

|  |  |
| --- | --- |
| SUBMITTED TO | SIR SHAHID BHATTI |
| TOPIC | LAB ASSIGNMENT 4 |
| PROJECT NAME | COIN FLIPPER |
| SUBMITTED BY | RANA DANISH(139),M QASIM(079) AND M TAYYAB(57) |

AUTHENTIFICATION.JAVA :

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import com.example.crypto\_price\_tracker.util.UserDAO;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*\**;

import javafx.scene.layout.*\**;

import javafx.stage.Stage;

public class Authentication {

    public static boolean isAuthenticated = false;

    private boolean isSignUpView = true;

    public void loadSignUpLayout() {

        Stage stage = new Stage();

        stage.setTitle("Sign Up / Login");

        Label signUpLabel = new Label("Sign Up");

        signUpLabel.setId("signUp");

        signUpLabel.setPadding(new Insets(0,0,0,80));

        Label alreadyHaveAccountLabel = new Label("Already have an account? Login");

        alreadyHaveAccountLabel.setStyle("-fx-text-fill: #cecece;-fx-font-size: 20px;");

        alreadyHaveAccountLabel.setOnMouseClicked(event -> switchToLoginView(stage));

        Label usernameLabel = new Label("Username:");

        TextField usernameField = new TextField();

        usernameField.setPromptText("Username");

        usernameField.setPrefHeight(50);

        Label emailLabel = new Label("Email:");

        TextField emailField = new TextField();

        emailField.setPromptText("Email");

        emailField.setPrefHeight(50);

        Label passwordLabel = new Label("Password:");

        PasswordField passwordField = new PasswordField();

        passwordField.setPromptText("Password");

        passwordField.setPrefHeight(50);

        Label passwordKeyLabel = new Label("Password Key:");

        PasswordField passwordKeyField = new PasswordField();

        passwordKeyField.setPromptText("Password Key (In case you forget your password)");

        passwordKeyField.setPrefHeight(50);

        Label passwordRecoveryInfoLabel = new Label("In case you forget your password, you can recover it using the Password Key.");

        passwordRecoveryInfoLabel.setId("passwordKey");

        Button signUpButton = new Button("Sign Up");

        signUpButton.setPrefWidth(100);

        signUpButton.setOnAction(event -> {

            String username = usernameField.getText();

            String email = emailField.getText();

            String password = passwordField.getText();

            String passwordKey = passwordKeyField.getText();

            if (username == null || username.trim().isEmpty()) {

                CUSTOMALERT.showAlert("Invalid Input", "Username cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            if (email == null || email.trim().isEmpty()) {

                CUSTOMALERT.showAlert("Invalid Input", "Email cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            if (password == null || password.trim().isEmpty()) {

                CUSTOMALERT.showAlert("Invalid Input", "Password cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            if (passwordKey == null || passwordKey.trim().isEmpty()) {

                CUSTOMALERT.showAlert("Invalid Input", "Password Key cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            if (!email.endsWith("@gmail.com")) {

                CUSTOMALERT.showAlert("Invalid Email", "Please enter a valid Gmail address (example: example@gmail.com)", Alert.AlertType.ERROR);

                return;

            }

            isAuthenticated = UserDAO.addUser(username, email, password, passwordKey);

            if (isAuthenticated) {

                CUSTOMALERT.showAlert("Success Signing Up", "Account Created Successfully", Alert.AlertType.INFORMATION);

                homeLayout.loadNavbar();

                homeLayout.updateStage();

                stage.close();

            } else {

                CUSTOMALERT.showAlert("Error Signing Up", "User already exists. Please try again", Alert.AlertType.ERROR);

            }

        });

        alreadyHaveAccountLabel.setId("alreadyHaveAccountLabel");

        VBox vbox=new VBox(20,signUpButton,alreadyHaveAccountLabel);

        signUpButton.setAlignment(Pos.CENTER);

        alreadyHaveAccountLabel.setAlignment(Pos.CENTER);

        vbox.setPadding(new Insets(20, 20, 20, 20));

        vbox.setAlignment(Pos.CENTER);

        GridPane gridPane = new GridPane();

        gridPane.setVgap(10);

        gridPane.setHgap(10);

        gridPane.setAlignment(Pos.CENTER);

        GridPane.setColumnSpan(signUpLabel, 2);

        gridPane.add(signUpLabel, 0, 0, 2, 1);

        gridPane.add(usernameLabel, 0, 1);

        gridPane.add(usernameField, 1, 1);

        gridPane.add(emailLabel, 0, 2);

        gridPane.add(emailField, 1, 2);

        gridPane.add(passwordLabel, 0, 3);

        gridPane.add(passwordField, 1, 3);

        gridPane.add(passwordKeyLabel, 0, 4);

        gridPane.add(passwordKeyField, 1, 4);

        gridPane.add(passwordRecoveryInfoLabel, 0, 5,2,1);

        gridPane.add(vbox, 1, 6);

        Scene signUpScene = new Scene(gridPane, 600, 500);

        signUpScene.getStylesheets().add(getClass().getResource("/styles/auth.css").toExternalForm());

        stage.setScene(signUpScene);

        stage.show();

    }

    private void switchToLoginView(Stage stage) {

        isSignUpView = false;

        Label titleLabel = new Label("Login");

        titleLabel.setId("login");

        Label signUpLabel = new Label("Don't Have an Account? Sign Up");

        signUpLabel.setId("signupLabel");

        signUpLabel.setOnMouseClicked(event -> switchToSignUpView(stage));

        Label forgotPasswordLabel = new Label("Forgot Password?");

        forgotPasswordLabel.setId("forgotPasswordLabel");

        forgotPasswordLabel.setOnMouseClicked(event -> loadForgotPasswordLayout(stage));

        Label usernameLabel = new Label("UserEmail:");

        TextField usernameField = new TextField();

        usernameField.setPromptText("Useremail");

        usernameField.setPrefHeight(50);

        usernameField.setPrefWidth(300);

        Label passwordLabel = new Label("Password:");

        PasswordField passwordField = new PasswordField();

        passwordField.setPromptText("Password");

        passwordField.setPrefHeight(50);

        passwordField.setPrefWidth(300);

        Button loginButton = new Button("Login");

        loginButton.setPrefWidth(150);

        loginButton.setOnAction(event -> {

            String username = usernameField.getText();

            String password = passwordField.getText();

*// Check for empty or null fields*

            if (username.isEmpty() || password.isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Username and Password cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            isAuthenticated = UserDAO.authenticateUser(username, password);

            if (isAuthenticated) {

                CUSTOMALERT.showAlert("Success Login", "Login Successful", Alert.AlertType.INFORMATION);

                homeLayout.loadNavbar();

                homeLayout.updateStage();

                stage.close();

            } else {

                CUSTOMALERT.showAlert("Error Login", "Invalid Credentials. Please try again", Alert.AlertType.ERROR);

            }

        });

        HBox usernameHBox = new HBox(10, usernameLabel, usernameField);

        usernameHBox.setAlignment(Pos.CENTER);

        HBox passwordHBox = new HBox(10, passwordLabel, passwordField);

        passwordHBox.setAlignment(Pos.CENTER);

        VBox loginLayout = new VBox(15);

        loginLayout.setAlignment(Pos.CENTER);

        loginLayout.getChildren().addAll(titleLabel, usernameHBox, passwordHBox, loginButton, signUpLabel, forgotPasswordLabel);

        Scene loginScene = new Scene(loginLayout, 400, 400);

        loginScene.getStylesheets().add(getClass().getResource("/styles/auth.css").toExternalForm());

        stage.setScene(loginScene);

    }

    private void switchToSignUpView(Stage stage) {

        isSignUpView = true;

        loadSignUpLayout();

    }

    private void loadForgotPasswordLayout(Stage stage) {

        Label titleLabel = new Label("Get Password");

        titleLabel.setId("getpassword");

        Label emailLabel = new Label("Email:");

        emailLabel.setPadding(new Insets(0,20,0,0));

        TextField emailField = new TextField();

        emailField.setPromptText("Email");

        emailField.setPrefHeight(50);

        emailField.setPrefWidth(300);

        Label passwordKeyLabel = new Label("Password Key:");

        PasswordField passwordKeyField = new PasswordField();

        passwordKeyField.setPromptText("Password Key");

        passwordKeyField.setPrefHeight(50);

        passwordKeyField.setPrefWidth(300);

        Button getPasswordButton = new Button("Get Password");

        getPasswordButton.setPrefWidth(150);

        getPasswordButton.setOnAction(event -> {

            String email = emailField.getText().trim();

            String passwordKey = passwordKeyField.getText().trim();

            if (email.isEmpty() || passwordKey.isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Email and Password Key cannot be empty.", Alert.AlertType.ERROR);

                return;

            }

            String password = UserDAO.getPasswordByEmail(email, passwordKey);

            if (password != null) {

                CUSTOMALERT.showAlert("Password", "Your Password is " + password, Alert.AlertType.INFORMATION);

                switchToLoginView(stage);

            } else {

                CUSTOMALERT.showAlert("Error Reset Password", "Invalid Email or Password Key. Please try again", Alert.AlertType.ERROR);

            }

        });

        HBox emailHBox = new HBox(50, emailLabel, emailField);

        emailHBox.setAlignment(Pos.CENTER);

        HBox passwordKeyHBox = new HBox(10, passwordKeyLabel, passwordKeyField);

        passwordKeyHBox.setAlignment(Pos.CENTER);

        VBox forgotPasswordLayout = new VBox(15);

        forgotPasswordLayout.setAlignment(Pos.CENTER);

        forgotPasswordLayout.getChildren().addAll(titleLabel, emailHBox, passwordKeyHBox, getPasswordButton);

        Scene forgotPasswordScene = new Scene(forgotPasswordLayout, 500, 400);

        forgotPasswordScene.getStylesheets().add(getClass().getResource("/styles/auth.css").toExternalForm());

        stage.setScene(forgotPasswordScene);

    }

    public void loadChangePasswordLayout() {

        Stage stage = new Stage();

*// Title Label*

        Label titleLabel = new Label("Change Password");

        titleLabel.setId("changePasswordTitle"); *// Use external style ID for title*

*// Old Password Label and Field*

        Label oldPasswordLabel = new Label("Old Password:");

        PasswordField oldPasswordField = new PasswordField();

        oldPasswordField.setPrefWidth(300);

        oldPasswordField.setPromptText("Old Password");

        oldPasswordField.setPrefHeight(50);

*// Email Label and Field*

        Label emailLabel = new Label("Email:");

        TextField emailField = new TextField();

        emailField.setPromptText("Enter Email");

        emailField.setPrefWidth(300);

        emailField.setPrefHeight(50);

        Label newPasswordLabel = new Label("New Password:");

        PasswordField newPasswordField = new PasswordField();

        newPasswordField.setPrefWidth(300);

        newPasswordField.setPromptText("New Password");

        newPasswordField.setPrefHeight(50);

*// Confirm New Password Label and Field*

        Label confirmNewPasswordLabel = new Label("Confirm New Password:");

        PasswordField confirmNewPasswordField = new PasswordField();

        confirmNewPasswordField.setPrefWidth(300);

        confirmNewPasswordField.setPromptText("Confirm New Password");

        confirmNewPasswordField.setPrefHeight(50);

*// Change Password Button*

        Button changePasswordButton = new Button("Change Password");

        changePasswordButton.setPrefWidth(200);

        changePasswordButton.setOnAction(event -> {

            String email = emailField.getText().trim();

            String oldPassword = oldPasswordField.getText().trim();

            String newPassword = newPasswordField.getText().trim();

            String confirmNewPassword = confirmNewPasswordField.getText().trim();

            if (email.isEmpty() || oldPassword.isEmpty() || newPassword.isEmpty() || confirmNewPassword.isEmpty()) {

                CUSTOMALERT.showAlert("Error", "All fields must be filled.", Alert.AlertType.ERROR);

                return;

            }

            if (newPassword.equals(confirmNewPassword)) {

                boolean isPasswordChanged = UserDAO.changePassword(email, oldPassword, newPassword);

                if (isPasswordChanged) {

                    CUSTOMALERT.showAlert("Password Change Success", "Your password has been changed successfully", Alert.AlertType.INFORMATION);

                    switchToLoginView(stage);

                } else {

                    CUSTOMALERT.showAlert("Error Changing Password", "Old password is incorrect. Please try again", Alert.AlertType.ERROR);

                }

            } else {

                CUSTOMALERT.showAlert("Password Mismatch", "New passwords do not match. Please try again", Alert.AlertType.ERROR);

            }

        });

        HBox oldPasswordHBox = new HBox(10, oldPasswordLabel, oldPasswordField);

        oldPasswordHBox.setAlignment(Pos.CENTER);

*// HBox for Email*

        HBox emailHBox = new HBox(50, emailLabel, emailField);

        emailHBox.setAlignment(Pos.CENTER);

*// HBox for New Password*

        HBox newPasswordHBox = new HBox(10, newPasswordLabel, newPasswordField);

        newPasswordHBox.setAlignment(Pos.CENTER);

*// HBox for Confirm New Password*

        HBox confirmNewPasswordHBox = new HBox(10, confirmNewPasswordLabel, confirmNewPasswordField);

        confirmNewPasswordHBox.setAlignment(Pos.CENTER);

*// VBox Layout*

        VBox changePasswordLayout = new VBox(15);

        changePasswordLayout.setAlignment(Pos.CENTER);

        changePasswordLayout.getChildren().addAll(titleLabel, emailHBox, oldPasswordHBox, newPasswordHBox, confirmNewPasswordHBox, changePasswordButton);

*// Scene*

        Scene changePasswordScene = new Scene(changePasswordLayout, 500, 400);

        changePasswordScene.getStylesheets().add(getClass().getResource("/styles/auth.css").toExternalForm());

        stage.setScene(changePasswordScene);

        stage.show();

    }

}

COINSCREENLAYOUT.JAVA :

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.models.ChartHistoricalData;

import com.example.crypto\_price\_tracker.models.CoinDetailsFetcher;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Objects;

public class CoinScreenLayout {

    public void loadLayout(String coinID, String coinName, String currency, Double perDayLow, Double perDayHigh) {

        Stage showScreen=CUSTOMALERT.showScreen("Fetching Coin Details. Please Wait...");

        showScreen.show();

        ArrayList<String> coinsDetails = new ArrayList<>();

        try {

            coinsDetails =  CoinDetailsFetcher.fetchCoinDetails(coinID, currency);

            ChartHistoricalData.fetchAndGenerateChart(coinID,currency);

        } catch (IOException e) {

            e.printStackTrace();

        } catch (Exception e) {

            throw new RuntimeException(e);

        }

        if (coinsDetails.isEmpty()) {

            CUSTOMALERT.showAlert("No Details", "Details about this coin are not available for now. Sorry for the inconvenience.", Alert.AlertType.ERROR);

            return;

        }

        Stage stage = new Stage();

        Image coinImage = new Image(coinsDetails.get(0));

        ImageView coinImageView = new ImageView(coinImage);

        coinImageView.setFitHeight(100);

        coinImageView.setFitWidth(100);

        coinImageView.setPreserveRatio(true);

        Label descriptionLabel = new Label(coinsDetails.get(2));

        descriptionLabel.setId("description");

        descriptionLabel.setWrapText(true);

        descriptionLabel.setPrefWidth(800);

        descriptionLabel.setPrefHeight(50);

        Image chartImage;

        try {

            chartImage = new Image(Objects.requireNonNull(getClass().getResource("/charts/historical\_coin\_price\_chart.png")).toString());

        } catch (Exception e) {

            CUSTOMALERT.showAlert("Chart Error", "Unable to load the chart image. Ensure the file is in the correct location.", Alert.AlertType.ERROR);

            return;

        }

        ImageView chartImageView = new ImageView(chartImage);

        chartImageView.setFitWidth(500);

        chartImageView.setFitHeight(300);

        chartImageView.setPreserveRatio(true);

        Button exitButton = new Button("Exit Coin");

        exitButton.setPrefWidth(150);

        exitButton.setAlignment(Pos.CENTER);

        exitButton.setId("exitButton");

        exitButton.setOnAction(e -> stage.close());

        VBox topSection = new VBox(10);

        topSection.setAlignment(Pos.CENTER);

        topSection.getChildren().addAll( coinImageView, descriptionLabel,exitButton);

        VBox chartSection = new VBox(20);

        chartSection.setAlignment(Pos.CENTER);

        chartSection.getChildren().add(chartImageView);

        HBox marketRankBox = new HBox(400);

        Label marketRankLabel = new Label("Market Rank");

        Label marketRank = new Label(coinsDetails.get(1));

        marketRankBox.getChildren().addAll(marketRankLabel, marketRank);

        HBox perDayLowBox = new HBox(400);

        Label perDayLowLabel = new Label("24hr Low");

        Label DayLow = new Label(String.valueOf(perDayLow));

        perDayLowBox.getChildren().addAll(perDayLowLabel, DayLow);

        HBox perDayHighBox = new HBox(400);

        Label perDayHighLabel = new Label("24hr High");

        Label dayHigh = new Label(String.valueOf(perDayHigh));

        perDayHighBox.getChildren().addAll(perDayHighLabel, dayHigh);

        HBox websiteLinkBox = new HBox(400);

        Label webLabel = new Label("Visit At");

        Label link = new Label(coinsDetails.get(3));

        websiteLinkBox.getChildren().addAll(webLabel, link);

        VBox bottomSection = new VBox(10);

        bottomSection.setId("bottomSection");

        bottomSection.setPadding(new Insets(0,0,0,315));

        bottomSection.setPrefWidth(500);

        bottomSection.setAlignment(Pos.CENTER);

        bottomSection.getChildren().addAll(marketRankBox, perDayLowBox, perDayHighBox, websiteLinkBox);

        VBox mainLayout = new VBox(30);

        mainLayout.setPadding(new Insets(20));

        mainLayout.setAlignment(Pos.CENTER);

        mainLayout.getChildren().addAll(topSection, chartSection, bottomSection);

        Scene scene = new Scene(mainLayout);

        scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/styles/coin.css")).toExternalForm());

        stage.setTitle("Coin Details - " + coinName);

        stage.setWidth(1200);

        stage.setHeight(730);

        stage.setScene(scene);

        stage.centerOnScreen();

        stage.setFullScreen(true);

        stage.show();

        showScreen.close();

    }

}

CRYPTO NEWSLAYOUT.JAVA :

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.models.CryptoNewsFetcher;

import com.example.crypto\_price\_tracker.models.News;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.ScrollPane;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.FlowPane;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.util.ArrayList;

public class CryptoNewsLayout {

    public void loadNewsLayout() {

        Stage showScreen = CUSTOMALERT.showScreen("Fetching News. Please Wait...");

        showScreen.show();

        CryptoNewsFetcher cryptoNewsFetcher = new CryptoNewsFetcher();

        ArrayList<News> newsItems = cryptoNewsFetcher.getNewsList();

        Label header1 = new Label("In Crypto We Trust");

        header1.setId("header1");

        header1.setAlignment(Pos.CENTER);

        Label header2 = new Label("Until Market Adjusts");

        header2.setId("header2");

        header2.setAlignment(Pos.CENTER);

        Button exitButton = new Button("Exit News");

      exitButton.setId("exitButton");

        FlowPane newsContainer = new FlowPane();

        newsContainer.setId("newsContainer");

        newsContainer.setVgap(20);

        newsContainer.setHgap(20);

        newsContainer.setPadding(new Insets(20, 20, 20, 100));

        newsItems.forEach(newsItem -> {

            String title = newsItem.getTitle() != null ? newsItem.getTitle() : "No Title Available";

            String descriptionText = newsItem.getDescription() != null ? newsItem.getDescription() : "No Description Available";

            String imageUrl = getClass().getResource("/defaultNewsImage.jpg").toExternalForm();

            String source = newsItem.getSourceName() != null ? "Source: " + newsItem.getSourceName() : "Source: Unknown";

            String link = newsItem.getLink() != null ? "Link: " + newsItem.getLink() : "Link: Not Available";

            VBox newsBox = new VBox(10);

            newsBox.setId("newsBox");

            newsBox.setPrefWidth(300);

            Image newsImage = new Image(imageUrl);

            ImageView newsView = new ImageView(newsImage);

            newsView.setFitWidth(300);

            newsView.setFitHeight(200);

            Label titleLabel = new Label(title);

            titleLabel.setId("newsTitle");

            titleLabel.setWrapText(true);

            titleLabel.setPrefWidth(300);

            titleLabel.setAlignment(Pos.CENTER\_LEFT);

            titleLabel.setPadding(new Insets(10));

            Label contentLabel = new Label(descriptionText);

            contentLabel.setId("newsDescription");

            contentLabel.setStyle("-fx-text-fill: #ccc;");

            contentLabel.setWrapText(true);

            contentLabel.setPrefHeight(150);

            contentLabel.setPadding(new Insets(10));

            Label sourceLabel = new Label(source);

            sourceLabel.setId("newsSource");

            sourceLabel.setStyle("-fx-text-fill: white;");

            sourceLabel.setPadding(new Insets(10));

            Label linkLabel = new Label(link);

            linkLabel.setId("newsLink");

            linkLabel.setStyle("-fx-text-fill: white;");

            linkLabel.setPadding(new Insets(10));

            newsBox.getChildren().addAll(newsView, titleLabel, contentLabel, sourceLabel, linkLabel);

            newsContainer.getChildren().add(newsBox);

        });

        ScrollPane scrollPane = new ScrollPane(newsContainer);

        scrollPane.setFitToWidth(true);

        scrollPane.setPrefHeight(400);

        scrollPane.setId("newsScrollPane");

        VBox mainLayout = new VBox(10);

        mainLayout.setPadding(new Insets(20));

        mainLayout.setAlignment(Pos.CENTER);

        mainLayout.getChildren().addAll(header1, header2, exitButton, scrollPane);

        Scene scene = new Scene(mainLayout);

        scene.getStylesheets().add(getClass().getResource("/styles/news.css").toExternalForm());

        Stage stage = new Stage();

        stage.setTitle("Crypto News");

        stage.setScene(scene);

        stage.setFullScreen(true);

        exitButton.setOnMouseClicked(event -> {

            stage.close();

        });

        showScreen.close();

        stage.show();

    }

}

HOME LAYOUT.JAVA

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.models.Coin;

import com.example.crypto\_price\_tracker.models.CoinsFetcher;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import javafx.beans.property.SimpleObjectProperty;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.concurrent.Task;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*\**;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.*\**;

import javafx.stage.Stage;

import java.util.Map;

import java.util.concurrent.atomic.AtomicReference;

public class homeLayout {

    private static final Map<String, String> CURRENCY\_SYMBOLS = Map.of(

            "USD", "$ ",

            "EUR", "€ ",

            "INR", "₹ "

    );

    private static HBox navbar=new HBox();

    public static AtomicReference<String>  selectedCurrency = new AtomicReference<>();

    public static  VBox mainLayout = new VBox();

    public static VBox welcomeSection=new VBox();

    public static HBox searchBox=new HBox();

    private static Stage stage=new Stage();

    public static Button signUpButton;

    private static   TableView<Coin> coinTable=new TableView<Coin>();

    private static ObservableList<Coin> fetchDatafromAPI(String currency) {

        ObservableList<Coin> fetchedCoinsList = FXCollections.observableArrayList();

        fetchedCoinsList = CoinsFetcher.fetchCoinsData(currency);

        return fetchedCoinsList;

    }

    public void loadLayout() {

        loadNavbar();

        HBox.setHgrow(new Region(), Priority.ALWAYS);

        Label title = new Label("Largest");

        title.setId("title");

        Label subtitle = new Label("Crypto Marketplace");

        subtitle.setId("subtitle");

        Label welcomeText = new Label("Welcome to the world's largest cryptocurrency marketplace.");

        welcomeText.setId("welcomeText");

        Label signupPrompt = new Label("Signup to explore more about crypto.");

        signupPrompt.setId("signupPrompt");

        welcomeSection.getChildren().addAll(title, subtitle, welcomeText, signupPrompt);

        welcomeSection.setAlignment(Pos.CENTER);

        welcomeSection.setSpacing(5);

        TextField searchField = new TextField();

        searchField.setId("searchField");

        searchField.setPromptText("Search Crypto...");

        Button searchButton = new Button("Search");

        searchButton.setId("searchButton");

         searchBox.getChildren().addAll(searchField, searchButton);

        searchBox.setAlignment(Pos.CENTER);

        searchBox.setSpacing(10);

        ObservableList<Coin> fetchedCoinsList = FXCollections.observableArrayList();

        fetchedCoinsList = fetchDatafromAPI(selectedCurrency.get());

        ObservableList<Coin> finalFetchedCoinsList = fetchedCoinsList;

        searchField.textProperty().addListener((observable, oldValue, newValue) -> {

            if (newValue.trim().isEmpty()) {

                updateTable(selectedCurrency.get(), finalFetchedCoinsList, mainLayout);

            } else {

                ObservableList<Coin> filteredCoins = FXCollections.observableArrayList();

                for (Coin coin : finalFetchedCoinsList) {

                    if (coin.getCoinName().toLowerCase().contains(newValue.toLowerCase().trim())) {

                        filteredCoins.add(coin);

                    }

                }

                if (!filteredCoins.isEmpty()) {

                    updateTable(selectedCurrency.get(), filteredCoins, mainLayout);

                }

            }

        });

        searchButton.setOnAction(e -> {

            String searchQuery = searchField.getText().toLowerCase().trim();

            ObservableList<Coin> filteredCoins = FXCollections.observableArrayList();

            for (Coin coin : finalFetchedCoinsList) {

                if (coin.getCoinName().toLowerCase().contains(searchQuery)) {

                    filteredCoins.add(coin);

                }

            }

            if (!filteredCoins.isEmpty()) {

                updateTable(selectedCurrency.get(), filteredCoins, mainLayout);

            } else {

                CUSTOMALERT.showAlert("Not Found", "The entered Coin is not Found", Alert.AlertType.ERROR);

            }

        });

        mainLayout.getChildren().addAll(navbar, welcomeSection, searchBox);

        updateTable(selectedCurrency.get(), fetchedCoinsList, mainLayout);

        mainLayout.setSpacing(10);

        mainLayout.setPadding(new Insets(10,20,20,20));

        mainLayout.setAlignment(Pos.TOP\_CENTER);

        Scene scene = new Scene(mainLayout, 1200, 700);

        scene.getStylesheets().add(getClass().getResource("/styles/home.css").toExternalForm());

        stage.setTitle("TokenTribe");

        stage.setScene(scene);

        stage.setFullScreen(true);

        stage.show();

    }

    private  TableView<Coin> createCoinTable(ObservableList<Coin> fetchedCoinsList, String selectedCurrency) {

        TableView<Coin> coinTable = new TableView<>();

        coinTable.setPrefWidth(800);

        coinTable.setPrefHeight(300);

        coinTable.setEditable(false);

        coinTable.setColumnResizePolicy(TableView.CONSTRAINED\_RESIZE\_POLICY);

        TableColumn<Coin, String> nameColumn = new TableColumn<>("Coin Name");

        nameColumn.setPrefWidth(200);

        nameColumn.setCellValueFactory(cellData -> cellData.getValue().coinNameProperty());

        nameColumn.setStyle("-fx-alignment: CENTER-LEFT; -fx-padding: 0 0 0 80;");

        TableColumn<Coin, ImageView> imageColumn = new TableColumn<>("Coin");

        imageColumn.setPrefWidth(100);

        imageColumn.setStyle("-fx-alignment: CENTER-LEFT; -fx-padding: 0 0 0 80;");

        imageColumn.setCellValueFactory(cellData -> {

            ImageView imageView = new ImageView();

            imageView.setFitHeight(30);

            imageView.setFitWidth(30);

            Task<Image> loadImageTask = new Task<>() {

                @Override

                protected Image call() {

                    return new Image(cellData.getValue().getCoinImageUrl(), false);

                }

            };

            loadImageTask.setOnSucceeded(e -> imageView.setImage(loadImageTask.getValue()));

            loadImageTask.setOnFailed(e -> imageView.setImage(new Image(getClass().getResource("/default-coin.jpeg").toExternalForm())));

            new Thread(loadImageTask).start();

            return new SimpleObjectProperty<>(imageView);

        });

        TableColumn<Coin, Double> priceColumn = new TableColumn<>("Current Price");

        priceColumn.setPrefWidth(150);

        priceColumn.setStyle("-fx-alignment: CENTER-LEFT; -fx-padding: 0 0 0 80;");

        priceColumn.setCellValueFactory(cellData -> cellData.getValue().currentPriceProperty().asObject());

        priceColumn.setCellFactory(col -> {

            return new TableCell<Coin, Double>() {

                @Override

                protected void updateItem(Double item, boolean empty) {

*super*.updateItem(item, empty);

                    if (empty || item == null) {

                        setText(null);

                        setStyle("");

                    } else {

                        String currencySymbol = CURRENCY\_SYMBOLS.getOrDefault(selectedCurrency, "$ ");

                        setText(String.format("%s%.1f", currencySymbol, item));

                        setStyle("-fx-alignment: CENTER;");

                    }

                }

            };

        });

        coinTable.setOnMouseClicked(event -> {

            if(Authentication.isAuthenticated){

                Coin selectedCoin = coinTable.getSelectionModel().getSelectedItem();

                if (selectedCoin != null) {

                    System.out.println("Coin ID: " + selectedCoin.getCoinID());

                    System.out.println("Selected Currency: " + selectedCurrency);

                    CoinScreenLayout coinLayout = new CoinScreenLayout();

                    coinLayout.loadLayout(selectedCoin.getCoinID(), selectedCoin.getCoinName(), selectedCurrency, selectedCoin.getPerDayLow(), selectedCoin.perDayLowProperty().get());

                }

            }else{

                CUSTOMALERT.showAlert("Authentication Required", "You need to be authenticated to details of Coin", Alert.AlertType.ERROR);

            }

        });

        TableColumn<Coin, Double> changeColumn = new TableColumn<>("24h Change");

        changeColumn.setPrefWidth(150);

        changeColumn.setCellValueFactory(cellData -> cellData.getValue().percentageChange24hProperty().asObject());

        changeColumn.setCellFactory(col -> {

            return new TableCell<Coin, Double>() {

                @Override

                protected void updateItem(Double item, boolean empty) {

*super*.updateItem(item, empty);

                    if (empty || item == null) {

                        setText(null);

                        setStyle("");

                    } else {

                        setText(String.format("%.2f%%", item));

                        if (item < 0) {

                            setStyle("-fx-text-fill: red; -fx-alignment: CENTER; -fx-padding: 0 60 0 0;");

                        } else if (item > 0) {

                            setStyle("-fx-text-fill: green; -fx-alignment: CENTER; -fx-padding: 0 60 0 0;");

                        } else {

                            setStyle("-fx-text-fill: gray; -fx-alignment: CENTER; -fx-padding: 0 60 0 0;");

                        }

                    }

                }

            };

        });

        TableColumn<Coin, Double> marketCapColumn = new TableColumn<>("Market Cap");

        marketCapColumn.setPrefWidth(200);

        marketCapColumn.setStyle("-fx-alignment: CENTER-LEFT; -fx-padding: 0 0 0 30;");

        marketCapColumn.setCellValueFactory(cellData -> cellData.getValue().marketCapProperty().asObject());

        marketCapColumn.setCellFactory(col -> {

            return new TableCell<Coin, Double>() {

                @Override

                protected void updateItem(Double item, boolean empty) {

*super*.updateItem(item, empty);

                    if (empty || item == null) {

                        setText(null);

                        setStyle("");

                    } else {

                        setText(String.format("%.2f%%", item));

                    }

                }

            };

        });

        coinTable.getColumns().addAll(nameColumn, imageColumn, priceColumn, changeColumn, marketCapColumn);

        coinTable.setItems(fetchedCoinsList);

        return coinTable;

    }

    private static void updateTable(String selectedCurrency, VBox mainLayout) {

        ObservableList<Coin> fetchedCoinsList = fetchDatafromAPI(selectedCurrency);

        mainLayout.getChildren().removeIf(node -> node instanceof TableView);

        homeLayout homeLayout=new homeLayout();

        TableView<Coin> coinTable =homeLayout.createCoinTable(fetchedCoinsList, selectedCurrency);

        mainLayout.getChildren().add(coinTable);

    }

    private void updateTable(String selectedCurrency, ObservableList fetchedCoinsList, VBox mainLayout) {

        mainLayout.getChildren().removeIf(node -> node instanceof TableView);

        coinTable = createCoinTable(fetchedCoinsList, selectedCurrency);

        mainLayout.getChildren().add(coinTable);

    }

    public static void loadNavbar(){

        Label logo = new Label("Coin Flipper");

        logo.setId("logo");

        Button newsButton = new Button("Get News");

        newsButton.setOnAction(e -> {

            if(Authentication.isAuthenticated){

                   new CryptoNewsLayout().loadNewsLayout();

            }else{

                CUSTOMALERT.showAlert("Sign In","Please sign in to read News",Alert.AlertType.ERROR);

            }

        });

        Button portfolioButton = new Button("Portfolio");

        portfolioButton.setOnAction(e -> {

            if(Authentication.isAuthenticated){

                Portfolio.loadPortfolio();

            }else{

                CUSTOMALERT.showAlert("Authentication Required","You need to be authenticated to view Portfolio",Alert.AlertType.ERROR);

            }

        });

        ComboBox<String> currencyList = new ComboBox<>();

        currencyList.setId("currencyList");

        currencyList.getItems().addAll("USD", "EUR", "INR");

        currencyList.setValue("USD");

        selectedCurrency = new AtomicReference<>(currencyList.getValue());

        currencyList.valueProperty().addListener((obs, oldVal, newVal) -> {

            selectedCurrency.set(newVal);

            updateTable(selectedCurrency.get(), mainLayout);

        });

        signUpButton = new Button();

        if(Authentication.isAuthenticated){

            signUpButton.setText("Sign Out");

        }else{

            signUpButton.setText("Sign Up");

        }

        signUpButton.setOnAction(e->{

            if(!Authentication.isAuthenticated) {

                Authentication auth = new Authentication();

                auth.loadSignUpLayout();

            }else{

                Authentication.isAuthenticated = false;

                loadNavbar();

                updateStage();

            };

        });

        Button changePasswordButton =new Button("Change Password");

        changePasswordButton.setId("changePasswordButton");

        changePasswordButton.setOnAction(e->{

            new Authentication().loadChangePasswordLayout();

        });

        Button exitButton = new Button("Exit");

        exitButton.setOnAction(e->{

            stage.close();

        });

        navbar.getChildren().clear();

        if(Authentication.isAuthenticated){

            navbar.getChildren().addAll( newsButton, portfolioButton,  currencyList, signUpButton,changePasswordButton, exitButton);

        }else{

            navbar.getChildren().addAll(logo,new Region(), newsButton, portfolioButton, currencyList, signUpButton, exitButton);

        }

        navbar.setId("navbar");

        navbar.setSpacing(50);

        navbar.setAlignment(Pos.CENTER);

    }

    public static void updateStage(){

        mainLayout.getChildren().clear();

        mainLayout.getChildren().addAll(navbar, welcomeSection, searchBox,coinTable);

    }

}

POPUPS.JAVA

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.models.TradeManager;

import com.example.crypto\_price\_tracker.models.TradeRecord;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import com.example.crypto\_price\_tracker.util.PortfolioOperationHandler;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.*\**;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.stage.FileChooser;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.File;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

public class PopUps {

    public static ArrayList<String> coins = new ArrayList<>(List.of(

            "Bitcoin", "Ethereum", "Litecoin", "Cardano", "Ripple", "Polkadot",

            "Dogecoin", "Solana", "Chainlink", "Binance Coin", "Shiba Inu", "Uniswap",

            "Avalanche", "Terra", "Litecoin", "Polygon", "VeChain", "Filecoin",

            "Cosmos", "Stellar", "Aave"

    ));

    public static void addRecordPopUp() {

        Stage popUpStage = new Stage();

        popUpStage.initModality(Modality.APPLICATION\_MODAL);

        popUpStage.setTitle("Add Trade Record");

        DatePicker datePicker = new DatePicker();

        datePicker.setPromptText("Select Trade Date");

        ComboBox<String> coinNameComboBox = new ComboBox<>();

        coinNameComboBox.getItems().addAll(coins);

        coinNameComboBox.setPromptText("Select Coin");

        ComboBox<String> tradeTypeComboBox = new ComboBox<>();

        tradeTypeComboBox.getItems().addAll("Buy", "Sell");

        tradeTypeComboBox.setPromptText("Select Trade Type");

        TextField quantityField = new TextField();

        quantityField.setPromptText("Enter Quantity");

        TextField priceField = new TextField();

        priceField.setPromptText("Enter Price per Coin (in $)");

        Label sellPriceLabel = new Label("Sell Price:");

        sellPriceLabel.setVisible(false);

        TextField sellPriceField = new TextField();

        sellPriceField.setPromptText("Enter Sell Price");

        sellPriceField.setVisible(false);

        Label quantityLabel = new Label("Quantity Bought:");

        Label priceLabel = new Label("Price per Coin (in $):");

*// Update form when trade type changes*

        tradeTypeComboBox.setOnAction(e -> {

            if ("Sell".equals(tradeTypeComboBox.getValue())) {

                sellPriceLabel.setVisible(true);

                sellPriceField.setVisible(true);  *// Show Sell Price field*

                quantityLabel.setText("Quantity Sold:");  *// Update to "Quantity Sold"*

            } else {

                sellPriceField.setVisible(false);

                sellPriceField.setVisible(false);  *// Hide Sell Price field*

                quantityLabel.setText("Quantity Bought:");  *// Update to "Quantity Bought"*

            }

        });

        Button saveButton = new Button("Save");

        saveButton.setOnAction(e -> {

            if (datePicker.getValue() == null) {

                CUSTOMALERT.showAlert("Error", "Please select a valid trade date.", Alert.AlertType.ERROR);

                return;

            }

            if (coinNameComboBox.getValue() == null || coinNameComboBox.getValue().isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please select a coin name.", Alert.AlertType.ERROR);

                return;

            }

            if (tradeTypeComboBox.getValue() == null || tradeTypeComboBox.getValue().isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please select a trade type (Buy/Sell).", Alert.AlertType.ERROR);

                return;

            }

            if (quantityField.getText().isEmpty() || !quantityField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                CUSTOMALERT.showAlert("Error", "Please enter a valid quantity (numeric).", Alert.AlertType.ERROR);

                return;

            }

            if (priceField.getText().isEmpty() || !priceField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                CUSTOMALERT.showAlert("Error", "Please enter a valid price (numeric).", Alert.AlertType.ERROR);

                return;

            }

            if (datePicker.getValue().isAfter(LocalDate.now())) {

                CUSTOMALERT.showAlert("Error", "Trade date cannot be greater than today's date.", Alert.AlertType.ERROR);

                return;

            }

            LocalDate tradeDate = datePicker.getValue();

            String coinName = coinNameComboBox.getValue();

            String tradeType = tradeTypeComboBox.getValue();

            double quantity = Double.parseDouble(quantityField.getText());

            double price = Double.parseDouble(priceField.getText());

            double sellPrice = 0;

            if ("Sell".equals(tradeType)) {

                if (sellPriceField.getText().isEmpty() || !sellPriceField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                    CUSTOMALERT.showAlert("Error", "Please enter a valid sell price (numeric).", Alert.AlertType.ERROR);

                    return;

                }

                sellPrice = Double.parseDouble(sellPriceField.getText());

            }

            if(sellPrice>0){

                TradeManager.addRecord(new TradeRecord(tradeDate,coinName,tradeType,quantity,price,sellPrice));

            }else{

                TradeManager.addRecord(new TradeRecord(tradeDate, coinName, tradeType, quantity, price));

            }

            CUSTOMALERT.showAlert("Record Added", "Your new record has been added successfully.", Alert.AlertType.INFORMATION);

            popUpStage.close();

            Portfolio.loadPortfolio();

        });

        saveButton.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-font-weight: bold;");

        Button cancelButton = new Button("Cancel");

        cancelButton.setStyle("-fx-background-color: #f44336; -fx-text-fill: white; -fx-font-weight: bold;");

        cancelButton.setOnAction(e -> popUpStage.close());

*// Layout for the form*

        GridPane formGrid = new GridPane();

        formGrid.setHgap(10);

        formGrid.setVgap(10);

        formGrid.setPadding(new Insets(15));

        formGrid.add(new Label("Trade Date:"), 0, 0);

        formGrid.add(datePicker, 1, 0);

        formGrid.add(new Label("Coin Name:"), 0, 1);

        formGrid.add(coinNameComboBox, 1, 1);

        formGrid.add(new Label("Trade Type:"), 0, 2);

        formGrid.add(tradeTypeComboBox, 1, 2);

        formGrid.add(quantityLabel, 0, 3);

        formGrid.add(quantityField, 1, 3);

        formGrid.add(priceLabel, 0, 4);

        formGrid.add(priceField, 1, 4);

        formGrid.add(sellPriceLabel, 0, 5);

        formGrid.add(sellPriceField, 1, 5);

*// Button layout*

        HBox buttonBox = new HBox(10, saveButton, cancelButton);

        buttonBox.setPadding(new Insets(10));

        buttonBox.setStyle("-fx-alignment: center;");

*// Main layout*

        VBox mainLayout = new VBox(10, formGrid, buttonBox);

        mainLayout.setPadding(new Insets(15));

        mainLayout.setStyle("-fx-background-color: #f9f9f9; -fx-border-color: #cccccc; -fx-border-width: 2px; -fx-border-radius: 10px; -fx-background-radius: 10px;");

*// Set the scene and show the pop-up*

        Scene scene = new Scene(mainLayout, 400, 400);

        popUpStage.setScene(scene);

        popUpStage.showAndWait();

    }

    public static void deleteRecordPopUp() {

        System.out.println("Entered in Method");

        Stage stage = new Stage();

*// Create UI components*

        VBox vbox = new VBox(20);

        vbox.setPadding(new Insets(10));

        vbox.setStyle("-fx-background-color: #f9f9f9; -fx-border-color: #cccccc; -fx-border-width: 2px; -fx-border-radius: 10px; -fx-background-radius: 10px;");

        Label label = new Label("Enter ID of Trade to Delete:");

        TextField textField = new TextField();

        textField.setPromptText("Enter ID of Trade to Delete");

        Button deleteButton = new Button("Delete");

*// Set button action*

        deleteButton.setOnAction(e -> {

            String input = textField.getText().trim();

*// Validate input*

            if (input.isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please enter the ID of the trade to delete.", Alert.AlertType.ERROR);

                return;

            }

            try {

                int id = Integer.parseInt(input);

                if (id < 0) {

                    CUSTOMALERT.showAlert("Error", "ID cannot be negative.", Alert.AlertType.ERROR);

                    return;

                }

*// Call the delete method*

                boolean isDeleted = TradeManager.deleteRecord(id);

                if (isDeleted) {

                    CUSTOMALERT.showAlert("Success", "Trade deleted successfully.", Alert.AlertType.INFORMATION);

                    textField.clear();

                    stage.close();

                    Portfolio.loadPortfolio();

                } else {

                    CUSTOMALERT.showAlert("Error", "Trade with the given ID does not exist.", Alert.AlertType.ERROR);

                }

            } catch (NumberFormatException ex) {

                CUSTOMALERT.showAlert("Error", "Please enter a valid numeric ID.", Alert.AlertType.ERROR);

            }

        });

*// Add components to VBox*

        vbox.getChildren().addAll(label, textField, deleteButton);

*// Set up the stage*

        Scene scene = new Scene(vbox, 300, 200);

        stage.setTitle("Delete Trade");

        stage.setScene(scene);

        System.out.println("Stage show");

        stage.show();

    }

    public static void getRecordPop() {

        Stage stage = new Stage();

        stage.setTitle("Retrieve Trade Record");

*// Create UI components*

        VBox vbox = new VBox(20);

        vbox.setPadding(new Insets(10));

        vbox.setStyle("-fx-background-color: #f9f9f9; -fx-border-color: #cccccc; -fx-border-width: 2px; -fx-border-radius: 10px; -fx-background-radius: 10px;");

        Label label = new Label("Enter ID of Trade to Retrieve:");

        TextField textField = new TextField();

        textField.setPromptText("Enter ID of Trade to Retrieve");

        Button getButton = new Button("Get");

        getButton.setOnAction(e -> {

            String input = textField.getText().trim();

*// Validate input*

            if (input.isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please enter the ID of the trade to retrieve.", Alert.AlertType.ERROR);

                return;

            }

            try {

                int recordId = Integer.parseInt(input); *// Ensure it's a number*

                TradeRecord foundRecord = null;

*// Search for the record*

                for (TradeRecord record : PortfolioOperationHandler.tradeRecords) {

                    if (record.getRecordId() == recordId) {

                        foundRecord = record;

                        break;

                    }

                }

                if (foundRecord != null) {

                    updateRecordPop(foundRecord);

                } else {

                    CUSTOMALERT.showAlert("Not Found", "No trade record found with ID: " + recordId, Alert.AlertType.WARNING);

                }

            } catch (NumberFormatException ex) {

                CUSTOMALERT.showAlert("Invalid Input", "Please enter a valid number for the trade ID.", Alert.AlertType.ERROR);

            }

        });

*// Add components to the VBox*

        vbox.getChildren().addAll(label, textField, getButton);

*// Setup scene and stage*

        Scene scene = new Scene(vbox, 400, 200);

        stage.setScene(scene);

        stage.show();

    }

    public static void updateRecordPop(TradeRecord record){

        Stage popUpStage = new Stage();

        popUpStage.initModality(Modality.APPLICATION\_MODAL);

        popUpStage.setTitle("Add Trade Record");

        DatePicker datePicker = new DatePicker();

        datePicker.setValue(record.getTradeDate());

        datePicker.setPromptText("Select Trade Date");

        ComboBox<String> coinNameComboBox = new ComboBox<>();

        coinNameComboBox.getItems().addAll(coins);

        coinNameComboBox.setValue(record.getCoinName());

        coinNameComboBox.setPromptText("Select Coin");

        ComboBox<String> tradeTypeComboBox = new ComboBox<>();

        tradeTypeComboBox.getItems().addAll("Buy", "Sell");

        tradeTypeComboBox.setValue(record.getTradeType());

        tradeTypeComboBox.setPromptText("Select Trade Type");

        TextField quantityField = new TextField();

        quantityField.setText(String.valueOf(record.getQuantity()));

        quantityField.setPromptText("Enter Quantity");

        TextField priceField = new TextField();

        priceField.setText(String.valueOf(record.getBuyPrice()));

        priceField.setPromptText("Enter Price per Coin (in $)");

        Label sellPriceLabel = new Label("Sell Price:");

        sellPriceLabel.setVisible(true);

        TextField sellPriceField = new TextField();

        sellPriceField.setText(String.valueOf(record.getSellPrice()));

        sellPriceField.setPromptText("Enter Sell Price");

        sellPriceField.setVisible(false);

        if(!record.getTradeType().equalsIgnoreCase("Buy")){

            sellPriceLabel.setVisible(true);

            sellPriceField.setVisible(true);

        }

        Label quantityLabel = new Label("Quantity Bought:");

        Label priceLabel = new Label("Price per Coin (in $):");

        tradeTypeComboBox.setOnAction(e -> {

            if ("Sell".equals(tradeTypeComboBox.getValue())) {

                sellPriceLabel.setVisible(true);

                sellPriceField.setVisible(true);

                quantityLabel.setText("Quantity Sold:");

            } else {

                sellPriceField.setVisible(false);

                sellPriceField.setVisible(false);

                quantityLabel.setText("Quantity Bought:");

            }

        });

        Button saveButton = new Button("Save");

        saveButton.setOnAction(e -> {

            if (datePicker.getValue() == null) {

                CUSTOMALERT.showAlert("Error", "Please select a valid trade date.", Alert.AlertType.ERROR);

                return;

            }

            if (coinNameComboBox.getValue() == null || coinNameComboBox.getValue().isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please select a coin name.", Alert.AlertType.ERROR);

                return;

            }

            if (tradeTypeComboBox.getValue() == null || tradeTypeComboBox.getValue().isEmpty()) {

                CUSTOMALERT.showAlert("Error", "Please select a trade type (Buy/Sell).", Alert.AlertType.ERROR);

                return;

            }

            if (quantityField.getText().isEmpty() || !quantityField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                CUSTOMALERT.showAlert("Error", "Please enter a valid quantity (numeric).", Alert.AlertType.ERROR);

                return;

            }

            if (priceField.getText().isEmpty() || !priceField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                CUSTOMALERT.showAlert("Error", "Please enter a valid price (numeric).", Alert.AlertType.ERROR);

                return;

            }

            if (datePicker.getValue().isAfter(LocalDate.now())) {

                CUSTOMALERT.showAlert("Error", "Trade date cannot be greater than today's date.", Alert.AlertType.ERROR);

                return;

            }

            LocalDate tradeDate = datePicker.getValue();

            String coinName = coinNameComboBox.getValue();

            String tradeType = tradeTypeComboBox.getValue();

            double quantity = Double.parseDouble(quantityField.getText());

            double price = Double.parseDouble(priceField.getText());

            double sellPrice = 0;

            if ("Sell".equals(tradeType)) {

                if (sellPriceField.getText().isEmpty() || !sellPriceField.getText().matches("[0-9]\*\\.?[0-9]+")) {

                    CUSTOMALERT.showAlert("Error", "Please enter a valid sell price (numeric).", Alert.AlertType.ERROR);

                    return;

                }

                sellPrice = Double.parseDouble(sellPriceField.getText());

            }

            TradeRecord updatedRecord=null;

            if(sellPrice>0){

                updatedRecord = new TradeRecord(tradeDate,coinName,tradeType,quantity,price,sellPrice);

            }else if(sellPrice==0){

                updatedRecord = new TradeRecord(tradeDate, coinName, tradeType, quantity, price);

            }

            TradeManager.updateRecord(updatedRecord.getTradeDate(),updatedRecord.getCoinName(),updatedRecord.getTradeType(),updatedRecord.getQuantity(),updatedRecord.getBuyPrice(),updatedRecord.getSellPrice(),updatedRecord.getTotalValue(),updatedRecord.getProfitOrLoss(),record.getRecordId());

            CUSTOMALERT.showAlert("Record Updated", "Your new record has been added successfully.", Alert.AlertType.INFORMATION);

            popUpStage.close();

            Portfolio.loadPortfolio();

        });

        saveButton.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-font-weight: bold;");

        Button cancelButton = new Button("Cancel");

        cancelButton.setStyle("-fx-background-color: #f44336; -fx-text-fill: white; -fx-font-weight: bold;");

        cancelButton.setOnAction(e -> popUpStage.close());

*// Layout for the form*

        GridPane formGrid = new GridPane();

        formGrid.setHgap(10);

        formGrid.setVgap(10);

        formGrid.setPadding(new Insets(15));

        formGrid.add(new Label("Trade Date:"), 0, 0);

        formGrid.add(datePicker, 1, 0);

        formGrid.add(new Label("Coin Name:"), 0, 1);

        formGrid.add(coinNameComboBox, 1, 1);

        formGrid.add(new Label("Trade Type:"), 0, 2);

        formGrid.add(tradeTypeComboBox, 1, 2);

        formGrid.add(quantityLabel, 0, 3);

        formGrid.add(quantityField, 1, 3);

        formGrid.add(priceLabel, 0, 4);

        formGrid.add(priceField, 1, 4);

        formGrid.add(sellPriceLabel, 0, 5);

        formGrid.add(sellPriceField, 1, 5);

*// Button layout*

        HBox buttonBox = new HBox(10, saveButton, cancelButton);

        buttonBox.setPadding(new Insets(10));

        buttonBox.setStyle("-fx-alignment: center;");

*// Main layout*

        VBox mainLayout = new VBox(10, formGrid, buttonBox);

        mainLayout.setPadding(new Insets(15));

        mainLayout.setStyle("-fx-background-color: #f9f9f9; -fx-border-color: #cccccc; -fx-border-width: 2px; -fx-border-radius: 10px; -fx-background-radius: 10px;");

*// Set the scene and show the pop-up*

        Scene scene = new Scene(mainLayout, 400, 400);

        popUpStage.setScene(scene);

        popUpStage.showAndWait();

    }

}

PORTFOLIO.JAVA :

package com.example.crypto\_price\_tracker.layouts;

import com.example.crypto\_price\_tracker.models.TradeManager;

import com.example.crypto\_price\_tracker.models.User;

import com.example.crypto\_price\_tracker.util.PortfolioOperationHandler;

import com.example.crypto\_price\_tracker.util.UserDAO;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.ScrollPane;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.*\**;

import javafx.stage.Stage;

public class Portfolio {

    private static Stage stage;

    public static void loadPortfolio() {

        if (stage == null) {

            stage = new Stage();

            stage.setFullScreen(true);

            stage.setTitle("Portfolio");

        }

        if (stage.isShowing()) {

            stage.close();

        }

        HBox mainLayout = new HBox();

        User currentUser = UserDAO.getCurrentUser();

        VBox sideBar = loadSideBar(currentUser);

        VBox portfolioSection = loadPortfolioSection();

        portfolioSection.setPadding(new Insets(20));

        sideBar.setMinWidth(250);

        portfolioSection.setMinWidth(1000);

        mainLayout.getChildren().clear();

        mainLayout.getChildren().addAll(sideBar, portfolioSection);

        Scene scene = new Scene(mainLayout,800,800);

        scene.getStylesheets().add(Portfolio.class.getResource("/styles/portfolio.css").toExternalForm());

        stage.setScene(scene);

        stage.setX(200);

        stage.setY(200);

        stage.setFullScreen(true);

        stage.show();

    }

    public static VBox loadSideBar(User user) {

        VBox userInfoSection = new VBox(10);

        userInfoSection.setPadding(new Insets(40,0,0,0));

        userInfoSection.setId("userInfo");

        userInfoSection.setAlignment(Pos.CENTER);

        Label userName = new Label(user.getUserName());

        userName.setAlignment(Pos.CENTER);

        Label userEmail = new Label(user.getUserEmail());

        userEmail.setAlignment(Pos.CENTER);

        userInfoSection.getChildren().addAll(userName, userEmail);

        VBox operationCenter = new VBox(10);

        operationCenter.setAlignment(Pos.CENTER);

        Button addRecordButton = new Button("Add Record");

        addRecordButton.setOnAction(e -> {

            PopUps.addRecordPopUp();

        });

        addRecordButton.setPrefWidth(150);

        addRecordButton.setAlignment(Pos.CENTER);

        Button updateRecordsButton = new Button("Update Records");

        updateRecordsButton.setOnAction(e -> {

            PopUps.getRecordPop();

        });

        updateRecordsButton.setPrefWidth(150);

        updateRecordsButton.setAlignment(Pos.CENTER);

        Button deleteRecordsButton = new Button("Delete Record");

        deleteRecordsButton.setOnAction(e -> {

            System.out.println("Button Clicked");

            PopUps.deleteRecordPopUp();

        });

        deleteRecordsButton.setPrefWidth(150);

        deleteRecordsButton.setAlignment(Pos.CENTER);

        Button changePasswordButton = new Button("Exit");

        changePasswordButton.setOnAction(e -> {

            stage.close();

        });

        changePasswordButton.setPrefWidth(150);

        changePasswordButton.setAlignment(Pos.CENTER);

        VBox summarySection = new VBox();

        String [] summaryInfo= TradeManager.getSummary();

        summarySection.setId("summary");

        summarySection.setAlignment(Pos.CENTER);

        Label part1 = new Label(summaryInfo[0]);

        Label part2 = new Label(summaryInfo[1]);

        Label part3 = new Label(summaryInfo[2]);

        Label part4 = new Label(summaryInfo[3]);

        Label part5 = new Label(summaryInfo[4]);

        Label part6 = new Label(summaryInfo[5]);

        Label part7= new Label(summaryInfo[6]);

        summarySection.getChildren().addAll(part1, part2, part3, part4, part5, part6, part7);

        operationCenter.getChildren().addAll(addRecordButton, updateRecordsButton, deleteRecordsButton, changePasswordButton);

        VBox sidebar = new VBox(80);

        sidebar.setId("Sidebar");

        sidebar.getChildren().addAll(userInfoSection, operationCenter,summarySection);

        return sidebar;

    }

    public static VBox loadPortfolioSection() {

        HBox headerSection = loadHeaderSection();

        VBox portfolioSection = new VBox(30);

        ScrollPane records = loadRecords();

        portfolioSection.getChildren().addAll(headerSection, records);

        portfolioSection.setPadding(new Insets(10));

        portfolioSection.setId("PortfolioSection");

        return portfolioSection;

    }

    public static HBox loadHeaderSection() {

        HBox headerBox = new HBox(100);

        headerBox.setPadding(new Insets(20));

        headerBox.setAlignment(Pos.CENTER);

        Label mainHeader = new Label("Stack Sats, Track Stats!");

        mainHeader.setId("mainHeader");

        mainHeader.setAlignment(Pos.CENTER);

        Label miniHeader = new Label("From moonshots to dips, let the coin flipper track your crypto trips.");

        miniHeader.setId("miniHeader");

        VBox headingSection = new VBox();

        headingSection.setAlignment(Pos.CENTER);

        headingSection.getChildren().addAll(mainHeader, miniHeader);

        headerBox.getChildren().addAll(headingSection);

        headerBox.setId("HeaderBox");

        return headerBox;

    }

    public static ScrollPane loadRecords() {

        FlowPane flowPane = new FlowPane();

        flowPane.setId("RecordsSection");

        flowPane.setPadding(new Insets(20));

        flowPane.setAlignment(Pos.CENTER);

        flowPane.getChildren().clear();

        PortfolioOperationHandler.tradeRecords.forEach(record -> {

            VBox vbox = new VBox(10);

            vbox.setId("Record");

            Label tradeId = new Label("Trade ID: " + record.getRecordId());

            Label coinName = new Label("Coin Name: " + record.getCoinName());

            Label tradeDate = new Label("Date: " + record.getTradeDate());

            Label tradeQuantity = new Label("Quantity: " + record.getQuantity());

            Label tradePrice = new Label("Buy Price: $" + record.getBuyPrice());

            Label tradeTotal = new Label("Sell Price: $" + record.getSellPrice());

            Label tradeProfit = new Label("Profit/Loss: $" + record.getProfitOrLoss());

            Label tradeType = new Label("Type: " + record.getTradeType());

            Image image = new Image(Portfolio.class.getResource("/images/trade2.jpeg").toExternalForm());

            ImageView imageView = new ImageView(image);

            imageView.setFitWidth(200);

            imageView.setPreserveRatio(true);

            vbox.getChildren().addAll(

                    imageView,

                    tradeId, coinName, tradeDate, tradeQuantity,

                    tradePrice, tradeTotal, tradeProfit, tradeType

            );

            vbox.setStyle("-fx-padding: 10; -fx-border-color: gray; -fx-border-width: 1; -fx-border-radius: 5;");

            flowPane.getChildren().add(vbox);

        });

        flowPane.setHgap(20);

        flowPane.setVgap(20);

        ScrollPane scrollPane = new ScrollPane(flowPane);

        scrollPane.setFitToWidth(true);

        scrollPane.setPannable(true);

        scrollPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.AS\_NEEDED);

        scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS\_NEEDED);

        scrollPane.setStyle("-fx-padding: 10;");

        scrollPane.setId("RecordsSection");

        return scrollPane;

    }

}

CHAT HISTORICALDATA.JAVA:

package com.example.crypto\_price\_tracker.models;

import okhttp3.OkHttpClient;

import okhttp3.Request;

import okhttp3.Response;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.Plot;

import org.jfree.chart.plot.XYPlot;

import org.jfree.data.time.Second;

import org.jfree.data.time.TimeSeries;

import org.jfree.data.time.TimeSeriesCollection;

import org.json.JSONArray;

import org.json.JSONObject;

import org.jfree.chart.ChartUtils;

import org.jfree.chart.title.LegendTitle;

import java.awt.Color;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class ChartHistoricalData {

    private static final String API\_KEY = "CG-qG85zkoPRrAkMh2tPJ6ryNvg";

    private static final String BASE\_URL = "https://api.coingecko.com/api/v3/";

    public static void fetchAndGenerateChart(String coinID,String currency) throws Exception {

        OkHttpClient client = new OkHttpClient();

        Request request = new Request.Builder()

                .url(BASE\_URL + "coins/" + coinID + "/market\_chart?vs\_currency=" +currency +"&days=7")

                .get()

                .addHeader("accept", "application/json")

                .addHeader("x-cg-api-key", API\_KEY)

                .build();

        Response response = client.newCall(request).execute();

        if (response.isSuccessful()) {

            assert response.body() != null;

            JSONObject jsonResponse = new JSONObject(response.body().string());

            JSONArray priceData = jsonResponse.getJSONArray("prices");

            List<Double> prices = new ArrayList<>();

            List<Long> timestamps = new ArrayList<>();

            for (int i = 0; i < priceData.length(); i++) {

                JSONArray dataPoint = priceData.getJSONArray(i);

                long timestamp = dataPoint.getLong(0);

                double price = dataPoint.getDouble(1);

                timestamps.add(timestamp);

                prices.add(price);

            }

            saveChartAsImage(timestamps, prices,currency);

        } else {

            throw new Exception("Failed to fetch historical data");

        }

    }

    public static void saveChartAsImage(List<Long> timestamps, List<Double> prices,String currency) throws IOException {

        TimeSeries priceSeries = new TimeSeries("Price ("+currency.toUpperCase()+")");

        for (int i = 0; i < timestamps.size(); i++) {

            long timestamp = timestamps.get(i);

            double price = prices.get(i);

            priceSeries.addOrUpdate(new Second(new java.util.Date(timestamp)), price);

        }

        TimeSeriesCollection dataset = new TimeSeriesCollection(priceSeries);

        JFreeChart chart = ChartFactory.createTimeSeriesChart(

                "Historical Coin Price",

                "Time",

                 currency.toUpperCase(),

                dataset,

                false,

                true,

                false

        );

        chart.setBackgroundPaint(new Color(169, 169, 169));

        Plot plot = chart.getPlot();

        if (plot instanceof XYPlot) {

            XYPlot xyPlot = (XYPlot) plot;

            xyPlot.setBackgroundPaint(new Color(50, 50, 50));

            xyPlot.setDomainGridlinePaint(new Color(200, 200, 200));

            xyPlot.setRangeGridlinePaint(new Color(200, 200, 200));

            chart.getTitle().setPaint(new Color(255, 255, 255));

            LegendTitle legend = chart.getLegend();

            if (legend == null) {

                chart.addLegend(new LegendTitle(chart.getPlot()));

            }

            if (chart.getLegend() != null) {

                chart.getLegend().setItemPaint(new Color(255, 0, 0));

            }

            xyPlot.getDomainAxis().setLabelPaint(new Color(0, 255, 255));

            xyPlot.getDomainAxis().setTickLabelPaint(new Color(0, 255, 255));

            xyPlot.getRangeAxis().setLabelPaint(new Color(0, 255, 255));

            xyPlot.getRangeAxis().setTickLabelPaint(new Color(0, 255, 255));

        }

        File outputFile = new File("src/main/resources/charts/historical\_coin\_price\_chart.png");

        if (outputFile.exists()) {

            boolean deleted = outputFile.delete();

            if (deleted) {

                System.out.println("Previous chart image deleted.");

            } else {

                System.out.println("Failed to delete the previous chart image.");

            }

        }

        ChartUtils.saveChartAsPNG(outputFile, chart, 800, 500);

        System.out.println("Chart saved as image: " + outputFile.getAbsolutePath());

    }

}

COIN.JAVA

package com.example.crypto\_price\_tracker.models;

import javafx.beans.property.*\**;

public class Coin {

    private StringProperty coinName = new SimpleStringProperty();

    private StringProperty coinID = new SimpleStringProperty();

    private DoubleProperty currentPrice = new SimpleDoubleProperty();

    private DoubleProperty percentageChange24h = new SimpleDoubleProperty();

    private DoubleProperty marketCap = new SimpleDoubleProperty();

    private IntegerProperty marketRank = new SimpleIntegerProperty();

    private DoubleProperty perDayHigh = new SimpleDoubleProperty();

    private DoubleProperty perDayLow = new SimpleDoubleProperty();

    private StringProperty coinImageUrl = new SimpleStringProperty();

    public Coin(String coinID, String coinName, double currentPrice, double percentageChange24h, double marketCap,

                String coinImageUrl, int marketRank, double perDayHigh, double perDayLow) {

*this*.coinID.set(coinID);

*this*.coinName.set(coinName);

*this*.currentPrice.set(currentPrice);

*this*.percentageChange24h.set(percentageChange24h);

*this*.marketCap.set(marketCap);

*this*.coinImageUrl.set(coinImageUrl);

*this*.marketRank.set(marketRank);

*this*.perDayHigh.set(perDayHigh);

*this*.perDayLow.set(perDayLow);

    }

    public StringProperty coinNameProperty() {

        return coinName;

    }

    public String getCoinName() {

        return coinName.get();

    }

    public void setCoinName(String coinName) {

*this*.coinName.set(coinName);

    }

    public StringProperty coinIDProperty() {

        return coinID;

    }

    public String getCoinID() {

        return coinID.get();

    }

    public void setCoinID(String coinID) {

*this*.coinID.set(coinID);

    }

    public DoubleProperty currentPriceProperty() {

        return currentPrice;

    }

    public double getCurrentPrice() {

        return currentPrice.get();

    }

    public void setCurrentPrice(double currentPrice) {

*this*.currentPrice.set(currentPrice);

    }

    public DoubleProperty percentageChange24hProperty() {

        return percentageChange24h;

    }

    public double getPercentageChange24h() {

        return percentageChange24h.get();

    }

    public void setPercentageChange24h(double percentageChange24h) {

*this*.percentageChange24h.set(percentageChange24h);

    }

    public DoubleProperty marketCapProperty() {

        return marketCap;

    }

    public double getMarketCap() {

        return marketCap.get();

    }

    public void setMarketCap(double marketCap) {

*this*.marketCap.set(marketCap);

    }

    public IntegerProperty marketRankProperty() {

        return marketRank;

    }

    public int getMarketRank() {

        return marketRank.get();

    }

    public void setMarketRank(int marketRank) {

*this*.marketRank.set(marketRank);

    }

    public DoubleProperty perDayHighProperty() {

        return perDayHigh;

    }

    public double getPerDayHigh() {

        return perDayHigh.get();

    }

    public void setPerDayHigh(double perDayHigh) {

*this*.perDayHigh.set(perDayHigh);

    }

    public DoubleProperty perDayLowProperty() {

        return perDayLow;

    }

    public double getPerDayLow() {

        return perDayLow.get();

    }

    public void setPerDayLow(double perDayLow) {

*this*.perDayLow.set(perDayLow);

    }

    public StringProperty coinImageUrlProperty() {

        return coinImageUrl;

    }

    public String getCoinImageUrl() {

        return coinImageUrl.get();

    }

    public void setCoinImageUrl(String coinImageUrl) {

*this*.coinImageUrl.set(coinImageUrl);

    }

    @Override

    public String toString() {

        return "Coin{" +

                "coinID='" + coinID.get() + '\'' +

                ", coinName='" + coinName.get() + '\'' +

                ", currentPrice=" + currentPrice.get() +

                ", percentageChange24h=" + percentageChange24h.get() +

                ", marketCap=" + marketCap.get() +

                ", marketRank=" + marketRank.get() +

                ", perDayHigh=" + perDayHigh.get() +

                ", perDayLow=" + perDayLow.get() +

                ", coinImageUrl='" + coinImageUrl.get() + '\'' +

                '}';

    }

    @Override

    public boolean equals(Object obj) {

        if (*this* == obj) return true;

        if (!(obj instanceof Coin)) return false;

        Coin coin = (Coin) obj;

        return coinName.get().equals(coin.coinName.get());

    }

    @Override

    public int hashCode() {

        return coinName.get().hashCode();

    }}

COIN DETAILFETCHER.JAVA

package com.example.crypto\_price\_tracker.models;

import okhttp3.OkHttpClient;

import okhttp3.Request;

import okhttp3.Response;

import org.json.JSONObject;

import java.util.ArrayList;

public class CoinDetailsFetcher {

    public static ArrayList<String> fetchCoinDetails(String coinID,String currency) throws Exception {

        OkHttpClient client = new OkHttpClient();

        Request request = new Request.Builder()

                .url("https://api.coingecko.com/api/v3/coins/" + coinID)

                .get()

                .addHeader("accept", "application/json")

                .addHeader("x-cg-api-key", "CG-qG85zkoPRrAkMh2tPJ6ryNvg")

                .build();

        Response response = client.newCall(request).execute();

        if (response.isSuccessful()) {

            assert response.body() != null;

            JSONObject jsonResponse = new JSONObject(response.body().string());

            String largeImageUrl = jsonResponse.getJSONObject("image").getString("large");

            String marketRank = String.valueOf(jsonResponse.getInt("market\_cap\_rank"));

            String description = jsonResponse.getJSONObject("description").optString("en","Description of Coin is not Available");

            String websiteLink = jsonResponse.getJSONObject("links").getJSONArray("homepage").optString(0,"Not Available");

            ArrayList<String> coinsDetails=new ArrayList<String>();

            coinsDetails.add(largeImageUrl);

            coinsDetails.add(marketRank);

            coinsDetails.add(description);

            coinsDetails.add(websiteLink);

            return coinsDetails;

        } else {

            throw new Exception("Unexpected response: " + response);

        }

    }

}

COINFETCHER.JAVA

package com.example.crypto\_price\_tracker.models;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import okhttp3.OkHttpClient;

import okhttp3.Request;

import okhttp3.Response;

import org.json.JSONArray;

import org.json.JSONObject;

import java.io.IOException;

public class CoinsFetcher {

    public static ObservableList<Coin> fetchCoinsData(String currency) {

        OkHttpClient client = new OkHttpClient();

        System.out.println("Request Made Coin List API");

        Request request = new Request.Builder()

                .url("https://api.coingecko.com/api/v3/coins/markets?vs\_currency=" + currency)

                .get()

                .addHeader("accept", "application/json")

                .addHeader("x-cg-api-key", "CG-qG85zkoPRrAkMh2tPJ6ryNvg")

                .build();

        Response response = null;

        try {

            response = client.newCall(request).execute();

            System.out.println("Response: " + response);

        } catch (IOException e) {

            throw new RuntimeException(e);

        }

        if (response.isSuccessful()) {

            assert response.body() != null;

            JSONArray jsonArray = null;

            try {

                jsonArray = new JSONArray(response.body().string());

            } catch (IOException e) {

                throw new RuntimeException(e);

            }

            ObservableList<Coin> coins = FXCollections.observableArrayList();

            for (int i = 0; i < jsonArray.length(); i++) {

                JSONObject jsonCoin = jsonArray.getJSONObject(i);

                String coinID = jsonCoin.optString("id", "Unknown");

                String coinName = jsonCoin.optString("name", "Unknown");

                double currentPrice = jsonCoin.optDouble("current\_price", 0.0);

                double percentageChange24h = jsonCoin.optDouble("price\_change\_percentage\_24h", 0.0);

                double marketCap = jsonCoin.optDouble("market\_cap", 0.0);

                String coinImageUrl = jsonCoin.optString("image", "");

                int marketRank = jsonCoin.optInt("market\_cap\_rank", 0);

                double perDayHigh = jsonCoin.optDouble("high\_24h", 0.0);

                double perDayLow = jsonCoin.optDouble("low\_24h", 0.0);

                Coin coin = new Coin(coinID, coinName, currentPrice, percentageChange24h, marketCap,

                        coinImageUrl, marketRank, perDayHigh, perDayLow);

                coins.add(coin);

            }

            return coins;

        } else {

            try {

                throw new IOException("Unexpected code " + response);

            } catch (IOException e) {

                throw new RuntimeException(e);

            }

        }

    }

}

CRYPTONEWSFETCHER.JAVA

package com.example.crypto\_price\_tracker.models;

import java.io.IOException;

import java.util.ArrayList;

import okhttp3.OkHttpClient;

import okhttp3.Request;

import okhttp3.Response;

import org.json.JSONArray;

import org.json.JSONObject;

public class CryptoNewsFetcher {

    private ArrayList<News> newsList;

    public CryptoNewsFetcher() {

*this*.newsList = new ArrayList<>();

        fetchNewsFromAPI();

    }

    public void addNews(String title, String description, String sourceName, String imageUrl, String link) {

        News news = new News(title, description, sourceName, imageUrl, link);

        newsList.add(news);

    }

    public ArrayList<News> getNewsList() {

        return newsList;

    }

    public void fetchNewsFromAPI() {

        String apiUrl ="https://newsdata.io/api/1/latest?apikey=pub\_51089f7f1f7851c261f9e490f8852c7f37612&q=finance&language=en&country=us";

        OkHttpClient client = new OkHttpClient();

        Request request = new Request.Builder()

                .url(apiUrl)

                .get()

                .build();

        Response response = null;

        try {

            response = client.newCall(request).execute();

            if (response.isSuccessful()) {

                String responseBody = response.body().string();

                JSONObject jsonResponse = new JSONObject(responseBody);

                JSONArray articles = jsonResponse.getJSONArray("results");

                for (int i = 0; i < articles.length(); i++) {

                    JSONObject article = articles.getJSONObject(i);

                    String title = article.optString("title", "No title available");

                    String description = article.optString("description", "No description available");

                    String sourceName = article.optString("source", "Unknown source");

                    String imageUrl = article.optString("image\_url", "");

                    String link = article.optString("link", "No link available");

                    addNews(title, description, sourceName, imageUrl, link);

                }

            } else {

                System.err.println("Error: " + response.message());

            }

        } catch (IOException e) {

            System.err.println("Network error: " + e.getMessage());

            e.printStackTrace();

        } catch (Exception e) {

            System.err.println("Unexpected error: " + e.getMessage());

            e.printStackTrace();

        } finally {

            if (response != null) {

                response.close();

            }

        }

    }

}

NEWS.JAVA

package com.example.crypto\_price\_tracker.models;

public class News {

    private String title;

    private String description;

    private String sourceName;

    private String imageUrl;

    private String link;

    public News(String title, String description, String sourceName, String imageUrl, String link) {

*this*.title = title;

*this*.description = description;

*this*.sourceName = sourceName;

*this*.imageUrl = imageUrl;

*this*.link = link;

    }

    public String getTitle() {

        return title;

    }

    public void setTitle(String title) {

*this*.title = title;

    }

    public String getDescription() {

        return description;

    }

    public void setDescription(String description) {

*this*.description = description;

    }

    public String getSourceName() {

        return sourceName;

    }

    public void setSourceName(String sourceName) {

*this*.sourceName = sourceName;

    }

    public String getImageUrl() {

        return imageUrl;

    }

    public void setImageUrl(String imageUrl) {

*this*.imageUrl = imageUrl;

    }

    public String getLink() {

        return link;

    }

    public void setLink(String link) {

*this*.link = link;

    }

    @Override

    public String toString() {

        return "News{" +

                "title='" + title + '\'' +

                ", description='" + description + '\'' +

                ", sourceName='" + sourceName + '\'' +

                ", imageUrl='" + imageUrl + '\'' +

                ", link='" + link + '\'' +

                '}';

    }

}

TRADE MANAGER.JAVA :

package com.example.crypto\_price\_tracker.models;

import com.example.crypto\_price\_tracker.layouts.Portfolio;

import com.example.crypto\_price\_tracker.util.CUSTOMALERT;

import com.example.crypto\_price\_tracker.util.PortfolioOperationHandler;

import com.example.crypto\_price\_tracker.util.UserDAO;

import javafx.scene.control.Alert;

import java.io.*\**;

import java.nio.file.*\**;

import java.util.*\**;

import java.time.LocalDate;

public class TradeManager {

    public static void addRecord(TradeRecord record) {

        PortfolioOperationHandler.addRecord(UserDAO.getCurrentUser().getUserEmail(),record.getCoinName(),String.valueOf(record.getTradeDate()), record.getTradeType(), record.getQuantity(), record.getBuyPrice(), record.getSellPrice(), record.getTotalValue(), record.getProfitOrLoss());

        PortfolioOperationHandler.loadRecordsFromDatabase(UserDAO.getCurrentUser().getUserEmail());

        System.out.println("REcord is saved");

    }

    public static boolean deleteRecord(int recordID){

        boolean found=false;

       for (TradeRecord record: PortfolioOperationHandler.tradeRecords){

           if(record.getRecordId()==recordID){

               PortfolioOperationHandler.deleteRecord(UserDAO.getCurrentUser().getUserEmail(), recordID);

               PortfolioOperationHandler.loadRecordsFromDatabase(UserDAO.getCurrentUser().getUserEmail());

               found=true;

               System.out.println("Record is deleted");

               return found;

           }

       }

       return found;

    }

    public static void updateRecord(LocalDate tradeDate,String coinName,String tradeType,double quantity,double buyPRice,double sellPrice,double totalValue,double profitLoss,int recordId){

        PortfolioOperationHandler.updateRecord(UserDAO.getCurrentUser().getUserEmail(),recordId, coinName,String.valueOf(tradeDate),tradeType,quantity,buyPRice,sellPrice,totalValue,profitLoss);

        PortfolioOperationHandler.loadRecordsFromDatabase(UserDAO.getCurrentUser().getUserEmail());

    }

    public static String[] getSummary() {

        String[] summary = new String[7];

*// Initialize required variables*

        int totalTrades = PortfolioOperationHandler.tradeRecords.size();

        double totalValue = 0;

        double totalProfit = 0;

        double totalLoss = 0;

        int winningTrades = 0; *// Trades with positive profit*

*// Iterate through trade records to calculate totals*

        for (TradeRecord record : PortfolioOperationHandler.tradeRecords) {

            totalValue += record.getTotalValue();

            double profitOrLoss = record.getProfitOrLoss();

            if (profitOrLoss > 0) {

                totalProfit += profitOrLoss;

                winningTrades++;

            } else {

                totalLoss += profitOrLoss;

            }

        }

        double  averageProfit = (totalTrades > 0) ? totalProfit / totalTrades : 0;

        double averageLoss = (totalTrades > 0) ? totalLoss / totalTrades : 0;

        double winProbability = (totalTrades > 0) ? (winningTrades \* 100) / totalTrades : 0;

*// Populate the summary array*

        summary[0] = "Total Trades: " + totalTrades;

        summary[1] = "Total Value: $" + totalValue;

        summary[2] = "Total Profit: $" + totalProfit;

        summary[3] = "Total Loss: $" + totalLoss;

        summary[4] = "Average Profit: $" + averageProfit;

        summary[5] = "Average Loss: $" + averageLoss;

        summary[6] = "Win Probability: " + winProbability + "%";

        return summary;

    }

}

TRADE RECORD .JAVA :

package com.example.crypto\_price\_tracker.models;

import java.time.LocalDate;

import java.math.BigDecimal;

import java.math.RoundingMode;

public class TradeRecord {

    private int recordId;

    private LocalDate tradeDate;

    private String coinName;

    private String tradeType;

    private double quantity;

    private double buyPrice;

    private double sellPrice;

    private double totalValue;

    private double profitOrLoss;

    public TradeRecord(LocalDate tradeDate, String coinName, String tradeType, double quantity, double buyPrice) {

*this*.tradeDate = tradeDate;

*this*.coinName = coinName;

*this*.tradeType = tradeType;

*this*.quantity = roundOff(quantity);

*this*.buyPrice = roundOff(buyPrice);

*this*.totalValue = roundOff(calculateTotalValue());

*this*.sellPrice = 0.0;

*this*.profitOrLoss = 0.0;

    }

*// Constructor with both buy price and sell price*

    public TradeRecord(LocalDate tradeDate, String coinName, String tradeType, double quantity, double buyPrice, double sellPrice) {

*this*.tradeDate = tradeDate;

*this*.coinName = coinName;

*this*.tradeType = tradeType;

*this*.quantity = roundOff(quantity);

*this*.buyPrice = roundOff(buyPrice);

*this*.sellPrice = roundOff(sellPrice);

*this*.totalValue = roundOff(calculateTotalValue());

        if (*this*.tradeType.equalsIgnoreCase("Sell")) {

*this*.profitOrLoss = roundOff(calculateProfitOrLoss());

        } else {

*this*.profitOrLoss = 0.0;

        }

    }

    public TradeRecord(LocalDate tradeDate, String coinName, String tradeType, double quantity, double buyPrice, double sellPrice, double totalValue, double profitOrLoss) {

*this*.tradeDate = tradeDate;

*this*.coinName = coinName;

*this*.tradeType = tradeType;

*this*.quantity = roundOff(quantity);

*this*.buyPrice = roundOff(buyPrice);

*this*.sellPrice = roundOff(sellPrice);

*this*.totalValue = roundOff(totalValue);

*this*.profitOrLoss = roundOff(profitOrLoss);

    }

    public TradeRecord(LocalDate tradeDate, String coinName, String tradeType, double quantity, double buyPrice, double sellPrice, double totalValue, double profitOrLoss, int recordId) {

*this*.tradeDate = tradeDate;

*this*.coinName = coinName;

*this*.tradeType = tradeType;

*this*.quantity = roundOff(quantity);

*this*.buyPrice = roundOff(buyPrice);

*this*.sellPrice = roundOff(sellPrice);

*this*.totalValue = roundOff(totalValue);

*this*.profitOrLoss = roundOff(profitOrLoss);

*this*.recordId = recordId;

    }

    private double calculateTotalValue() {

        if (*this*.tradeType.equalsIgnoreCase("Sell")) {

            return roundOff(*this*.quantity \* *this*.sellPrice); *// calculate total value for sell*

        }

        return roundOff(*this*.quantity \* *this*.buyPrice); *// calculate total value for buy*

    }

    private double calculateProfitOrLoss() {

        if (*this*.tradeType.equalsIgnoreCase("Sell")) {

            return roundOff((*this*.sellPrice - *this*.buyPrice) \* *this*.quantity);

        }

        return 0.0; *// No profit or loss for a buy transaction*

    }

    private double roundOff(double value) {

        BigDecimal bd = new BigDecimal(value).setScale(3, RoundingMode.HALF\_UP);

        return bd.doubleValue();

    }

    public LocalDate getTradeDate() {

        return tradeDate;

    }

    public int getRecordId() {

        return recordId;

    }

    public void setRecordId(int recordId) {

*this*.recordId = recordId;

    }

    public void setTradeDate(LocalDate tradeDate) {

*this*.tradeDate = tradeDate;

    }

    public String getCoinName() {

        return coinName;

    }

    public void setCoinName(String coinName) {

*this*.coinName = coinName;

    }

    public String getTradeType() {

        return tradeType;

    }

    public void setTradeType(String tradeType) {

*this*.tradeType = tradeType;

    }

    public double getQuantity() {

        return quantity;

    }

    public void setQuantity(double quantity) {

*this*.quantity = roundOff(quantity);

    }

    public double getBuyPrice() {

        return buyPrice;

    }

    public void setBuyPrice(double buyPrice) {

*this*.buyPrice = roundOff(buyPrice);

    }

    public double getSellPrice() {

        return sellPrice;

    }

    public void setSellPrice(double sellPrice) {

*this*.sellPrice = roundOff(sellPrice);

    }

    public double getTotalValue() {

        return totalValue;

    }

    public void setTotalValue(double totalValue) {

*this*.totalValue = roundOff(totalValue);

    }

    public double getProfitOrLoss() {

        return profitOrLoss;

    }

    public void setProfitOrLoss(double profitOrLoss) {

*this*.profitOrLoss = roundOff(profitOrLoss);

    }

    @Override

    public boolean equals(Object o) {

        if (*this* == o) return true;

        if (!(o instanceof TradeRecord)) return false;

        TradeRecord that = (TradeRecord) o;

        return *this*.recordId == that.recordId;

    }

    @Override

    public String toString() {

        return recordId + "|" + tradeDate + "|" + coinName + "|" + tradeType + "|" + quantity + "|" + buyPrice + "|" + sellPrice + "|"

                + totalValue + "|" + profitOrLoss;

    }

}

USER.JAVA :

package com.example.crypto\_price\_tracker.models;

public class User {

    private static int userCounter=1;

    private String userID;

    private String userName;

    private String userEmail;

    private String userPassword;

    private String passwordKey;

    public User(String userName, String userEmail, String userPassword, String passwordKey) {

*this*.userName = userName;

*this*.userEmail = userEmail;

*this*.userPassword = userPassword;

*this*.passwordKey = passwordKey;

*this*.userID = "USER"+userCounter++;

    }

    public String getUserID() {

        return userID;

    }

    public void setUserID(String userID) {

*this*.userID = userID;

    }

    public String getUserName() {

        return userName;

    }

    public void setUserName(String userName) {

*this*.userName = userName;

    }

    public String getUserEmail() {

        return userEmail;

    }

    public void setUserEmail(String userEmail) {

*this*.userEmail = userEmail;

    }

    public String getUserPassword() {

        return userPassword;

    }

    public void setUserPassword(String userPassword) {

*this*.userPassword = userPassword;

    }

    public String getPasswordKey() {

        return passwordKey;

    }

    public void setPasswordKey(String passwordKey) {

*this*.passwordKey = passwordKey;

    }

}