

```
1 package application;
2
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.event.ActionEvent;
6 import javafx.event.EventHandler;
7 import javafx.fxml.FXML;
8 import javafx.fxml.FXMLLoader;
9 import javafx.fxml.Initializable;
10 import javafx.geometry.Side;
11 import javafx.scene.Parent;
12 import javafx.scene.Scene;
13 import javafx.scene.chart.PieChart;
14 import javafx.scene.control.Button;
15 import javafx.scene.control.CheckBox;
16 import javafx.scene.control.ChoiceBox;
17 import javafx.scene.control.ContextMenu;
18 import javafx.scene.control.DatePicker;
19 import javafx.scene.control.Label;
20 import javafx.scene.control.ListView;
21 import javafx.scene.control.MenuItem;
22 import javafx.scene.control.PasswordField;
23 import javafx.scene.control.RadioButton;
24 import javafx.scene.control.TableColumn;
25 import javafx.scene.control TableView;
26 import javafx.scene.control.TextArea;
27 import javafx.scene.control.TextField;
28 import javafx.scene.control.ToggleGroup;
29 import javafx.scene.control.cell.PropertyValueFactory;
30 import javafx.scene.layout.BorderPane;
31 import javafx.scene.layout.HBox;
32 import javafx.stage.Stage;
33
34 import java.net.URL;
35 import java.sql.Connection;
36 import java.sql.Date;
37 import java.sql.DriverManager;
38 import java.sql.PreparedStatement;
39 import java.sql.ResultSet;
40 import java.sql.SQLException;
41 import java.sql.Statement;
42 import java.time.LocalDate;
43 import java.util.ArrayList;
44 import java.util.HashMap;
45 import java.util.Map;
46 import java.util.ResourceBundle;
47
48 public class MainController implements Initializable {
49     @FXML
```

```
50     private TextField userNameText;
51
52     @FXML
53     private PasswordField passwordText;
54
55     @FXML
56     private Button login;
57
58     @FXML
59     private Button signUp;
60
61     @FXML
62     private TextField userNameTextS;
63     @FXML
64     private TextField passwordTextS;
65
66     @FXML
67     private TextField checkPasswordText;
68
69     @FXML
70     private Button addTransaction;
71
72     @FXML
73     private TextField expenseName;
74
75     @FXML
76     private ChoiceBox<String> expenseType = new ChoiceBox<>();
77
78     @FXML
79     private TextField priceText;
80
81     @FXML
82     private TextArea description;
83
84     @FXML
85     private Button submit;
86
87     @FXML
88     private DatePicker expenseDate = new DatePicker();
89
90     @FXML
91     public ListView<String> ExpenseList = new ListView<String>();
92
93     @FXML
94     Button refresh;
95
96     private Statement statement;
97
98     @FXML
```

```
99     private HBox expenseBox = new HBox(ExpenseList);
100
101     @FXML
102     private PieChart incomePieChart = new PieChart();
103
104     @FXML
105     private PieChart expensePieChart = new PieChart();
106
107     @FXML
108     private Label incomeLabel = new Label();
109
110     @FXML
111     private Label expenseLabel = new Label();
112
113     @FXML
114     private Label oweLabel = new Label();
115
116     @FXML
117     private Label owedLabel = new Label();
118
119     @FXML
120     private TableView<OweData> oweTable = new TableView<>();
121     @FXML
122     private TableColumn<OweData, String> oweUser = new TableColumn<>();
123     @FXML
124     private TableColumn<OweData, Double> oweAmount = new TableColumn<>();
125
126     @FXML
127     private TableView<OwedData> owedTable = new TableView<>();
128     @FXML
129     private TableColumn<OwedData, String> owedUser = new TableColumn<>();
130     @FXML
131     private TableColumn<OwedData, Double> owedAmount = new TableColumn<>();
132
133     @FXML
134     private TableView<ExpenseData> transactionsTable = new TableView<>();
135     @FXML
136     private TableColumn<ExpenseData, String> titleColumn = new TableColumn<>();
137     @FXML
138     private TableColumn<ExpenseData, String> typeColumn = new TableColumn<>();
139     @FXML
140     private TableColumn<ExpenseData, Double> amountColumn = new TableColumn<>();
141     @FXML
142     private TableColumn<ExpenseData, String> descriptionColumn = new TableColumn<>();
143
144     @FXML
145     private RadioButton incomeChartButton = new RadioButton();
146     @FXML
147     private RadioButton expenseChartButton = new RadioButton();
```

```
148
149 String currentUser = "";
150 Parent root;
151 Stage primaryStage = new Stage();
152
153 ArrayList<OwedData> owedList = new ArrayList<>();
154 ArrayList<OweData> oweList = new ArrayList<>();
155
156 public TextField getUsernameText() {
157     return userNameText;
158 }
159
160 public void setUsernameText(TextField userNameText) {
161     this.userNameText = userNameText;
162 }
163
164 /**
165  * initialize - sets up everything for the program to function properly.
166  */
167 @Override
168 public void initialize(URL arg0, ResourceBundle arg1) {
169     this.expenseType.getItems().addAll("Food", "Income", "Other");
170     oweAmount.setCellValueFactory(new PropertyValueFactory<OweData, Double>("oweAmount"));
171     oweUser.setCellValueFactory(new PropertyValueFactory<OweData, String>("oweUser"));
172     owedAmount.setCellValueFactory(new PropertyValueFactory<OwedData, Double>("owedAmount"));
173     owedUser.setCellValueFactory(new PropertyValueFactory<OwedData, String>("owedUser"));
174
175     titleColumn.setCellValueFactory(new PropertyValueFactory<ExpenseData, String>("titleColumn"));
176     typeColumn.setCellValueFactory(new PropertyValueFactory<ExpenseData, String>("typeColumn"));
177     amountColumn.setCellValueFactory(new PropertyValueFactory<ExpenseData, Double>("amountColumn"));
178     descriptionColumn.setCellValueFactory(new PropertyValueFactory<ExpenseData, String>("descriptionColumn"));
179
180     initializeExpenses();
181     initializeAmountExpected();
182     initializeAmountOwe();
183     initializeTotalExpense();
184     initializeTotalIncome();
185
186     simplifyExpenses();
187     incomePieChart();
188     expensePieChart();
189
190     ToggleGroup group = new ToggleGroup();
191     incomeChartButton.setToggleGroup(group);
192     expenseChartButton.setToggleGroup(group);
193
194     incomePieChart.setOpacity(1);
195     expensePieChart.setOpacity(0);
196 }
```

```
197
198 }
199 /**
200  * SimplifyExpense - it displays the expense data in a smaller table.
201  */
202 private void simplifyExpenses() {
203     HashMap<String, Double> simplified = new HashMap<>();
204     for(OweData data:this.oweList) {
205         if (simplified.containsKey(data.getOweUser())) {
206             simplified.put(data.getOweUser(), simplified.get(data.getOweUser()) - data.getOweAmount());
207         } else {
208             simplified.put(data.getOweUser(), -1*data.getOweAmount());
209         }
210     }
211     for(OwedData owed:this.owedList) {
212         if (simplified.containsKey(owed.getOwedUser())) {
213             simplified.put(owed.getOwedUser(), simplified.get(owed.getOwedUser()) + owed.getOwedAmount());
214         } else {
215             simplified.put(owed.getOwedUser(), owed.getOwedAmount());
216         }
217     }
218     this.owedList = new ArrayList<>();
219     this.oweList = new ArrayList<>();
220
221     Double owed = (double) 0;
222
223     Double owe = (double) 0;
224
225     for (Map.Entry<String,Double> entry : simplified.entrySet()) {
226         if(entry.getValue() > 0) {
227             owed += entry.getValue();
228             this.owedList.add(new OwedData(entry.getKey(), entry.getValue()));
229         } else {
230             owe += (-1 * entry.getValue());
231             this.oweList.add(new OweData(entry.getKey(), -1 * entry.getValue()));
232         }
233     }
234
235     this.owedTable.getItems().addAll(this.owedList);
236     this.oweTable.getItems().addAll(this.oweList);
237
238     owedLabel.setText(owed.toString());
239     oweLabel.setText(owe.toString());
240
241
242 }
243
244 /**
245  * Login method - searches mysql database MYAPP for matches of Username and Password
```

```

246     * it launches MainPage.fxml if the the executed Result Set is true.
247     * @param Event
248     * @throws Exception
249     */
250     public void login(ActionEvent Event) throws Exception {
251         try {
252             Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
253                 "password");
254
255             String sql = "SELECT count(*) FROM MYAPP.USERS where username = ? and password = ? ";
256             PreparedStatement pst = conn.prepareStatement(sql);
257             pst.setString(1, getUserNameText().getText());
258             pst.setString(2, passwordText.getText());
259             ResultSet rs = pst.executeQuery();
260             this.currentUser = getUserNameText().getText();
261             System.out.println(this.currentUser);
262             if (rs.next()) {
263
264                 User u = new User();
265                 u.name = userNameText.getText();
266                 UserHolder userholder = UserHolder.getInstance();
267                 userholder.setUser(u);
268                 userholder.setName(userNameText.getText());
269
270                 BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource("MainPage.fxml"));
271
272                 Scene scene = new Scene(root, 1251, 856);
273                 scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
274
275                 primaryStage.setScene(scene);
276                 primaryStage.show();
277
278             } else {
279                 System.out.println("Login Failed");
280             }
281         } catch (Exception e) {
282             System.out.println("Exception: " + e.toString());
283         }
284     }
285     /**
286     * SignUpButton method - opens SignUp.fxml when signUp button is clicked.
287     * @param Event
288     * @throws Exception
289     */
290     public void signUpButton(ActionEvent Event) throws Exception {
291         BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource("SignUp.fxml"));
292         Scene scene = new Scene(root, 334, 361);
293         scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
294         primaryStage.setScene(scene);

```

```

295     primaryStage.show();
296 }
297
298 /**
299  * SignUp method - it inserts the data entered from the signUp.fxml into the mysql database
300  * @param Event
301  * @throws Exception
302  */
303 public void signUp(ActionEvent Event) throws Exception {
304     try {
305         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
306             "password");
307         PreparedStatement pst = conn.prepareStatement("INSERT INTO MYAPP.USERS values(?,?)");
308         pst.setString(1, userNameTextS.getText());
309         pst.setString(2, passwordTextS.getText());
310         if (passwordTextS.getText().equals(checkPasswordText.getText())) {
311             pst.executeUpdate();
312             System.out.println("Data Registered");
313         } else
314             System.out.println("something went wrong");
315     } catch (Exception e) {
316         System.out.println("Exception: " + e.toString());
317     }
318 }
319 /**
320  * AddTransaction - opens AddTransaction.fxml
321  * @param Event
322  * @throws Exception
323  */
324 public void addTransaction(ActionEvent Event) throws Exception {
325     BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource("AddTransaction.fxml"));
326     Scene scene = new Scene(root, 519, 425);
327     scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
328     primaryStage.setScene(scene);
329     primaryStage.show();
330     // user.setUserName(currentUser);
331 }
332 /**
333  * initializedExpenses - it executes a through the Details table
334  * to get all the data, then takes the total amount for each and
335  * inserts them into the transaction table.
336  * @return
337  */
338 public ListView<String> initializeExpenses() {
339     try {
340         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
341             "password");
342         PreparedStatement ps = conn.prepareStatement("SELECT DETAILS.detailId, DETAILS.title, " + "CASE "
343             + "WHEN TRANSACTIONS.fromUser = ? AND TRANSACTIONS.toUser = 'NONE' THEN 'Expense' "

```

```

344         + "WHEN TRANSACTIONS.toUser = ? AND TRANSACTIONS.fromUser = 'NONE' THEN 'Income' "
345         + "ELSE 'Shared' " + "END AS TRANSACTION_TYPE, "
346         + "MAX(TRANSACTIONS.transactionDate) AS TRANSACTION_DATE, SUM(TRANSACTIONS.amount) AS TRANSACTION_AMOUNT "
347         + "FROM `MYAPP`.TRANSACTIONS, `MYAPP`.DETAILS " + "WHERE DETAILS.detailId = TRANSACTIONS.detailId "
348         + "AND (TRANSACTIONS.fromUser = ? OR TRANSACTIONS.toUser = ? )"
349         + "GROUP BY DETAILS.detailId, DETAILS.title, TRANSACTION_TYPE " + "ORDER BY DETAILS.detailId");
350
351     ps.setString(1, UserHolder.getInstance().getName());
352     ps.setString(2, UserHolder.getInstance().getName());
353     ps.setString(3, UserHolder.getInstance().getName());
354     ps.setString(4, UserHolder.getInstance().getName());
355     statement = conn.createStatement();
356     ResultSet rs = ps.executeQuery();
357     ObservableList<ExpenseData> expenses = FXCollections.observableArrayList();
358     while (rs.next()) {
359         ExpenseData expenseData = new ExpenseData();
360         expenseData.setTitleColumn(rs.getString("title"));
361         expenseData.setTypeColumn(rs.getString("TRANSACTION_TYPE"));
362         expenseData.setDescriptionColumn(rs.getDate("TRANSACTION_DATE").toString());
363         expenseData.setAmountColumn(rs.getDouble("TRANSACTION_AMOUNT"));
364         expenses.add(expenseData);
365     }
366     transactionsTable.setItems(expenses);
367 } catch (Exception e) {
368
369     e.printStackTrace();
370 }
371 return ExpenseList;
372 }
373 /**
374  * initializeTotalExpense - takes data from TRANSACTIONS table stored in the mySQL database.
375  * then takes the amount under any expense category and adds them together.
376  */
377 public void initializeTotalExpense() {
378     try {
379         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
380             "password");
381         PreparedStatement ps = conn
382             .prepareStatement("SELECT SUM(TRANSACTIONS.amount) AS TOTAL_EXPENSES " + "FROM MYAPP.TRANSACTIONS "
383                 + "WHERE TRANSACTIONS.fromUser = ? " + "AND TRANSACTIONS.touser = 'NONE'");
384
385
386         ps.setString(1, UserHolder.getInstance().getName());
387         statement = conn.createStatement();
388         ResultSet rs = ps.executeQuery();
389
390         while (rs.next()) {
391             System.out.println(rs.getString("TOTAL_EXPENSES"));
392             this.expenseLabel.setText(rs.getString("TOTAL_EXPENSES"));

```



```
393     }
394 } catch (Exception e) {
395     e.printStackTrace();
396 }
397 }
398 }
399 /**
400  * initializeTotalIncome - takes data from TRANSACTION table stored in the mySQL database.
401  * then takes the amount under any income category and adds them together.
402  */
403 public void initializeTotalIncome() {
404     try {
405         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
406             "password");
407         PreparedStatement ps = conn
408             .prepareStatement("SELECT SUM(TRANSACTIONS.amount) AS TOTAL_INCOME " + "FROM MYAPP.TRANSACTIONS "
409                 + "WHERE TRANSACTIONS.touser = ? " + "AND TRANSACTIONS.fromUser = 'NONE'");
410
411         ps.setString(1, UserHolder.getInstance().getName());
412         statement = conn.createStatement();
413         ResultSet rs = ps.executeQuery();
414
415         while (rs.next()) {
416             System.out.println(rs.getString("TOTAL_INCOME"));
417             this.incomeLabel.setText(rs.getString("TOTAL_INCOME"));
418         }
419     }
420 } catch (Exception e) {
421     e.printStackTrace();
422 }
423 }
424 }
425
426 /**
427  * initializeAmountExpected - finds the amount of money owed by other user
428  */
429 public void initializeAmountExpected() {
430     try {
431         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
432             "password");
433         PreparedStatement ps = conn.prepareStatement("SELECT touser, SUM(TRANSACTIONS.amount) AMOUNT_EXPECTED "
434             + "FROM MYAPP.TRANSACTIONS " + "WHERE fromuser != 'NONE' AND touser != 'NONE' "
435             + "AND fromUser = ? " + "GROUP BY touser");
436
437         ps.setString(1, UserHolder.getInstance().getName());
438         statement = conn.createStatement();
439         ResultSet rs = ps.executeQuery();
440         while (rs.next()) {
```

```
442         OwedData currentRow = new OwedData();
443         currentRow.setOwedUser(rs.getString("touser"));
444         currentRow.setOwedAmount(rs.getDouble("AMOUNT_EXPECTED"));
445         owedList.add(currentRow);
446     }
447
448     } catch (Exception e) {
449         e.printStackTrace();
450     }
451 }
452 /**
453  * initializeAmountOwe - finds the amount you own to other users
454  */
455 public void initializeAmountOwe() {
456     try {
457         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
458             "password");
459         PreparedStatement ps = conn.prepareStatement("SELECT fromuser AS USER, SUM(TRANSACTIONS.amount) AMOUNT_OWED "
460             + "FROM MYAPP.TRANSACTIONS " + "WHERE fromuser != 'NONE' AND touser != 'NONE' " + "AND touser = ? "
461             + "GROUP BY fromuser");
462
463         ps.setString(1, UserHolder.getInstance().getName());
464         statement = conn.createStatement();
465         ResultSet rs = ps.executeQuery();
466         while (rs.next()) {
467             OwedData currentRow = new OwedData();
468             currentRow.setOweUser(rs.getString(1));
469             currentRow.setOweAmount(rs.getDouble("AMOUNT_OWED"));
470             this.owedList.add(currentRow);
471
472             System.out.println(rs.getString(1) + " " + rs.getDouble("AMOUNT_OWED"));
473         }
474     } catch (Exception e) {
475         e.printStackTrace();
476     }
477 }
478
479 }
480
481 /**
482  * refresh - refreshes main page after pressing the refresh button.
483  * @param event
484  */
485 public void refresh(ActionEvent event) {
486     Stage stage = (Stage) ((Button) event.getSource()).getScene().getWindow();
487     stage.close();
488     try {
489         BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource("MainPage.fxml"));
490     }
```

```

491         Scene scene = new Scene(root, 1003, 745);
492         scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
493
494         primaryStage.setScene(scene);
495         primaryStage.show();
496     } catch (Exception e) {
497         e.printStackTrace();
498     }
499 }
500 /**
501  * expensePieChart - Inserts data into the expense pie chart.
502  */
503 public void expensePieChart() {
504     try {
505         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
506             "password");
507         PreparedStatement ps = conn.prepareStatement("SELECT subType, SUM(amount) AS sum_amount FROM MYAPP.DETAILS,
MYAPP.TRANSACTIONS "
508             + "WHERE TRANSACTIONS.detailId = DETAILS.detailId and toUser = 'NONE' and fromUser = ? GROUP by subType");
509         ps.setString(1, UserHolder.getInstance().getName());
510         ResultSet rs = ps.executeQuery();
511         ObservableList<PieChart.Data> pieChartData = FXCollections.observableArrayList();
512         while(rs.next()) {
513
514             pieChartData.add(new PieChart.Data(rs.getString("subType"), rs.getDouble("sum_amount")));
515
516         }
517
518         expensePieChart.getData().addAll(pieChartData);
519
520
521     } catch (SQLException e) {
522         // TODO Auto-generated catch block
523         e.printStackTrace();
524     }
525 }
526
527 /**
528  * incomePieChart - Inserts data into the income pie chart.
529  */
530 public void incomePieChart() {
531     try {
532         Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/MYAPP", "root",
533             "password");
534         PreparedStatement ps = conn.prepareStatement("SELECT subType, SUM(amount) AS sum_amount FROM MYAPP.DETAILS,
MYAPP.TRANSACTIONS "
535             + "WHERE TRANSACTIONS.detailId = DETAILS.detailId and fromUser = 'NONE' and toUser = ? GROUP by subType");
536         ps.setString(1, UserHolder.getInstance().getName());
537         ResultSet rs = ps.executeQuery();

```

```
538         ObservableList<PieChart.Data> pieChartData = FXCollections.observableArrayList();
539         while(rs.next()) {
540
541             pieChartData.add(new PieChart.Data(rs.getString("subType"), rs.getDouble("sum_amount")));
542
543         }
544         incomePieChart.getData().addAll(pieChartData);
545
546
547     } catch (SQLException e) {
548         // TODO Auto-generated catch block
549         e.printStackTrace();
550     }
551 }
552
553 /**
554  * incomeChartButton - Displays Income PieChart
555  * @param event
556  */
557 public void incomeChartButton(ActionEvent event) {
558     incomePieChart.setOpacity(1);
559     expensePieChart.setOpacity(0);
560 }
561 /**
562  * expenseChartButton - Displays Expense PieChart
563  * @param event
564  */
565 public void expenseChartButton(ActionEvent event) {
566     incomePieChart.setOpacity(0);
567     expensePieChart.setOpacity(1);
568 }
569 }
570
```