

MIT WORLD PEACE UNIVERSITY

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

MINI PROJECT WITH JAVA - PRICE GUESSING
GAME
"How Much?"

PROJECT REPORT

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 23, 2022

Contents

1	Introduction	1
2	Methodology Used	1
3	Platform	1
4	Requirements	1
5	Installation and Running	1
6	Database Management	1
6.1	MongoDB	1
6.2	Local CSV Files	2
7	Unique Features	3
7.1	Dark Mode	3
7.2	Data Backup	3
7.3	Web Scrapping	3
7.4	Working Login and Account Creation	3
8	Screenshots of the Project	3
8.1	The Login Page	3
8.2	The Menu Screen	4
8.3	The Topic Selection Screen	4
8.4	The Highscore Screen	5
8.5	The Help and About	5
8.6	The Game Over Screen	6
9	Output Files Produced	7
10	Walk-Through of the Files	7
10.1	Project Structure	7
10.2	TopicsFrame.java	9
10.3	MongoManager.java	9
10.4	MenuFrame.java	9
10.5	Main.java	9
10.6	LoginFrame.java	9
10.7	HighscoreFrame.java	9
10.8	HelpFrame.java	9
10.9	GameOverFrame.java	9
10.10	GameFrame.java	9
10.11	DataBaseManager.java	9
10.12	Colors.java	9
10.13	BackgroundPanel.java	9
10.14	AmazonScrapper.java	9
11	Conclusion and Topics Learnt	9

1 Introduction

2 Methodology Used

3 Platform

Operating System: Arch Linux x86-64

IDEs or Text Editors Used: IntelliJ Idea Ultimate Edition for Java

Compilers : javac, with JDK 18.0.2 for Java

Database : MongoDB 6.0.3.1

4 Requirements

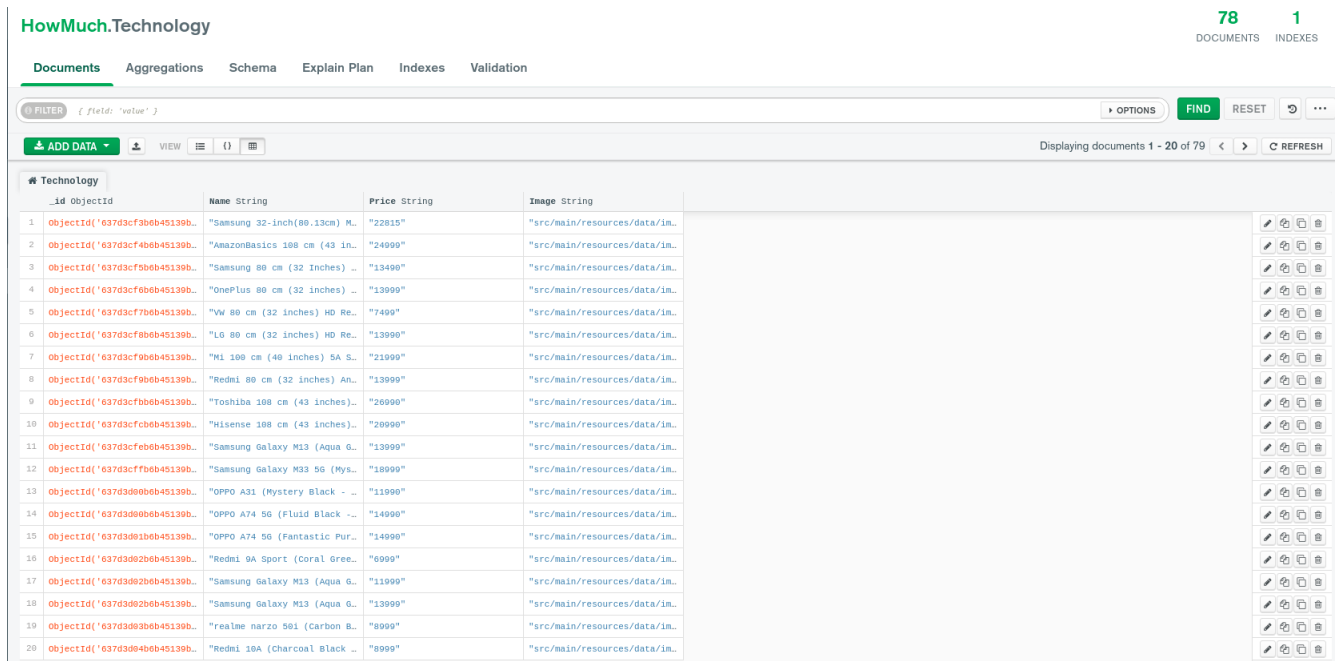
5 Installation and Running

Download the .jar file from the releases when it is released that is. Navigate there from your terminal

```
java -jar ./How_Much.jar
```

6 Database Management

6.1 MongoDB



The screenshot shows the MongoDB Compass interface for a database named 'HowMuch.Technology'. The 'Documents' tab is active, displaying a list of 20 records. Each record represents a smartphone with fields for _id, Name, Price, and Image. The records are numbered 1 through 20 in the left margin. The interface includes a filter bar at the top, a table view toggle, and a 'Displaying documents 1 - 20 of 79' indicator. The table has columns: _id, Objectid, Name String, Price String, and Image String.

	_id	Objectid	Name String	Price String	Image String
1	Objectid('637d3cf3b0b45139b...		"Samsung 32-inch(80.13cm) M...	"22815"	"src/main/resources/data/im...
2	Objectid('637d3cf4b0b45139b...		"AmazonBasics 108 cm (43 in...	"24999"	"src/main/resources/data/im...
3	Objectid('637d3cf5b0b45139b...		"Samsung 80 cm (32 Inches) ...	"13499"	"src/main/resources/data/im...
4	Objectid('637d3cf6b0b45139b...		"OnePlus 80 cm (32 Inches) ...	"13999"	"src/main/resources/data/im...
5	Objectid('637d3cf7b0b45139b...		"VW 80 cm (32 inches) HD Re...	"7499"	"src/main/resources/data/im...
6	Objectid('637d3cf8b0b45139b...		"LG 80 cm (32 inches) HD Re...	"13999"	"src/main/resources/data/im...
7	Objectid('637d3cf9b0b45139b...		"M1 108 cm (40 inches) 5A S...	"21999"	"src/main/resources/data/im...
8	Objectid('637d3cfa0b45139b...		"Redmi 80 cm (32 inches) An...	"13999"	"src/main/resources/data/im...
9	Objectid('637d3cfbb0b45139b...		"Toshiba 108 cm (43 inches)...	"26999"	"src/main/resources/data/im...
10	Objectid('637d3cfc0b45139b...		"Hisense 108 cm (43 inches)...	"26999"	"src/main/resources/data/im...
11	Objectid('637d3cfe0b45139b...		"Samsung Galaxy M13 (Aqua G...	"13999"	"src/main/resources/data/im...
12	Objectid('637d3cff0b45139b...		"Samsung Galaxy M33 5G (Mys...	"18999"	"src/main/resources/data/im...
13	Objectid('637d3d000b45139b...		"OPPO A31 (Mystery Black - ...	"11999"	"src/main/resources/data/im...
14	Objectid('637d3d000b45139b...		"OPPO A74 5G (Fluid Black ~...	"14999"	"src/main/resources/data/im...
15	Objectid('637d3d010b45139b...		"OPPO A74 5G (Fantastic Pur...	"14999"	"src/main/resources/data/im...
16	Objectid('637d3d020b45139b...		"Redmi 9A Sport (Coral Gree...	"6999"	"src/main/resources/data/im...
17	Objectid('637d3d020b45139b...		"Samsung Galaxy M13 (Aqua G...	"11999"	"src/main/resources/data/im...
18	Objectid('637d3d020b45139b...		"Samsung Galaxy M13 (Aqua G...	"13999"	"src/main/resources/data/im...
19	Objectid('637d3d030b45139b...		"realme narzo 501 (Carbon B...	"8999"	"src/main/resources/data/im...
20	Objectid('637d3d040b45139b...		"Redmi 10A (Charcoal Black ...	"8999"	"src/main/resources/data/im...

Figure 1: A Screenshot of the MongoDB Compass Showing Records Stored in the Teachnology Schema

OOPJC Mini Project Report

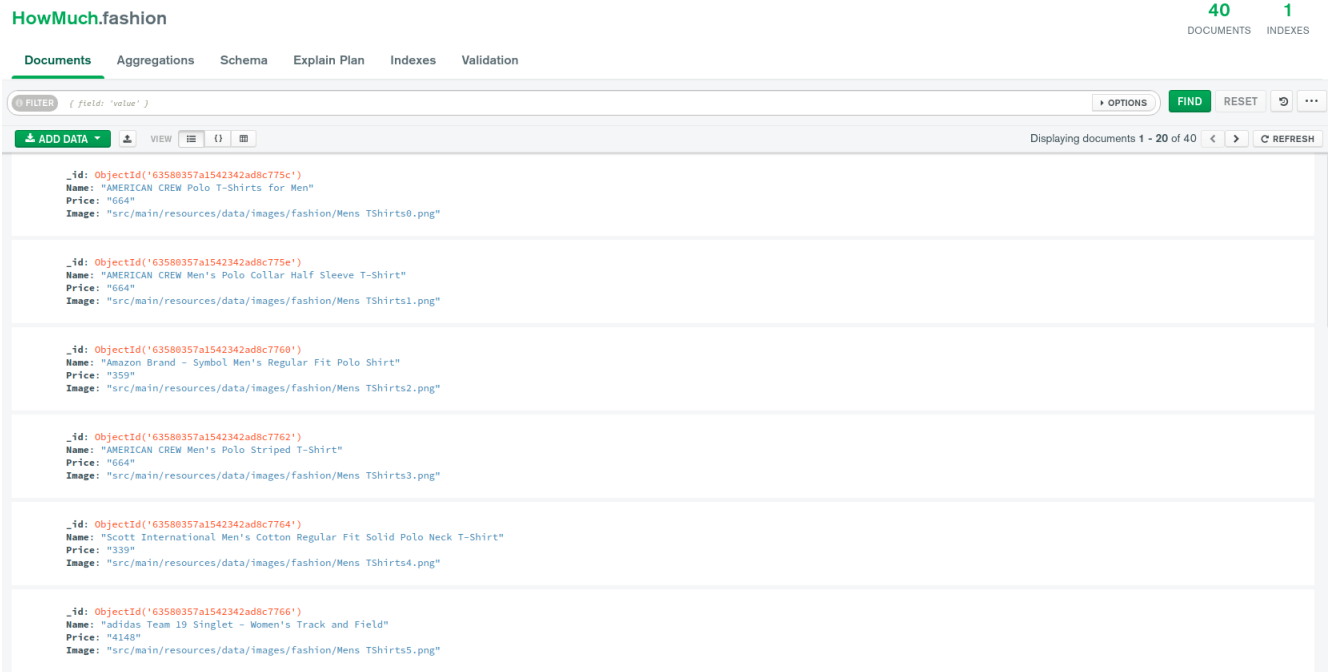


Figure 2: Record Showing the Fashion Schema Documents

6.2 Local CSV Files

C1	C2	C3
1 EPPE Men's Combo of Round Neck Half Sleeve Dryfit...	630	src/main/resources/data/images/fashion/Mens TShirts0.png
2 Allen Solly Men's Regular Fit T-Shirt	433	src/main/resources/data/images/fashion/Mens TShirts1.png
3 BLUELANDER Printed Round Neck Half Sleeve T-Shirt...	299	src/main/resources/data/images/fashion/Mens TShirts2.png
4 Allen Solly Men's Regular Fit T-Shirt	539	src/main/resources/data/images/fashion/Mens TShirts3.png
5 Allen Solly Men Polo	758	src/main/resources/data/images/fashion/Mens TShirts4.png
6 U.S. POLO ASSN. Men T-Shirt	352	src/main/resources/data/images/fashion/Mens TShirts5.png
7 Amazon Brand - Symbol Men T-Shirt	919	src/main/resources/data/images/fashion/Mens TShirts6.png
8 AELOMART Men's T Shirt	497	src/main/resources/data/images/fashion/Mens TShirts7.png
9 Scott International Men's Regular Fit T-Shirt (Pa...	474	src/main/resources/data/images/fashion/Mens TShirts8.png
10 Amazon Brand - Symbol Men's Regular T-Shirt	859	src/main/resources/data/images/fashion/Mens TShirts9.png
11 Park Avenue Full Sleeve Shawl Collar Dark Brown S...	4399	src/main/resources/data/images/fashion/Formal Suits0.png
12 Park Avenue Solid Rayon Blend Dark Blue Regular F...	4599	src/main/resources/data/images/fashion/Formal Suits1.png
13 Park Avenue Dark Grey Suits	4499	src/main/resources/data/images/fashion/Formal Suits2.png
14 Park Avenue Medium Grey Suits	5849	src/main/resources/data/images/fashion/Formal Suits3.png
15 Razab Enterprises (SAAYA) 5 Button Bandhgala/Jodh...	3045	src/main/resources/data/images/fashion/Formal Suits4.png
16 Arrow Men's Polyester Blend Formal Business Suit ...	4979	src/main/resources/data/images/fashion/Formal Suits5.png

Figure 3: Screenshot of the Local CSV File

7 Unique Features

7.1 Dark Mode

7.2 Data Backup

7.3 Web Scrapping

7.4 Working Login and Account Creation

8 Screenshots of the Project

8.1 The Login Page

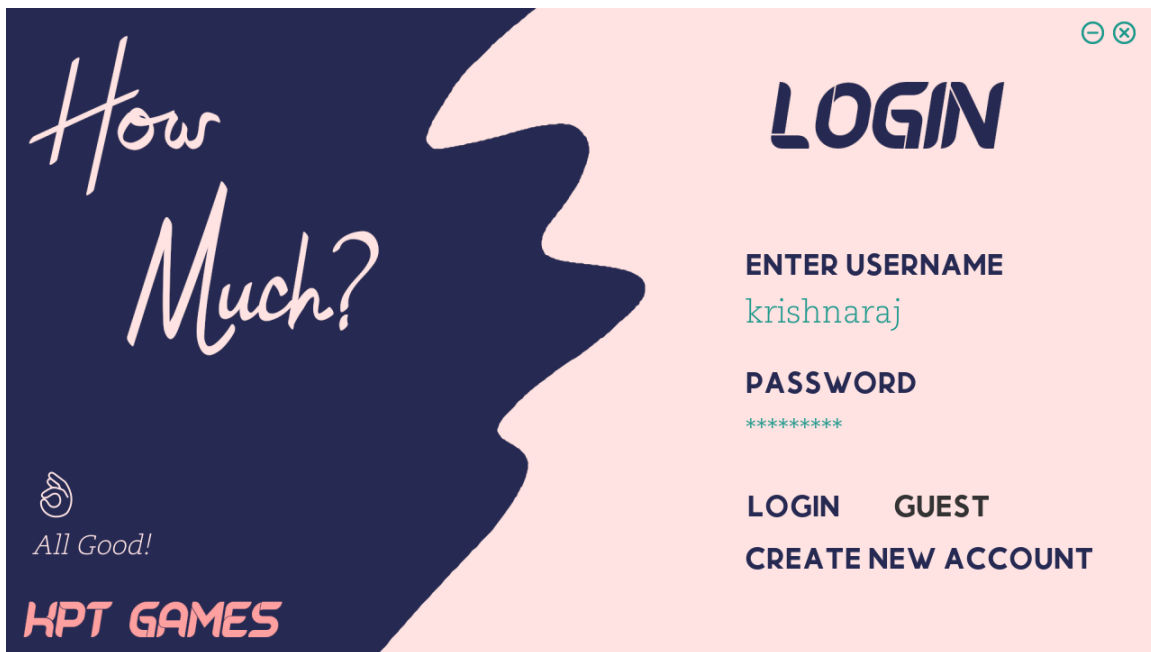


Figure 4: The Login page after a successful login

8.2 The Menu Screen



Figure 5:

8.3 The Topic Selection Screen



Figure 6: The Login page after a successful login

8.4 The Highscore Screen



Figure 7: The Login page after a successful login

8.5 The Help and About



Figure 8: The Login page after a successful login

8.6 The Game Over Screen



Figure 9: The Login page after a successful login

9 Output Files Produced

10 Walk-Through of the Files

10.1 Project Structure

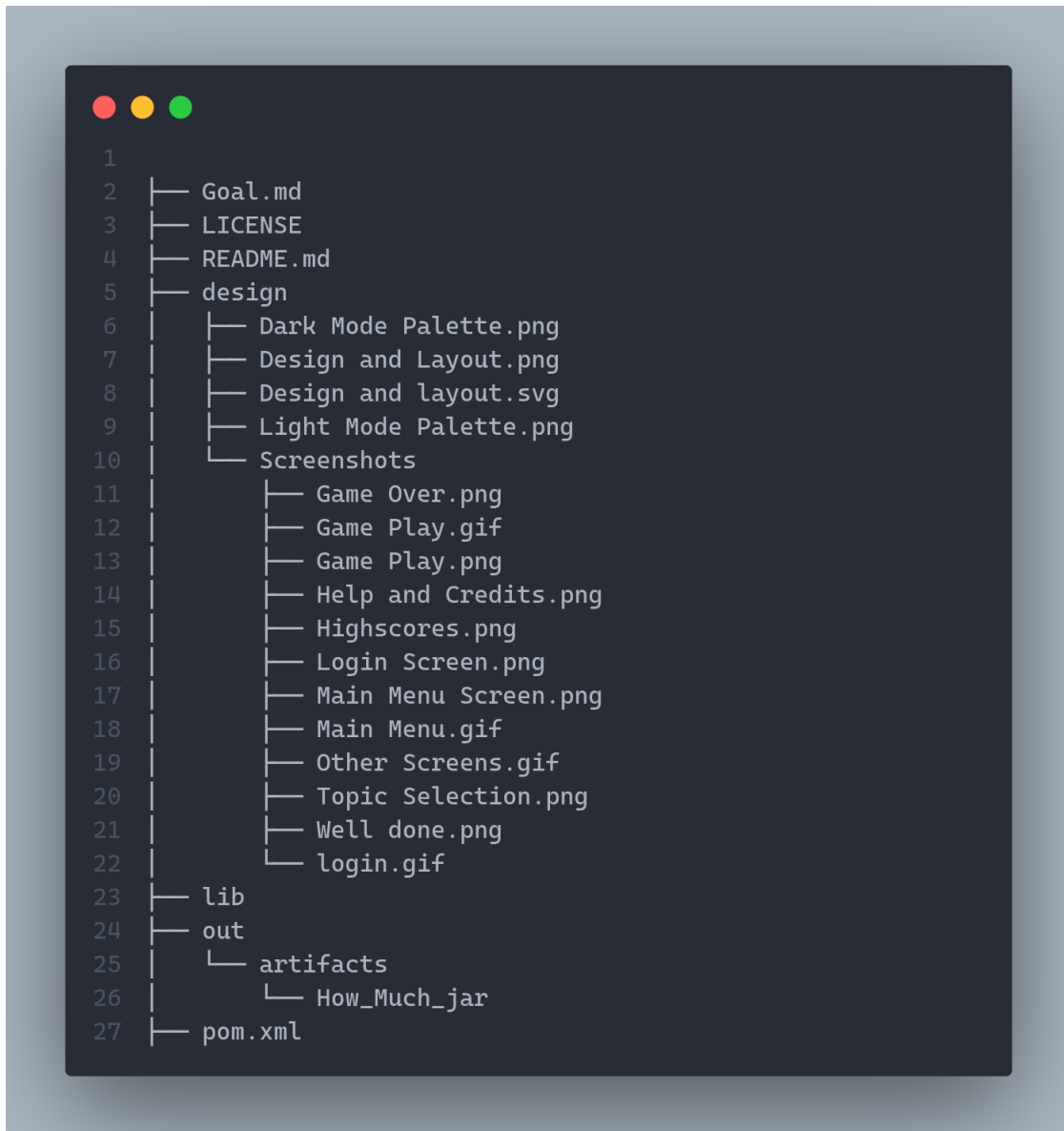


Figure 10:

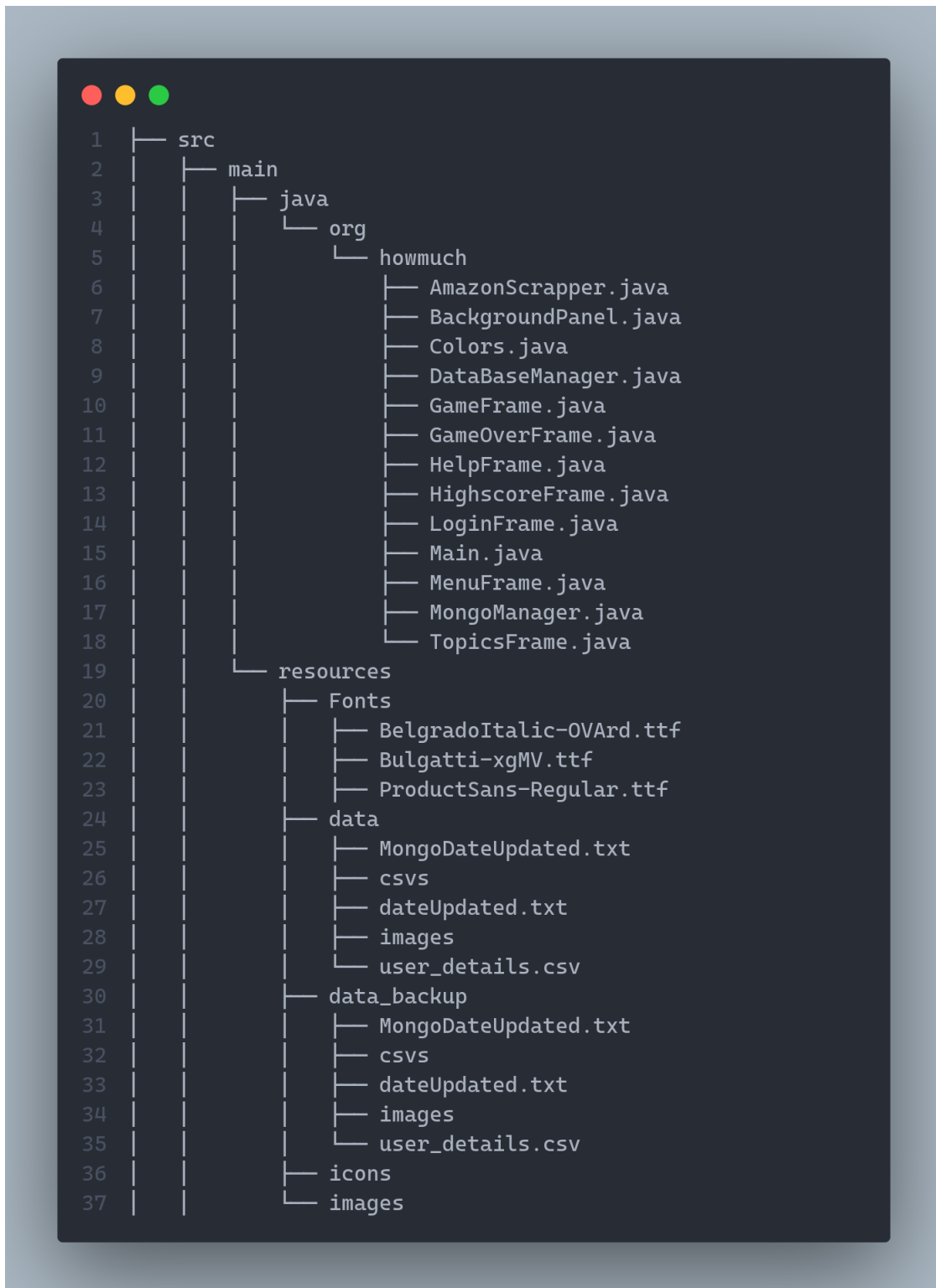


Figure 11:

10.2 TopicsFrame.java

10.3 MongoManager.java

10.4 MenuFrame.java

10.5 Main.java

10.6 LoginFrame.java

10.7 HighscoreFrame.java

10.8 HelpFrame.java

10.9 GameOverFrame.java

10.10 GameFrame.java

10.11 DataBaseManager.java

10.12 Colors.java

10.13 BackgroundPanel.java

10.14 AmazonScrapper.java

11 Conclusion and Topics Learnt

12 Code File Main.java

```
1 package org.howmuch;
2
3 import org.xml.sax.SAXException;
4 import javax.swing.*;
5 import javax.xml.parsers.ParserConfigurationException;
6 import java.awt.*;
7 import java.io.*;
8 import java.time.LocalDate;
9
10 // You have to extend the thread class to create a new thread for running the
    databases.
11 public class Main extends Thread {
12
13     // Statically defining important variables used throughout the game. They are
14     // statically defined coz they are used by other classes very often.
15     public static String[] Topics = new String[] { "Technology", "Fashion", "
Household", "Miscellaneous" };
16     public static String currentTopic = Topics[0];
17     static final int WIDTH = 1280, HEIGHT = 720;
18     static boolean maximized = false, isGuest = true, grantAccess = false,
isLocalDatabaseUpToDate = false,
19         isMongoUpToDate = false, usingMongo = false;
20
21     // Declaring Objects of other classes that we are going to call from main.
22     static LoginFrame loginFrame;
23     static MenuFrame menuFrame;
24     static HelpFrame helpFrame;
25     static HighscoreFrame highscoreFrame;
```

```
26 static TopicsFrame topicsFrame;
27 static GameFrame gameFrame;
28 static GameOverFrame gameOverFrame;
29
30 static Font buttonFont, textFont, password_font, options_font, emoji_font;
31 static JButton exit_btn, resize_btn, minimize_btn;
32 static JPanel basicButtons_pnl;
33
34 // These are the icons from where we get the resize, exit and the minimize
35 // button. They are custom made coz they look better,
36 // eliminate the need for the titlebar making the UI look cleaner, albeit less
37 // useful.
38 // They also let you have full control over what you want to do when they are
39 // pressed, and what you wanna call, which you cant do without them.
40 // You can also control now exactly the resizing behaviour of your software.
41 static ImageIcon exit = new ImageIcon("src/main/resources/icons/circle_delete.
png");
42 static Image exit_image = exit.getImage().getScaledInstance(25, 25, Image.
SCALE_SMOOTH);
43 static ImageIcon minimize = new ImageIcon("src/main/resources/icons/
circle_minus.png");
44 static Image minimize_image = minimize.getImage().getScaledInstance(25, 25,
Image.SCALE_SMOOTH);
45 static ImageIcon resizeUp = new ImageIcon("src/main/resources/icons/resize_3.
png");
46 static Image resizeUp_image = resizeUp.getImage().getScaledInstance(25, 25,
Image.SCALE_SMOOTH);
47 static ImageIcon resizeDown = new ImageIcon("src/main/resources/icons/resize_4
.png");
48 static Image resizeDown_image = resizeDown.getImage().getScaledInstance(25,
25, Image.SCALE_SMOOTH);
49
50 /**
51  * Creates fonts by instantiating the font objects with their respective fonts
52  * stored locally. Static and used everywhere. Its an important function and
53  * gets called in almost every class constructor.
54  */
55 public static void createFonts() {
56     try {
57         GraphicsEnvironment ge = GraphicsEnvironment.
getLocalGraphicsEnvironment();
58
59         // Used for Buttons Almost everywhere.
60         buttonFont = Font
61             .createFont(Font.TRUETYPE_FONT, new File("/run/media/
krishnaraj/Programs/Java/How Much/src/main/resources/Fonts/BelgradoItalic-OVard
.ttf"))
62             .deriveFont(50f);
63         // Used Mostly on the Login Page.
64         textFont = Font.createFont(Font.TRUETYPE_FONT, new File("/run/media/
krishnaraj/Programs/Java/How Much/src/main/resources/Fonts/MomcakeBold-WyonA.
otf"))
65             .deriveFont(50f);
66         // Used for password Entering
67         password_font = Font
68             .createFont(Font.TRUETYPE_FONT, new File("src/main/resources/
Fonts/CaeciliaLTPro45Light.TTF"))
69             .deriveFont(35f);
70         // Used only for Emojis
```

```
71         emoji_font = Font.createFont(Font.TRUETYPE_FONT,
72             new File("/run/media/krishnaraj/Programs/Java/How Much/src/
main/resources/Fonts/NotoEmoji-VariableFont_wght.ttf")).deriveFont(35f);
73         // Used to show the Price, needs to contain the Rupee symbol
74         options_font = Font
75             .createFont(Font.TRUETYPE_FONT, new File("/run/media/
krishnaraj/Programs/Java/How Much/src/main/resources/Fonts/ProductSans-Regular.
ttf"))
76             .deriveFont(35f);
77
78         // registering them locally, not required.
79         ge.registerFont(textFont);
80         ge.registerFont(buttonFont);
81         ge.registerFont(password_font);
82         ge.registerFont(emoji_font);
83         ge.registerFont(options_font);
84
85     } catch (FontFormatException | IOException e) {
86         e.printStackTrace();
87         System.out.println("Couldnt create the fonts. ");
88     }
89 }
90
91 /*
92  * Function to Create the resize, minimize and the exit button, they are all
93  * placed in a panel, so that you can move them around easily without the
hassle
94  * of moving each thing. Just move the panel. Here we define them.
95  */
96 public static void createBasicButtonPanel() {
97     basicButtons_pnl = new JPanel();
98     FlowLayout fl = new FlowLayout(FlowLayout.LEFT, 10, 0);
99     basicButtons_pnl.setLayout(fl);
100
101     exit_btn = new JButton();
102     exit_btn.setIcon(new ImageIcon(exit_image));
103     exit_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
104     exit_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
105     exit_btn.setBounds(new Rectangle(25, 25));
106     exit_btn.setFont(buttonFont.deriveFont(44f));
107     exit_btn.setFocusPainted(false);
108     exit_btn.setContentAreaFilled(false);
109     exit_btn.setOpaque(true);
110     exit_btn.setBorder(null);
111
112     resize_btn = new JButton();
113     if (Main.maximized) {
114         resize_btn.setIcon(new ImageIcon(resizeDown_image));
115     } else {
116         resize_btn.setIcon(new ImageIcon(resizeUp_image));
117     }
118     resize_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
119     resize_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
120     resize_btn.setBounds(new Rectangle(25, 25));
121     resize_btn.setFont(buttonFont.deriveFont(44f));
122     resize_btn.setFocusPainted(false);
123     resize_btn.setContentAreaFilled(false);
124     resize_btn.setOpaque(true);
125     resize_btn.setBorder(null);
```

```
126
127     minimize_btn = new JButton();
128     minimize_btn.setIcon(new ImageIcon(minimize_image));
129     minimize_btn.setAlignmentY(Box.CENTER_ALIGNMENT);
130     minimize_btn.setAlignmentX(Box.CENTER_ALIGNMENT);
131     minimize_btn.setBounds(new Rectangle(25, 25));
132     minimize_btn.setFont(buttonFont.deriveFont(44f));
133     minimize_btn.setFocusPainted(false);
134     minimize_btn.setContentAreaFilled(false);
135     minimize_btn.setOpaque(true);
136     minimize_btn.setBorder(null);
137
138     // Adding them to the panel here.
139     basicButtons_pnl.add(minimize_btn);
140     basicButtons_pnl.add(resize_btn);
141     basicButtons_pnl.add(exit_btn);
142 }
143
144 /**
145  * @param status = 1: Call Main Menu <br>
146  *                status = 2: Call Topic Selection<br>
147  *                status = 3: Call Help and Credits<br>
148  *                status = 4: View Highscores<br>
149  *                status = 5: Update Database<br>
150  *                status = 6: Start Game<br>
151  *                status = 7: Game over Screen<br>
152  *                status = 0: Exit Game<br>
153  *
154  *                Important Function as it decides to change to another frame,
155  *                provides some security with grantedAccess boolean,
156  *                and Also does the mandatory things that need to be done if
157  *                the
158  *                close button is pressed.
159  */
160 public static void changeFrame(int status) {
161     // Status is 0 when you wanna quit, so we gotta do some stuff before you
162     // quit,
163     // like creating the backup.
164     if (status == 0) {
165         // Create a local backup of the users file irrespective of what was
166         // done during
167         // gameplay.
168         DataBaseManager.createLocalDatabaseBackupOfUsers();
169
170         // If the user is a guest, then the index is less than 0, in which
171         // case dont
172         // update anything.
173         if (DataBaseManager.USER_INDEX < 0) {
174             System.out.println("You are a guest, so not updating anything. \n");
175         };
176         } else {
177             // If its a user, then update the user score. The index is known
178             // already in a
179             // static variable.
180             DataBaseManager.updateUserScore();
181         }
182     }
183
184     // This is what keeps track of when the last time was that the
```

```
database was
179     // updated. You dont need to update it every time you run the game.
180     String lastUpdateDate = "";
181
182     // Opening the Backup Date file and checking the last time it was
backed up.
183     File dateFile = new File(DataBaseManager.LOCAL_BACKUP_DATEFILE);
184     if (dateFile.exists()) {
185         try (BufferedReader br = new BufferedReader(new FileReader(
dateFile))) {
186             lastUpdateDate = br.readLine();
187
188             // If the database was backed up last today, then dont do it.
189             if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
190                 System.out.println("Backup DataBases are Up to Date!");
191             } else {
192                 // Else update it.
193                 DataBaseManager.createLocalDatabaseBackup();
194             }
195         } catch (IOException e) {
196             throw new RuntimeException(e);
197         } catch (NullPointerException exception) {
198             System.out.println("Nothing in the Date File. ");
199         }
200     } else {
201         // If the file itself doesnt exist, then clearly there doesnt
exist any backup,
202         // so we better back up at that point.
203         try {
204             DataBaseManager.createLocalDatabaseBackup();
205         } catch (Exception e) {
206             System.out.println("You havent really created the database yet
, so not creating backup either. ");
207         }
208     }
209     // Exit game
210     System.out.println("Thanks for Playing! ");
211     System.exit(0);
212 }
213 if (grantAccess) {
214     System.out.println("Access Granted!");
215     switch (status) {
216         case 1 -> {
217             // Showing Main Menu
218             grantAccess = false;
219             menuFrame = new MenuFrame();
220         }
221         case 2 -> {
222             // Showing the TopicsFrame
223             grantAccess = false;
224             topicsFrame = new TopicsFrame();
225         }
226         case 3 -> {
227             // Showing the Help Screen
228             grantAccess = false;
229             helpFrame = new HelpFrame();
230         }
231         case 4 -> {
232             // Showing Highscores
```



```
233         grantAccess = false;
234         highscoreFrame = new HighscoreFrame();
235     }
236     case 5 -> {
237         System.out.println("Updating Database");
238
239         // Instead of overwriting the files, or appending to them, as
they contain old
240         // data,
241         // we will just erase them altogether and create them again.
242         DataBaseManager.clearLocalDatabase();
243         MongoManager.clearMongoDb();
244
245         // Scrap everything and Start Saving
246         AmazonScrapper obj = new AmazonScrapper();
247         try {
248             AmazonScrapper.scrapAndSave();
249         } catch (Exception e) {
250             System.out.println("Couldnt update the database, there was
some problem. It was");
251             System.out.println(e.getMessage());
252         }
253         // Just copy everything to the backup either way.
254         DataBaseManager.createLocalDatabaseBackup();
255
256         File dateFile;
257
258         // Updating the Mongo and Local Database File.
259         dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
260         try (FileWriter f = new FileWriter(dateFile, false)) {
261             f.write(String.valueOf(LocalDate.now()));
262         } catch (IOException e) {
263             throw new RuntimeException(e);
264         }
265         dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
266         try (FileWriter f = new FileWriter(dateFile, false)) {
267             f.write(String.valueOf(LocalDate.now()));
268         } catch (IOException e) {
269             throw new RuntimeException(e);
270         }
271     }
272     case 6 -> {
273         // Showing Game Screen
274         grantAccess = false;
275         gameFrame = new GameFrame();
276     }
277     case 7 -> {
278         // This is only called by the gameframe, which has a timer,
which is what calls
279         // this function, and as its in a different class,
280         // you have to close the things from here coz that timer cant
access its parent
281         // class properties.
282         gameFrame.setVisible(false);
283         gameFrame.dispose();
284
285         // Show GameOverScreen
286         gameOverFrame = new GameOverFrame();
287     }
```

```
288         default -> {
289             // In Case something goes really wrong, just backup and exit.
290             DataBaseManager.createLocalDatabaseBackup();
291
292             // Exit game
293             System.out.println("Thanks for Playing! ");
294             System.exit(0);
295         }
296     }
297     } else {
298         System.out.println("Access Denied Who are you? What are you trynna do
299 here? ");
300         System.exit(0);
301     }
302 }
303
304 /*
305  * This function is overridden from the Thread class, coz its empty there, and
306  * thread.start calls this function.
307  * And this is where you put loops or something in case you wanna do something
308  * for ever as a game Loop and access data members stored somewhere else and
309  * written to by some other classes.
310  * The Job of this function here is important in that its the first function
311  * that is real multithread. It checks the database, and if they are not up to
312  * date, it updates them.
313  */
314 public void run() {
315     // Just establish the connection, and if thats not possible, then we
316 clearly
317     // arent gonna be using mongo.
318     usingMongo = MongoManager.establishConnectionWithMongo();
319
320     // Same logic as demod in changeFrame()
321     String lastUpdateDate = "";
322
323     // Checking the Local CSV Files
324     File dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
325     if (dateFile.exists()) {
326         try (BufferedReader br = new BufferedReader(new FileReader(dateFile)))
327         {
328             lastUpdateDate = br.readLine();
329             System.out.println(lastUpdateDate);
330             if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
331                 System.out.println("Local DataBases are Up to Date!");
332                 isLocalDatabaseUpToDate = true;
333             }
334         } catch (IOException e) {
335             throw new RuntimeException(e);
336         } catch (NullPointerException exception) {
337             System.out.println("Nothing in the Local Date File. ");
338         }
339     }
340
341     // Now check the mongodb database date file to check when was the last
342 time it
343     // was updated. Same Logic tho.
344     dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
345     if (dateFile.exists()) {
```

```
343         try (BufferedReader br = new BufferedReader(new FileReader(dateFile)))
344         {
345             lastUpdateDate = br.readLine();
346             System.out.println(lastUpdateDate);
347             if (lastUpdateDate.equals(String.valueOf(LocalDate.now()))) {
348                 System.out.println("Mongo DataBases are Up to Date!");
349                 isMongoUpToDate = true;
350             }
351         } catch (IOException e) {
352             throw new RuntimeException(e);
353         } catch (NullPointerException exception) {
354             System.out.println("Nothing in the mongo Date File. ");
355         }
356
357         // If say one of them is not updated, then we gotta scrap amazon.
358         if (!isLocalDatabaseUpToDate || (usingMongo && !isMongoUpToDate)) {
359
360             System.out.println("Beginning to Scrap Data From Amazon, as one of the
361             DataBases isnt updated. ");
362             if (!isLocalDatabaseUpToDate) {
363                 // As an edge case, if mongo isnt up to date, we dont wanna clear
364                 the local one.
365                 DataBaseManager.clearLocalDatabase();
366             }
367             if (usingMongo && !isMongoUpToDate) {
368                 // If the local one isnt up to date we dont wanna clear mongo.
369                 MongoManager.clearMongoDb();
370             }
371
372             // Scrap and save, as at this point we already know what works and
373             what doesnt,
374             // and what is updated and what isnt,
375             // AmazonScrapper class can figure out where to save stuff. After that
376             // everything would have to be updated.
377             AmazonScrapper obj = new AmazonScrapper();
378             try {
379                 AmazonScrapper.scrapAndSave();
380                 isLocalDatabaseUpToDate = true;
381
382                 // writing to the date file coz we must have updated at this point
383                 dateFile = new File(DataBaseManager.LOCAL_DATEFILE);
384                 try (FileWriter f = new FileWriter(dateFile, false)) {
385                     f.write(String.valueOf(LocalDate.now()));
386                 } catch (IOException e) {
387                     throw new RuntimeException(e);
388                 }
389                 System.out.println("Updated the local database, no need to depend
390                 on the backup anymore");
391
392                 if (usingMongo) {
393                     // Coz at this point it has to be, as we just scrapped and
394                     didnt get any erros.
395                     isMongoUpToDate = true;
396
397                     // writing to the date file coz we must have updated at this
398                     point
399                     dateFile = new File(DataBaseManager.LOCAL_MONGODATEFILE);
400                     try (FileWriter f = new FileWriter(dateFile, false)) {
```

```
395         f.write(String.valueOf(LocalDate.now()));
396     } catch (IOException e) {
397         throw new RuntimeException(e);
398     }
399     System.out.println("Updated the Mongo database, no need to
depend on the local one anymore");
400 }
401 } catch (Exception e) {
402     System.out.print("Couldnt update one of the databases, in the case
that one of them wasnt updated. ");
403     System.out.println(e.getMessage());
404 }
405
406     // This has to happen at this point as a forced minimum.
407     isLocalDatabaseUpToDate = true;
408 }
409 }
410
411 public static void main(String[] args) {
412
413     // This is so that the fonts are rendered correctly in Swing gui.
414     System.setProperty("awt.useSystemAAFontSettings", "on");
415     System.setProperty("swing.aatext", "true");
416
417     // This is to call the thread, so we can check the databases.
418     Main t1 = new Main();
419     t1.start();
420
421     // As the thread starts, we start the game. Usually it has to read from
the
422     // backup file if the database isnt updated yet. After which it would
start
423     // reading from there. As downloading the images and putting them in the
424     // database takes time and we cant wait that long, that job is
multithreaded.
425     // The use of the backup database is :
426     // 1. It has some basic images that are shipped with the jar file so in
case
427     // someone doesnt have internet, atleast they have something.
428     // 2. It is the fallback in case something goes wrong while doing or
reading
429     // something from one of the files.
430     // 3. It serves as the Primary database when we are updating the local
database,
431     // and we still need to show stuff to the user so they can play the game.
This
432     // is the most important one.
433     loginFrame = new LoginFrame();
434 }
435 }
```

Listing 1: Main Java file