Aplikacija za rezervacije u hotelu

Senković, Ivan

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:910978

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Jednopredmetni preddiplomski studij informatike

Ivan Senković

Aplikacija za rezervacije u hotelu

Završni rad

Mentor: doc. dr. sc. Miran Pobar

Rijeka, rujan 2022.

FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA
SVEUČILIŠTE U RIJECI

Rijeka, 30.5.2022.

Zadatak za završni rad

Pristupnik: Ivan Senković

Naziv završnog rada: Aplikacija za rezervacije u hotelu

Naziv završnog rada na eng. jeziku: Hotel reservations application

Sadržaj zadatka: Proučiti biblioteke i alate koje se uobičajeno koriste za izradu desktop aplikacija sa grafičkim sučeljem u programskom jeziku Python. Predložiti i opisati arhitekturu sustava te opisati alate koji se koriste za razvoj. Izraditi prototip aplikacije koja omogućuje unos i izmjenu podataka o gostima hotela te izdavanje računa. Opisati važne dijelove implementacije aplikacije sa dijelovima koda koji su korišteni u razvoju.

Mentor

Doc. dr. sc. Miran Pobar

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 30.5.2022.

(potpis pristupnika)

SADRŽAJ

SAŽETAK

U završnom radu opisan je razvoj aplikacije koja služi za check-in i check-out gostiju u hotel, te za evidenciju gostiju koji su trenutno u hotelu. Aplikacija je razvijena u programskom jeziku Python, a za izradu grafičkog sučelja aplikacije korišten je Python modul ili paket „tkinter“. U aplikaciji korisnik ima mogućnost unosa i ažuriranja podataka o gostu koji boravi u hotelu i informacije o cijenama i smještajnim jedinicama. Na kraju boravka gostu se ispiše račun koji je spremljen lokalno na računalo u pdf obliku pod brojem rezervacije. Za ispis računa koristi se modul „reportLab“ koji je jedan od mnogih biblioteka za izradu računa u obliku .docx ili .pdf. Za izradu aplikacije korišten je IDE pycharm koji je odabran zbog osobne preferencije, a za bazu podataka koriti se SQLite.

Ključne riječi: tkinter, aplikacija, evidencija, izdavanje računa, pycharm, Python, SQLite, pdf, spremanje podataka.

UVOD

Hotel kao objekt mora imati informacije o gostima koji u njemu trenutno borave i bez odgovarajuće aplikacije vođenje evidencije bi bilo mnogo teže. Pošto je aplikacija automatizirana, to jest sve je povezano bazom i ostalim funkcijama, proces prijave i odjave gosta u hotel brži je i efikasniji. Informacije koje ova aplikacija sprema u bazu mogu koristiti također i ostali odjeli u hotelu što je još jedan od razloga za izradu aplikacije.

Početni cilj rada je na bio izrada aplikacije za vođenje evidencije rezervacija gosti koji borave u hotelu, koji je proširen pa je evidencija samo mali dio onoga što program može na kraju raditi. Razvijena je desktop aplikacija s grafičkim sučeljem s obzirom da će se aplikacija koristiti na takvom radnom mjestu gdje nije potreban pristup preko weba.

Aplikacija je zamišljena na način da se korisnik, to jest djelatnik, prijavi putem svojeg korisničkog imena i lozinke čime nakon toga započinje svoj rad s prijavljivanjem gostiju u hotel ili objekt. Dodana mogućnost je ne samo da običan korisnik ima pristup nego tako i njegov nadređeni koji ima posebnu funkciju izmjenu cjenika soba ovisno o potrebi, bilo to drugo godišnje doba ili nekakva posebna ponuda.

U sljedećim cjelinama opisan je detaljan proces rada aplikacije od same prijave pa sve do izdavanja računa prilikom odlaska gosta iz objekta. Opisan je detaljan cilj uporabe svake biblioteke i zašto je korišten određeni pristup umjesto nekog drugog. Sve je zajedno povezano s bazom podataka i više tablica koje služe spremanju podataka o gostima, cijenama soba, kategorijama i brojevima soba ovisno o kategoriji. Na kraju rada opisan je proces izdavanja računa zato što je to u praksi službeno zadnji dio koji djelatnik ima s gostom koji boravi u objektu. U samom zaključku rada rezimiran je cijeli rad i mišljenje o odabiru ovakve teme kao i popis literature, slike korištene i dijelovi kodova.

KORIŠTENE TEHNOLOGIJE

Prva i najvažnija tehnologija korištena je tkinter koji je zaslužan za cijeli izgled aplikacije. Za razliku od nekih ostalih biblioteka za izradu grafičkih sučelja, tkinter nema svoj službeni program koji pomaže kod definiranja izgleda to jest pozicioniranja elemenata na ekranu ili same funkcionalnosti.

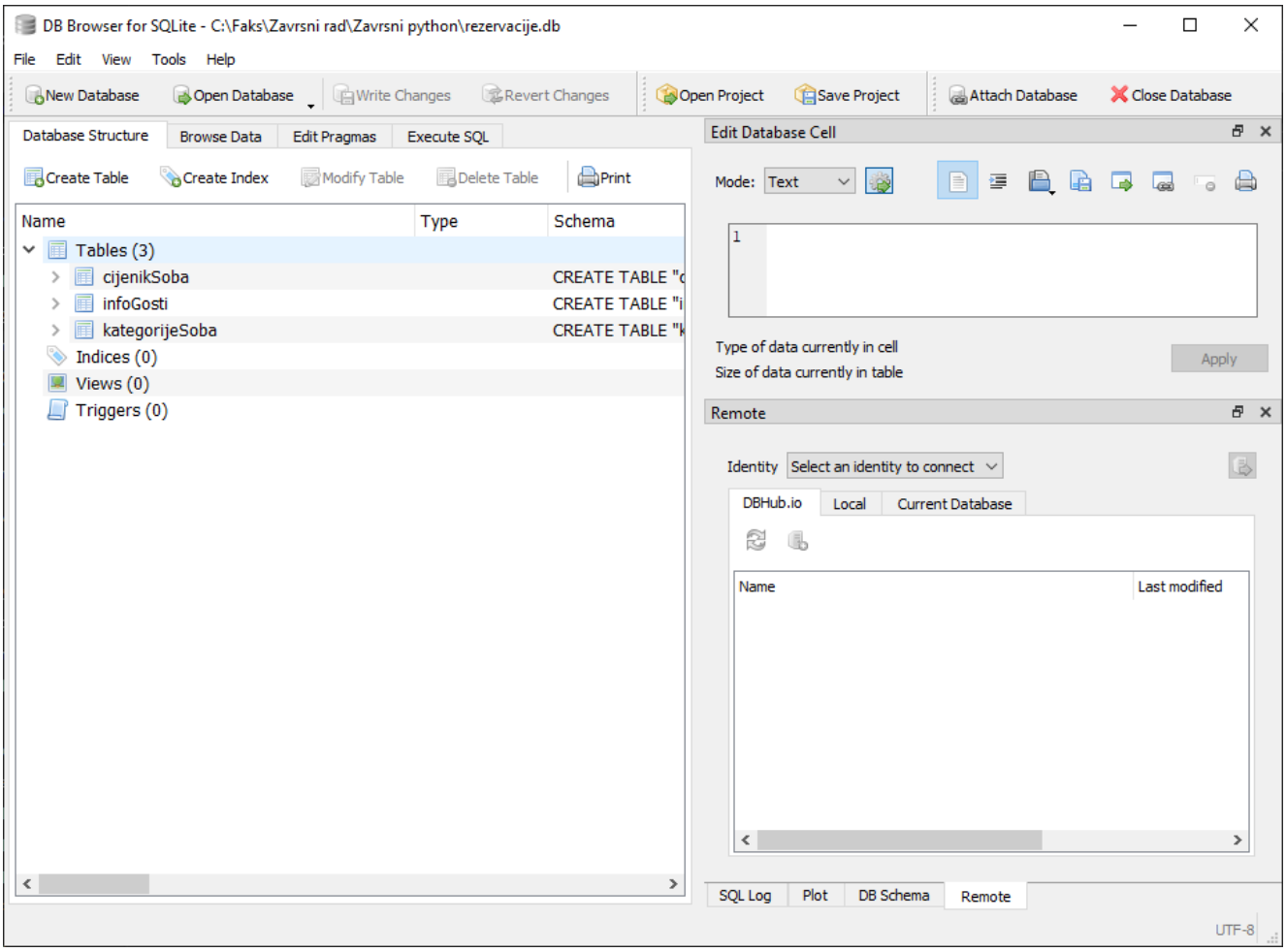
Jedna od biblioteka koje je pri početku bila korištena je „openpyxl“ koja služi za spremanje podataka koje korisnik odredi, u excel tablicu. Ova biblioteka bila je nepouzdana zato što pri razvoju se vidjelo da nije napravljena za manipulirati podatcima koji su zapisano nego većinom samo za pregled. Sljedeća biblioteka je „tkcalendar“ koja proširuje mogućnosti tkintera, ali je zasebna i sama po sebi služi za grafički odabir datuma. Biblioteke za mogućnost korištenja datuma i vremena u samom kodu su „datetime“ i „time“ preko kojih se može prikazati na primjer današnji datum i vrijeme pa tako i kako će oni izgledati u aplikaciji. Bitna tehnologija koja je korištena je SQLite koja se mora uvesti (eng. import) kako bi se moglo upravljati bazom podataka bez koje ova aplikacija ne bi imala traženu funkcionalnost. Zadnja biblioteka koja je korištena je „reportlab“ koja nam služi za kreiranje pdf dokument izravno iz samog koda s time da se cijeli izgleda konačnog pdf dokumenta mora tako rečeno na slijepo iz koda napraviti.

```
from tkinter import *
from tkcalendar import DateEntry
import tkinter.messagebox
from tkinter import ttk
from tkinter import messagebox
import datetime
import time
import sqlite3
from login import *
import os
import numpy as np

from reportlab.platypus import SimpleDocTemplate, Table, Paragraph, TableStyle
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import getSampleStyleSheet
```

Slika 1. Korištene biblioteke

Na kraju koristim uvođenje drugim Python kodova na isti način na koji uvodimo ostale biblioteke. To radimo zato što na našem primjeru imamo prozor za prijavu koji je bolje pustiti u svojoj vlastitoj zasebnoj datoteci. Za pregled i kreiranje baze podataka korišten je program

„DB browser“ koji se koristi u mnogo slučajeva za SQLite zato što nema svoj program za kreiranje baza podataka. DB Browser služio sam za izrade baze podataka i za izradu tablica unutar baze zato što je efikasniji način rada. Najbitnije opcije su kreiranje nove baze, kreiranje i brisanje tablica, uređivanje već postojećih tablica.



Slika 2. DB Browser program

Tkinter

Tkinter je Python biblioteka koja omogućuje izradu aplikacija s grafičkim korisničkim sučeljem. Tkinter ima svoje razne biblioteke koje služe za prikaz standardnih grafičkih kontrola i prozora, npr. za ručni unos i prikaz polja, unos pomoću već deklariranih vrijednosti, odabir datuma i tako dalje. Podržava Windows, macOS i razne Unix platforme. Ostale GUI biblioteke nude svoj vlastiti program za grafičku izradu sučelja, ali Tkinter sam odabrao zbog dobre prakse za kasnije korištenje ostalih alata u struci jer smatram da je dobro kao korijen za kasnije korištenje.

Tkinter nije jedna biblioteka nego se sastoji od raznih modula koji se koriste po potrebi. Tk je paket koji se koristi za sve ono što korisnik može vidjeti, ali s time da sada ima nova

verzija koja se zove „themed Tk“ ili ttk koja je korištena zbog boljeg izgleda pojedinih polja u aplikaciji. Kada Python aplikacija koristi klase u Tkinteru, kao na primjer za napraviti neko polje za unos, tkinter modul prvo sastavi Tk/Tcl niz naredba, nakon čega taj niz komanda on pošalje u svoj unutrašnji vlastiti modul koji na kraju pozove svoj interpreter da provjeri. Tcl je interpreterski programski jezik kao i Python koji je implementiran u „C“ jeziku.

Implementiranje ttk vrši se uvođenjem biblioteke putem „from tkinter import ttk“. Ttk se posbeno mora pozvati neovisno o tome što radi istu funkciju kao Tk. Razlika kod Tk i ttk je i imenovanje nekih parametara, kao na primjer Tk zahtjeva „bg“ za prikaz boje pozadine, dokle ttk zahtjeva „background“.

```
ttk.Label(self.root, text="Ime", padding='5').grid(row=1, column=0)
self.ime_field = ttk.Entry(self.root)
self.ime_field.grid(row=1, column=1, ipadx="100")
self.ime_field.focus()
```

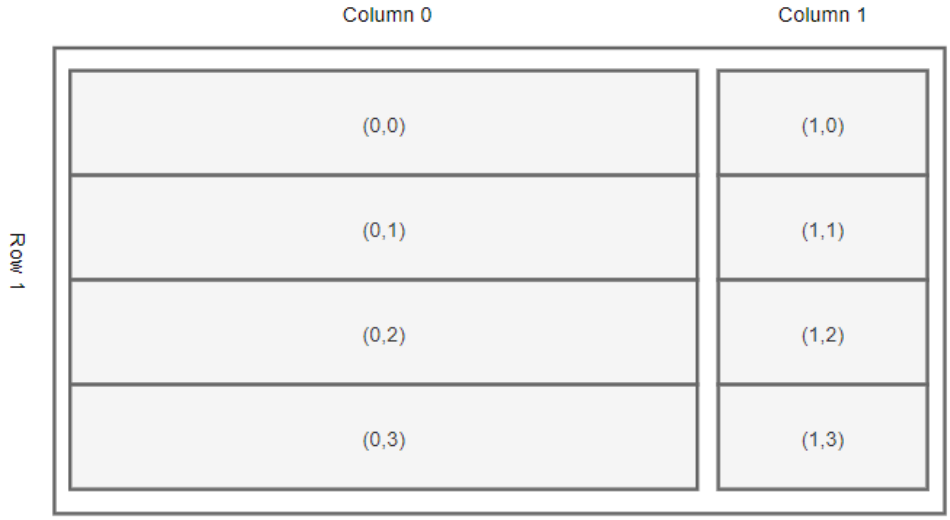
Slika 3. Primjer oznake

Kao što je vidljivo na slici 3, „label“ i „entry“ su dijelovi modula ttk, a unutra zagrada je sama struktura i funkcija polja. Root je instanca klase Tk i svaki novostvoreni prozor mora imati svoj root da može koristiti i prikazati sva potrebna polja. Na kraju samog koda definiramo da je root pozvan od strane Tk također i „app“ definiramo koja pozove glavnu klasu, to jest prozor same aplikacije. Oznaka „label“ služi kao prikaz onoga što je potrebno u trenutku i nije moguće ništa unositi u to polje, ali to ne znači da nema više svrha. Unutar oznake definiramo njen tekst koji je statičan i može također ne samo prikazat zadan tekst nego i može biti pozvana neka od metoda. Na kraju svake oznake mora se istaknuti pozicija koja se vrši s više naredba kao na primjer: grid, place ili pack. Svaka vrsta pozicioniranja ima svoje prednosti i mane dokle u ovom radu koristio sam većinom grid za polja zato što se može poredati kao polja u tablici. Place je korišten za slučajeve gdje je potreban prikaz većih elemenata kao na primjer prikaz tablice s podatcima o gostima. „Self“ se koristi prije imenovanja svake oznake zato što je dio te klase i također ostale metode u samoj klasi mogu koristiti varijable i ostale oznake kako bi zajedničko sve moglo funkcionirati. Uz oznake veliku većinu puta nalazi se i polje za unos koje ima mnogo vrsta, a u ovom radu najviše se koristi polje za ručni unos, da li ti bio unos brojeva ili riječi.

```
from tkinter import *
from tkinter import ttk
root = Tk()
frm = ttk.Frame(root, padding=10)
frm.grid()
ttk.Label(frm, text="Hello World!").grid(column=0, row=0)
ttk.Button(frm, text="Quit", command=root.destroy).grid(column=1, row=0)
root.mainloop()
```

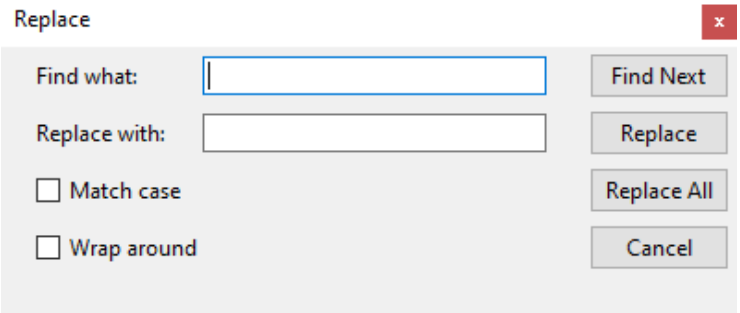
Slika 4. Primjer tkinter biblioteke

Na primjeru iz slike možemo vidjeti kako je u kodu uvedeno sve iz tkinter biblioteke i isto tako ttk. Na slici 5 je za prikaz korišten „frame“ koji se također može koristiti za prikaz prozora. Frame je dobro koristiti s naredbom pack jer ako hoćemo pozicionirati sve na jednu stranu kao na primjer donju stranu, onda se automatski rasporedi jedan ispod drugog kako ne bi bilo sve zajedno i nepregledno.



Slika 5. Primjer grid rasporeda

Na slici 5 se vidi kako je moguće rasporediti elemente na prozoru putem grid pozicioniranja gdje je dobra praksa napraviti da nam oznake budu s lijeve strane, a ulazna polja odmah pored s desne. Za unos podataka iz polja koristimo „button-e“ kojima primaju komande od metoda koje definiramo u kodu. Button može imati ne samo jednu funkciju, nego čak i više, na primjer možemo definirati metodu koja pritiskom na button spremi podatke i usput otvori novi prozor. Na slici 6 je prikazano 4 buttona koji svaki služi nečemu, prva tri buttona čitaju vrijednosti iz polja i onda traže po dokumentu i zadnji poništi i zatvori prozor.



Slika 6. Raspored elemenata putem grid

Još jedne od vrsta koje se koriste za unos su: checkbutton, radiobutton i combobox. Ova tri elementa za unos su najviše u aplikacijama zato što ima bolji izgled unosa s time da su vrijednosti već prije u kodu definirane. Koristeći combobox, moguće je u kodu definirati ručno vrijednosti koje će biti prikazane, ali tako je i moguće povući podatke iz neke liste gdje će biti prikazano kao lista i zapisane kasnije kao samo 1 vrijednost. U tkinter-u kada trebamo pristupiti vrijednostima koje korisnik unese u polja ili odabere, dobijemo putem naredbe „.get()“ koja pročita vrijednost neovisno o vrsti, bilo to riječ ili broj.

Još jedna od mogućnosti ove biblioteke je „filemenu“ koji služi kao izbornik na vrhu prozora aplikacije i putem izbornika možemo napraviti listu s elementima koji služe kao buttoni za razne funkcije u aplikaciji. Može biti povezano s ostalim buttonima ili zasebno da omogući korisniku mnoge razne opcije kako bi aplikacije bila više interaktivna. Zadnja vlastita vrsta prikaza koja spada pod ttk i korištena u radu je „treeview“ čija je funkcija prikaz podataka na prozoru. Podatci koji mogu biti prikazani su razni jer se čak mogu uvesti iz baze podataka i kako će biti prikazani je izbor same osobe koja piše kod. Treeview sadrži listu vrijednosti koje možemo kasnije pretvoriti u željenu vrstu atributa ovisno što program zahtjeva.

Datetime

Sljedeća biblioteka nema mnogo funkcionalnosti kao i ostale jer nam služi najviše za prikazivanje datuma. Ionako nema mnogo funkcija, ova biblioteka je bitna za prikazivanje na primjer današnjeg vremena i datuma ili prikaz datuma koji je zadan kodom. Mnogo puta ne možemo birati ako će se datum koji smo zapisali spremiti kao datum ili string pa onda nam služi klasa datetime koji može riječ ili „string“ pretvoriti natrag u datum. Naredba za promijeniti ne samo prikaz datuma, nego već i kako će se taj datum koristiti u kodu je „strftime“ koja pretvara zadan datum u onaj koji je nama potreban. Jedan o primjera prikazan je na slici 7 gdje atribut „now“ pokazuje trenutno vrijeme i datum nakon čega putem „strftime“ pretvorimo u oblik koji će biti potreban za rad. Moguće je posebno izdvojiti dan, mjesec i godinu, ali za ovaj

rad u kasnijem primjeru vidjet ćemo korištenje svega zajedno di se pretvori u cjelokupan datum. Još jedna od mogućnosti datetime je naredba „strptime“ koje je vrlo bitna ako uzimamo vrijednost gdje nije moguće zapisati datum nego već string. Za funkcionalnost ovoga rada „strptime“ naredba je bila korištena mnogo puta za izvoza datum iz baze podataka. Putem datetime moguće je međusodno datume računati, to jest zbrajati ili oduzimati na način ako nam je potrebno dobiti razliku dana između dva datuma onda jednostavno oduzemo datume i dobijemo rezultat. Rezultat kada oduzimamo datume dobijemo ispisan u danima koji nakon toga pretvorimo u broj ili integer kako bi kasnije mogli to koristiti u daljnje svrhe.

```
from datetime import datetime

now = datetime.now() # current date and time

year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)

date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:",date_time)
```

Slika 7. Korištenje datetime naredbe

SQLite

Možda druga najbitnija biblioteka korištena je SQLite koja je namijenjena uvozu i izvozu podataka iz baze podataka. SQLite ima mogućnost pristupa bazi bez prijave u sustav što je bilo efikasnije za ovaj rad jer nije potrebno zaštiti bazu od ostalih prijetnji ili slično. Kreiranje baze podataka moguće je napraviti u samom Python kodu, ali u ovom slučaju to je bilo napravljeno korištenjem programa DB Browser da bude i kasnije lakše pregledati bazu ili urediti. Na slici 8 je prikazano kako se koristi SQLite za navesti koja se baza podataka koristi u kodu s time da se na slici 8 vidi da je veza do baze referencirana putem metode. Za izradu ove aplikacije napravljeno je da pozicija baze i kako će se baza uvesti na način da u atributu spremimo vrijednost lokacije i ime baze podataka. Povezivanje na bazu radimo putem „sqlite3.connect()“ gdje među zagrade se nalazi referenca na bazu. Dobra praksa kad imamo veći kod je da metodom povežemo bazu podataka kako bi mogli onda tu metodu koristiti kad

god bez ponovnog povezivanja i na kraju kad se poveže, također se treba tu vezu i zatvoriti putem naredbe „close()“.

```
import sqlite3
from sqlite3 import Error


def create_connection(db_file):
    """ create a database connection to a SQLite database """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        print(sqlite3.version)
    except Error as e:
        print(e)
    finally:
        if conn:
            conn.close()


if __name__ == '__main__':
    create_connection(r"C:\sqlite\db\pythonsqlite.db")
```

Slika 8. SQLite3 biblioteka

Stvaranje tablica u bazi možemo učiniti kodom putem SQL naredbi: „CREATE TABLE“ stvori tablicu pod vlastitim imenom nakon čega slijedi imenovanje stupaca gdje će podatci biti spremljeni. Za spremanje vrijednosti u bazu koristimo „INSERT INTO“ koje je najefikasnije smjestiti u jednu od metoda koju onda pozivamo kad god bi bilo potrebno. Za spremanje podataka i ostale naredbe koristimo većinom na button-ima tako da kad god nam je potrebno iz polja spremiti podatke samo pritiskom na button to učinimo. Korisnost baza je velika kao na primjer možemo imati jednu tablicu s podatcima koja u samoj aplikaciji može poprimiti mnogo funkcija, na primjer za ovu aplikaciju koristimo jednu tablicu za uvoz podataka o gostima, ažuriranje informacija o njima i na kraju čak i za brisanje. Mana koju sam primijetio kod SQLite je što ne podržava datum kao tip podatka pa kada hoćemo spremiti datum moramo učiniti kao da je to tekst ili broj umjesto pravi datum. Naredbu „SELECT“ za odabir pojedinih podataka pozivamo svaki put kad nam je potrebno izvesti podatke, ali ne mora to biti prikaz isključivo, nego i taj odabir možemo upotrijebiti za daljnju uporabu za unos tih podataka u aplikaciju.

Reportlab

Zadnja biblioteka je „reportlab“ zaslužna za prikaz podataka u pdf obliku koji se sprema lokalno na računalo. Reportlab je samo jedna od mnogih biblioteka za prikaz podataka, ali odabrao sam ovu zbog toga što ima mnogo primjera kako rasporediti podatke. Mogućnosti izrade su raznovrsne od običnog rasporeda kako će određeni podatci izgledat na stranici pa sve do grafičkog prikaza podataka ili neke statistike.

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

canvas = canvas.Canvas("form.pdf", pagesize=letter)
canvas.setLineWidth(.3)
canvas.setFont('Helvetica', 12)

canvas.drawString(30,750,'OFFICIAL COMMUNIQUE')
canvas.drawString(30,735,'OF ACME INDUSTRIES')
canvas.drawString(500,750,"12/12/2010")
canvas.line(480,747,580,747)

canvas.drawString(275,725,'AMOUNT OWED:')
canvas.drawString(500,725,"$1,000.00")
canvas.line(378,723,580,723)

canvas.drawString(30,703,'RECEIVED BY:')
canvas.line(120,700,580,700)
canvas.drawString(120,703,"JOHN DOE")

canvas.save()
```

Slika 9Primjer canvas za reportlab

Na slici 9 možemo vidjet kako se implementira biblioteka i primjer kako se koristi, ali to je samo jedan od primjer na koji se način to može izvesti. Primjer na slici 9 prikazuje korištenje reportlab-a s uvođenjem atributa canvas koji se koristi za izradu pdf dokumenta. Canvas definira koji naziv će dokument imati nakon što se spremi i na slici 9 je vidljivo da putem naredbe „pagesize“ definira se veličina samog dokumenta. Veličina dokumenta može biti svakakva kao na primjer letter, A3 i A4 koji su samo jedni od mnogih. Ovaj primjer radi na način da se može definirati gdje će se što nalaziti na dokumentu. Naredba „drawString“ pomoću x i y koordinata ispiše tekst koji je zadan od strane korisnika i taj tekst ne mora biti odmah zadan nego čak može biti referenca na nešto što se nalazi u kodu ili iz datoteke. Sljedeća naredba je „line“ koja je namijenjena za iscrtavanje linija za bolju preglednost i eventualno za postoji nekakav red gdje će se što nalaziti na x i y osi dokumenta. Dodatna mogućnost je prikazivanje loga ili nekakve slike ovisno o potrebi i vrsti dokumenta. Za prikaz podataka ja sam koristio

tablice zato što je namjena dokumenta da bude račun koji se izdaje gostu dolaskom iz objekta. Namjena tablica može biti svakakva i u moje mišljenje za ovaj rad je bolje zato što se ne mora definirat pozicija putem x i y, nego znamo točno koliko nam je potrebno redaka i stupaca za svaki račun.

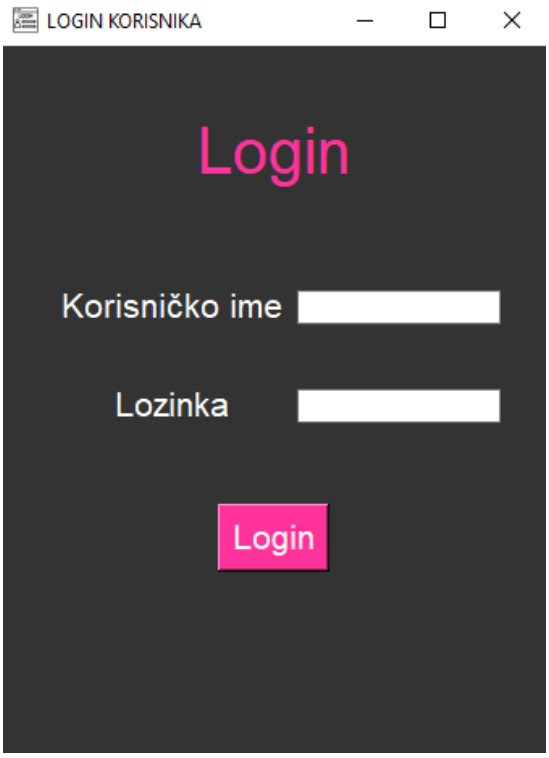
To su biblioteke korištene za izradu ove aplikacije od kojih sam koristio samo određene funkcije koje one mogu zapravo služiti.

Implementacija aplikacije

Ova aplikacija je pisana u programskom jeziku Python koji je jedan od mnogih sposoban za izradu objektno orijentirane aplikacije. Cijeli kod je napisan koristeći pycharm IDE ne zato što ima posebne funkcije, već zbog osobne preference. Jedini dio za koji se pycharm nije koristio je pri izradi baze podataka i tablica zato što je jednostavnije i efikasnije izraditi putem programa „DB manager“. Aplikacija koristi dva različita glavna prozora, prvi dočeka korisnika za se prijaviti putem svog korisničkog imena i lozinke, a drugi glavni prozor se pojavi čim se korisnik prijavi i može započet svoj rad gdje prijavljuje goste u objekt. U slučaju da je potrebno dodavanje novih korisnika to je kasnije moguće napraviti s kreiranjem dodatne tablice u bazi za koju će nadređeni biti odgovoran i prijaviti novog djelatnika.

Login prozor

Prijavu u aplikaciju imaju dvije osobe gdje običan zaposlenik ima funkcije potrebne njemu za svakodnevni rad, dokle njemu nadređeni ima dodatnu funkciju gdje može mijenjati cijene soba u objektu. Ova opcija dodana je zato što cijene su ovisne o godišnjim dobima ili nekim posebnim prigodama pa zato su sve cijene zapisane u bazu podataka gdje se može pristupiti i promijeniti kad god. Spremanjem podataka u bazu omogućuje da ova aplikacija ne bude namijenjena za samo jedan objekt nego već i više pošto nije definirano da mora biti određen broj soba ili kategorija.



Slika 10. Izgled login prozora

Na slici 10 je prikazan početan prozor za prijavu korisnika na kojem se nalaze dvije oznake s korisničkim imenom i lozinkom, dva ulazna polja za oznake i na kraju je dugme „Login“ s kojim se korisnik prijavi u sustav i otvara se novi prozor. Sama klasa zove se autentifikacija i sadrži metode potrebne za prikaz prozora i provjerom koji je korisnik prevaljen. Prije nego što se metode otvaraju imamo zadano ime i lozinku dva korisnika koje koristimo kod provjere za prijavu što je vidljivo na slici 11.

```
7   class Autentifikacija:
8       user = 'user'
9       passw = 'user'
10
11      managerUser = 'manager'
12      managerPass = 'manager'
```

Slika 11. Metoda Autentifikacija

Na samom početku koda definiramo naslov koji ćemo prikazati na vrhu prozora, veličinu koja će biti kad se prozor otvori sa pozicijom gdje će se prikazati čim pokrenemo aplikaciju. S naredbom „geometry()“ odredimo širinu i visinu s time da se može nakon toga definirati gdje će prozor se prikazati na ekranu koristeći x i y koordinate. Naredbom

„iconphoto“ dodao sam malu ikonu za prozor što nije potrebno ali bila je još jedna od opcija koje sam htio proučiti i prikazati. Za izgled prozor početnog jedino što nam je ostalo je boja pozadine koju smo stavili tamno sivu sa šifrom „#333333“.

Prikazivanje svih elemenata obavljamo korištenjem frame i odabrao sam ovakav prikaz zato što ovaj prozor nije velike veličine i nema puno elemenata pa za samo istraživanje sam odabrao ovakav prikaz. Oznake imaju iste boje pozadine kao i prozor zato što nisam htio da se istaknu drugačije od ostalih elemenata nego samo da se vidi tekst dokle za ulazni element potrebno je samo definirati kakve vrste će biti, u ovom slučaju običan „entry box“ i gdje ćemo ga smjestiti. Za pozicije na prozoru koristio sam „grid“ za sve jer nema mnogo elemenata pa je moguće lagano definirati s redovima i stupcima gdje će se što nalaziti i za dugme stavio sam „columnspan=2“ što znači da spaja 2 stupca i s „pady=30“ povećali malo oblik.

```
def __init__(self, root):
    self.root = root
    photo = PhotoImage(file="login.png")
    self.root.iconphoto(False, photo)
    self.root.title('LOGIN KORISNIKA')
    self.root.configure(bg='#333333')
    self.root.geometry("340x440+1300+200")
    frame = tkinter.Frame(bg='#333333')

    '''Username i Password'''
    login_label = tkinter.Label(
        frame, text="Login", bg='#333333', fg="#FF3399", font=("Arial", 30))
    login_label.grid(row=0, column=0, columnspan=2, sticky="news", pady=40)

    Label(frame, text=' Korisničko ime ', bg='#333333', fg="#FFFFFF",
          font=("Arial", 16)).grid(row=1, column=0)
    self.username = ttk.Entry(frame)
    self.username.grid(row=1, column=1, pady=20)

    Label(frame, text=' Lozinka ', bg='#333333', fg="#FFFFFF",
          font=("Arial", 16)).grid(row=2, column=0)
    self.password = ttk.Entry(frame, show='*')
    self.password.grid(row=2, column=1, pady=20)

    # Button
    login_button = tkinter.Button(frame, text="Login", bg="#FF3399",
                    fg="#FFFFFF", font=("Arial", 16), command=self.login_user)
    login_button.grid(row=3, column=0, columnspan=2, pady=30)
    frame.pack()
```

Slika 12. Kod za login prozor

Za prozor prijave korisnika jedino što nam ostaje je provjera koji je korisnik pristupio prozoru nakon čega slijedi povezivanje na glavnu aplikaciju ako je prijava bila uspješna. Provjera se radi na način samo jednostavno povučemo s polja za unos vrijednosti imena i lozinke i ako odgovaraju prije navedenim vrijednostima. Imamo dvije provjere, jedna je ako je običan korisnik, dokle druga ako je nadređena osoba. Ako je provjera bila uspješna pojavi se mali prozor s porukom o prijavi nakon čega slijedi novi prozor. Prije otvaranja prozora trenutni prozor uništimo s naredbom „destroy()“ i otvara se klasa novog prozora s deklariranim novim root.

```
def login_user(self):

    '''Provjera točnosti unesenih podataka'''
    if self.username.get() == self.user and self.password.get() == self.passw:
        messagebox.showinfo(title="Uspješan login", message="Ulogirani ste u aplikaciju.")
        # Uništavanje ovog prozora ali popravit da se samo sakrije
        root.destroy()

        # Otvori novi prozor
        newroot = Tk()
        application = StartPage(newroot, self.user)
        newroot.mainloop()

    elif self.username.get() == self.managerUser and self.password.get() == self.managerPass:
        messagebox.showinfo(title="Uspješan login", message="Ulogirani ste u aplikaciju.")
        # Uništavanje ovog prozora ali popravit da se samo sakrije
        root.destroy()

        # Otvori novi prozor
        newroot = Tk()
        application = StartPage(newroot, self.managerUser)
        newroot.mainloop()
```

Slika 13. Provjera korisnika

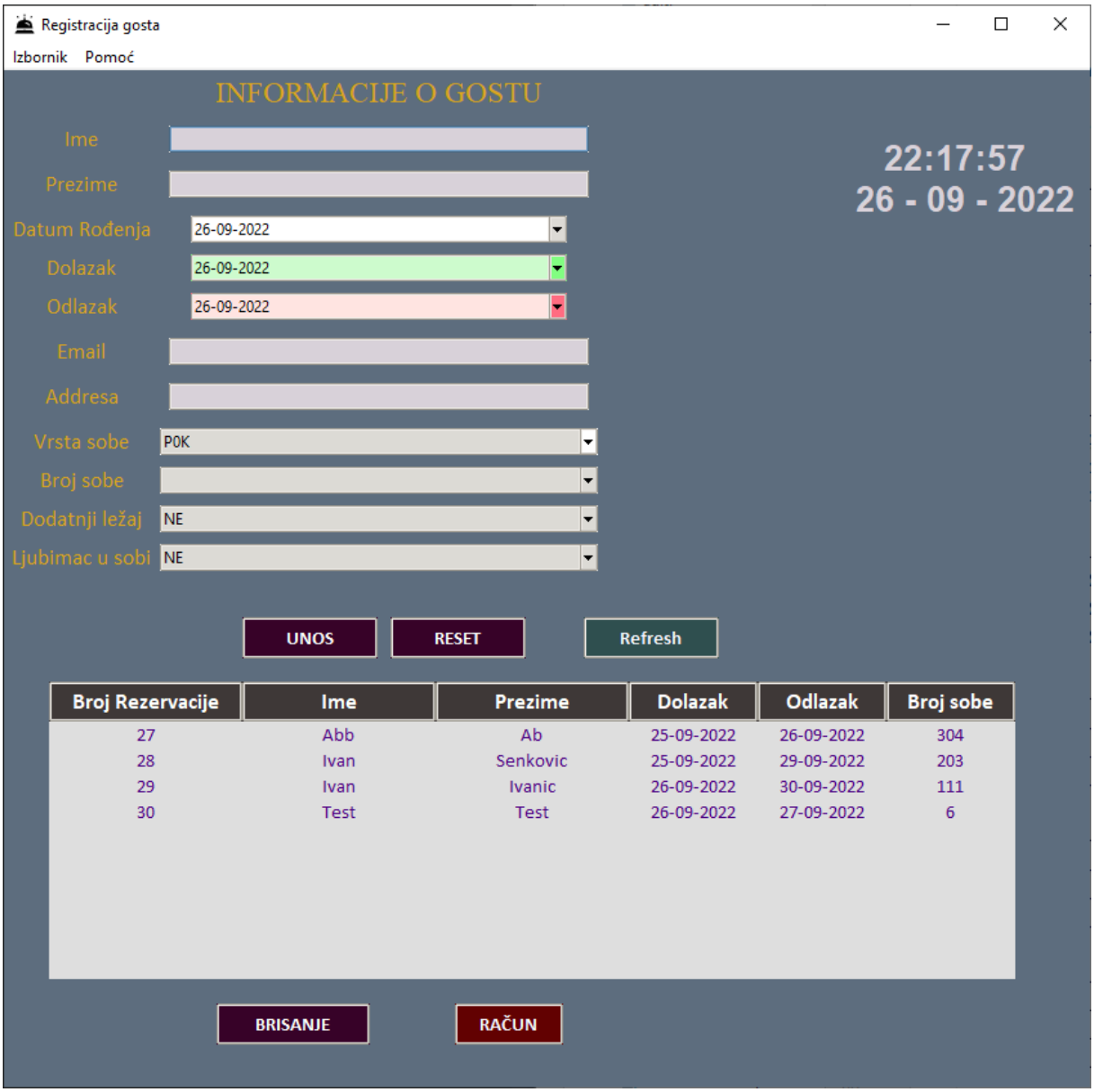
Kao dodatnu funkciju dodao sam da program vraća poruke ovisno o tome gdje je korisnik pogriješio s prijavom. Ako su oboje korisničko ime i lozinka neispravni dobije poruku da je oboje pogrešno, ako je pogrešno samo ime izbaci se obična poruka, a ako je samo lozinka kriva ne samo da se izbaci poruka, nego se i polje obriše da bude lakše za ponovni unos.

```
else:
    if self.username.get() != self.user and self.password.get() != self.passw:
        messagebox.showerror(title="Greška",
                             message="Neuspješan login. Molimo pokušajte ponovno!")

    elif self.password.get() != self.passw:
        messagebox.showerror(title="Greška",
                             message="Pogrešna lozinka. Molimo pokušajte ponovno!")
        self.password.delete(0, END)

    elif self.username.get() != self.user:
        messagebox.showerror(title="Greška",
                             message="Pogrešno korisničko ime. Molimo pokušajte ponovno!")
```

Slika 14. Povratna poruka kod prijave

Glavni prozor aplikacije

Nakon uspješne prijave korisnika dolazimo do glavnog prozora gdje korisnik ima mogućnost prijave gost u određeni objekt. Na slici 15 je prikazan glavni prozor na kojem se nalaze oznake kao i na ekranu za prijavu, polja za ulazne vrijednosti koje se spremaju u bazu, gumbovi za koji svaki ima svoju metoda koja radi posebne operacija. Ono što je dodatno je „filemenu“ koji služi za nekoliko funkcija aplikacije i klikom na izbornik dobijemo listu funkcija. Prva funkcija novo služi samo za brisanje vrijednosti polja ako je korisniku potrebno unijeti nove vrijednosti, druga opcija je trenutno vidljiva zato što je prijavljen nadređeni i samo on ima ovu funkciju koja otvara novi prozor koji služi za promjenu cijena soba u objektu, funkcija „ažuriraj“ je za ažuriranje informacija o gostu koji je trenutno u objektu i zadnja „Izlaz“ je samo za izlaz iz aplikacije. Dugme pomoć moguće je napraviti da služi na mnogo načina, ali za ovaj rad učinio sam da otvara mali prozor pritiskom na funkciju „o programu“ koji ispiše tko je napravio aplikaciju.



Slika 15. Prikaz glavnog prozora aplikacije

Prikaz informacija

Ono što je ostalo na prozoru su funkcije koje prikazuju nešto, a prva prikazuje trenutno vrijeme koje se uživo mijenja i druga je tablica ili „treeview“ koje uzima podatke putem metode iz baze podataka i prikazuje na prozor u tabličnom obliku.

```
def tick():
    datum = datetime.datetime.now()
    today = '{:%d - %m - %Y}'.format(datum)

    mytime = time.strftime('%H:%M:%S')
    self.lblInfo.config(text=(mytime + ' \n ' + today))
    self.lblInfo.after(200, tick)

self.lblInfo = Label(font=('arial', 20, 'bold'), fg='#D8D1D9', bg="#5D6D7E")
self.lblInfo.place(x=650, y=50)
tick()
```

Slika 16. Metoda za prikaz sata

Metodom „tick“ definiramo da koristimo datum i vrijeme koje pokazuje na današnji dan i vrijeme koje se konstantno mijenja. Datum pomoću „format“ stavio sam na ima prikaz kakav po izboru izgleda dobro na prozoru i također vrijeme isto koje pokazuje sate, minute i na kraju sekunde. Zadnje što je potrebno je definirati gdje će vrijeme i datum biti prikazani i pomoću oznake odredimo izgled fonta, veličinu, boja slova, pozadine i na kraju poziciju na prozoru. Jedino što nam je preostalo ju pozvati na način da samo u kodu napišemo ime metode.

Sljedeće i zadnje za prikaz je tablica s informacijama o gostima. U tablicu možemo prikazati koje god podatke je potrebno, ali ja sam odlučio da bude prikazan broj rezervacije, ime i prezime, dolazak, odlazak i broj sobe jer smatram da je su to najbitniji podatci koje trebaju zaposleniku za brzi pregled. Za prikaz tablice potrebno je putem metode definirati gdje će se prikazati, u ovom slučaju treeview i također podatke koje uzimamo iz baze. Metoda „prikaz_podataka“ služi za definiranje koji će se točno podatci prikazati i iz koje tablice u bazi podataka uzima vrijednosti. Na početku metode nalazi se „for petlja“ s kojom se svi podatci u stablu brišu kako bi sa svakim pozivom metode mogli nove podatke prikazati da ne bi došlo do greške u ispisu s viškom informacija. Nakon brisanja prijašnjih podataka slijedi definiranje koje podatke koristimo iz baze podataka. Poziva se metoda za korištenje baze podataka koja je već definirana i kažemo da nam trebaju svi podatci iz tablice „infoGosti“, ali naravno nećemo sve koristiti. Sve podatke spremimo u varijablu „pod_baza“ iz koje u petlji uzimamo red po red i privremeno se sprema u varijablu „data“. Zadnji korak koji je ostao potrebno je odrediti koje podatke treba smjestiti u tablicu stabla, a polja koja su potrebna znamo po mjestima u tablici baze po redu kako su spremljena pa na primjer prva 3 polja su broj rezervacije, ime i przime. Naredbom „insert“ se unose vrijednosti u stablo, prvo polje za unos vrste kako će se prikazati podatci a to je „string“, pomoću END odredimo gdje će se smjestiti podatci, u ovom slučaju na kraj tablice i moguće je zapisati koji god broj hoćemo ovisno o potrebnom mjestu. Kad smo

odredili smještaj podataka slijedi još samo da se prikaže koji tekst će se prikazivati i koje podatci, tekst uzimamo od stupaca iz baze i red na samo određenim pozicijama.

```
def prikaz_podataka(self):
    elem = self.tree.get_children()
    for n in elem:
        self.tree.delete(n)
    query = 'SELECT * FROM infoGosti'
    pod_baza = self.baza_pokreni(query)
    for data in pod_baza:
        self.tree.insert('', END, text=data[0], values=(data[0],
                                data[1], data[2], data[4], data[5], data[9]))
```

Slika 17. Metoda za prikaz podataka

Zadnje je preostalo samo definirati stablo pomoću „treeview“ gdje odredimo da je produžen oblik stabla, broj stupaca koji će trebati za podatke i opciju da budu ispisani nazivi stupaca. Red po red definira se tekst stupca i na kojoj poziciji u stablu će se prikazati putem brojeva i da će biti centrirano. Nakon definiranja redova i stupaca još je potrebno smjestiti stablo na prozor naredbom place.

```
self.tree = ttk.Treeview(self.root, selectmode="extended",
                         column=['', '', '', '', '', ''], show='headings')
self.tree.column("#1", anchor=CENTER, stretch=NO, width=150)
self.tree.heading("#1", text="Broj Rezervacije")
self.tree.column("#2", anchor=CENTER, width=150)
self.tree.heading("#2", text="Ime")
self.tree.column("#3", anchor=CENTER, width=150)
self.tree.heading("#3", text="Prezime")
self.tree.column("#4", anchor=CENTER, width=100)
self.tree.heading("#4", text="Dolazak")
self.tree.column("#5", anchor=CENTER, width=100)
self.tree.heading("#5", text="Odlazak")
self.tree.column("#6", anchor=CENTER, width=100)
self.tree.heading("#6", text="Broj sobe")

self.tree.place(x=35, y=475)

self.prikaz_podataka()
```

Slika 18. Kod za izgled stabla

Unos podataka

Metoda za unos podataka u bazu je moguće najbitnija pošto je tema rada evidencija podataka koje unosimo ovom metodom. Sam unos podataka unosi se pritiskom na gumb „UNOS“ koji se nalazi odmah ispod polja za unos informacija o gostima. Metodu sam u kodu

nazvao „dodaj_podatke“ koja uzima dodatne tri vrijednosti a to su: kategorija sobe, dolazak i odlazak gosta. Tri dodatne vrijednosti nisu potrebne, ali sam ih dodao tako kasnije bude lakše za unos cijene koja se automatski izračuna. Prije nego što krene postupak unosa podataka, prvo je provjera ako su vrijednosti upisane u svakom polju jer ako nije onda nam iskoči prozor s upozorenjem da moraju sva polja biti popunjena. Nakon provjere metodom „provjera“, slijedi glavni dio koda gdje odmah na početku odredimo što je potrebno da se unese u bazu. Prvo polje je „NULL“ zato što je to primarni ključ koji je jedinstven i brojevi rezervacije sami se upisuju redoslijedom, a ostale vrijednosti su pod upitnikom „?“ jer nemamo definirane konstantne vrijednosti koje uvrštavamo, nego već ovise o poljima za unos koje odredi korisnik. Putem atributa „parameters“ zapišemo parametre koji su nam podatci potrebni i pošto moramo unijeti nove podatke uzimamo naredbom „.get()“, koja u tkinteru prepoznaje unos podataka, uzmemo sva polja u koja je korisnik nešto napisao. Broj parametra mora biti jednak broju vrijednosti koji je zadan kad smo definirali da unosimo neke podatke u bazu jer ako nije jednak broj dobit ćemo error koji govori koliko zahtjeva vrijednosti. Poslije definiranja parametra još je preostalo pozvati ponovno metodu koja unosi podatke u bazu i sprema red pod red. Što se tiče dodavanja podataka to bi bilo dovoljno, ali sam dodao par funkcija i prva je da ime i prezime se unose s velikim početnim slovom, poruka na glavnom prozoru koja ispiše koji je gost dodan i na kraju čišćenje ili reset polja tako da sva polja budu čista za nosi unos podataka. Na kraju metode još pozovemo metodu za ispis novih podataka u stablo koje smo prije definirali.

```python
def provjera(self):
    return len(self.ime_field.get()) != 0 and len(self.prezime_field.get()) != 0 and len(
        self.datRod_field.get()) != 0 and len(self.email_field.get()) != 0 and \
           len(self.kat_field.get()) != 0 and len(self.dol_field.get()) != 0 and \
           len(self.odl_field.get()) != 0 and len(self.adresa_field.get()) != 0 and\
            len(self.soba_field.get()) != 0 and len(self.lezaj_field.get()) != 0 and \
           len(self.pet_field.get()) != 0

def dodaj_podatke(self, kat_fld, dol_dat, odl_dat):
    dat_danas = datetime.date.today()
    if self.provjera():
        # provjera da dolazak mora biti danas i odlazak vise od danas
        if ((dat_danas == self.dol_field.get_date()) and (dat_danas < self.odl_field.get_date())):
            query = 'INSERT INTO infoGosti VALUES (NULL,?,?,?,?,?,?,?,?,?,?,?)'
            parameters = (
                self.ime_field.get().capitalize(), self.prezime_field.get().capitalize(),
                self.datRod_field.get(), self.dol_field.get(), self.odl_field.get(),
                self.email_field.get(), self.adresa_field.get().capitalize(), self.kat_field.get(),
                str(self.soba_field.get()), self.cijena_ukupno(kat_fld, dol_dat, odl_dat),
                self.lezaj_field.get(), self.pet_field.get())
            self.baza_pokreni(query, parameters)
            self.message['text'] = 'Gost {} {} dodan!'.format(self.ime_field.get(),
                                                              self.prezime_field.get())
            self.root.after(2000, lambda: self.message.grid_forget())   # BRISANJE PORUKE NAKON 2 SEC
            self.message.grid(row=12, column=1)
            '''Brisanje vrijednosti polja nakon unosa'''
            self.ime_field.delete(0, END)
            self.prezime_field.delete(0, END)
            self.datRod_field.delete(0, END)
            self.dol_field.delete(0, END)
            self.odl_field.delete(0, END)
            self.email_field.delete(0, END)
            self.adresa_field.delete(0, END)
            self.soba_field.set('')  # NAKON UNOSA COMBOBOX POSTANE PRAZAN TAKO DA VISE NE PISE BROJ SOBE
            self.lezaj_field.set('')
            self.pet_field.set('')

        else:
            messagebox.showinfo("Pogrešan unos.", "Dolazak mora biti današnji dan!")

    else:
        messagebox.showinfo("Pogrešan unos.", "Niste unijeli vrijednosti u sva polja!")

    self.prikaz_podataka()
```

Slika 19. Metoda za dodavanje podataka

Brisanje podataka

Sljedeća metoda koja slijedi je brisanje podataka iz baze, ali također i brisanje iz stabla glavnog prozora. Kad je potrebno neki podatak izbrisati iz baze i iz stabla onda je potrebno prvo odabrati traženi red ili rezervaciju, moguće je i upisom vrijednosti u polje, ali ovaj način sam stavio jer je više interaktivan, onda odabirom reda u stablu brišemo podatke. Metodu sam nazvao „brisanje podataka“ koja se poziva odmah nakon klika na gumb „BRISANJE“. Kao i kod dodavanja podataka tako i ovdje postoji prvo vrsta provjere a to je provjera ako je korisnik

prije brisanja odabrao red u stablu i ako je odabrao kod se nastavlja, a ako nije onda metoda vraća poruku da se odabere polje potrebno za brisanje. Nakon odabira polja odredimo da broj rezervacije će biti će pokazati koju red se mora obrisati, a taj broj rezervacije spremamo u atribut kojeg ćemo koristit kao referencu. Za brisanje podataka iz baze potrebno je definirati koji stupac će tražiti vrijednosti, a u ovom slučaju to je „br_rez“ nakon čaga briše se cijeli redak s odabranim brojem. Još je samo potrebno da metoda ispiše poruku o obrisanoj rezervaciji i nakon čega je potrebno pozvati metodu za prikazati kako izgleda konačna baza podataka u tabličnom obliku.

```
def brisanje_podataka(self):
    # brisanje prijasnjeg teksta
    self.message['text'] = ''

    try:
        self.tree.item(self.tree.selection())['values'][1]

    except IndexError as e:
        self.message['text'] = 'Molim odaberite polje za izbrisati!'
        return

    self.message['text'] = ''

    brRez = self.tree.item(self.tree.selection())['text']
    query = 'DELETE FROM infoGosti WHERE br_rez = ?'
    # zarez tkao da cita rezeracije s više znamenaka
    self.baza_pokreni(query, (brRez,))
    # brisanje poruke nakon 2 sekunde
    self.message['text'] = 'Rezervacija broj {} je izbrisana!'.format(brRez)
    self.root.after(2000, lambda: self.message.grid_forget())
    self.message.grid(row=12, column=1)

    self.prikaz_podataka()
```

Slika 20. Metoda za brisanje podataka

Za brisanje podataka izradio sam još jednu metodu „brisanje_datuma“ koja prilikom pritiska na gumb „osvježi“ briše iz baze sve rezervacije čiji je odlazak na današnji dan poslije 12 sati. Definirano je današnji datum putem naredbe „datetime.date.today()“ koja se pretvara u potreban format za uspoređivanje drugog datuma iz baze. Nakon provjere putem „if“ petlje ako je prošlo 12 sati pokreče se naredba za brisanje polja s isteklim datumom odlaska.

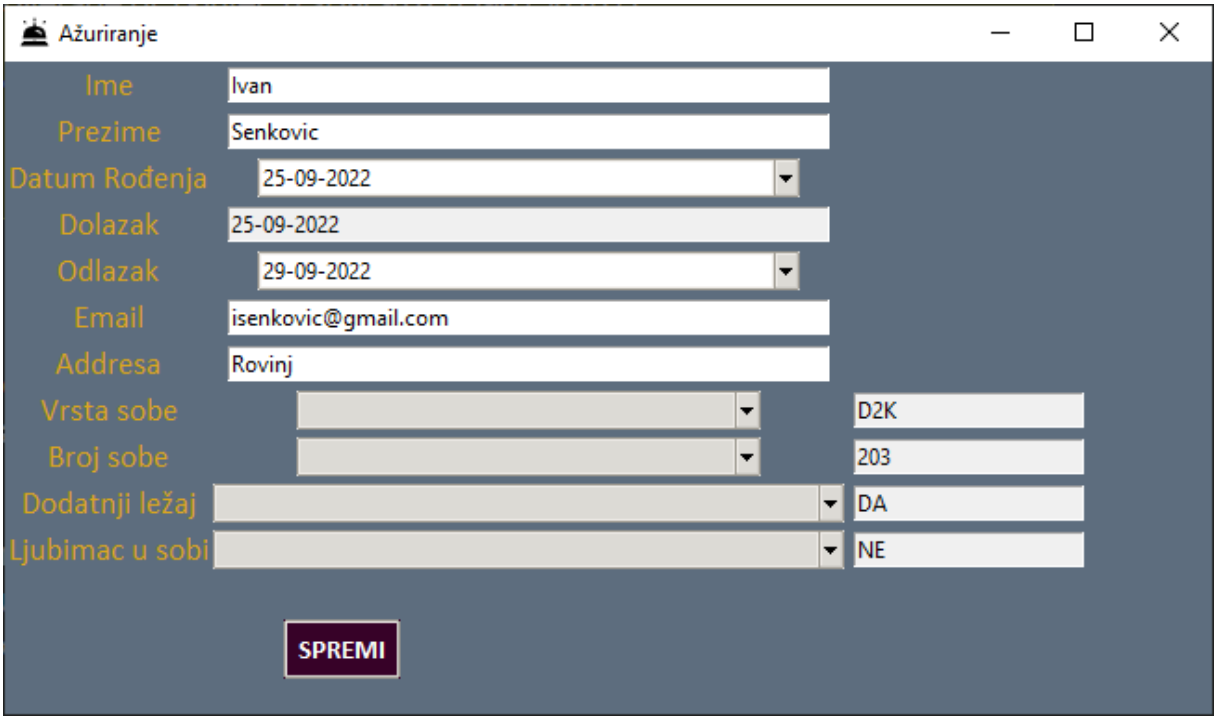
```python
def birsanjeDatuma(self):
    # brise sve rezervacije koje su na danasnji dan ili prije
    dat_danas = datetime.date.today()   # danasnji datum
    # danasnji datum plus 1 dan da je lakse za brisat iz baze
    dns_dat = dat_danas + datetime.timedelta(days=1)
    dat_danas1 = dns_dat.strftime("%d-%M-%Y")
    vrijeme_dns = datetime.datetime.now()

    conn = sqlite3.connect("rezervacije.db")
    cursor = conn.cursor()
    # brisanje rezervacija prije 12h
    if (vrijeme_dns.hour > 12):
        query = "DELETE FROM infoGosti WHERE odl_fld < ?"
        cursor.execute(query, (dat_danas1,))
        conn.commit()
    else:
        pass

    self.prikaz_podataka()
```

Slika 21. Metoda za brisanje datuma

Ažuriranje podataka o gostu

Ažuriranje podataka se poziva klikom na gumb koji se nalazi u meni izborniku na vrhu prozora i za pokretanje prozora za ažuriranje potrebno je prvo odabrati red iz stabla kao što se treba i za brisanje. Kako bi mogli ažurirati već postojeće podatke potrebno je napraviti novi prozor s podatcima koji su zapisani u bazi. Metoda „azu_glavna“ započinje proces ažuriranja, ali prvo kao i kod brisanja je provjera ako je korisnik odabrao red iz stabla i nakon provjere slijedi glavni dio koda. Ovim pristupom naučio sam kako se mogu podatci iz stabla čitati i u isto vrijeme uzeti vrijednosti iz baze odnosno na vrijednost iz stabla. Prvo se mora ponovno odrediti veličina i naslov prozora kao i za glavni prozor i nakon toga uzmemo vrijednost prvog stupca koji odgovara broju rezervacije u bazi i na taj način se ispišu sve trenutne vrijednosti polja. Raspored polja za unos i oznaka jednak je onom na glavnom prozoru, ali jedino što je drugačije je dodatak polja koja su samo za pregled prijašnjih vrijednosti za usporedbu. Vrijednosti se vuku iz baze na isti način kao i s metodom za prikaz vrijednosti i u ovom slučaju to su sva polja iz baze čak i cijena. Kako bi se vrijednosti ažurirale, to jest spremile u bazu natrag, definiran je gumb „SPREMI“ koji zove novu metodu koja služi pokretanju baze podataka i davanju naredbe da se podatci ovaj put „UPDATE-aju“ ili ažuriraju. U novoj „azu_podatke“ metodi potrebno je navesti sva polja koja ažuriramo nakon čega slijedi ponovno pokazivanje da se podatci spremaju prema broju rezervacije gosta. Osim polja za unos bitna

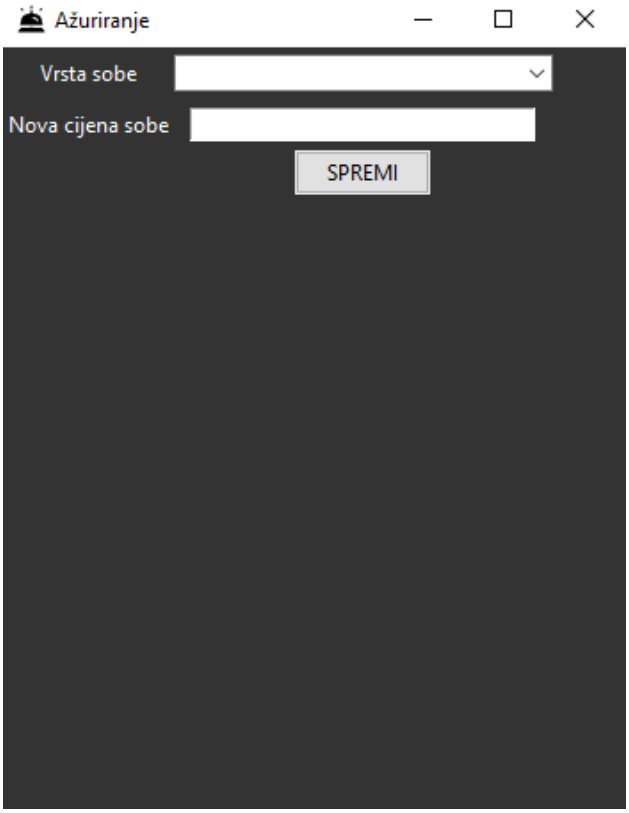
stvar je da se mora pozvati metoda koja je zadužena za računati cijenu noćenja ovisno o danima koji se oduzmu između dva datuma. Uspješnim ažuriranjem korisnik je obaviješten porukom nakon čega se ovaj prozor zatvara i vraća na glavni prozor.



Slika 22. Prozora za ažuriranje podataka

Izmjena cijena

Dodatna opcija za izmjenu cijena je dodana zbog toga što sam htio da aplikacija je mora biti vezana za samo jedan određeni objekt i da se cijene soba mogu po potrebi mijenjati. Kao što sam naveo prije ovaj gumb je moguć jedino u slučaju da je prijavljen nadređeni i nalazi se u gornjem izborniku. Za ovaj prozor potrebno je samo kliknuti na gumb i otvara se novi mali prozor koji uzima samo vrijednosti o kojoj se kategoriji sobe radi i polja za unos nove cijene. Metoda „azuCijena“ prima vrijednosti „cijena_fld“ i „kat_sobe“ koje služe kako bi odredili novu cijenu i odredili za koju kategoriju sobe se odnosi. Novu tablicu u bazi sam izradio tako da uvije postoji mogućnost za kasnije dodati metodu koja će imati mogućnosti dodavati sobe i kategoriju ovisno o potrebama pojedinog objekta. Metoda je jednostavna i samo moramo kao i prije reći za koji stupac i red u tablici ćemo promijeniti. Ova metoda povezuje se na drugu metoda koje je napravljena kao prozor za ažuriranje cijena i definirana je na istom principu kao i za ažuriranje informacija o gostima.



Slika 23. Prozor za ažuriranje cijena

```python
def azuCijena(self):
    self.edit_root = Toplevel()
    self.edit_root.title('Ažuriranje')
    self.edit_root.geometry('350x450+1300+200')
    photo = PhotoImage(file="image.png")
    self.edit_root.iconphoto(False, photo)
    self.edit_root.configure(bg='#333333')
    # global kat_sobe

    kat = []
    query1 = 'SELECT kategorija FROM cijenikSoba'
    db_table = self.baza_pokreni(query1)
    for n in db_table:
        kat.append(n)
    Label(self.edit_root, text="Vrsta sobe", bg='#333333',
          fg="#FFFFFF", pady=5).grid(row=1, column=0)
    kat_sobe = ttk.Combobox(self.edit_root, state="readonly")
    kat_sobe['values'] = (kat)
    kat_sobe.grid(row=1, column=1, ipadx="35")

    Label(self.edit_root, text="Nova cijena sobe", bg='#333333',
          fg="#FFFFFF", pady=5).grid(row=2, column=0)
    cijena_fld = Entry(self.edit_root)
    cijena_fld.grid(row=2, column=1, ipadx="35")

    submit1 = ttk.Button(self.edit_root, text="SPREMI",
                         command=lambda: self.azuFukcija(cijena_fld.get(),
                                                         kat_sobe.get()))
    submit1.grid(row=3, column=1)
```

Slika 24. Metoda za ažuriranje cijena

Izdavanje računa

Gumb za izdavanje računa je zadnja od glavnih metoda koja pokreće proces izdavanja računa gostu na odlasku iz hotela. Izgled samog računa odlučio sam izraditi pomoću „reportlab“ biblioteke zato što mi je izgledalo zanimljivo kako pomoću tablica je moguće izraditi cijeli račun i usput te tablice primijeniti na mnogo načina.

Račun se ispisuje ovisno koja rezervacija je odabrana iza stabla i kao za ostale metode uzimamo podatke iz baze. Odmah na početku petlje napravio sam da ima provjera ako gostu je potreban dodatni ležaj ili ako ima ljubimca i ako za obje kategorije cijene su fiksne i količine koje se ispisuju. Slijedi izgled tablica i kako će podatci na kraju izgledati na računu i napravio sam da prva tablica ima samo ime goste i broj sobe za bolju preglednost i iznad se nalazi broj

rezervacije kao naslov veličine dva. Izgled prve tablice je takav da će tekst biti centriran, spojiti će se stupci da bolje izgleda i na kraju samo razmak podebljanosti da ne izgledaju spojene tablice. Kod tablica za ovu biblioteku kad zadajemo nove stilove u zagradama prvo odredimo broj stupca pa broj retka i kraj stil se radi unatrag s negativnim brojevima kao što se radi s listama. Prije druge tablice napravio sam malu formulu za računanje noćenja gostu ovisno o dolasku i odlasku.

U drugu tablicu ispisuje se cijena smještaja, dodatni ležaj i ljubimac s time da odmah u prvom stupcu sam stavio da se automatski mijenja tekst ako gosti ima ove dodatne usluge ili ne. U stupcu za količinu je za sobu koliko je noćenja gost proveo, pod cijenom cijena jednog noćenja i na kraju je ukupna cijena cijelog boravka. Na kraju tablice ispiše se cijena svih usluga u eurima, ispod slijedi cijena u kunama i nakon svega pustio sam mjesto da se gost može potpisat na kopiju koja ostaje hotelu ili da je djelatnik potpiše da je gost platio. Izgled tablice ja drugačiji nego prvi pa ima u ovom slučaju crte koje određuju stupce i redove, pozadina je tamno sive boje s bijelim tekstom i rubovi su smeđe boje.

Zadnje preostaje odrediti gdje će se pdf račun spremiti i pod kojim nazivom. Odlučio sam napraviti da svaki pdf račun izdan ima svoj broj rezervacije u imenu kako ne bi došlo do zbrke među računima i veličina računa biti će A4 da bude za printati također spremno. Ostalo je samo pozvati naredbu koja pdf račun napravi pod svim uvjetima koje smo odredili. Na kraju na slici 25 je primjer izdanog računa gostu kao što je definirano u kodu i polje za potpis odvojio sam crvenom crtom za bolju preglednost.

Broj rezervacije: 11

Ime gosta Ivan Senkovic

Broj sobe 204

Usluga	Kolicina	Cijena	Ukupna Cijena
Smještaj	2	300	600
Dodatni ležaj: DA	1	30	30
Ljbimac u sobi: NE	0	0	0

UKUPNO	= 630 €
	= 4725.0 kn
Potpis	_____

Slika 25. Prikaz računa

Ostale metode

Ostalo je još nekoliko metoda koje nisam prije opisao jer povezuju više metoda ili služe u pozadini za povezat međusobno sve funkciju aplikacije.

Prva metoda koja je najbitnija je „baza_pokreni“ jer je zaslužna za povezivanje aplikacije na bazu podataka. Prilikom korištenja metoda moramo prvobitno izjasniti dva atributa, prvi je „query“ koji je tekstualni podatak gdje je potrebno izraziti koju operaciju moramo raditi na bazi podataka i drugi atribut služi za parametre, bile to vrijednosti koje je potrebno uvrstiti ili tražena vrijednost iz baze. Unutar metode definiramo da „sqlite3“ se poveže na ime baze koja je zapisana na početku samog koda i da poprimi ime „conn“ za daljnje korištenje. Naredba „cursor“ je nužna zato što preko nje koristimo funkcije biblioteke „sqlite3“, na primjer ona pokreće proces prikupljanja podataka iz baze da dobijemo rezultate i isto ima mogućnost prikupljanja samo jednog podatka ako je to potrebno. „execute()“ je jedna od naredba za prikupljanje podataka, a ostale su „fetchone“ „fetchall“ „fetchmany“ od kojih je samo jedna samo korištena u drugoj metodi. Na kraju metoda vraća rezultate koje možemo spremiti u bilo koji atribut.

```
def baza_pokreni(self, query, parameters=()):
    with sqlite3.connect(self.ime_db) as conn:
        cursor = conn.cursor()
        query_result = cursor.execute(query, parameters)
        conn.commit()
    return query_result
```

Slika 26. Metoda za pokretanje baze podataka

Sljedeća metoda bitna je „cijene_sobe“ putem koje sam naučio na svoj način kako da uzmem cijeli stupac iz baze podataka i koristim samo jedan podatak koji je u ovom slučaju cijena za određenu kategoriju sobe. Za ovu metodu se od „cursor“ naredbe koristi prije spomenuta naredba „fetchone“ za dohvaćanje samo jedne cijene. Ova metoda povezana je s metodom „cijena_ukupno“ koja računa cijenu noćenja gosta. Formulu za računanje izradio sam na svoj način tako što upisane datume dolaska i odlaska se oduzme i dobije broj dana, zatim broj dana sam pomnožio s cijenom sobe koja dolazi iz prijašnje metode. Rezultat ove metode morao sam stavit „string“ kako ne bi došlo do grešaka s upisom u bazu. Putem ove metode naučio sam kako s može raditi matematičke operacije između datum i dobiti rezultat u obliku

koji je potreban. Kao i prije navedeno, dodao sam 3 atributa koje je potrebno unijeti da se bolje može koristiti s drugim metodama.

```
def cijene_sobe(self, kat_fld):
    kat_sobe = kat_fld.get()

    con = sqlite3.connect("rezervacije.db")
    # zarez tako da ne bude tuple
    cursor = con.execute("SELECT kat_cijena FROM cijenikSoba "
                         "WHERE kategorija = ?", (kat_sobe,))
    return int(cursor.fetchone()[0])
```

Slika 27Metoda za dohvat cijene

```
def cijena_ukupno(self, kat_fld, dol_dat, odl_dat):
    self.razlika_datum = odl_dat.get_date() - dol_dat.get_date()
    self.ukupno_cijena = (int(self.cijene_sobe(kat_fld))) * abs(int(self.razlika_datum.days))
    # abs jer za vece datume daje neg vrijednost

    return str(self.ukupno_cijena)
```

Slika 28. Metoda za izračun ukupne cijene

Ovo su sve metode korištene koje sam izradio kako bi cijela aplikacija mogla funkcionirati. Metode su međusobno povezane tamo gdje je to bilo potrebno i pokušao sam da svugdje di je moguće, da se koristi baza podataka kako bi se aplikacija mogla koristiti za razne korisnike.

Zaključak

Rad koji sam sam napravio ispao je mnogo bolji nego što sam očekivao zato što se putem pisanja koda i izrade aplikacije pokazalo da ima mnogo više potencijala nego što sam prvobitno imao na umu. Mnogo sam naučio što mi je trebalo za ovaj rad i trebat će mi kasnije u struci. Objektno orijentirano programiranje se koristi svugdje u struci i Python je jezik koji sam uvijek htio bolje naučit i ovim putem vidio sam kako se razne biblioteke i različite vrste atributa mogu koristiti i međusobno sudjelovati. Prilikom pisanja koda imao sam mnogo puta greške zato što sam prvi puta radio s bazom podataka u Pythonu i nisam na početku znao izvor grešaka niti kako ih otkloniti. Kroz razna istraživanja na internetu i čitanja službenih dokumentacija uspio sam sve greške otkloniti i u nekim slučajevima čak izraditi kod na svoj način koji ja razumijem. Zadovoljan sam što sam odabrao ovu temu za rad jer se nadam da će mi pomoći za daljnje studiranje i također za naći posao u struci.

Završna verzija aplikacije je napravljena da bude kao scenarij prijave i odjave gosta iz hotela, ali naravno u hotela ima mnogo različitih operacija koje mogu i ne moraju biti vezane za prijavu i odjavu. Htio sam napraviti da se aplikacija može koristiti za hotel, ali i za ostale objekte, čak i iznajmljivanje apartmana. U prvoj verziji nisam koristio bazu podataka nego povezivanje na excel tablicu, što mi je drago da sam pokušao neovisno o tome što nije to zadnja verzija jer sam uspio tim putem naučiti dodatne mogućnosti Pythona i upravljanje datotekama. U kasnijim verzijama aplikacije moguće je dodati da bude uključeno i što je osoba konzumirala tijekom noćenja, kao na primjer restoran, wellness, bar i slično. Sa svim što sam naučio ovim radom znao bih kako unaprijediti aplikaciju i kako napraviti neku drugu aplikaciju ovisno o temi.

Literatura

A Simple Step-by-Step Reportlab Tutorial. (Rujan 2022.). Dohvaćeno iz https://www.blog.pythonlibrary.org/2010/03/08/a-simple-step-by-step-reportlab-tutorial/

GeeksforGeeks. (Rujan 2022.). Dohvaćeno iz https://www.geeksforgeeks.org/

Hide, Recover and Delete Tkinter Widgets. (Rujan 2022.). Dohvaćeno iz https://www.delftstack.com/howto/python-tkinter/how-to-hide-recover-and-delete-tkinter-widgets/

How to create a pop up message when a button is pressed in Python – Tkinter? (Rujan 2022.). Dohvaćeno iz https://www.geeksforgeeks.org/how-to-create-a-pop-up-message-when-a-button-is-pressed-in-python-tkinter/

HowTos - tkcalendar. (Rujan 2022.). Dohvaćeno iz https://tkcalendar.readthedocs.io/en/stable/howtos.html

Python datetime. (Rujan 2022.). Dohvaćeno iz https://www.programiz.com/python-programming/datetime#datetime

Python SQLite - Cursor Object. (Ruajn 2022.). Dohvaćeno iz https://www.tutorialspoint.com/python_data_access/python_sqlite_cursor_object.htm

Python Tkinter Menu bar – How to Use. (Rujan 2022.). Dohvaćeno iz https://pythonguides.com/python-tkinter-menu-bar/

Python Tutorial. (Rujan 2022.). Dohvaćeno iz https://www.pythontutorial.net/

SQLite Python: Creating a New Database. (Rujan 2022.). Dohvaćeno iz https://www.sqlitetutorial.net/sqlite-python/creating-database/

sqlite3 — DB-API 2.0 interface for SQLite databases. (Rujan 2022.). Dohvaćeno iz https://docs.python.org/3/library/sqlite3.html

stack overflow. (Ruajn 2022.). Dohvaćeno iz https://stackoverflow.com/

Tables and TableStyles. (Rujan 2022.). Dohvaćeno iz https://docs.reportlab.com/reportlab/userguide/ch7_tables/

tkinter — Python interface to Tcl/Tk. (Rujan 2022.). Dohvaćeno iz https://docs.python.org/3/library/tkinter.html

Tkinter Treeview. (Rujan 2022.). Dohvaćeno iz https://www.pythontutorial.net/tkinter/tkinter-treeview/

ttkthemes documentation. (Rujan 2022.). Dohvaćeno iz https://ttkthemes.readthedocs.io/en/latest/index.html

Using Existing Themes. (Rujan 2022.). Dohvaćeno iz https://tkdocs.com/tutorial/styles.html#usetheme

Popis slika