

```
1 ...
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 from reportlab.pdfbase import pdfmetrics
16 from reportlab.pdfbase.ttfonts import TTFont
17 pdfmetrics.registerFont(TTFont('MiLanProVF',
18 r'C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\MiLanProVF.ttf'))
19 from reportlab.pdfgen import canvas
20 from reportlab.lib.pagesizes import A4
21 from reportlab.platypus import SimpleDocTemplate, Paragraph
22 from reportlab.lib.styles import getSampleStyleSheet
23 from reportlab.lib.units import cm
24 from reportlab.lib.enums import TA_JUSTIFY
25 from reportlab.lib.pagesizes import letter
26 from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image
27 from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
28 from reportlab.lib.units import inch
29 from tabulate import tabulate
30
31 print("FJH")
32 import sys, os
33 class HiddenPrints:
34     def __enter__(self):
35         self._original_stdout = sys.stdout
36         sys.stdout = open(os.devnull, 'w')
37     def __exit__(self, exc_type, exc_val, exc_tb):
38         sys.stdout.close()
39         sys.stdout = self._original_stdout
40         print()
41
42 def telegram_bot_sendtext(bot_message):
43     import requests
44     with HiddenPrints():
45         bot_token = '1215404401:AAEvVBwzogEh0vBaW5iSpHRbz3Tnc7fCZis'
46         bot_chatID = '680917769'
47         send_text = 'https://api.telegram.org/bot' + bot_token + '/sendMessage?
48 chat_id=' + bot_chatID + '&parse_mode=Markdown&text=' + bot_message
49         response = requests.get(send_text)
50         return response.json()
51
52 import getpass, time, pathlib, os, sqlite3
53 import os #library used to open the notepad application to display the sales records
54 if os.path.exists(r'userblock.zconf'):
55     print("Decrypting authentication blobs...")
56     print(" ")
57     p = Path('userblock.zconf')
58     p.rename(p.with_suffix('.txt'))
59 if os.path.exists(r'userblock.txt'):
```

```

59     userblock = open(r"userblock.txt","r") #Opening / creating (if it doesn't exist
60     valfn = 1
61 else:
62     valfn = 0
63 if os.path.exists(r'userblock.txt'):
64     userblock.close()
65     os.remove(r'userblock.txt')
66 elif os.path.exists(r'userblock.zconf'):
67     userblock.close()
68     os.remove(r'userblock.zconf')
69
70 if os.path.exists(r'DBFA.zconf'):
71     pass
72 else:
73     print("Getting the database online.....")
74     time.sleep(0.5)
75     con = sqlite3.connect(r'DBFA.db')
76     print("Rebuilding database..")
77     c = con.cursor()
78     #c.execute("DROP TABLE cust;")
79     c.execute("""CREATE TABLE IF NOT EXISTS cust
80             (custt INTEGER PRIMARY KEY,
81              custname TEXT,
82              email TEXT);""")  

83
84
85 ssh = sqlite3.connect(r'DBFA_handler.db')
86 ssh7 = ssh.cursor()
87 #c.execute("DROP TABLE cust;")
88 ssh7.execute("""CREATE TABLE IF NOT EXISTS sshandler
89             (prodid INTEGER,
90              ssstock INTEGER);""")  

91 ssh = sqlite3.connect('DBFA_handler.db')
92 ssh7 = ssh.cursor()
93 if os.path.exists(r'DBFA_handler.db'):
94     pass
95 else:
96     ssh7.execute("""CREATE TABLE IF NOT EXISTS sshandler
97             (prodid INTEGER,
98              ssstock INTEGER);""")  

99 ssh.close()
100
101
102 # FOR CUSTOMER RECORDS
103 def massmaintainer(inxstock): #defining a function to input data into the SQL
104     database's table
105     try:
106         idList =
107 [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
108 32,33,34,35,36,37,38,39,40]
109         ssxconn = sqlite3.connect(r"DBFA_handler.db")
110         ssxsql = 'DELETE FROM sshandler'
111         ssxcur = ssxconn.cursor()
112         ssxcur.execute(ssxsql)
113         ssxconn.commit()
114     except sqlite3.Error as error:
115         print("Failed to flush multiple records from sqlite table", error)
116
117     ssh = sqlite3.connect(r'DBFA_handler.db')

```

```

115     ssh7 = ssh.cursor()
116     namiex = ["TV 4K OLED 50", "TV FHD OLED 50", "8K QLED 80", "Redmi K20 PRO",
117               "Redmi K20", "Redmi Note 9 PRO", "POCOPHONE F", "Mi MIX ALPHA", "Wireless
118               Headphones", "Noise-Cancelling Wireless Headphones", "Essentials Headphones", "Gaming
119               Headphones", "Truly-Wireless Earphones", "Neckband-Style Wireless Earphones",
120               "Essentials Earphones", "Gaming Earphones", "30W Bluetooth Speakers", "20W Bluetooth
121               Speakers", "9""Essentials Bluetooth Speaker", "BOSE QC35", "Essentials Home Theatre",
122               "Wired Speaker - 5.", "Essentials Wired Speaker - STEREO", "Tactical Power Bank
123               30000mah", "5""Essentials Power Bank 0000mah", "Essentials Mouse", "Logitech RGB
124               Gaming Mouse with Traction & Weight Adjustment", "Tactical Essentials Keyboard",
125               "Mechanical Cherry MX (Red) RGB Gaming Keyboard", "Polowski Tactical Flashlight",
126               "OneFiber Wi-Fi Router AX7", "Mijia Mesh Wi-Fi Router", "lapcare 0W Laptop Adapter",
127               "lapcare 60W Laptop Adapter", "Spigen Phone Case(s)", "Essentials Phone Charger
128               150W", "HyperPower Type-C Gallium-Nitride Charger 100W", "ASUS Zephyrus G4 Gaming
129               Laptop", "L XPS 5 Content Creator's Laptop", "Hewlett-Packard Essential's Student's
130               Laptop (Chromebook)"]
131     for crrt in namiex:
132         gg = (namiex.index(crrt)) + 1
133         str = "insert into sshandler(prodid, ssstock) values({}, {})".format(gg, inxstock)
134         # io = (gg)
135         ssh7.execute(str)
136         ssh.commit()
137     ssh7.close()
138     time.sleep(1)
139     print("DBFA QuickVend service - Stock universally enforced to", inxstock)
140     time.sleep(1)
141
142 # FOR CUSTOMER RECORDS
143 def ssxupdatescript(inxssincremental, prodid):
144     ssh = sqlite3.connect('DBFA_handler.db')
145     ssh7 = ssh.cursor()
146     updatetr = """UPDATE sshandler SET ssstock = ssstock + ? WHERE prodid = ?"""
147     xrindicator = (inxssincremental, prodid)
148     ssh7.execute(updatetr, xrindicator)
149     ssh.commit()
150     ssh7.close()
151     time.sleep(1)
152     print("DBFA QuickVend Service - Stock added for", prodid, "as", inxssincremental)
153
154 def ssxsuperfetch():
155     ssh = sqlite3.connect('DBFA_handler.db')
156     ssh7 = ssh.cursor()
157     print("Connecting to QuickVend Service... ~~~") #SQL connection prompt
158     print("Store Stock:: ")
159     time.sleep(1.5)
160     #Re-writing to refresh connection
161     ssh7 = ssh.cursor()
162     ssh7.execute("SELECT DISTINCT prodid, ssstock FROM sshandler")
163     rows = ssh7.fetchall()
164     for row in rows:
165         print(row)
166         #print(" ")
167
168 def ssxstockmaintainer(prodid):
169     ssh = sqlite3.connect('DBFA_handler.db')
170     ssh7 = ssh.cursor()
171     updatetrtt = """UPDATE sshandler SET ssstock = ssstock - 1 WHERE prodid = ?"""
172     xrindicatortt = (prodid,)
```

```

161     ssh7.execute(updatetrtt, xrindicatortt)
162     ssh.commit()
163     ssh7.close()
164     time.sleep(1)
165     toaster.show_toast("DBFA QuickVend Service - Background Sync", duration = 0.3427)
166
167 def ssxstockmaster(prodid):
168     global ssxvarscheck
169     ssxvarscheck = 0
170     ssh = sqlite3.connect('DBFA_handler.db')
171     ssh.row_factory = lambda cursor, row: row[0]
172     ssh7 = ssh.cursor()
173     csrr = ("SELECT ssstock FROM sshandler WHERE prodid = (?);")
174     csrrxt = (prodid,)
175     ssh7.execute(csrr, csrrxt)
176     rows = ssh7.fetchall()
177     # print(rows) #debug point
178     values = ','.join(str(v) for v in rows)
179     print("Current product stock: ", values)
180     ssxdsccheck = "1 2 3 4"
181     if values in ssxdsccheck:
182         print("[Stock running out] Currently in stock: ", values, "pieces. Restock ASAP...")
183         ssxvarscheck = 1
184     elif values == "0":
185         ssxvarscheck = 2
186     elif values != "0":
187         ssxvarscheck = 1
188     time.sleep(1)
189     toaster.show_toast("DBFA QuickVend Service - Background Sync", duration = 0.3427)
190
191 #Values stored in two dictionaries
192 data = {"1":40000, "2":55000, "3":67000, "4":25000, "5":21000, "6":14000, "7":13000,
193 "8":220000, "9":4500, "10":17000, "11":1200, "12":3700, "13":4500, "14":2200,
194 "15":700, "16":2750, "17":6499, "18":1499, "19":799, "20":27000, "21":6750,
195 "22":2100, "23":1199, "24":3210, "25":989, "26":750, "27":1700, "28":600, "29":2175,
196 "30":890, "31":2100, "32":7158, "33":597, "34":347, "35":500, "36":300, "37":1097,
197 "38":80000, "39":87900, "40":23790}
198 namie = {"1":"TV 4K OLED 50", "2":"TV FHD OLED 50", "3":"8K QLED 80", "4":"Redmi K20 PRO", "5":"Redmi K20", "6":"Redmi Note 8 PRO", "7":"POCOPHONE F1", "8":"Mi MIX ALPHA", "9":"Wireless Headphones", "10":"Noise-Cancelling Wireless Headphones", "11":"Essentials Headphones", "12":"Gaming Headphones", "13":"Truly-Wireless Eadphones", "14":"Neckband-Style Wireless Earphones", "15":"Essentials Earphones", "16":"Gaming Earphones", "17":"30W Bluetooth Speakers", "18":"10W Bluetooth Speakers", "19":"Essentials Bluetooth Speaker", "20":"ULTRA Home Theatre", "21":"Essentials Home Theatre", "22":" Wired Speaker - 5.1", "23":" Essentials Wired Speaker - STEREO", "24":"Tactical Power Bank 30000mah", "25":"Essentials Power Bank 10000mah", "26":"Essentials Mouse", "27":"Logitech RGB Gaming Mouse with Traction & Weight Adjustment", "28":"Tactical Essentials Keyboard", "29":"Mechanical Cherry MX (Red) RGB Gaming Keyboard", "30":"Polowski Tactical Flashlight", "31":"OneFiber Wi-Fi Router AX17", "32":"Mijia Mesh Wi-Fi Router", "33":"lapcare 120W Laptop Adapter", "34":"lapcare 60W Laptop Adapter", "35":"Spigen Phone Case(s)", "36":"Essentials Phone Charger 10W", "37":"HyperPower Type-C Gallium-Nitride Charger 100W", "38":"ASUS Zephyrus G14 Gaming Laptop", "39":"L XPS 15 Content Creator's Laptop", "40":"Hewlett-Packard Essential's Student's Laptop (Chromebook)"}
199 namiex = ["TV 4K OLED 50", "TV FHD OLED 50", "8K QLED 80", "Redmi K20 PRO", "Redmi K20", "Redmi Note 9 PRO", "POCOPHONE F1", "Mi MIX ALPHA", "Wireless Headphones", "Noise-Cancelling Wireless Headphones", "Essentials Headphones", "Gaming Headphones", "Truly-Wireless Eadphones", "Neckband-Style Wireless Earphones", "Essentials Earphones", "Gaming Earphones", "30W Bluetooth Speakers", "20W Bluetooth Speakers",

```

```

"9""Essentials Bluetooth Speaker", "BOSE QC35", "Essentials Home Theatre", "Wired
Speaker - 5.", "Essentials Wired Speaker - STEREO", "Tactical Power Bank 30000mah",
"5""Essentials Power Bank 0000mah", "Essentials Mouse", "Logitech RGB Gaming Mouse
with Traction & Weight Adjustment", "Tactical Essentials Keyboard", "Mechanical
Cherry MX (Red) RGB Gaming Keyboard", "Polowski Tactical Flashlight", "OneFiber Wi-Fi
Router AX7", "Mijia Mesh Wi-Fi Router", "lapcare 0W Laptop Adapter", "lapcare 60W
Laptop Adapter", "Spigen Phone Case(s)", "Essentials Phone Charger 150W", "HyperPower
Type-C Gallium-Nitride Charger 100W", "ASUS Zephyrus G4 Gaming Laptop", "L XPS 5
Content Creator's Laptop", "Hewlett-Packard Essential's Student's Laptop
(Chromebook)"]
195 datax = [40000, 55000, 67000, 25000, 21000, 14000, 3000, 220000, 4500, 17000, 1200,
3700, 4500, 2200, 700, 2750, 6499, 1499, 799, 27000, 6750, 2100, 1199, 3210, 989,
750, 1700, 600, 2175, 890, 2100, 7158, 597, 347, 500, 300, 1097, 80000, 87900, 23790]
196 dataxr = []
197 for i in datax:
198     i = "₹" + '%d' % i
199     dataxr.append(i)
200 tablx = zip(namix, dataxr)
201 titlex = ["Product:", "Pricing:"]
202
203 '''
204 def floodscreen():
205     import cv2
206     image = cv2.imread("imagepx.png")
207     cv2.imshow("Initializing... ", image)
208     cv2.waitKey(3000)
209     cv2.destroyAllWindows()
210 '''
211
212 print("DBFA Billing Framework: Version 2.227 (alpha) ")
213 print(" Licensed under the GNU PUBLIC LICENSE")
214 print("<DBFA> Copyright (C) 2020 Pranav Balaji")
215 print(" ")
216 print("Visit: www.github.com/deltaonealpha/deltaBillingFramework for complete
licensing terms. ")
217 time.sleep(1.3)
218 command = "cls"
219 os.system(command)
220
221 def mainmenu(): #defining a function for the main menu
222     from colorama import init, Fore, Back, Style #color-settings for the
partner/sponsor adverts
223     init(convert = True)
224     print(Fore.RED) #red-line to indicate program start
225     print("-----")
226     print(Fore.WHITE)
227     print('A word from our partner: ' + Fore.BLACK + Back.CYAN + 'HOTEL? Trivago!')
#Text over here #Custom advert
228     print(Style.RESET_ALL)
229     print("-----DBFA standardised billing framework-----")
230     print("Input: ")
231     print("'1' to GENERATE INVOICE")
232     print("'2' to REGISTER CUSTOMER,")
233     print("'3' to VIEW REGISTERED CUSTOMERS,")
234     print("'4' to VIEW CUSTOMER PURCHASE RECORDS")
235     print("'5' to VIEW GENERATED INVOICES,")
236     print("'6' to REVIEW STORE LISTING,")
237     print("'7' to REVIEW LICENSING INFORMATION,")
238     print("'8' to VIEW OR UPDATE STOCK,")
239     print("and '9' to exit the framework.")

```

```

240     print("~ input TPM code to enter testing mode ~")
241     print("-----")
242     print()
243     print()
244
245
246 #void-setup phase
247 from datetime import datetime #for reporting the billing time and date
248 now = datetime.now()
249 dt_string = now.strftime("%d/%m/%Y %H:%M:%S") #datetime object containing current
date and time
250 logger = open(r"registry.txt", "a+") #Opening / creating (if it doesn't exist
already) the .txt record file
251 logger.write("----- \n")
252 logger.write("DBFA Billing Framework by Pranav Balaji\n")
253 logger.write("Licensed under the GNU PUBLIC LICENSE\n")
254 logger.write('ed')
255 logger.write("\n")
256 logger.write("Automated Store Registry:\n")
257 import time #to provide delays to make the system run seamlessly
258
259 xon = sqlite3.connect(r'DBFA_CUSTCC.db')
260 xbr7 = xon.cursor()
261 #c.execute("DROP TABLE cust;")
262 xbr7.execute("""CREATE TABLE IF NOT EXISTS custcc
(custt INTEGER PRIMARY KEY,
custname VARCHAR(500),
purchasecount INTEGER,
ptotalx INTEGER);""")
263 xon = sqlite3.connect('DBFA_CUSTCC.db')
264 xbr7 = xon.cursor()
265 if os.path.exists(r'DBFA_CUSTCC.db'):
266     pass
267 else:
268     xbr7.execute("""CREATE TABLE IF NOT EXISTS custcc
(custt INTEGER PRIMARY KEY,
custname VARCHAR(500),
purchasecount INTEGER,
ptotalx INTEGER);""")
269 xon.close()
270
271 conn = sqlite3.connect('DBFA.db')
272 if os.path.exists(r'DBFA.db'):
273     pass
274 else:
275     conn.execute('''CREATE TABLE COMPANY
        (ID INT PRIMARY KEY      NOT NULL,
        NAME           TEXT      NOT NULL,
        AGE            INT       NOT NULL,
        ADDRESS        CHAR(50),
        SALARY         REAL);'''')
276 def inserter(custt, custname, email): #defining a function to input data into the
SQL database's table
277     global conn
278     str = "insert into cust(custt, custname, email) values('%s', '%s', '%s')"
279     io = (custt, custname, email)
280     conn.execute(str % io)
281     conn.commit()
282     print("Customer", custname, "registered in store directory")
283

```

```

297 # FOR CUSTOMER RECORDS
298 def custcc(custtt, purchasecount, ptotalx): #defining a function to input data into
299     the SQL database's table
300     global xon
301     xon = sqlite3.connect(r'DBFA_CUSTCC.db')
302     xbr7 = xon.cursor()
303     str = "insert into custcc(custtt, purchasecount, ptotalx) values('%s', '%s',
304     '%s')"
305     io = (custtt, purchasecount, ptotalx)
306     xbr7.execute(str % io)
307     xon.commit()
308     xbr7.close()
309     print("FJHG")
310
311 # FOR CUSTOMER RECORDS
312 def updatescript(custtt, pincrement):
313     try:
314         xon = sqlite3.connect('DBFA_CUSTCC.db')
315         xbr7 = xon.cursor()
316         # hidden prints here ig
317         updatexr = """UPDATE custcc SET purchasecount = purchasecount + 1 WHERE custtt
318 = ?"""
319         updatexxr = """UPDATE custcc SET ptotalx = ptotalx + ? WHERE custtt = ?"""
320         indicator = (custtt)
321         xrindicator = (pincrement, custtt)
322         xbr7.execute(updatexr, indicator)
323         xbr7.execute(updatexxr, xrindicator)
324         xon.commit()
325         xbr7.close()
326     except sqlite3.Error as error:
327         pass
328
329 #floodscreen()
330 from win10toast import ToastNotifier
331 toaster = ToastNotifier()
332 toaster.show_toast("DFBA Framework Runtime Broker", "Please read operational and
333 licensing documentation prior to use.", duration = 2)
334 print("Heyy there!", 'ed')
335 time.sleep(1.34)
336 if valfn == 0:
337     logger.write("Oauth bypass - registering for security")
338     time.sleep(1)
339     print("-----DBFA standardised billing framework-----")
340     print("We highly value the security of our code, and our customers.")
341     toaster.show_toast("DFBA Framework Runtime Broker", "Unauthenticated login
342 detected!")
343     print("It has been detected that you have bypassed the login process.")
344     time.sleep(1)
345     print("Please obtain a genuine version of this program and provide proper
346 authentication.")
347     time.sleep(1)
348     print("-----")
349     time.sleep(5)
350     exit()
351
352 while(1): #while (always) true

```

```

351     mainmenu() #mainmenu
352     writer = ""
353     telethon = ""
354     time.sleep(0.3) #for a seamless experience
355     decfac = int(input("Select option: "))
356     #Bill Mode
357     if decfac == 1:
358         print()
359         print("Invoicing: ")
360         print()
361         custt = input("Enter customer ID (enter if unregistered): ")
362         logger.write("----- ") #writing to log file
363         logger.write("Cust. ID: \n")
364         logger.write(custt)
365         logger.write("\n")
366         logger.write("Date and time: \n") #including the date and time of billing (as
taken from the system)
367         logger.write(dt_string)
368         logger.write("\n")
369         abcd1 = 1
370         time.sleep(0.3) #for a seamless experience
371         telethon = "DBFA Billing System" + "\n" + dt_string + "\n" + "Customer: " +
custt + "\n"
372         writer = writer + "DBFA Billing Framework" + "\n" + "One-stop solution for
all your billing needs!" + "\n" + "\n" + "Billing time: " + dt_string + "\n" +
"Customer ID: " + custt + "\n" + "-----" + "\n" + "\n"
373         numfac = int(input("Number of purchased items: "))
374         time.sleep(0.34) #for a seamless experience
375         afac = 0
376         billiemaster = 0 #variable for totalling the price
377         while(afac!=numfac):
378             item = input("Enter purchased product code: ")
379             time.sleep(0.3) #for a seamless experience
380             if item in data:
381                 ssxstockmaster(item)
382                 if ssxvarscheck == 1:
383                     billiemaster+=data[item]
384                     print("Product purchased: ", namie[item], " costing: ",
data[item])
385                     print("---")
386                     priceprod = "₹" + '%d' % data[item]
387                     logger.write("Appending product to order: \n") #writing to file
388                     logger.write(namie[item])
389                     ssxstockmaintainer(item)
390                     logger.write("\n")
391                     writer = writer + "Purchased: " + "\n" + namie[item] + "\n" +
priceprod + "\n"
392                     else:
393                         print("This product is currently not in stock... The
inconvenience is regretted...")
394                         continue
395
396                     else:
397                         print("Invalid entry! Retry: ")
398                         print("---")
399                     afac+=1
400
401                     #tax = int(input("Enter the net tax %: ")) #comment and uncomment tkinter
lines to use GUI-based input
402                     print("18% standard GST - Invoicing!")

```

```

403     time.sleep(0.4) #for a seamless experience
404     #discount =
405     int(simpledialog.askstring(title="deltaSTOREMANAGER", prompt="Enter the discount
percentage: "))
406     discount = int(input("Enter discount % (if any): ")) #comment and uncomment
tkinter lines to use GUI-based input
407     print(discount, "% net discount - Invoicing!")
408     time.sleep(0.2) #for a seamless experience
409     print("Invoicing... DBFA")
410     time.sleep(0.4) #for a seamless experience
411     tota = (((18/100)*billiemaster)+billiemaster)
412     total = tota-((discount/100)*tota)
413     discountx = '%d' % discount
414     telethon = telethon + "\n" + "Tax amount: 18%" + "\n" + "Discount: " +
discountx + "%" + "\n" + "\n"
415     writer = writer + "\n" + "\n" + "-----" + "\n" + "Tax
amount: 18%" + "\n" + discountx + "\n" + "\n"
416     rupeesymbol = "\u20B9".encode("utf-8")
417     print("Invoice ID: ", abcd1, "; Total: ", total)
418     toaster.show_toast("DFBA Framework Runtime Broker:      Total billed for-
", str(total), duration = 1)
419     logger.write("Total amount billed for: \n") #writing to file
420     #regin.write("NET TOTAL: \n") #writing to file
421     telethon = telethon + "NET TOTAL: \n" + "\u20B9" + str(total) + "\n"
422     writer = writer + "NET TOTAL: \n" + str(total) + "\n"
423     logger.write(str(total))
424     logger.write("\n")
425     #regin.write(str(total))
426     #regin.write("\n")
427     updatescript(custt, total)
428     abcd1+=1
429     afac+=1
430     now = datetime.now()
431     dt_string = now.strftime("%d/%m/%Y %H:%M:%S") #datetime object containing
current date and time
432     daterey = (dt_string.replace("/", "")).replace(":", "")
433     namer = 'invoice'+daterey+'.pdf'
434     can = SimpleDocTemplate(namer, pagesize=A4,
435                             rightMargin=2*cm, leftMargin=2*cm,
436                             topMargin=2*cm, bottomMargin=2*cm)
437     #can.setFont("MiLanProVF", 24)
438     can.build([Paragraph(writer.replace("\n", "<br />"), getSampleStyleSheet()
['Normal'])])
439
440     import shutil
441     source = namer
442     destination =
443     r'C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\Generated_invoices'
444     dest = shutil.move(source, destination)
445     time.sleep(1.5) #for a seamless experience
446
447     import os, sys
448     #regin.close()
449     with HiddenPrints():
450         try:
451             sender = telegram_bot_sendtext(telethon)
452             print(sender)
453             except Exception:
454                 pass

```

```
454
455     print()
456 #Register Customer
457 elif decfac == 2:
458     print("Loading server connection....") #SQL connection prompt
459     time.sleep(0.4) #for a seamless experience
460     #conn.execute("select * from cust")
461     #takes values from the SQL database
462     cursor = conn.cursor()
463     cursor.execute("select * from cust")
464     results = cursor.fetchall()
465     idd = len(results)+1
466     print("Registering customer with ID: ", idd)
467     custname = input("Name: ")
468     email = input("Customer's E-mail ID: ")
469     inserter(idd, custname, email) #argumental function to insert values into the
SQL database
470     nullvalue = 0
471     custcc(idd, nullvalue, nullvalue)
472     print(" ")
473     logger.write("----- \n")
474     logger.write(" \n")
475     logger.write("Date and time: ") #including the date and time of billing (as
taken from the system)
476     logger.write(dt_string)
477     logger.write(" \n")
478     logger.write("New customer registered: ")
479     x = " custname: " + custname + " custemail: " + email + "\n"
480     logger.write(x)
481     logger.write("----- \n")
482     print("Customer ID", idd, "registered in directory.")
483     print("-----")
484     print(" ")
485     print(" ")
486     #conn.close()
487     time.sleep(1) #for a seamless experience
488 #VIEW ALL CUSTOMERS
489 elif decfac == 3:
490     print()
491     print("Loading server connection....") #SQL connection prompt
492     time.sleep(0.7) #for a seamless experience
493     print("The registered customers are: ")
494     #Re-writing to refresh connection
495     cur = conn.cursor()
496     cur.execute("SELECT * FROM cust")
497     rows = cur.fetchall()
498     for row in rows:
499         print(row)
500         print(" ")
501     toaster.show_toast("DFBA Superfetch Service", "Superfetch: Database
acessed!", duration = 2)
502     #takes values from the SQL database
503     ''
504     while row is not None:
505         print(row)
506         #row = conn.fetchone()
507         ''
508     logger.write("----- \n")
509     logger.write(" \n")
```

```

510     logger.write("Date and time: ") #including the date and time of billing (as
511     taken from the system)
512     logger.write(dt_string)
513     logger.write("\n")
514     logger.write("Customer registry accessed! \n")
515     logger.write("----- \n")
516     conn.close()
517     conn.close()
518     print()
519     print()
520     time.sleep(2) #delay for easy-table viewing
521
522     #View Customer Purchase Records
523     elif decfac == 4:
524         xon = sqlite3.connect(r'DBFA_CUSTCC.db')
525         xbr7 = xon.cursor()
526         xbr7.execute("SELECT * FROM custcc")
527         rows = xbr7.fetchall()
528         for row in rows:
529             print(row)
530             print(" ")
531             xbr7.close()
532             toaster.show_toast("DFBA Superfetch Service", "Superfetch: Database
533             acessed!", duration = 0.5)
534
535             #View Generated Bills
536             elif decfac == 5:
537                 #password verification as sales record is not to be shown to all;
538                 print("Password echo shall be supressed for security.")
539                 passw = getpass.getpass(prompt='Enter root password to view store activity
540                 registry: ', stream=None)
541                 logger.write("\n")
542                 logger.write("Date and time: ") #including the date and time of billing (as
543                 taken from the system)
544                 logger.write(dt_string)
545                 logger.write("\n")
546                 if passw == "root":
547                     time.sleep(1) #for a seamless experience
548                     print("Hold on, moneybags.")
549                     with HiddenPrints():
550                         try:
551                             sender = telegram_bot_sendtext(dt_string + "\n" + "Registry
552                             files accessed - DBFA SECURITY")
553                             print(sender)
554                         except Exception:
555                             pass
556                         time.sleep(0.4)
557                         print("There ya go:: ")
558                         time.sleep(0.2) #for a seamless experience
559                         logger.write("Log file access attempt - Oauth complete \n")
560                         logger.close() #to change file access modes
561                         logger = open("registry.txt", "r+")
562                         # Uncomment the below lines if the program has to be modified to show
563                         # the records in the shell itself and not externally
564                         # print(logger.read())
565                         # print()
566                         # print("Opening sales log externally now. ")
567                         time.sleep(1.4) #for a seamless experience
568                         os.startfile('registry.txt') #to open the external notepad
569
570             application

```

```

563     else:
564         logger.write(" \n")
565         logger.write("Date and time: ") #including the date and time of billing
566         (as taken from the system)
567         logger.write(dt_string)
568         logger.write(" \n")
569         time.sleep(1) #for a seamless experience
570         logger.write("Log file access attempt - Oauth failiure!!! \n")
571         print("Wrong, sneaky-hat. Try again: ")
572         print(" ")
573         print("Password echo shall be supressed for security.")
574         passw = getpass.getpass(prompt='Enter root password to view store
575         activity registry: ', stream=None)
576         if passw == "root":
577             time.sleep(1) #for a seamless experience
578             print("Hold on, moneybags.")
579             with HiddenPrints():
580                 try:
581                     sender = telegram_bot_sendtext(dt_string + "\n" +
582 "Registry files accessed - DBFA SECURITY: ATTEMPT 02")
583                     print(sender)
584                 except Exception:
585                     pass
586                     print("There ya go: ")
587                     time.sleep(0.6) #for a seamless experience
588                     logger.write(" \n")
589                     logger.write("Date and time: \n") #including the date and time of
590                     billing (as taken from the system)
591                     logger.write(dt_string)
592                     logger.write(" \n")
593                     logger.write("Log file access attempt - AUTHORIZATION SUCCESS
594                     \n")
595                     logger.close() #to change file access modes
596                     logger = open("log.txt", "r+")
597                     # print(logger.read())
598                     # print()
599                     # print("Opening sales log externally now. ")
600                     time.sleep(1.4) #for a seamless experience
601                     os.startfile('log.txt')
602             else:
603                 with HiddenPrints():
604                     try:
605                         sender = telegram_bot_sendtext(dt_string + "\n" + "[ACCESS
606                         DENIED!!] - Registry file - DBFA SECURITY [ACCESS DENIED!!]")
607                         print(sender)
608                     except Exception:
609                         pass
610                         print("Multiple Unsuccesfull Attempts Detected. Re-run the program to
611                         login now. ")
612                         logger.write("(MULTIPLE ATTEMPTS!): Log file access attempt -
613                         AUTHORIZATION FAILED!!! \n")
614                         time.sleep(1.4) #for a seamless experience
615                         print()
616                         print()
617                         #View Listing Option
618                         elif decfac == 6:
619                             print("Store listing (as per updated records): ")
620                             print(tabulate(tablx, headers = titlex, floatfmt = ".4f"))
621                         elif decfac == 8:

```

```

615     decsfactor = str(input("Enter 'a' to VIEW STOCK, 'b' to ADD STOCK and 'c' to
ENFORCE MASS STOCK: "))
616     if decsfactor == "a":
617         ssxsuperfetch()
618     elif decsfactor == "b":
619         objid = int(input("Enter the product ID to add stock for: "))
620         stockincrement = int(input("Enter the amount of stock to be added: "))
621         ssxupdatescript(stockincrement, objid)
622         print("Stock updated by", stockincrement, "for product ID:", objid)
623     elif decsfactor == "c":
624         ggrtrr = int(input("Enter stock to universally enforce: "))
625         massmaintainer(ggrtrr)
626
627     #Exit System
628     elif decfac == 9:
629         if os.path.exists(r'userblock.txt'):
630             userblock.close()
631             os.remove(r'userblock.txt')
632         if os.path.exists(r'userblock.zconf'):
633             userblock.close()
634             os.remove(r'userblock.zconf')
635         toaster.show_toast("DFBA Framework Runtime Broker", "Obfuscating program...", duration = 2)
636         def logoprintxrt():
637             print(" _____")
638             time.sleep(0.5)
639             print(" / /____/ / / /____/ / / /____/ / / /")
640             time.sleep(0.5)
641             print(" / / / / / / / / / / / / / / / / / /")
642             time.sleep(0.5)
643             print(" / / / / / / / / / / / / / / / / / / / /")
644             time.sleep(0.5)
645             print(" / / / / / / / / / / / / / / / / / / / /")
646             time.sleep(0.5)
647             print(" / / / / / / / / / / / / / / / / / / / /")
648             time.sleep(0.5)
649             print(" / / / / / / / / / / / / / / / / / / / /")
650             time.sleep(0.5)
651             print(" / / / / / / / / / / / / / / / / / / / /")
652             time.sleep(0.5)
653             print(" / / / / / / / / / / / / / / / / / / / /")
654             print(" ")
655             print(" ")
656             logoprintxrt()
657             time.sleep(2)
658             break
659             exit()
660             os.close('securepack.pyw')
661     elif decfac == 7:
662         print("Fetching latest licensing information.....")
663         print(" ")

```

```

664     print(" ")
665     logoprintxrt()
666     time.sleep(1.5)
667     print(" ")
668     print(" ")
669     print("DBFA by Pranav Balaji, 2020")
670     print(" ")
671     print("____ Licensing ____")
672     print("      GNU PUBLIC LICENSE - TERMS AND CONDITIONS")
673     print("      <deltaBillingFramework> Copyright (C) 2020 Pranav Balaji")
674     print("      This program comes with ABSOLUTELY NO WARRANTY; for details type"
675     "*show w*.")
676     print("      This is free software, and you are welcome to redistribute it")
677     print("      under certain conditions; type *show c* for details. ")
678     toaster.show_toast("DFBA Framework Runtime Broker", "©2020: DBFA by Pranav"
679     Balaji", duration = 1.5)
680     print(" ")
681     print(" ")
682     print(" ")
683     print("Visit: www.github.com/deltaonealpha/deltaBillingFramework for complete"
684     licensing terms. ")
685     print(" ")
686     print(" ")
687     aacsbcfac = int(input("Enter '1' to view complete licensing stuff or '2' to"
688     return: ""))
689     if aacsbcfac == 1:
690         print(" ")
691         print("Please select to open with your prefered text viewer/ edittor.")
692         os.startfile(r"C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\LICENSE")
693         print(" ")
694         print(" ")
695         print("-----")
696         #CIT
697         elif decfac == 10:
698             print("CIT INTERNAL TESTING MODE")
699             ffxfac = str(input("Enter CIT Testing Mode? (y/n):: "))
700             if ffxfac == "y":
701                 ffrxfac = str(input("Entering CIT may lead to data loss. Confirm entering"
702                 CIT? (y/n):: "))
703                 if ffrxfac == "y":
704                     print("DBFA CIT MODE")
705                     print("Initialising DELTA dependancies...")
706                     print(" ")
707                     print(" ")
708                     print("CIT INPUTABLES::")
709                     print("Enter '1' to CLEAR ALL CUSTOMER RECORDS")
710                     print("Enter '2' to exit CIT")
711                     citfacin = int(input("Waiting for input / | \ | / | \ |:: "))
712                     if citfacin == 1:
713                         # window.close()
714                         os.startfile(r'securepack.py')
715                     else:
716                         continue
717                         continue
718             else:
719                 continue
720             elif ffxfac == "n":
721                 print("Exiting CIT")

```

```
718         time.sleep(1)
719         continue
720     else:
721         print("Invalid input. . . . ")
722         time.sleep(1)
723
724     else:
725         continue
726
727 # End of program
728 # Available on github: www.github.com/deltaonealpha/DBFA
729 # https://deltaonealpha.github.io/DBFA/
```