Database Management System
Mini Project
On
Personal Finance Management System

Submitted by

PA29 Sahaj Mishra (1032211066)
PA12 Manan Makhija (1032210753)
PA17 Pratyush Chowdhury (1032210864)
PA21 Gaurav Choudhary (1032210933)
PA25 Nandana Nambiar (1032210983)

Under the guidance of
Dr. Himangi Pande

School of Computer Engineering and Technology
MIT World Peace University, Kothrud,
Pune 411 038, Maharashtra - India

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

C E R T I F I C A T E

This is to certify that

PA29 Sahaj Mishra (1032211066)
PA12 Manan Makhija (1032210753)
PA17 Pratyush Chowdhury (1032210864)
PA21 Gaurav Choudhary (1032210933)
PA25 Nandana Nambiar (1032210983)

of SY-BTech. (Computer Engineering and Technology – Cybersecurity and Forensics) have partially completed their Mini Project report on Personal Finance Management System and have submitted this End term partial report towards fulfillment of the requirement for the Degree-Bachelor of Computer Science & Engineering (BTech CSE CSF) for the academic year 2022-2023.

Dr. Himangi Pande
Mini Project Guide
School of CET
MIT WPU, Pune

INDEX

1. Introduction

The goal of the Personal Finance Management project is to create a web-based application that will assist people in properly managing their personal money. Users can monitor their earnings, outgoing costs, and savings using the system, which also offers a number of tools to aid in planning and achieving their financial objectives.

The programme offers an interface for adding and classifying spending, creating budgets. Additionally, users can make personalised savings programmes and monitor their progress towards their objectives.

The system is developed using Python and MySQL. Users may easily manage their finances while on journeys due to the application's availability from any device with an internet connection.

The overall goal of this project is to give consumers the tools they need to manage their money better.

2. Future prospects of the project

As technology continues to evolve, project finance management systems are likely to integrate with emerging technologies such as artificial intelligence (AI), blockchain, and the Internet of Things (IoT). These technologies can enhance the system's capabilities, providing real-time data analysis, and improving the accuracy and efficiency of financial reporting

3. Problem Statement

Many people struggle to effectively manage their personal finances, leading to financial stress, overspending, and poor financial decisions. To address this problem, we propose the development of a personal finance management app that helps users track their income, expenses, and overall financial health.

The app will include several key features, including a login system to ensure user privacy and security, a dashboard that provides a quick overview of the user's financial status, and functionality to track revenue earned and expenditures. The app will be designed to be intuitive and user-friendly, with clear visualisations and easy-to-use interfaces that enable users to quickly and easily view and manage their finances.

4. Entities

1. **Users :** Users will be the one who is using the Personal Finance Management System for Overlooking their account. Users of the system will have an Account id. Customer entities will store details like the user's name, date of birth, account number, mobile number, registered date, user id, password, email id.

2. **Accounts :** Accounts will contain details like account type, balance, interest rate, open date, account id, ifsc code, user id. Account id will be Primary key and User id would be the Foreign key.

3. **Expenditures :** Expenditure will be storing details such as Expense id, Account id, User id, Date, Category (Home, Education, Professional, etc related). Account id will be Primary key and User id would be the Foreign key, Expense id will be the Primary key.

4. **Income :** Income will enlist the amount, income id, account id, user id and date of income. Account id and user id will be the foreign key and account id will be the primary key.
5. **Investments :** Investments will have the purchase date, investment name, purchase price, investment type, account id, user id, current value, return rate, investment id.

5. Relationships

1. **User to Account :** Every User is associated with an account. An User will create its account having all the basic account details. The relation name is has'.

2. **User to Income :** Users' income will be displayed after the user will upload income details from its account. The relation name is has'.

3. **User to Investments :** Users will be managing the details of investments they have done from their account. The relation name is 'manage'.

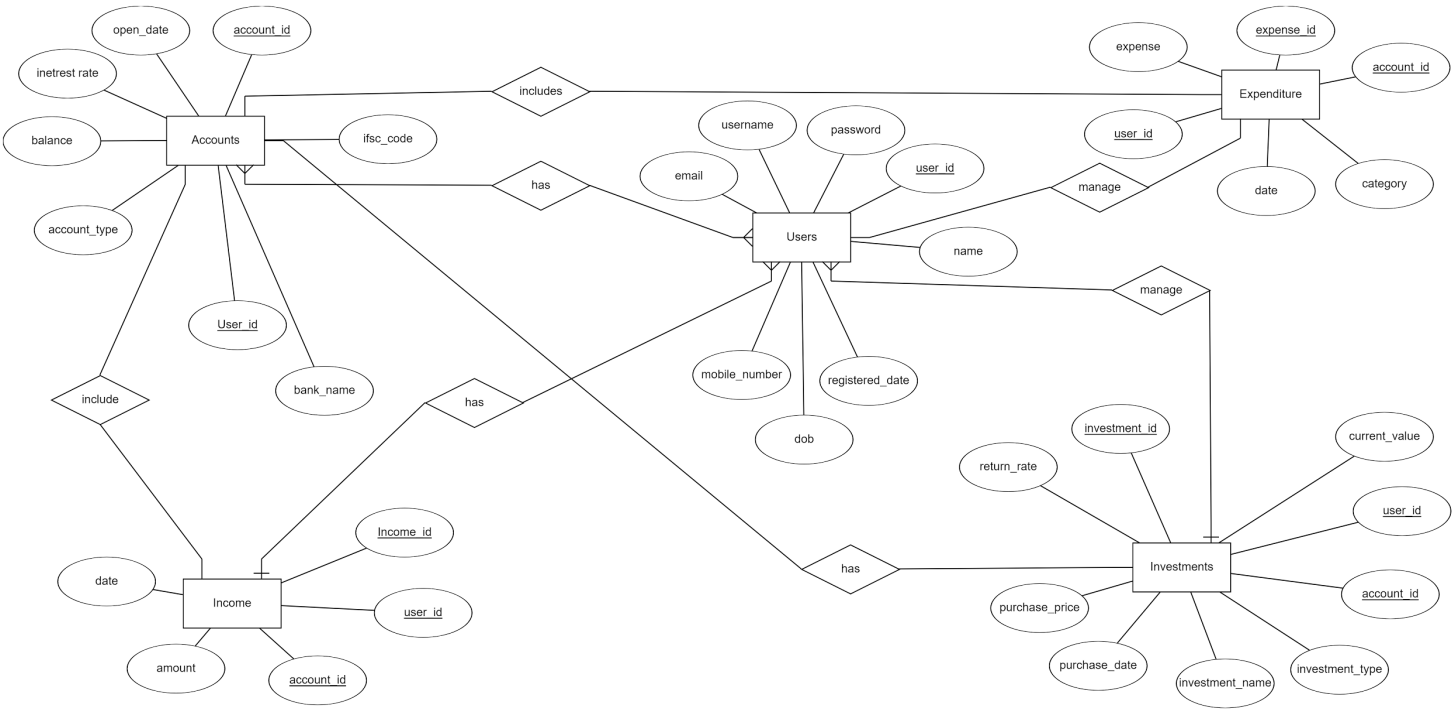
4. **User to Expenditure :** Users will be managing and keeping record of their expenditures via their accounts. The relation name is 'manage'.

5. **Accounts to Expenditure :** Users' Accounts will manage and contain the details of their expenditures and transactions. The relation name is 'includes'.

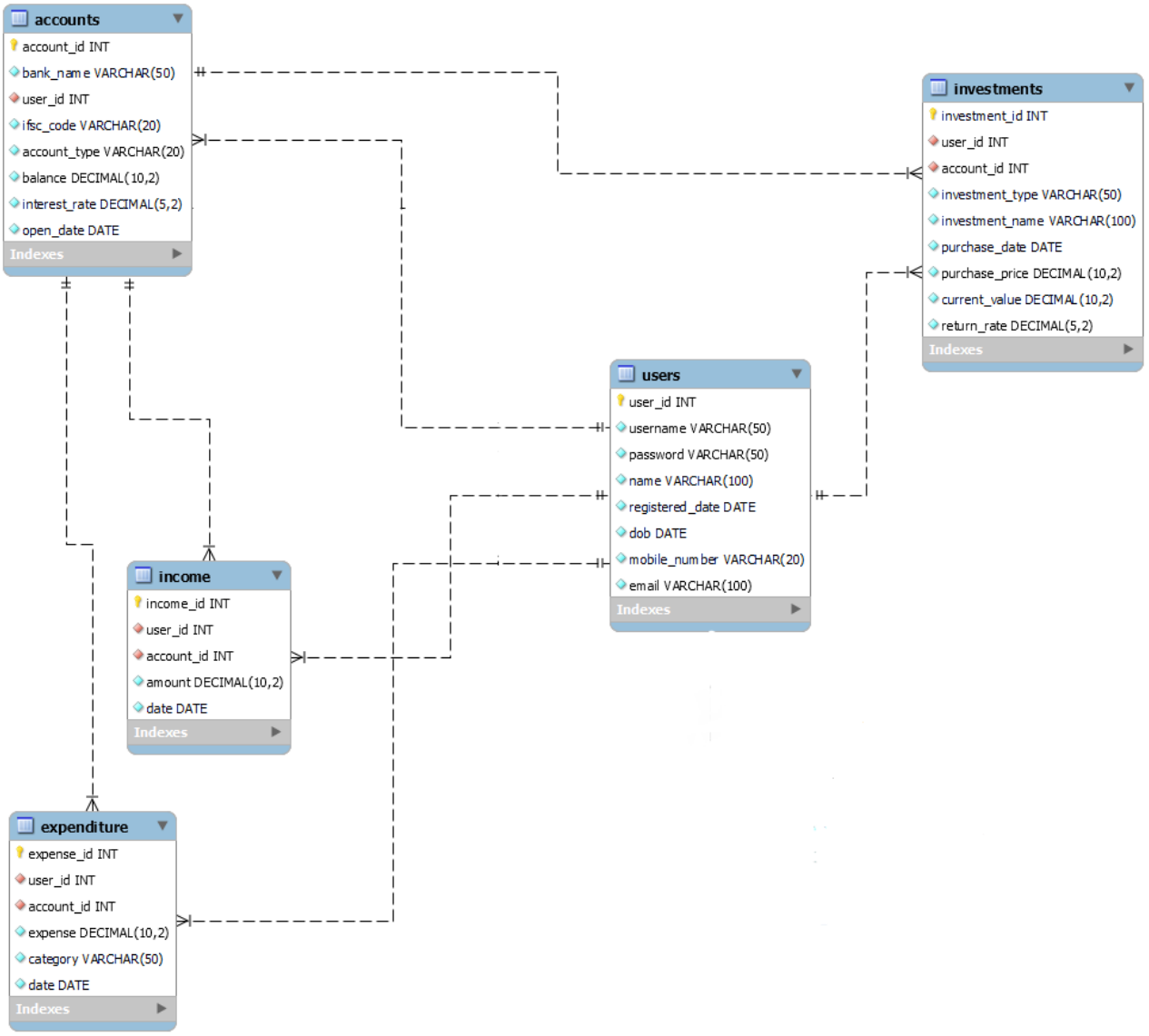
6. **Accounts to Income :** Users' Accounts will manage and contain the details of their income. The relation name is 'includes'.

7. **Accounts to Investments :** Users' can view the details of investments they have made from their accounts. The relation name is 'has'.

6. ER Diagram



7. Schematic diagram (ER Diagrams into tables)



8. SQL Statements

```
CREATE TABLE if not exists User (
user_id INT PRIMARY KEY,
username VARCHAR(50) UNIQUE NOT NULL,
password VARCHAR(50) NOT NULL,
name VARCHAR(100) NOT NULL,
registered_date DATE NOT NULL,
dob DATE NOT NULL,
mobile_number VARCHAR(20) NOT NULL,
email VARCHAR(100) unique NOT NULL
);;

CREATE TABLE if not exists Account (
account_id BIGINT PRIMARY KEY,
bank_name VARCHAR(50) NOT NULL,
user_id INT NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id),
ifsc_code VARCHAR(20) NOT NULL,
account_type VARCHAR(20) NOT NULL,
balance DECIMAL(20, 2) NOT NULL,
nominee_name VARCHAR(100) NOT NULL,
interest_rate DECIMAL(5, 2) NOT NULL,
open_date DATE NOT NULL
);;
```

```
CREATE TABLE IF NOT EXISTS Income (
income_id INT PRIMARY KEY,
user_id INT NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id),
account_id BIGINT NOT NULL,
FOREIGN KEY (account_id) REFERENCES Account(account_id),
amount DECIMAL(15, 2) NOT NULL,
income_date DATE NOT NULL,
source VARCHAR(255) NOT NULL,
remarks VARCHAR(255) not null,
category VARCHAR(255) NOT NULL
);;

CREATE TABLE if not exists Expenditure (
expense_id INT PRIMARY KEY,
user_id INT NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id),
ccount_id BIGINT NOT NULL,
FOREIGN KEY (account_id) REFERENCES Account(account_id),
expense DECIMAL(15, 2) NOT NULL,
category VARCHAR(50) NOT NULL,
expense_date DATE NOT NULL,
source VARCHAR(255) NOT NULL,
remarks VARCHAR(255) not null
);;
```

```
CREATE TABLE if not exists Investment (
investment_id INT PRIMARY KEY,
user_id INT NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id),
account_id BIGINT NOT NULL,
FOREIGN KEY (account_id) REFERENCES Account(account_id),
investment_type VARCHAR(50) NOT NULL,
investment_name VARCHAR(100) NOT NULL,
purchase_date DATE NOT NULL,
purchase_price DECIMAL(15, 2) NOT NULL,
current_value DECIMAL(15, 2) not null,
return_rate DECIMAL(5,2) NOT NULL,
number_of_units DECIMAL(10, 2) not null,
remarks VARCHAR(255) not null
);;
```

9. Normalisation

Here we have checked for normalisation (1NF, 2NF, 3NF, 3.5NF BCNF)

9.1. User table:

1NF: The table is in 1NF because all attributes have atomic values and there are no repeating groups.
2NF: The table is in 2NF because it is in 1NF and there are no partial dependencies, as there is only one candidate key (user_id).
3NF: The table is in 3NF because it is in 2NF and there are no transitive dependencies.
3.5NF (BCNF): The table is in BCNF because every determinant is a candidate key (user_id is the only determinant).

9.2. Account table:

1NF: The table is in 1NF because all attributes have atomic values and there are no repeating groups.
2NF: The table is in 2NF because it is in 1NF and there are no partial dependencies, as there is only one candidate key (account_id).
3NF: The table is in 3NF because it is in 2NF and there are no transitive dependencies.
3.5NF (BCNF): The table is in BCNF because every determinant is a candidate key (account_id is the only determinant).

9.3. Income table:

1NF: The table is in 1NF because all attributes have atomic values and there are no repeating groups.
2NF: The table is in 2NF because it is in 1NF and there are no partial dependencies, as there is only one candidate key (income_id).
3NF: The table is in 3NF because it is in 2NF and there are no transitive dependencies.
3.5NF (BCNF): The table is in BCNF because every determinant is a candidate key (income_id is the only determinant).

9.4. Expenditure table:

1NF: The table is in 1NF because all attributes have atomic values and there are no repeating groups.
2NF: The table is in 2NF because it is in 1NF and there are no partial dependencies, as there is only one candidate key (expense_id).
3NF: The table is in 3NF because it is in 2NF and there are no transitive dependencies.
3.5NF (BCNF): The table is in BCNF because every determinant is a candidate key (expense_id is the only determinant).

9.5. Investment table:

1NF: The table is in 1NF because all attributes have atomic values and there are no repeating groups.
2NF: The table is in 2NF because it is in 1NF and there are no partial dependencies, as there is only one candidate key (investment_id).
3NF: The table is in 3NF because it is in 2NF and there are no transitive dependencies.
3.5NF (BCNF): The table is in BCNF because every determinant is a candidate key (investment_id is the only determinant).

10. PL SQL Statements and CRUD Operations

10.1. Creating the New Database for the Project and Reading the SQL script from a file to create all the tables that are required for the project

```
try:
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root"
    )
    if connection.is_connected():
        cursor = connection.cursor()
                                    cursor.execute("CREATE DATABASE IF NOT EXISTS
Personal_Finance_Management")
        connection.database = 'Personal_Finance_Management'
        with open(path + 'login.sql') as f:
            queries = f.read().split(';\n\n')
        for query in queries:
            cursor.execute(query)
        connection.commit()

except Error as e:
    print(f"Error: {e}")
    return None

finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
```

10.2. This is how we connect the Python Application to the MySQL Database, we use the <u>Cursors</u> to execute the SQL queries and fetch the results

```
try:
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="Personal_Finance_Management"
    )
    if connection.is_connected():
        cursor = connection.cursor()
        query = "SELECT username FROM User WHERE BINARY username = %s"
        cursor.execute(query, (username,))
        if cursor.fetchone() or " " in username or len(username) < 5 or len(username) > 20 :
            return False
        else:
            return True

except Error as e:
    print(f"Error: {e}")
    return False

finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
```

10.3. Python Functions that has the same functionality as the PL/SQL functions and procedures

```
def is_valid_name(name):
        return all(char.isalpha() or char.isspace() for char in name)

def is_valid_date(date_str):
    try:
        input_date = datetime.strptime(date_str, '%d-%m-%Y')
        current_date = datetime.now()
        if input_date > current_date:
            return False
        return True
    except ValueError:
        return False

def is_valid_mobile_number(mobile_number):
    if len(mobile_number) == 10 and mobile_number.isdigit():
        return True
    else:
        return False

def is_valid_email(email):
    try:
        valid = validate_email(email)
        return True
    except EmailNotValidError as e:
        return False

def generate_unique_number():
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="Personal_Finance_Management"
    )

    if connection.is_connected():
        cursor = connection.cursor()
        query = "SELECT user_id FROM User"
        cursor.execute(query)
        user_id_list = cursor.fetchall()

        # Convert the list of tuples to a set of user_ids
        if user_id_list:
            used_numbers = {user_id[0] for user_id in user_id_list}
        else:
            used_numbers = set()
```

```
        while True:
            num = random.randint(1000, 9999)
            if num not in used_numbers:
                return num

def validate_current_balance(current_balance):
    if not current_balance.isdigit():
        return False
    return True

def validate_ifsc_code(ifsc_code):
    if not re.match(r'^[A-Z]{4}0\d{6}$', ifsc_code):
        return False
    return True

def validate_account_number(account_number):
    if not account_number.isdigit():
        return False
    elif len(account_number) not in [8, 12]:
        return False
    return True

def validate_nominee_name(nominee_name):
    return all(char.isalpha() or char.isspace() for char in nominee_name)

def is_valid_bank(bank_name):
    return all(char.isalpha() or char.isspace() for char in bank_name)
```

10.4. Using the SQL query to **Display** the user information on the Dashboard page

```
WITH UserInfo AS (
    SELECT user_id, username, email, mobile_number, dob, registered_date
    FROM User
    WHERE user_id = {user_id}
  ),
  AccountInfo AS (
    SELECT COUNT(*) as num_accounts
    FROM Account
    WHERE user_id = {user_id}
  ),
  TransactionInfo AS (
    SELECT COUNT(*) as num_transactions
    FROM (
      SELECT income_id as trans_id FROM Income WHERE user_id = {user_id}
      UNION ALL
      SELECT expense_id as trans_id FROM Expenditure WHERE user_id = {user_id}
    ) t
  ),
  InvestmentInfo AS (
    SELECT COUNT(*) as num_investments
    FROM Investment
    WHERE user_id = {user_id}
  )
  SELECT CONCAT('User ID = ', user_id) as info
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('User Name = ', username)
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('E Mail = ', email)
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('Mobile Number = ', mobile_number)
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('DOB = ', dob)
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('Registered Date = ', registered_date)
  FROM UserInfo
  UNION ALL
  SELECT CONCAT('# of Accounts = ', num_accounts)
  FROM AccountInfo
  UNION ALL
  SELECT CONCAT('# of Transactions = ', num_transactions)
```

```
   FROM TransactionInfo
   UNION ALL
   SELECT CONCAT('# of Investments = ', num_investments)
   FROM InvestmentInfo;
```

10.5. Showing all the Accounts information of the user in the Table in GUI, we use the Treeview widget to display the data in the table and <u>Cursors</u> to execute the SQL query and fetch the results.

```
mydb = mysql.connector.connect(
   host='localhost',
   user='root',
   password='root',
   database='Personal_Finance_Management'
)

cursor = mydb.cursor()
user_id = myaccounts.read_loggedin_userid()

cursor.execute(f"SELECT account_id, bank_name, account_type, balance, open_date
FROM Account WHERE user_id = '{user_id}' ORDER BY open_date")

rows = cursor.fetchall()

for row in rows:
      self.tree.insert('', 'end', text=row[0], values=(row[0], row[1], row[2], row[3], row[4]),
tags=('larger'))

self.tree.tag_configure('larger', font=('Lexend Deca', 16), background='#195d68',
foreground='#ffffff')

mydb.commit()
mydb.close()
```

10.6. Handling the **Exceptional cases** in the Python Application

```
if editaccount.validate_nominee_name(nominee_name) == False:
    tk.messagebox.showerror("Error", "Invalid Nominee Name")
    return

if all ([old_account_nom, nominee_name]) and not all ([old_account_del, new_account]):
    result = editaccount.is_valid_old_account(old_account_nom, nominee_name)
    if result :
        tk.messagebox.showinfo("Success", "Nominee name changed successfully")
        clear_entry_fields()
        controller.show_frame("myaccounts")
    else :
        tk.messagebox.showerror("Error", "Invalid Account ID")
        clear_entry_fields()

elif all ([old_account_del, new_account]) and not all ([old_account_nom, nominee_name]):
    result = editaccount.is_valid_new_account(old_account_del, new_account)
    if result :
        tk.messagebox.showinfo("Success", "Account deleted successfully")
        clear_entry_fields()
        controller.show_frame("myaccounts")
    else :
        tk.messagebox.showerror("Error", "Invalid Account ID")
        clear_entry_fields()

else :
    tk.messagebox.showerror("Error", "Invalid Input")
    clear_entry_fields()
```

10.7. Using the Delete and Update SQL queries with Cursors to delete and update the Account information of the user. This is the Python alternate of Triggers in PL/SQL as here after the deletion of the Account, the corresponding Income and Expenditure records are deleted and the balance is updated in the Account table

```
try:
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="Personal_Finance_Management"
    )
    if connection.is_connected():
        cursor = connection.cursor()
        loggedin_userid = editaccount.read_loggedin_userid()
            query = f"SELECT balance FROM Account WHERE account_id = '{old_account}' 
and user_id = '{loggedin_userid}'"
        cursor.execute(query)
        row = cursor.fetchall()
             query0 = "SELECT account_id FROM Account WHERE account_id = %s and 
user_id = %s"
        cursor.execute(query0, (new_account, loggedin_userid))
        row0 = cursor.fetchall()

        if row and row0 is not None:
            q = f"delete from Income where account_id = '{old_account}'"
            cursor.execute(q)
            q1 = f"delete from Expenditure where account_id = '{old_account}'"
            cursor.execute(q1)
              q2 = f"Update Investment set account_id = '{new_account}' where account_id = 
'{old_account}'"
            cursor.execute(q2)
            query2 = f"delete from Account where account_id = '{old_account}'"
            cursor.execute(query2)
                  query3 = f"UPDATE Account SET balance = balance + '{row[0][0]}' WHERE 
account_id = '{new_account}'"
            cursor.execute(query3)
            connection.commit()
            return True
        else:
            return False

except Error as e:
    print(f"Error: {e}")
    return False

finally:
```

```
    if connection.is_connected():
        cursor.close()
        connection.close()
```

10.8. Similarly, we have also used the <u>Triggers to update</u> the balance of the Account after any transaction is made or any investment is made by buying or selling the stocks or other securities.

10.9. Using the <u>Yfinance scraping library</u> of Python to get the stock price of the company

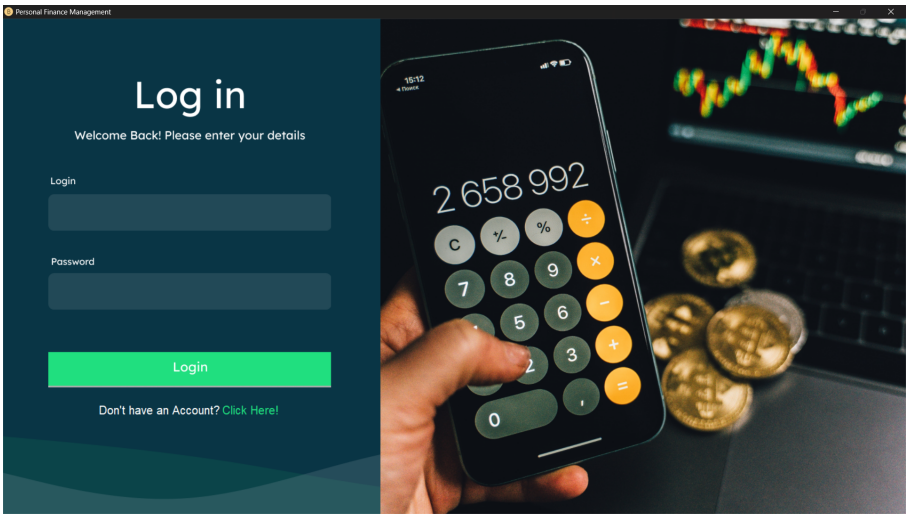
```
def get_stock_currency(ticker):
    stock = yf.Ticker(ticker)
    info = stock.info
    return info.get('currency')

def get_stock_price_on_date(ticker, date, days_range = 5):
    stock = yf.Ticker(ticker)
    date_obj = datetime.strptime(date, "%Y-%m-%d")
    start_date = (date_obj - timedelta(days=days_range)).strftime("%Y-%m-%d")
    end_date = (date_obj + timedelta(days=days_range)).strftime("%Y-%m-%d")
    data = stock.history(start=start_date, end=end_date)
    if not data.empty:
        timezone = data.index[0].tzinfo
        date_obj = date_obj.replace(tzinfo=timezone)
        closest_date = min(data.index, key=lambda x: abs(x - date_obj))
        price = data.loc[closest_date]['Close']
        return price
    else:
        print(f"No data available for {ticker} around {date}")
        return None

def get_current_price(symbol):
    stock_info = yf.Ticker(symbol)
    stock_data = stock_info.history(period="1d")
    return stock_data['Close'][0]
```

10.10. Creating the Graph using Matplotlab Library of Python to plot the Historical data of the Token

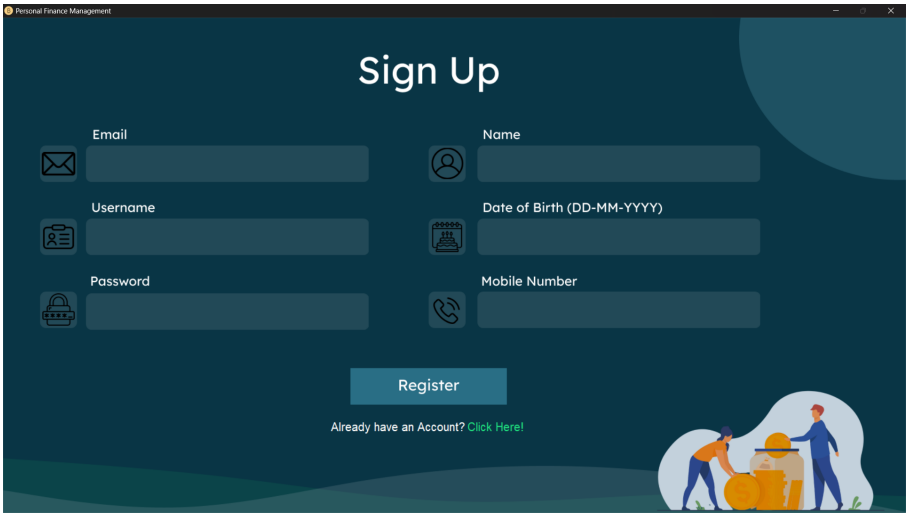
```
fig = plt.figure(figsize=(13.6, 7), dpi=100)
fig.patch.set_facecolor("#092f40")
fig.patch.set_facecolor('#092f40')
ax = fig.add_subplot(111)
ax.set_facecolor("#092f40")

ax.plot(history.index, history['Close'], color='#39FF14')

ax.set_title(f"{symbol} Historical Stock Data",
          color="#FFFF33", fontsize=16, pad=20)
ax.set_xlabel("Date", color="#FFFF33",
              fontsize=16, labelpad=20)
ax.tick_params(axis='x', colors="#87CEEB", labelsize=14)
ax.tick_params(axis='y', colors='#87CEEB', labelsize=14)

ax.spines['bottom'].set_color('#FFFF33')
ax.spines['top'].set_color('#FFFF33')
ax.spines['left'].set_color('#FFFF33')
ax.spines['right'].set_color('#FFFF33')

if buystocks.get_stock_currency(symbol) == 'USD':
   ax.set_ylabel("Price (in US Dollars)",
                 color="#FFFF33", fontsize=16, labelpad=20)
else:
   ax.set_ylabel("Price (in Indian Ruppees)",
                 color="#FFFF33", fontsize=16, labelpad=20)

canvas = FigureCanvasTkAgg(fig, master=temp_frame)
canvas.draw()
canvas.get_tk_widget().place(x=96, y=424)
```

11. Graphical User Interface

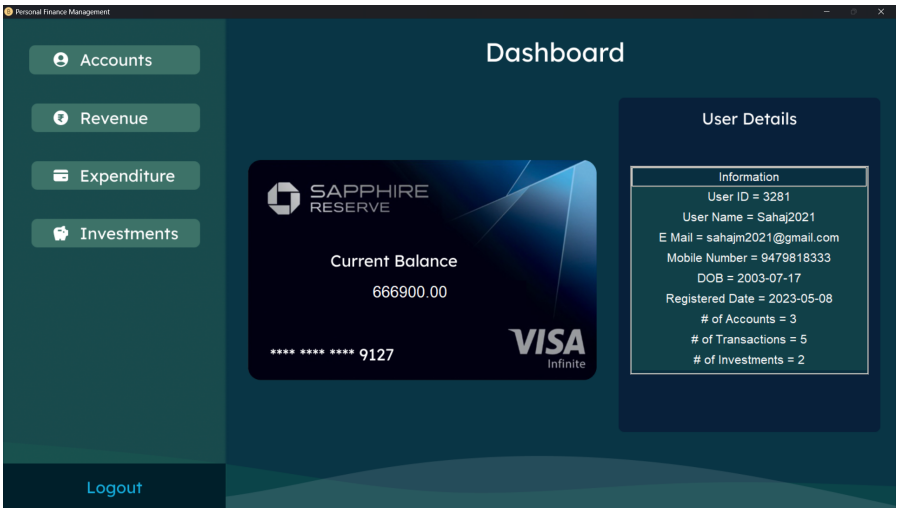
1. Login Page



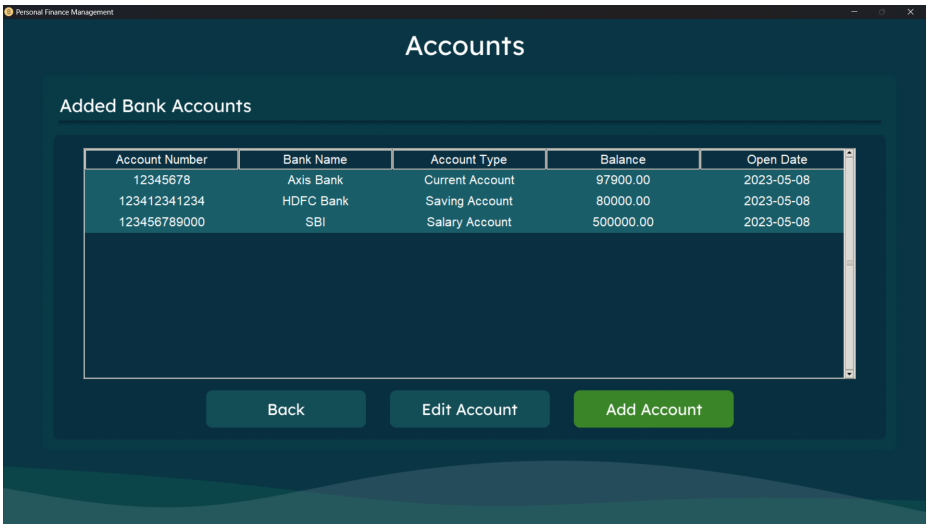
2. Registration Page



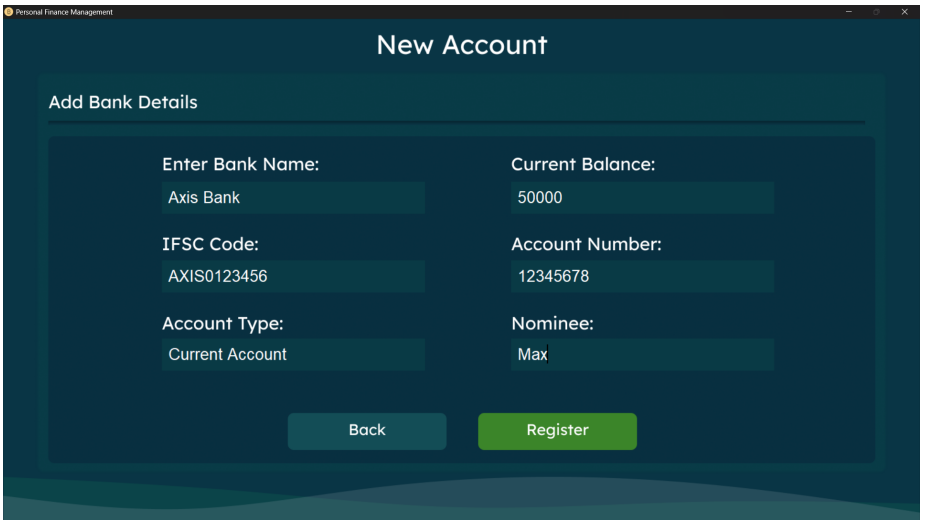
3. Dashboard



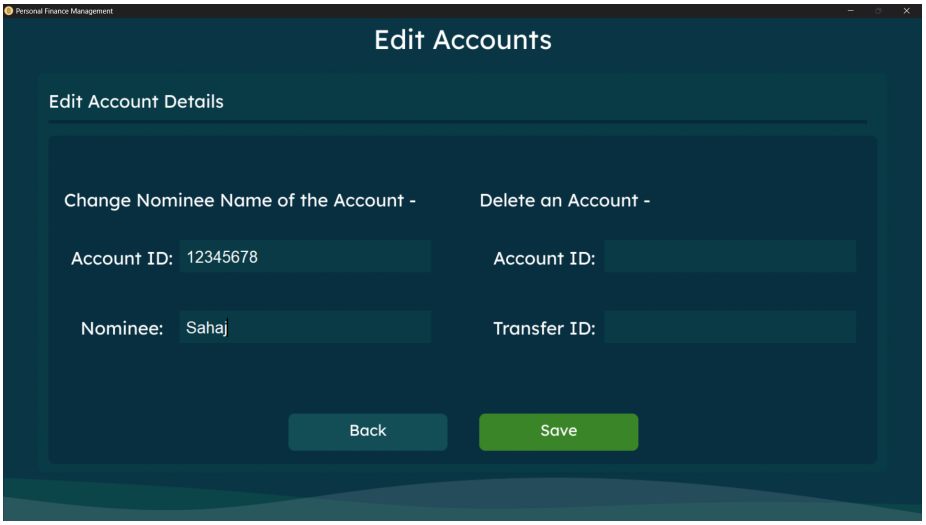
4. Accounts page



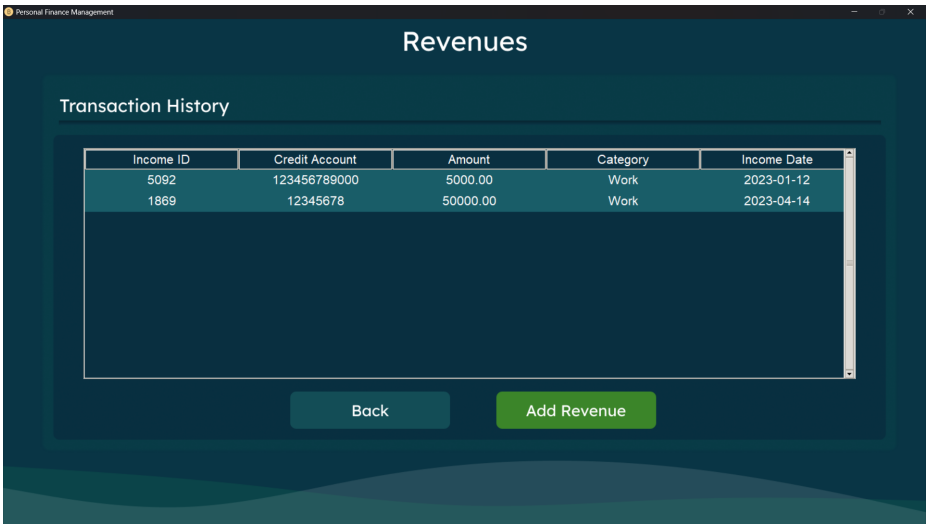
5. New Account Page



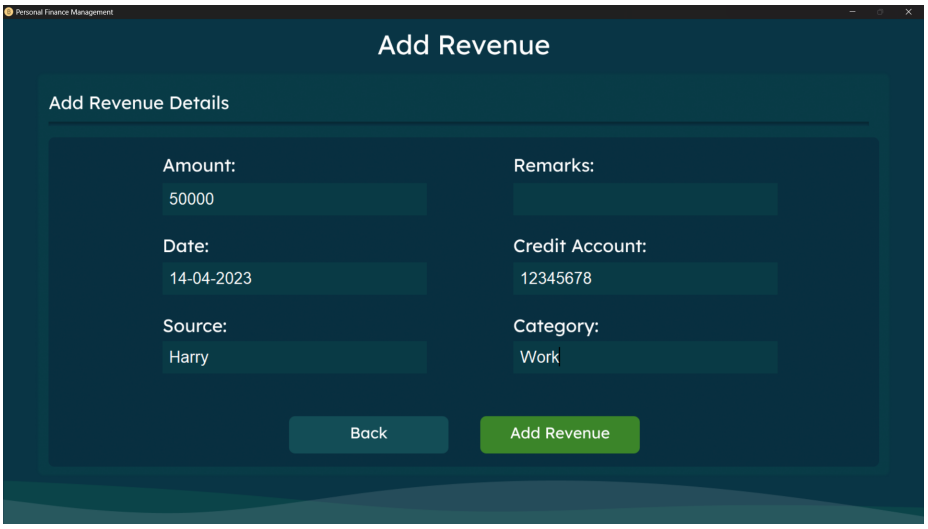
6. Edit Account Page



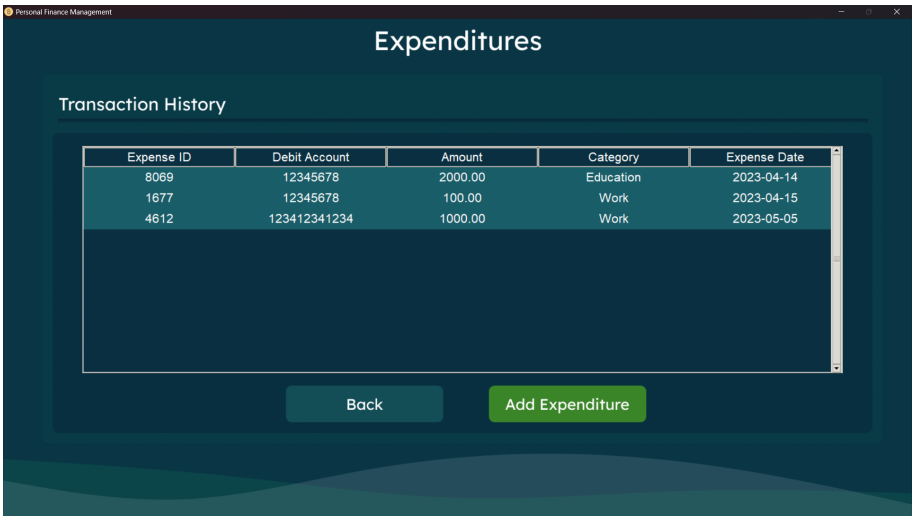
7. Revenue Page



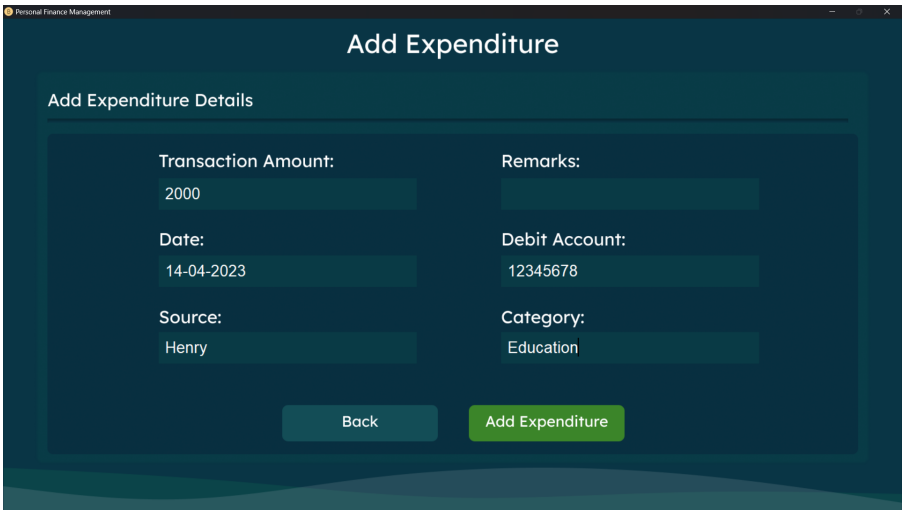
8. Add Revenue Page



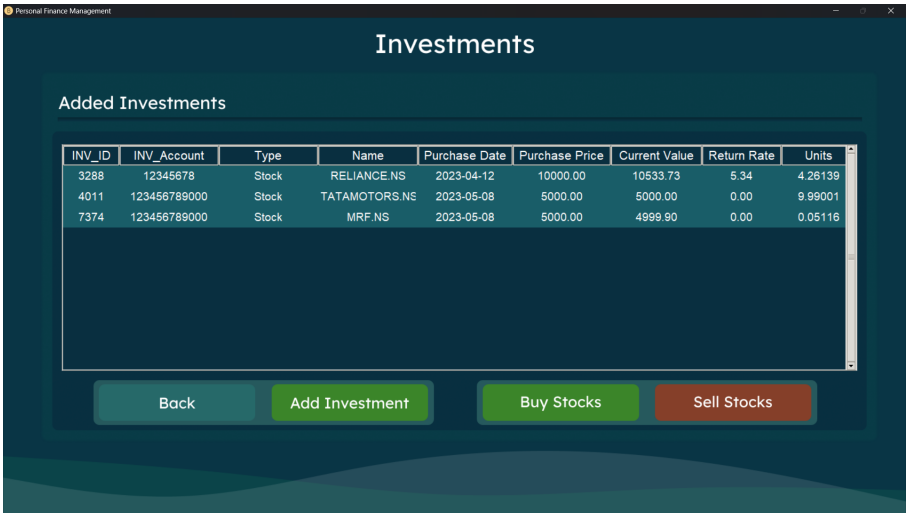
9. Expenditure Page



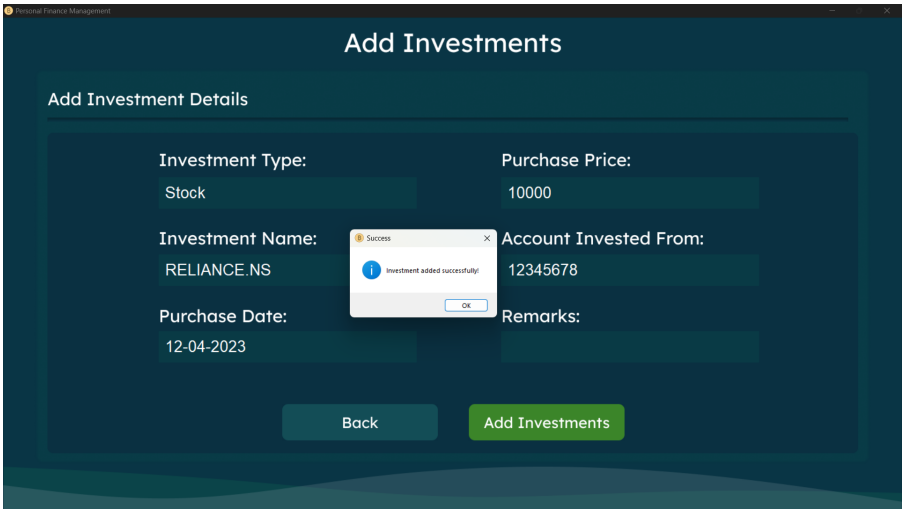
10. Add Expenditure Page



11. Investments Page

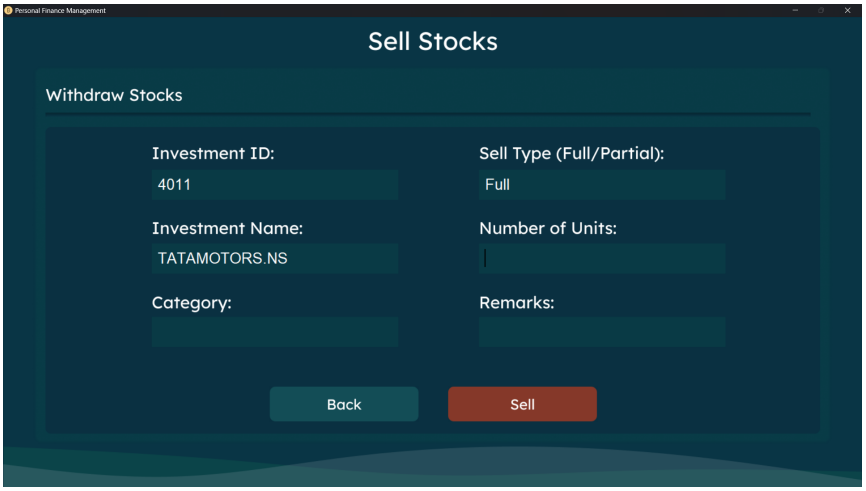


12. Add Investment

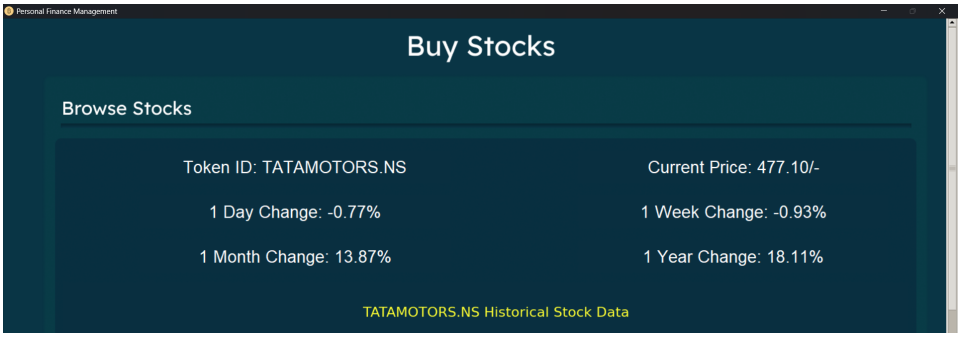


13. Current Stock Rates Page

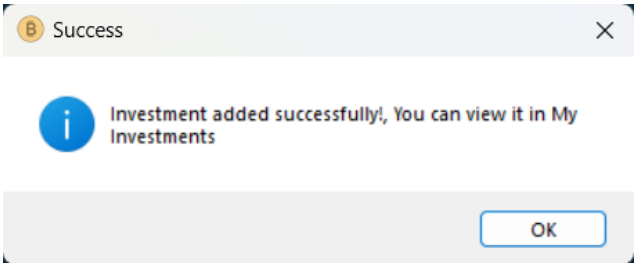
14. Sell Stocks Page



15. Buy Stocks Page



16. Modals



17. Database

Database: personal_finance_management Table: user

	user_id	username	password	name	registered_date	dob	mobile_number	email
1	3281	Sahaj2021	Sahaj@2021	Sahaj Mishra	2023-05-08	2003-07-17	9479818333	sahajm2021@gmail.com
2	5227	Prats2003	Prats@2003	Pratyush Choudha...	2023-05-09	2003-02-14	5214698523	pratyush31@gmail.com
3	6040	Sahaj2022	Sahaj@2022	Sahaj Mishra	2023-05-09	2003-07-17	9479818333	sahajm2022@gmail.com
4	6237	Manan31	Manan@31	Manan Makhija	2023-05-09	2003-05-04	9479818333	mananmakhija31@gmail.co...
5	7174	Gaurav31	Gauraav@31	Gaurav Chowdhary	2023-05-09	2003-05-14	1234567891	cgaurav31@gmail.com

Database: personal_finance_management Table: account

	account_id	bank_name	user_id	ifsc_code	account_type	balance	nominee_name	interest_rate	open_date
1	11111111	Axis Bank	6040	AXIS0666666	Saving Account	49518000....	Jane	3.50	2023-05-09
2	12345678	Axis Bank	3281	AXIS0123456	Current Accou...	87900.00	Sahaj	0.00	2023-05-08
3	55555555	HDFC Bank	7174	HDFC0123456	Saving Account	2000.00	Manan	3.50	2023-05-09
4	77777777	Axis Bank	6237	AXIS0123456	Salary Account	960000.00	Gaurav	3.00	2023-05-09
5	123123123123	Axis Bank	5227	AXIS0123457	Saving Account	10000.00	Sahaj	3.50	2023-05-09
6	123412341234	HDFC Bank	3281	HDFC0123456	Saving Account	79000.00	Tom	3.50	2023-05-08
7	123456789000	SBI	3281	SBIN0123456	Salary Account	500000.00	Jerry	3.00	2023-05-08
8	444444444444	PNB	6040	PNBI0999999	Current Accou...	3999900.00	Tom	0.00	2023-05-09

Database: personal_finance_management Table: income

	income_id	user_id	account_id	amount	income_date	source	remarks	category
1	1869	3281	12345678	50000.00	2023-04-14	Harry	None	Work
2	3320	6040	11111111	20000.00	2003-04-14	Tommy	None	Work
3	4350	6040	444444444444	100000.00	2023-04-08	Sahaj	None	Work
4	4932	6237	77777777	10000.00	2023-04-14	Sahaj	None	Work
5	5092	3281	123456789000	5000.00	2023-01-12	Boss	Extra	Work

Database: personal_finance_management Table: expenditure

	expense_id	user_id	account_id	expense	category	expense_date	source	remarks
1	1677	3281	12345678	100.00	Work	2023-04-15	Jason	None
2	2222	6040	444444444444	100.00	Fun	2023-04-04	Pratyush	None
3	3273	6040	11111111	2000.00	Food	2023-02-10	Manan	None
4	4612	3281	123412341234	1000.00	Work	2023-05-05	Shop	None
5	6422	6237	77777777	50000.00	Food	2023-04-15	Romeo	None
6	8069	3281	12345678	2000.00	Education	2023-04-14	Henry	None

Database: personal_finance_management Table: investment

	investment_id	user_id	account_id	investment_type	investment_name	purchase_date	purchase_price	current_value	return_rate	number_of_units	remarks
1	2159	6040	444444444444	Stock	AAPL	2023-05-09	100000.00	99999.95	0.00	7.05038	None
2	2938	6040	11111111	Stock	MSFT	2023-05-09	500000.00	500000.03	0.00	19.81600	None
3	2995	7174	55555555	Index Fund	^BSESN	2023-03-14	8000.00	8533.97	6.67	0.13817	None
4	3288	3281	12345678	Stock	RELIANCE.NS	2023-04-12	10000.00	10533.73	5.34	4.26139	None
5	7374	3281	123456789000	Stock	MRF.NS	2023-05-08	5000.00	4999.90	0.00	0.05116	None
6	9284	5227	123123123123	Crypto Currency	DOGE-USD	2023-04-14	10000.00	8106.07	-18.94	1377.75704	None

12. Conclusion

During the course of this project, we learnt a lot of the work and best practices that go into creating a database, the rules to construct a good ER diagram, How to come up with relational schema mapping from the ER diagram, deriving the functional dependencies and how to normalise the relational schema. We learnt how to design a system from Database perspective and how to efficiently store and manipulate data.

13. References

- https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-fetchall.html
- https://www.oracletutorial.com/python-oracle/querying-data-using-fetchone-fetchmany-and-fetchall-methods/
- https://pynative.com/python-cursor-fetchall-fetchmany-fetchone-to-read-rows-from-table/
- https://www.guru99.com/triggers-pl-sql.html
- https://www.youtube.com/watch?v=_auZ8TTkojQ
- https://www.youtube.com/watch?v=0WafQCaok6g
- https://vegibit.com/python-yfinance-tutorial/#:~:text=Yfinance%20is%20a%20Python%20library,company%20name%20and%20stock%20exchange