DOKUZ EYLUL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2210

Object Oriented Analysis and Design

MOCRYPTO

By

Ege Kemal GERMEN

Güney SÖĞÜT

İsmail Can ÇANAKÇI

CHAPTER ONE

1.1 SUBJECT OF PROJECT

The subject of the project is developing Mock Cryptocurrency Trading App

1.2 PURPOSE OF PROJECT

Our aim is to create an application that lets you track trends of real-life cryptocurrencies such as Bitcoin, Ethereum, Solana, Chiliz and many others; and have the ability to experiment on trading without any real-world financial risk.

Our application will provide a chance for beginners to practice buying and selling cryptocurrencies by using imaginary currency. We believe that using real-life data encourages beginners to step into the world of Cryptocurrency Trading to learn and experiment with real investment strategies.

1.3 SCOPE OF PROJECT

Our app will use a public API (CoinMarketCap or Binance) to get real-time cryptocurrency data. We will be using Java and a MySQL database to store all user transactions. Public and private keys representing recipient and sender wallets are arbitrarily generated.

Users will be able to:

- Pick cryptocurrencies to track trends in REAL TIME.
- Buy and sell cryptocurrencies with fictional money (Every new account is created with 10.000$ to spare).
- Track profits and losses through portfolio tab
- Navigate through the application easily, with a clean and intuitive UI.

CHAPTER TWO

Our application will use an API to get real-time cryptocurrency data. The user can see the currencies' current real-life market value and buy or sell the cryptocurrency of choice with fictional money. Moreover, users can navigate through their portfolio, cryptocurrencies, and transaction history; as well as use features such as margins. We will be using a MySQL database to store different user profiles and their transaction details.

To achieve this, we will need various requirements which are described below.

REQUIREMENTS

2.1 Classes:

LoginPage

- **Attributes:**
  - GUI COMPONENTS
- **Methods:**
  - **display()** -> return None: Displays login page
  - **fetchAccount(String username,String pass)** -> return Account: Returns account from database; of whom the information is given as input
  - **initializeDatabase()** -> return boolean: Used for creating necessary tables in database.

AdminPage

- **Attributes:**
  - GUI COMPONENTS
  - private CryptocurrencyAPI
  - private DefaultTableModel model_allcrypto_list;
  - private DefaultTableModel model_systemcrypto_list;
  - private Object[] row_allcryptocurrency_list;
  - private Object[] row_systemcryptocurrency_list;
  - private int selectedCryptocurrencyIndex;
  - private Cryptocurrency selectedCryptocurrency;

    - private ArrayList<Cryptocurrency> systemCryptocurrencyList = getSystemCryptocurrencies();
    - private ArrayList<Cryptocurrency> allCryptocurrencyList = cryptocurrencyAPI.getCryptocurrencyList();
- **Methods:**
    - **display()** -> return None: Displays login page
    - **add(String uuid,String name,String shortname, double price, double volume)** -> Returns boolean: Adds cryptocurrency with given attributes to the database.
    - **delete(String id)** -> Returns boolean: Removes cryptocurrency with the given uuid from the database.
    - **loadAllCryptocurrencyModel(ArrayList<Cryptocurrency> cryptocurrencyList)** -> Returns None: Loads ALL cryptocurrencies read from API into database.
    - **loadSystemCryptocurrencyModel(ArrayList<Cryptocurrency> systemCryptocurrencyList)** -> Returns None: Loads cryptocurrencies specified in the parameter ArrayList (systemCryptocurrencyList) into database.

MainPage

- **Attributes:**
    - GUI COMPONENTS
    - private User currentUser;
    - private CryptocurrencyAPI cryptocurrencyAPI = new CryptocurrencyAPI();
    - private ArrayList<Cryptocurrency> cryptocurrencyList = new ArrayList<>();
- **Methods:**
    - **display()** -> return None: Displays main page
    - public static boolean addDatabase(User user) -> return boolean: Function to store user portfolio in the database
    - **loadModel()** functions -> return boolean: Functions to update GUI using data from the database

CryptocurrencyAPI

- **Methods:**
    - **getCryptocurrencyList()** -> return ArrayList<Cryptocurrency>: Function to fetch list of cryptocurrencies available through API
    - **getExchangeRate(Cryptocurrency fromCoin, Cryptocurrency toCoin)** -> return double: Function to calculate exchange rates between two given currencies

Account

- **Attributes:**
    - private int id;
    - private String name;
    - private String surname;
    - private String username;
    - private String password;
    - private String type;
    - private Portfolio portfolio;

    - *double balance; (for User subclass)*
- **Methods:**
    - **Getters & setters**

Cryptocurrency

- **Attributes:**
    - private final String uuid;
    - private String name;
    - private String shortname;
    - private double price;
    - private double volume;
    - private double amount;
- **Methods:**
    - **Getters & setters**

Portfolio

- **Attributes:**
    - private String name;
    - private int id;
    - private int address;
    - private double currentValue = 0;
    - private ArrayList<Cryptocurrency> cryptocurrencies;
- **Methods:**
    - **getAmountOfSpecificCoin(String shortName)** -> return double
    - **Getters & setters**

Exchange

- **Attributes:**

    - private Cryptocurrency targetCryptocurrency;
    - private Cryptocurrency baseCryptocurrency;
    - private User user;
- **Methods:**
    - **addCryptoCurrencyToPortfolio(Portfolio portfolio,double amount)** -> return Portfolio: Function to add a new Cryptocurrency to given portfolio
    - **buyCryptocurrency(double baseCoinAmount, String type)** -> return Transaction: Function triggered by event listener (btn_buy)
    - **getIndexOfCurrency (Portfolio portfolio, String shortName)** -> return int: Helper function used to find a currency's index inside given portfolio

Transaction

- **Attributes:**
    - private String type;
    - private double amount;
    - private String timestamp;
    - private String targetCryptocurrency;
    - private String baseCryptocurrency;
- **Methods:**
    - **Getters & setters**

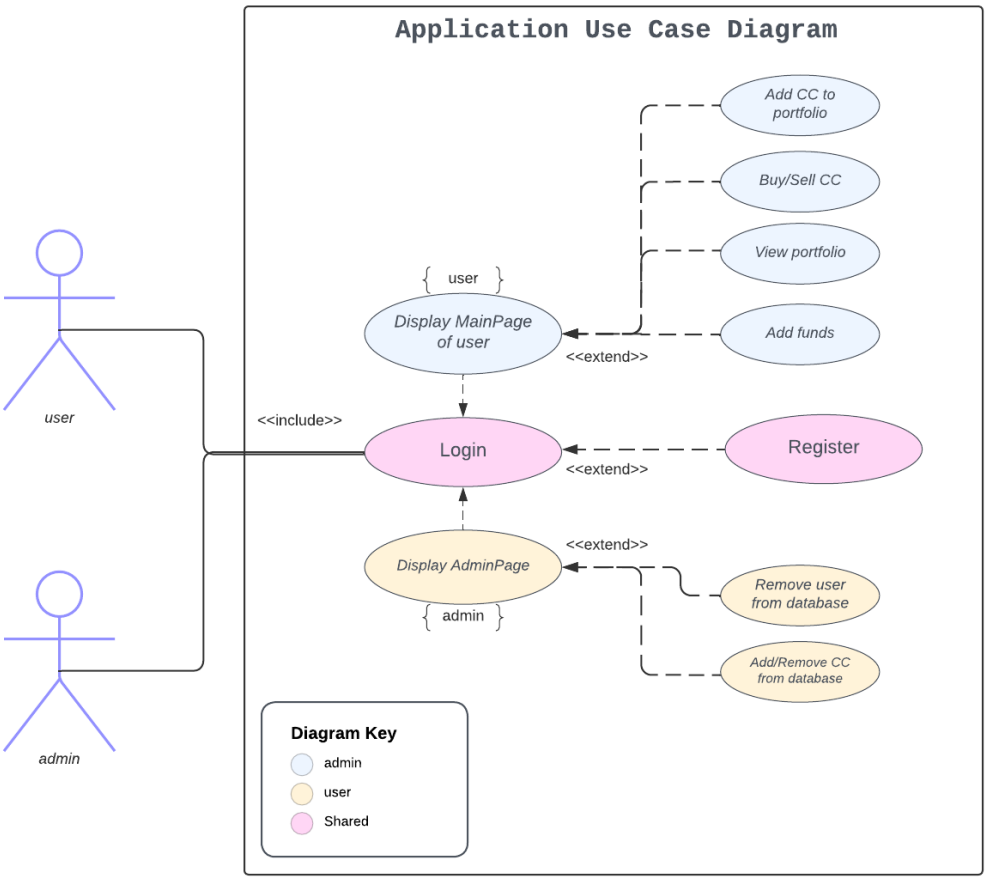
CHAPTER THREE

3.1 Use Case Diagram:

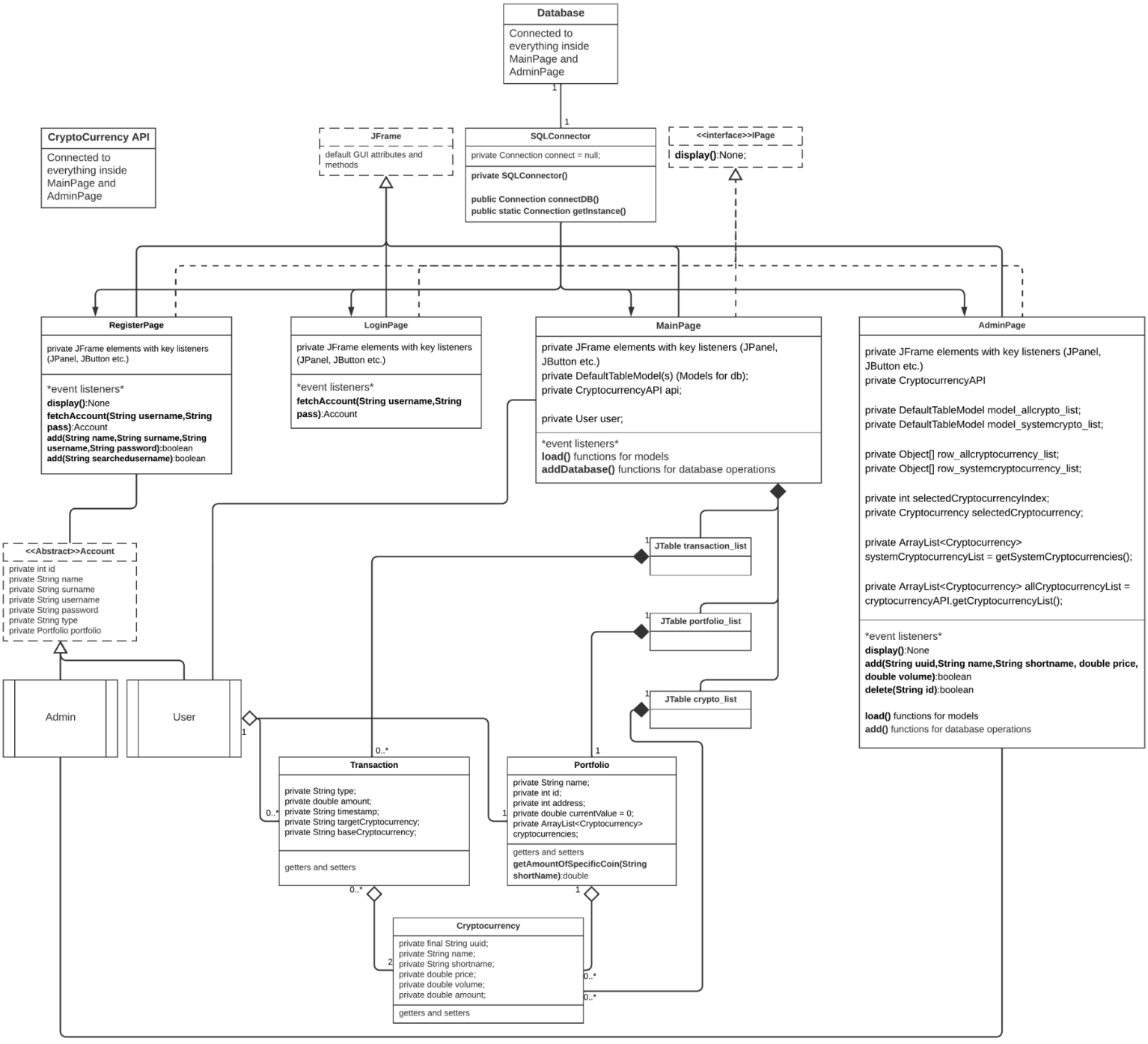
The system handles two different types of actors: **user** and **admin**. Since a user account and an admin account serve different purposes, the MainPage is also different for each.

Even though they share the same login operation, **admin** does not have access to regular **user** functionality (and vice versa), it is merely used for operations related to database regulation (addition and removal of new cryptocurrency types, management of existing users on the database, etc.). The **user** class on the other hand doesn't have direct access to the database, but has access to operations related to their own account.

Following diagram displays high-level functionality of the application:



3.2 Class Diagram:



Some methods (such as some getters & setters, event listeners and load functions) have been grouped for the sake of visual clarity.

Alongside the classes, a mySQL database was implemented into the project to store necessary information such as cryptocurrencies to include, account information, transaction records and more.

Interfaces:

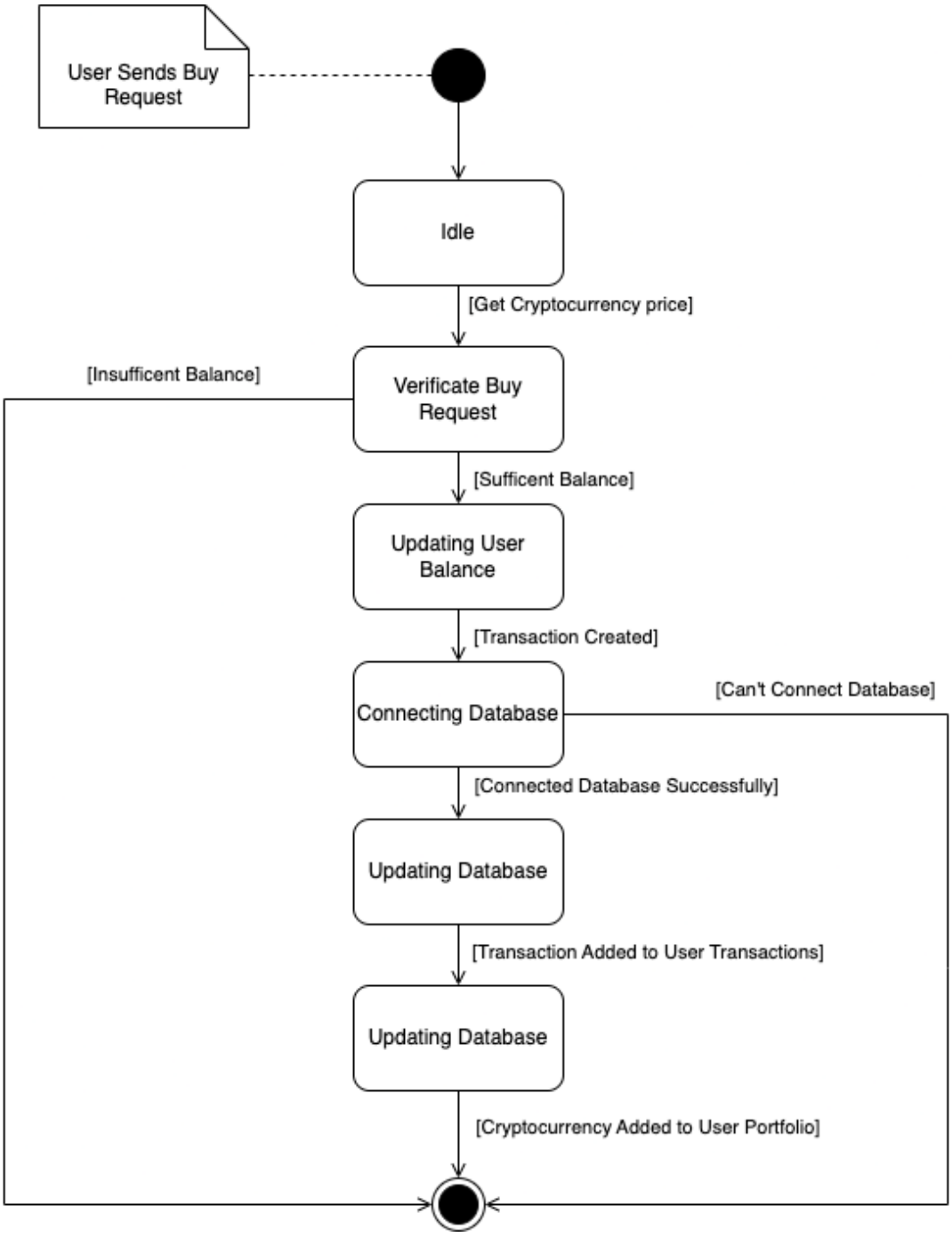
- IPage, implemented by:
  - LoginPage: Login page shared for both admin and user logins.
  - MainPage: Main UI where tabs can be viewed, event listeners.
  - AdminPage: Moderation UI, event listeners.
  - RegisterPage: Registration page, event listeners.

Abstract Classes:

- Account, including most fundamental information, inherited by:
  - user: Personal account in which the primary functionality of the project application can be experienced.
  - admin: Moderation account used for database and application content management.

Classes:

- Cryptocurrency: Represents a cryptocurrency, has its own id in the database. Data is pulled from real life data using a cryptocurrency API.
- Portfolio: Used to store ownership information of a user.
- Transaction: Holds information about a transaction, such as the recipient or the sender info (for simplicity purposes, transactions are treated as if they were one-sided - not client-to-client), timestamp, type, amount etc.
- Exchange: Objects that hold transaction information.
- Cryptocurrency API: API of choice is "*coinranking API*".
- SQLConnector: A single instance of this class was used for all Database connections, as we decided to adopt the Singleton Design Pattern for its purpose.

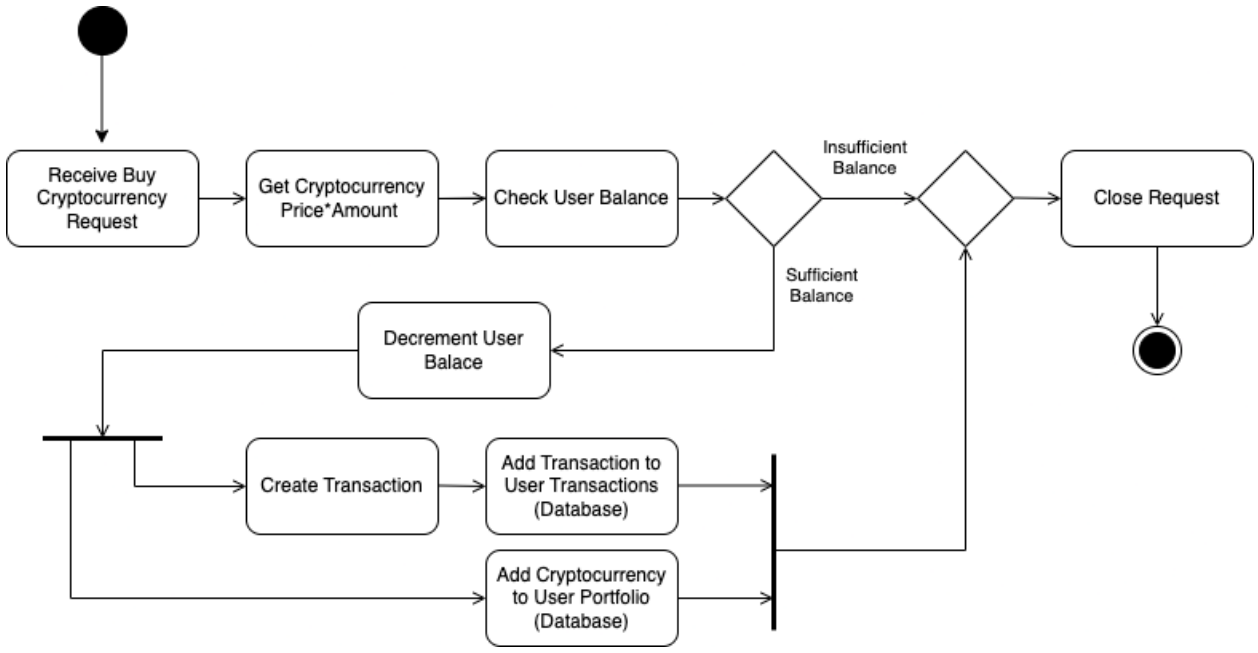


3.3 State Diagram Of Exchange Object

This State Diagram shows the states of Exchange Object when the user sends a buy request.

Firstly, the buy request will be validated by the object. After the verification step, user's balance will be decremented by (amount*price) of the cryptocurrency. Later, the program will create a transaction object to store exchange data. Lastly, if database connection is secured, it will update **User Transactions** table and **User Portfolio** table of the database.

3.4 Activity Diagram Of Exchange Object

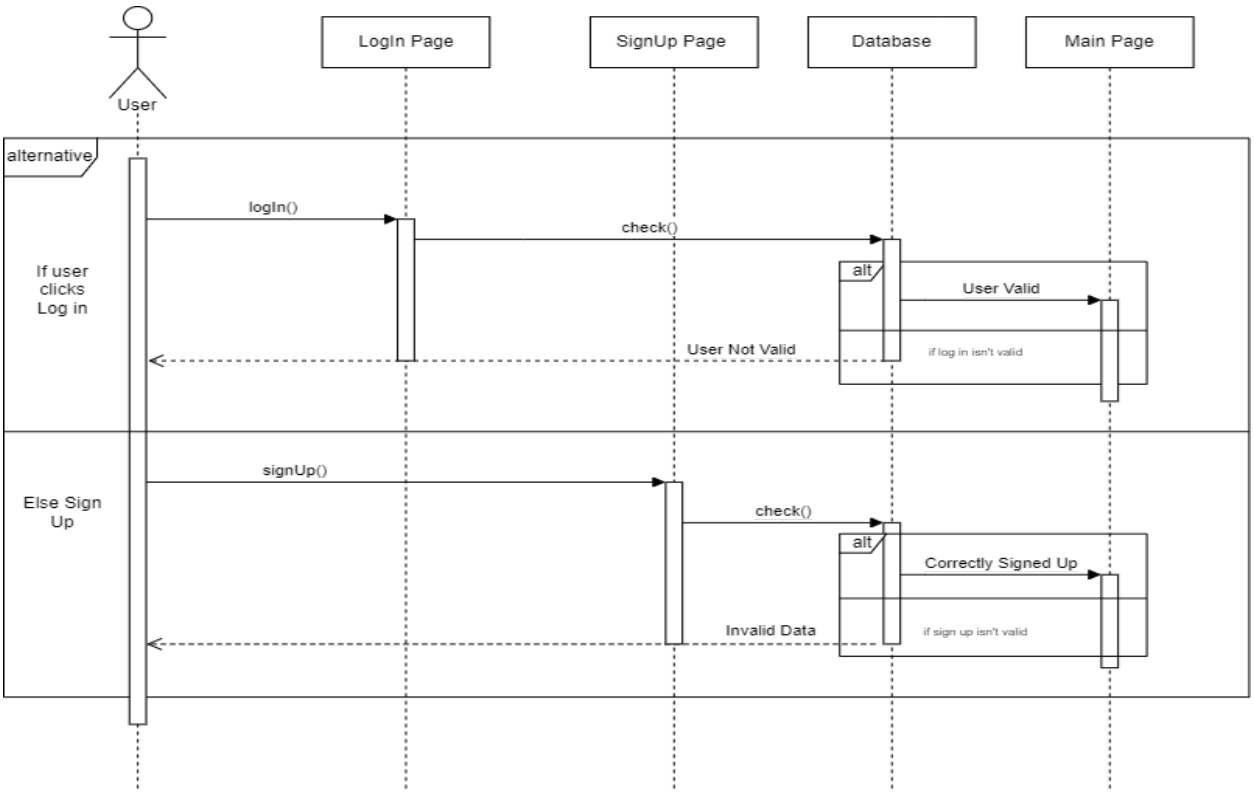


This Activity Diagram shows events performed by the Exchange Object when a user sends a buy request.

Firstly, program will calculate total cost of request (price*amount). Then it will check whether the user has an adequate amount of balance or not. Later, if the user has adequate amount of balance, it will decrement user balance by price*amount of selected cryptocurrency. After that, it will create a Transaction object. Lastly, It will add the Transaction to **User Transactions** table and selected cryptocurrency to **User Portfolio** table of the database.

3.5 Sequence Diagrams

- **Login and Sign up Diagram:**



If the user has an account, the user clicks the login button then, the user enters his/her account information. However, if the user doesn't have an account, he/she clicks the sign up button, then the user should fill the necessary fields on the sign up page. The program has 2 scenarios as mentioned:

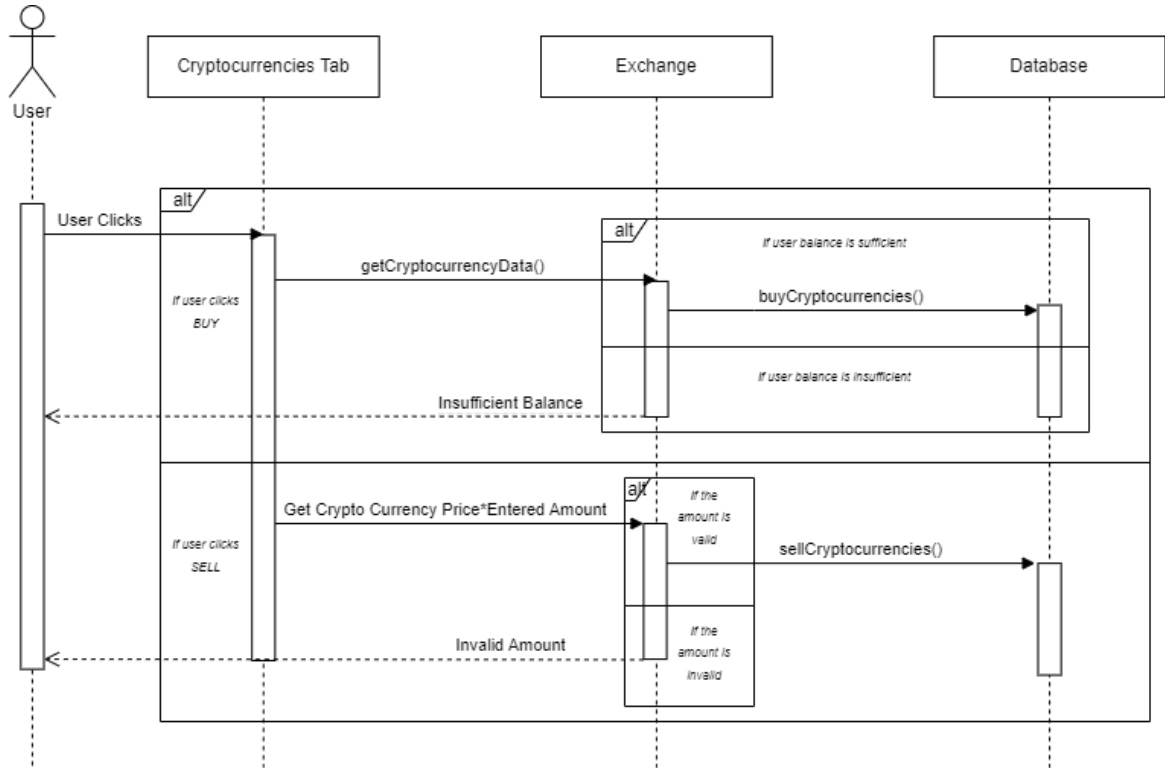
Scenario 1: **Login** (User has an account)

The program checks the entered user login info from the database. If the info is correct, the program proceeds to the main page. If the info is incorrect, the program returns to the login page.

Scenario 2: **Sign up** (User doesn't have an account)

After the newcomer enters his/her personal info, the program checks the entered sign up data from the database. If the data is not valid, the program returns to the sign up page. If the sign up data is valid, an unique portfolio and user id will be created.

- **Transaction Diagram:**



When the user wants to buy or sell a currency, the Transaction tab must be opened. The user has 2 options on the Transaction Tab after selecting the cryptocurrency. The options are **buy** or **sell**.

Option 1 is **Buy**:

In order to begin the buying process, the user must enter the amount that he/she wants to buy. Then the entered amount will be checked from the user's funds. If the balance is sufficient, the buyCryptoCurrencies() function will be called then the buying process will be started. buyCryptoCurrencies() function decreases the user's funds balance by the entered amount and adds the bought cryptocurrency to the user's portfolio.

Option 2 is **Sell**:

In order to begin the selling process, the user must enter the amount as in the buying process. Then the amount will be checked from the user's portfolio (the user may enter an amount greater than he/she has). If the entered amount is valid, the sellCryptoCurrencies() function will be called. If the user enters the whole amount that he/she has, the chosen

cryptocurrency will be removed from the user's portfolio, then the sold amount will be added to the user's funds. If the user enters not the whole amount of the chosen currency, the entered amount will be reduced from the user's portfolio and the sold amount will be added to the user's funds.

3.6 SQLConnector and its Design Pattern

The Singleton design pattern was chosen for the implementation of the SQLConnector class in our application to ensure the existence of only one instance of connection and to provide a global access point to that instance. Any unnecessary processing cost is avoided by eliminating the need to establish multiple connections. This results in enhanced performance and more efficient utilization of system resources.

Furthermore, the Singleton pattern promotes data consistency and integrity within the application. By centralizing access to the database through the Singleton instance of the SQLConnector, potential conflicts arising from simultaneous connections are avoided, which is particularly crucial in the context of a financial application such as a cryptocurrency trading platform, even though it's a mock-up…

CHAPTER FOUR

4.1 Screenshots

AdminPage

```
public AdminPage(Admin admin) {

    // Deleting system's cryptocurrencies from all cryptocurrencies
    for (int i = 0; i < systemCryptocurrencyList.size(); i++) {

        for (int j = 0; j < allCryptocurrencyList.size(); j++) {

            if ((systemCryptocurrencyList.get(i).getUuid().equals(allCryptocurrencyList.get(j).getUuid()))) {

                allCryptocurrencyList.remove(j);
            }
        }
    }
    display();

    button_admin_quit.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            dispose();
            LoginPage loginPage = new LoginPage();
        }
    });
}
```

In Admin Page's constructor, system's cryptocurrencies are removed from allCryptocurrencyList to display which cryptocurrencies that are not in the system.

Also, an event listener is added to the quit button which closes Admin Page and displays the login page when clicked.

display():

```
// Arranging page display properties
add(wrapper);
setSize(1000,500);
int x= Helper.screenCenterPoint("x",getSize());
int y=Helper.screenCenterPoint("y",getSize());
setLocation(x,y);
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
setTitle(Config.PROJECT_TITLE);
setVisible(true);

table_systemcrypto_list.getSelectionModel().addListSelectionListener(e -> {
    try{
        String selected_crypto_name = table_systemcrypto_list.getValueAt(table_systemcrypto_list.getSelectedRow(),1).toString();
        field_removecrypto_name.setText(selected_crypto_name);
    }catch (Exception exception){

    }

});

table_crypto_list.getSelectionModel().addListSelectionListener(e -> {
    try{
        String selected_crypto_name = table_crypto_list.getValueAt(table_crypto_list.getSelectedRow(),1).toString();
        field_crypto_name.setText(selected_crypto_name);
    }catch (Exception exception){

    }

});
```

After arranging page display properties, we added event listeners to tables to display selected cryptocurrency in the selected row user by using the selected row's index.

```
// Adding eventlistener to remove button
button_crypto_remove.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        // Storing selected cryptocurrency and cryptocurrency index
        selectedCryptocurrencyIndex = table_systemcrypto_list.getSelectedRow();
        selectedCryptocurrency = systemCryptocurrencyList.get(selectedCryptocurrencyIndex);

        // Removing selected cryptocurrency from system
        if(delete(selectedCryptocurrency.getUuid())) {
            Helper.showMsg("done");
        }

        // Updating cryptocurrency lists
        allCryptocurrencyList.add(selectedCryptocurrency);
        systemCryptocurrencyList = getSystemCryptocurrencies();

        // Updating cryptocurrency tables
        loadAllCryptocurrencyModel(allCryptocurrencyList);
        loadSystemCryptocurrencyModel(systemCryptocurrencyList);

    }
});
```

We added an event listener to remove the button so that the admin can remove a cryptocurrency from the system.

```
// Creating all cryptocurrency table
model_allcrypto_list = new DefaultTableModel();
model_allcrypto_list.setColumnIdentifiers(col_cryptoList);
row_allcryptocurrency_list = new Object[col_cryptoList.length];
loadAllCryptocurrencyModel(allCryptocurrencyList);
table_crypto_list.setModel(model_allcrypto_list);
table_crypto_list.getColumnModel().getColumn(0).setMaxWidth(75);
table_crypto_list.getTableHeader().setReorderingAllowed(false);


// Creating system's cryptocurrency table
model_systemcrypto_list = new DefaultTableModel();
model_systemcrypto_list.setColumnIdentifiers(col_cryptoList);
row_systemcryptocurrency_list = new Object[col_cryptoList.length];
loadSystemCryptocurrencyModel(systemCryptocurrencyList);
table_systemcrypto_list.setModel(model_systemcrypto_list);
table_systemcrypto_list.getColumnModel().getColumn(0).setMaxWidth(75);
table_systemcrypto_list.getTableHeader().setReorderingAllowed(false);
```

To display all cryptocurrencies and system cryptocurrencies, we created necessary JFrame elements and set their properties to give a nice display.

LoginPage

```
public LoginPage(){

    display();
    btn_register.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            dispose();
            RegisterPage registerPage = new RegisterPage();
        }
    });
}
```

We added an event listener to the register button in Login Page's constructor in order to close the current page and open a Register Page when it is clicked.

```
public void display() {

    // Arranging page display properties
    add(wrapper);
    setSize(400,500);
    setLocation(Helper.screenCenterPoint("x",getSize()),Helper.screenCenterPoint("y",getSize()));
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setTitle(Config.PROJECT_TITLE);
    setResizable(false);
    setVisible(true);

    // Opening page corresponding to account type
    btn_login.addActionListener(e -> {
        if(Helper.isFieldEmpty(fld_user_uname) || Helper.isFieldEmpty(fld_user_pass)){
            Helper.showMsg("fill");
        }else {
            Account account = fetchAccount(fld_user_uname.getText(),fld_user_pass.getText());
            if(account == null){
                Helper.showMsg("Account not found.");
            }else{
                switch (account.getType()){
                    case "admin" :
                        AdminPage adminPage = new AdminPage((Admin) account);
                        break;
                    case "user" :
                        MainPage mainPage = new MainPage((User) account);
                        break;

                }
                dispose();
            }
        }
    });
}
```

After arranging page display properties, we added an event listener to the login button which opens ***MainPage*** or ***AdminPage*** with respect to account type after checking if the necessary fields are filled when it is clicked.

```java
// Retrieving account from database with respect to specified username and password
public static Account fetchAccount(String username,String pass){
    Account obj=null;
    String query="SELECT * FROM account WHERE username=? AND password=?";

    try {
        PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);
        pr.setString(1,username);
        pr.setString(2,pass);
        ResultSet rs=pr.executeQuery();
        if (rs.next()){
            switch (rs.getString("type")){
                case "user":
                    obj=new User();
                    break;
                case "admin":
                    obj=new Admin();
                    break;
                default:
                    Helper.showMsg("error");
            }

            obj.setId(rs.getInt("id"));
            obj.setName(rs.getString("name"));
            obj.setSurname(rs.getString("surname"));
            obj.setUsername(rs.getString("username"));
            obj.setPassword(rs.getString("password"));
            obj.setType(rs.getString("type"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return obj;
}
```

This function is used when the user clicks the login button. After creating the query, it inserts given parameters to the query then executes it. After that, with respect to returned data from the database, it creates an admin or user object with respect to type data. Lastly, it assigns returned data to a newly created object and returns it.

```java
// Creating necessary tables in MySQL database
private static void initializeDatabase () throws SQLException {

    String query = "DROP TABLE `account`;";

    PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);;


    pr.execute();

    query = "DROP TABLE `cryptocurrency`;";

    pr = SQLConnector.getInstance().prepareStatement(query);
    pr.execute();

     query="CREATE TABLE `account` (\n" +
            "\t`id` INT NOT NULL AUTO_INCREMENT,\n" +
            "\t`name` VARCHAR(255) NOT NULL,\n" +
            "\t`surname` VARCHAR(255) NOT NULL,\n" +
            "\t`username` VARCHAR(255) NOT NULL,\n" +
            "\t`password` VARCHAR(255) NOT NULL,\n" +
            "\t`balance` DOUBLE NOT NULL,\n" +
            "\t`type` VARCHAR(255) NOT NULL,\n" +
            "\tPRIMARY KEY (`id`)\n" +
            ");";

    pr = SQLConnector.getInstance().prepareStatement(query);
    pr.execute();


    query="CREATE TABLE `cryptocurrency` (\n" +
            "\t`uuid` VARCHAR(255) NOT NULL,\n" +
            "\t`name` VARCHAR(255) NOT NULL,\n" +
            "\t`shortname` VARCHAR(255) NOT NULL,\n" +
            "\t`price` DOUBLE NOT NULL,\n" +
            "\t`volume` DOUBLE,\n" +
            "\tPRIMARY KEY (`uuid`)\n" +
            ");";

    pr = SQLConnector.getInstance().prepareStatement(query);
    pr.execute();
```

In this method, we created necessary queries to initialize database with necessary tables.

MainPage

Constructor:

```
// Retrieving user portfloio from database
currentUser.setPortfolio(new Portfolio());
currentUser.getPortfolio().setCryptocurrencies(getPortfolio());

display();

// Updating cryptocurrencies prices and user's balance
btn_refresh.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        for (Cryptocurrency cryptocurrency: cryptocurrencyList) {
            cryptocurrency.setPrice(cryptocurrencyAPI.getExchangeRate(cryptocurrency));

            System.out.println(cryptocurrency.getPrice());
        }

        currentUser.setBalance(currentUser.getBalance());

        lbl_mainpage_totalbalance.setText("Your total balance is: " + currentUser.getBalance() + " USD");

        loadCryptocurrencyModel(cryptocurrencyList);
    }

});

// Displaying selected cryptocurrency
tbl_crypto_list.getSelectionModel().addListSelectionListener(e -> {
    try{
        String selected_crypto_name = tbl_crypto_list.getValueAt(tbl_crypto_list.getSelectedRow(),1).toString();
        fld_cryptobuy_name.setText(selected_crypto_name);
    }catch (Exception exception){

    }

});
```

We created a new Portfolio for the user. Then we retrieved his/her cryptocurrencies from the database.

We added an event listener to refresh button which loops through cryptocurrencies and retrieves their live prices by using API and updates cryptocurrencies prices when it is clicked.

We added an event listener to the cryptocurrency table to display selected cryptocurrency to user by using the selected row's index.

```
// Going back to login page when user click quit
btn_logout.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        // Adding user's cryptocurrencies to database
        addDatabase(currentUser);

        dispose();
        LoginPage loginPage = new LoginPage();
    }
});

// Adding bought cryptocurrency to user portfolio and transaction
btn_crypto_buy.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Cryptocurrency selectedCoin = cryptocurrencyList.get(tbl_crypto_list.getSelectedRow());
        String shortName = (String) combo_box_currencies.getSelectedItem();
        int index = Exchange.getIndexOfCurrency(currentUser.getPortfolio(),shortName);
        Cryptocurrency baseCoin = cryptocurrencyList.get(index);
        Exchange exchange = new Exchange(currentUser,selectedCoin,baseCoin);
        double amount = Double.parseDouble(fld_cryptobuy_amount.getText());
        addDatabase(exchange.buyCryptocurrency(amount,"BUY"));
        display();
    }
});

// Displaying selected cryptocurrency
tbl_portfolio_list.getSelectionModel().addListSelectionListener(e -> {
    try{
        String selected_crypto_name = tbl_portfolio_list.getValueAt(tbl_portfolio_list.getSelectedRow(),1).toString();
        fld_cryptosell_name.setText(selected_crypto_name);
    }catch (Exception exception){

    }

});
```

We added an event listener to the register button in Login Page's constructor in order to close the current page and open a Register Page when it is clicked.

We added an event listener to the buy button in order to perform the operation. Takes the selected coin from the ***tbl_crypto_list*** as the target coin and takes the selected item from the ***combo_box_currencies*** as the base coin. Then it calls the ***buyCryptocurrency()*** function. If the process is valid, add the returned transaction to the database.

We added an event listener to the portfolio table to display selected cryptocurrency to user by using the selected row's index.

```java
// Selling user selected cryptocurrency from user portfolio and storing transaction
btn_crypto_sell.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Cryptocurrency selectedCoin = currentUser.getPortfolio().getCryptocurrencies().get(tbl_portfolio_list.getSelectedRow());
        String shortName = (String) combo_box_portfolio.getSelectedItem();
        int index = Exchange.getIndexOfCurrency(currentUser.getPortfolio(),shortName);
        Cryptocurrency baseCoin = currentUser.getPortfolio().getCryptocurrencies().get(index); // temporary --<vodka USDT
        Exchange exchange = new Exchange(currentUser,baseCoin,selectedCoin);
        double amount = Double.parseDouble(fld_cryptosell_amount.getText());
        addDatabase(exchange.buyCryptocurrency(amount,"SELL"));
        display();
    }
});
```

We added an event listener to the sell button in order to perform the operation. Takes the selected coin from the ***user portfolio*** as the base coin and takes the selected item from the ***combo_box_portfolio*** as the target coin. Then it calls the ***buyCryptocurrency()*** function. If the process is valid, add the returned transaction to the database.

display():

```
// Arranging page display properties
add(wrapper);
setSize(1000,500);
int x= Helper.screenCenterPoint("x",getSize());
int y=Helper.screenCenterPoint("y",getSize());
setLocation(x,y);
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
setTitle(Config.PROJECT_TITLE);
setVisible(true);

// Getting system's cryptocurrencies
cryptocurrencyList = getCryptocurrencyList();


lbl_mainpage_welcome.setText("Welcome: " + currentUser.getName() + " " + currentUser.getSurname());
lbl_mainpage_totalbalance.setText("Your total balance is: " + currentUser.getBalance() + " USD");


comboBoxModelForCoinList = new DefaultComboBoxModel<>();
combo_box_currencies.setModel(comboBoxModelForCoinList);
loadBaseCoinListModel(currentUser.getPortfolio().getCryptocurrencies(),combo_box_currencies);

comboBoxModelForPortfolio = new DefaultComboBoxModel<>();
combo_box_portfolio.setModel(comboBoxModelForPortfolio);
loadBaseCoinListModel(currentUser.getPortfolio().getCryptocurrencies(),combo_box_portfolio);
```

After arranging page display properties, we retrieved cryptocurrencies which are added to the system from the database.

We printed the user's name, surname and balance.

We created a combo box for display cryptocurrencies which can buyed by retrieving them from the database.

We created a combo box for display cryptocurrencies which can selled by retrieving them from the database.

```
// Creating cryptocurrencies table
model_crypto_list = new DefaultTableModel();
Object[] col_cryptoList= {"Name","Symbol", "Current Price"};
model_crypto_list.setColumnIdentifiers(col_cryptoList);
row_cryptocurrency_list = new Object[col_cryptoList.length];
loadCryptocurrencyModel(cryptocurrencyList);
tbl_crypto_list.setModel(model_crypto_list);
tbl_crypto_list.getColumnModel().getColumn(0).setMaxWidth(75);
tbl_crypto_list.getTableHeader().setReorderingAllowed(false);

// Creating portfolio table
model_portfolio_list = new DefaultTableModel();
Object[] col_portfolioList= {"Name","Symbol", "Amount","Current Price"};
model_portfolio_list.setColumnIdentifiers(col_portfolioList);
row_portfolio_list = new Object[col_portfolioList.length];
loadPortfolioModel(currentUser.getPortfolio().getCryptocurrencies());
tbl_portfolio_list.setModel(model_portfolio_list);
tbl_portfolio_list.getColumnModel().getColumn(0).setMaxWidth(75);
tbl_portfolio_list.getTableHeader().setReorderingAllowed(false);

// Creating transactions table
model_transaction_list = new DefaultTableModel();
Object[] col_transactionList= {"Type","Base", "Target","Amount", "Date"};
model_transaction_list.setColumnIdentifiers(col_transactionList);
row_transaction_list = new Object[col_transactionList.length];
loadTransactionModel(getTransactions());
tbl_transaction_list.setModel(model_transaction_list);
tbl_transaction_list.getColumnModel().getColumn(0).setMaxWidth(75);
tbl_transaction_list.getTableHeader().setReorderingAllowed(false);
```

To display system cryptocurrencies, user’s portfolio and user’s transactions, we created necessary JFrame elements and set their properties to give a nice display.

```java
// Function for updating user portfolio
private void loadPortfolioModel(ArrayList<Cryptocurrency> portfolioCryptocurrencyList) {
    DefaultTableModel clearModel=(DefaultTableModel) tbl_portfolio_list.getModel();
    clearModel.setRowCount(0);
    int i=0;
    for(Cryptocurrency obj: portfolioCryptocurrencyList){
        i=0;
        row_portfolio_list[i++] = obj.getName();
        row_portfolio_list[i++] = obj.getShortname();
        row_portfolio_list[i++] = obj.getAmount();
        row_portfolio_list[i++] = obj.getPrice();
        model_portfolio_list.addRow(row_portfolio_list);
    }
}

// Function for retrieving user portfolio from database
private static ArrayList<Cryptocurrency> getPortfolio() {

    ArrayList<Cryptocurrency> cryptocurrencyList =new ArrayList<>();

    Cryptocurrency cryptocurrency;
    try {
        Statement st= SQLConnector.getInstance().createStatement();
        ResultSet rs=st.executeQuery("SELECT * from portfolio WHERE user_id = " + currentUser.getId());
        while (rs.next()){

            String uuid = rs.getString("uuid");
            String shortName = rs.getString("short_name");
            String name = rs.getString("name");
            double amount = rs.getDouble("amount");
            cryptocurrency = new Cryptocurrency(uuid, name, shortName, amount);

            cryptocurrency.setPrice(cryptocurrencyAPI.getExchangeRate(cryptocurrency));
            cryptocurrencyList.add(cryptocurrency);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return cryptocurrencyList;

}
```

First function is used to update the portfolio table. It loops through a given ***Arraylist<Cryptocurrency>*** and adds its contents to the table.

Second function is used to retrieve user portfolio from the database. Firstly, It returns all cryptocurrencies from the database with respect to current user id, then it creates cryptocurrency objects by using them. Lastly, it stores cryptocurrency objects in arraylist and returns the arraylist.

```java
// Function for storing user portfolio
public static boolean addDatabase(User user) {

    String query = "DELETE FROM portfolio WHERE user_id = " + currentUser.getId();

    try {
        PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);
        int response= pr.executeUpdate();

        if(response == -1){
            Helper.showMsg("error");
        }

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }

    for (Cryptocurrency cryptocurrency : user.getPortfolio().getCryptocurrencies()) {
        System.out.println(cryptocurrency.getShortname());
        query="INSERT INTO portfolio (user_id,uuid,short_name,name,amount) VALUES (?,?,?,?,?)";

        try {
            PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);
            pr.setInt(1, user.getId());
            pr.setString(2,cryptocurrency.getUuid());
            pr.setString(3,cryptocurrency.getShortname());
            pr.setString(4,cryptocurrency.getName());
            pr.setDouble(5,cryptocurrency.getAmount());
            int response= pr.executeUpdate();

            if(response == -1){
                Helper.showMsg("error");
                return false;
            }


        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }


    return true;

}
```

This function is used to store the user’s portfolio in the database. Firstly, it deletes user’s previous bought cryptocurrency records, then it loops through the user's portfolio and inserts them in the database.

```java
// Function for storing user transactions
public static boolean addDatabase(Transaction transaction){
    String query="INSERT INTO transaction (user_id,type,amount,base_crypto,target_crypto,time_stamp) VALUES (?,?,?,?,?,?)";

    try {
        PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);
        pr.setInt(1,currentUser.getId());
        pr.setString(2,transaction.getType());
        pr.setDouble(3,transaction.getAmount());
        pr.setString(4,transaction.getBaseCryptocurrency());
        pr.setString(5,transaction.getTargetCryptocurrency());
        pr.setString(6,transaction.getTimestamp());
        int response= pr.executeUpdate();

        if(response == -1){
            Helper.showMsg("error");
        }
        return response != -1;

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }

    return true;
}
```

This function is used to store a transaction in the database. After creating the insert query, it inserts given parameters to the query then executes it.

RegisterPage

```java
// Function for updating user transactions
private void loadTransactionModel(ArrayList<Transaction> transactionList) {
    DefaultTableModel clearModel=(DefaultTableModel) tbl_transaction_list.getModel();
    clearModel.setRowCount(0);
    int i=0;
    for(Transaction obj: transactionList){
        i=0;
        row_transaction_list[i++] = obj.getType();
        row_transaction_list[i++] = obj.getBaseCryptocurrency();
        row_transaction_list[i++] = obj.getTargetCryptocurrency();
        row_transaction_list[i++] = obj.getAmount();
        row_transaction_list[i++] = obj.getTimestamp()

        ;
        model_transaction_list.addRow(row_transaction_list);
    }
}

// Function for retrieving user transactions from database
private static ArrayList<Transaction> getTransactions() {

    ArrayList<Transaction> transactionsList =new ArrayList<>();

    Transaction transaction;
    try {
        Statement st= SQLConnector.getInstance().createStatement();
        ResultSet rs=st.executeQuery("SELECT * from transaction WHERE user_id = " + currentUser.getId());
        while (rs.next()){

            String type = rs.getString("type");
            double amount = rs.getDouble("amount");
            String base_crypto = rs.getString("base_crypto");
            String target_crypto = rs.getString("target_crypto");
            String time_stamp = rs.getString("time_stamp");

            transaction = new Transaction(type, amount, target_crypto, base_crypto,time_stamp);

            transactionsList.add(transaction);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return transactionsList;

}
```

First function is used to update the transaction table. It loops through the given ***ArrayList<Transaction>*** arraylist and writes its content to transaction table.

Second function is used to retrieve user transactions from the database. Firstly, It returns all transactions from the database with respect to current user id, then it creates transaction objects by using them. Lastly, it stores transaction objects in arraylist and returns the arraylist.

display():

```
public void display() {

    // Arranging page display properties
    add(wrapper);
    setSize(400,500);
    setLocation(Helper.screenCenterPoint("x",getSize()),Helper.screenCenterPoint("y",getSize()));
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setTitle(Config.PROJECT_TITLE);
    setResizable(false);
    setVisible(true);

    btn_user_register.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            if(Helper.isFieldEmpty(fld_user_name)|| Helper.isFieldEmpty(fld_user_surname) || Helper.isFieldEmpty(fld_user_username)
                    ||Helper.isFieldEmpty(fld_user_password)){
                Helper.showMsg("fill");
            }else{
                String name=fld_user_name.getText();
                String surname=fld_user_surname.getText();
                String username=fld_user_username.getText();
                String pass=fld_user_password.getText();
                if(add(name,surname,username,pass)){
                    System.out.println("searched username : " + username);
                    add(username); // Initialize the user portfolio (added to the database)
                    Helper.showMsg("done");
                    dispose();
                    LoginPage loginPage = new LoginPage();
                }
            }

        }
    });

}
```

After arranging page display properties, we added an event listener to register button which adds the user to the database and initializes its portfolio if the given username is not taken and all necessary fields are filled.

add():

```java
// Function for adding user to database
public static boolean add(String name,String surname,String username,String password){
    String query="INSERT INTO account (name,surname,username,password,balance,type) VALUES (?,?,?,?,10000,'user')";
    Account findUser = RegisterPage.fetchAccount(username);
    if(findUser!=null){
        Helper.showMsg("This username is taken, please choose another username");
        return false;
    }
    try {
        PreparedStatement pr=SQLConnector.getInstance().prepareStatement(query);
        pr.setString(1,name);
        pr.setString(2,surname);
        pr.setString(3,username);
        pr.setString(4,password);
        int response= pr.executeUpdate();

        if(response == -1){
            Helper.showMsg("error");
        }
        return response != -1;

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }

    return true;
}
```

This function is used to store a user in the database. After creating the insert query, we checked if the username is taken or not by using ***fetchAccount()*** method. If the username isn't taken, it inserts given parameters to the query then executes it.

add():

```java
// Function for adding default cryptocurrency to user portfolio
public boolean add(String searchedUsername){

    String query="INSERT INTO portfolio (user_id,uuid,short_name,name,amount) VALUES (?,?,?,?,?)";
    String query2="SELECT * FROM account WHERE username =?";

    try {
        Cryptocurrency cryptocurrency = new Cryptocurrency("HIVsRcGKkPFtW","Tether USD","USDT",1.0,18942563028.0);
        cryptocurrency.setAmount(10000);

        PreparedStatement pr2 = SQLConnector.getInstance().prepareStatement(query2);

        pr2.setString(1,searchedUsername);


        ResultSet rs=pr2.executeQuery();

        int userID = -1;

        if (rs.next()) {
            if (rs.getString("username").equals(searchedUsername)) {
                userID = rs.getInt("id");
            }
        }
        PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);

        pr.setInt(1, userID);
        pr.setString(2,cryptocurrency.getUuid());
        pr.setString(3,cryptocurrency.getShortname());
        pr.setString(4,cryptocurrency.getName());
        pr.setDouble(5,cryptocurrency.getAmount());
        int response= pr.executeUpdate();

        if(response == -1){
            Helper.showMsg("error");
        }
        else
            Helper.showMsg("Added to the database");
        return response != -1;

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        Helper.showMsg(e.getMessage());
    }

    return true;
}
```

This function's main purpose is to create a new portfolio for a newcomer and add it to the database. In order to connect the portfolio to the user, the user id must be checked and got. Thus, we have a query that searches the unique username to find the user id.

```java
// Function for retrieving specified accont from database
public static Account fetchAccount(String username){
    Account obj=null;
    String query="SELECT * FROM account WHERE username =?";

    try {
        PreparedStatement pr = SQLConnector.getInstance().prepareStatement(query);
        pr.setString(1,username);
        ResultSet rs=pr.executeQuery();
        if (rs.next()){
            obj=new User();
            obj.setId(rs.getInt("id"));
            obj.setName(rs.getString("name"));
            obj.setUsername(rs.getString("username"));
            obj.setPassword(rs.getString("password"));
            obj.setType(rs.getString("type"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return obj;
}
```

Making a query on the Account table in the database during the registration process in order to check if the entered username was already taken.

SQLConnector

```java
public class SQLConnector {

    private Connection connect = null;
    private static SQLConnector instance = null; // Singleton instance

    // Private constructor to prevent direct instantiation
    private SQLConnector() {
    }

    // Creating database connection
    public Connection connectDB() {
        try {
            this.connect = DriverManager.getConnection(Config.DB_URL, Config.DB_USERNAME, Config.DB_PASSWORD);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return this.connect;
    }

    // Singleton getInstance() method to obtain the single instance
    public static Connection getInstance() {
        if (instance == null) {
            synchronized (SQLConnector.class) {
                if (instance == null) {
                    instance = new SQLConnector();
                }
            }
        }
        return instance.connectDB();
    }
}
```

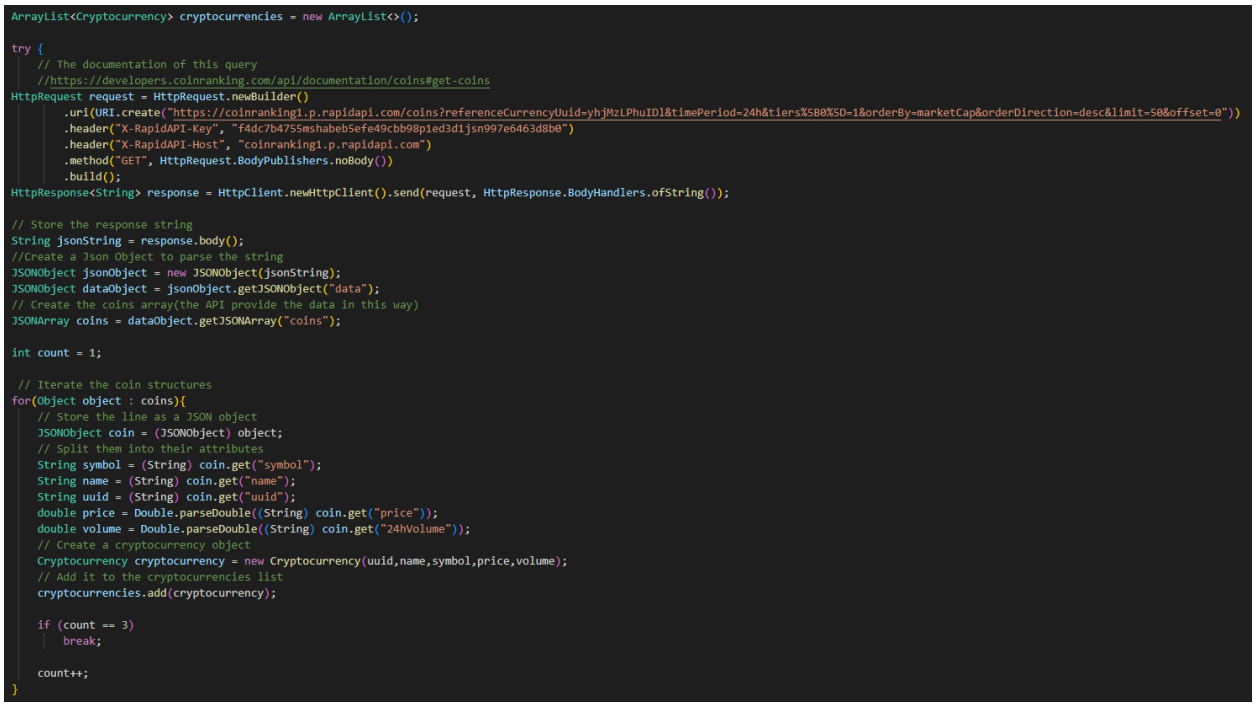
Since this class is implemented by using **Singleton Design Pattern,** we have an empty constructor to prevent random instantiation.

In the connection***()*** method, we are creating connection to the database by using database username and database password which are retrieved from the project's configuration.

In the getInstance***()*** method, we are checking if the **SQLConnector** instance is initialized in the project or not. If it is not initialized, we initialize a new instance and return it.

CryptocurrencyAPI

getCryptocurrencyList():



```
ArrayList<Cryptocurrency> cryptocurrencies = new ArrayList<>();

try {
    // The documentation of this query
    //https://developers.coinranking.com/api/documentation/coins#get-coins
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("https://coinranking1.p.rapidapi.com/coins?referenceCurrencyUuid=yhjMzLPhuIDl&timePeriod=24h&tiers%5B0%5D=1&orderBy=marketCap&orderDirection=desc&limit=50&offset=0"))
        .header("X-RapidAPI-Key", "f4dc7b4755mshabeb5efe49cbb98p1ed3d1jsn997e6463d8b0")
        .header("X-RapidAPI-Host", "coinranking1.p.rapidapi.com")
        .method("GET", HttpRequest.BodyPublishers.noBody())
        .build();
HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());

// Store the response string
String jsonString = response.body();
//Create a Json Object to parse the string
JSONObject jsonObject = new JSONObject(jsonString);
JSONObject dataObject = jsonObject.getJSONObject("data");
// Create the coins array(the API provide the data in this way)
JSONArray coins = dataObject.getJSONArray("coins");

int count = 1;

 // Iterate the coin structures
for(Object object : coins){
    // Store the line as a JSON object
    JSONObject coin = (JSONObject) object;
    // Split them into their attributes
    String symbol = (String) coin.get("symbol");
    String name = (String) coin.get("name");
    String uuid = (String) coin.get("uuid");
    double price = Double.parseDouble((String) coin.get("price"));
    double volume = Double.parseDouble((String) coin.get("24hVolume"));
    // Create a cryptocurrency object
    Cryptocurrency cryptocurrency = new Cryptocurrency(uuid,name,symbol,price,volume);
    // Add it to the cryptocurrencies list
    cryptocurrencies.add(cryptocurrency);

    if (count == 3)
        break;

    count++;
}
```

This function is connected to Coinranking API. The function sends a request to the API with a unique key, then the API returns a list f cryptocurrencies in JSON format. After getting the response from the API, the function parses the JSON file and then set the attributes of each currency and adds them to the ***Arraylist<Cryptocurrency>***.

getExchangeRate():

```java
public double getExchangeRate(Cryptocurrency fromCoin, Cryptocurrency toCoin){
   double exchangeRate = -1.0;
    try {
    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("https://coinranking1.p.rapidapi.com/coin/"+fromCoin.getUuid()+"/price?referenceCurrencyUuid="+toCoin.getUuid()))
            .header("X-RapidAPI-Key", "f4dc7b4755mshabeb5efe49cbb98p1ed3d1jsn997e6463d8b0")
            .header("X-RapidAPI-Host", "coinranking1.p.rapidapi.com")
            .method("GET", HttpRequest.BodyPublishers.noBody())
            .build();
    HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
        // Store the response string
        String jsonString = response.body();
        //Create a Json Object to parse the string
        JSONObject jsonObject = new JSONObject(jsonString);
        JSONObject dataObject = jsonObject.getJSONObject("data");
        exchangeRate = Double.parseDouble((String) dataObject.get("price"));

    }catch (InterruptedException | IOException e) {
        throw new RuntimeException(e);
    }
   return exchangeRate;
}


public double getExchangeRate(Cryptocurrency cryptocurrency){
   double exchangeRate = -1.0;
    try {
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("https://coinranking1.p.rapidapi.com/coin/"+cryptocurrency.getUuid()+"/price?referenceCurrencyUuid=HIVsRcGKkPFtW"))
                .header("X-RapidAPI-Key", "f4dc7b4755mshabeb5efe49cbb98p1ed3d1jsn997e6463d8b0")
                .header("X-RapidAPI-Host", "coinranking1.p.rapidapi.com")
                .method("GET", HttpRequest.BodyPublishers.noBody())
                .build();
        HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
        // Store the response string
        String jsonString = response.body();
        //Create a Json Object to parse the string
        JSONObject jsonObject = new JSONObject(jsonString);
        JSONObject dataObject = jsonObject.getJSONObject("data");
        // Get the exchange rate
        exchangeRate = Double.parseDouble((String) dataObject.get("price"));

    }catch (InterruptedException | IOException e) {
        throw new RuntimeException(e);
    }
    return exchangeRate;
}
```

There are 2 types of this function. The upper one gets 2 parameters that return the exchange rate of the first coin based on the second coin (e.g. getExchangeRate(“BTC”,”ETH”) returns the value of 1 BTC’s ETH value) from the API. The below one returns the USDT value of the given coin from the API.

Exchange

buyCryptocurrency():

```java
public Transaction buyCryptocurrency(double baseCoinAmount, String type){

    System.out.println("initial target amount : " + targetCryptocurrency.getAmount());
    System.out.println("initial base amount : " + baseCryptocurrency.getAmount());
    Portfolio portfolio = user.getPortfolio();

    System.out.println(baseCryptocurrency.getName());
    int baseCoinIndex = getIndexOfCurrency(portfolio,baseCryptocurrency.getShortname());
    if(baseCoinIndex == user.getPortfolio().getCryptocurrencies().size()){
        Helper.showMsg(baseCryptocurrency.getShortname() + " isn't available in your portfolio!");
        return null;
    }
    Cryptocurrency baseCryptocurrencyInPortfolio = portfolio.getCryptocurrencies().get(baseCoinIndex);
    System.out.println("wanted amount : " + baseCoinAmount);
    System.out.println("portfolio amount : " + baseCryptocurrencyInPortfolio.getAmount());

    if(baseCoinAmount <= baseCryptocurrencyInPortfolio.getAmount()) { // If user has sufficient balance
        System.out.println("!Sufficient Balance!");
        CryptocurrencyAPI API = new CryptocurrencyAPI();
        double targetCoinAmount = baseCoinAmount * 1/API.getExchangeRate(targetCryptocurrency,baseCryptocurrency); // Store the amount of the target coin
        // Add the target coin to the user portfolio
        portfolio = addCryptoCurrencyToPortfolio(portfolio,targetCoinAmount);

        // Decrease the amount of the base coin
        baseCryptocurrencyInPortfolio = portfolio.getCryptocurrencies().get(baseCoinIndex);
        double oldAmount = baseCryptocurrencyInPortfolio.getAmount();
        if(baseCoinAmount == oldAmount){ // If all the balance of the base coin is spending
            portfolio.getCryptocurrencies().remove(baseCryptocurrencyInPortfolio); // remove the coin from the portfolio
        }
        else {
            portfolio.getCryptocurrencies().get(baseCoinIndex).setAmount(oldAmount-baseCoinAmount); // decrease the amount of the base coin
        }
        // Update the user portfolio
        user.setPortfolio(portfolio);

        System.out.println();
        for(Cryptocurrency cryptocurrency : user.getPortfolio().getCryptocurrencies()){
            System.out.println(cryptocurrency.getShortname() + " " + cryptocurrency.getAmount());
        }
        System.out.println();
        System.out.println();
        System.out.println();
        System.out.println();


        // Create a transaction
        Transaction transaction = new Transaction(type,targetCoinAmount,targetCryptocurrency.getShortname(),baseCryptocurrency.getShortname());
        return transaction;
    }
```

This function is used for buy and sell operations. Logically buy and sell operations are the same operations with different flows. When the user wants to buy a new coin the target coin is the newer one and the base coin is the selected coin by the user from his/her portfolio. In the sell operation, the base coin is the coin user wants to sell from the portfolio. The first thing that the function does is check the base coin amount from the portfolio. If the balance is sufficient, it calculates the exact amount of the target coin by the live exchange rates. Then calls the ***addCryptocurrenctToPortfolio()*** function to add the new coin to the portfolio. After adding the coin to the portfolio, the function reduces the amount of the base coin from the portfolio (deletes the coin if the amount is 0(zero)). Then create a transaction for the successful exchange and returns it.

addCryptocurrencyToPortfolio():

```
public Portfolio addCryptoCurrencyToPortfolio(Portfolio portfolio,double amount){
    // Get the coin list
    ArrayList<Cryptocurrency> cryptocurrencies = portfolio.getCryptocurrencies();

    Iterator<Cryptocurrency> iterator = cryptocurrencies.iterator();
    boolean contains = false;
    int indexCounter = 0; // Store the index of the coin if occurs
    while (iterator.hasNext()){ // Check if the target coin is already bought
        Cryptocurrency check = iterator.next();
        if (check.getShortname().equals(targetCryptocurrency.getShortname())){
            contains = true;
            break;
        }
        indexCounter++;
    }
    if (contains) {
        System.out.println("!CONTAINS!");
        // Add the new amount to the coin (update the amount)
        cryptocurrencies.get(indexCounter).addAmount(amount);
    }
    else{ // If the target coin is not in the user's portfolio
        System.out.println("!NOT CONTAINS!");
        // Create a new coin object not to change original coin list
        Cryptocurrency cryptocurrency = targetCryptocurrency;
        // Add the amount to the coin
        cryptocurrency.setAmount(amount);
        // Add the coin to user's portfolio
        cryptocurrencies.add(cryptocurrency);
    }
    // Update the user's portfolio list
    portfolio.setCryptocurrencies(cryptocurrencies);

    return portfolio;
} // end addCryptoCurrencyToPortfolio
```

This method is called in ***buyCryptocurrency()*** function. Basically checks if the coin that the user wants to buy is already in his/her portfolio. If already available in the portfolio, just updates the amount of the currency. If the coin is not in the portfolio, add a new coin to the user portfolio with the wanted amount.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

The nature of the project required us to explore different aspects of application development: Alongside our main focus of object-oriented programming, we experimented with databases and GUI design.

There are a few unimplemented improvements we can think of such as visualization of profits/losses, price charts, UI upgrades and accessibility improvements. Nevertheless, since we implemented majority of the functionality we planned, we are more than satisfied with the final result as it is, and think we went above and beyond the project requirements.