UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI
FAKULTET
DEPARTMAN ZA MATEMATIKU
I INFORMATIKU

Petar Maletin, 286/21

InvoiceApp – Aplikacija za fakturisanje
- seminarski rad iz predmeta Skript jezici-

Novi Sad, 2022.

Sadržaj

Sadržaj 2
1. Uvod 3
2. Opis programa 4
2.1 Grafički korisnički interfejs 4
2.2 Baze podataka 6
2.3 Izvoz podataka u PDF 9
2.4 Grafici 14
2.5 Ostale metode 16
3. Zaključak 19
4. Literatura 20

1. Uvod

InvoiceApp predstavljaće aplikaciju sa grafičkim korisničkim interfejsom(eng. GUI – Graphical User Interface), koja će služiti za kreiranje faktura i njihov izvoz u PDF formatu ukoliko se korisnik odluči za to. Aplikacija će čuvati svoje podatke u MySQL bazama podataka.

Postojaće dva tipa korisnika aplikacije: administrator i korisnik.

Administrator će imati kontrolu nad celom aplikacijom, te će moći da kreira, menja i briše korisnike, ali i da prati ulaz/izlaz novca za svakog korisnika putem grafika, gde su prikazani podaci za sve korisnike aplikacije.

Kako bi korisnik mogao da pristupi samoj aplikaciji, administrator će morati da kreira nalog za njega. Od trenutka kada korisnik dobije pristupne parametre za prijavljivanje u aplikaciju, otvorene su mu sve mogućnosti.

Mogućnosti koje će korisnik imati jesu unos proizvoda u bazu podataka, kao i brisanje i promenu istih, prodaja proizvoda i kreiranje fakture i izvoz iste u PDF fajl. Takođe, drugi korisnik će moći da unese fakturu, čime će se njemu dodati iznos fakture u izlazni protok novca, dok će korisniku koji je kreirao fakturu dodati iznos fakture u ulazni protok novca. Ulaz i izlaz novca korisnik će moći pogledati na grafiku, ali za razliku od admina, korisnik će videti samo svoj grafik.

2. Opis programa

2.1 Grafički korisnički interfejs

Python omogućava rad sa mnoštvom biblioteka, među kojima se nalazi i biblioteka tkinter, koja omogućava rad sa grafičkim elementima, kako bi se korisnicima lakše prezentovala funkcija programa.

Tkinter ima mnoštvo elemenata, a za potrebe ove aplikacije, korišćeni su label, button, entry, frame i radiobutton.

```
def main_screen():
    global screen1
    screen1 = Tk()
    screen1.title("InvoiceApp")
    screen1.iconbitmap("C:\\Users\petar\Desktop\Seminarski rad A -
projekat\Photo\Icon.ico")
    screen1.configure(bg="gray")
    Label(screen1, text="", bg="gray",).pack()
    Label(screen1, text="Dobro dosli!", font=("Arial", 20), bg="gray",
fg="black").pack()
    Label(screen1, text="", bg="gray").pack()
    Label(screen1, text="", bg="gray").pack()
    frame = LabelFrame(screen1, text="Log in", padx=15, pady=10, bg="gray")
    frame.pack()
    Label(frame, text="", bg="gray").pack()
    global userName
    Label(frame, text="Korisnicko ime:", font=("Arial", 14),
bg="gray").pack()
    userName = Entry(frame, width=30)
    userName.pack()
    global password
    Label(frame, text="Lozinka:", font=("Arial", 14), bg="gray").pack()
    password = Entry(frame, width=30)
    password.pack()
    Label(frame, text="", bg="gray").pack()
    Button(frame, text="Ulogujte se", font=("Arial", 14), width=15,
height=2, command=log_in).pack()
```

Listing 1. Metoda main_screen

Primer elemenata biblioteke Tkinter imamo na listingu 1. Tu je prikazana metoda mainscreen, u kojoj pravimo početni ekran uz pomoć gore pomenute biblioteke. Prvo je potrebno napraviti objekat, u ovom slučaju screen1, na kom ćemo kreirati elemente. Takođe, moguće je promeniti naziv prozora, ikonicu i boju pozadine. Labele koristimo da bismo ispisali nešto, ali i da bismo

odvojili neke elemente, ukoliko ne goristimo metodu grid. Kako bismo dodatno organizovali elemente imamo element frame, koji sada postaje glavno polje za kreiranje elemenata.

Na njega smo dodali nekoliko labela, ali i polja za unos(Entry) i dugme(Button). Najvažnija metoda koju možemo pozvati za polje za unos jeste get metoda, koja nam vraća šta se nalazi u samom polju u trenutku njenog pozivanja, a isto tako najvažniji argument dugmeta jeste argument command, kojim pozivamo metodu koja će se izvršiti kada kliknemo na dugme.

```
    r = IntVar()
    r.set("1")
    Radiobutton(frame_faktura, text="Da.", bg = "gray", variable=r, value =
1, command=radiobut1).pack()
    Radiobutton(frame_faktura, text="Ne.", bg = "gray",variable=r, value =
2, command=radiobut2).pack()
```

Listing 2. Radiobutton – Tkinter

```
def radiobut1():
    r.set("1")

def radiobut2():
    r.set("2")
```

Listing 3. Radiobutton command metode

Radiobutton se, kao i ostali elementi, kreira na sličan način, s tim što kod njega moramo da imamo varijablu, pomoću koje pristupamo vrednosti elementa, što možemo videti na listingu 2. Kako bismo menjali vrednosti varijable r nakon klika na neki od radiobuttona, kreiramo metode(listing 3.), koje postavljaju r na određenu vrednost(u ovom slučaju 1 ili 2). Kada kliknemo na neki radiobutton, on pozove svoju funkciju definisanu argumentom command i tada se ona izvršava.

```
    screen_width = screen1.winfo_screenwidth()
    screen_height = screen1.winfo_screenheight()

    width = 300
    height = 400
    x = (screen_width/2) - (width/2)
    y = (screen_height/2) - (height/2)
    screen1.geometry('%dx%d+%d+%d' % (width, height, x, y))

    screen1.mainloop()
```

Listing 4. Centriranje prozora

Kako bismo dobili aplikaciju na centru ekrana, i kako bi se svaki prozor otvorio na istom mestu, kreiramo dve varijable i dodeljujemo im sirinu i visinu ekrana. Width i height su nam dimenzije aplikacije. X i y su zapravo koordinate, koje dobijamo na način koji je prikazan na

listingu 4. i na kraju podešavamo širinu, visinu i koordinate aplikacije pomoću metode geometry iz Tkintera. Mainloop je metoda kojom procesujemo prozor i prikazujemo ga.

```
def log_out_admin():
    screen2.destroy()
    main_screen()
```

Listing 5. Metoda za rešavanje viška prozora

Kako ne bismo imali previše aktivnih prozora, odnosno tkinter objekata, kreiramo nekoliko metoda, pomoću kojih uništavamo jedan objekat, a pozivamo drugu metodu, koja kreira drugi objekat i prikazuje ga. Tako izbegavamo nepotreban nered na radnom prostoru, ali i efikasnije koristimo memoriju. Jedna od tih metoda nalazi se na listingu 5.

2.2 Baze podataka

Za način čuvanja podataka odabrao sam baze podataka umesto fajlova, jer baze podataka omogućavaju lakše izmene, brisanje, ali i pretragu samih podataka, jer su svi povezani nekom kolonom tabele. Za potrebe aplikacije, koristio sam MySQL baze podataka.

Slika 1. MySQL

Koristeći aplikaciju MySQL Workbench kreirao sam bazu podataka sa istim nazivom kao i 4 tabele unutar nje, a te tabele su users, products, invoices i balance.

Users tabela sadrži podatke o korisnicima, a to su: user_id(ključni podatak), user_name, user_password, company_name, company_address, company_city, company_mail i company_bank_account.

Products tabela sadrži podatke o proizvodima, koje su korisnici uneli. Kolone u ovoj tabeli su sledeće: user_id, product_id, product_name, product_price, product_amount i product_measure.

Invoices tabela kreirana kako bi podaci o fakturama, koje korisnik pravi, bili na jednom mestu. Kolone ove tabele su: id, invoice_id, product_id, user_id, product_name, product_amount, product_price, product_measure i used.

Balance je veoma prosta tabela sa tri kolone: user_id, user_out i user_in.

Python je omogućio kreiranje modula, te sam zahvaljujući toj funkcionalnosti, odvojio kod kojim manipulišemo bazama podataka u poseban python fajl(BazaUpis.py).

```
import mysql.connector

bazaKonekcija = mysql.connector.connect(
  host="localhost",
  user="root",
  password="petar",
  database="invoiceapp"
)

mycursor = bazaKonekcija.cursor()
```

Listing 6. Konekcija na bazu podataka

Kako bismo pristupili MySQL bazi podataka, moramo napraviti konektor pomoću biblioteke mysql.connector. Kao što možemo videti na listingu 6. kreiramo varijablu kojoj dodeljujemo povratnu vrednost funkcije iz prethodno navedene biblioteke. Kao parametre za ovu funkciju prosleđujemo naziv hosta, naziv korisnika, lozinku i naziv baze. Na kraju, kreiramo varijablu mycursor koja je zapravo objekat, koji se kreira pomoću prethodno napravljene konekcije i metode cursor. Metoda cursor kreira objekat, pomoću kog dalje možemo manipulisati podacima.

Za manipulaciju podacima koristimo četiri osnovne ključne reči MySQL baza podataka, a to su: select, insert, update i delete. Na ove ključne reči dodaju se još i neke druge, koje ćemo videti u narednim listinzima.

```
def sel_all_kor():
  try:
    sql = "SELECT * FROM users"
    mycursor.execute(sql)
    data = mycursor.fetchall()
    return data
  except Exception as e:
    print(e)
```

Listing 7. MySQL Select

Metoda sel_all_kor, koju možemo videti na listingu 7. nalazi se u modulu BazaUpis, i u njoj možemo prvo zapaziti ključne reči try i except. Kako bismo izbegli neprijatne situacije, koje su veoma česte pri radu sa bazama podataka, preporučljivo je koristiti try i except. U try pokušavamo da izvršimo naredbu sql, koju smo napisali pomoću ključne reči select. Pored select imamo još i *, from i users. Zvezdica(*) predstavlja da želimo da uzmemo sve podatke

iz tabele(from) users. Na kraju to izvršavamo(execute) i uzimamo podatke sa fetchall i vraćamo ih na mesto odakle je metoda pozvana. U slučaju except ispisuje se greška u console.

```
def reg_kor_baza(id, username, password, name, address, city, mail, bank):
  try:
    sql = "INSERT INTO users (user_id, user_name, user_password,
company_name, company_address, company_city, company_mail,
company_bank_account) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
    val = (id, username, password, name, address, city, mail, bank)
    mycursor.execute(sql, val)
    bazaKonekcija.commit()
  except Exception as e:
    print(e)
```

Listing 8. MySQL Insert

Kao i za prethodni pristup bazi podataka, kreiramo posebnu metodu. Na listingu 8. možemo videti metodu reg_kor_baza, koja prima osam argumenata, što je ekvivalentno broju kolona u users tabeli naše baze podataka. Koristimo try i except kako bismo izbegli nepravilnosti pri radu. Pokušavamo da izvršimo kod u try. Prvo kreiramo varijablu sql, kojoj dodajemo vrednost tipa string, odnosno, komandu napisanu MySQL ključnim rečima. Insert into predstavljaju ključne reči pomoću kojih upisujemo u našu tabelu users. Kako bismo nešto upisali u tabelu, moramo navesti koje kolone upisujemo. Njih navodimo u zagradama. Na kraju komande, dodajemo vrednosti koje ćemo upisati u bazu. Da bi bilo preglednije, umesto vrednosti stavljamo %s, a nakon toga, kreiramo drugu promenljivu i dodajemo joj sve vrednosti koje smo prosledili kao argument funkcije. Da bismo izvršili naredbu, kao i u prethodnoj funkciji, pozivamo metodu execute i prosleđujemo joj ove dve varijable i na kraju pomoću funkcije commit upisujemo u tabelu. Ukoliko je pokušaj neuspešan, u consoli će se ispisati greška.

```
def upd_kor_baza(id, username, password, name, address, city, mail, bank):
  try:
    sql = "UPDATE users SET user_name = %s, user_password = %s,
company_name = %s, company_address = %s, company_city = %s, company_mail =
%s, company_bank_account = %s WHERE user_id = %s"
    val = (username, password, name, address, city, mail, bank, id)
    mycursor.execute(sql, val)
    bazaKonekcija.commit()
  except Exception as e:
    print(e)
```

Listing 9. MySQL Update

Kako bi naša aplikacija dobila na funkcionalnosti, i kako bismo menjali podatke o korisniku i podatke iz ostalih tabela, koristimo update MySQL ključnu reč. Na listingu 9. kreirali smo metodu upd_kor_baza i prosleđujemo joj argumente. Inicijalizujemo varijablu sql na vrednost stringa, koji zapravo predstavlja MySQL naredbu update. Pored update, koristimo i ključne

reči set i where. Set postavlja određene kolone na %s vrednosti, koje kasnije dodajemo u varijablu val. Ključna reč where nam omogućava da odredimo tačnu vrstu, koja će se izmeniti. Obično se za potrebe where koristi kolona koja predstavlja ključ te tabele, ali to nije nužno.

```
def del_kor_baza(id):
  try:
    sql = "DELETE FROM users WHERE user_id = %s"
    val = (id,)
    mycursor.execute(sql, val)
    bazaKonekcija.commit()
  except Exception as e:
    print(e)
```

Listing 10. MySQL Delete

Kada želimo da obrišemo nešto iz baze podataka, moramo veoma povesti računa, da ne bismo izbrisali sve podatke iz tabele. Kako bismo to sprečili, koristimo where ključnu reč i proveravamo određenu kolonu da li je jednaka nekoj vrednosti. Na listingu 10. kolona koju proveravamo jeste user_id. Takođe, pored delete, koristimo i from, što nas upućuje tabelu u kojoj želimo da obrišemo neki red. Kao i kod ostalih, koristimo execute metodu sa parametrima sql i val. Pored execute, koristimo i commit metodu, koja se koristi kod insert i update komandi, dok kod select koristimo fetchone ili fetchall metodu.

2.3 Izvoz podataka u PDF

Kako bi ova aplikacija dobila na funkcionalnosti, dodao sam i izvoz faktura u pdf fajl. Za potrebe izvoza, koristio sam biblioteku FPDF.

Pošto ova biblioteka ne podržava kreiranje tabela, morao sam da je override-ujem. Pomoću literature 3. uspeo sam da kreiram tabele u pdf fajlu(listing 11.).

```
def create_table(self, table_data, title='', data_size = 10, title_size=12,
align_data='L', align_header='L', cell_width='even',
x_start='x_default',emphasize_data=[],
emphasize_style=None,emphasize_color=(0,0,0)):
        default_style = self.font_style
        if emphasize_style == None:
            emphasize_style = default_style
        def get_col_widths():
            col_width = cell_width
            if col_width == 'even':
                col_width = self.epw / len(data[0]) - 1
            elif col width == 'uneven':
                col_widths = []
```

```
                for col in range(len(table_data[0])):
                    longest = 0
                    for row in range(len(table_data)):
                        cell_value = str(table_data[row][col])
                        value_length = self.get_string_width(cell_value)
                        if value_length > longest:
                            longest = value_length
                    col_widths.append(longest + 4)
            elif isinstance(cell_width, list):
                col_width = cell_width
            else:
                col_width = int(col_width)
            return col_width
        if isinstance(table_data, dict):
            header = [key for key in table_data]
            data = []
            for key in table_data:
                value = table_data[key]
                data.append(value)
                data = [list(a) for a in zip(*data)]
        else:
            header = table_data[0]
            data = table_data[1:]

        line_height = self.font_size * 2.5

        col_width = get_col_widths()
        self.set_font(size=title_size)

        if x_start == 'C':
            table_width = 0
            if isinstance(col_width, list):
                for width in col_width:
                    table_width += width
            else:
                table_width = col_width * len(table_data[0])
                margin_width = self.w - table_width
            center_table = margin_width / 2
            x_start = center_table
            self.set_x(x_start)
        elif isinstance(x_start, int):
            self.set_x(x_start)
        elif x_start == 'x_default':
            x_start = self.set_x(self.l_margin)
        if title != '':
            self.multi_cell(0, line_height, title, border=0, align='j',
ln=3, max_line_height=self.font_size)
            self.ln(line_height)
```

```
        self.set_font(size=data_size)
        y1 = self.get_y()
        if x_start:
            x_left = x_start
        else:
            x_left = self.get_x()
        x_right = self.epw + x_left
        if  not isinstance(col_width, list):
            if x_start:
                self.set_x(x_start)
            for datum in header:
                self.multi_cell(col_width, line_height, datum, border=0,
align=align_header, ln=3, max_line_height=self.font_size)
                x_right = self.get_x()
            self.ln(line_height)
            y2 = self.get_y()
            self.line(x_left,y1,x_right,y1)
            self.line(x_left,y2,x_right,y2)

            for row in data:
                if x_start:
                    self.set_x(x_start)
                for datum in row:
                    if datum in emphasize_data:
                        self.set_text_color(*emphasize_color)
                        self.set_font(style=emphasize_style)
                        self.multi_cell(col_width, line_height, datum,
border=0, align=align_data, ln=3, max_line_height=self.font_size)
                        self.set_text_color(0,0,0)
                        self.set_font(style=default_style)
                    else:
                        self.multi_cell(col_width, line_height, datum,
border=0, align=align_data, ln=3, max_line_height=self.font_size)
                self.ln(line_height)
        else:
            if x_start:
                self.set_x(x_start)
            for i in range(len(header)):
                datum = header[i]
                self.multi_cell(col_width[i], line_height, datum, border=0,
align=align_header, ln=3, max_line_height=self.font_size)
                x_right = self.get_x()
            self.ln(line_height)
            y2 = self.get_y()
            self.line(x_left,y1,x_right,y1)
            self.line(x_left,y2,x_right,y2)
            for i in range(len(data)):
                if x_start:
                    self.set_x(x_start)
```

```
                row = data[i]
                for i in range(len(row)):
                    datum = row[i]
                    if not isinstance(datum, str):
                        datum = str(datum)
                    adjusted_col_width = col_width[i]
                    if datum in emphasize_data:
                        self.set_text_color(*emphasize_color)
                        self.set_font(style=emphasize_style)
                        self.multi_cell(adjusted_col_width, line_height,
datum, border=0, align=align_data, ln=3, max_line_height=self.font_size)
                        self.set_text_color(0,0,0)
                        self.set_font(style=default_style)
                    else:
                       self.multi_cell(adjusted_col_width, line_height,
datum, border=0, align=align_data, ln=3, max_line_height=self.font_size)
                self.ln(line_height)
        y3 = self.get_y()
        self.line(x_left,y3,x_right,y3)
```

Listing 11. Override FPDF klase

```
def lista_proiz():
    kol1 = BazaUpis.prod_amount_baza(int(proiz_id.get()))
    if(kol1[0]>=int(proiz_kol.get()) and int(proiz_kol.get())>0):
        lista_id.append(int(proiz_id.get()))
        lista_kol.append(int(proiz_kol.get()))
        edit_proiz()
    else:
        messagebox.showerror(title="Fakturisanje", message="Neuspesno
fakturisanje!")
```

Listing 12. Metoda lista_proiz

```
def edit_proiz():
    kol1 = BazaUpis.prod_amount_baza(int(proiz_id.get()))
    kol2 = int(kol1[0]) - int(proiz_kol.get())
    BazaUpis.upd_proiz_baza(int(proiz_id.get()), kol2)
    messagebox.showinfo(title="Fakturisanje", message="Uspesno
fakturisanje!")
```

Listing 13. Metoda edit_proiz

Kako bismo kreirali fakturu, klikom na dugme dodaj u fakturu, pozivamo metodu lista_proiz(listing 12.), koja povlači podatke iz baze o količini proizvoda sa zadanom šifrom i ukoliko postoji tolika količina na stanju, dodaje na lista_id id tog proizvoda, a na lista_amount kolicinu unetu u polju za unos. I poziva metodu edit_proiz(listing 13.). U edit_proiz koristimo update funkciju kako bismo promenili količinu proizvoda na stanju za određen id, odnosno, umanjujemo postojeću količinu proizvoda za količinu koju smo dodali u fakturu.

```
def faktura_pdf():
    if r.get() == 1:
        pdf = PDF('P', 'mm', 'A4')
        pdf.add_page()
        pdf.set_auto_page_break(True, margin=15)
        pdf.set_font('Times', '', 10)
        x = datetime.datetime.now()
        naziv_pdf = str(x.year) + str(x.month) + str(x.day) + str(x.hour) +
str(x.minute) + str(x.second)
        data = []
        data2 = ['ID', 'Proizvod', 'Cena', 'Kolicina', 'Mera', 'Ukupno']
        data.append(data2)
        j = 0
        uk = 0.0
        for i in lista_id:
            p = BazaUpis.pdf_baza(i)
            ukupno = round(float(p[3]) * lista_kol[j], 2)
            uk = uk + float(ukupno)
            data1 = [str(p[1]), str(p[2]), str(p[3]), str(lista_kol[j]),
str(p[5]), str(ukupno)]
            BazaUpis.ins_inv_baza(naziv_pdf, i, p[2], p[3], lista_kol[j],
p[5], user_data[0])
            j = j + 1
            data.append(data1)

        balpod = BazaUpis.bal(user_data[0])
        ukbal = float(balpod[1]) + uk
        BazaUpis.upd_out_bal(user_data[0], ukbal)

        data2 = ['','','','','',str(uk)]
        data.append(data2)
        pdf.create_table(table_data = data, cell_width = 'even')
        pdf.ln(5)
        fk = "Faktura broj: " + naziv_pdf
        pdf.cell(15, 5, fk)
        pdf.ln(25)
        pdf.cell(0, 0, "M.P.", align='C')
        visina = 106 + 5 * j
        pdf.image("stamp.png", 80, visina, 50)
        naziv_pdf = naziv_pdf + ".pdf"
        pdf.output(name = naziv_pdf)
        lista_id.clear()
        lista_kol.clear()
```

Listing 14. Kreiranje fakture sa izvozom u PDF

```
elif r.get()==2:
        x = datetime.datetime.now()
        naziv_pdf = str(x.year) + str(x.month) + str(x.day) + str(x.hour) +
str(x.minute) + str(x.second)
        j = 0
        for i in lista_id:
            p = BazaUpis.pdf_baza(i)
            ukupno = round(float(p[3]) * lista_kol[j], 2)
            uk = uk + float(ukupno)
            BazaUpis.ins_inv_baza(naziv_pdf, i, p[2], p[3], lista_kol[j],
p[5])
            j = j + 1
        lista_id.clear()
        lista_kol.clear()
```

Listing 15. Kreiranje fakture bez izvoza u PDF

Metoda faktura_pdf(listing 14. i listing 15.) predstavlja kreiranje fakture i u zavisnosti od želje korisnika, izvozi fakturu u pdf fajl. Ukoliko korisnik želi da fakturu izveze u pdf(listing 14.), pomoću prethodno override-ovane klase FPDF(sada PDF), kreiramo objekat pdf. Njemu dodajemo novu stranicu metodom add_page i podešavamo margine, kao i automatsko prelamanje stranice, ali i font. Kako bi svaka faktura imala jedinstven naziv, odlučio sam se da naziv zapravo bude datum i vreme kreiranja iste, odnosno, ime fakture sačinjeno je od godine, meseca, dana, sata, minute i sekunde kreiranja pdf-a. Kako bismo dobili sve ove podatke koristimo biblioteku datetime. Takođe, uz pomoć objekta pdf kreiramo tabelu na stranici, te za kolone koristimo listu, koju dodajemo u listu, što znači da imamo listu lista. Takođe, u spoljašnju listu pomoću for petlje dodajemo podatke koji će biti ispisani u tabeli. Nakon što to uradimo, dodajemo fakturu u našu bazu podataka sa svim potrebnim argumentima. I ažuriramo naš protok novca, jer je korisnik kreiranjem fakture prodao svoje proizvode.

Ukoliko korisnik ne želi izvoz u pdf fajl(listing 15.), samo kreiramo naziv fakture i upisujemo ga u bazu u tabelu invoices sa svim argumentima koje treba da prosledimo i ažuriramo protok novca tom korisniku.

2.4 Grafici

Grafici su postali naša svakodnevnica, te nam je u većini slučajeva lakše analizirati neke podatke sa grafika, pogotovo ukoliko ih ima previše, nego „peške“. Pošto je ovo aplikacija ekonomskog tipa, smatram da je veoma bitno analiziranje podataka, zbog čega sam dodao grafike.

Grafike sam dodao pomoću biblioteka matplotlib i numpy.

Matplotlib direktno služi za crtanje grafika različitih oblika, dok numpy biblioteka služi za analiziranje prosleđenih podataka, sortiranje i još mnoštvo funkcija.

U aplikaciji imamo dve metode za grafike(administratorski i korisnički grafici).

```
def view_graph():
    labels = []
    user_out = []
    user_in = []
    userd = BazaUpis.sel_all_kor()
    for i in userd:
        baldata = BazaUpis.bal(i[0])
        usern = i[3]
        labels.append(usern)
        user_out.append(baldata[1])
        user_in.append(baldata[2])
    x = np.arange(len(labels))
    width = 0.35

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, user_out, width, label='Prodato')
    rects2 = ax.bar(x + width/2, user_in, width, label='Kupljeno')
    ax.set_title('Admin grafikon')
    ax.set_ylabel('Novac(din)')
    ax.set_xlabel('Naziv firme')
    ax.set_title('Grafik po protoku novca')
    ax.set_xticks(x, labels)
    ax.legend()

    ax.bar_label(rects1, padding=3)
    ax.bar_label(rects2, padding=3)

    fig.tight_layout()

    plt.show()
```

Listing 16. Kreiranje administratorskog grafika

Kao što smo naveli u uvodu, administrator ima uvid u podatke svih korisnika. Kako ne bismo imali beskorisne grafike, odlučio sam da kreiram grafike za protok novca korisnika. Ti podaci sačuvani su u našoj bazi podataka, tačnije u tabeli balance. U toj tabeli, kao što je i ranije navedeno, imamo tri kolone: user_id, user_out, user_in. User_out kolona predstavlja novac, koji je korisnik stekao prilikom prodaje svojih proizvoda, dok user_in kolona predstavlja novac, koji je korisnik potrošio kupujući proizvode drugih korisnika putem fakture. Na listingu 16. možemo videti funkciju graph_view u kojoj kreiramo naš administratorski grafik. Kako bismo nešto prikazali na grafiku, moramo prvo prikupiti podatke, te pozivom metode sel_all_kor iz modula BazaUpis dobijamo listu, u kojoj se nalaze liste sa podacima korisnika. Kako bi dobili podatke iz balance tabele prolazimo kroz for petlju i za svaki user_id pozivamo metodu bal iz istog modula. Kreirali smo prethodno tri liste, i na njih dodajemo naše podatke. Labels lista predstavlja nazive kompanija, i ona će biti zapravo na x osi, dok će druge dve liste biti zapravo protok novca u zavisnosti od kompanije. Uz pomoć numpy biblioteke sortiramo podatke iz liste labels i nakon toga kreiramo objekte mathplotliba, uz pomoć kojih kreiramo grafik.

```
def graph_user():
    baldata = BazaUpis.bal(user_data[0])
    usern = BazaUpis.name_baza(user_data[0])
    labels = []
    labels.append(usern[0])
    user_out = []
    user_in = []
    user_out.append(baldata[1])
    user_in.append(baldata[2])

    x = np.arange(len(labels))
    width = 0.35

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, user_out, width, label='Prodato')
    rects2 = ax.bar(x + width/2, user_in, width, label='Kupljeno')

    ax.set_title('Korisnik grafikon')
    ax.set_ylabel('Novac(din)')
    ax.set_title('Grafik po protoku novca')
    ax.set_xticks(x, labels)
    ax.legend()

    ax.bar_label(rects1, padding=3)
    ax.bar_label(rects2, padding=3)

    fig.tight_layout()

    plt.show()
```

Listing 17. Kreiranje korisničkog grafika

Za razliku od administatorskog grafika, koji ima podatke svih korisnika iz tabele bal na grafiku, na korisničkom grafiku biće prikazani podaci iz tabele balance samo za tog korisnika, što možemo videti na listingu 17.

2.5 Ostale metode

Pored svih ovih funkcionalnosti, imamo još nekoliko metoda koje su od krucijalnog značaja. Neke od njih su zapravo sistem za prijavljivanje, ali i sistem za kupovinu, odnosno unos fakture drugog korisnika.

Sistem za prijavljivanje je veoma jednostavan. Postoje dva tipa prijava: admin i korisnik. Ukoliko se admin prijavljuje, on mora da unese svoj username i password, kao i korisnik, ali se zapravo prvo ispituje da li se admin prijavljuje.

Ukoliko se admin ne prijavljuje, prelazi se na korisnika. Da bi se korisnik ulogovao povlačimo podatke iz tabele users i proveravamo da li postoji korisnik sa zadatim username-om. Ukoliko postoji, proveravamo da li je prosleđena šifra ispravna, te ukoliko nije ispisuje se greška u messagebox-u, koji smo import-ovali uz pomoć tkinter biblioteke. Izvorni kod sistema za prijavljivanje se nalazi u listingu 18.

```
def log_in():
    global user_data
    b = 0
    if (userName.get() == "admin" and password.get() == "admin"):
        screen1.destroy()
        admin_screen()
        b = 1
    else:
        user_data = BazaUpis.log_in_baza(userName.get(), password.get())
        b = 1
        screen1.destroy()
        user_screen()
    if b == 0:
        messagebox.showerror(title="Prijavljivanje", message="Neuspesna
prijava")
```

Listing 18. Sistem za prijavljivanje

```
def unos_fakture():
    try:
        fakdata = BazaUpis.sel_inv_baza(int(Brfak.get()))
        used = False
        baldata = BazaUpis.bal(user data[0])
        suma = 0.0
        for i in fakdata:
            suma = suma + float(i[3] * i[4])
            if i[7]==1:
                used=True
        suma = suma + float(baldata[2])
        if used==False:
            BazaUpis.upd_in_bal(user_data[0], suma)
            for i in fakdata:
                BazaUpis.used(i[0])
        else:
            messagebox.showerror("Neuspesno", "Faktura je uneta.")
    except Exception as e:
        print(e)
```

Listing 19. Sistem za kupovinu/unos fakture

Kako bi ova aplikacija upotpunila svoju funkcionalnost, logičan sled radnji jeste da korisnici mogu da prodaju/kupuju proizvode između sebe. Upravo to omogućava sistem za kupovinu/unos fakture, koji se nalazi na listingu 19. Kao i kod svih rizičnijih radnji, koristimo try i except zbog sigurnosti od pucanja programa i silnih grešaka. Šta zapravo radimo u metodi unos_fakture? Zapravo je to veoma jednostavan algoritam. Povlačimo podatke iz tabele invoices, koja se nalazi u našoj bazi podataka. Nakon toga, postavljamo promenljivu used na false. Takođe, povlačimo podatke i iz tabele balance, kako bismo promenili status kolone user_in za određenog korisnika. Kroz for petlju množimo količinu proizvoda i njihovu cenu, te ih sabiramo sa prethodnom vrednošću varijable suma. Takođe, proveravamo da li je faktura već iskorištena, kako bismo sprečili zloupotrebu. Ukoliko faktura nije upotrebljena, ažuriraćemo status naše kolone user_in za datog korisnika, ali i ažuriramo kolonu used, te postavljamo njenu vrednost na 1 što označava da je iskorišćena, i da više neće moći biti iskorišćena. Ukoliko je faktura upotrebljena, u messagebox-u ispisaće se greška.

3. Zaključak

Kroz ovaj izvorni kod, kreirali smo veoma funkcionalnu aplikaciju sa grafičkim korisničkim interfejsom. Verujem da se algoritam koji je primenjen ovde, primenjuje i u mnogo većim trgovačkim aplikacijama, te smatram da je ovo veoma mala, ali kompaktna i funckionalna aplikacija, koja može biti unapređena.

U ovoj aplikaciji postoje dva tipa korisnika: administrator i korisnik.

Administrator je zapravo upravljač ove aplikacije. On kreira korisnike i daje im mogućnost unošenja svojih proizvoda, prodavanja istih, ali i kupovinu tuđih putem faktura. Svakako, administrator ima mogućnost da menja podatke o korisniku, ali i da ga obriše. Kako bi administrator imao uvid u sve radnje korisnika i kako ne bi došlo do zloupotrebe, može pratiti protok novca putem grafika, za svakog korisnika ponaosob. Grafici su stupičastog tipa.

Korisnik, za razliku od administratora, ima malo ograničenije mogućnosti, koje su i dovoljne za solidnu trgovinu. Korisnik može da dodaje, briše i ažurira svoje proizvode, prodaje iste kreiranjem faktura, ali i da unosi fakture koje su drugi korisnici kreirali. Fakture mogu biti unete samo jednom, dok je moguće kreirati bezbroj istih, sve dok ima proizvoda na stanju. Korisnik za razliku od administratora, na stupičastom grafiku vidi samo podatke o sopstvenom protoku novca.

Kada sam pomenuo unapređenje aplikacije, mislim da bi optimizacija, proširivanje baze podataka i poboljšanje grafičkog izgleda bili prvi na listi za isto. Svakako dodavanje još funkcionalnosti ne bi bilo da odmet, ali svakako su u ovaj program uključene većine funkcija, koje se zapravo i upotrebljavaju u većim aplikacijama ovog tipa.

4. Literatura

1. Predavanja u okviru predmeta Seminarski rad A, https://www.dmi.uns.ac.rs/
2. Python Software Foundation, https://www.python.org/
3. Python Tutorial, https://www.w3schools.com/python/default.asp
4. MySQL Tutorial, https://www.w3schools.com/mysql/mysql_sql.asp
5. Bvalgard – Github, https://github.com/bvalgard/create-pdf-with-python-fpdf2
6. Chart Explorers YouTube Tutorials, https://www.youtube.com/c/ChartExplorers/featured