Application Development(APD545NDD.12037.2251)
Final Project

Full Name: Radmehr Behzadfar
Id: 148786221
Seneca email: rbehzadfar@myseneca.ca

Code

Main.java

```
package application;



import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.scene.image.Image;

import javafx.scene.layout.BorderPane;

import javafx.stage.Stage;

import model.DBInitializer;



import java.io.IOException;
```

```
import java.io.InputStream;



public class Main extends Application {



  private static Stage primaryStage;



  @Override

  public void start(Stage primaryStage) {

    DBInitializer.initialize();



    try {

      Main.primaryStage = primaryStage;



      FXMLLoader loader = new FXMLLoader();

      loader.setLocation(getClass().getResource("/fxml/MainView.fxml"));

      BorderPane root = loader.load();
```

```
        Scene scene = new Scene(root);



        InputStream iconStream =
getClass().getResourceAsStream("/images/budget_781760.png");

        if (iconStream != null) {

           Image appIcon = new Image(iconStream);

           primaryStage.getIcons().add(appIcon);

        }



        primaryStage.setScene(scene);

        primaryStage.setTitle("Personal Finance Management");

        primaryStage.show();

     } catch (IOException e) {

        e.printStackTrace();

     }

  }



  public static Stage getPrimaryStage() {
```

```
        return primaryStage;
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

MainController.java

```
package application;


import javafx.fxml.FXML;
import javafx.scene.control.TabPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;


public class MainController {
```

```
    @FXML
    private TabPane mainTabPane;


    @FXML
    private ImageView logoImageView;


    @FXML
    public void initialize() {
        // Load the logo image
        try {
            Image logo = new
Image(getClass().getResourceAsStream("/images/budget_781760.png"));
            logoImageView.setImage(logo);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

BudgetController.java

```
package controller;



import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.*;

import javafx.beans.property.SimpleObjectProperty;

import model.Budget;

import model.Database;



import java.sql.SQLException;

import java.util.List;

import java.util.Optional;



public class BudgetController {
```

```
@FXML

private TableView<Budget> budgetTable;

@FXML

private TableColumn<Budget, String> categoryColumn;

@FXML

private TableColumn<Budget, Double> limitColumn;



@FXML

private TextField categoryField;

@FXML

private TextField limitField;



private ObservableList<Budget> budgetList = FXCollections.observableArrayList();



@FXML

public void initialize() {

   categoryColumn.setCellValueFactory(cellData ->
```

```
        new SimpleObjectProperty<>(cellData.getValue().getCategory())
    );
    limitColumn.setCellValueFactory(cellData ->
        new SimpleObjectProperty<>(cellData.getValue().getLimit())
    );

    budgetTable.setItems(budgetList);

    loadBudgets();
}

private void loadBudgets() {
    budgetList.clear();
    try {
        List<Budget> all = Database.getAllBudgets();
        budgetList.addAll(all);
    } catch (SQLException e) {
```

```
        e.printStackTrace();
    }
}

@FXML
private void handleAddBudget() {
    String category = categoryField.getText();
    if (category == null || category.trim().isEmpty()) {
        showAlert("Invalid Category", "Category cannot be empty.");
        return;
    }
    double limit;
    try {
        limit = Double.parseDouble(limitField.getText());
        if (limit <= 0) {
            showAlert("Invalid Limit", "Budget limit must be positive.");
            return;
```

```
    }
} catch (NumberFormatException e) {
    showAlert("Invalid Limit", "Please enter a valid number for budget limit.");
    return;
}

Budget b = new Budget();
b.setCategory(category);
b.setLimit(limit);

try {
    Database.insertBudget(b);
    loadBudgets();
    clearFields();
    showAlert("Success", "Budget added successfully.");
} catch (SQLException e) {
    e.printStackTrace();
```

```
        showAlert("Database Error", "Could not add budget.");
    }
}

@FXML
private void handleEditBudget() {
    Budget selected = budgetTable.getSelectionModel().getSelectedItem();
    if (selected == null) {
        showAlert("No Selection", "Please select a budget to edit.");
        return;
    }

    categoryField.setText(selected.getCategory());
    limitField.setText(String.valueOf(selected.getLimit()));

    // Confirm editing
    Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION);
```

```
confirmAlert.setTitle("Edit Budget");
confirmAlert.setHeaderText("Editing budget with ID: " + selected.getId());
confirmAlert.setContentText("Click OK to apply changes.");

Optional<ButtonType> result = confirmAlert.showAndWait();
if (result.isPresent() && result.get() == ButtonType.OK) {
  // Validate
  String category = categoryField.getText();
  if (category == null || category.trim().isEmpty()) {
    showAlert("Invalid Category", "Category cannot be empty.");
    return;
  }
  double limit;
  try {
    limit = Double.parseDouble(limitField.getText());
    if (limit <= 0) {
      showAlert("Invalid Limit", "Budget limit must be positive.");
```

```
        return;
    }
} catch (NumberFormatException e) {
    showAlert("Invalid Limit", "Please enter a valid number for budget limit.");
    return;
}

selected.setCategory(category);
selected.setLimit(limit);

try {
    Database.updateBudget(selected);
    loadBudgets();
    clearFields();
    showAlert("Success", "Budget updated successfully.");
} catch (SQLException e) {
    e.printStackTrace();
```

```
            showAlert("Database Error", "Could not update budget.");
        }
    }
}

@FXML
private void handleDeleteBudget() {
    Budget selected = budgetTable.getSelectionModel().getSelectedItem();
    if (selected == null) {
        showAlert("No Selection", "Please select a budget to delete.");
        return;
    }

    Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION);
    confirmAlert.setTitle("Delete Budget");
    confirmAlert.setHeaderText("Deleting budget with ID: " + selected.getId());
    confirmAlert.setContentText("Are you sure you want to delete this budget?");
```

```
    Optional<ButtonType> result = confirmAlert.showAndWait();
    if (result.isPresent() && result.get() == ButtonType.OK) {
        try {
            Database.deleteBudget(selected.getId());
            loadBudgets();
            showAlert("Success", "Budget deleted successfully.");
        } catch (SQLException e) {
            e.printStackTrace();
            showAlert("Database Error", "Could not delete budget.");
        }
    }
}

private void clearFields() {
    categoryField.clear();
    limitField.clear();
```

```
    }

    private void showAlert(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

SummaryController.java

```
package controller;


import javafx.concurrent.Task;
import javafx.fxml.FXML;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.PieChart;
```

```
import javafx.scene.chart.XYChart;

import javafx.scene.control.Alert;

import model.Database;

import model.Transaction;



import java.sql.SQLException;

import java.time.YearMonth;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;



public class SummaryController {



    @FXML

    private PieChart categoryPieChart;

    @FXML

    private BarChart<String, Number> monthlyBarChart;
```

```
@FXML
public void initialize() {
   // Called automatically when FXML loads
}


@FXML
private void handleLoadSummary() {
   // Demonstrate concurrency by loading transactions in a background thread
   Task<Void> loadTask = new Task<>() {
      @Override
      protected Void call() throws Exception {
         updateMessage("Loading data...");
         buildCharts();
         return null;
      }
   };
```

```
    loadTask.setOnSucceeded(e -> {
        // Show a message or do something else when done
        showAlert("Summary Loaded", "Summary charts updated.");
    });

    loadTask.setOnFailed(e -> {
        showAlert("Error", "Failed to load summary data.");
        loadTask.getException().printStackTrace();
    });

    Thread thread = new Thread(loadTask);
    thread.setDaemon(true);
    thread.start();
}

private void buildCharts() {
```

```
    try {
        List<Transaction> transactions = Database.getAllTransactions();
        // Calculate expenses by category
        Map<String, Double> expenseByCategory = new TreeMap<>();
        for (Transaction t : transactions) {
            if ("expense".equalsIgnoreCase(t.getType())) {
                expenseByCategory.put(t.getCategory(),
                        expenseByCategory.getOrDefault(t.getCategory(), 0.0) + t.getAmount());
            }
        }

        // JavaFX UI must be updated on the Application Thread
        javafx.application.Platform.runLater(() -> {
            categoryPieChart.getData().clear();
            for (Map.Entry<String, Double> entry : expenseByCategory.entrySet()) {
                categoryPieChart.getData().add(new PieChart.Data(entry.getKey(),
entry.getValue()));
            }
```

```
});



// Build monthly net totals (income - expense)
Map<YearMonth, Double> monthlyTotals = new TreeMap<>();
for (Transaction t : transactions) {
   YearMonth ym = YearMonth.from(t.getDate());
   double current = monthlyTotals.getOrDefault(ym, 0.0);
   if ("income".equalsIgnoreCase(t.getType())) {
      current += t.getAmount();
   } else {
      current -= t.getAmount();
   }
   monthlyTotals.put(ym, current);
}



XYChart.Series<String, Number> series = new XYChart.Series<>();
series.setName("Monthly Net");
```

```
        for (Map.Entry<YearMonth, Double> entry : monthlyTotals.entrySet()) {
            series.getData().add(new XYChart.Data<>(entry.getKey().toString(),
entry.getValue()));
        }

        javafx.application.Platform.runLater(() -> {
            monthlyBarChart.getData().clear();
            monthlyBarChart.getData().add(series);
        });

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

private void showAlert(String title, String content) {
    javafx.application.Platform.runLater(() -> {
```

```
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    });
  }
}
```

TransactionController.java

```
package controller;


import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.beans.property.SimpleObjectProperty;
import model.Database;
```

```
import model.Transaction;

import util.DateUtil;



import java.sql.SQLException;

import java.time.LocalDate;

import java.util.List;

import java.util.Optional;



public class TransactionsController {



    @FXML

    private TableView<Transaction> transactionsTable;

    @FXML

    private TableColumn<Transaction, LocalDate> dateColumn;

    @FXML

    private TableColumn<Transaction, String> categoryColumn;

    @FXML
```

```
    private TableColumn<Transaction, Double> amountColumn;

    @FXML

    private TableColumn<Transaction, String> typeColumn;



    @FXML

    private TextField dateField;

    @FXML

    private TextField amountField;

    @FXML

    private TextField categoryField;

    @FXML

    private TextField descriptionField;

    @FXML

    private ChoiceBox<String> typeChoiceBox; // "income" or "expense"



    private ObservableList<Transaction> transactionList =
FXCollections.observableArrayList();
```

```
@FXML
public void initialize() {
    // Configure table columns
    dateColumn.setCellValueFactory(cellData ->
        new SimpleObjectProperty<>(cellData.getValue().getDate())
    );
    categoryColumn.setCellValueFactory(cellData ->
        new SimpleObjectProperty<>(cellData.getValue().getCategory())
    );
    amountColumn.setCellValueFactory(cellData ->
        new SimpleObjectProperty<>(cellData.getValue().getAmount())
    );
    typeColumn.setCellValueFactory(cellData ->
        new SimpleObjectProperty<>(cellData.getValue().getType())
    );

    transactionsTable.setItems(transactionList);
```

```
    // Initialize type choice box
    typeChoiceBox.getItems().addAll("income", "expense");
    typeChoiceBox.setValue("expense");

    loadTransactions();
}

private void loadTransactions() {
    transactionList.clear();
    try {
        List<Transaction> all = Database.getAllTransactions();
        transactionList.addAll(all);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
@FXML
private void handleAddTransaction() {
    // Validate date
    String dateText = dateField.getText();
    if (!DateUtil.validDate(dateText)) {
        showAlert("Invalid Date", "Please enter a valid date (yyyy-MM-dd).");
        return;
    }
    // Validate amount
    double amount;
    try {
        amount = Double.parseDouble(amountField.getText());
        if (amount <= 0) {
            showAlert("Invalid Amount", "Amount must be positive.");
            return;
        }
```

```
} catch (NumberFormatException e) {
    showAlert("Invalid Amount", "Please enter a valid number for amount.");
    return;
}
// Validate category
String category = categoryField.getText();
if (category == null || category.trim().isEmpty()) {
    showAlert("Invalid Category", "Category cannot be empty.");
    return;
}
// Get description and type
String description = descriptionField.getText();
String type = typeChoiceBox.getValue();


Transaction t = new Transaction();
t.setDate(LocalDate.parse(dateText));
t.setAmount(amount);
```

```
    t.setCategory(category);
    t.setDescription(description);
    t.setType(type);

    try {
        Database.insertTransaction(t);
        loadTransactions();
        clearFields();
        showAlert("Success", "Transaction added successfully.");
        // Check budget status for the transaction's category
        checkBudgetNotification(category);
    } catch (SQLException e) {
        e.printStackTrace();
        showAlert("Database Error", "Could not add transaction.");
    }
}
```

```
@FXML
private void handleEditTransaction() {
    Transaction selected = transactionsTable.getSelectionModel().getSelectedItem();
    if (selected == null) {
        showAlert("No Selection", "Please select a transaction to edit.");
        return;
    }

    // Pre-fill fields
    dateField.setText(selected.getDate().toString());
    amountField.setText(String.valueOf(selected.getAmount()));
    categoryField.setText(selected.getCategory());
    descriptionField.setText(selected.getDescription());
    typeChoiceBox.setValue(selected.getType());

    // Confirm editing
    Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION);
```

```
confirmAlert.setTitle("Edit Transaction");
confirmAlert.setHeaderText("Editing transaction with ID: " + selected.getId());
confirmAlert.setContentText("Click OK to apply changes.");

Optional<ButtonType> result = confirmAlert.showAndWait();
if (result.isPresent() && result.get() == ButtonType.OK) {
  // Validate & update
  if (!DateUtil.validDate(dateField.getText())) {
    showAlert("Invalid Date", "Please enter a valid date (yyyy-MM-dd).");
    return;
  }
  double amount;
  try {
    amount = Double.parseDouble(amountField.getText());
    if (amount <= 0) {
      showAlert("Invalid Amount", "Amount must be positive.");
      return;
```

```
    }
} catch (NumberFormatException e) {
    showAlert("Invalid Amount", "Please enter a valid number for amount.");
    return;
}
String category = categoryField.getText();
if (category == null || category.trim().isEmpty()) {
    showAlert("Invalid Category", "Category cannot be empty.");
    return;
}
String description = descriptionField.getText();
String type = typeChoiceBox.getValue();



selected.setDate(LocalDate.parse(dateField.getText()));
selected.setAmount(amount);
selected.setCategory(category);
selected.setDescription(description);
```

```
        selected.setType(type);



        try {
            Database.updateTransaction(selected);
            loadTransactions();
            clearFields();
            showAlert("Success", "Transaction updated successfully.");
            // Check budget status after editing
            checkBudgetNotification(category);
        } catch (SQLException e) {
            e.printStackTrace();
            showAlert("Database Error", "Could not update transaction.");
        }
    }
}



@FXML
```

```
private void handleDeleteTransaction() {
    Transaction selected = transactionsTable.getSelectionModel().getSelectedItem();
    if (selected == null) {
        showAlert("No Selection", "Please select a transaction to delete.");
        return;
    }

    Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION);
    confirmAlert.setTitle("Delete Transaction");
    confirmAlert.setHeaderText("Deleting transaction with ID: " + selected.getId());
    confirmAlert.setContentText("Are you sure you want to delete this transaction?");

    Optional<ButtonType> result = confirmAlert.showAndWait();
    if (result.isPresent() && result.get() == ButtonType.OK) {
        try {
            Database.deleteTransaction(selected.getId());
            loadTransactions();
```

```
            showAlert("Success", "Transaction deleted successfully.");
        } catch (SQLException e) {
            e.printStackTrace();
            showAlert("Database Error", "Could not delete transaction.");
        }
    }
}

private void clearFields() {
    dateField.clear();
    amountField.clear();
    categoryField.clear();
    descriptionField.clear();
    typeChoiceBox.setValue("expense");
}

private void showAlert(String title, String content) {
```

```
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(content);
    alert.showAndWait();
  }

  // New method to check budget notifications for a given category
  private void checkBudgetNotification(String category) {
    try {
      double totalExpenses = Database.getTotalExpenses(category);
      double limit = Database.getBudgetLimit(category);
      if (limit > 0) {
        double ratio = totalExpenses / limit;
        if (ratio >= 1.0) {
          showAlert("Budget Alert", "Your spending for " + category + " has exceeded the
budget limit!");
        } else if (ratio >= 0.8) {
```

```
                int percent = (int)(ratio * 100);

                showAlert("Budget Warning", "Warning: Your spending for " + category + " has
reached " + percent + "% of your budget.");

            }

        }

    } catch (SQLException e) {

        e.printStackTrace();

    }

  }

}
```

Budget.java

```
package model;



public class Budget {

  private int id;

  private String category;

  private double limit;
```

```
    public Budget() {}

    public Budget(int id, String category, double limit) {
        this.id = id;
        this.category = category;
        this.limit = limit;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }

    public double getLimit() { return limit; }
    public void setLimit(double limit) { this.limit = limit; }
}
```

Database.java

```
package model;



import java.sql.*;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;



public class Database {



    private static final String DB_URL = "jdbc:sqlite:personal_finance.db";



    // Keep a single connection instance if desired (Singleton pattern)

    private static Connection connection = null;



    public static Connection getConnection() throws SQLException {

        if (connection == null || connection.isClosed()) {
```

```
        connection = DriverManager.getConnection(DB_URL);
    }
    return connection;
  }

  // Insert Transaction
  public static void insertTransaction(Transaction transaction) throws SQLException {
    String sql = "INSERT INTO transactions(date, amount, category, description, type)
VALUES(?,?,?,?,?)";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, transaction.getDate().toString());
        pstmt.setDouble(2, transaction.getAmount());
        pstmt.setString(3, transaction.getCategory());
        pstmt.setString(4, transaction.getDescription());
        pstmt.setString(5, transaction.getType());
        pstmt.executeUpdate();
    }
```

```
    }



    // Update Transaction

    public static void updateTransaction(Transaction transaction) throws SQLException {

        String sql = "UPDATE transactions SET date=?, amount=?, category=?, description=?,
type=? WHERE id=?";

        try (Connection conn = getConnection();

             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, transaction.getDate().toString());

            pstmt.setDouble(2, transaction.getAmount());

            pstmt.setString(3, transaction.getCategory());

            pstmt.setString(4, transaction.getDescription());

            pstmt.setString(5, transaction.getType());

            pstmt.setInt(6, transaction.getId());

            pstmt.executeUpdate();

        }

    }
```

```
// Delete Transaction
public static void deleteTransaction(int transactionId) throws SQLException {
   String sql = "DELETE FROM transactions WHERE id=?";
   try (Connection conn = getConnection();
       PreparedStatement pstmt = conn.prepareStatement(sql)) {
      pstmt.setInt(1, transactionId);
      pstmt.executeUpdate();
   }
}

// Get All Transactions
public static List<Transaction> getAllTransactions() throws SQLException {
   List<Transaction> list = new ArrayList<>();
   String sql = "SELECT * FROM transactions ORDER BY date DESC";
   try (Connection conn = getConnection();
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery(sql)) {
```

```
        while (rs.next()) {
            Transaction t = new Transaction();
            t.setId(rs.getInt("id"));
            t.setDate(LocalDate.parse(rs.getString("date")));
            t.setAmount(rs.getDouble("amount"));
            t.setCategory(rs.getString("category"));
            t.setDescription(rs.getString("description"));
            t.setType(rs.getString("type"));
            list.add(t);
        }
    }
    return list;
}

// Insert Budget
public static void insertBudget(Budget budget) throws SQLException {
    String sql = "INSERT INTO budgets(category, limit_amount) VALUES(?,?)";
```

```
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, budget.getCategory());
        pstmt.setDouble(2, budget.getLimit());
        pstmt.executeUpdate();
    }
}

// Update Budget
public static void updateBudget(Budget budget) throws SQLException {
    String sql = "UPDATE budgets SET category=?, limit_amount=? WHERE id=?";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, budget.getCategory());
        pstmt.setDouble(2, budget.getLimit());
        pstmt.setInt(3, budget.getId());
        pstmt.executeUpdate();
```

```
    }
}

// Delete Budget
public static void deleteBudget(int budgetId) throws SQLException {
    String sql = "DELETE FROM budgets WHERE id=?";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, budgetId);
        pstmt.executeUpdate();
    }
}

// Get All Budgets
public static List<Budget> getAllBudgets() throws SQLException {
    List<Budget> list = new ArrayList<>();
    String sql = "SELECT * FROM budgets";
```

```
    try (Connection conn = getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            Budget b = new Budget();
            b.setId(rs.getInt("id"));
            b.setCategory(rs.getString("category"));
            b.setLimit(rs.getDouble("limit_amount"));
            list.add(b);
        }
    }
    return list;
}

// Check if a category's spending exceeds its budget
public static boolean isOverBudget(String category) throws SQLException {
    // Get total expenses for the category
```

```
    String expenseSql = "SELECT SUM(amount) as total_expenses FROM transactions
WHERE category=? AND type='expense'";

    double totalExpenses = 0;

    try (Connection conn = getConnection();

        PreparedStatement pstmt = conn.prepareStatement(expenseSql)) {

       pstmt.setString(1, category);

       ResultSet rs = pstmt.executeQuery();

       if (rs.next()) {

          totalExpenses = rs.getDouble("total_expenses");

       }

    }



    // Get the budget limit

    String budgetSql = "SELECT limit_amount FROM budgets WHERE category=?";

    double limit = 0;

    try (Connection conn = getConnection();

        PreparedStatement pstmt = conn.prepareStatement(budgetSql)) {

       pstmt.setString(1, category);
```

```
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            limit = rs.getDouble("limit_amount");
        }
    }

    return limit > 0 && totalExpenses > limit;
}

// New helper: Get total expenses for a given category
public static double getTotalExpenses(String category) throws SQLException {
    String expenseSql = "SELECT SUM(amount) as total_expenses FROM transactions
WHERE category=? AND type='expense'";
    double totalExpenses = 0;
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(expenseSql)) {
        pstmt.setString(1, category);
        ResultSet rs = pstmt.executeQuery();
```

```
        if (rs.next()) {
            totalExpenses = rs.getDouble("total_expenses");
        }
    }
    return totalExpenses;
}

// New helper: Get budget limit for a given category
public static double getBudgetLimit(String category) throws SQLException {
    String budgetSql = "SELECT limit_amount FROM budgets WHERE category=?";
    double limit = 0;
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(budgetSql)) {
        pstmt.setString(1, category);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            limit = rs.getDouble("limit_amount");
```

```
        }
      }
      return limit;
   }
}
```

DBInitializer.java

```
package model;


import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;


public class DBInitializer {


   public static void initialize() {
      try (Connection conn = Database.getConnection();
          Statement stmt = conn.createStatement()) {
```

```
// Create Transactions table if it doesn't exist
String createTransactionsTable = "CREATE TABLE IF NOT EXISTS transactions ("
    + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
    + "date TEXT NOT NULL, "
    + "amount REAL NOT NULL, "
    + "category TEXT NOT NULL, "
    + "description TEXT, "
    + "type TEXT NOT NULL"
    + ");";
stmt.execute(createTransactionsTable);


// Create Budgets table if it doesn't exist
String createBudgetsTable = "CREATE TABLE IF NOT EXISTS budgets ("
    + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
    + "category TEXT NOT NULL, "
    + "limit_amount REAL NOT NULL"
```

```
                + ");";

            stmt.execute(createBudgetsTable);



        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

}
```

Transaction.java

```
package model;



import java.time.LocalDate;



public class Transaction {

    private int id;

    private LocalDate date;

    private double amount;
```

```
    private String category;

    private String description;

    private String type; // "income" or "expense"



    public Transaction() {}



    public Transaction(int id, LocalDate date, double amount, String category, String
description, String type) {

        this.id = id;

        this.date = date;

        this.amount = amount;

        this.category = category;

        this.description = description;

        this.type = type;

    }



    public int getId() { return id; }

    public void setId(int id) { this.id = id; }
```

```
    public LocalDate getDate() { return date; }

    public void setDate(LocalDate date) { this.date = date; }



    public double getAmount() { return amount; }

    public void setAmount(double amount) { this.amount = amount; }



    public String getCategory() { return category; }

    public void setCategory(String category) { this.category = category; }



    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }



    public String getType() { return type; }

    public void setType(String type) { this.type = type; }

}
```

DateUtil.java

```
package util;



import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;



public class DateUtil {



  private static final String DATE_PATTERN = "yyyy-MM-dd";

  private static final DateTimeFormatter DATE_FORMATTER =

      DateTimeFormatter.ofPattern(DATE_PATTERN);



  public static String format(LocalDate date) {

    if (date == null) {

      return null;

    }

    return DATE_FORMATTER.format(date);
```

```
    }

    public static LocalDate parse(String dateString) {
        try {
            return LocalDate.parse(dateString, DATE_FORMATTER);
        } catch (DateTimeParseException e) {
            return null;
        }
    }

    public static boolean validDate(String dateString) {
        return parse(dateString) != null;
    }
}
```

BudgetView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
```

```
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml"

    fx:controller="controller.BudgetController"

    spacing="10"

    xmlns="http://javafx.com/javafx/17"

    style="-fx-padding: 10;">



  <TableView fx:id="budgetTable" prefHeight="200">

    <columns>

      <TableColumn fx:id="categoryColumn" text="Category" prefWidth="150"/>

      <TableColumn fx:id="limitColumn" text="Limit" prefWidth="100"/>

    </columns>

  </TableView>



  <HBox spacing="10">

    <VBox spacing="5">

      <Label text="Category"/>
```

```
        <TextField fx:id="categoryField"/>
      </VBox>
      <VBox spacing="5">
        <Label text="Limit"/>
        <TextField fx:id="limitField"/>
      </VBox>
    </HBox>

    <HBox spacing="10">
      <Button text="Add" onAction="#handleAddBudget"/>
      <Button text="Edit" onAction="#handleEditBudget"/>
      <Button text="Delete" onAction="#handleDeleteBudget"/>
    </HBox>
  </VBox>
```

MainView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Tab?>
```

```
<?import javafx.scene.control.TabPane?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.layout.BorderPane?>

<?import javafx.scene.image.ImageView?>

<BorderPane xmlns:fx="http://javafx.com/fxml"

        fx:controller="application.MainController"

        xmlns="http://javafx.com/javafx/17">



  <!-- Top: Title -->

  <top>

    <Label text="Personal Finance Management"

          style="-fx-font-size: 18; -fx-font-weight: bold; -fx-padding: 10;" />

  </top>



  <!-- Left: Logo -->

  <left>

    <ImageView fx:id="logoImageView" fitWidth="100" preserveRatio="true"
```

```
            style="-fx-padding: 10;" />
    </left>


    <!-- Center: TabPane -->
    <center>
       <TabPane fx:id="mainTabPane">
          <Tab text="Transactions">
             <content>
                <fx:include source="TransactionsView.fxml" />
             </content>
          </Tab>
          <Tab text="Budget">
             <content>
                <fx:include source="BudgetView.fxml" />
             </content>
          </Tab>
          <Tab text="Summary">
```

```
            <content>
              <fx:include source="SummaryView.fxml" />
            </content>
          </Tab>
        </TabPane>
    </center>
</BorderPane>
```

SummaryView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.chart.BarChart?>
<?import javafx.scene.chart.CategoryAxis?>
<?import javafx.scene.chart.NumberAxis?>
<?import javafx.scene.chart.PieChart?>
<?import javafx.scene.layout.VBox?>
<VBox xmlns:fx="http://javafx.com/fxml"
    fx:controller="controller.SummaryController"
```

```
    spacing="10"
    xmlns="http://javafx.com/javafx/17"
    style="-fx-padding: 10;">

  <PieChart fx:id="categoryPieChart" title="Expense by Category" prefHeight="200" />

  <BarChart fx:id="monthlyBarChart" prefHeight="200">
    <xAxis>
      <CategoryAxis label="Month"/>
    </xAxis>
    <yAxis>
      <NumberAxis label="Net Amount"/>
    </yAxis>
  </BarChart>

  <Button text="Load Summary" onAction="#handleLoadSummary" />
</VBox>
```

TransactionsView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>

<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml"

    fx:controller="controller.TransactionsController"

    spacing="10"

    xmlns="http://javafx.com/javafx/17"

    style="-fx-padding: 10;">



  <TableView fx:id="transactionsTable" prefHeight="200">

    <columns>

      <TableColumn fx:id="dateColumn" text="Date" prefWidth="100"/>

      <TableColumn fx:id="categoryColumn" text="Category" prefWidth="100"/>

      <TableColumn fx:id="amountColumn" text="Amount" prefWidth="80"/>

      <TableColumn fx:id="typeColumn" text="Type" prefWidth="80"/>

    </columns>
```

```
</TableView>



<HBox spacing="10">
   <VBox spacing="5">
      <Label text="Date (yyyy-MM-dd)"/>
      <TextField fx:id="dateField"/>
   </VBox>
   <VBox spacing="5">
      <Label text="Amount"/>
      <TextField fx:id="amountField"/>
   </VBox>
   <VBox spacing="5">
      <Label text="Category"/>
      <TextField fx:id="categoryField"/>
   </VBox>
   <VBox spacing="5">
      <Label text="Description"/>
```

```
        <TextField fx:id="descriptionField"/>
      </VBox>
      <VBox spacing="5">
        <Label text="Type"/>
        <ChoiceBox fx:id="typeChoiceBox"/>
      </VBox>
    </HBox>

    <HBox spacing="10">
      <Button text="Add" onAction="#handleAddTransaction"/>
      <Button text="Edit" onAction="#handleEditTransaction"/>
      <Button text="Delete" onAction="#handleDeleteTransaction"/>
    </HBox>
  </VBox>
```