Python Programming
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
Prof. Poonam Bhagat

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

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.

Prof. Poonam Bhagat
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

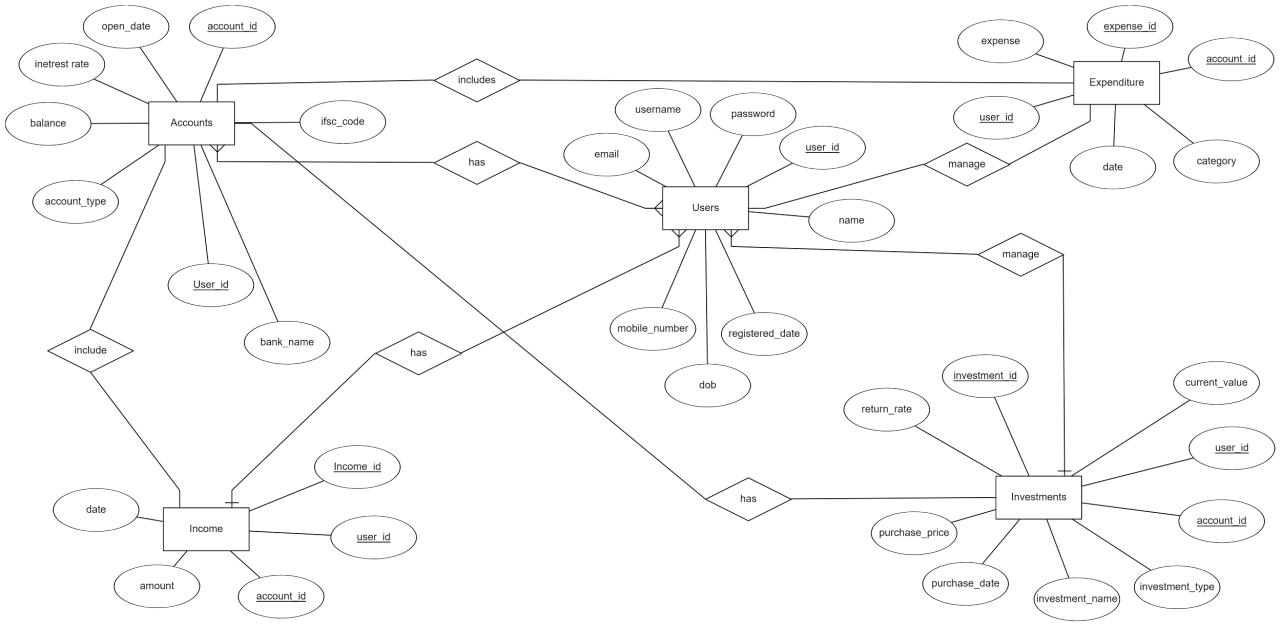
As technology continues to evolve, project finance management systems are likely to integrate with emerging technologies such as artificial intelligence (AI), blockchain, and the Internet of Things (IoT). These technologies can enhance the system's capabilities, providing real-time data analysis, and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

3. Problem Statement

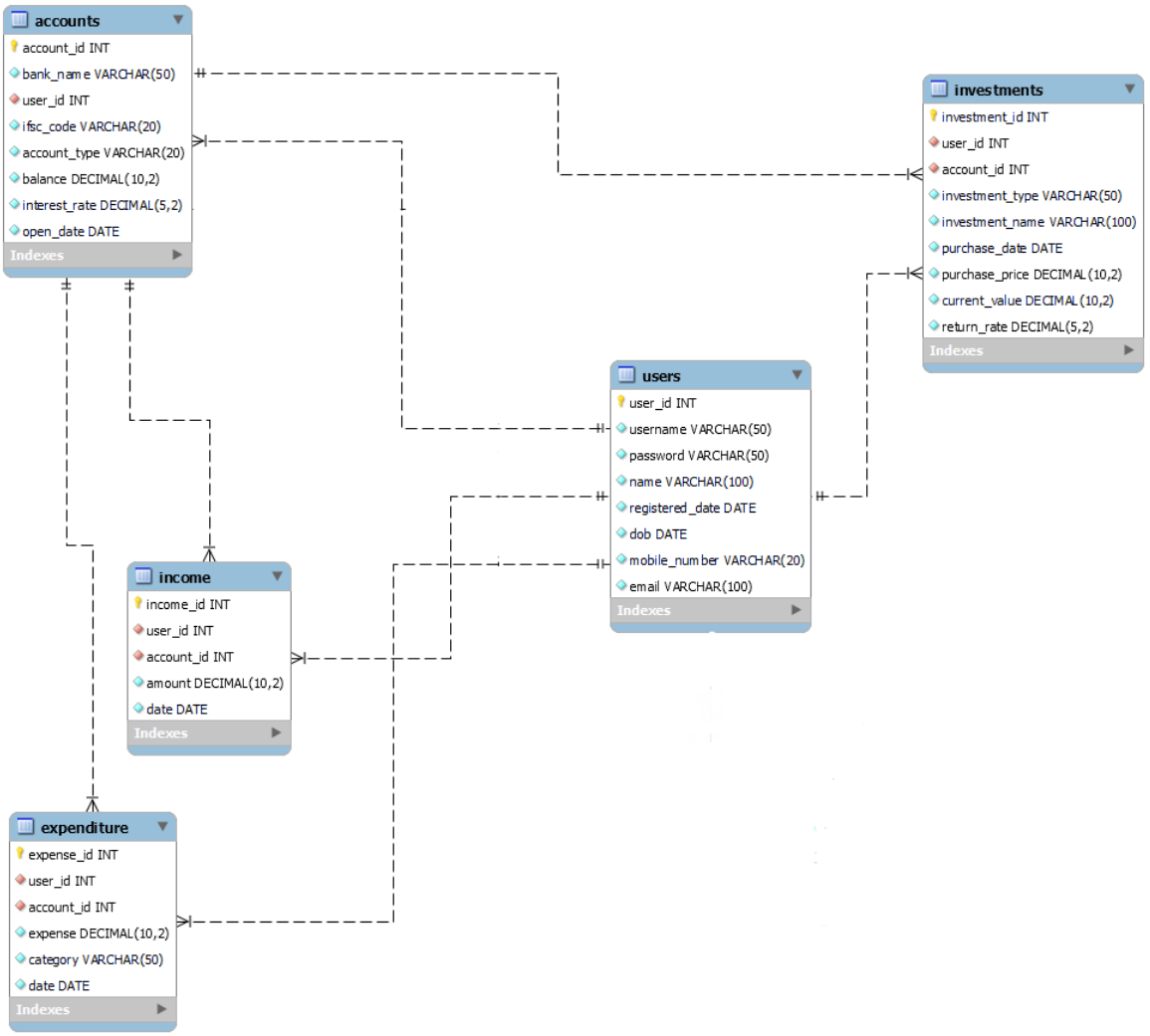
Many people struggle to effectively manage their personal finances, leading to financial stress, overspending, and poor financial decisions. To address this problem, we propose the development of a personal finance management app that helps users track their income, expenses, and overall financial health.

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. ER Diagram



5. Schematic diagram (ER Diagrams into tables)



6. Dataset Description and Data Pre-processing (Yfinance)

The yfinance library is a popular Python package that provides a convenient way to download historical market data from Yahoo Finance. It allows you to fetch stock prices, options data, and other financial information.

Here's a basic example of how to use the yfinance library to download historical stock data:

6.1. Install the yfinance library using pip :

pip install yfinance

6.2. Import the library in your Python script :

import yfinance as yf

6.3. Use the yf.download function to retrieve historical data for a specific stock:

```
# Download data for Apple (ticker symbol: AAPL)
data = yf.download('AAPL', start='2022-01-01', end='2022-12-31')
```

In the above example, we specify the ticker symbol 'AAPL' for Apple and the start and end dates for the desired data range.

6.4. You can then work with the downloaded data, such as accessing specific columns or performing calculations :

```
# Print the first few rows of the data
print(data.head())

# Access the 'Close' column
close_prices = data['Close']

# Perform calculations on the data
max_price = close_prices.max()
min_price = close_prices.min()
```

The yf.download function returns a Pandas DataFrame containing the historical data for the specified stock. You can access specific columns (e.g., 'Open', 'Close', 'Volume') or perform various operations on the data using the capabilities of Pandas.

7. Tasks Performed and Tools/Library Used

7.1. Creating the Tree View Widget and scrollbar in Tkinter library of python to Display the table of 8 columns from MySQL :

```
self.tree = ttk.Treeview(self, columns=(
    'INV_ID', 'INV_Account', 'Type', 'Name', 'Purchase Date', 'Purchase Price', 'Current Value', 'Return Rate', 'Units'), show='headings')

self.tree.place(x=100, y=215, width=1353, height=384)

style = ttk.Style()
style.theme_use("clam")

style.configure('Treeview', rowheight=35, fieldbackground="#092f40")

style.configure('Treeview.Heading', font=('Lexend Deca', 16),
            foreground='white', background="#092f40")

self.tree.column('INV_ID', width=95,
                anchor="center", minwidth=95)

self.tree.heading('INV_ID', text='INV_ID')

self.tree.column('INV_Account', width=168, anchor="center", minwidth=168)

self.tree.heading('INV_Account', text='INV_Account')

self.tree.column('Type', width=168,
                anchor="center", minwidth=168)

self.tree.heading('Type', text='Type')

self.tree.column('Name', width=168, anchor="center", minwidth=168)

self.tree.heading('Name', text='Name')

self.tree.column('Purchase Date', width=159, anchor="center", minwidth=159)
```

```
self.tree.heading('Purchase Date', text='Purchase Date')

self.tree.column('Purchase Price', width=165,
                 anchor="center", minwidth=165)

self.tree.heading('Purchase Price', text='Purchase Price')

self.tree.column('Current Value', width=154,
                 anchor="center", minwidth=154)

self.tree.heading('Current Value', text='Current Value')

self.tree.column('Return Rate', width=134,
                 anchor="center", minwidth=134)

self.tree.heading('Return Rate', text='Return Rate')

self.tree.column('Units', width=126,
                 anchor="center", minwidth=126)

self.tree.heading('Units', text='Units')

scrollbar = ttk.Scrollbar(
   self, orient='vertical', command=self.tree.yview)

scrollbar.place(x=1433, y=216, height=382, width=20)

self.tree.configure(yscrollcommand=scrollbar.set)

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

cursor = mydb.cursor()
```

```
user_id = myinvestments.read_loggedin_userid()

cursor.execute(
   f"select investment_name from Investment where user_id = '{user_id}'")
rows_temp = cursor.fetchall()

for row in rows_temp:

   stock_info = yf.Ticker(row[0])
   stock_data = stock_info.history(period="1d")
   new_value = stock_data['Close'][0]

   if myinvestments.get_stock_currency(row[0]) == 'USD':
      new_value = new_value * 81.75

   cursor.execute(
            'UPDATE Investment SET current_value = %s * number_of_units,
return_rate = (((%s * number_of_units) - purchase_price) / purchase_price) *
100 WHERE investment_name = %s',
      (float(new_value), float(new_value), row[0])
   )
cursor.execute(
    f"SELECT investment_id, account_id, investment_type, investment_name,
purchase_date, purchase_price, current_value, return_rate, number_of_units
FROM Investment WHERE user_id = '{user_id}' ORDER BY purchase_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], row[5], row[6],
row[7], row[8]),
                tags=('larger'))

self.tree.tag_configure('larger', font=(
   'Lexend Deca', 14), background='#195d68', foreground='#ffffff')

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

7.2. Creating a Container in Tkinter to store all the frames in a application window

```
import tkinter as tk
class App(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}
                for F in (login, signup, dashboard, myaccounts, addaccount, editaccount, myrevenues, addrevenue, myexpenses, addexpense, myinvestments, addinvestment, buystocks, sellstocks):
            page_name = F.__name__
            frame = F(parent = container, controller = self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")

    self.show_frame("login")
    def show_frame(self, page_name):
            frame = self.frames[page_name]
            if page_name == "myaccounts":
                frame.update_accounts()
            if page_name == "myexpenses":
                frame.update_expenses()
            if page_name == "myrevenues":
                frame.update_revenues()
            if page_name == "myinvestments":
                frame.update_investments()
            if page_name == "dashboard":
                frame.update_dashboard()
            frame.tkraise()
if __name__ == "__main__":
    Application = App()
    Application.mainloop()
```

7.3. We have used PyMySQL library and MySQL Connector library to connect the Python Application to our MySQL Database.

```
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()
```

7.4. Python Functions to check whether the user input data is correct or not.

```
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)
```

7.6. 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()
```

7.7. Try, Catch and Finally Block to handle the exceptions while connecting to the MySQL Database.

```
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()
```

7.8. Using the Yfinance scraping library 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]
```

7.9. <u>Creating the Graph</u> using Matplotlab Library of Python to plot the Historical data of the Token which is obtained using the yfinance dataset.

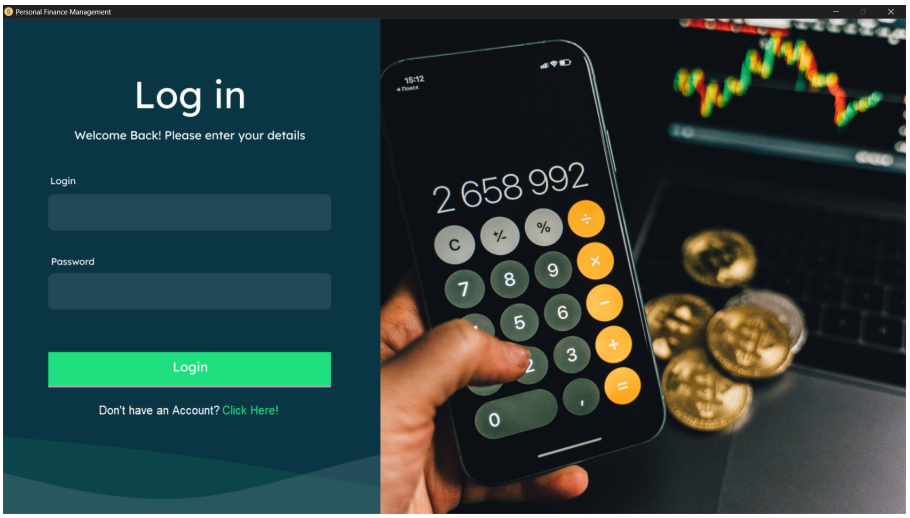
```
try:

    if entry0.get() == "":
        messagebox.showerror("Error", "Please enter a stock symbol.")
        return

    symbol = entry0.get().upper()

    # Retrieve the historical stock data for the last year
    end_date = datetime.now().strftime('%Y-%m-%d')
    start_date = (datetime.now() - timedelta(days=365)).strftime('%Y-%m-%d')
    history = yf.download(symbol, start=start_date, end=end_date)

     # Calculate the growth percentages for one day, one week, one month, and one
year
    price_today = history['Close'][-1]
    price_1d_ago = history['Close'][-2]
    price_1w_ago = history['Close'][-6]
    price_1m_ago = history['Close'][-31]
    price_1y_ago = history['Close'][0]

    if buystocks.get_stock_currency(symbol) == 'USD':
        price_today = price_today * 81.75
        price_1d_ago = price_1d_ago * 81.75
        price_1w_ago = price_1w_ago * 81.75
        price_1m_ago = price_1m_ago * 81.75
        price_1y_ago = price_1y_ago * 81.75

    growth_1d = (price_today - price_1d_ago) / price_1d_ago * 100
    growth_1w = (price_today - price_1w_ago) / price_1w_ago * 100
    growth_1m = (price_today - price_1m_ago) / price_1m_ago * 100
    growth_1y = (price_today - price_1y_ago) / price_1y_ago * 100

    entry2_font = ("Lexend Deca", 20)

          global growth_1d_label, growth_1w_label, growth_1m_label, growth_1y_label,
name_label, price_label, fig, ax, canvas

    name_label = tk.Label(temp_frame, text=f"Token ID: {symbol}", bg="#092f40",
                          highlightthickness=0,
                          font=entry2_font,
                          fg="white")
           price_label = tk.Label(temp_frame, text=f"Current Price: {price_today:.2f}/-",
bg="#092f40",
                          highlightthickness=0,
```

```
                font=entry2_font,
                fg="white")
    growth_1d_label = tk.Label(temp_frame, text=f"1 Day Change: {growth_1d:.2f}%",
bg="#092f40",
                highlightthickness=0,
                font=entry2_font,
                fg="white")
    growth_1w_label = tk.Label(temp_frame, text=f"1 Week Change: {growth_1w:.2f}%",
bg="#092f40",
                highlightthickness=0,
                font=entry2_font,
                fg="white")
    growth_1m_label = tk.Label(temp_frame, text=f"1 Month Change: {growth_1m:.2f}%",
bg="#092f40",
                highlightthickness=0,
                font=entry2_font,
                fg="white")
    growth_1y_label = tk.Label(temp_frame, text=f"1 Year Change: {growth_1y:.2f}%",
bg="#092f40",
                highlightthickness=0,
                font=entry2_font,
                fg="white")

  name_label.place(x=220, y=212, width=500, height=54)
  price_label.place(x=925, y=212, width=500, height=54)
  growth_1d_label.place(x=220, y=284, width=500, height=54)
  growth_1w_label.place(x=925, y=284, width=500, height=54)
  growth_1m_label.place(x=220, y=356, width=500, height=54)
  growth_1y_label.place(x=925, y=356, width=500, height=54)

  # Create the matplotlib figure
  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")

  # Plot the historical stock data
  ax.plot(history.index, history['Close'], color='#39FF14')

  # Set the plot title and axis labels
  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')
```

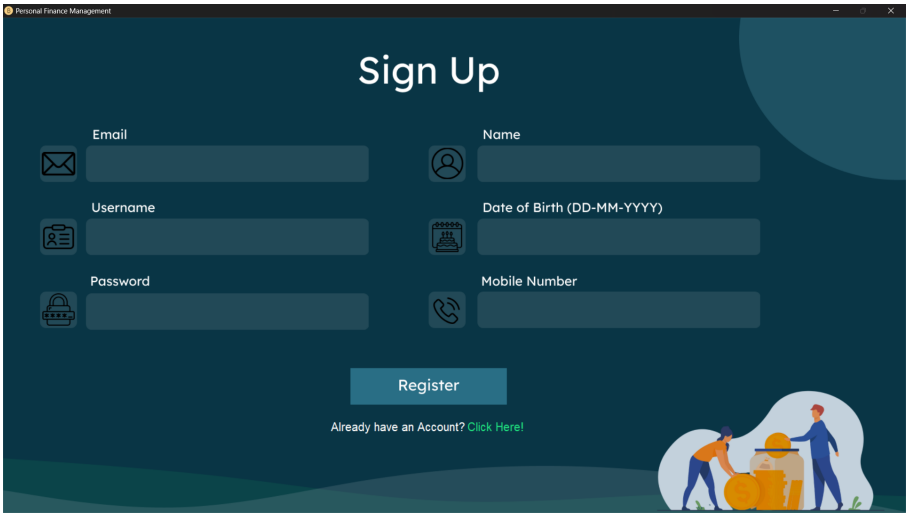
```
    ax.spines['top'].set_color('#FFFF33')
    ax.spines['left'].set_color('#FFFF33')
    ax.spines['right'].set_color('#FFFF33')

    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)

    # Create the canvas to embed the plot in the GUI window
    canvas = FigureCanvasTkAgg(fig, master=temp_frame)
    canvas.draw()
    canvas.get_tk_widget().place(x=96, y=424)


except Exception as e:
    messagebox.showerror("Error", "Invalid Stock Symbol")
    print(f"Error: {e}")
```

8. Output and Visualisation screenshots (GUI)

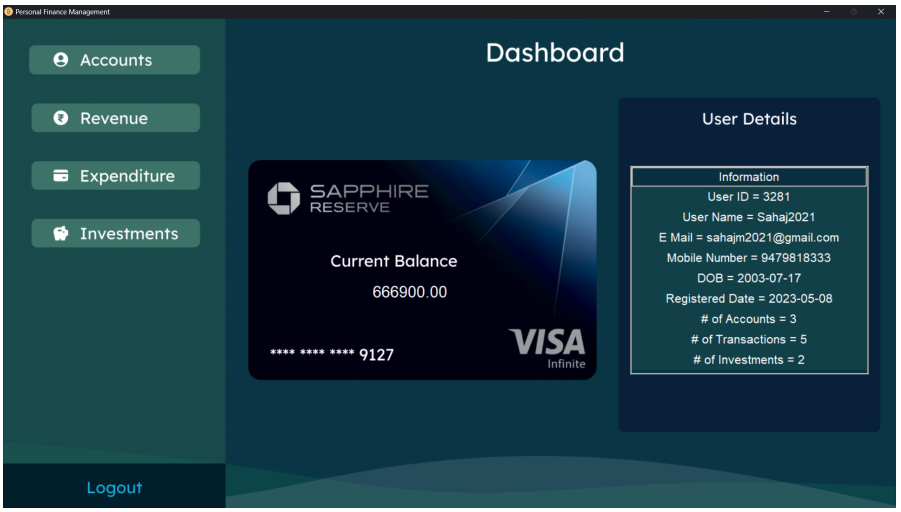
1. Login Page



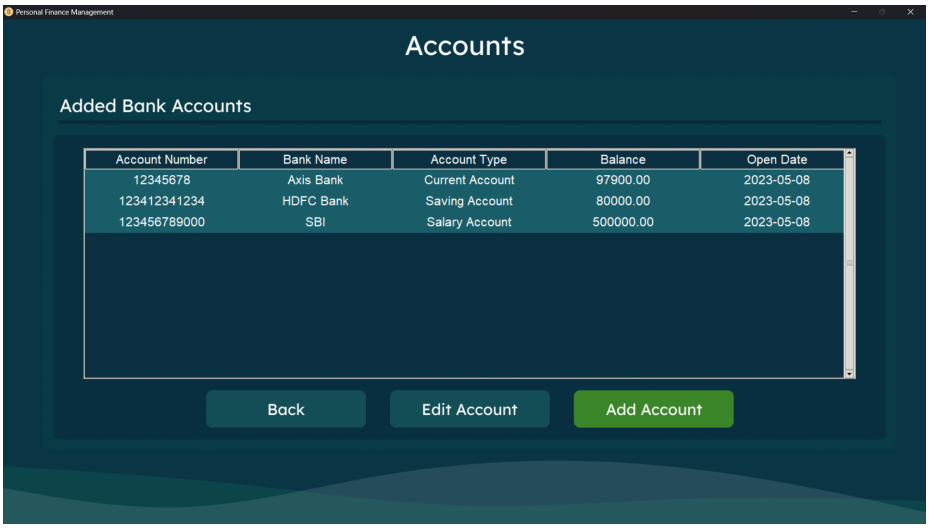
2. Registration Page



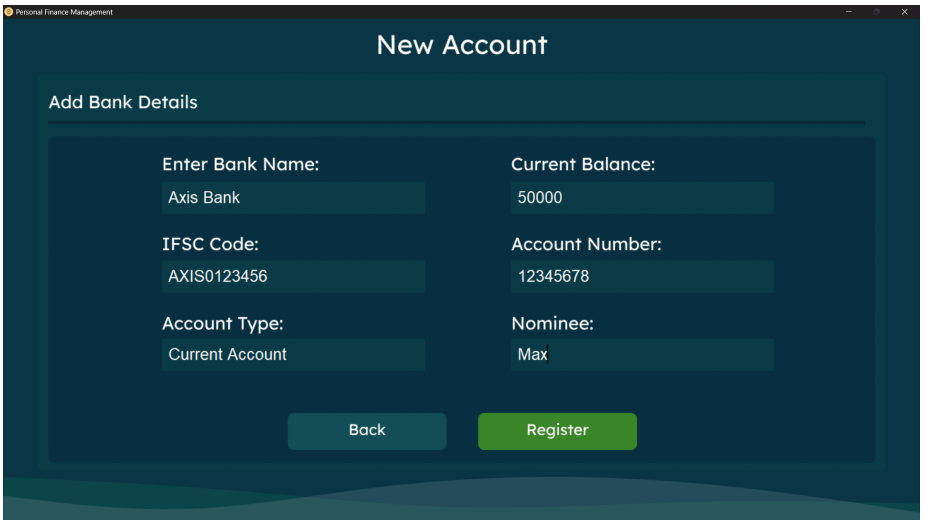
3. Dashboard



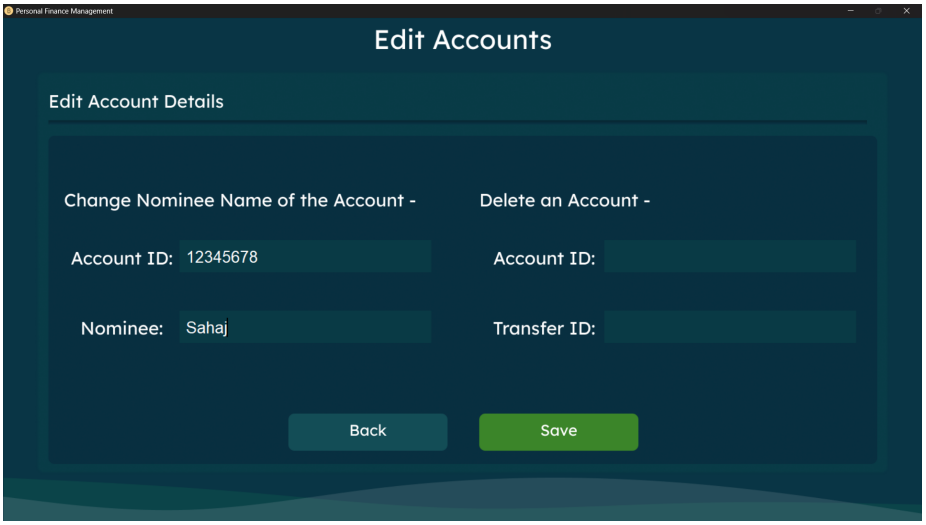
4. Accounts page



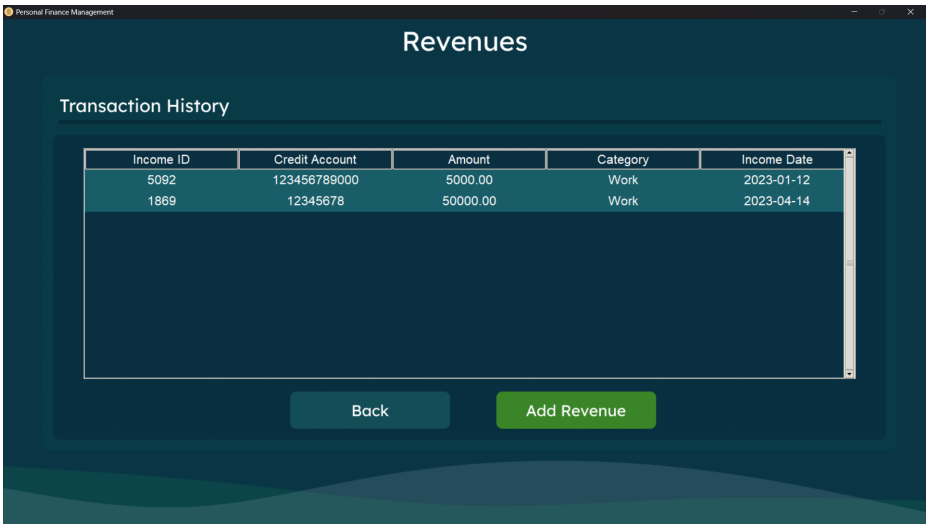
5. New Account Page



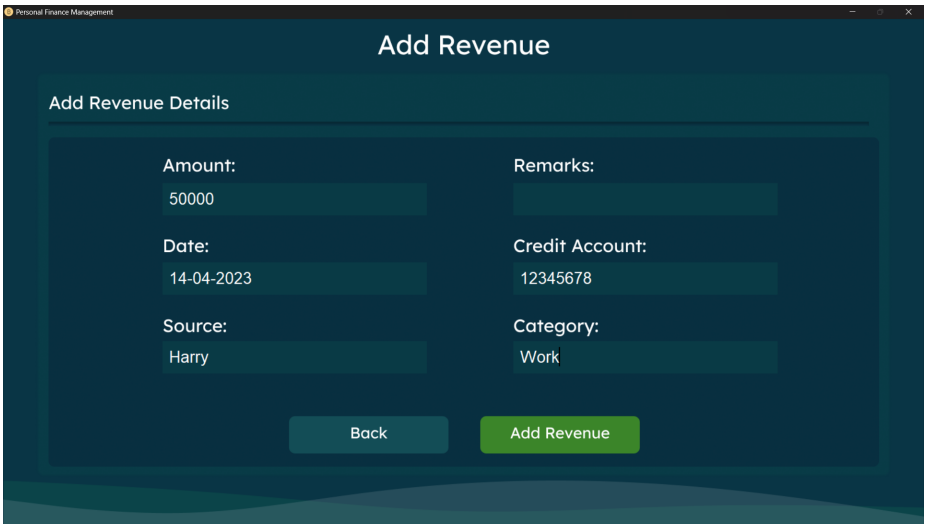
6. Edit Account Page



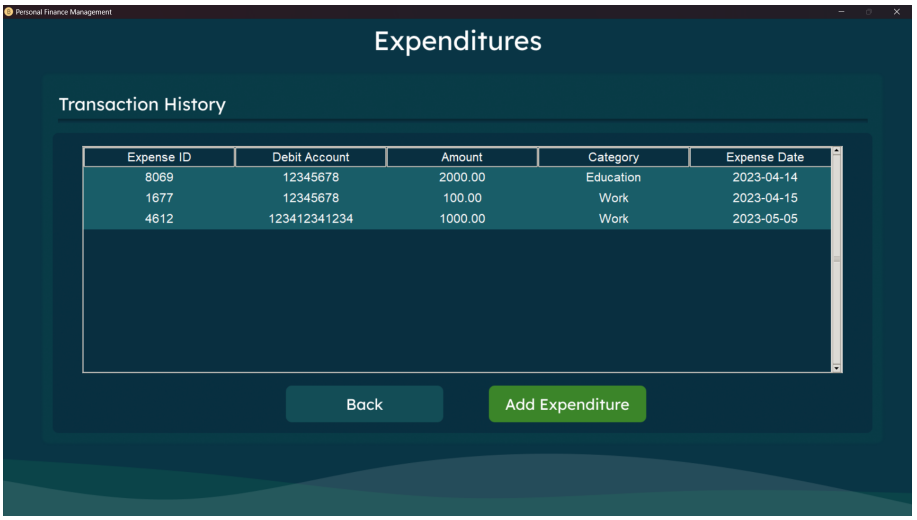
7. Revenue Page



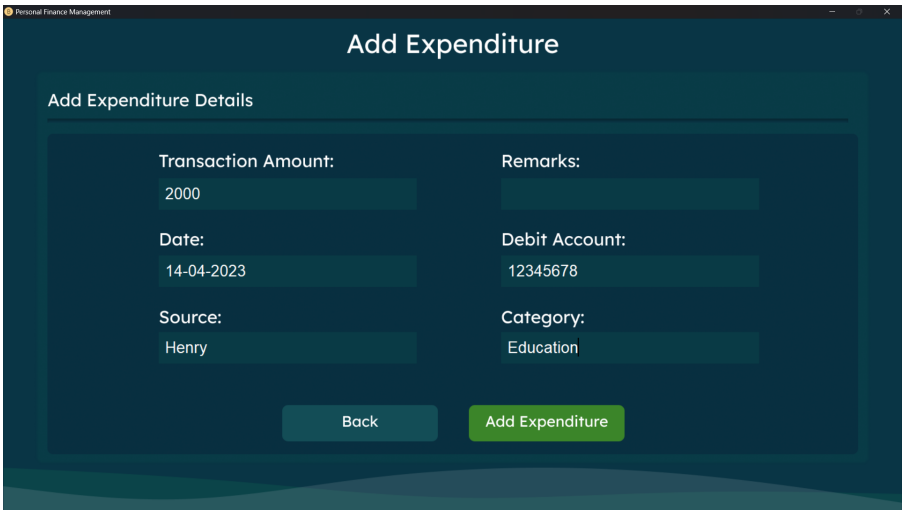
8. Add Revenue Page



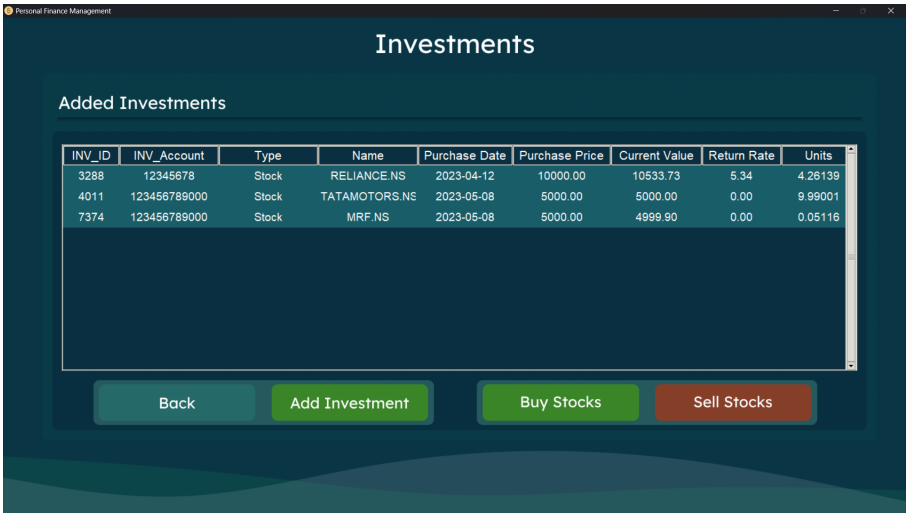
9. Expenditure Page



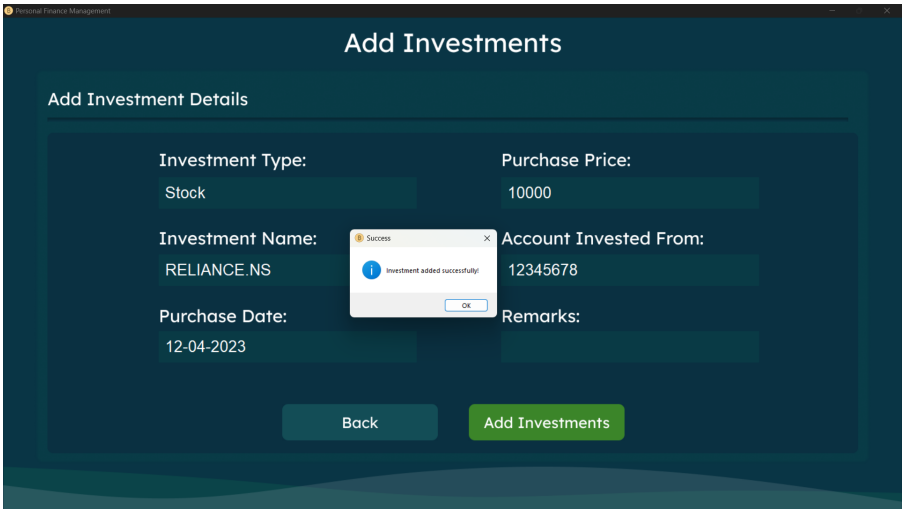
10. Add Expenditure Page



11. Investments Page

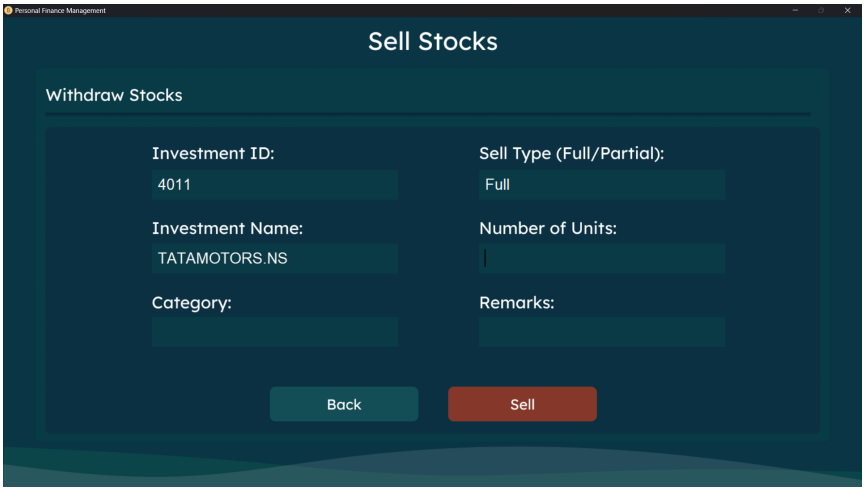


12. Add Investment

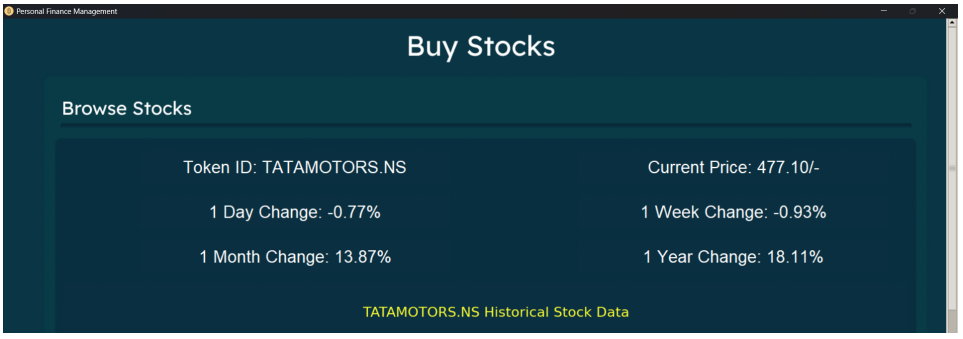


13. Current Stock Rates Page

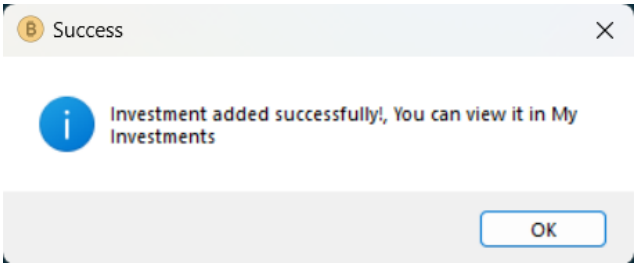
14. Sell Stocks Page



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

9. 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 a Database perspective and how to efficiently store and manipulate data.

10. 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://www.scaler.com/topics/module-and-package-in-python/
- https://pythonprogramming.net/change-show-new-frame-tkinter/
- https://stackoverflow.com/questions/41918796/why-frames-not-raising-with-tkinter-tkraise
- https://pythonbasics.org/tkinter-frame/
- https://vegibit.com/python-yfinance-tutorial/#:~:text=Yfinance%20is%20a%20Python%20library,company%20name%20and%20stock%20exchange