                'no_of_shares',
                'share_class',
                'nominal_value',
                'consideration',
                'total',
                'status',
                'created_at',
                'event_date',
                'transfer_details'
            )
        })
    )

admin.site.register(Company, CompanyAdmin)
admin.site.register(ShareHolder, ShareHolderAdmin)
admin.site.register(Shares, SharesAdmin)
admin.site.register(TransferDetails, TransferDetailsAdmin)
admin.site.register(Transaction, TransactionAdmin)
admin.site.register(ConvertDetails, ConvertDetailsAdmin)
```

3.5 Forms.py

```
from django import  forms

from core.models import (
    ShareHolder,
    Transaction,
    Company
)

share_classes = [
                ('Ordinary', 'Ordinary Shares'),
                ('A', 'A'),
                ('B', 'B'),
                ('C', 'C'),('A1', 'A1'),('B1', 'B1'),('C1', 'C1')
              ]

Currency = [
            ('₹','₹ Rs'),('€','€ EUR'),('$','$ USD') ,('£','£ GBP'),('₣','₣ CHF')
]

def get_folio():
    return [(f.folio_no, str(f)) for f in ShareHolder.objects.all()]

def get_comp():
    return [(f.name, str(f)) for f in Company.objects.all()]

class AddShareholderForm(forms.ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
        self.fields['name'].widget.attrs.update({'required': '', 'name': 'sh_name', 'id':
'sh_name', 'type': 'text'})
        self.fields['address'].widget.attrs.update({'required': '', 'name': 'sh_address', 'id':
'sh_address', 'type': 'textarea', 'rows': '4'})
        self.fields['legal_form'].widget.attrs.update({'required': '', 'name': 'sh_lf', 'id':
'sh_lf', 'type': 'textarea', 'rows': '4'})

    class Meta:
        model = ShareHolder
        fields = ['name', 'address', 'legal_form']

class AddcompanyForm(forms.Form):
```

```
    name = forms.CharField(max_length=255)
    address = forms.CharField(max_length=10000)
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
        #self.fields['name'] = forms.ChoiceField(choices=get_comp)
        self.fields['name'].widget.attrs.update({'required': '', 'name': 'cp_name', 'id':
'cp_name', 'type': 'text'})
        self.fields['address'].widget.attrs.update({'required': '', 'name': 'cp_address', 'id':
'cp_address', 'type': 'textarea', 'rows': '4'})
        self.fields['currency'] = forms.ChoiceField(choices=Currency)
        self.fields['currency'].widget.attrs.update({'required': '', 'name': 'currency', 'id':
'currency', 'class': 'form-select'})

    class Meta:
        #model = Company
        fields = ['name', 'address']

class AddTransactionForm(forms.ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
        self.fields['folio_no'] = forms.ChoiceField(choices=get_folio)
        self.fields['folio_no'].widget.attrs.update({'required': '', 'name': 'folio_no', 'id':
'folio_no', 'class': 'form-select'})
        self.fields['event'].widget.attrs.update({'required': '', 'name': 'event', 'id': 'event',
'class': 'form-select'})
        self.fields['no_of_shares'].widget.attrs.update({'required': '', 'name':
'no_of_shares', 'id': 'no_of_shares', 'type': 'number', 'step': 'any'})
        self.fields['nominal_value'].widget.attrs.update({'required': '', 'name':
'nominal_value', 'id': 'nominal_value', 'type': 'number', 'step': 'any'})
        self.fields['consideration'].widget.attrs.update({'required': '', 'name':
'consideration', 'id': 'consideration', 'type': 'number', 'step': 'any'})
        self.fields['total'].widget.attrs.update({'required': '', 'name': 'total', 'id': 'total', 'type':
'number', 'step': 'any'})
        self.fields['status'].widget.attrs.update({'type': 'hidden', 'name': 'status', 'id':
'status', 'value': 'Active'})
        self.fields['event_date'] = forms.DateField(widget=forms.TextInput(attrs={'name':
'event_date', 'id': 'event_date','class': 'form-control', 'type':'date'}))
        # self.fields['event_date'].widget.attrs.update({'type': 'date', 'name': 'event_date',
'id': 'event_date'})
        self.fields['class_of_share'] = forms.ChoiceField(choices=share_classes)
```

```
        self.fields['class_of_share'].widget.attrs.update({'required': '', 'name':
'class_of_share', 'id': 'class_of_share', 'class': 'form-select'})

    class Meta:
        model = Transaction
        fields = ['folio_no', 'event', 'no_of_shares', 'nominal_value', 'consideration', 'total',
'status']
        extra_fields = ['class_of_share', 'event_date']

class TransferShareForm(forms.Form):
    event = forms.CharField(max_length=255)
    no_of_shares = forms.DecimalField(max_digits=15, decimal_places=2)
    nominal_value = forms.DecimalField(max_digits=15, decimal_places=2,
initial=0.01)
    consideration = forms.DecimalField(max_digits=15, decimal_places=2)
    event_date = forms.DateField(widget=forms.TextInput(attrs={'class': 'form-control',
'type':'date'}))
    transferer = forms.ChoiceField(choices=get_folio)
    transferee = forms.ChoiceField(choices=get_folio)
    share_class = forms.ChoiceField(choices=share_classes)
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
            # self.fields['event_date'].widget.attrs.update({'type': 'Date'})

class ConversionShareForm(forms.Form):
    folio_no = forms.ChoiceField(choices=get_folio)
    event_date = forms.DateField(widget=forms.TextInput(attrs={'class': 'form-control',
'type':'date'}))
    share_class = forms.ChoiceField(choices=share_classes)
    convert_class = forms.ChoiceField(choices=share_classes)
    convert_quantity = forms.DecimalField(max_digits=15, decimal_places=2)
    consideration = forms.DecimalField(max_digits=15, decimal_places=2)
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'form-control'
```

3.6 Urls.py

```
from django.urls import path, include
from ms_identity_web.django.msal_views_and_urls import MsalViews
from django.conf import settings



from transaction.views import *
msal_urls = MsalViews(settings.MS_IDENTITY_WEB).url_patterns()

urlpatterns = [
    # path('', home, name = "Home"),
    path('', index, name = "index"),
    path('transaction/add', addtransaction, name = "Add Share"),
    path('transaction/transfer', transfer, name = "Add Transfer"),
    path('transaction/convert', convert, name = "Add Conversion"),
    path('addsh', addshareholder, name = "Add Shareholder"),
    path('addtr', addtransaction, name = "Add Transaction"),
    path('transaction', transaction, name="Transaction"),
    path('generate', generate, name = "Generate Report"),
    path('company', company, name = "company"),
    path('addcomp', addcompany , name = "Add Company")
    # path(f'{settings.AAD_CONFIG.django.auth_endpoints.prefix}/',
include(msal_urls))
]
```

3.7 Views.py

Path /transaction

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.shortcuts import get_object_or_404
from django.http import HttpResponse , StreamingHttpResponse
from wsgiref.util import FileWrapper
# from ms_identity_web import MSALApp
import requests
import mimetypes
import os
from django.conf import settings
from core.models import (
    Transaction,
    ShareHolder,
```

```
    Shares,
    TransferDetails,
    ConvertDetails,
    Company
)

ms_identity_web = settings.MS_IDENTITY_WEB
from core.utils import (
    generate_header,
    generate_row,
    generate_frontpage
)

from transaction.forms import (
    AddShareholderForm,
    AddTransactionForm,
    TransferShareForm,
    ConversionShareForm,
    AddcompanyForm
)

import datetime
import openpyxl

def get_graph_token():
    url = settings.AD_URL
    headers = {'content-Type': 'application/x-www-form-
urlencoded','Access':'application/json'}
    data = {
        'grant_type'
    }
def index(request):

    data = Transaction.objects.all()
    args = {}
    args['data'] = data
    return render(request, 'transactionview.html', args)

@ms_identity_web.login_required

def update_shares(folio_no, share_class, no_of_shares):
    try:
        print('A', share_class)
```

```
        existing_obj = get_object_or_404(Shares, folio_no = folio_no,
class_of_shares=share_class)
        print(existing_obj)
        existing_obj.no_of_shares = float(existing_obj.no_of_shares) +
float(no_of_shares)
        existing_obj.save()
        return existing_obj
    except Exception as e:
        print(share_class)
        share_obj = {}
        share_obj['no_of_shares'] = no_of_shares
        share_obj['class_of_shares'] = share_class
        share_obj['folio_no'] = folio_no
        print(share_obj)
        new_obj = Shares.objects.create(**share_obj)
        new_obj.save()
        return new_obj

# Create your views here.
@ms_identity_web.login_required
def home(request):
    sh_form = AddShareholderForm()
    tr_form = AddTransactionForm()
    cp_form = AddcompanyForm()

    args = {}
    args['sh_form'] = sh_form
    args['tr_form'] = tr_form
    args['cp_form'] = cp_form
    print(request.identity_context_data)
    return render(request, 'addshares.html', args)
    # return render(request, 'index.html')

@ms_identity_web.login_required
def addshareholder(request):
    form = AddShareholderForm(request.POST)
    if ShareHolder.objects.filter(name = request.POST['name']).all():
        messages.error(request, 'Shareholder name already exists')
        return redirect('/transaction/add')
    form = AddShareholderForm(request.POST)
    if form.is_valid():
        form.save()
        messages.success(request, 'Shareholder added successfully')
        return redirect('/transaction/add')
```

```
    for field, errors in form.errors.items():
        for error in errors:
            messages.error(request, error)
    return redirect('/')


@ms_identity_web.login_required

def company(request):
    cp_form = AddcompanyForm()
    args = {}
    args['cp_form'] = cp_form
    companies = Company.objects.all()
    args['companies'] = companies
    return render(request, 'company.html',args)

@ms_identity_web.login_required
def addcompany(request):
    if request.POST:
        if Company.objects.filter(name = request.POST['name']).all():
           messages.error(request, 'company name already exists')
           return redirect('/company')
        form_data = {
        'name': request.POST['name'],
        'address': request.POST['address'],
        'user_log' : request.POST['hid_name'],
        'currency' : request.POST['currency']
        #'event_date': datetime.datetime.strptime(request.POST['event_date'], "%Y-
%m-%d"),
        }
        print("Here", form_data)
        print(request.POST)
        form = Company.objects.create(**form_data)
        if form:
                form.save()
                messages.success(request, 'company added successfully')
                return redirect('/company')
        messages.error(request, 'Error adding transaction')
        return redirect('/')
    for field, errors in form.errors.items():
            print(form.errors)
            for error in errors:
                messages.error(request, error)
                print(error)
```

```
    return redirect('/company')

@ms_identity_web.login_required
def addtransaction(request):
    # print("Here ",(request.identity_context_data.__dict__))
    headers = {
        "access_token": request.identity_context_data._access_token
    }
    # response =
requests.get(f"https://graph.microsoft.com/v1.0/groups/{request.identity_context_dat
a._id_token_claims['groups'][0]}", headers=headers)
    # print(response.json())
    # print("Here ",(request.identity_context_data._id_token_claims['roles']))

    if request.POST:
        if Transaction.objects.filter(folio_no = request.POST['folio_no']).all() and
request.POST['event'] == "Incorporation":
            messages.error(request, 'Shareholder already Incorporated')
            return redirect('/transaction/add')
        if (not Transaction.objects.filter(folio_no = request.POST['folio_no']).all()) and
request.POST['event'] == "Subscription":
            messages.error(request, 'Shareholder First entry should be Incorporation')
            return redirect('/transaction/add')
        form_data = {
            'folio_no': ShareHolder.objects.get(folio_no = request.POST['folio_no']),
            'event': request.POST['event'],
            'no_of_shares': request.POST['no_of_shares'],
            'share_class': request.POST['class_of_share'],
            'nominal_value': request.POST['nominal_value'],
            'consideration': request.POST['consideration'],
            'total': 0,
            'status': 'Active',
            'created_at': datetime.datetime.now(),
            'event_date': datetime.datetime.strptime(request.POST['event_date'], "%Y-
%m-%d"),
            'user_log' : request.POST['hid_name']
            # 'class_of_share': request.POST['class_of_share']
            # 'transfer_details': Null
        }
        print("Here", form_data)
        # form = AddTransactionForm(form_data)
        print(request.POST)
        if form_data['event'] in ['Subscription', 'Incorporation']:
            form = Transaction.objects.create(**form_data)
```

```
            if form:
                update_shares(form_data['folio_no'], request.POST['class_of_share'],
form_data['no_of_shares'])
                form.save()
                messages.success(request, 'Transaction added successfully')
                return redirect('/transaction/add')
            messages.error(request, 'Error adding transaction')
            return redirect('/transaction/add')
        for field, errors in form.errors.items():
            print(form.errors)
            for error in errors:
                messages.error(request, error)
                print(error)
        return redirect('/transaction/add')
    sh_form = AddShareholderForm()
    tr_form = AddTransactionForm()
    args = {}
    args['sh_form'] = sh_form
    args['tr_form'] = tr_form
    return render(request, 'addshares.html', args)

@ms_identity_web.login_required
def transfer(request):
    if request.POST:
        form_data = request.POST.copy()
        print(request.POST)
        form_data.update({
            'event': 'Transfer'
        })
        form = TransferShareForm(form_data)
        print(form)
        transferer = ShareHolder.objects.get(folio_no = request.POST['transferer'])
        transferee = ShareHolder.objects.get(folio_no = request.POST['transferee'])
        if not transferer == transferee:
            if form.is_valid():
                transferer = ShareHolder.objects.get(folio_no = request.POST['transferer'])
                transferee = ShareHolder.objects.get(folio_no =
request.POST['transferee'])
                print(form.cleaned_data)
                try:
                    available_shares = Shares.objects.get(folio_no = transferer,
class_of_shares = form.cleaned_data['share_class'])
                    if available_shares.no_of_shares >= form.cleaned_data['no_of_shares']:
                        transfer_data = {
```

```
                    'transferer': transferer,
                    'transferee': transferee,
                    'share_class': form.cleaned_data['share_class'],
                    'no_of_shares': form.cleaned_data['no_of_shares'],
                    'nominal_value': 0
                }
                print(transfer_data)
                transfer_details = TransferDetails.objects.create(**transfer_data)
                if transfer_details:
                    transferer_data = {
                        'folio_no': transferer,
                        'event': form_data['event'],
                        'no_of_shares': -abs(float(form_data['no_of_shares'])),
                        'share_class': form.cleaned_data['share_class'],
                        'nominal_value': 0,
                        'consideration': form_data['consideration'],
                        'total': 0,
                        'status': 'Active',
                        'created_at': '',
                        'event_date': form_data['event_date'],
                        'user_log' : request.POST['hid_name'],
                        'transfer_details': transfer_details
                    }
                    Transaction.objects.create(**transferer_data)
                    update_shares(transferer, transfer_data['share_class'],
transferer_data['no_of_shares'])
                    transferee_data = {
                        'folio_no': transferee,
                        'event': form_data['event'],
                        'no_of_shares': form_data['no_of_shares'],
                        'share_class': form.cleaned_data['share_class'],
                        'nominal_value': 0,
                        'consideration': form_data['consideration'],
                        'total': 0,
                        'status': 'Active',
                        'created_at': '',
                        'event_date': form_data['event_date'],
                        'user_log' : request.POST['hid_name'],
                        'transfer_details': transfer_details
                    }
                    Transaction.objects.create(**transferee_data)
                    update_shares(transferee, transfer_data['share_class'],
transferee_data['no_of_shares'])
                    messages.success(request, 'Transfer successfull.')
```

```
                    return redirect('/transaction/transfer')
                messages.error(request, 'Error Transferring shares.')
                return redirect('/transaction/transfer')

            else:
                messages.error(request, 'Error Transferring shares. Insufficient
Balance.')
                return redirect('/transaction/transfer')
        except Exception as e:
            messages.error(request, 'Error Transferring shares. Please check
balance.')
            return redirect('/transaction/transfer')
    else :
        messages.error(request, 'Cannot transfer funds to same shareholder')
        return redirect('/transaction/transfer')

    for field, errors in form.errors.items():
        print(form.errors)
        for error in errors:
            messages.error(request, error)
            print(error)
    return redirect('/transaction/transfer')

  sh_form = AddShareholderForm()
  tr_form = TransferShareForm()
  args = {}
  args['sh_form'] = sh_form
  args['tr_form'] = tr_form
  return render(request, 'transfershares.html', args)

@ms_identity_web.login_required
def convert(request):
  if request.POST:
    print(request.POST)
    shareholder = ShareHolder.objects.get(folio_no = request.POST['folio_no'])
    try:
      share_qty = Shares.objects.get(folio_no = shareholder, class_of_shares =
request.POST['share_class'])
      if share_qty.no_of_shares >= sum(int(a) for a in
request.POST.getlist('convert_quantity')):
        transaction_data = {
          'folio_no': shareholder,
          'event': 'Conversion',
```

```
                'no_of_shares': sum(int(a) for a in
request.POST.getlist('convert_quantity')),
                'share_class': request.POST['share_class'],
                'nominal_value': 0.01,
                'consideration': request.POST['consideration'],
                'total': 0,
                'status': 'Active',
                'created_at': datetime.datetime.now(),
                'user_log' : request.POST['hid_name'],
                'event_date': datetime.datetime.now()
            }
            transaction = Transaction.objects.create(**transaction_data)
            if transaction:
                # print(request.POST.getlist('convert_class'))
                for i, row in enumerate(request.POST.getlist('convert_class')):
                    # print(i, row)
                    convert_data = {
                        'convert_from': request.POST['share_class'],
                        'convert_to': request.POST.getlist('convert_class')[i],
                        'no_of_shares': request.POST.getlist('convert_quantity')[i],
                        'transaction_id': transaction
                    }
                    update_one = update_shares(shareholder,
request.POST['share_class'], -abs(float(request.POST.getlist('convert_quantity')[i])))
                    update_two = update_shares(shareholder,
request.POST.getlist('convert_class')[i],
abs(float(request.POST.getlist('convert_quantity')[i])))
                    if update_one and update_two:
                        ConvertDetails.objects.create(**convert_data)
                messages.success(request, 'Conversion Successfull')
                return redirect('/transaction/convert')
            messages.error(request, 'Error while converting shares')
            return redirect('/transaction/convert')
        messages.error(request, 'Insufficient Funds')
    except Exception as e:
        print(e)
        messages.error(request, 'Error occured : No shares Converted')
sh_form = AddShareholderForm()
cs_form = ConversionShareForm()
args = {}
args['sh_form'] = sh_form
args['cs_form'] = cs_form
return render(request, 'convertshares.html', args)
```

```
@ms_identity_web.login_required
def transaction(request):
    data = Transaction.objects.all()
    args = {}
    args['data'] = data
    return render(request, 'transactionview.html', args)

@ms_identity_web.login_required
def generate(request):
    sh_data = ShareHolder.objects.all()
    wb = openpyxl.Workbook()
    for data in sh_data:
        ws = wb.create_sheet()
        ws.title = str(data.folio_no) + "("+ data.name +")"
        generate_header(data, ws)
        tr_data = Transaction.objects.filter(folio_no = data.folio_no)
        curr_total = {}
        for row in tr_data:
            # curr_total += row.no_of_shares
            curr_total = generate_row(row, ws, curr_total)
    wb['Sheet'].title='company1'
    generate_frontpage(wb['company1'])

    filename = 'company1.l.xlsx'
    wb.save(filename)
    wb.close()
    base_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

    filepath = base_dir + '/' + filename
    thefile = filepath
    filename =  os.path.basename(thefile)
    chunk_size = 8192
    response =
StreamingHttpResponse(FileWrapper(open(thefile,'rb'),chunk_size),content_type=mi
metypes.guess_type(thefile)[0])
    response['Content-Length'] = os.path.getsize(thefile)
    response['Content-Disposition'] = "Attachment;filename=%s" % filename
    print("file downloaded")
    return response
```

3.8 aadconfig.json

```
{
    "type": {
    "client_type": "CONFIDENTIAL",
    "authority_type": "SINGLE_TENANT",
    "framework": "DJANGO"
    },
    "client": {
    "client_id": "6c33bad1-7123-4c54-a7d6-d54f72dc89f1",
    "client_credential": "eTw8Q~Q3zaTq.~Ci34Zn_7qu2g~DnrBQT0gjkdrG",
    "authority": "https://login.microsoftonline.com/bc4d7125-f283-4cf1-bfee-
330efffeee26"
    },
    "auth_request": {
    "redirect_uri": null,
    "scopes": [],
    "response_type": "code"
    },
    "flask": null,
    "django": {
    "id_web_configs": "MS_ID_WEB_CONFIGS",
    "auth_endpoints": {
    "prefix": "auth",
    "sign_in": "sign_in",
    "edit_profile": "edit_profile",
    "redirect": "redirect",
    "sign_out": "sign_out",
    "post_sign_out": "post_sign_out"
    }
}
}
```

3.9 HTML code

Base.html

```
<!-- {% load static %} -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
    <title>Home</title>
    <link
      href="../static/css/bootstrap.min.css"
      rel="stylesheet"

      crossorigin="anonymous"
    />
    <link
      href="../static/css/datatables.min.css"
      rel="stylesheet"

      crossorigin="anonymous"
    />
    <script src="../static/js/jquery-3.7.1.min.js"  crossorigin="anonymous"></script>

    <script src="../static/js/bootstrap.min.js"  crossorigin="anonymous"></script>
    <script src="../static/js/dataTables.min.js"  crossorigin="anonymous"></script>
    <!-- <link rel="stylesheet" href="{% static 'static/css/bootstrap.min.css' %}"> -->
    <!-- integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH
" -->
    <style>
      .body {
        min-height: 90vh;
      }
      .center-box {
        width: 40rem;
        min-height: 30rem;
      }
    </style>
  </head>
  <body class="body">
    <!-- NavBar -->
    <div class="container">
      <nav class="navbar navbar-expand-lg bg-body-tertiary">
        <div class="container-fluid">
          <a class="navbar-brand" href="/transaction/add">PMWA</a>
          <button
            class="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarNav"
            aria-controls="navbarNav"
            aria-expanded="false"
```

```
          aria-label="Toggle navigation"
        >
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
          {% if request.identity_context_data.authenticated %}
          <ul class="navbar-nav me-auto  mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page"
href="/transaction">View</a>
            </li>
            <li class="nav-item">
              <a class="nav-link active" aria-current="page"
href="/transaction/add">Add</a>
            </li>
            <li class="nav-item">
              <a class="nav-link active" aria-current="page"
href="/transaction/transfer">Transfer</a>
            </li>
            <li class="nav-item">
              <a class="nav-link active" aria-current="page"
href="/transaction/convert">Convert</a>
            </li>



            </ul>
            <ul class="navbar-nav ml-auto">
              <li class="nav-item">
                <a class="nav-link active" aria-current="page" href="/company">Switch
Company</a>
              </li>
            <li class="nav-item">
              <a class="nav-link">Username: <b>{{
request.identity_context_data.username }}</b></a>
            </li>
            <!-- {{ request.identity_context_data}}
            {{request.claims.type}}
            {{request.claims.value}} -->
            <li class="nav-item">
              <form class="d-flex">
                <!-- <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search"> -->
                <a href="{% url 'sign_out' %}"><button class="btn btn-sm btn-danger w-
100 my-auto" type="button">Logout</button></a>
```

```
            </form>

          </li>
        </ul>
          {% endif %}

      </div>
    </div>
  </nav>
</div>
<!-- NavBar Ends Here -->
 <!-- Error Messages -->
 <div class="container">
  {% if messages %} {% for message in messages %} {% if message.tags %}
  {% if message.tags == 'error' %}
  <div class="alert alert-danger alert-dismissible fade show" role="alert">
    {{ message }}<button
      type="button"
      class="btn-close"
      data-bs-dismiss="alert"
      aria-label="Close"
    ></button>
  </div>
  {% elif message.tags == 'success' %}
  <div class="alert alert-success alert-dismissible fade show" role="alert">
    {{ message }}<button
      type="button"
      class="btn-close"
      data-bs-dismiss="alert"
      aria-label="Close"
    ></button>
  </div>
  {% else %}
  <div class="alert alert-primary alert-dismissible fade show" role="alert">
    {{ message }}<button
      type="button"
      class="btn-close"
      data-bs-dismiss="alert"
      aria-label="Close"
    ></button>
  </div>
  {% endif %} {% endif %} {% endfor %} {% endif %}
</div>
 <!-- Error Messages Ends -->
```

```
    <!-- Body -->
    {% block body %}

    {% endblock body %}
    <!-- Body Ends Here -->
    <!-- <script
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
      integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
      crossorigin="anonymous"
    ></script> -->


  </body>
  {% block lowerscript %}

  {% endblock lowerscript %}
</html>
```

Addshares.html

```
{% extends 'base.html' %} {% block body %}
{% include 'Models/addshareholder.html' %}
<div class="container align-items-center justify-content-center">
  <div class="container center-box my-3 p-3 border">
    <div class="container">
      <center><h2>Add Shares</h2></center>
      <div class="d-flex flex-row-reverse mb-3">
        <div class="">
          <input
            type="button"
            class="btn btn-sm btn-primary"
            value="Add Shareholder"
            data-bs-toggle="modal"
            data-bs-target="#exampleModal"
          />
        </div>
      </div>

      <div>
        <form action="/addtr" method="post" onsubmit="myFunction()">
          {% csrf_token %}
          <div class="row mb-3">
            <div class="col-6">
```

```
            <label for="folio_no" class="form-label">Folio Number</label>
            {{ tr_form.folio_no }}
          </div>
          <div class="col-6">
            <label for="event" class="form-label">Event</label>
            {{ tr_form.event }}
          </div>
        </div>
        <input type="hidden" name="hid_name"
value="{{request.identity_context_data.username}}"/>
        <div class="row mb-3">
          <div class="col-md-6">
            <label for="class_of_share">Class of Shares</label>
            {{ tr_form.class_of_share }}
          </div>
          <div class="col-md-6">
            <label for="event_date">Event Date</label>
            {{ tr_form.event_date }}
          </div>
        </div>
        <div class="row mb-3">
          <div class="col-6">
            <label for="no_of_shares" class="form-label"
              >Number of Shares</label
            >
            {{ tr_form.no_of_shares }}
          </div>
          <div class="col-6">
            <label for="no_of_shares" class="form-label">Nominal Value</label>
            {{ tr_form.nominal_value }}
          </div>
        </div>
        <div class="row mb-3">
          <div class="col">
            <label for="consideration" class="form-label"
              >Consideration</label
            >
            {{ tr_form.consideration }}
          </div>
        </div>
        <div class="row mb-3 justify-content-center">
          <div class="col-auto">
            <input id="submit"
              type="submit"
```

```
            value="Submit"
            class="btn btn-sm btn-outline-primary"
          />
          <script>
            function myFunction() {
              // window.location.href = '/transaction/transfer'
              var element = document.getElementById("submit");
              element.innerHTML = "Submitting";
              element.classList.remove("btn-outline-primary");
              element.classList.add("btn-outline-warning");
              element.disabled = true;

              setTimeout(function () {
                var element = document.getElementById("submit");
                element.innerHTML = "Submit";
                element.classList.remove("btn-outline-warning");
                element.classList.add("btn-outline-primary");
                element.disabled = false;

              }, 30000)
            }

          </script>

          </div>
        </div>
      </form>
    </div>
  </div>
 </div>
</div>
{% endblock body %}
```

Convershares.html

```
{% extends 'base.html' %} {% block body %} {% include
'Models/addshareholder.html' %}
<div class="container align-items-center justify-content-center">
  <div class="container center-box my-3 p-3 border">
    <div class="container">
      <center><h2>Convert Shares</h2></center>
      <div class="d-flex flex-row-reverse mb-3">
        <div class="">
          <input
```

```
            type="button"
            class="btn btn-sm btn-primary"
            value="Add Shareholder"
            data-bs-toggle="modal"
            data-bs-target="#exampleModal"
          />
        </div>
      </div>
      <div>
        <form action="/transaction/convert" method="post" onsubmit="myFunction()">
          {% csrf_token %}
          <div class="row mb-3">
            <div class="col-6">
              <label for="folio_no" class="form-label">Folio Number</label>
              {{ cs_form.folio_no }}
            </div>
            <input type="hidden" name="hid_name"
value="{{request.identity_context_data.username}}"/>
            <div class="col-6">
              <label for="folio_no" class="form-label"
                >Date of transaction</label
              >
              <input type="date" class="form-control" />
            </div>
          </div>
          <input type="hidden" name="hid_name"
value="{{request.identity_context_data.username}}"/>
          <div class="row mb-3">
            <div class="col-6">
              <label for="folio_no" class="form-label">Class of Shares</label>
              {{ cs_form.share_class }}
            </div>
          </div>
          <div class="row mb-3 mb-12"></div>
          <div class="border p-3" style="border-radius: 1rem">
            <div class="col-md-2 align-content-end">
              <button
                type="button"
                class="btn btn-sm btn-outline-primary"
                id="conversion-button"
              >
                +
              </button>
            </div>
```

```
  <div class="row mb-3">
    <div class="col" id="conversion-data">
      <div class="row mb-1" id="conversion-row">
        <div class="col">
          <div class="row">
            <div class="col-md-6">
              <label for="conversion-class">Class</label>
              {{ cs_form.convert_class }}
            </div>
            <div class="col-md-6">
              <label for="conversion-quantity">Quantity</label>
              {{ cs_form.convert_quantity }}
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="row mb-3">
  <div class="col">
    <label for="consideration" class="form-label"
      >Consideration</label
    >
    {{ cs_form.consideration }}
  </div>
</div>
<div class="row mb-3 justify-content-center">
  <div class="col-auto">
    <input id="submit"
      type="submit"
      value="Submit"
      class="btn btn-sm btn-outline-primary"
    />
    <script>
      function myFunction() {
          // window.location.href = '/transaction/transfer'
          var element = document.getElementById("submit");
          element.innerHTML = "Submitting";
          element.classList.remove("btn-outline-primary");
          element.classList.add("btn-outline-warning");
          element.disabled = true;

          setTimeout(function () {
```

```
                    var element = document.getElementById("submit");
                    element.innerHTML = "Submit";
                    element.classList.remove("btn-outline-warning");
                    element.classList.add("btn-outline-primary");
                    element.disabled = false;

                }, 30000)
            }
                    </script>
          </div>
        </div>
      </form>
    </div>
  </div>
 </div>
</div>
{% endblock body %} {% block lowerscript %}
<script>
 function addConversionRow() {}
 $("#conversion-button").click(function () {
  $("#conversion-data").append($("#conversion-row").clone());
 });
</script>
{% endblock lowerscript %}
```

Transfershares.html

```
{% extends 'base.html' %} {% block body %}
{% include 'Models/addshareholder.html' %}
<div class="container align-items-center justify-content-center">
  <div class="container center-box my-3 p-3 border">
    <div class="container">
      <center><h2>Transfer Shares</h2></center>
      <div class="d-flex flex-row-reverse mb-3">
        <div class="">
          <input
            type="button"
            class="btn btn-sm btn-primary"
            value="Add Shareholder"
            data-bs-toggle="modal"
            data-bs-target="#exampleModal"
          />
        </div>
      </div>
```

```
        <div>
          <form action="/transaction/transfer" method="post" onsubmit="myFunction()">
            {% csrf_token %}
            <div class="row mb-3">
              <div class="col-6">
                <label for="folio_no" class="form-label">Transferer</label>
                {{ tr_form.transferer }}
              </div>
              <input type="hidden" name="hid_name"
value="{{request.identity_context_data.username}}"/>
              <div class="col-6">
                <label for="folio_no" class="form-label">Transferee</label>
                {{ tr_form.transferee }}
              </div>
            </div>
            <div class="row mb-3">
               <div class="col-6">
                 <label for="folio_no" class="form-label">Date of transaction</label>
                 {{ tr_form.event_date }}
               </div>
               <div class="col-6">
                 <label for="folio_no" class="form-label">Class of Shares</label>
                 {{ tr_form.share_class }}
               </div>
             </div>
            <div class="row mb-3">
              <div class="col-6">
                <label for="no_of_shares" class="form-label"
                  >Number of Shares</label>
                {{ tr_form.no_of_shares }}
              </div>
              <div class="col-6">
                <label for="no_of_shares" class="form-label">Nominal Value</label>
                {{ tr_form.nominal_value }}
              </div>
            </div>
            <div class="row mb-3">
              <div class="col">
                <label for="consideration" class="form-label"
                  >Consideration</label
                >
                {{ tr_form.consideration }}
              </div>
            </div>
```

```
          <div class="row mb-3 justify-content-center">
            <div class="col-auto">
              <!-- <button id="submit" type="submit"
              value="Submit" onclick="myFunction()" class="btn btn-sm btn-outline-
primary">Submit</button> -->
              <input
              id="submit"
                type="submit"

                value="Submit"
                class="btn btn-sm btn-outline-primary"

              />

              <script>
                function myFunction() {
                    // window.location.href = '/transaction/transfer'
                    var element = document.getElementById("submit");
                    element.innerHTML = "Submitting";
                    element.classList.remove("btn-outline-primary");
                    element.classList.add("btn-outline-warning");
                    element.disabled = true;

                    setTimeout(function () {
                        var element = document.getElementById("submit");
                        element.innerHTML = "Submit";
                        element.classList.remove("btn-outline-warning");
                        element.classList.add("btn-outline-primary");
                        element.disabled = false;

                    }, 30000)
                }
              </script>
              </div>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>

{% endblock body %}
```

Index.html

```
{% extends 'base.html' %}

{% block body %}
<div class="mt-4">
    <a href="{%  url 'sign_in' %}">home</a>
</div>
{% if request.identity_context_data.authenticated %}
   You're signed in, {{ request.identity_context_data.username }}!
   <a href="/">Logout</a>
{% else %}
   You're not signed in.
   <a href="{% url 'sign_in' %}">Login</a>
{% endif %}

{% endblock %}

{% load static %}
```

Transactionview.html

```
{% extends 'base.html' %}
{% block body %}
{% if request.identity_context_data.authenticated %}
<div class="container">
    <div class="table-responsive">
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Folio Number</th>
                    <th>Date Time</th>
                    <th>Event</th>
                    <th>Number of Shares</th>
                    <th>Nominal value</th>
                    <th>Performed by</th>
                    <th>Total</th>
                </tr>
            </thead>
            <tbody>
                {% for row in data %}
                <tr>
                    <td>{{ row.transaction_id }}</td>
```

```
                <td>{{ row.folio_no }}</td>
                <td>{{ row.event_date }}</td>
                <td>{{ row.event }}</td>
                <td>{{ row.no_of_shares }}</td>
                <td>{{ row.nominal_value }}</td>
                <td>{{ row.user_log }}</td>
                <td>{{ row.total }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
    <button id="dreport" onclick="myFunction()" class="btn btn-sm btn-outline-
primary">Download Report</button>
  </div>
</div>
<script>
  function myFunction() {
    window.location.href = '/generate'
    var element = document.getElementById("dreport");
    element.innerHTML = "Download Inprogress";
    element.classList.remove("btn-outline-primary");
    element.classList.add("btn-outline-warning");
    element.disabled = true;

    setTimeout(function () {
      var element = document.getElementById("dreport");
      element.innerHTML = "Download Report";
      element.classList.remove("btn-outline-warning");
      element.classList.add("btn-outline-primary");
      element.disabled = false;

    }, 10000)
  }

</script>
{% else %}
<!-- You're not signed in.
  <a href="{% url 'sign_in' %}">Login</a> -->
{% include 'auth/401.html' %}
{% endif %}
{% endblock body %}
```

Addcompany.html

```
<div
  class="modal fade"
  id="exampleModal"
  tabindex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div class="modal-dialog">
    <form action="/addcomp" method="post">
      {% csrf_token %}
    <div class="modal-content">
      <div class="modal-header">
        <h1 class="modal-title fs-5" id="exampleModalLabel">Add company</h1>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body">
        <div class="container">
            <div class="row">
              <label for="name" class="form-label">Name</label>
              <!-- <input type="text" name="sh-name" id="" class="form-control"> -->
              {{ cp_form.name }}
            </div>
            <input type="hidden" name="hid_name"
value="{{request.identity_context_data.username}}"/>

            <div class="row">
              <label for="sh-address" class="form-label">Address</label>
              <!-- <textarea name="sh-address" id="" class="form-control"
rows="3"></textarea> -->
              {{ cp_form.address }}
            </div>

              <div class="col-md-6">
                <label for="currency">currency</label>
                {{ cp_form.currency }}
              </div>
```

```
      </div>
    </div>
    <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">
        Close
      </button>
      <button type="submit" class="btn btn-primary">Save changes</button>
    </div>
  </div>
 </form>
 </div>
</div>
```

Addshares.html

```
<div
 class="modal fade"
 id="exampleModal"
 tabindex="-1"
 aria-labelledby="exampleModalLabel"
 aria-hidden="true"
>
 <div class="modal-dialog">
  <form action="/addsh" method="post">
   {% csrf_token %}
  <div class="modal-content">
   <div class="modal-header">
    <h1 class="modal-title fs-5" id="exampleModalLabel">Add Shareholder</h1>
    <button
     type="button"
     class="btn-close"
     data-bs-dismiss="modal"
     aria-label="Close"
    ></button>
   </div>
   <div class="modal-body">
    <div class="container">
        <div class="row">
          <label for="name" class="form-label">Name</label>
          <!-- <input type="text" name="sh-name" id="" class="form-control"> -->
          {{ sh_form.name }}
         </div>
        <div class="row">
          <label for="sh-address" class="form-label">Address</label>
```

```
            <!-- <textarea name="sh-address" id="" class="form-control"
rows="3"></textarea> -->
            {{ sh_form.address }}
          </div>
          <div class="row">
            <label for="sh-lf" class="form-label">Legal form</label>
            <!-- <textarea name="sh-lf" id="" class="form-control"
rows="3"></textarea> -->
            {{ sh_form.legal_form }}
          </div>
      </div>
    </div>
    <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">
        Close
      </button>
      <button type="submit" class="btn btn-primary">Save changes</button>
    </div>


  </div>
 </form>
 </div>
</div>
```

Login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Microsoft Identity Python Web App: Login</title>
</head>
<body>
  <h1>Microsoft Identity Python Web App</h1>

  {% if user_code %}
  <ol>
    <li>To sign in, type <b>{{ user_code }}</b> into
      <a href='{{ auth_uri }}' target=_blank>{{ auth_uri }}</a>
      to authenticate.
    </li>
    <li>And then <a href="{{ url_for('auth_response') }}">proceed</a>.</li>
  </ol>
```

```
    {% else %}
    <ul><li><a href='{{ auth_uri }}'>Sign In</a></li></ul>
    {% endif %}

    <hr>
    <footer style="text-align: right">Microsoft identity platform Web App Sample {{
version }}</footer>
</body>
</html>
```

3.10 Dockerfile

```
FROM registry.access.redhat.com/ubi9/ubi:9.4-1181

LABEL maintainer="Likith B R"

ENV PYTHONUNBUFFERED 1

COPY ./requirements.txt /tmp/requirements.txt


COPY ./app /app
WORKDIR /app
EXPOSE 9000
COPY ./aadconfig.json /
COPY ./redhat1.cer /etc/pki/ca-trust/source/anchors/redhat1.cer
RUN openssl s_client -connect cdn.onetrust.com:443
RUN update-ca-trust enable; update-ca-trust extract

ENV PIP_ROOT_USER_ACTION: ignore
RUN dnf install -y git
RUN yum install -y \
    python3.11  && \
    ln -s /usr/bin/python3.11 /usr/bin/python && \
    curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py" && \
    python get-pip.py && \
    python -m pip install mssql-django && \
    python -m pip install -r /tmp/requirements.txt && \
    rm -rf /tmp && \
    curl https://packages.microsoft.com/config/rhel/9/prod.repo | tee
/etc/yum.repos.d/mssql-release.repo && \
    yum remove unixODBC-utf16 unixODBC-utf16-devel -y && \
    ACCEPT_EULA=Y yum install -y msodbcsql18 && \
    ACCEPT_EULA=Y yum install -y mssql-tools18 && \
```

```
    echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bashrc && \
    source ~/.bashrc && \
    yum install -y unixODBC-devel && \
    adduser \
        --no-create-home \
        rhel-user && \
    mkdir -p /vol/web/media && \
    mkdir -p /vol/web/static && \
    chown -R rhel-user:rhel-user /vol && \
    chmod -R 755 /vol

ENV PATH="/scripts:/py/bin:$PATH"

USER rhel-user
```

3.11 Docker-compose.yml

```
FROM registry.access.redhat.com/ubi9/ubi:9.4-1181

LABEL maintainer="Likith B R"

ENV PYTHONUNBUFFERED 1

COPY ./requirements.txt /tmp/requirements.txt


COPY ./app /app
WORKDIR /app
EXPOSE 9000
COPY ./aadconfig.json /
COPY ./redhat1.cer /etc/pki/ca-trust/source/anchors/redhat1.cer
RUN openssl s_client -connect cdn.onetrust.com:443
RUN update-ca-trust enable; update-ca-trust extract

ENV PIP_ROOT_USER_ACTION: ignore
RUN dnf install -y git
RUN yum install -y \
    python3.11  && \
    ln -s /usr/bin/python3.11 /usr/bin/python && \
    curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py" && \
    python get-pip.py && \
    python -m pip install mssql-django && \
    python -m pip install -r /tmp/requirements.txt && \
    rm -rf /tmp && \
```

```
    curl https://packages.microsoft.com/config/rhel/9/prod.repo | tee
/etc/yum.repos.d/mssql-release.repo && \
    yum remove unixODBC-utf16 unixODBC-utf16-devel -y && \
    ACCEPT_EULA=Y yum install -y msodbcsql18 && \
    ACCEPT_EULA=Y yum install -y mssql-tools18 && \
    echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bashrc && \
    source ~/.bashrc && \
    yum install -y unixODBC-devel && \
    adduser \
        --no-create-home \
        rhel-user && \
    mkdir -p /vol/web/media && \
    mkdir -p /vol/web/static && \
    chown -R rhel-user:rhel-user /vol && \
    chmod -R 755 /vol

ENV PATH="/scripts:/py/bin:$PATH"

USER rhel-user
```

Chapter 4: Testing

Testing is the process of evaluating a software product or application to ensure it meets the required specifications and functions as expected. It's a crucial part of the software development lifecycle (SDLC) that aims to identify defects, errors, or bugs in the software before it's released to end-users.

Types of Testing:

Functional Testing: Focuses on verifying the functionality of the software against the specified requirements. Examples include:

- **Unit Testing:** Testing individual components or modules of the software.
- **Integration Testing:** Testing the interaction between different modules.
- **System Testing:** Testing the entire integrated system.
- **Acceptance Testing:** Testing to determine if the software meets the end-user's needs and is ready for deployment.

Non-Functional Testing: Focuses on aspects other than the functionality, such as:

- **Performance Testing:** Evaluating the software's responsiveness, stability, and scalability under various loads.
- **Security Testing:** Identifying vulnerabilities and ensuring the software is protected against unauthorized access.
- **Usability Testing:** Assessing how easy and intuitive the software is to use.
- **Compatibility Testing:** Ensuring the software works correctly across different environments (browsers, operating systems, devices).

Testing Techniques: Based on the tester's knowledge of the software's internal workings:

- **Black-Box Testing:** Testing without knowledge of the internal code structure.
- **White-Box Testing:** Testing with knowledge of the internal code structure.
- **Gray-Box Testing:** Testing with partial knowledge of the internal code structure.

4.1 white box manual testing

White-box testing is a software testing technique that focuses on examining the internal structure and workings of an application. It involves testing the code itself, including its logic, flow, and structure. Testers who perform white-box testing need to have knowledge of the internal workings of the code being tested.

Key Aspects

- **Internal Structure:** White-box testing focuses on verifying that the internal operations of a software system work as expected. This includes testing code paths, branches, statements, and conditions.

- **Code Knowledge:** Testers must have programming knowledge to understand the code and design effective tests.
- **Thoroughness:** It aims to ensure that all parts of the code have been executed at least once.

The testing strategy for the Portfolio Manager and Workflow Automation tool will incorporate white box manual testing techniques to ensure the thorough examination of the system's internal structure and code. This will involve the creation of test cases that cover various aspects of the code.

Techniques Used in White-Box Manual Testing

Condition Coverage: The tester aims to test all possible outcomes.

The goal of this testing phase is to ensure the application functions as expected, validates user inputs correctly, and provides a seamless user experience. We will employ a combination of functional testing techniques to verify different aspects of the application.

The following types of functional tests will be conducted:

- **Login and Authentication Testing:** This category will focus on verifying the user login process, including redirection through Azure App Service and the implementation of Multi-Factor Authentication (MFA). We will ensure users can successfully log in and that the MFA flow is correctly triggered.
- **Navigation Testing:** This will involve validating the application's navigation flow, ensuring that buttons and links redirect users to the intended pages. Specifically, we will verify the navigation to the "Add Shareholder" page.
- **Data Validation Testing:** This crucial area will focus on ensuring the application correctly validates user inputs. This includes:
    - **Mandatory Field Validation:** We will test that the system prevents users from submitting forms without filling in all required fields on pages like "Add Shareholder" and "Add Share." We expect the system to display appropriate error messages.
    - **Data Type Validation:** We will verify that the system handles different data types correctly in relevant fields. For example, we will test the acceptance of alphanumeric characters in text fields and the rejection of non-numeric characters in numeric fields like "Consideration."
    - **Business Rule Validation:** We will test the implementation of specific business rules, such as preventing the addition of a second "Incorporation" event for the same shareholder.
- **Functional Workflow Testing:** This will involve testing specific user workflows, such as the process of adding a new shareholder and adding share transactions. We will ensure that data is correctly processed and stored in the database.
- **Logout Testing:** We will verify the application's logout functionality, ensuring users are properly redirected to the Microsoft logged-out page after initiating the logout process.

Phase 1

Test cases Table

Test Case	Objective	Description /steps	Test data	Expected Result	Actual result	Status
PW-01	To validate login from Azure app service	Redirect to Server URL provided in azure portal	URL - https://localhost:9000/home	Redirect to home page	Redirect to home page	PASS
PW-02	To validate MFA for login	Redirect to MFA approval page	URL - https://localhost:9000/home	Redirect to home page	Redirect to home page	PASS
PW-03	To validate Add shareholder button	Redirect to add shareholder page	https://localhost:9000/transaction/add	Redirect to add shareholder page	Redirect to add shareholder page	PASS
PW-04	To validate Add shareholder page Mandatory field	error occurs no success message	Name : ABC Address : NULL Legal from : NULL	error message "Please fill out this field "	Shareholder added Successfully	FAIL
PW-05	To validate Add shareholder page Alpha numeric Text field	Success message occurs updates in database	Name : ABC$ Address : ##123 Legal from : @122frff	Shareholder added Successfully	Shareholder added Successfully	PASS
PW-06	To validate Add share page to add money to sharehol	Success message occurs updates in database	Folio Number: ABC$ Event : Incorporation Class of Shares : Ordinary Shares Event Date : "select today" Number of shares: 1000	Transaction added Successfully	Transaction added Successfully	PASS

	der database		Nominal Value: 0.01 Consideration : 1000			
PW-07	To validate Add share page to add money to sharehol der database with incorpora tion event at second time	error occurs no success message	Folio Number: ABC$ Event : Incorporation Class of Shares : Ordinary Shares Event Date : "select today" Number of shares: 1000 Nominal Value: 0.01 Consideration : 1000	Sharehold er already incorporat ed	Sharehol der already incorpora ted	PASS
PW-08	To validate Add shares page Mandato ry field	error occurs no success message	Folio Number: NULL Event : Incorporation Class of Shares : NULL Event Date :NULL Number of shares: NULL Nominal Value: 0.01 Consideration : NULL	error message "Please fill out this field "	Transacti on added Successfu lly	FAIL
PW-09	To validate logout button	Redirects to Microsoft logged out page	click button logout	Redirects to Microsoft logged out page	Redirects to Microsoft logged out page	PASS
PW-10	To validate Add share page to add money to sharehol der database	error occurs no success message	Folio Number: ABC$ Event : Incorporation Class of Shares : Ordinary Shares Event Date : "select today" Number of shares: ABC Nominal Value: 0.01 Consideration : 1000	No data fill in the field	No data fill in the field	PASS

	with Alphabet in considera tion field					

Table-7: Test cases phase 1

Phase 2

The following types of functional tests will be conducted within the "Share Transactions" and "Company Management" modules:

Following the initial phase of testing focused on core login and shareholder management functionalities, this section outlines the planned testing activities for the "Share Transactions" and "Company Management" modules of the application. The primary objective is to ensure the correct processing of share transfers, share conversions, and the management of company information, including robust data validation and adherence to business rules.

- **Share Transfer Testing:** This category will focus on validating the functionality for transferring shares between shareholders. We will test successful transfers, attempts to transfer to the same shareholder, and scenarios where the transferring shareholder has insufficient shares.
- **Share Conversion Testing:** This area will involve verifying the process of converting shares from one class to another. We will test successful conversions, mandatory field validation on the conversion page, attempts to convert without selecting target classes, and scenarios where there are insufficient shares for conversion.
- **Mandatory Field Validation (Revisited):** We will continue to ensure that mandatory fields on various forms within these modules (e.g., "Add Shareholder," "Add Share," "Transfer Shares," "Convert Shares," "Add Company") are correctly validated, and the system prevents submission with missing required information.
- **Navigation Testing (Expanded):** We will validate navigation to specific pages within these modules, such as the "Add Company," "Transfer," and "Convert" pages, ensuring the relevant buttons and links function as expected.
- **Data Validation (Specific Fields):** We will specifically test the acceptance of alphanumeric characters in relevant text fields on the "Add Company" page.
- **Business Rule Validation (Transfer & Conversion):** We will verify the implementation of business rules specific to share transfers and conversions, such as preventing self-transfers and ensuring sufficient funds/shares are available.
- **User Experience Testing (Brief - Button Behaviour):** We will briefly assess the behaviour of certain buttons, such as the "Submit" and "Download" buttons, to ensure they provide appropriate feedback or delays (e.g., a 10-second halt as indicated) to prevent unintended duplicate submissions or download requests.

- **Security/Access Control (Basic):** We will perform a basic test to ensure that unauthenticated users attempting to access specific URLs within these modules are correctly redirected to the login page, reinforcing the application's security measures.

Test cases after ‘Transfer’ and ‘Conversion of share’ function integration into the project.

Test Case	Objective	Description /steps	Test data	Expected Result	Actual result	Status
PW2-01	To validate Transfer of shares from Shareholder A to shareholder B	Success message occurs transfer money to shareholder B updates in database	Transferer:XYZ Transferee:ABC$ Date of Transaction: "select today" class of shares:ordinary shares Number of shares:1000 Nominal Value:0.01 Consideration:1000	Transfer successful	Transfer successful	PASS
PW2-02	To validate Transfer of shares from Shareholder A to shareholder A	error message occurs do not transfer money to shareholder	Transferer:XYZ Transferee:XYZ Date of Transaction: "select today" class of shares:ordinary shares Number of shares:1000 Nominal Value:0.01 Consideration:1000	error "Cannot transfer funds to same share holder"	error "Cannot transfer funds to same share holder"	PASS
PW2-03	To validate Transfer of shares from Shareholder A to shareholder B if no funds in	error message occurs do not transfer money to shareholder	Transferer:XYZ Transferee:ABC$ Date of Transaction: "select today" class of shares:ordinary shares Number of shares:2000 Nominal Value:0.01 Consideration:1000	error "Cannot transfer funds Insufficient funds "	error "Cannot transfer funds Insufficient funds "	PASS

	Shareholder A					
PW2-04	To validate Add shareholder page Mandatory field	error occurs no success message	Name : ABC Address : NULL Legal from : NULL	error message "Please fill out this field "	error message "Please fill out this field "	PASS
PW2-05	To validate Add shares page Mandatory field	error occurs no success message	Folio Number: NULL Event : Incorporation Class of Shares : NULL Event Date :NULL Number of shares: NULL Nominal Value: 0.01 Consideration : NULL	error message "Please fill out this field "	error message "Please fill out this field "	PASS
PW2-06	To validate Transfer shares page Mandatory field	error occurs no success message	Transferer:NULL Transferee:NULL Date of Transaction: NULL class of shares:NULL Number of shares:NULL Nominal Value:NULL Consideration:NULL	error message "Please fill out this field "	error message "Please fill out this field "	PASS
PW2-07	To validate convertion of shares of Shareholder A	Success message occurs convert shares of shareholder A updates in database	Folio Number: XYZ Date of Transaction: "select today" Class of shares :ordinary shares convertion in to classes selection class : A quantity:50 class : B quantity:50 consideration : 100	Convertion successful	Convertion successful	PASS

PW2-08	To validate convert shares page Mandatory field	error occurs no success message	Folio Number: NULL Date of Transaction: NULL Class of shares :ordinary shares convertion in to classes selection class : NULL quantity:NULL class : NULL quantity:NULL consideration : NULL	error message "Please fill out this field "	error message "Please fill out this field "	PASS
PW2-09	To validate convert shares if no class selected to convert	error occurs no success message	Folio Number: XYZ Date of Transaction: "select today" Class of shares :ordinary shares convertion in to classes selection consideration : 100	error message "Please fill out this field "	error message "Please fill out this field "	PASS
PW2-10	To validate convert shares if no fund to convert	error occurs no success message	Folio Number: XYZ Date of Transaction: "select today" Class of shares :ordinary shares convertion in to classes selection class : A1 quantity:50 class : B quantity:50 consideration : 100	error message "Error occurred no shares converted "	error message "Error occurred no shares converted "	PASS
PW2-11	To validate submit button halt for 10 seconds	Success message no two entries for same transaction	Folio Number: XYZ Date of Transaction: "select today" Class of shares :ordinary shares convertion in to classes selection class : A quantity:50 class : B quantity:50 consideration : 100	Convertion successful	Convertion successful	PASS

PW2-12	To validate download button halt for 10 seconds	document downloads successfully no two copies download request	click on download button	Download successful	Download successful	PASS
PW2-13	To validate Add company button	Redirect to add company page	https://localhost:9000/company	Redirect to add company page	Redirect to add company page	PASS
PW2-14	To validate Add company page Alpha numeric Text field	Success message occurs updates in database	Name : ABC$ Address : ##123 curency : Rs	Company added Successfully	Company added Successfully	PASS
PW2-15	To validate Transfer button	Redirect to add Transfer page	URL-https://localhost:9000/transaction/transfer	Redirect to transfer page	Redirect to transfer page	PASS
PW2-16	To validate convert button	Redirect to add convert page	URL-https://localhost:9000/transaction/convert	Redirect to convert page	Redirect to convert page	PASS
PW2-17	Accessing url with out login	Redirect to login page login after clicking button and uses Azure app service to login	URL-https://localhost:9000/transaction/convert	Redirect to login page	Redirect to login page	PASS

Table-7: Testcase phase 2

4.2 Conclusion

This project, "Portfolio Manager and Workflow Automation," represents a significant step towards streamlining and enhancing the management of complex financial portfolios. By leveraging the Django framework, Microsoft SQL database, and Docker Compose, we've created a robust and scalable solution capable of handling the intricate demands of managing numerous company portfolios, shareholders, and transactions. The integration with Azure App Services for authentication further strengthens the system's security and reliability.

Key Achievements and Features:

- **Database Management:** The Microsoft SQL database, containerized for easy deployment and management, provides a reliable and efficient storage solution for the vast amount of data generated by the application. This includes detailed information on over 100 company portfolios, each with 100+ shareholders, and a comprehensive record of all transactions.
- **User Input and Data Handling:** The application's ability to take user input and append data to the database is crucial for maintaining accurate and up-to-date records. This functionality ensures that all relevant information, from shareholder details to transaction specifics, is captured and stored securely.
- **Report Generation:** The capability to generate formatted Excel reports is a key feature for providing stakeholders with clear, concise, and easily digestible information. These reports can be customized to meet specific needs, offering valuable insights into portfolio performance, transaction history, and other critical metrics.
- **Role-Based Access Control (RBAC):** The implementation of RBAC is essential for ensuring global compliance. By restricting access to sensitive data and functionalities based on user roles (management and employees), the system mitigates the risk of unauthorized access and data breaches, thereby promoting a secure and compliant environment.
- **Scalability and Maintainability:** The use of Docker Compose for containerization simplifies the deployment and management of the application and database. This approach enhances scalability, allowing the system to handle increasing data volumes and user loads. It also improves maintainability, making it easier to update, debug, and expand the application in the future.
- **Azure App Services Integration:** Integrating Azure App Services for authentication provides a secure and reliable way to manage user access. This integration leverages Microsoft's robust security infrastructure, ensuring that only authorized users can access the application and its data.

Challenges and Solutions:

The development of this project involved several challenges, including:

- **Data Volume:** Managing the sheer volume of data associated with 100+ company portfolios and their respective shareholders required careful database design and

optimization. We addressed this by implementing efficient data structures, indexing strategies, and query optimization techniques.

- **Transaction Complexity:** Recording and tracking transactions between multiple companies presented a complex data modelling challenge. We solved this by designing a relational database schema that accurately captures the relationships between companies, shareholders, and transactions, ensuring data integrity and consistency.
- **Report Generation:** Generating formatted Excel reports required the use of a suitable library and careful consideration of report layout and content. We selected a robust library and developed a flexible reporting mechanism that allows for customization and efficient data retrieval.
- **Security:** Ensuring the security of sensitive financial data was paramount. The integration of Azure App Services for authentication, combined with the RBAC implementation, provides a strong security foundation.

Future Directions:

This project lays a solid foundation for future development and expansion. Potential areas for improvement and enhancement include:

- **Enhanced Reporting:** Implementing more advanced reporting features, such as interactive dashboards, data visualization tools, and customizable report templates.
- **Workflow Automation:** Expanding the workflow automation capabilities to include automated transaction processing, portfolio rebalancing, and compliance checks.
- **API Integration:** Developing an API to allow for seamless integration with other financial systems and applications.
- **Mobile Accessibility:** Creating a mobile application to provide users with convenient access to portfolio information and management tools on the go.
- **Advanced Analytics:** Incorporating advanced analytics and machine learning techniques to provide insights into portfolio performance, identify trends, and support informed decision-making.

In conclusion, the "Portfolio Manager and Workflow Automation" project successfully addresses the complex needs of managing large-scale financial portfolios. By combining the power of the Django framework, Microsoft SQL database, and Docker Compose with the security of Azure App Services, we have created a robust, scalable, and secure solution. The application's ability to handle vast amounts of data, generate formatted reports, and enforce RBAC makes it a valuable tool for financial institutions and organizations managing complex investment portfolios. The project's modular design and scalability ensure that it can adapt to future growth and evolving business requirements.

4.3 References

- Django: The official Django documentation is a great starting point: https://www.djangoproject.com/
- Docker Compose: Docker's documentation on using Compose: https://docs.docker.com/compose/
- Azure App Services: Microsoft's documentation for deploying web apps: https://azure.microsoft.com/en-in/products/app-service/
- Django and Azure: Tutorial on deploying Django with MSSQL on Azure: https://learn.microsoft.com/en-us/azure/app-service/
- YouTube for Azure cloud : https://youtube.com
- For deployment : https://learn.microsoft.com/en-us/shows/azure-friday/python-on-azure-part-2-deploying-django-services-to-azure-web-apps
- Stack overflow for code: https://stackoverflow.com/questions/13197574/openpyxl-adjust-column-width-size

Links

https://youtube.com

https://www.geeksforgeeks.com

https://stackoverflow.com

https://learn.microsoft.com

4.4 Future implementation

The "Portfolio Manager and Workflow Automation" project, with its foundation in Django, Microsoft SQL or PostgreSQL, Docker Compose, and Azure App Services, provides a robust platform for managing complex financial portfolios. Its current capabilities, including handling extensive data, generating reports, and enforcing RBAC, address core industry needs. However, several future implementations could significantly enhance its functionality, expand its user base, and capitalize on emerging technologies. This document outlines potential future directions, categorized for clarity:

I. Enhanced Functionality and User Experience

- **Interactive Dashboards and Data Visualization:**
    - Current reporting relies on Excel files. Integrating interactive dashboards with real-time data visualization would provide users with dynamic insights. Libraries like Chart.js, D3.js, or even Django-based solutions could be employed.
    - Users could customize dashboards to track specific KPIs, visualize portfolio performance over time, and identify trends at a glance.
- **Advanced Search and Filtering:**
    - Implementing more sophisticated search functionalities would allow users to quickly find specific data points within the vast database.
    - This could include full-text search, faceted search, and the ability to combine multiple filters (e.g., filter transactions by date range, company, and shareholder).
- **Customizable Workflows:**
    - Expanding the workflow automation capabilities beyond basic transaction recording is crucial.
    - This could involve allowing users to define custom workflows for various processes, such as:
        - Automated portfolio rebalancing based on predefined rules.
        - Approval processes for significant transactions.
        - Automated generation of compliance reports.
- **Mobile Accessibility:**
    - Developing a mobile application (or a responsive web interface) would enable users to access and manage portfolios on the go.
    - This would improve convenience and accessibility, particularly for stakeholders who need to monitor portfolio performance or approve transactions remotely.
- **Client Portal:**
    - Creating a secure client portal where shareholders can access their portfolio information, view transaction history, and communicate with company management would enhance transparency and investor relations.

II. Integration and Expansion

- **API Integration:**

    - Developing a well-documented API would allow the system to integrate with other financial platforms and services.
    - This could include:
        - Integration with market data providers for real-time stock quotes.
        - Integration with accounting software for seamless financial reporting.
        - Integration with payment gateways for automated transaction processing.
- **Multi-Tenancy:**
    - Adapting the system to support multi-tenancy would allow it to serve multiple organizations or clients within a single instance.
    - This would be beneficial for financial institutions that manage portfolios for various clients.
- **Support for Additional Asset Classes:**
    - Currently, the system focuses on company portfolios and shareholders. Expanding support to include other asset classes, such as:
        - Bonds
        - Real estate
        - Commodities
        - Cryptocurrencies
    - This would broaden the system's appeal to a wider range of investors and portfolio managers.
- **Global Expansion:**
    - To cater to a global audience, the system could be enhanced with:
        - Multi-language support.
        - Support for different currencies.
        - Compliance with international financial regulations.

III. Advanced Technologies

- **AI-Powered Analytics:**
    - Integrating artificial intelligence (AI) and machine learning (ML) could provide valuable insights and automate complex tasks.
    - Potential applications include:
        - Predictive analytics for forecasting portfolio performance.
        - Risk assessment and management using ML algorithms.
        - Automated fraud detection.
        - Personalized investment recommendations for shareholders.
- **Blockchain Integration:**
    - Exploring the use of blockchain technology could enhance security, transparency, and efficiency.
    - For example, blockchain could be used to:
        - Securely record and track transactions.
        - Manage shareholder ownership and voting rights.
        - Automate dividend distribution through smart contracts.
- **Cloud Optimization:**
    - Leveraging advanced cloud services on Azure could further improve scalability, reliability, and cost-effectiveness.

- This could involve:
  - Using Azure Kubernetes Service (AKS) for container orchestration.
  - Implementing serverless computing with Azure Functions.
  - Utilizing Azure Cosmos DB for globally distributed data storage.

IV. Security Enhancements

- **Multi-Factor Authentication (MFA):**
  - Implementing MFA would add an extra layer of security to user authentication, reducing the risk of unauthorized access.
- **Biometric Authentication:**
  - Exploring biometric authentication methods, such as fingerprint or facial recognition, could provide a more secure and user-friendly login experience.
- **Regular Security Audits:**
  - Establishing a process for regular security audits and penetration testing would help identify and address potential vulnerabilities.
- **Data Encryption:**
  - Implementing end-to-end data encryption, both in transit and at rest, would ensure the confidentiality and integrity of sensitive financial information.

By implementing these future enhancements, the "Portfolio Manager and Workflow Automation" project can evolve into a comprehensive, cutting-edge solution for managing financial portfolios in an increasingly complex and interconnected world.