Izrada aplikacija s grafičkim sučeljem u jeziku Python

Sudec, Robert

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Sudec

Izrada aplikacija s grafičkim sučeljem u jeziku „Python“

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Sudec

Matični broj: 44882/16-R

Studij: Informacijski sustavi

Izrada aplikacija s grafičkim sučeljem u jeziku „Python“

ZAVRŠNI RAD

Mentor/Mentorica:

Dr. sc. Miran Zlatović

Varaždin, kolovoz 2019.

Robert Sudec

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema rada je izrada aplikacije s grafičkim sučeljem u jeziku Python. U prvom dijelu usporedit ćemo tako aplikacije s grafičkim sučeljem sa aplikacijama koje koriste tekstualno sučelje, te aplikacijama koje nemaju korisničko sučelje. Nadalje, odabrao sam nekoliko razvojnih okvira od kojih će svaki biti opisan, te će se u praktičnom dijelu u tim okvirima razvijati ista aplikacija. Na kraju pregledati ćemo prednosti i nedostatke svakog od razvojnih okvira. Koristit ćemo TkInter, PyQt, Kivy, WxPython, PySimpleGUI

Ključne riječi: python; gui; framework; tkinter; pyqt; kivy; pysimplegui; wxpython.

Sadržaj

1. Uvod

Tema rada kojeg čitate je izrada aplikacije s grafičkim sučeljem u programskom jeziku „Python“. Cijeli rad tako prati izradu jednostavne aplikacije u nekoliko odabranih razvojnih okvira.

Za početak objasnit ću ključne pojmove koji se već spominju u samoj temi ovog rada. Što je to sučelje, što je to grafičko sučelje, što je Python i kako se koriste razni razvojni okviri.

Python je vrlo popularan interpreterski programski jezik visokog stupnja. Stvorio ga je Guido van Rossum 1991. godine. Python je napravljen tako da naglašava čitljivost koda korištenjem praznog prostora (eng. *whitespace*). Također, Python nudi određene konstrukte, te objektno-orijentirani pristup kako bi korisnici lakše razvijali manje i veće projekte, kao što je to rekao Kuhlman [1].

Python je popularan jezik, a grafička su sučelja najzastupljenija, estetski najugodnija oku, te najjednostavnija za rad. Svrha rada je proširenje znanja o Pythonu kao programskom jeziku, te njegovim okvirima za razvoj aplikacija s grafičkim sučeljem s ciljem da nakon čitanja čitatelj bez problema kreće u samostalnu izradu aplikacija s grafičkim sučeljem u Pythonu.

Motivacija za izradu rada na ovu temu proizlazi iz želje studenta da proširi svoja znanja u ovim područjima, te činjenica da, uz razvojne okvire o kojima ću pisati, je moguće izrađivati aplikacije za različite platforme, a naglašavam mobilne platforme.

2. Metode i tehnike rada

Ovaj rad možemo podijeliti u dva dijela. Prvi dio će biti teorijski i igrat će ulogu detaljnijeg uvoda u temu kako bi čitatelj postao spreman za čitanje praktičnog, glavnog dijela ove teme. Tako će se opisivati poznata korisnička sučelja, što su to razvojni okviri (eng. *framework*) koji se koriste u ovom radu, te kratki opis korištenih alata.

Drugi dio, praktični dio će se sastojati od općenitog opisa željene aplikacije, te koraci izrađivanja te aplikacije u odabranim okvirima, pojašnjenja dijelova koda, prikaz prozora aplikacije. Rad će završiti usporedbom tih okvira.

Korišten će biti Python 3.7.3 zajedno sa JetBrains PyCharm Professional 2019.1 okruženjem i upravitelj paketa za Python - pip 19.2.2. Kasnije ću navesti koji su paketi potrebni za koji razvojni okvir, te kako instalirati te pakete

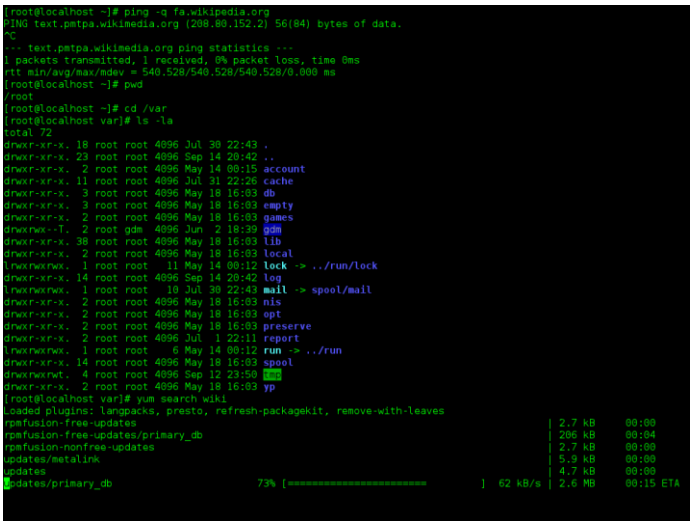
3. Vrste korisničkih sučelja kod aplikacija

U ovom području informatike definicija sučelja glasi da je sučelje veza između dvije ili više odvojenih komponenata u računalnom sustavu preko koje oni razmjenjuju informacije. Sučelje tako može povezivati software, hardware, periferijske uređaje, ljude. Tako kaže Hookway [2].Nas će u ovom radu zanimati korisničko sučelje (eng. *User interface)*, odnosno mjesto gdje se obavlja razmjena informacija između računala i čovjeka. Opet, postoje različiti tipovi korisničkih sučelja i već se desetljećima razvijaju. Kako je rečeno, razvijat ćemo najmoderniji tip, GUI (eng. *graphical user interface)*, odnosno grafičko korisničko sučelje.

3.1. Command-line interface (CLIs)

Sučelje naredbenog retka je sučelje koje održava interakciju s računalnim programom na način da korisnik govori programu što da radi u obliku linije teksta, naredbe. Program koji odrađuje tu interakciju naziva se interpreter naredbenog retka, ili Shell.

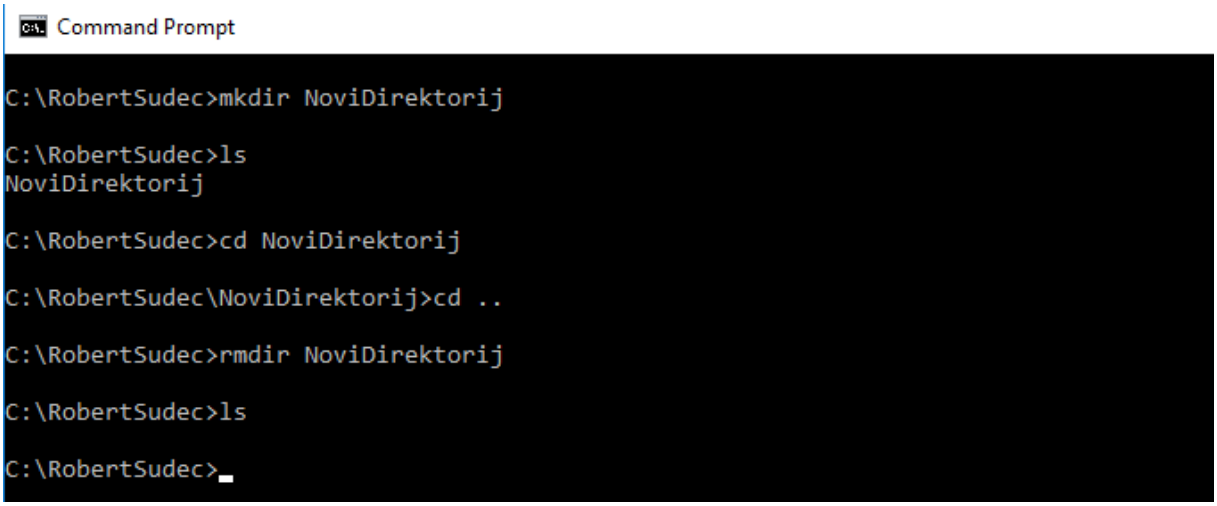
Ovakvo sučelje je bilo primarni tip interakcije s većinom računalnih sustava u kasnijim 1960-ima, a nastavili su se koristiti do danas.



Slika 1 Primjer sučelja naredbenog retka [3]

CLI se koristi kada postoji veliki broj naredbi ili upita sa zasebnim opcijama. Tada je lakše i brže reći programu što da radi nego da implementiramo sve funkcije u grafičko sučelje. Najčešće je to primjer sa operacijskim sustavima i njihovim naredbenim recima. Mogu se

koristiti i u sustavima koji nemaju dovoljno resursa da pruže grafičko sučelje. [4]

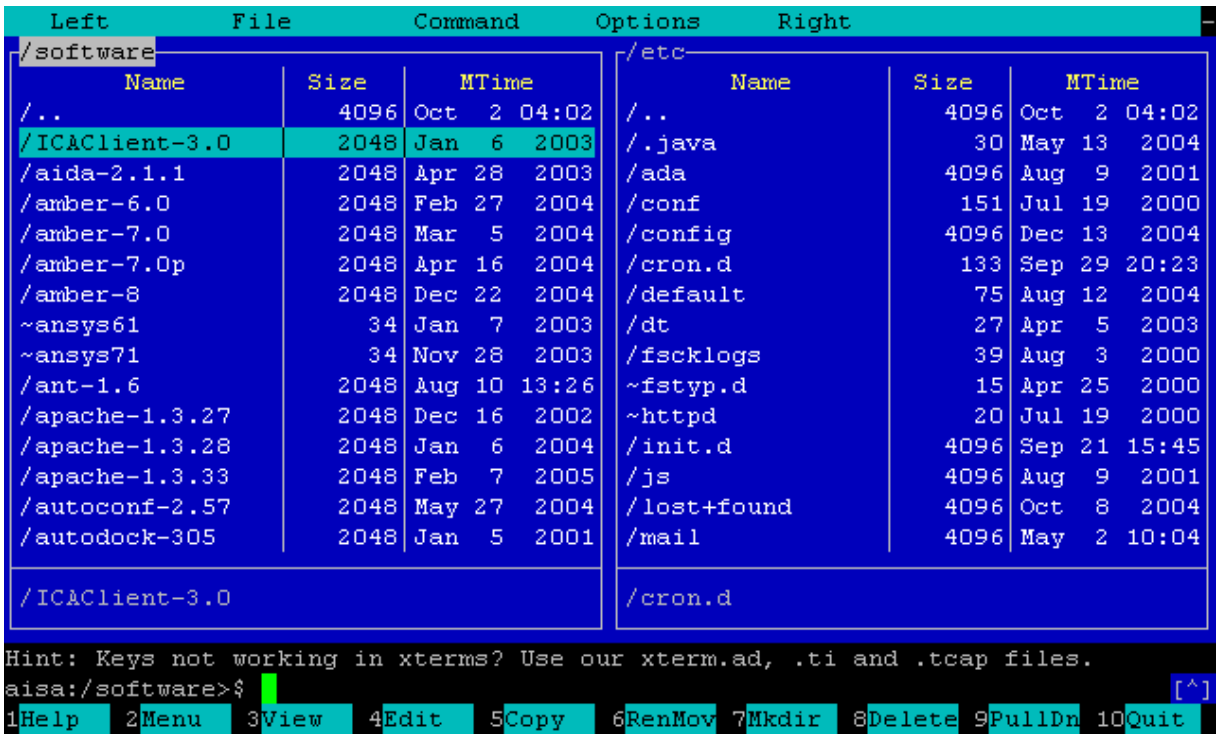


Slika 2. Rad sa direktorijima u Windows command prompt (autorski rad)

3.2. Text based user interface ili SAA

1985. godine, na početku razvoja Microsoft Windows i ostalih grafičkih korisničkih sučelja, IBM je kreirao Systems Application Architecture (SAA) standard koji je uključivao Common User Access (CUA). CUA je ono što danas koristimo u Windows OS-u, također je tada većina Windows Console Applications koristila taj standard.

Taj standard je definirao da sustav izbornika treba biti na vrhu prozora, prikaz statusa na dnu aplikacije i da tipke prečice ostanu na istoj funkcionalnosti [5].



Slika 3. Primjer tekstualnog sučelja [6]

3.3. Graphical user interface

Grafičko korisničko sučelje je oblik korisničkog sučelja koje omogućava korisnima interakciju sa elektroničkim uređajima pomoću ikona i vizualnih indikatora.

1963. godine razvijen je Sketchpad kojim se smatra prvi program sa „grafikom“. U 1970-ima proširuju se ideje o korisničkom sučelju u Xerox-u te počinju koristiti GUI kao glavno sučelje iz kojih nastaju većina današnjih grafičkih korisničkih sučelja. [7]

Takav sustav sastojao se od prozora, izbornika, radio dugmići i check box-ova. Alternativni akronim WIMP označava *windows, icons, menus, pointing device.*

Godinama se razvijalo ovakvo sučelje dok sredinom 1980-ih Apple nije popularizirao takva sučelja.

Slika 4. 1983. izdana je Apple Lisa [8]

Još uvijek kritičari nisu podržavali koncept grafičkih sučelja navodeći da se javljaju problemi sa hardware-om i kompatibilnim software-om. Godinu dana kasnije, 1984, Apple reklamom nagovara javnost da razmišlja o takvom sučelju kao svoje, osobno računalo, Friedmann [9], Chapter 5].

Tako nastaje Windows 95 i ostvaruje veliki uspjeh, te ubrzo postaje najpopularniji operacijski sustav za osobna računala.

Slika 5. Windows 95 [10]

Slika 6. Windows 10 [11]

Na slikama možemo vidjeti unaprjeđenje u GUI od 1995 do danas.

4. Razvojni okvir

U računalnom programiranju, razvojni okvir je apstrakcija u kojem se software koji pruža općenitu funkcionalnost mijenja korisničkim kodom i tako se svrha software-a sužava, odnosno postaje specifičnija. To je univerzalno okruženje koje pruža određenu funkcionalnost kao dio veće platforme za razvoj programa, aplikacija i rješenja. Takvi okviri mogu sadržavati potporne programe, biblioteke, skupove alata, API (eng. *Application programmable interface)* koji spajaju komponente u razvoju projekta ili sustava.

Postoje ključne razlike koje razvojne okvire čine razvojnim okvirima, za razliku od običnih biblioteka. U okviru, tijek rada kontrolira sami razvojni okvir, a ne korisnik. Korisnik može proširiti razvojni okvir prepisivanjem (eng. *override*) ili dodavanjem koda određene funkcionalnosti. Kod razvojnog okvira se općenito ne mijenja. Riehle [12]

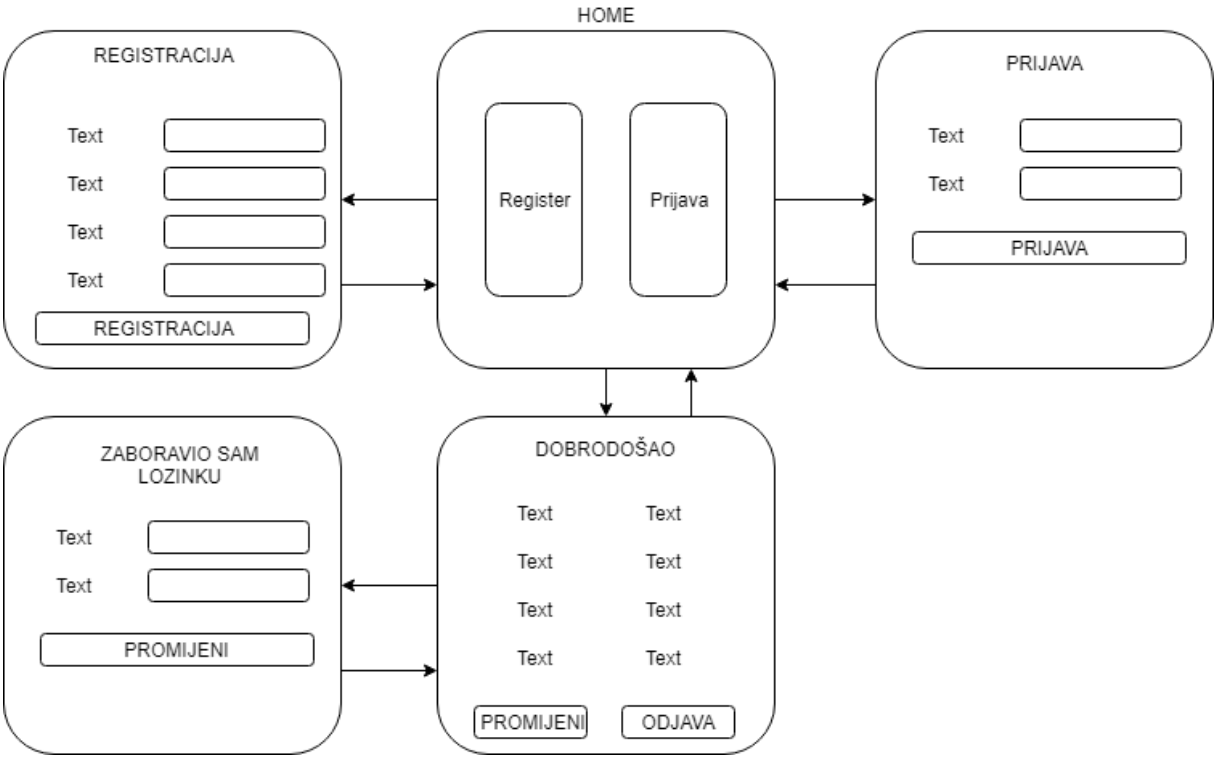
5. Aplikacija – rad s datotekama

5.1. Opis aplikacije

U ovom radu prikazat ću kako se razvija aplikacija i njen GUI. No aplikacija koja ima samo grafičko sučelje, a bez programske logike nam ne vrijedi. Stoga će se moja aplikacija na logičkoj razini baviti upravljanjem datotekama i to prikazati kroz sustav korisničkog profila. Korisnik će se morati registrirati sa svojim podacima i lozinkom gdje se korisnik zapisuje u, možemo ju zvati, tekstualnu bazu podataka. Zatim se korisnik mora prijaviti, a aplikacija će provjeriti u našoj bazi podataka postoji li taj korisnik i ako postoji, prikazat će se njegov korisnički profil u kojemu pišu podaci o korisniku. Tako smo pokrili čitanje i pisanje od glavnih operacija za rad s datotekama. Na korisničkom profilu, korisnik ima dvije mogućnosti. Prvo, može se odjaviti što ga vodi na početni izbornik registracija/prijava, te drugo, promijeniti lozinku što još simulira operaciju ažuriranja tekstualne datoteke.

5.2. Razvoj

Struktura aplikacije u svim okvirima teži prema jednostavnosti i sličnosti. Znači da aplikacije vizualno nisu uređivane i nisu estetski napredne. Aplikacija će imati 4 prozora (eng. *window, frame, form*).



Slika 7. Grafički prikaz izgleda aplikacije i navigacije (autorski rad)

Na slici vidimo zapravo 5 prozora, ali prave prozore čine „Registracija“, „Prijava“, „Home“, i „Zaboravio sam lozinku“, dok se korisnički profil prikazuje na „Home“ prozoru kada se korisnik prijavi. Također vidimo kako se korisnik može kretati kroz aplikaciju.

Svaka aplikacija prati ovaj dijagram, odnosno sadrži 4 prozora, osim u okvirima koji ne dozvoljavaju takvu izradu. Ideja je za svaki sljedeći razvojni okvir navesti općenite informacije o samom okviru i prikazati jedan do dva spomenuta prozora.

5.2.1. TkInter

TkInter je vezan za Tk set alata i može se reći da je to standardni razvojni okvir za GUI u Python-u. Software je besplatan i spada pod licencu Pythona, a dolazi sa standardnim instalacijama Python-a na Linux, Microsoft Windows i Mac OS X sustavima.

Kod TkIntera postoje nekoliko jednostavnih koraka za izradu aplikacije s grafičkim sučeljem. Uvezi (eng. *import*) Tkinter modul, stvori glavni prozor, dodaj programčiće, tzv. *widget-e* u aplikaciju i pokreni glavnu petlju koja prati sve događaje u aplikaciji.

Postoji broj *widget*-a kao što su tipke (eng. *button*), labele, polja za unos teksta i slično. U mojoj aplikaciji neću koristiti sve, nego najosnovnije *widget*-e. Zatim svaki od njih ima atribute, svojstva kojima ih upravljamo, kao što su dimenzije, boje, font i sl. [13]

Postoje 3 načina upravljanja rasporedom kod TkInter-a, a to su *pack()* koji slaže *widget-e* redom kako nastaju, kao blokove. Zatim imamo *grid()* metodu slaganja koja organizira programčiće u tabelarnu strukturu, te *place()* koji nam omogućava apsolutno pozicioniranje. [14]

U nastavku ću prikazati dio moje aplikacije koja se bavi registracijom korisnika. Za početak moramo uvesti nama potrebne module i stvoriti klase koje opisuju prozore. Osim tkinter-a, trebat će nam i modul operacijskog sustava za upravljanje datotekama.

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import os


class GlavniWin:
      def __init__(self, master):
            self.master = master
            self.frame = Frame(self.master)
            self.setup("")
            def setup(self, username):
                  self.btnReg = Button(self.frame, text='Registracija',
                        font=("Montserrat", 11, "bold"), width=25,
                        command=self.clickReg, height=30, bd=8,
                        bg="#c3c3c3")
                  self.btnReg.pack(pady=30, padx=15, side=LEFT)
                  self.btnLog = Button(self.frame, text='Prijava',
                        font=("Montserrat", 11, "bold"), width=25,
                        command=self.clickPrijava, height=30, bd=8,
                        bg="#c3c3c3")
```

```
            self.btnLog.pack(pady=30, padx=15, side=LEFT)
            self.frame.pack()
def clickReg(self):
      self.newWindow = Toplevel(self.master)
      self.newWindow.geometry("620x480")
      self.app = RegWin(self.newWindow)
```

Ovo je klasa koja opisuje prozor koji se prvi pojavljuje prilikom pokretanja. U Tkinteru bitne su veze *parent-child*, ovdje će *master* biti objekt *Tk()*, korijenski objekt za sve Tkinter aplikacije. Prvi *widget* s kojim se susrećemo je *Frame()* i argumentom ga vežemo za roditelja. *Frame* postoji kako bi mogli u njega slagati ostale *widget-e* koji se prikazuju. Metoda *setup()* će nam poslagati sve *widget-e*. Napomenut ću da je ovo kod samo za funkcionalnost registracije, ostatak koda sam izostavio zbog jednostavnoosti.

Slijedi instanciranje željenih *widget-a*, a to su 2 *Button-a*, jedan za registraciju, drugi za prijavu. Prilikom instanciranja, odmah navodimo, tj. specificiramo određene atribute, tipa *text*, što će pisati na samom *button-u*, *font, width, height* i sl. Nama najbitnije svojstvo je *command* pomoću kojeg određujemo funkciju/metodu koja se poziva pritiskom na dugme, a to će biti metoda *clickReg()*. *Button* je instanciran, ali još se neće prikazati na ekranu, sve dok se ne pozove spomenuta metoda za organizaciju *widget-a*, *pack().* Widget će se automatski pozicionirati u svojeg roditelja koji je naveden kao prvi argument prilikom instanciranja. Također, možemo odrediti neka svojstva, tipa *padding, side* za odabir strane i sl. Prisjetite se da smo onaj *Frame() widget* samo instancirali, pa stoga moramo još i njemu pozvati metodu *pack().* S ovim što sada imamo, *GlavniWin()* izgledao bi ovako:

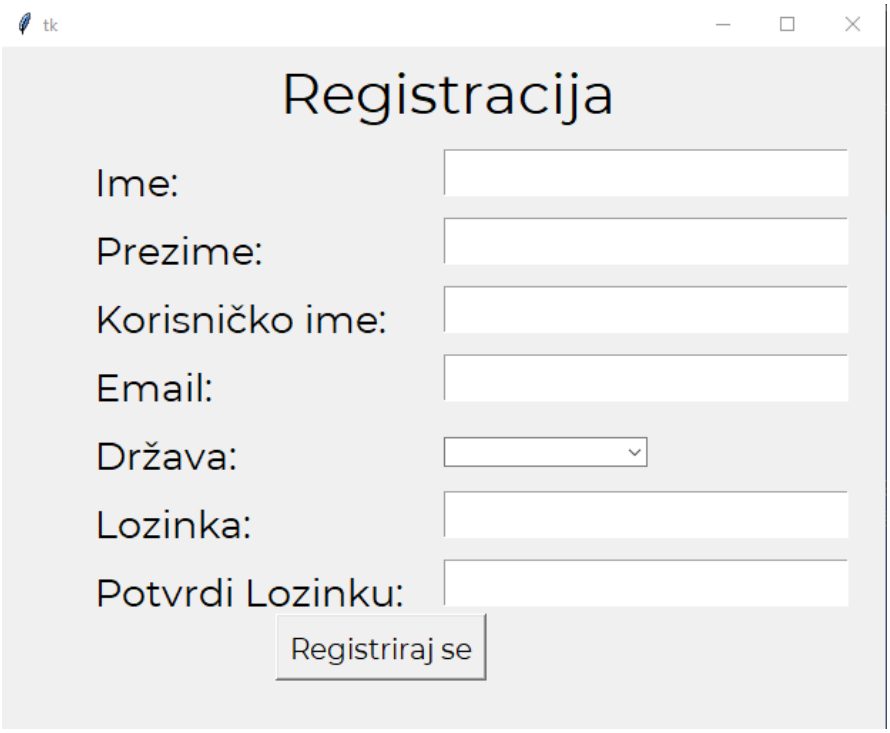
Slika 8. Izgled "Home" prozora u TkInteru [Autorski rad]

Klikom na Registraciju instanciramo widget tipa *Toplevel()* što označava samostalni prozor i pozivamo konstruktor klase prozora za registraciju *RegWin()*.

```
class RegWin:
      def __init__(self, master):
            self.master = master
            self.lblnaslov = Label(self.master, text="Registracija",
                  font=("Montserrat", 30))
            self.lblnaslov.place(relx=0.31, rely=0)
            self.lblEmail = Label(self.master, text="Email:",
                  font=("Montserrat", 20))
            self.lblEmail.place(relx=0.1, rely=0.45)
            self.txtEmail = Entry(self.master, font=("Montserrat", 16))
            self.txtEmail.place(relx=0.50, rely=0.45)
            self.lblCountry = Label(self.master, text="Država:",
                  font=("Montserrat", 20))
            self.lblCountry.place(relx=0.1, rely=0.55)
            self.listCountry = ttk.Combobox(self.master,
                  values=["Hrvatska",          "Srbija", "Bosna i
                  Hercegovina", "Slovenija"], height=20)
            self.listCountry.place(relx=0.5, rely=0.57)
            self.lblLozinka = Label(self.master, text="Lozinka:",
                  font=("Montserrat", 20))
            self.lblLozinka.place(relx=0.1, rely=0.65)
            self.txtLozinka = Entry(self.master, font=("Montserrat", 16),
                  show="*")
            self.txtLozinka.place(relx=0.50, rely=0.65)
            self.btnReg = Button(self.master, text="Registriraj se",
                  font=("Montserrat", 15))
            self.btnReg.place(relx=0.31, rely=0.83)
            self.btnReg.bind("<Button 1>", self.readData)
```

Ne želim ponavljati iste stvari, tako da sam i ovdje izdvojio bitne dijelove. Krećemo sa *Label() widget-om*, to je jednostavno tekst koji se prikazuje, najčešće za opisivanje i označavanje. Na ovom prozoru koristit ćemo drugu metodu organiziranja rasporeda, a to je *place()* koja radi apsolutno pozicioniranje po širini i visini. Uzmimo *lblnaslov* za primjer, ona će biti postavljena na 31% x-osi, te na 0% y-osi, ako znamo da gornji lijevi kut ima koordinate (0,0) znači da će *lblnaslov* biti na vrhu i horizontalno bliže sredini. Kako se na prozoru registracije ponavljaju *Label()* i *Entry()* za skoro svaki podatak, prikazao sam samo za podatak e-mail. Također ima svoju labelu *lblEmail* i pripadajuće polje za unos teksta, što se naziva *Entry()*, njegovi atributi su analogni ostatku *widget-a*. Zatim odabir države stavljen je u padajuću listu, odnosno *Combobox()* kojeg smo uveli iz *ttk* modula. Tom *widget-u* moramo specifirati atribut *values* koji prima polje s željenim državama. I na kraju podatak lozinka analogno ostalim ima svoj *lblLozinka* i *txtLozinka* kao *Entry()*, ali s posebnim atributom *show* pomoću kojeg specificiramo koji će se znak pokazivati u polju za unos. Na kraju dolazi nam *Button()* koji će potvrditi unos korisnika [15]. Već smo vidjeli kako se on instancira, ali ovdje vidimo drugi način vezanja metode za pritisak na dugme, a to je metoda *bind()*, a *<Button 1>*

je oznaka za lijevi klik miša i time se poziva metoda *readData().* Prozor „Registracija“ sada bi trebao izgledati ovako:



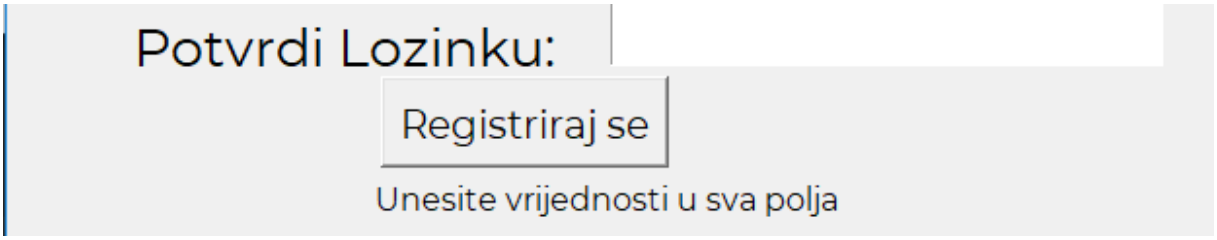
Slika 9. Izgled prozora Registracije u TkInter

Metoda *readData()* bavi se preuzimanjem podataka iz svih polja, tj. Preuzima korisničke podatke pomoću metode *get().*

```
def readData(self, event):
      self.ime = self.txtIme.get()
      self.prezime = self.txtPrezime.get()
      self.username = self.txtUsername.get()
      self.email = self.txtEmail.get()
      self.country= self.listCountry.get()
      self.lozinka = self.txtLozinka.get()
      self.potvrdaloz = self.txtPotvrdiLoz.get()
      if(not self.ime or not self.prezime or not self.username or not
                  self.email or not self.country or not self.lozinka or not
                  self.potvrdaloz):
            self.lblErr["text"] = "Unesite vrijednosti u sva polja"
      elif(not(self.lozinka == self.potvrdaloz)):
            self.lblErr["text"] = "Lozinke ne odgovaraju"
      else:
            self.f = open("users.txt", 'a')
            self.f.writelines(self.ime + ";" + self.prezime + ";" +
                  self.username + ";" +                    self.email + ";"
                  + self.country + ";" + self.lozinka + ";" + "\n")
            self.f.close()
            messagebox.showinfo(title="Registracija", message="Uspješno ste
                  se registrirali, od sada se možete prijaviti u svoj
                  račun")
            self.master.destroy()
```

Zatim moramo provjeriti podatke, jer korisnik mora obavezno unijeti podatke u sva polja. Labela *lblErr* označava labelu na dnu prozora koja se instancira prazna, tj. bez ikakvog

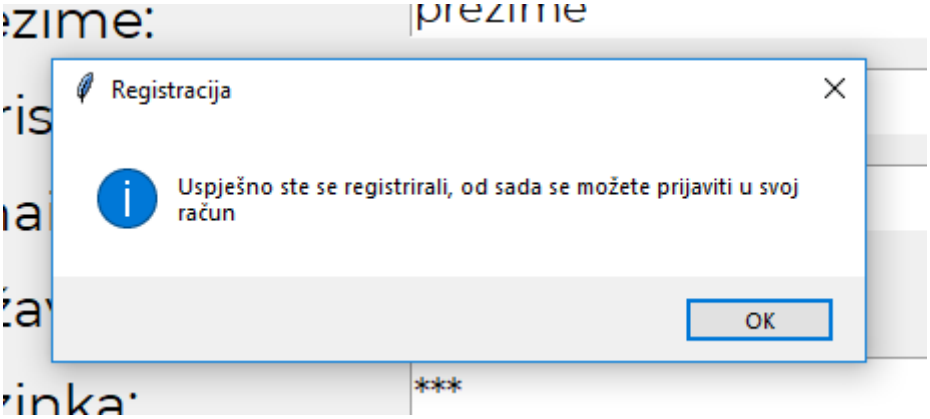
teksta i ona služi za povratnu informaciju korisniku, informacije o greškama. Isto se događa ako polja *lozinka* i *potvrdaLozinke* nemaju iste vrijednosti. To izgleda ovako:



Slika 10. Label sa informacijom o greškama [Autorski rad]

Tek kad su podaci ispravni otvaramo datoteku *users.txt* u *append ('a')* načinu rada pomoću metode *open(). Append* mod nam omogućava pisanje u datoteku na kraj, znači dodavanje teksta. Pomoću metode *writelines()* zapisujemo u datoteku korisnika kojem su podaci odvojeni znakom „;“. Primjer: *ime;prezime;korisničko_ime;email;država;lozinka*. Tako će svaki korisnik zauzeti jedan red u datoteci.

Još jedan način davanja povratne informacije je *messagebox()* koji je zapravo mali iskočni prozor (eng. *pop-up window)* koji osigurava da informacija dođe do korisnika.



Slika 11. Iskočni prozor - tkInter [Autorski rad]

Iskočni prozor otvara se u *modal* načinu rada, odnosno izvršavanje koda ovdje staje sve dok se prozor ne zatvori. Kada se ugasi *pop-up* slijedi metoda *destroy()* koja će ugasiti, odnosno zatvoriti prozor „Registracija“.

Ništa još od ove aplikacije ne radi, samo smo definirali klase i metode. Kao što je rečeno, nedostaje nam korijenski objekt svake Tkinter aplikacije, *Tk()* i pozivanje glavne petlje aplikacije. To možemo dodati na kraj naše .py datoteke jer kako bi mogli pozvati *GlavniWin()*, ta klasa mora biti napisana prije pozivanja.

```
root = Tk()
root.geometry("640x480")
app = GlavniWin(root)
root.mainloop()
```

Tako će *root* postati *Tk()* objekt i on će biti roditelj našem prozoru *GlavniWin()*, te ga pokrećemo sa metodom *mainloop().* Tako izgleda razvoj aplikacije s grafičkim sučeljem u okviru Tkinter.

5.2.2. PyQt

PyQt jedan je od najpopularnijih razvojnih okvira za Python koji koristi Qt *cross-platform* okvir. Razvili su ga u „Riverbank Computing Limited“.

PyQt4 se može koristiti na Windows, Linux, Mac OS X i raznim UNIX platformama, dok se novija verzija PyQt5, koju ćemo mi koristiti, može pokretati i na Android i iOS sustavima. Najnovije stabilne verzije možete preuzeti na njihovoj web stranici *riverbankcomputing.com.*
PyQt je također baziran na OOP, a sam API sadrži više od 400 različitih klasa. Jedna od najbitnijih klasa je *QApplication* koja upravlja postavke i tijek aplikacije. Ona sadrži glavnu petlju koja procesuira događaje koje stvaraju ostali elementi. Zatim klasa *QWidget* koja je temelj za sve objekte na korisničkom sučelju, od kojih su neki *QLabel, QLineEdit, QPushButton, QComboBox, QDialog* i slični. [16]

Spomenut ću da PyQt *installer* dolazi sa pomoćnim alatom koji ima svoj GUI, a naziva se *Qt Designer*. Koristeći *drag-and-drop* sučelje možemo izgraditi svoj GUI bez pisanja koda. Spomenut ću da nije IDE kao što je *Visual Studio*, nema mogućnosti *debug* i *build.*
Kao što je i Tkinter imao mogućnosti organiziranja *widget-a*, ima i PyQt. Bilo koji *QWidget* može se pomoću metode *setGeometry()* pozicionirati apsolutno navođenjem koordinata na ekranu. No, postoje i *widget-i* koji su zapravo „upravitelji rasporeda“ (eng. *layout manager)*. Prednosti nad apsolutnim pozicioniranjem jer da se *widget* automatski prilagođava, osigurava se isti izgled na ekranima s različitom rezolucijom, te je puno lakše dodati novi ili obrisati stari *widget.* To su: *QBoxLayout, QGridLayout, QFormLayout.*[17]

U nastavku ću prikazati, opet dijelove svoje aplikacije vezane za prijavu korisnika u aplikaciju. Ovdje ćemo samo uvesti sve iz paketa *QtWidgets* i *os*.

```
from PyQt5.QtWidgets import *
import os

class GlavniWin:
        def __init__(self):
                self.glavni = QWidget()
                self.glavni.resize(640, 480)
                self.logged = False
                self.username= ""
                self.setup(self.username)
```

Instanciranjem *QWidget()* dobijemo prazan prozor kojemu možemo postaviti veličinu i ostale atribute. Kao što sam rekao, početni izbornik prijava/registracija će biti zajedno sa korisničkim

profilom na jednom prozoru, zato imamo *logged* zastavicu kako bi pratili što trebamo prikazati na prozoru u metodi *setup()* i čuvamo trenutno prijavljenog korisnika u varijabli *username*.

```
def setup(self, username):
      self.glavni.destroy()
      self.glavni = QWidget()
      self.glavni.resize(640, 480)
      self.glavni.setWindowTitle("PyQt Demo Robert Sudec")
```

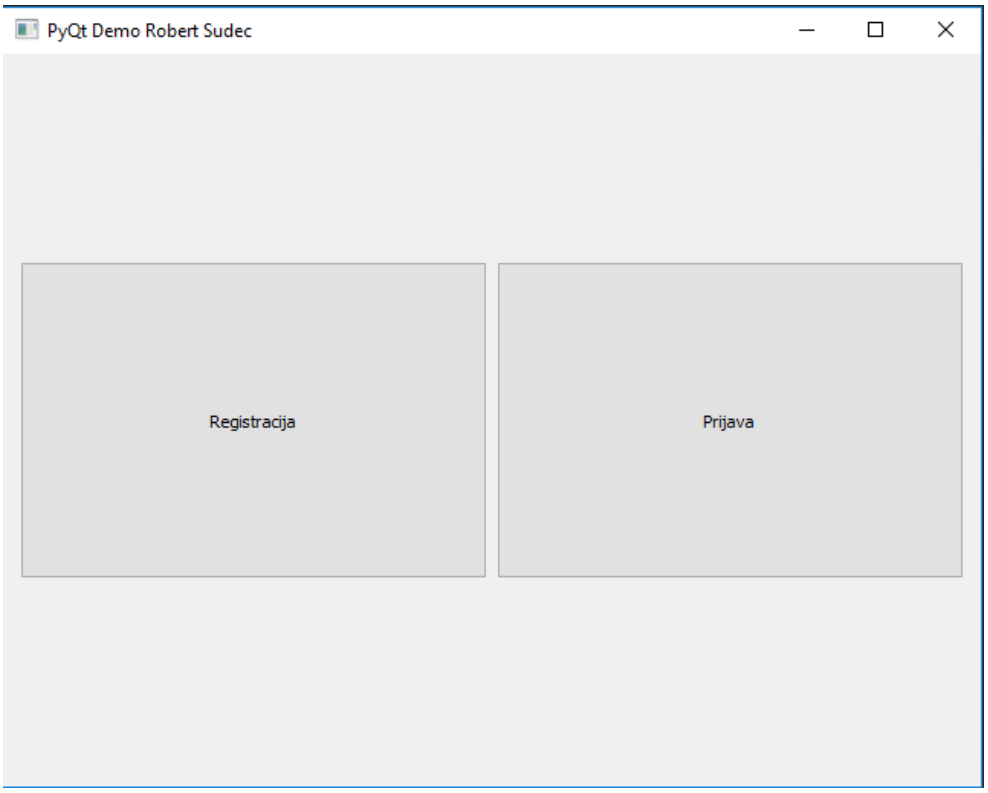
Metoda *setup()* služi za postavljanje rasporeda, a kako se raspored mijenja prilikom prijave ili odjave, *widgete* moramo brisati i ponovo dodavati, zato ćemo prilikom ponovnog postavljanja uništiti stari prozor i stvoriti novi, radi jednostavnosti.

```
      if not self.logged:
            self.layout = QHBoxLayout()
            self.registerButton = QPushButton("Registracija")
            self.loginButton = QPushButton("Prijava")
            self.registerButton.setStyleSheet("""
                  height: 200%;
            """)
            self.loginButton.setStyleSheet("""
                  height:200%;
            """)
```

Zatim ćemo provjeriti je li zastavica *logged* istinita ili ne i po tome znamo koje je stanje aplikacije. U slučaju da nije istinita, moramo prikazati početni prozor prijava/registracija. Koristimo *layout* za organizaciju rasporeda i u ovom slučaju on će biti *QHBoxLayout()*, što je horizontalni jednoredni raspored u koji se *widgeti* slažu slijedno. Odmah dodajemo dva *QPushButton() widgeta*, odnosno tipke za prijavu i registraciju koji u argument primaju tekst koji će opisivat te *buttone*. Zanimljiva stvar je da PyQt podržava stilske listove, odnosno CSS3 stilove kod svojih *widgeta* tako da možemo lako oblikovati sve elemente.

```
            self.registerButton.clicked.connect(self.register)
            self.loginButton.clicked.connect(self.login)
            self.layout.addWidget(self.registerButton)
            self.layout.addWidget(self.loginButton)
            self.glavni.setLayout(self.layout)
            self.glavni.show()
```

Bitna stvar kod *Button-a* je onaj događaj koji korisnik izazove pritiskom na njega, tako moramo i ovdje definirati pomoću svojstva *clicked* i metode *connect()* koja će se akcija poduzeti. U ovom primjeru bavimo se prijavom korisnika, pa ćemo ubrzo pogledati metodu *login()* koja se poziva ovdje. Morat ćemo nakon stvaranja *widgeta* iste dodati u naš raspored, *layout*. Slijedno dodajemo pomoću *addWidget()* metode prvo *registerButton*, a zatim *loginButton*. Još moramo na glavni prozor *glavni* pridružiti *layout* metodom *setLayout()* i napokon prikazati prozor *glavni* metodom *show().* [17]



Slika 12. "Home" prozor u PyQt [Autorski rad]

Novi *Login* prozor otvaramo pomoću metode *login()*:

```
def login(self):
      self.loginS = LoginWin()
```

Još nismo završili sa glavnim prozorom, prikazali smo što se dešava ako korisnik nije prijavljen, ostao nam je *else* slučaj kada je korisnik prijavljen, no idemo se prvo pokušati prijaviti. (Napomena: od sada neću prikazivati *setStyleSheet()* metode)

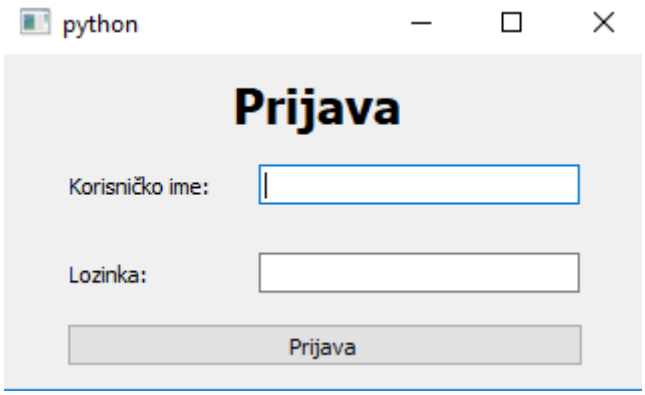
```
class LoginWin:
      def __init__(self):
            self.loginScreen = QWidget()
            self.loginScreen.resize(300, 150)
            self.layoutMain = QVBoxLayout()
```

Prozor s prijavom slagat ćemo vertikalno, zato smo ovdje koristili *QVBoxLayout()*. Kako bi mogli sada postaviti *QLabel* i *QLineEdit* horizontalno moramo ih zatvoriti u poseban *container* koji će imati svoj *layout.* Prvo, *userContainer* će biti tipa *QWidget* kako bi mu mogli dodati poseban *QLayout*.

```
            self.userContainer = QWidget()
            self.userLayout = QHBoxLayout()
            self.lblUser = QLabel("Korisničko ime: ")
            self.tboxUser = QLineEdit()
            self.lblUser.setFixedWidth(75)
            self.tboxUser.setFixedWidth(160)
            self.userLayout.addWidget(self.lblUser)
            self.userLayout.addWidget(self.tboxUser)
            self.userContainer.setLayout(self.userLayout)
```

Kreiramo i *userLayout* koji je horizontalan, te instanciramo *lblUser* kao labelu i *tboxUser* kao polje za unos teksta. Sada pridružujemo *widgete* u *userLayout* i onda postavljamo da *userContainer* koristi *userLayout.* Koristim *user* jer se taj *container* odnosi na unos korisničkog imena. Dio za unos lozinke analogan ovome, samo na objekt tipa *QLineEdit(),* koji se zove *tboxPass* pozovemo metodu *setEchoMode(QLineEdit.Password)* gdje je *QLineEdit.Password* predefinirano svojstvo koje će prikazivati "*". Još moramo dodati *container-e* na *layoutMain*, postaviti glavni layout i prikazati sam prozor:

```
self.layoutMain.addWidget(self.userContainer)
self.loginScreen.setLayout(self.layoutMain)
self.loginScreen.show()
```



Slika 13. Login prozor - PyQt [Autorski rad]

Još ću pokazati logički dio ove funkcionalnosti, odnosno metodu koja se poziva pritiskom na *button* Prijava, a to je *loginBtnClicked().*

```
def loginBtnClicked(self):
      self.username = self.tboxUser.text()
      self.lozinka = self.tboxPass.text()
      if not self.username or not self.lozinka:
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Warning)
            msg.setText("Unesite korisničko ime i lozinku!")
            msg.setWindowTitle("Upozorenje")
            msg.exec_()
```

Metoda preuzima podatke iz polja za unos teksta pomoću metode *text()* te provjerava jesu li prazni, ako je jedno polje prazno javlja se skočni prozor *QMessageBox()* kojem postavljamo ikonicu, tekst, naslov, a metoda *exec_()* pokreće glavnu petlju za *pop-up.*

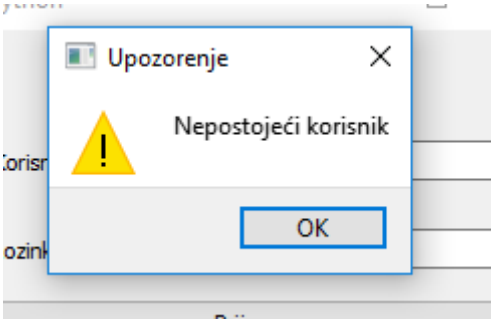
```
else:
      f = open("users.txt", "r")
      for line in f:
            user = line.split(";")
            if user[2] == self.username and user[5] ==
            self.lozinka:
                  window.logged = True
                  window.setup(self.username)
                  break
```

```
f.close()
```

U slučaju da su popunjena oba polja, metoda otvara datoteku, iterira kroz njene redove i preuzima podatke iz reda za svakog korisnika. Pomoću *split()* metode dobijemo polje *user* kojemu je svaki element jedan podatak između „;“ iz datoteke. Treći element je korisničko ime, a 6 element je lozinka, tako da te vrijednosti uspoređujemo sa podacima iz forme za prijavu, te ako smo pronašli korisnika, postavljamo zastavicu *logged* od glavnog prozora na *True* i pozivamo metodu za ponovni raspored glavnog prozora zajedno s korisnikom koji se trenutno prijavio. Još valja provjeriti jesmo li prošli kroz sve korisnike u datoteci, a to možemo jednostavno učiniti opet pomoću *logged* zastavice.

```
if window.logged:
      self.loginScreen.close()
else:
      msg = QMessageBox()
      msg.setIcon(QMessageBox.Warning)
      msg.setText("Nepostojeći korisnik")
      msg.setWindowTitle("Upozorenje")
      msg.exec_()
      self.tboxUser.clear()
      self.tboxPass.clear()
```

Ako se niti jedan korisnik nije pronašao, onda se javlja *popup* sa informacijom da ne postoji korisnik, aplikacija očisti polja za unos metodom *clear()* i korisnik se ponovo pokušava prijaviti.



Slika 14. QMessageBox - PyQt [Autorski rad]

Tako smo definirali naše prozore koji se koriste, treba još samo pokrenuti aplikaciju sa 3 linije koda:

```
app = QApplication([])
window = GlavniWin()
app.exec_()
```

Pozivamo GlavniWin(), te app.exec_() pokreće glavnu petlju te se pali „Home“ prozor.

5.2.3. Kivy

Kivy je besplatan i *open source* Python razvojni okvir koji se najviše koristi za mobilne aplikacije, odnosno aplikacije za uređaje koji podržavaju *touch/multitouch,* te se mogu pokretati na Android, iOS, Linux, OS X i Windows sustavima. Prva verzija je izdana 2011. godine. Ovaj

okvir sadrži sve elemente potrebne za izradu aplikacije kao što su podrška za miš, tipkovnicu, *multitouch* događaje, grafičku biblioteku, širok raspon *widgeta* koji također podržavaju *multitouch,* te napredni *Kv* jezik koji se koristi za opisivanje korisničkog sučelja i interakcije. U tom jeziku mogu se stvarati i vlastiti *custom widgets*. [19]

Kivy se može na Windows operacijskom sustavu instalirati pomoću pip upravitelja paketima sljedećom naredbom:

```
python -m pip install kivy==1.11.1
```

Kivy aplikacije se jednostavno izrađuju na način da se naslijedi klasa *App*, te se implementira metoda *build()* tako da metoda vraća objekt *widget* te se naša klasa instancira i pozove se metoda *run().*

Kivy također koristi *layout* za organizaciju ostalih *widgeta*, a postoji ih više, to su *BoxLayout, GridLayout, StackLayout, AnchorLayout, FloatLayout, RelativeLayout.* Opet ćemo koristiti *BoxLayout,* uz *GridLayout*.[18]

Kako će se naša aplikacija razvijati, stvaranje *widget tree-a* postaje teško održivo i nespretno. Zato ćemo koristiti Kv jezik, koji nam omogućava da stvorimo *widget tree* na deklarativni način. Omogućava brze izmjene korisničkog sučelja, a dobra značajka je to da će odvojiti logiku aplikacije od korisničkog sučelja. Logika će biti u našoj .py datoteci, dok će UI biti u posebnoj datoteci .kv.

Možemo krenuti na razvoj aplikacije, s time da ću u ovom okviru pokazati prijavu, korisnički profil i odjavu. Napomenut ću da u ovom okviru nema prozora, sve se odvija u jednom prozoru (jer je pretežno za *touch* uređaje), ali postoji *widget Screen* koji se ponaša kao prozor, a između različitih *Screen*-ova postoje tranzicije .

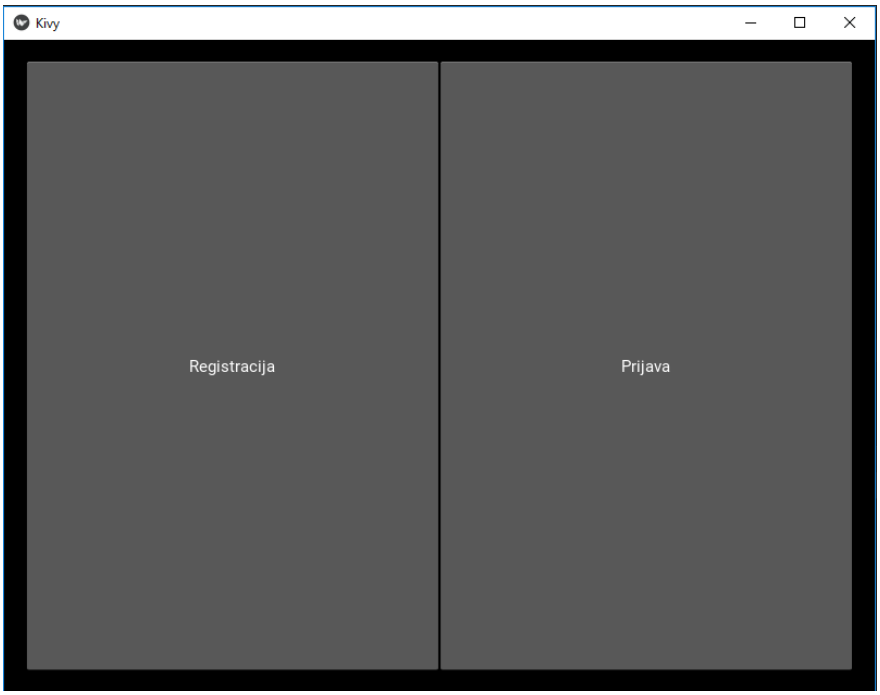
```
from kivy.app import App
from kivy.uix.label import *
from kivy.uix.button import *
from kivy.uix.boxlayout import *
from kivy.uix.popup import Popup
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen
import os

class Home(Screen):
    pass
class WindowManager(ScreenManager):
    pass
kv = Builder.load_file("Kivy.kv")
class KivyApp(App):
    def build(self):
        return kv

KivyApp().run()
```

Dohvatimo sve module koji su nam potrebni, zatim svaki prozor, odnosno 'ekran' u ovom slučaju ima svoju klasu koja nasljeđuje *Screen* klasu i jednu klasu upravitelj 'ekranima'. Potrebna je jedna klasa koja nasljeđuje *App* kao što sam rekao u uvodu ovog dijela i aplikacija

se pokreće metodom *run().* Ovo je sasvim dovoljno koda kada bi nam aplikacija imala samo *Home Screen* i izgledala bi ovako:



Slika 15. Home screen - Kivy [Autorski rad]

Od logike nema ništa, ekrani su povezani u Kivy.kv datoteci. Prvo navodimo za WindowManager klasu koje sve *window* elemente koristimo. Zapamti, svi *window* elementi koji su tu nabrojani, moraju imati svoje klase u logičkom dijelu aplikacije.

```
WindowManager:
    Home:
    Register:
    Login:
    Profile:
    Forgot:

<Home>:
    name: "Home"
    BoxLayout:
        orientation: "horizontal"
        padding:20
        Button:
            text: "Registracija"
            on_press:
                app.root.current = "Register"
                root.manager.transition.direction = "right"
        Button:
            text: "Prijava"
            on_press:
                app.root.current = "Login"
                root.manager.transition.direction = "left"
```

Sljedeće krećemo s definiranjem sučelja za „Home“ prozor. Atribut *name* označava ime preko kojeg možemo pristupiti tom *screen-u*, najčešće u slučaju navigacije. Zatim *widget tree* sadrži

BoxLayout kojeg definiramo kao horizontalnog i njemu pridružujemo 2 *Button-a*, registracija i prijava, koji imaju svoje *on_press* događaje, te se tu odvija navigiranje u preostale ekrane. Zadajemo aplikaciji trenutni ekran preko svojstva *app.root.current*.[19]

Prikazat ću kako izgleda profil, tu nam je potreban i logički dio. Ovdje se javlja događaj *on_parent* koji se poziva prilikom učitavanja roditeljskog *widgeta.* Tako ćemo dohvatiti sve podatke o korisniku iz aplikacije. Primijeti da *root* označava istoimenu klasu.

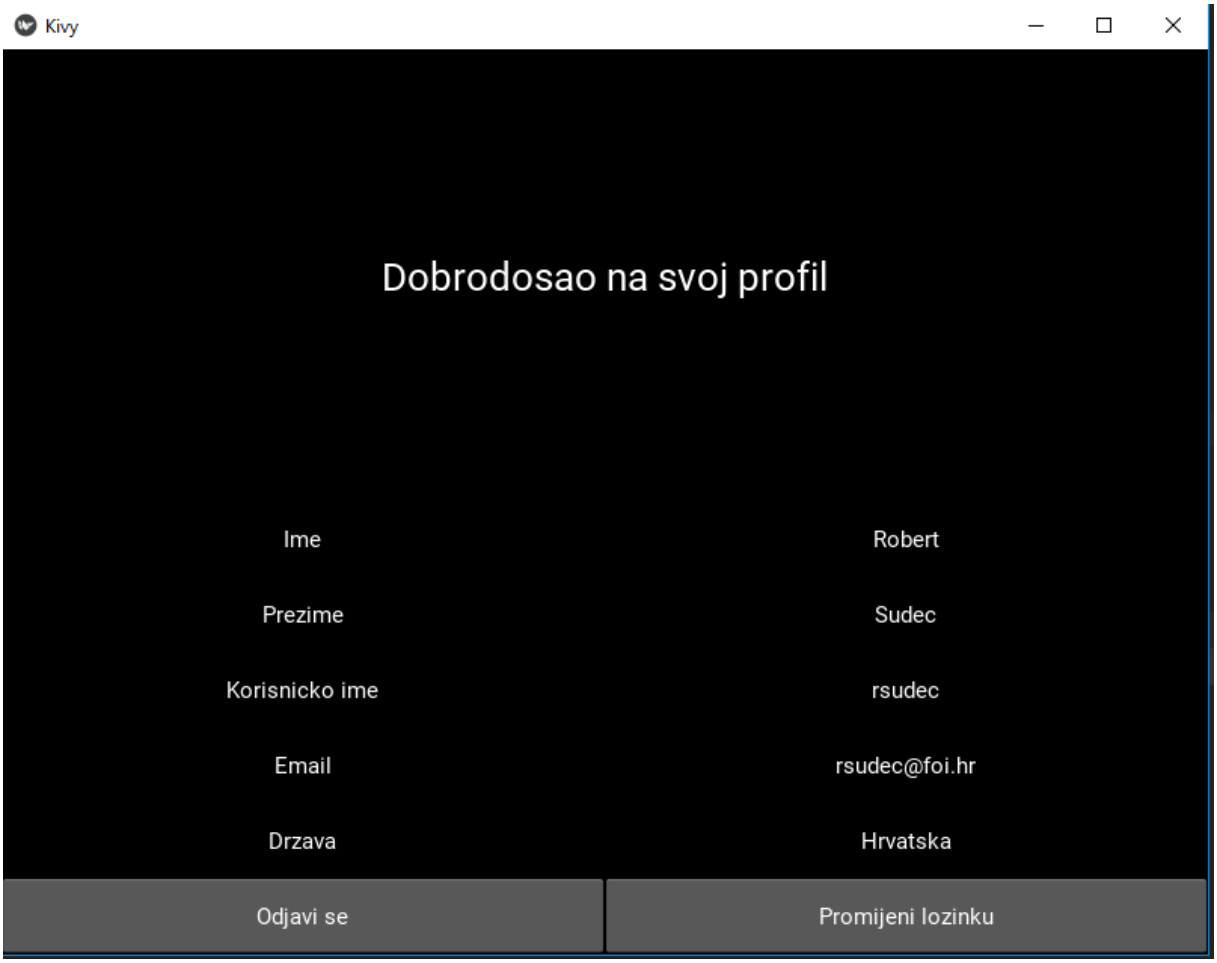
```
<Profile>
    name: "Profile"
    on_parent:
        user = root.osvjezi()
        tboxIme.text = root.getName()
    BoxLayout:
        orientation: "vertical"
        Label:
            text: "Dobrodosao na svoj profil"
            font_size: "20pt"
        GridLayout:
            cols: 2
            Label:
                text: "Ime"
            Label:
                id: tboxIme
                text: ""
                  .
                  .
                  .
            Button:
                text: "Odjavi se"
                on_release:
                    app.root.current = "Home"
                    root.manager.transition.direction = "right"
            Button:
                text: "Promijeni lozinku"
                on_release:
                    app.root.current = "Forgot"
                    root.manager.transition.direction = "left"
```

va istoimenu klasu. Vidimo da će biti vertikalne orijentacije sa jednim *Label* elementom kao naslovom. Onda nastupa *GridLayout* koji se organizira po stupcima, vidimo *cols* atribut, i onda se elementi pune redom od lijeve prema desnoj strani. Ovdje idu podaci o korisniku, vidimo samo za Ime korisnika, ali ostali podaci su analogni tome i imaju svoje metode implementirane u klasi. Na kraju imamo 2 *Button-a*, pomoću jednog se odjavljujemo, a pomoću drugog otvaramo formu za promjenu lozinke.

```
class Profile(Screen):
    user = ""
    def osvjezi(self):
        f = open("users.txt", "r")
        for line in f:
            Profile.user = line.split(";")
            if Profile.user[2] == Login.logged:
                print(Profile.user)
                break
```

```
        f.close()
    def getName(self):
        return Profile.user[0]
```

Ovako izgleda logički dio za profil, dohvaćamo pomoću trenutnog korisnika podatke iz datoteke i pomoću *getName()* metode možemo te atribute dohvatiti u Kivy.kv datoteke. Možemo vidjeti kako izgleda korisnički profil na slici:



Slika 16. Profile screen – Kivy [Autorski rad]

5.2.4.WxPython

WxPython je *cross-platform* GUI set alata za Python jezik. Omogućava programerima da stvaraju aplikacije sa robusnim, funkcionalnim grafičkim sučeljima. Implementiran je oko GUI elemenata popularne wxWidgets biblioteke napisana u C++ jeziku. WxPython je *open-source* i slobodan svima na korištenje. Trenutno podržane platforme su Windows, Mac OS X, Linux. U većini slučajeva na svakoj platformi koriste se *native widgets*, odnosno 'prirodni' *widgeti* za svaku platformu.

WxPython stvorio je Robin Dunn kada mu je trebalo grafičko korisničko sučelje za HP-UX sustave u 1984. godine. Tada je naletio na wxWidget alate, pa je naučio Python i u kratkom

vremenu postao jedan od glavnih developera wxPython okvira. Prva moderna verzija najavljena je 1998. godine.

Jako je jednostavno krenuti s WxPython, sve što je potrebno za aplikaciju je uvesti *wx* paket, stvoriti aplikacijski objekt *App()*. Zatim stvoriti okvir, odnosno *Frame(),* prikazati ga metodom *show()* i pokrenuti aplikaciju, odnosno glavnu petlju metodom *MainLoop().*

Opet je moguće pozicionirati elemente apsolutno, no to ne želimo i to možemo izbjeći *Sizer widget-ima* , od kojih postoje *BoxSizer, StaticBoxSizer, GridSizer, FlexGridSizer, GridBagSizer*. Mi ćemo se zadovoljiti *BoxSizer-om* koji nam omogućava da slažemo elemente u stupac ili u red, ovisno koju orijentaciju postavimo.[20]

Od aplikacije je ostao dio sa mijenjanjem lozinke pa ćemo tu funkcionalnost prikazati pomoću ovog okvira. Ovaj put možemo pretpostaviti da je korisnik odmah prijavljen u aplikaciju. Dodajemo si potrebne pakete, pomoću *super* osiguramo da se pozvao konstruktor s argumentima. *Panel* će označavati mjesto koje će popuniti *mainLayout*, odnosno *BoxSizer*.

```
import wx
import os
class HomePage(wx.Frame):
      logged = True ### inače je False, ali pretpostavimo da je True
      logout = False
      user = [„robert“, „sudec“, „rsudec“, „rsudec@foi.hr“, „Hrvatska“,
„d“]
      def __init__(self, *args, **kw):
            super(HomePage, self).__init__(*args, **kw, size=(600,600))
            self.panel = wx.Panel(self)
            self.mainLayout = wx.BoxSizer(wx.VERTICAL)
            self.setup()
      def setup(self):
            if HomePage.logged:
                  self.clean()
                  self.showProfile()
```

Metoda *setup()* dalje čisti *panel* i prikazuje profil svojom metodom. Metoda *showProfile()* definira *mainLayout* kao vertikalni *BoxSizer,* te puni taj layout svojim elementima. Prvi element koji se pojavljuje će biti *StaticText*, odnosno naslov. Taj *widget* uzima panel na kojem će se pojaviti, svoj ID, tekst koji će pisati, i stilove. Naknadno možemo postaviti font tom objektu metodom *SetFont()* koja uzima objekt tipa *Font* kao argument.

```
      def showProfile(self):
            self.mainLayout = wx.BoxSizer(wx.VERTICAL)
            self.welcome = wx.StaticText(self.panel, -1, "Dobrodošao na
svoj profil", style=wx.ALIGN_CENTER)
            font = self.welcome.GetFont()
            font.PointSize += 15
            font = font.Bold()
            self.welcome.SetFont(font)

            self.lblFont = wx.StaticText(self.panel, -1, "").GetFont()
```

```
            self.lblFont.PointSize += 7
            self.txtFont = self.lblFont.Bold()

            self.cntnIme = wx.BoxSizer(wx.HORIZONTAL)
            self.lblIme = wx.StaticText(self.panel, -1, "Ime: ",
style=wx.ALIGN_CENTER)
            self.lblIme.SetFont(self.lblFont)
            self.Ime = wx.StaticText(self.panel, -1, HomePage.user[0],
style=wx.ALIGN_CENTER)
            self.Ime.SetFont(self.txtFont)
            self.cntnIme.Add(self.lblIme, 1, wx.EXPAND | wx.ALIGN_LEFT |
wx.ALL, 5)
            self.cntnIme.Add(self.Ime, 1, wx.EXPAND | wx.ALIGN_LEFT |
wx.ALL, 5)
```

Zatim osiguravam da je *lblFont* 7pt veći od *default*-nog, te da je *txtFont* iste te veličine, samo podebljani, *bold. LblFont* koristimo za *label* tipa „Ime“, „Prezime“, a *txtFont* koristim za *label* koji zapravo sadrže podatke korisnika.

Sada mi trebaju dva *StaticText* elementa horizontalno i stoga ih moram omotati dodatnim *container-om,* odnosno dodatnim *BoxSizer-om*. Dodajem *StaticText-ove* dok jednome pridružujem podatak o korisniku preko varijable klase, odnosno varijabla koju dijele sve instance jedne klase, *user*. Nakon toga dodajem ih u *cntnIme* pomoću *Add()* metode. Malo zbunjujeć je drugi argument, odnosno tzv. *proporcija*. Ona označava smije li element-dijete promijeniti veličinu u smjeru orijentacije samog *BoxSizer-a.* Broj 0 označava da se veličina ne mijenja, dok se brojevi veći od 0 interpretiraju relativno sa elementima-braćom. Analogno ovome napisani su ostali podaci na korisničkom profilu, pa tako i *Button* elementi.

```
            self.cntnButtons = wx.BoxSizer(wx.HORIZONTAL)
            self.btnForgot = wx.Button(self.panel, -1, "Promijeni lozinku")
            self.btnForgot.Bind(wx.EVT_BUTTON, self.btnForgotClick)
            self.btnLogout = wx.Button(self.panel, -1, "Odjava")
            self.btnLogout.Bind(wx.EVT_BUTTON, self.btnLogoutClick)
            self.cntnButtons.Add(self.btnForgot, 1, wx.EXPAND)
            self.cntnButtons.Add(self.btnLogout, 1, wx.EXPAND)
```

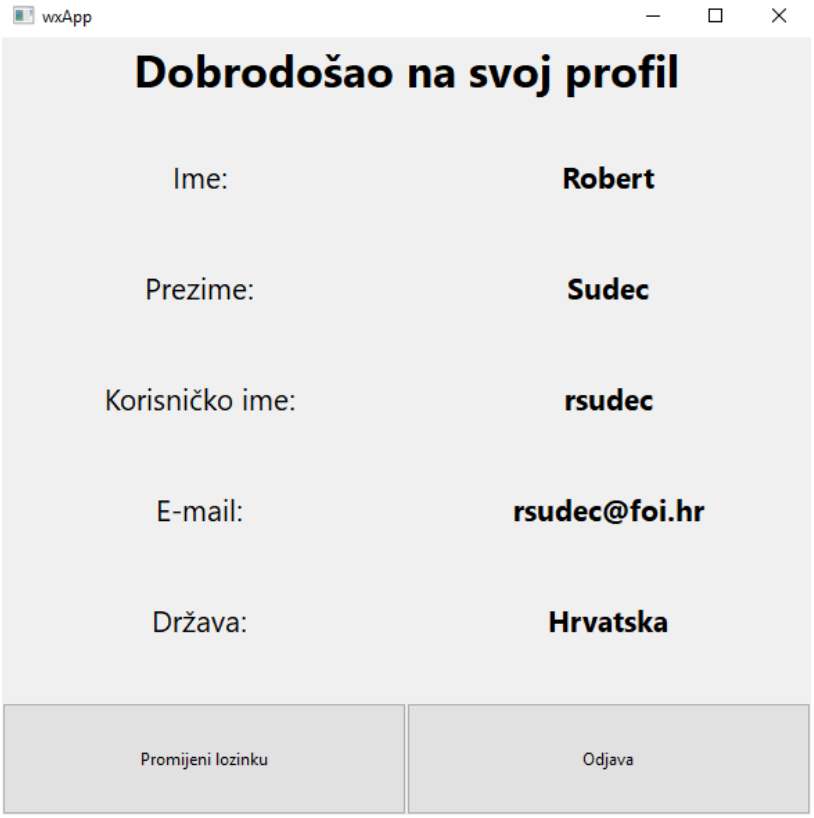
Button elementi ovdje se ponašaju kao i ostali *widgeti*, uzimaju roditeljski element, ID, i tekst u argumente, dok je sada bitan dio o vezanju metode na određeni događaj, odnosno klik na *button.* To se odvija pomoću metode *Bind()* kojoj dajemo predefinirani događaj *EVT_BUTTON* i metodu koju pozivamo *btnForgotClick().*

Kako bi dovršili izgradnju sučelja, moramo još sve *container* elemente dodani na *mainLayout*.

```
            self.mainLayout.Add(self.welcome, 1, wx.EXPAND)
            self.mainLayout.Add(self.cntnIme, 1, wx.EXPAND)
            self.mainLayout.Add(self.cntnButtons, 1 , wx.EXPAND)
            self.panel.SetSizer(self.mainLayout)
            self.panel.Layout()
```

Kada je sve dodano u najveći *Sizer*, odnosno u ovom slučaju, *mainLayout.* Još je potrebno postaviti taj *layout* na naš glavni *panel*, te pozvati metodu *Layout()* koja ponovno postavlja elemente na *panel.*.

Naš profil sada izgleda ovako:



Slika 17. Profile prozor – wxPython [Autorski rad]

Pritiskom na *Promijeni lozinku* otvara se novi prozor *ForgotScreen*

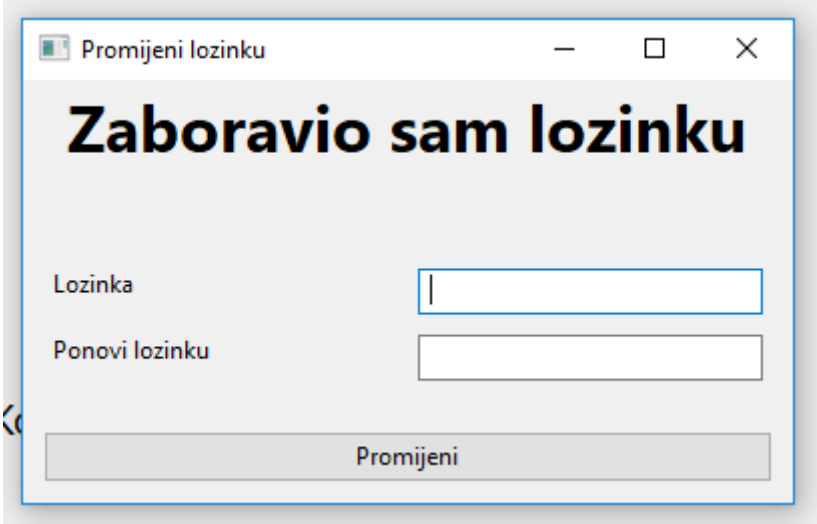
```
def btnForgotClick(self, event):
      frm = ForgotScreen(None, title="Promijeni lozinku")
      frm.Show()
```

Otvaranjem novih prozora moguće je vratiti se u prošli prozor i recimo otvoriti 10 prozora *ForgotScreen*, no to ne želimo i moramo ih napraviti modalnim i ručno povezati *EVT_CLOSE* događaj (to je kad se stisne X) sa metodom koja će vratiti prozor u „nemodalni“ i ugasiti prozor.

```
class ForgotScreen(wx.Frame):
      def __init__(self, *args, **kw):
            super(ForgotScreen, self).__init__(*args, **kw)
            self.MakeModal()
            self.Bind(wx.EVT_CLOSE, self.exit)
            self.panel = wx.Panel(self)
            self.mainLayout = wx.BoxSizer(wx.VERTICAL)
            self.forgotLayout = wx.BoxSizer(wx.VERTICAL)
            self.passLayout = wx.BoxSizer(wx.HORIZONTAL)
            self.lblPass = wx.StaticText(self.panel, -1, label="Lozinka")
```

```
        self.txtPass = wx.TextCtrl(self.panel, style=wx.TE_PASSWORD)
        self.passLayout.Add(self.lblPass, 1, wx.ALIGN_LEFT | wx.ALL, 5)
        self.passLayout.Add(self.txtPass, 1, wx.ALIGN_LEFT | wx.ALL, 5)
        self.forgotLayout.Add(self.passLayout, 0, wx.EXPAND)
        self.forgotBtn = wx.Button(self.panel, -1, "Promijeni")
        self.forgotBtn.Bind(wx.EVT_BUTTON, self.btnForgotClick)
        self.mainLayout.Add(self.forgotLayout, 1, wx.EXPAND|wx.ALL, 10)
        self.mainLayout.Add(self.forgotBtn, 0, wx.EXPAND|wx.ALL, 10)
        self.panel.SetSizer(self.mainLayout)
```

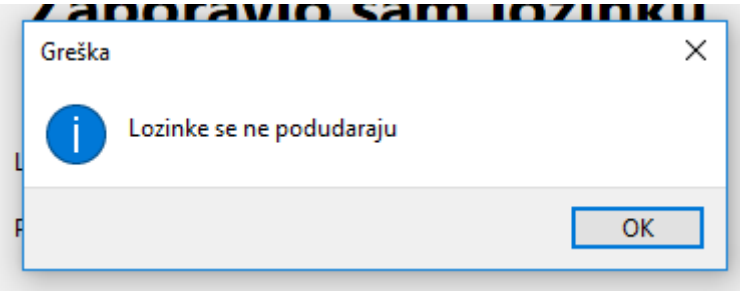
Ovdje vidimo sučelje za prozor „Promijeni lozinku“ i opet nisu svi elementi ovdje. Želim istaknuti polje za unos teksta *txtPass* koji je zapravo objekt tipa *TextCtrl(),* te koji uzima kao argument stil *wx.TE_PASSWORD* kako bi se umjesto lozinke prikazali znakovi „*“. Ostatak sučelja smo već vidjeli pa će taj prozor izgledati ovako:



Slika 18. "ForgotScreen" - wxPython [Autorski rad]

Još nedostaje logički dio mijenjanja lozinke. Izmjenu, odnosno ažuriranje teksta u datoteci obavit ćemo prepisivanjem dijela koji se ne mijenja u novu datoteku, kada dođemo do dijela koji se ažurira, zapišemo novi tekst, te nastavimo prepisivati ostatak iz stare datoteke u novu. Na kraju obrišemo staru datoteku, a novu datoteku preimenujemo u ime stare datoteke. To će se odraditi nakon što aplikacija provjeri je li korisnik uopće unio i je li unio jednake lozinke.

```
    def btnForgotClick(self, event):
        self.password = self.txtPass.GetValue()
        self.password1 = self.txtrePass.GetValue()
        if not self.password or not self.password1:
            wx.MessageDialog(self.panel, "Popuni sva polja",
caption="Greška", style=wx.OK | wx.STAY_ON_TOP).ShowModal()
```



Slika 19. MessageDialog - wxPython [Autorski rad]

```
else:
      f = open("users.txt", "r")
      nf = open("users1.txt", "a")
      for line in f:
            user = line.split(";")
            if user[2] == HomePage.user[2]:
                  user[5] = self.password
                  nf.writelines(user[0] + ";" + user[1] + ";" +
                  user[2] + ";" + user[3] + ";" + user[4] + ";"
                  + user[5] + ";" + "\n")
      nf.close()
      f.close()
      os.remove("users.txt")
      os.rename("users1.txt", "users.txt")
      self.MakeModal(modal=False)
      self.Close()
```

To je sada bila opisana funkcionalnost zamijene lozinke. Još nam, kao i kod svakog okvira do sad ostaje da pokrenemo aplikaciju. To se odrađuje na način da instanciramo objekt *App* kao korijen, te stvorimo objekt jedne od klasa koje nasljeđuju *wx.Frame* kako bi im mogli pozvati metodu *Show()* te pokrenemo glavnu petlju nad korijenom aplikacije,.

```
app = wx.App()
frm = HomePage(None, title='wxApp')
frm.Show()
app.MainLoop()
```

5.2.5. PySimpleGUI

Dosad smo pokrili 4 najveća, najkorištenija paketa/okvira za razvoj GUI aplikaciju s Python-om, tkinter, PyQt, Kivy, wxPython. Uz njih postoje i paketi koji pojednostavljuju prethodno nabrojane okvire, te im uz to smanjuju raspon mogućnosti. Početnici će se mučiti sa nekim od većih paketa, zato se upuštaju u jednostavnije pakete koji im ne mogu ponuditi izradu prilagođenog (eng. *custom)* sučelja. Zato postoji PySimpleGUI, koji nastoji riješiti navedene probleme tako što pruža jednostavno, lako-razumljivo sučelje za izradu prilagodljivog GUI. [21]

Sa nekim sučeljima slaganje *widgeta* često zahtijeva nekoliko linija koda, dok PySimpleGUI koristi tzv. *auto-packer* koji automatski definira raspored elemenata. Ovdje ne postoje koncepti za koje smo čuli prethodno, tipa *pack(), grid(), place(), QBoxLayout()* i sl.

PySimpleGUI se pokreće na Windows, Linux i Mac sustavima, tj. ako možete pokrenuti paket kojeg PySimpleGUI koristi, možete i njega. Za sada je Tkinter podržan u cijelosti, dok se *wrapper* za Qt i Wx još razvija. A sve što trebate učiniti je pomoću upravitelja paketa instalirati PySimpleGUI:

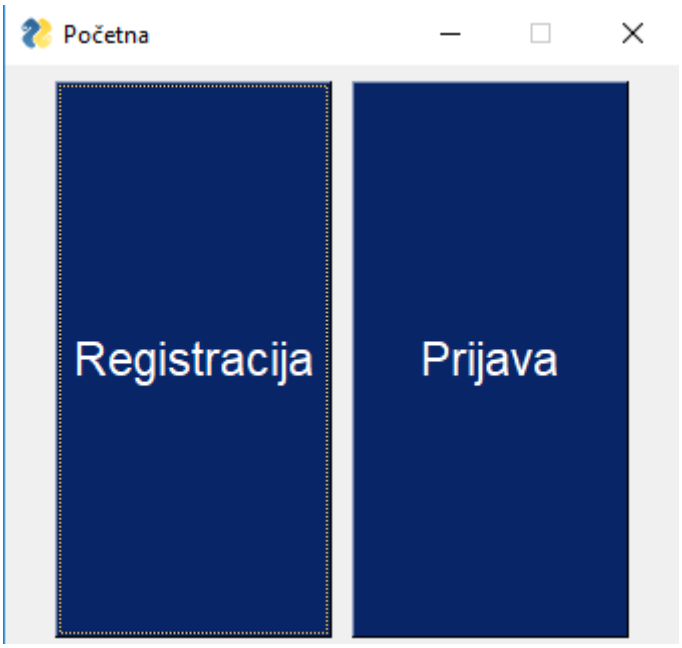
```
pip install --upgrade PySimpleGUI
```

I to nakon što ste osigurali da imate instaliran paket Tkinter.[22]

Kroz prethodna 4 okvira prošli smo kroz cijelu aplikaciju tako da ćemo sada ponovno prikazivati „Home“ i „Register“ prozore. Počnimo sa pozivanjem, *import*, potrebnog paketa, te definiranjem klase početnog prozora koji se sastoji od 2 *Buttona*.

```
import PySimpleGUI as sg
import os
class HomeWindow():
      def __init__(self):
            self.layout = [
                  [sg.Button("Registracija",  size=(10,10),  font=("Arial",
                        17)),sg.Button("Prijava", size=(10,10),
                        font=("Arial", 17))]
            ]
            self.win = sg.Window("Početna", self.layout)
            self.show()
```

Pri inicijalizaciji objekta definirat ćemo raspored svakog prozora na način da definiramo *layout* kao dvo-dimenzionalnu listu (polje). Možemo reći da ta lista predstavlja rešetku, *grid* i da svaki element te liste čini jedan red rasporeda prozora. Na početnoj stranici vidimo da imamo samo jedan element, te ćemo zato imati jedan red. Analogno, prvi element liste je lista sa 2 elementa, na taj način definiramo stupce (pojedinačno za svaki red). Takav *layout* će sadržavati dva *Buttona* postavljena horizontalno jedan do drugoga. Naravno, usput definiramo i atribute *widgeta*, tipa tekst koji piše na elementu, veličinu, font i još mnogo njih. *Window* je objekt pomoću kojeg pokrećemo prozor i njemu prenosimo naslov prozora i raspored, *layout* koji će prozor prikazati.



Slika 20. "Home" prozor – PySimpleGUI

Prozor je pokrenut, ali nije prikazan. Pogledajmo što radi metoda *show().*

```
def show(self):
	while True:
		self.dogadaj, self.values = self.win.Read(timeout=100)
		if(self.dogadaj == "Registracija"):
			self.win.Hide()
			RegWindow(self.win)
		if(self.dogadaj == "Prijava"):
			self.win.Hide()
			LoginWindow(self.win)
```

Ulazimo u beskonačnu petlju, te pomoću metode *Read()* objekta Window zapravo omogućujemo komunikaciju korisnika i aplikacije, odnosno ona nam svakih 100ms očitava pokrenute događaje i sve vrijednosti unesene u sučelje (atribut *timeout*). Ostatak koda ovisi na koji *Button* kliknemo *Registracija* ili *Prijava.* Ovaj prozor sakrivamo pomoću *Hide()* metode, te proslijeđujemo trenutni *Window* objekt u nove prozore.[23]

Početni prozor će ostati u pozadini sakriven kako bi korisnik mogao aktivno komunicirati samo s jednim prozorom odjednom, te prikazujemo prozor „Registracija“ koji je implementiran na isti način, samo s malo više programske logike.

```
class RegWindow():
	def __init__(self, parent):
		self.parent = parent
		self.layout = [
			[sg.Text("Registracija")],
			[sg.Text("Ime: "), sg.Input()],
			[sg.Text("Prezime: "), sg.Input()],
			[sg.Text("Korisničko ime: "), sg.Input()],
			[sg.Text("E-mail: "), sg.Input()],
```

```
                  [sg.Text("Država:",sg.DropDown(["Hrvatska","Srbija",
"Bih"])],
                  [sg.Text("Lozinka: ", sg.Input(password_char='*')],
                  [sg.Text("Potvrdi lozinku:", sg.Input(password_char='*')],
                  [sg.Button("Registriraj se")]
            ]
            self.win = sg.Window("Registracija", self.layout)
            self.show()
```

Ovdje se bolje vidi sustav raspoređivanja, vidimo da prvi i zadnji elementi (redovi) imaju po jedan element u sebi, naslov „Registracija“ i *Button* „Registriraj se“. Elementi između imaju opisni *Text* i element tekstualnog unosa, ili pak *DropDown* element za odabir. Kod *DropDown* objekta bitno je navesti polje mogućih odabira, a kod unošenja lozinke dobro je staviti znak koji će se prikazivati u *Input* elementu pomoću atributa *password_char*. Spomenut ću samo da smo roditeljski prozor dobili preko konstruktora i trebat će nam kasnije u metodi *show().*

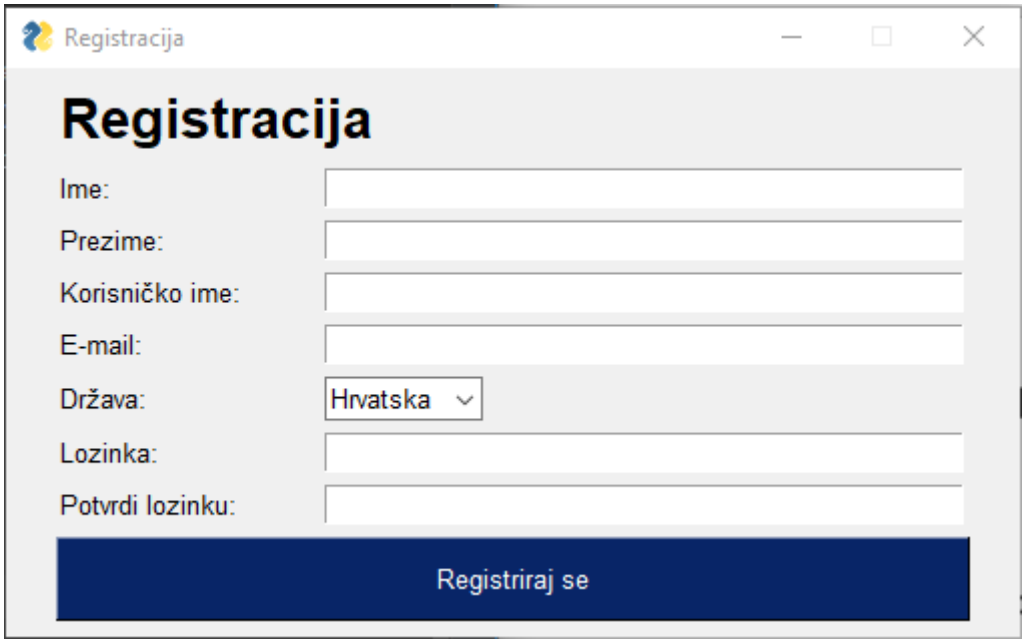
Na ovom prozoru trebat će nam više logike, stoga ćemo imati koju metodu više. Jedna od tih metoda je *checkValues()* pomoću koje provjeravamo jesu li u polja unesene ispravne vrijednosti, te metoda *registerUser()* koja se bavi zapisivanjem u datoteku.

```
def show(self):
      while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Registriraj se"):
                  if self.checkValues(self.values) == 0:
                        sg.Popup("Unesite sve vrijednosti")
                  elif self.checkValues(self.values) == -1:
                        sg.Popup("Lozinke se ne podudaraju")
                  else:
                        self.registerUser(self.values)
                        self.win.Close()
                        self.parent.UnHide()
                        break
def checkValues(self, values):
      for val in values:
            if not values[val]:
                  return 0
      if values[5] != values[6]:
            return -1
      return 1
def registerUser(self, values):
      f = open("users.txt", 'a')
      f.writelines(values[0] + ";" + values[1] + ";" + values[2] + ";" +
            values[3] + ";" + values[4] + ";" + values[5] + ";" + "\n")
      f.close()
```

Dakle, svakih 100ms čitamo pokrenute događaje, a bitan nam je događaj po imenu „Registriraj se“. Događaj dobije ime po tekstu koji piše za *Button* element. Klikom na taj *Button* ulazimo u selekciju i provjeravamo podatke koje smo dobili, također, metodom *Read()* i koji su spremljeni u *values* polju. Tamo će se naći vrijednosti iz svih *„input“* elemenata, i to redom kojim su dodani na *layout*. Ako je ijedan od tih elemenata prazan, metoda *checkValues()* vraća 0 te javlja

korisniku poruku pomoću iskočnog prozora – *Popup()* objekta. Isto to radi i ako su polja *lozinka* i *potvrdi lozinku* različitog sadržaja. Korisnik se može registrirati kada su podaci ispravni, te se zatim trenutni prozor „Registracija“ zatvara metodom *Close()*, pomoću objekta kojeg smo prenijeli konstruktorom pristupamo prozoru „Početna“ i ponovno ga prikazujemo metodom *UnHide()* i iskačemo iz beskonačne petlje ključnom riječju *break*.

Ovih skromnih 50-ak linija koda omogućava nam u potpunosti funkcionalan, organiziran prozor pun *widgeta* koji izgleda ovako:



Slika 21. "Register" prozor – PySimpleGUI

6. Usporedba

Usporediti ove razvojne okvire nije lako. Svaki od njih imao je svoje prednosti i nedostatke. Tkinter je bio jako jednostavan za rad, došao uključen s instalacijom Pythona. No smatram da nije imao veliki broj *widget*-a i da nije toliko opsežan. No opsežnost bi ga i zakomplicirala.

Iako je tkinter jednostavan, više mi se svidio sustav pozicioniranja kod ostalih okvira, a to je pretežno *BoxLayout*. Istina je da se mora vezati prozor za *layout*, pa *widgeti* za taj *layout,* i toga sam se uplašio kada sam došao do slaganja sučelja sa PyQt, no kada sam ušao u tu kolotečinu, uopće nije problem raditi s takvim sustavom. Ima malo više pisanja, ali isplati se. Veliki plus je što PyQt, za razliku od ostalih koji imaju svoje čudne atribute za uređivanje elemenata, podržava stilske listove, odnosno CSS3. To znači da tko je god bacio oko na CSS, na one najosnovnije atribute, znao bi uređivati *widgete* u PyQT.

Mislim da je Kivy bio jedan od težih, a opet jedan od boljih okvira. Najveći minus je to što da bi koristio Kivy moraš naučiti 2 sintakse, jednu za pisanje koda u Pythonu u sklopu Kivy-a, a drugo *Kv* datoteke kojima se opisuje sučelje. Plus nasuprot ovom minusu je da je odlična stvar odvojiti logiku vlastite aplikacije od prezentacije, tj. sučelja. Veća je čitljivost koda, brže uređivanje i sučelja i logike. Čudno je to što dosad niti jedan okvir nije po *default-u* raširivao *widget-e* preko cijelog ekrana, osim Kivy okvira, tu se elementi moraju smanjivati. Još jedna pozitivna stvar je što postoji jedan fizički *window*, a izmjenjuju se *screen-ovi*. To mu daje moderan štih jer smo navikli na web i mobilne aplikacije koje se izvode u jednom fizičkom prozoru.

Na red je došao wxPython i već sam mislio da ću u njemu aplikaciju napraviti bez problema jer se na prvi pogled činio vrlo sličan svim ostalima. Tako je i bilo, jednostavno za korištenje, osim definiranih stilova, za koje sam već rekao da mogu biti jako čudni. Najviše mi je smetao atribut *proporcija* kad dodavanja elemenata na *BoxSizer.* Nedostatak mi je bio što se prozori nisu otvarali kao modalni, pa sam morao ručno ih pretvarati iz nemodalnih u modalne i obrnuto, a to je povuklo i ručno povezivanje *Close* tipke sa vlastitom funkcijom.

Zadnji, ali ne i najmanje važan, bio je PySimpleGUI paket. On sam po sebi nije poseban okvir, nego *wrapper* nekog drugog okvira, u ovom slučaju je to Tkinter. Uglavnom takvi paketi pojednostavljuju izradu aplikacija, te je ovaj paket to vrhunski odradio. Moram priznati da koristi drugačije principe rada nego preostala 4 okvira, ali puno jednostavnije je za početnika snaći se u ovakvom paketu. Mogu se koristiti svi *widgeti* kao i u ostalim okvirima, ali smatram da nije toliko moćan, jer je prejednostavan i ne bi bio dobar za složenija sučelja. Trebalo mi je 5 minuta da napravim funkcionalan „Home“ prozor, tako da ovaj paket preporučam svim početnicima.

7. Zaključak

Izradio sam jako jednostavnu aplikaciju sa grafičkim sučeljem u 4 razvojna okvira u jeziku Python. Meni je to u početku stvaralo problem jer nisam imao previše iskustva s Pythonom, a izrađivao sam GUI aplikacije u Microsoft Visual Studio IDE u C#, što nije ni blizu ovome što me dočekalo. No ubrzo sam se snašao i izradio aplikaciju u Tkinter, PyQt, Kivy, wxPython i PySimpleGUI razvojnim okvirima.

Za one koji žele napraviti najjednostavnije aplikacije, mogu predložiti PySimpleGUI kao jedan od najjednostavnijih paketa, ali i s najmanje mogućnosti. Brzo ga se može naučiti iako nemate iskustva s Pythonom i ovim područjem kao što sam to bio ja.

Meni je sigurno najdraži bio Kivy, no zahtijevao je više truda i vremena. Zasigurno je estetski najljepši, moglo bi se reći da postoji i moderan UX (User Experience) jer odmah postoje animacije i lijepi elementi.

Mnogo sam toga naučio u vremenu pisanja ovog završnog rada. Najbolje je to što sam se „sprijateljio“ s Python-om, upraviteljem paketa, njegovim razvojnim okruženjem, te to što sam shvatio da koliko god nešto nepoznato izgledalo i susrećete se s tim prvi put, nije problem to savladati uz uloženo vrijeme i trud.

Popis literature

[1] D. Kuhlman, *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.
[2] B. Hookway, “Chapter 1: The Subject of the Interface,” *B Hookway Interface*, pp. 1–58, 2014.
[3] ZxxZxxZ, *Screenshot of a sample Bash session in GNOME Terminal 3, Fedora 15*. 2011.
[4] “What is CLI.” [Online]. Available: https://www.w3schools.com/whatis/whatis_cli.asp. [Accessed: 21-Aug-2019].
[5] “Text User Interface Development Series: Part One - T.U.I. Basics.” [Online]. Available: http://www.petesqbsite.com/sections/express/issue21/tuiseriespart1.htm. [Accessed: 21-Aug-2019].
[6] *Some file managers implement a TUI (here: Midnight Commander)*. 2006.
[7] A. Powell, “Web 101: A History of the GUI,” *Wired*, 19-Dec-1997.
[8] *Apple Lisa Personal Computer*. .
[9] T. Friedman, *Electric Dreams: Computers in American Culture*. NYU Press, 2005.
[10] *Screenshot of Windows 95, taken shortly after installation*. 2017.
[11] Microsoft, *Windows 10*. .
[12] D. Riehle, “DOKTOR DER TECHNISCHEN WISSENSCHAFTEN (DOCTOR OF TECHNICAL SCIENCES),” p. 230.
[13] “Introduction to GUI programming with tkinter — Object-Oriented Programming in Python 1 documentation.” [Online]. Available: https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html. [Accessed: 21-Aug-2019].
[14] “Introduction to GUI programming with tkinter — Object-Oriented Programming in Python 1 documentation.” [Online]. Available: https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html#layout-options. [Accessed: 23-Aug-2019].
[15] “Introduction to GUI programming with tkinter — Object-Oriented Programming in Python 1 documentation.” [Online]. Available: https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html#widget-classes. [Accessed: 23-Aug-2019].
[16] M. Summerfield, *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Pearson Education, 2007.
[17] “PyQt5 tutorial 2019: Create a GUI with Python and Qt.” [Online]. Available: https://build-system.fman.io/pyqt5-tutorial. [Accessed: 21-Aug-2019].
[18] “Getting Started — Kivy 1.11.1 documentation.” [Online]. Available: https://kivy.org/doc/stable/gettingstarted/#. [Accessed: 21-Aug-2019].
[19] “Kv language — Kivy 1.11.1 documentation.” [Online]. Available: https://kivy.org/doc/stable/guide/lang.html. [Accessed: 21-Aug-2019].
[20] “wxPython - Quick Guide.” [Online]. Available: https://www.tutorialspoint.com/wxpython/wxpython_quick_guide.htm. [Accessed: 21-Aug-2019].
[21] “Tutorial - PySimpleGUI.” [Online]. Available: https://pysimplegui.readthedocs.io/en/latest/tutorial/#the-secret. [Accessed: 23-Aug-2019].
[22] “PySimpleGUI.” [Online]. Available: https://pysimplegui.readthedocs.io/en/latest/#installing-pysimplegui. [Accessed: 23-Aug-2019].
[23] “Tutorial - PySimpleGUI.” [Online]. Available: https://pysimplegui.readthedocs.io/en/latest/tutorial/#display-gui-get-results. [Accessed: 23-Aug-2019].

Popis slika

Prilozi (KivyApp.py, Kivy.kv, PySimpleGUI.py)

1. **Izvorni kod Kivy aplikacije i pripadajuća Kivy.kv:**

```
from kivy.app import App
from kivy.uix.label import *
from kivy.uix.button import *
from kivy.uix.boxlayout import *
from kivy.uix.popup import Popup
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen
import os

class Home(Screen):
    pass
class Register(Screen):
    def registrirajKorisnika(self, ime, prezime, user, email, drzava,
lozinka, relozinka ):
        if not ime or not prezime or not user or not email or drzava ==
"Odaberi..." or not lozinka or not relozinka:
            return 0
        elif lozinka != relozinka:
            return -1
        else:
            self.f = open("users.txt", 'a')
            self.f.writelines(
                ime + ";" + prezime + ";" + user + ";" + email + ";" +
drzava + ";" + lozinka + ";" + "\n")
            self.f.close()
            return 1
    pass
class Login(Screen):
    logged = ""
    def prijaviKorisnika(self, user, lozinka):
        if not user or not lozinka:
            return 0
        else:
            self.postoji = False
            f = open("users.txt", 'r')
            for line in f:
                korisnik = line.split(";")
                if korisnik[2] == user and korisnik[5] == lozinka:
                    Login.logged = korisnik[2]
                    self.postoji = True
                    break
            f.close()
            if self.postoji:
                layout = BoxLayout(orientation="vertical")
                poruka = Label(text='Prijava uspješna')
                closeButton = Button(text="Ok")
                layout.add_widget(poruka)
                layout.add_widget(closeButton)
                popup = Popup(title='Demo Popup', content=layout)
                #content=(Label(text='This is a demo pop-up')))
                popup.open()
                closeButton.bind(on_press=popup.dismiss)
                return 1
            else:
                return -1
```

```
    pass
class Profile(Screen):
    user = ""
    def osvjezi(self):
        f = open("users.txt", "r")
        for line in f:
            Profile.user = line.split(";")
            if Profile.user[2] == Login.logged:
                print(Profile.user)
                break
        f.close()
    def getName(self):
        return Profile.user[0]
    def getSurname(self):
        return Profile.user[1]
    def getUsername(self):
        return Profile.user[2]
    def getEmail(self):
        return Profile.user[3]
    def getCountry(self):
        return Profile.user[4]
    pass
class Forgot(Screen):
    def provjeraLozinke(self, pass1, pass2):
        if not pass1 or not pass2:
            return -1
        elif pass1 != pass2:
            return 0
        else:
            return 1
    def promijeniLozinku(self, password):
        f = open("users.txt", "r")
        nf = open("users1.txt", "a")

        for line in f:
            self.korisnik = line.split(";")
            if self.korisnik[2] == Login.logged:
                self.korisnik[5] = password
            nf.writelines(
                self.korisnik[0] + ";" + self.korisnik[1] + ";" +
self.korisnik[2] + ";" + self.korisnik[3] + ";" + self.korisnik[4] + ";" +
self.korisnik[5] + ";" + "\n")
        nf.close()
        f.close()
        os.remove("users.txt")
        os.rename("users1.txt", "users.txt")
    pass
class WindowManager(ScreenManager):
    pass


kv = Builder.load_file("Kivy.kv")
class KivyApp(App):
    def build(self):
        return kv

KivyApp().run()



WindowManager:
```

```
    Home:
    Register:
    Login:
    Profile:
    Forgot:

<Home>:
    name: "Home"
    BoxLayout:
        orientation: "horizontal"
        padding:20
        Button:
            text: "Registracija"
            on_press:
                app.root.current = "Register"
                root.manager.transition.direction = "right"
        Button:
            text: "Prijava"
            on_press:
                app.root.current = "Login"
                root.manager.transition.direction = "left"

<Register>:
    name: "Register"

    BoxLayout:
        orientation: "vertical"
        Label:
            text: "Registracija"
            font_size: '20pt'
        GridLayout:
            cols: 2
            Label:
                text: "Ime"
            TextInput:
                id: tboxIme
            Label:
                text: "Prezime"
            TextInput:
                id: tboxPrezime
            Label:
                text: "Korisnicko ime"
            TextInput:
                id: tboxUsername
            Label:
                text: "Email"
            TextInput:
                id: tboxEmail
            Label:
                text: "Drzava"
            Spinner:
                id: cboxCountry
                text: "Odaberi..."
                values: ["Hrvatska","Srbija","Bosna"]
            Label:
                text: "Lozinka"
            TextInput:
                id: tboxPassword
                password: True
            Label:
                text: "Potvrdi lozinku"
```

```
            TextInput:
                id: tboxRePassword
                password: True
            Button:
                text: "Pocetna"
                on_release:
                    app.root.current = "Home"
                    root.manager.transition.direction = "left"

            Button:
                id: btnRegister
                text: "Registracija"
                on_release:
                    provjera = root.registrirajKorisnika(tboxIme.text,
tboxPrezime.text, tboxUsername.text, tboxEmail.text, cboxCountry.text,
tboxPassword.text, tboxRePassword.text)
                    btnRegister.text =  "Popuni sva polja" if provjera == 0
else "Lozinke ne odgovaraju" if provjera == -1 else "Registracija"
                    app.root.current = "Home" if provjera == 1 else
"Register"
                    root.manager.transition.direction = "left"

<Login>:
    name: "Login"

    BoxLayout:
        orientation: "vertical"
        Label:
            text: "Prijava"
            font_size: '20pt'
        GridLayout:
            cols: 2
            Label:
                text: "Korisnicko ime"
            TextInput:
                id: tboxIme
            Label:
                text: "Lozinka"
            TextInput:
                id: tboxPassword
                password: True
                multiline: False
            Button:
                text: "Pocetna"
                on_release:
                    app.root.current = "Home"
                    root.manager.transition.direction = "right"
            Button:
                id: btnLogin
                text: "Prijava"
                on_release:
                    provjera = root.prijaviKorisnika(tboxIme.text,
tboxPassword.text)
                    btnLogin.text =  "Popuni sva polja" if provjera == 0
else "Ne postoji korisnik" if provjera == -1 else "Prijava"
                    app.root.current = "Profile" if provjera == 1 else
"Login"
                    root.manager.transition.direction = "right"
<Profile>
    name: "Profile"
    on_parent:
```

```
        user = root.osvjezi()
        tboxIme.text = root.getName()
        tboxPrezime.text = root.getSurname()
        tboxUser.text = root.getUsername()
        tboxEmail.text = root.getEmail()
        tboxDrzava.text = root.getCountry()
    BoxLayout:

        orientation: "vertical"
        Label:
            text: "Dobrodosao na svoj profil"
            font_size: "20pt"
        GridLayout:
            cols: 2
            Label:
                text: "Ime"
            Label:
                id: tboxIme
                text: ""
            Label:
                text: "Prezime"
            Label:
                id: tboxPrezime
                text: ""
            Label:
                text: "Korisnicko ime"
            Label:
                id: tboxUser
                text: ""
            Label:
                text: "Email"
            Label:
                id: tboxEmail
                text: ""
            Label:
                text: "Drzava"
            Label:
                id: tboxDrzava
                text: ""
            Button:
                text: "Odjavi se"
                on_release:
                    app.root.current = "Home"
                    root.manager.transition.direction = "right"
            Button:
                text: "Promijeni lozinku"
                on_release:
                    app.root.current = "Forgot"
                    root.manager.transition.direction = "left"
<Forgot>
    name: "Forgot"
    BoxLayout:
        orientation: "vertical"
        Label:
            text: "Promijeni lozinku"
            font_size: '20pt'
        GridLayout:
            cols: 2
            Label:
                text: "Nova lozinka"
            TextInput:
```

```
                id: tboxPass
                password: True
            Label:
                text: "Nova lozinka"
            TextInput:
                id: tboxRepass
                password: True
                multiline: False
            Button:
                text: "Profil"
                on_release:
                    app.root.current = "Profile"
                    root.manager.transition.direction = "right"
            Button:
                id: btnForgot
                text: "Promijeni"
                on_release:
                    provjera = root.provjeraLozinke(tboxPass.text,
tboxRepass.text)
                    btnForgot.text =  "Popuni sva polja" if provjera == -1
else "Lozinke se ne podudaraju" if provjera == 0 else "Promijeni"
                    root.promijeniLozinku(tboxPass.text) if provjera == 1
else root.provjeraLozinke("", "")
                    app.root.current = "Profile" if provjera == 1 else
"Forgot"
                    root.manager.transition.direction = "right"
```

2. Izvorni kod PySimpleGUI aplikacije:

```
import PySimpleGUI as sg
import os
class HomeWindow():
    def __init__(self):
        self.layout = [
            [sg.Button("Registracija", size=(10,10), font=("Arial", 17)),
sg.Button("Prijava", size=(10,10), font=("Arial", 17))]
        ]
        self.win = sg.Window("Početna", self.layout)
        self.show()
    def show(self):
        while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Registracija"):
                self.win.Hide()
                RegWindow(self.win)
            if(self.dogadaj == "Prijava"):
                self.win.Hide()
                LoginWindow(self.win)

class ProfileWindow():
    def __init__(self, parent, user):
        self.parent = parent
        self.user = user
        self.layout = [
            [sg.Text('Dobrodošli',  font=("Arial bold", 20))],
            [sg.Text('Ime: ', size=(25,1)), sg.Text(user[0], font=("arial
bold", 12))],
            [sg.Text('Prezime: ', size=(25,1)), sg.Text(user[1],
font=("arial bold", 12))],
            [sg.Text('Korisničko ime: ', size=(25,1)), sg.Text(user[2],
font=("arial bold", 12))],
```

```
            [sg.Text('E-mail', size=(25,1)), sg.Text(user[3], font=("arial
bold", 12))],
            [sg.Text('Država', size=(25,1)), sg.Text(user[4], font=("arial
bold", 12))],
            [sg.Button("Promijeni lozinku", size=(25,2)), sg.Button('Odjavi
se',  size=(25,2))],
        ]
        self.win = sg.Window("Zaboravio sam lozinku", self.layout)
        self.show()
    def show(self):
        while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Promijeni lozinku"):
                self.win.Hide()
                ForgotWindow(self.win, self.user)
            elif self.dogadaj == "Odjavi se":
                self.win.Close()
                self.parent.UnHide()
                break
class ForgotWindow():
    def __init__(self, parent, user):
        self.parent = parent
        self.user = user
        self.layout = [
            [sg.Text("Zaboravio sam lozinku", font=("Arial bold", 20))],
            [sg.Text("Lozinka: ",size=(15,1)),
sg.Input(password_char='*')],
            [sg.Text("Potvrdi lozinku: ",size=(15,1)),
sg.Input(password_char='*')],
            [sg.Button("Promijeni",size=(56,2))]
        ]
        self.win = sg.Window("Promijeni lozinku", self.layout)
        self.show()

    def show(self):
        while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Promijeni"):
                if self.checkValues(self.values) == 0:
                    sg.Popup("Unesite sve vrijednosti")
                elif self.checkValues(self.values) == -1:
                    sg.Popup("Lozinke nisu jednake")
                else:
                    self.promijeniLozinku(self.values)
                    self.win.Close()
                    self.parent.UnHide()
                    break
    def promijeniLozinku(self, values):
        f = open("users.txt", "r")
        nf = open("users1.txt", "a")

        for line in f:
            user = line.split(";")
            if user[2] == self.user[2]:
                user[5] = values[0]
            nf.writelines(user[0] + ";" + user[1] + ";" + user[2] + ";" +
user[3] + ";" + user[4] + ";" + user[
                5] + ";" + "\n")
        nf.close()
        f.close()
        os.remove("users.txt")
```

```
        os.rename("users1.txt", "users.txt")
    def checkValues(self, values):
        for val in values:
            if not values[val]:
                return 0
        if values[0] != values[1]:
            return -1
        return 1
class LoginWindow():
    def __init__(self, parent):
        self.parent = parent
        self.user = None
        self.layout = [
            [sg.Text("Prijava" , font=("Arial bold", 20))],
            [sg.Text("Korisničko ime: " ,size=(15,1)), sg.Input()],
            [sg.Text("Lozinka: ", size=(15,1)),
sg.Input(password_char='*')],
            [sg.Button("Prijavi se", size=(56,2))]
        ]
        self.win = sg.Window("Prijava", self.layout)
        self.show()

    def show(self):
        while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Prijavi se"):
                if self.checkValues(self.values) == 0:
                    sg.Popup("Unesite sve vrijednosti")
                elif self.checkValues(self.values) == -1:
                    sg.Popup("Nepostojeći korisnik")
                else:
                    self.win.Close()
                    ProfileWindow(self.parent, self.user)
                    break

    def checkValues(self, values):
        for val in values:
            if not values[val]:
                return 0
        f = open("users.txt", "r")
        for line in f:
            self.user = line.split(";")
            if self.user[2] == values[0] and self.user[5] == values[1]:
                return 1

        return -1
class RegWindow():
    def __init__(self, parent):
        self.parent = parent
        self.layout = [
            [sg.Text("Registracija", font=("Arial bold", 20))],
            [sg.Text("Ime: ", size=(15,1)), sg.Input()],
            [sg.Text("Prezime: ", size=(15,1)), sg.Input()],
            [sg.Text("Korisničko ime: ", size=(15,1)), sg.Input()],
            [sg.Text("E-mail: ", size=(15,1)), sg.Input()],
            [sg.Text("Država: ", size=(15,1)), sg.DropDown(["Hrvatska",
"Srbija", "Bih"])],
            [sg.Text("Lozinka: ", size=(15,1)), sg.Input()],
            [sg.Text("Potvrdi lozinku: ", size=(15,1)), sg.Input()],
            [sg.Button("Registriraj se", size=(56,2))]
        ]
```

```
        self.win = sg.Window("Registracija", self.layout)
        self.show()

    def show(self):
        while True:
            self.dogadaj, self.values = self.win.Read(timeout=100)
            if(self.dogadaj == "Registriraj se"):
                if self.checkValues(self.values) == 0:
                    sg.Popup("Unesite sve vrijednosti")
                elif self.checkValues(self.values) == -1:
                    sg.Popup("Lozinke se ne podudaraju")
                else:
                    self.registerUser(self.values)
                    self.win.Close()
                    self.parent.UnHide()
                    break

    def checkValues(self, values):
        for val in values:
            if not values[val]:
                return 0
        if values[5] != values[6]:
            return -1
        return 1
    def registerUser(self, values):
        f = open("users.txt", 'a')
        f.writelines(
            values[0] + ";" + values[1] + ";" + values[2] + ";" + values[3]
+ ";" + values[4] + ";" + values[5] + ";" + "\n")
        f.close()

app = HomeWindow()
```