Izrada multiplatformskih aplikacija s grafičkim korisničkim sučeljem

Bijelić, Josip

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Bijelić

IZRADA MULTIPLATFORMSKIH APLIKACIJE S GRAFIČKIM KORISNIČKIM SUČELJEM

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Bijelić

Matični broj: 43391/14–R

Studij: Informacijski sustavi

IZRADA MULTIPLATFORMSKIH APLIKACIJA S GRAFIČKIM KORISNIČKIM SUČELJEM

ZAVRŠNI RAD

Mentor:

Dr. sc. Miran Zlatović

Varaždin, siječanj 2018.

Josip Bijelić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski 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

Rad obuhvaća temu izrade multiplatformskih aplikacija s grafičkim korisničkim sučeljem u Python-u. Glavna tema je problematika pisanja portabilnog koda, usmjeravanje na bitne razlike između različitih operacijskih sustava kao što su Linux i Windows te problemi pri tranziciji programskog koda s jednog na drugi operacijski sustav. U radu su obuhvaćene općenite razlike između operacijskih sustava, procesora, mapiranja, sigurnosti i pristupačnosti određenim resursima operacijskog sustava. Obrađeni su problemi kod različitih datotečnih sustava i razlike u dinamičkim bibliotekama pojedinog operacijskog sustava.

Uz teoretski dio priložen je i praktični dio, tj. desktop aplikacija izrađena u Python programskom jeziku pomoću biblioteke PyQt5. Aplikacija obuhvaća domenu kupovine odjeće preko računala. Aplikacija je pokrenuta na Linux operacijskom sustavu(Ubuntu 16.04) i Windows operacijskom sustavu(Windows 10). Iz razloga što Python sam po sebi radi i kompatibilan je na skoro svim operacijskim sustavima, nije bilo problema kod tranzicije programskog rješenja s jednog operacijskog sustava na drugi.

Sadržaj

1. Uvod

Portabilnost programskog koda s jedne platforme(operacijskog sustava) na drugu danas predstavlja jedan od izazova u informatičkoj industriji. Ovaj rad pokriva sve bitne koncepte problematike portabilnosti programskog koda i kao rezultat usmjerava čitatelja na relevantnost i dobre strane pisanja takvog koda.

Pisanje portabilnog koda ima svoje dobre strane. Naime, isplativost pisanja portabilnog koda se očituje u tome da postoje mnoga tržišta koja zahtijevaju izradu multiplatformskih aplikacija. Iako sporiji i teži način pisanja koda, izrada multiplatformskih aplikacija danas mora uzimati u obzir portabilnost koda kako bi kompanije koje se time bave ostale kompetitivne na tržištu.

Zbog usporenog načina pisanja portabilnog programskog koda razvili su se alati koji omogućuju lakše razvijanje multiplatformskih GUI (Graphical User Interface) aplikacija. Takvi alati su danas popularni jer bitno ubrzavaju proces pisanja portabilnog koda. Alat koji će biti prezentiran u ovom radu je Qt. Točnije , kroz praktičan primjer izrade multiplatformske aplikacije u PyQt prikazat će se glavni problemi izrade multiplatformske aplikacije s grafičkim korisničkim sučeljem. Ovaj rad nije temeljen na pojašnjavanju prevođenja koda s jedne platforme u drugu jer je to mukotrpan i težak posao za bilo kojeg programera, već se temelji na samom razmišljanju od razvoja do testiranja aplikacije i pisanja koda koji je portabilan.Za lakše razumijevanje ove problematike u ovom radu je odabran razvoj desktop aplikacije koja obuhvaća domenu kupovanja odjeće preko računala.

2. Metode i tehnike pisanja završnog rada

Tema završnog rada obuhvaća pokrivanje teoretskog dijela problematike izrade multiplatformske aplikacije i praktičnog dijela izrade same aplikacije u programskom jeziku Python.

Pri izradi teoretskog dijela završnog rada koristio sam literaturu koju sam pronašao na internetu pomoću Google Chrome preglednika. Cijeli tekstualni dio pisao sam pomoću programa Microsoft Word 2016 po predlošku za pisanje završnog i diplomskog rada.

Kao što sam ranije naveo za izradu aplikacije s grafičkim korisničkim sučeljem koristio sam Python programski jezik i PyQt biblioteku. Pri izradi koristio sam sljedeće verzije programskih alata i jezika:

- Python 3.6.2.
- PyQt5
- Qt Designer – programski alat za izradu grafičkog sučelja aplikacija
- PyCharm Community Edition 2017.2.2. – editor za pisanje programskog koda
- MySQL Workbench.X 6.3. CE – programski alat za izradu baze podataka

Primjeri izrade grafičkog sučelja i korištenja programskog alata Qt Designer detaljnije su opisani u poglavlju 7.1. Potrebni alati za izradu aplikacije. Grafički prikaz ERA modela u programskom alatu MySQL Workbench je prikazan u poglavlju 7.2. ERA model.

Informacije potrebne za izradu svih segmenata aplikacija od baze podataka do grafičkog sučelja aplikacije pronašao sam u dokumentacijama navedenih alata i knjigama navedenih u literaturi na kraju dokumenta.

3. Tehnike pisanja portabilnog softvera

Teško je naći strogo definiranu i dokumentiranu tehniku pisanja portabilnog softvera. U ovom poglavlju će biti objašnjene osnovne tehnike ili savjeti za lakše rješavanje problema portabilnosti programskog koda.

Nove programske jezike i nove biblioteke programskih jezika poželjno je izbjegavati. Razlog tomu je to što znaju biti nedovršeni, tek za daljnje verzije su pouzdani i sigurni za korištenje, a biblioteke i ostali dodaci programskim jezicima uglavnom nisu podržani i testirani na svim operacijskim sustavima.

Multiplatformske biblioteke mogu funkcionirati kako samo ime kaže na više platformi. One su podržane od strane više operacijskih sustava kao što su Windows, Linux, Android itd. No problem je u tome što neki operacijski sustavi ne podržavaju funkcionalnosti koje su potrebne za rad s određenim bibliotekama. Hook(2005) navodi kao primjer kako prijašnje verzije Mac OS-a nisu imale dovoljno raspoložive virtualne memorije, nedostatak dretvi kod DOS-a, Windows mutex(eng. mutual exclusion locks) i Windows TreeView kontrola što je onemogućavalo bibliotekama da izvrše funkcionalnosti za koje su namijenjeni. Prvi najjednostavniji pristup rješavanju problema nedostatka funkcionalnosti operacijskih sustava je konstantno preispitivanje sustava sadrži li on komponente za realizaciju traženih funkcionalnosti. Funkcionalnosti se preispituju if else naredbama. Ako su funkcionalnosti podržane program nastavlja s radom, ako nisu program vraća grešku ili se traži alternativno rješenje za taj problem (if else).

Drugi pristup je nekorištenje funkcionalnosti koji su samo djelomično omogućene na nekim operacijskih sustavima, što donekle ubrzava proces razvoja softvera ali smanjuje efikasnost ključnih funkcionalnosti aplikacije. Treći pristup je ručna implementacija funkcionalnosti koje se razlikuju. Ono što nedostaje ovom pristupu je performansa samog softvera. Razvoj softvera je dosta sporiji jer programer u nekim slučajevima mora duplirati broj programskog koda što na kraju softver čini sporijim. Ukratko najbolje je koristiti izvornu implementaciju funkcionalnosti softvera koje su moguće na određenoj platformi , dok ostale treba ručno implementirati.

4. Problematika izrade multiplatformskih aplikacija

Portabilnost je mogućnost softvera da se prenosi s jedne platforme na drugu. Kako bi programer pisao kod koji je portabilan mora se upoznati sa sljedećim elementima operacijskog sustava:

- Razvojni alati(kompajler, linker, alati za razvoj projekta)
- Programska podrška (debugeri)
- Vrsta procesora (32-bit, 64-bit)
- Vrsta operacijskog sustava (Windows, Linux, Android)
- Biblioteke i API-ji

Svaki od ovih elemenata može imati veliki utjecaj na razvoj softvera.(Hook 2005.)

Prebacivanje softvera s jedne platforme na drugu nije faza razvojnog ciklusa aplikacije, već se proteže kroz cijeli razvojni ciklus aplikacije od specifikacije pa sve do testiranja i uvođenja programa u rad.

Ovaj rad se bazira na poticanju čitatelja da razmišlja o portabilnosti programskog koda kroz cijeli razvojni ciklus aplikacije pa su navedene neke osnovne dobre navike kod pisanja razvoja takve aplikacije. Portabilan kod je potrebno stalno testirati kroz cijeli razvojni ciklus aplikacije i testirati ga na više razvojnih okruženja. Kod je portabilan tek kad se pokrene na više operacijskih sustava. Iz tog razloga je dobro prebacivati kod s jednog operacijskog sustava na drugi što češće jer je bitno pronaći grešku dok se ona još može jednostavno popraviti. Dobra praksa kod pisanja portabilnog koda je raditi u heterogenoj okolini. Različiti sustavi imaju različite kompajlere. Stoga je dobro kroz razvoj softvera koristiti više kompajlera čime bi se izbjegla zastajanja u razvoju zbog promjena istih. (Hook 2005.).

4.1. Operacijski sustavi

Operacijski sustavi upravljaju svim dijelovima računala, a što je najvažnije za ovo poglavlje je da upravljaju svim resursima koje računalo sadrži i na taj način bitno utječu na portabilnost softvera.

Povećanjem kompleksnosti aplikacija došlo je do potrebe za više memorije. Memorija je danas dosta jeftina, ali s druge strane postoje brojne aplikacije koje su zahtjevne i zauzimaju veliku količinu radne memorije. Radna memorija često nije dovoljna da bi se cijeli program spremio i pokrenuo što je prije ograničavalo programere u izgradnji aplikacija. Danas se to rješava straničenjem, gdje je u radnoj memoriji samo dio aplikacije(koji se u tom trenutku koristi), a za ostatak je adresiran dio memorije na disku. Memory mapping (mapiranje memorije) služi za mapiranje sadržaja datoteke u memoriju. Ovo je jedna od izvjesnih problematika kod stvaranja portabilnog koda. Moderan operacijski sustav ima traženu funkcionalnost, ali kad bi softver prebacivali na stariji operacijski sustav aplikacija se ne bi mogla pokrenuti.

Različiti su načini rješavanja tog problema:

- Implementacija
- Pretpostavka da je mapiranje dostupno
- Prioritiziranje portabilnosti

Ručna implementacija ove funkcionalnosti je dosta teška i zahtjevna. Ako pretpostavimo da je mapiranje dostupno u puno slučajeva će nam to olakšati posao, ali opet se mora implementirati mapiranje kako bi se u slučaju nedostupnosti sustavskog mapiranja mogla realizirati tražena funkcionalnost aplikacije.[1]

Proces je osnovna radna jedinica operacijskog sustava. Sastoji se od koda, stoga, pokazivača, registara i stanja. Problematiku ću objasniti na primjeru. Određena web aplikacija treba stvoriti proces za svakog korisnika koji se spoji. Nažalost procesi ne nasljeđuju svojstva procesa roditelja direktno. U tom slučaju potrebno je stvoriti duplikat procesa. To se realizira naredbom fork() u Unix-u. Windows nema direktnu zamjenu za naredbu fork() ali se proces može stvoriti naredbom CreateProcess().[1]

Dretve su dijelovi procesa koje se sastoje od registara, stoga, programskog brojila i ostalih podataka potrebnih za definiranje dretve koje u ovom radu nećemo spominjati. Razvojem dretvi programeri su postupno predstavljali svoje implementacije dretvi pa tako su kako je naveo Hook (2005) nastale Unix International(UI) dretve, DCE dretve, C-threads, Mach dretve. Win32 dretve, OS/2 dretve, Linux dretve i na kraju POSIX pthreads koji su postale standard Unix i Windows sustava. Ukoliko neki sustav ne sadrži dretve potrebno je

implementirati akcije, suspenzije, ubijanje dretvi, sinkronizaciju i ostale vezane karakteristike za dretve što programiranje čini mukotrpnim i zahtjevnim.[1]

Računala mogu biti korištena od strane više korisnika. Ako jedno računalo dijeli više korisnika postoji potreba da svaki korisnik ima sebi prilagođene postavke. Od izgleda i rezolucije zaslona pa sve do jačine zvuka i raspodjele datoteka prema pojedinačnom korisniku računala. Svako računalo bi trebalo imati mogućnost konfiguracije računala po pojedinačnom korisniku. Zbog navedene problematike Microsoft je razvio *Microsoft Windows Registry*. *Microsoft Windows Registry* koristi centralnu bazu podataka koja se naziva registar. Registar sadrži sistemske podatke i može sadržavati konfiguracijske podatke za pojedinačnog korisnika. Registar funkcionira pomoću ključa koji hijerarhijski smješta podatke u datotečni sustav. U ključu su sadržani podaci kao putanja do pojedinačnih konfiguracijskih datoteka. Primjera radi, korisnik želi promijeniti postavke zvuka na svom računalu. Putanja do konfiguracijskih svojstava izgleda ovako:

HKEY_TRENUTNI_KORISNIK\folder1\folder2\konfiguracija_zvuka:

HKEY_TRENUTNI_KORISNIK predstavlja ključ kojim se dolazi do konfiguracijskih svojstava svakog pojedinog korisnika. Naime ovaj registar ima svoje prednosti i svoje mane. Točnije ukoliko se datoteka korumpira sustav će postati neupotrebljiv, dok s druge strane šteta koja nastane se lagano može lokalizirati. Administratori teško mogu modificirati i upravljati konfiguracijom datoteka jer je potreban alat specijaliziran za navigaciju do dotičnog registra, no ukoliko je potrebno prebacivati konfiguraciju s jednog mjesta na drugo, jednostavno je jer je potrebno prebaciti samo jednu datoteku. (Hook 2005.)

Za istu svrhu Linux je razvio *Linux User Data.* Linux je od početka smišljen kao višekorisnički operacijski sustav. Zato *Linux User Data* nema mogućnost spremanja konfiguracije sustava po pojedinom korisniku, svu konfiguraciju sprema u jednu datoteku. (Hook 2005).

4.2. Datotečni sustavi

Datotečni sustav operacijskog sustava služi za pristupanje i upravljanje datotekama spremljenim na tvrdom disku ili USB-u. Operacijski sustavi i aplikacije imaju svoj jedinstven pristup datotekama što otežava pisanje portabilnog koda. Svi moderni sustavi imaju hijerarhijski sustav raspodjele datoteka. Takva raspodjela ne predstavlja problem kod malog broja datoteka, ali kada se sam sustav poveća, dolazi do komplikacije kod navigabilnosti kroz datotečni sustav. Da bi se olakšao pristup aplikacijama stvoreni su prečaci (eng. *shorcuts*). Kod prečaca dolazi do problema u portabilnosti softvera kada se piše aplikacija kojoj se pristupa preko komandne linije. (Hook 2005.)

Za upravljanje prečacima Windows je stvorio svoje Windows LNK datoteke. Točnije svakom prečacu Windows dodaje nastavak .lnk (skraćeno od link) uz naziv datoteke ili aplikacije do koje link sadržan u toj datoteci vodi. Ako pristupamo datoteci preko komandne linije (eng. command prompt) može doći do pogreške. Na primjer, želimo pristupiti datoteci fakultet koja se nalazi negdje na disku u komandnu liniju upisujemo:

C:\Documents and Settings\josbijeli**cd fakultet**

Ukoliko datoteka ima ekstenziju .lnk , doći će do pogreške jer zapravo prečac koji smo pokušali otvoriti sadrži link do druge datoteke.

Unix sustav ima dvije različite vrste linkova. Tvrdi i mekani linkovi (eng. soft and hard links). Tvrdi linkovi su jednostavno imena neke datoteke. U Unix-u je moguće imati datoteku koja ima više naziva koji predstavljaju linkove do tražene datoteke. Tvrde linkove kreiramo upisom naredbe *ln* u terminal i pridruživanja imena linka traženoj datoteci.

ln staraDatoteka noviLink

U terminalnu upišemo naredbu *ln* i poslije naredbe ime datoteke na koju želimo referencirati tvrdi link i poslije toga naziv novog linka za tu datoteku.

Mekani linkovi su zapravo pokazivači na datoteku. Oni su samo po sebi datoteke kojima se može manipulirati, što može uzrokovati dosta problema ako na to ne obratimo pozornost.

$ ln -s izvornaDatoteka nazivLinka

Putanje do određene datoteke se bitno razlikuju između različitih operacijskih sustava što otežava rad s datotekama. Hijerarhijski datotečni sustavi koriste se direktorijima ili folderima za raspodjelu datoteka po memoriji. *Path seperator* (separator putanje datoteke) koje koristi Windows je *backslash*(\). Linux s druge strane koristi *forward slash(/).* To uzrokuje probleme u portabilnosti kada radimo s datotekama. Kod pisanja relativnih putanja do određene datoteke, točnije lociranja direktorija u kojem se datoteka nalazi ili direktorija roditelja operacijski sustavi

se razlikuju po specijalnim znakovima koje koriste. Tablica 13.1 prikazuje kako se određeni specijalni znakovi razlikuju s obzirom na različite operacijske sustave.

Tablica 1 : Specijalni znakovi kod prikaza putanje na operacijskim sustavima

Operacijski sustav	Korijenski direktorij	Trenutni direktorij	Direktorij - roditelj
Windows	\	.	..
Linux	/	.	..
Mac OS	Naziv diska	N/A	

Kao što vidimo specijalni znakovi imaju različita značenja i stoga je preporučljivo da pri izradi multiplatformske aplikacije za imena datoteka ne koristimo specijalne znakove kao što je razmak, točka, uskličnik.

Putanja do određenog direktorija ili datoteke ima svoju ograničenu duljinu. Kako navodi Hook(2005):

- MS-DOS/Windows FAT32 i Win32 NTFS datotečni sustavi dopuštaju 256 znakova
- MS-DOS FAT16 datotečni sustav se temeljio na 8.3 formatu(8 znakova i 3 ekstenzija)
- Linux ReiserFS datotečni sustav dopušta 255 znakova
- Mac OS datotečni sustav dopušta 32 znaka

Jedna od bitnih problematika kod različitih datotečnih sustava kada govorimo o portabilnosti softvera je osjetljivost na mala i velika slova (eng. *case sensitivity*). Ako imamo dvije datoteke npr. Designer.py i designer.py i ako premještamo te dvije datoteke sa sustava koji je case sensitive na sustav koji nije, datoteke će se smatrati identične i program neće raditi.

Sigurnost je jedan od bitnih aspekata kod izgradnje softvera, ali sigurnost donosi određene prepreke kada govorimo o portabilnosti softvera. Programer može pisati softver pretpostavljajući da ima pristup svim resursima operacijskog sustava. Kada prebacujemo softver s jednog takvog operacijskog sustava koji dopušta pristup svim direktorijima na operacijski sustav koji ima određenu zaštitu podataka i ne dopušta spremanje datoteka na određene dijelove datotečnog sustava dolazi do pogreške u radu. Stoga nije sigurno pretpostavljati pristupačnost određenim dijelovima operacijskog sustava, jedino i sigurno rješenje je spremati sve podatke na korisnički direktorij koji bi trebao biti siguran i dostupan na svakom datotečnom sustavu.

4.3. Dinamičke biblioteke

Razvoj aplikacije često zahtjeva pisanje velike količine programskog koda koji često programeru uzme većinu vremena i dosta memorije na računalu. Kako bi se smanjio broj potrebnih linija koda i smanjio udio memorije koje aplikacije zauzimaju, stvorene su biblioteke koje u sebi sadrže funkcije, klase i ostale komponente. Danas programeri više ne moraju ponavljati svoj kod u aplikacijama već kreiraju biblioteke iz kojih pozivaju ili klasu ili funkciju te ih koriste u samom razvoju aplikacije.

Dinamičko povezivanje biblioteka i programa naziva se dinamičko referenciranje. Dinamičko referenciranje može funkcionirati na dva načina. Program se može povezati s uvezenom bibliotekom koja sadrži referencu na traženu dinamičku biblioteku ili se povezuje s dinamičkom bibliotekom direktno. Kada se program pokrene, operacijski sustav saznaje koje sve vanjske biblioteke trebaju biti referencirane na pokrenutu aplikaciju. Operacijski sustav nakon toga učitava dinamičke biblioteke koje se koriste u programu, a ukoliko je referenca neispravna ili na toj adresi biblioteka više ne postoji operacijski sustav će javiti grešku.[1]

S druge strane, program može dinamički i eksplicitno učitati cijelu biblioteku i referencirati podatke i funkcije potrebne za izvršavanje funkcionalnosti aplikacije. Na taj način operacijski sustav neće izbacivati grešku, ignorirajući ih ako određene funkcije nisu dostupne. Najčešća upotreba dinamičkog učitavanja su dodaci (eng. *plug-inovi*) koje koristimo na aplikacijama. Dinamičke biblioteke mogu biti verzionirane, točnije mogu se ažurirati. Ažuriranje biblioteka u nekim situacijama može biti „dvosjekli mač“. Točnije ispravnost aplikacije ovisi o ispravnosti biblioteka koje aplikacija koristi. Korištenjem biblioteka aplikacije postaju zavisne o ispravnosti biblioteka i time dolazi do promjene (ažuriranja) biblioteke, to naravno utječe na samu aplikaciju i u tom slučaju može doći do greške pri njezinom izvođenju. Rješenje za problem ažuriranja je konstantno preispitivanje verzije biblioteke selekcijama (if else).[1]

Kako navodi Hook (2005) biblioteke mogu biti spremljene bilo gdje na računalu. Zbog tog problema Windows okruženje teži da DLL datoteke sprema u računalo po sljedećem redoslijedu:

1. Direktorij aplikacije
2. Sistemski direktorij Windowsa(System32 ili System64)
3. Windows direktorij
4. Cjelokupna putanja
5. Trenutni direktorij

Windows je kreirao svoje vrste biblioteka a to su DLL-ovi. DLL sadrži funkcije ili klase koje se pozivom mogu koristiti u programskom kodu. DLL programer mora specificirati koje funkcije trebaju biti javno dostupne. U kasnijim verzijama Windowsa funkcije su se dohvaćale jedinstvenim nazivom. Kompiliranjem i referenciranjem programskog koda nastaju dvije biblioteke, jedna je izvozna(.lib), a druga je .dll(dinamička biblioteka). Za pristupanje DLL-ovima koristi se direktiva __declspec(export). Direktiva daje do znanja kompajleru da se određene funkcije ili klase trebaju izvesti. S druge strane aplikacija treba uvesti biblioteku kako bi koristila njene funkcije. U tu svrhu prilikom pisanja koda mogu se koristiti dva zaglavlja. Jedno zaglavlje služi za pristupanje DLL-u, a drugo zaglavlje za korištenje zadanih funkcija DLL-a(__declspec(import)). Ovako korištenje DLL-a je dosta nespretno i često dolazi do javljanja grešaka u programskom kodu. Iz tog razloga programeri koji izrađuju DLL-ove definiraju konstantu koja je postavljena da, ovisno o situaciji, na ispravan način koordinira uvozom i izvozom funkcija iz biblioteke.[1]

Linux je razvio javne objekte koji su u ekvivalenciji s dinamičkim bibliotekama Windowsa. Prilikom spremanja datoteke kao javne objekte Linux je odredio da svaka takva datoteka ima nastavak .so , dok prefiks kao i kod ostalih biblioteka mora bi lib.[1] Tako možemo kreirati biblioteku u Linuxu pod naziv libName.so gdje lib predstavlja prefiks po kojem operacijski sustav prepoznaje da se radi o biblioteci, Name je proizvoljno ime biblioteke i *.so* je nastavak po kojem operacijski sustav zna da se radi o javnom objektu. Kod kreiranja biblioteke kao javnog objekta potrebno je aktivirati neke od zastavica a to se postiže na sljedeći način.

gcc -shared -fPIC -o libName.so Name.c [1]

5. Platforme za izradu multiplatformskih aplikacija s grafičkim korisničkim sučeljem

Današnja tehnologija omogućuje lakše i jednostavnije programiranje multiplatformske aplikacije. Napraviti portabilan softver je danas dosta jednostavnije zbog alata koji su se razvili u zadnjih nekoliko godina. Platforme kao što su Haxe, Qt, Electron, B4J, omogućuju jednostavno pisanje portabilnog koda.

5.1. Haxe

Haxe je alat za razvoj multiplatformskih aplikacija. Haxe koristi svoj programski jezik koji se temelji na ECMAScript standardu ali se može prevesti u skoro bilo koji programski jezik. Haxe programski jezik specijalno je dizajniran za Haxe koji omogućuje alatu da bude fleksibilan i da se aplikacija razvijena u Haxe-u ponaša isto na više različitih platformi. Programski jezik je vrlo jednostavan i sličan svim drugim objektno orijentiranim jezicima. Nakon što se napiše program u Haxe-u i spremi na računalo, dokument dobije ekstenziju .hx i može se prevesti u druge programske jezike npr. u JavaScript:

haxe – main AppName -js AppName.js,

gdje je Haxe naredba za prevođenje, a AppName predstavlja ime programa koji treba prevesti u JavaScript. Kod se može prevesti u jezike kao što si JavaScript, PHP, C++, Python i ostale.[5]

Sljedeći kod prikazuje jednostavan primjer ispisa poruke „Hello World“ u programskog jeziku Haxe.

```
class Main {
  static public function main():Void {
    trace("Hello World");
  }
}
```

Program se sprema pod ekstenzijom .hx i moguće ga je testirati u command prompt-u(cmd). Naredbom *haxe -main Main --interp* u command prompt-u ispisat će se poruka „Hello World“. Prethodna komandna linija Haxe alata sastoji se od 2 parametra. Prvi je *main Main* koji označava metodu main klase Main koja se treba izvršiti.[5]

Drugi parametar je *interp* koji označava interpretiranje programa koristeći unutrašnji makro sistem(eng. internal macro system)[6].

5.2. Electron

Electron je zapravo biblioteka razvijena od strane GitHuba za izgradnju multiplatformskih aplikacija koja se temelji na HTML-u, CSS-u i JavaScript jeziku. Može se koristiti na operacijskim sustavima kao što su Mac, Windows i Linux.

Na primjeru jednostavne aplikacije koja u pregledniku ispisuje „Hello World“ bit će prikazane osnove samo platforme. Za kreiranje Electron aplikacije prvo trebamo kreirati 3 osnovne datoteke: index.html, main.js i package.json.

package.json

```
{
"name"    : "electron-hello-world",
"version" : "0.1.0",
"main"    : "main.js"
}
```

U datoteci package.json nalazi se ime i verzija skripte koja pokreće aplikaciju.

Main.js

```
const electron = require('electron')
// Module to control application life.
const app = electron.app
// Module to create native browser window.
const BrowserWindow = electron.BrowserWindow

// Keep a global reference of the window object, if you don't, the window
will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

function createWindow () {
  // Create the browser window.
  mainWindow = new BrowserWindow({width: 800, height: 600})

  // and load the index.html of the app.
  mainWindow.loadURL(`file://${__dirname}/index.html`)

  // Open the DevTools.
  mainWindow.webContents.openDevTools()

  // Emitted when the window is closed.
  mainWindow.on('closed', function () {
    // Dereference the window object, usually you would store windows
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    mainWindow = null
  })
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow)
```

```
// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On OS X it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') {
    app.quit()
  }
})

app.on('activate', function () {
  // On OS X it's common to re-create a window in the app when the
  // dock icon is clicked and there are no other windows open.
  if (mainWindow === null) {
    createWindow()
  }
})

// In this file you can include the rest of your app's specific main
process
// code. You can also put them in separate files and require them here.
```

Main.js pokreće samu aplikaciju[11].

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Electron Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Datoteka index.html je samo sučelje. Aplikacija se pokreće u cmd-u pomoću naredbe:

.electron [11]

U Electronovom pregledniku aplikaciju jedino JavaScript može debugirati. Pokretanje debugiranja koda se vrši preko komandne linije upisivanjem naredbi:

--inspect

Ili

--inspect – brk,

Uz ove dvije naredbe ide i port na koji debuger mora biti spojen prilikom debugiranja određenog programskog koda. Electron koristi Chromium za prikaz web stranica napisanih u HTML-u i CSS-u. Koristi web stranice kao svojevrstan GUI kojim se upravlja pomoću JavaScript programskog jezika.

Platforme na kojima je podržan Electron:

- MacOS – 64-bitni sustav, minimalna verzija macOS 10.9.
- Windows – Windows 7, Windows 8 ... Windows 10, podržan na x86 i x64, dok na ARM verzijama Windowsa još nije.
- Linux – ia32(i686) i x64(amd64), Ubuntu 12.04 i novije verzije, Fedora 21, Debian 8

Neke od poznatih aplikacija pisanih u Electronu : GitHub Desktop, Hyper, Visual Studio Code, WordPress.com itd...[10]

5.3. B4J

B4J je besplatan alat za pisanje desktop aplikacije, servera i rješenja za takozvani „Internet of things“. Ovaj alat omogućuje pisanje koda u B4X jeziku koji je moderna verzija Visual Basic programskog jezika. B4J se može pokrenuti na operacijskim sustavima Windows, Linux i macOS što ga čini alatom za pisanje multiplatformskih aplikacija.

```
Sub Process_Globals
   Public srvr As Server
End Sub

Sub AppStart (Args() As String)
   Log.("HelloWorld")
End Sub
```

Ova jednostavna aplikacija prikazuje kako izgleda kod u B4X programskom jeziku. Aplikacija jednostavno ispisuje „HelloWorld“.[9]

6. Qt

Qt je okvir za izgradnju multiplatformskih aplikacija. Platforme na kojima je Qt podržan su OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, SailFish OS i ostali. Qt je aplikacijski okvir napisan u C++-u. Pri izradi Qt-a koristio se MOC(eng. Meta-Object Compiler) pomoću kojeg je proširen jezik C++ bibliotekama i funkcijama za izradu multiplatformskih aplikacija. Prije kompilacije koda napisanog u Qt-u, MOC generira datoteku koja u sebi sadrži izvorni C++ kod i prosljeđuje ga standardnim kompajlerima kao što su Clang, GCC i ostali.

S instalacijom Qt-a dolazi i integrirano razvojno okruženje Qt Creator. Qt Creator sadrži automatsko dovršavanje i isticanje koda, debuger i ostale kontrolne sisteme kojima se olakšava programiranje i sam razvoj aplikacije. Qt Linguist omogućuje prijevod aplikacija u lokalne jezike više zemalja. Ova funkcionalnost se realizira pomoću lupdate, lrelease i lconvert funkcija koje olakšavaju prevođenje aplikacija na lokalni jezik. Za kreiranje aplikacija s grafičkim korisničkim sučeljem razvojni tim Qt-a je razvio Qt Designer. Qt Designer služi za kreiranje prozora, responzivnih dijaloga i drugih grafičkih elemenata aplikacije. Drugi način izrade aplikacija s grafičkim korisničkim sučeljem je pomoću QtQuick modula. QtQuick je razvijen pomoću jezika QML. QML je deklarativni programski jezik koji integrira JavaScript programski jezik za svrhe proceduralnog programiranja.[6]

6.1. PyQt

PyQt biblioteke kreirane su s ciljem povezivanja Python programskog jezika i Qt aplikacijskog okvira. PyQt je podržan na više operacijskih sustava kao što su Linux, Windows, Android i ostali. Dvije su verzije PyQt-a : PyQt4 i PyQt5. U ovom radu ću raditi s PyQt5 bibliotekom pa ću iz tog razloga opisati komponente PyQt5 biblioteke.

PyQt5 ima brojne komponente instalirane na PyQt5 paketu. PyQt5 sadrži sljedeće mogućnosti:

- Pyuic5 konvertira QtWidgets u čisti Python kod
- Pyrcc5 služi za ugrađivanje proizvoljnih resursa(ikona, slika...)
- Pylupdate5 služi za prevođenje koda u lokalni jezik
    - kreira datoteku .ts (eng. translation file) koristi QtLinguist

PyQt biblioteka omogućuje jednostavnije kreiranje aplikacija s grafičkim korisničkim sučeljem u Pythonu. U ovom poglavlju će biti objašnjeni jedni od glavnih modula koji olakšavaju izgradnju takvih aplikacija.

Najvažniji modul za izgradnju aplikaciju s grafičkim korisničkim sučeljem je QtWidgets modul koji sadrži klase kao što su QWidget, QPushButton, QLineEdit, QLabel i koje služe za kreiranje korisničkog sučelja.[7]

Modul QtGui sadrži klase koje služe za integraciju samog sustava(eng. window system integration), rukovanje događajima(eng. event handling), 2D grafiku, slike, font i sam izgled teksta.[7]

Modul QtPrintSupport sadrži klase za povezivanje aplikacije s lokalnim printerom i generiranje PDF dokumenata.[7]

Klasa QApplication omogućava uspostavljanje kontrolnog toka aplikacije(eng. *control flow*). QApplication kontrolira inicijaliziranje i kreiranje prozora aplikacija(eng. *widgets*). Svaka aplikacija koja koristi klasu QApplication sadrži samo jedan QApplication objekt bez obzira koliko se prozora koristi u aplikaciji. Ova klasa ima sljedeće odgovornosti:

- Inicijalizira aplikaciju s obzirom na korisničke postavke računala
- Kontrolira rukovanje događajima tako što prima događaje s glavnog sučelja i šalje ih drugim prozorima.
- Definira početnu boju aplikacije i lokalizaciju teksta na grafičkom sučelju
- Upravlja rukovanje kursorom miša…[7]

7. Aplikacija ShoppingApp

Praktični dio ovog rada bazira se na izradi multiplatformske aplikacije s grafičkim korisničkim sučeljem. Iz tog razloga u ovom radu je odabrana izrada pojednostavljene desktop aplikacije za kupnju odjeće preko računala. „ShoppingApp“ aplikacija treba biti pokrenuta i kompilirana na više platformi a za ovaj rad odabrana su dva operacijska sustava, Windows i Linux. Pet je glavnih funkcionalnosti ove aplikacije:

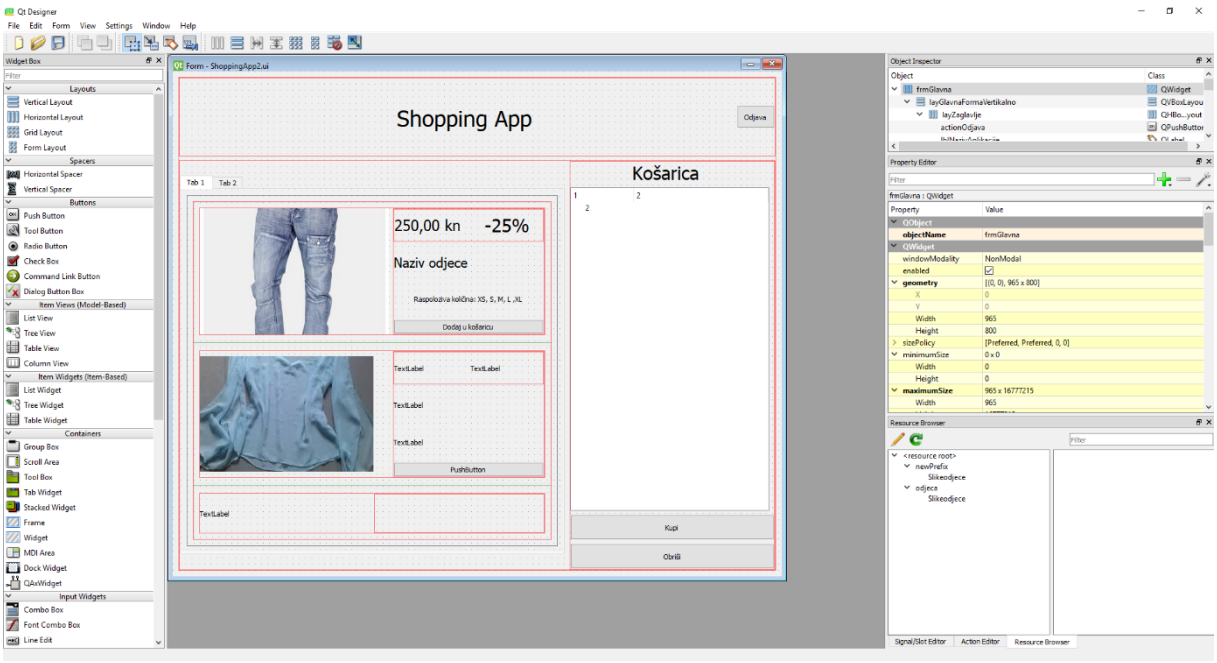
- Registracija korisnika
- Prijava korisnika
- Pregled odjeće
- Kupovina odjeće
- Administracija

Korisnik unutar forme za registraciju unosi osnovne podatke koji se spremaju u kasnije opisanu bazu podataka. Registrirani korisnik koristi svoje korisničko ime i lozinku kako bi se kasnije prijavio i na taj način koristio funkcionalnosti aplikacije. Prijava je u jedan korak, gdje korisnik upisuje svoje ispravno korisničko ime i lozinku te klikne gumb za prijavu kojim se prijavljuje i koristi aplikaciju. Ukoliko korisnik ne unese ispravno korisničko ime ili lozinku sukladnu podacima u bazi podataka, aplikacija javlja upozorenje da korisničko ime ili lozinka nisu ispravni.

Nakon prijave korisniku se otvara glavni prozor aplikacije. Lijevi(zapadni) dio glavnog prozora aplikacije čini jednostavna navigacijska traka. Pomoću navigacijske trake može se pregledavati odjeća po kriteriju spola, uzrasta ili popusta. Točnije na navigacijskoj traci sadržani su elementi : „Muška odjeća“, „Ženska Odjeća“, „Dječja odjeća“ i „Aktualno“. Klikom na „Muška Odjeća“ otvara se prozor na kojem je prikazana samo odjeća za muškarce. Ženska odjeća se može vidjeti klikom na element „Ženska Odjeća“. Dječja odjeća se može vidjeti u prozoru koji se otvara klikom na „Dječja Odjeća“. Ulaskom u aplikaciju ili klikom na element navigacijske trake „Aktualno“ otvara se prozor na kojem se nalazi ona odjeća za koju vrijedi određeni popust. Unutar prozora kojeg otvorimo klikom na neki od elemenata navigacijske trake nalazi se popis odjeće. Svaki komad odjeće sadrži svoju sliku, cijenu, raspoloživu količinu i raspoloživu veličinu (S, XS, M, L...). Uz navedene karakteristike uz svaki komad odjeće prikazan na prozoru generira se gumb „dodaj u košaricu“ i gumb „odaberi veličinu“. Klikom na gumb „Kupi“ otvara se prozor s informacijom da je odjeća kupljena te klikom na gumb „Obriši“ briše se sve iz košarice.

7.1. Potrebni alati za izradu aplikacije

Pri izradi multiplatformske aplikacije korišten je skriptni jezik Python. Korištena je najnovija verzija Pythona 3.6.2. Python kao interpreterski jezik je odličan izbor za izradu multiplatformskih aplikacija jer se Python interpreter pokreće na više platformi pa programski kod nije potrebno izmjenjivati pri prebacivanju s jedne platforme na drugu. Za grafičko korisničko sučelje korištene su PyQt biblioteke. Prilikom izrade grafičkog korisničkog sučelja korišten je Qt designer. Qt designer je alat pomoću kojeg se grafički može izraditi aplikacijski okvir. Alat ne daje mogućnost pisanja programske logike u programskom jeziku Python, ali zato omogućuje programiranje u jeziku C++.



Slika 1. Primjer izrade grafičkog sučelja u Qt Designeru

Na slici 1 se može vidjeti primjer izrade grafičkog korisničkog sučelja u Qt Designeru. Datoteka izrađena u Qt Designeru se sprema kao .ui datoteka. Datoteka s ekstenzijom .ui nije čitljiva Pythonu ili drugim programskim jezicima. Potrebno je generirati Python kod iz .ui datoteke koji se onda koristi kao grafičko korisničko sučelje pri izradi aplikacije. Pyhton omogućuje navedeno generiranje koda pomoću naredbe komandne linije pyuic5 za PyQt5 ili pyuic4 za PyQt4 biblioteku. Primjera radi, datoteka.ui se nalazi u datoteci čija je putanja C:\Users\User1\datoteka.ui. Za generiranje Python koda unutar komandne linije potrebno je upisati:

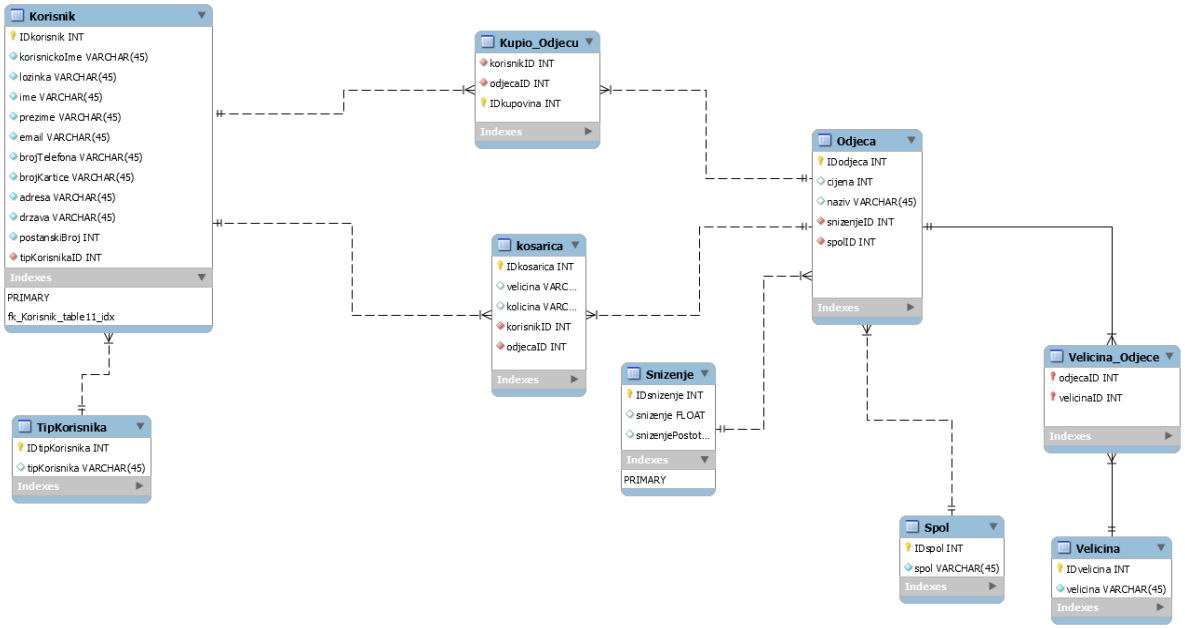
C:\Users\User1> pyuic5 datoteka.ui -o ui_datoteka.py

Ovom linijom generirana je python datoteka ui_datoteka.py. Argument -o omogućuje da se generirani kod spremi u datoteku. Ova linija radi i na terminalu(Linux operacijski sustav).

Programski kod je pisan u PyCharm Community Edition 2017.2.2. PyCharm kao editor za pisanje programskog koda odličan je izbor jer podržava osvjetljavanje koda i podržava automatsko dovršavanje koda što olakšava i skraćuje vrijeme programiranja. Za manipuliranje podacima iz baze podatka koristio sam biblioteku MySQLdb.

7.2. ERA model

Za spremanje podataka potrebnih za izvršavanje funkcionalnosti aplikacije i spremanja podataka o korisnicima i odjeći korištena je MySQL baza podataka. Za samu izradu baze podataka korišten je MySQL Workbench.X 6.3. CE. U njemu su kreirane tablice i potrebne veze između tablica.



Slika 2: Era model

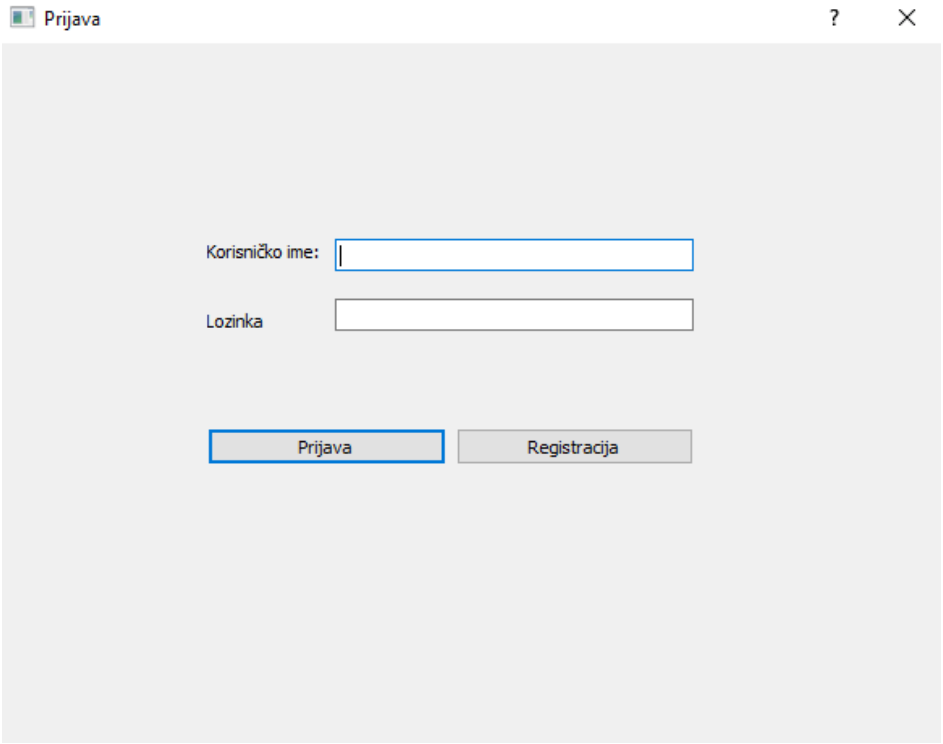
Korisnik pri registraciji unosi svoje osobne podatke koji se upisuju u tablicu „Korisnik“. Tip korisnika je po „defaultu“ registrirani korisnik. Odjeća ima svoju cijenu i naziv spremljene u tablici „Odjeca“. Svaka odjeća ima svoje veličine (XS, S, M, L, XL, XXL). Preko tablice „Velicina_Odjece“ odjeći se pridružuje raspoloživa veličina iz tablice „Velicina“. Tablica „Spol“ ima vanjski ključ na tablicu „Odjeca“ pomoću kojeg se određuje pripada li određena odjeća ženskoj, muškoj ili dječjoj odjeći. U ovom primjeru određeni komad odjeće može imati samo jedno sniženje koje se realiziralo pomoću vanjskog ključa „snizenjeID“ na tablicu „Odjeca“. Tablica „Snizenje“ u sebi sadrži postotak potreban za prikaz sniženja u aplikaciji i decimalni broj koji služi kako bi se izračunala snižena cijena artikla. Za prikaz i realizacije funkcionalnosti dodavanja artikla u košaricu koristi se tablica „kosarica“ koja sprema vrijednost primarnog ključa označene odjeće i vrijednost primarnog ključa prijavljenog korisnika.

7.3. Grafičko korisničko sučelje

U ovom poglavlju bit će prikazane sve forme koje su korištene pri realizaciji funkcionalnosti aplikacije. Kao što sam i ranije naveo, forme su kreirane pomoću Qt-Designera. Kreirane forme kao što su forma za prijavu i registraciju, glavna forma s prikazom popisa odjeće i ostale forme za administraciju odjeće i korisnika bit će objašnjene u sljedećim poglavljima a programski kod generiran pomoću pyuic5 naredbe je dostupan na Git-u čiji link sam stavio na kraju dokumenta

7.3.1. Prijava

Prilikom pokretanja aplikacije otvara se forma za prijavu prikazana na sljedećoj slici:



Slika 3 : Forma za prijavu

Na slici 3 prikazana je forma za prijavu. Ovo grafičko korisničko sučelje nasljeđuje QtWidgets klasu u kojoj su sadržani neki elementi koje nudi PyQt biblioteka:

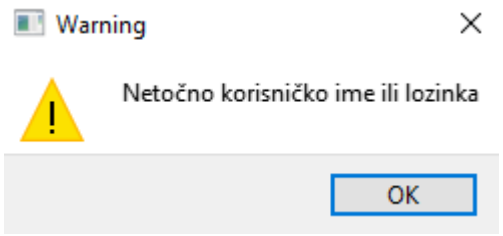
- Label – tekstualni okvir
- LineEdit – tekstualno polje
- QpushButton – komponenta grafičkog sučelja za izvršavanje naredbi

Korisnik u zadana tekstualna polja unosi korisničko ime i lozinku. Kreira se upit koji dohvaća korisnika iz baze podataka ako postoji takav kojemu su korisničko ime ili lozinka jednaki unesenom. Ako postoji takav korisnik, otvara se glavna forma koja prikazuje popis odjeće za traženu kategoriju. Glavna forma je detaljno opisana u poglavlju 7.3.3.

Sljedeći kod prikazuje realiziranu funkcionalnost prijave u aplikaciju:

```
def provjeraKorisnika(self):
    korisnik = Korisnik.Korisnik()
    korisnickoIme = self.lineEditKorisnickoIme.text()
    lozinka = self.lineEditLozinka.text()
    korisnikID = korisnik.prijavaProvjeriKorisnika(korisnickoIme, lozinka)
    if korisnikID:
        self.ShoppingApp = ui_ShoppingApp.Ui_frmGlavna(korisnikID)
        self.ShoppingApp.show()
        self.close()
    else:
        self.prikaziUpozorenje('Warning', 'Netočno korisničko ime ili
lozinka')
```

Ukoliko lozinka ili korisničko ime nisu ispravni kreira se prozor(upozorenje) u kojem je navedeno da korisničko ime ili lozinka nisu ispravni što možemo vidjeti u kodu iznad i na slici ispod.

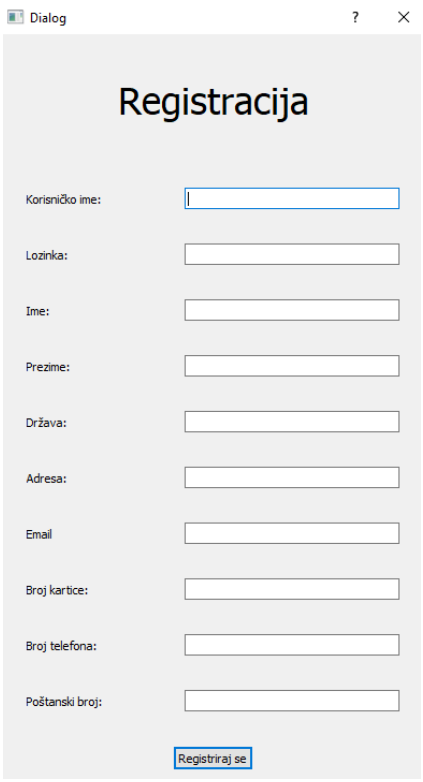


Slika 4. Upozorenje za krivu prijavu

7.3.2. Registracija

Klikom na gumb za registraciju u formi za prijavu otvara se forma za registraciju korisnika. Korisnik kreira svoje korisničko ime i lozinku te unosi ostale osobne podatke kao što su ime, prezime, država, adresa, email i ostali podaci.

Sljedeća slika prikazuje navedenu formu:



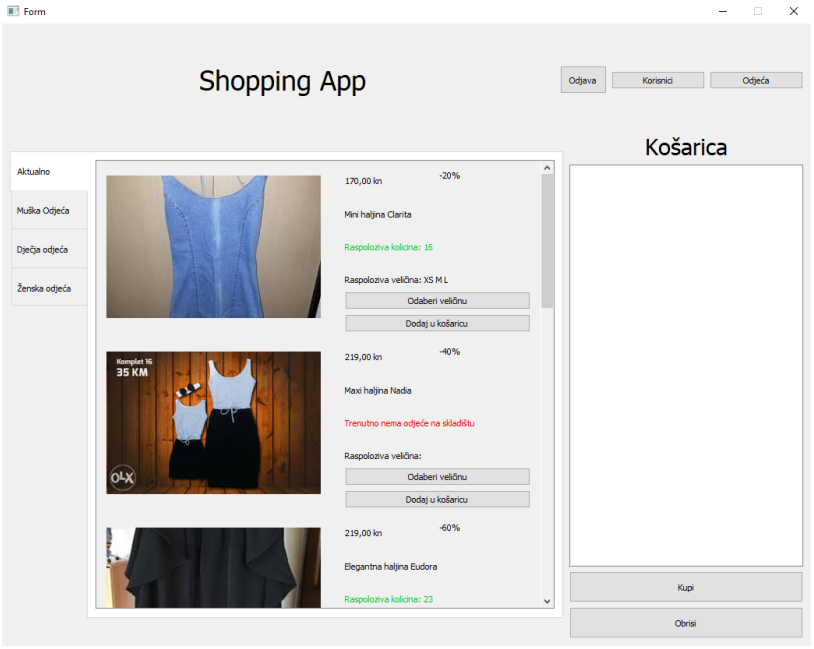
Slika 5. Forma za registraciju

Slika 5 prikazuje formu za registraciju koja u sebi sadrži tekstualne okvire i tekstualna polja za unos podatka u bazu. Korisnik unosi proizvoljno korisničko ime i lozinku. Ostale podatke mora unijeti kako bi se registrirao. Nakon što korisnik unese sve podatke potrebne za registraciju klikom na gumb „Registriraj se“ podaci se spremaju u bazu podatka, točnije u tablicu „Korisnik“. Ukoliko korisnik ne unese sve podatke, aplikacija javlja upozorenje da nisu uneseni svi podaci i sprječava registraciju što pokazuje sljedeći isječak koda:

```
if(korisnickoIme and lozinka and ime and prezime and drzava and
postanskiBroj and adresa and brojKartice and brojTelefona and email):
    korisnik = Korisnik.Korisnik()
    korisnik.registracijaKorisnika(korisnickoIme, lozinka, ime, prezime,
email, drzava, postanskiBroj, brojTelefona, brojKartice, adresa)
    self.prijava = ui_login.Ui_Prijava()
    self.prijava.show()
    self.close()
else:
    self.Upozorenje('Upozorenje', 'Nisu uneseni svi podaci!')
```

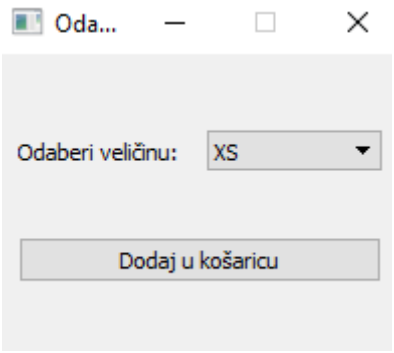
7.3.3. Glavno sučelje aplikacije

Uspješnom prijavom korisnika u aplikaciju otvara se glavno sučelje aplikacije, prozor u kojem se nalazi popis sve odjeće i košarica za kupnju:



Slika 6: Glavna forma aplikacije

Formu prikazanu na slici 6 možemo podijeliti na više dijelova. Lijevo se nalazi navigacijska traka koja prikazuje odjeću čija vrsta odgovara muškoj, ženskoj, dječjoj ili odjeći s popustom(aktualno). Ovisno o odabranoj vrsti u prozoru za prikaz odjeće prikazana je odjeća s pripadajućom slikom, nazivom, cijenom, veličinom i količinom. Ako postoji popust na određenu odjeću, popust je prikazan u gornjem desnom kutu prozora. Kraj slike svake pojedine odjeće kreirana su 2 gumba. Kada korisnik kupuje odjeću prvo mora odabrati željenu veličinu. Klikom na gumb „Odaberi veličinu“ otvara se sljedeća forma:



Slika 9: Odabir veličine

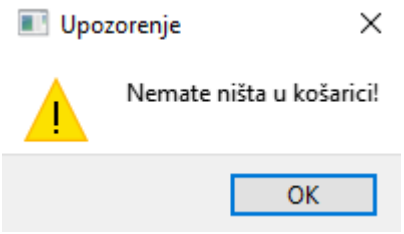
Na formi sa slike 9, korisnik odabire veličinu koja je raspoloživa za odabranu odjeću. Klikom na gumb „Dodaj u košaricu“ aplikacija odmah dodaje podatke u tablicu „kosarica“ što možemo vidjeti u sljedećem kodu:

```
def odaberiVelicinu(self):
    velicina = str(self.cboxOdabirVelicine.currentText())
    kosarica = Kosarica.Kosarica()
    odjeca = Odjeca.Odjeca()
    raspolozivaKolicinaOdjece =
odjeca.dohvatiRaspolozivuKolicinuKomadaOdjecePoPojedinojVelicini(velicina,
self.odjecaID)
    kolicinaUKosarici =
kosarica.provjeriKolicinuOdjecePoVelicini(self.odjecaID, velicina)
    if kolicinaUKosarici:
        if int(raspolozivaKolicinaOdjece[0][0]) >
int(kolicinaUKosarici[0][0]):
            kosarica.dodajUKosaricu(self.korisnikID, self.odjecaID,
velicina)
        else:
            print("Nema dovoljno robe na skladištu")
    else:
        kosarica.dodajUKosaricu(self.korisnikID, self.odjecaID, velicina)

    self.close()
```

Odabirom veličine korisnik se vraća u glavnu formu i klikom na „Dodaj u košaricu“ zapravo se u listi samo prikazuje sadržaj košarice.

```
def dodajUListuKosarica(self):
    odjeca = Odjeca.Odjeca()
    kursor = odjeca.dohvatiSveIzKosarice(self.korisnikID)
    self.listWidget.clear()
    item = QtWidgets.QListWidgetItem()
    self.listWidget.addItem(item)
    for i in range(kursor.rowcount):
        row = kursor.fetchone()
        item = QtWidgets.QListWidgetItem()
        str1 = row[0]
        str2 = str(row[1]) + ",00 kn"
        str3 = row[2]
        str4 = str(row[3])
        item.setText(str1 + "   " + str2 + "     " + str3 + "     " + str4)
        self.listWidget.addItem(item)
```

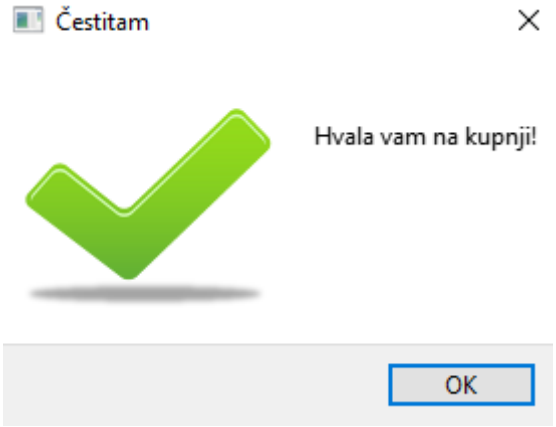
Ispod liste koja prikazuje podatke iz korisnikove košarice, nalazi se gumb kojim korisnik kupuje sve što je „stavio u košaricu“. Ako se korisniku ne sviđa šta je izabrao može obrisati cijelu košaricu. Ukoliko je prilikom kupovine košarica prazna, program javlja sljedeću poruku:



Slika 8. Prazna košarica

Nakon što korisnik klikne na gumb „Kupi“ svi podaci iz košarice se spremaju u tablicu „Kupio odjecu“ i korisniku se javlja poruka da je odjeću kupio.

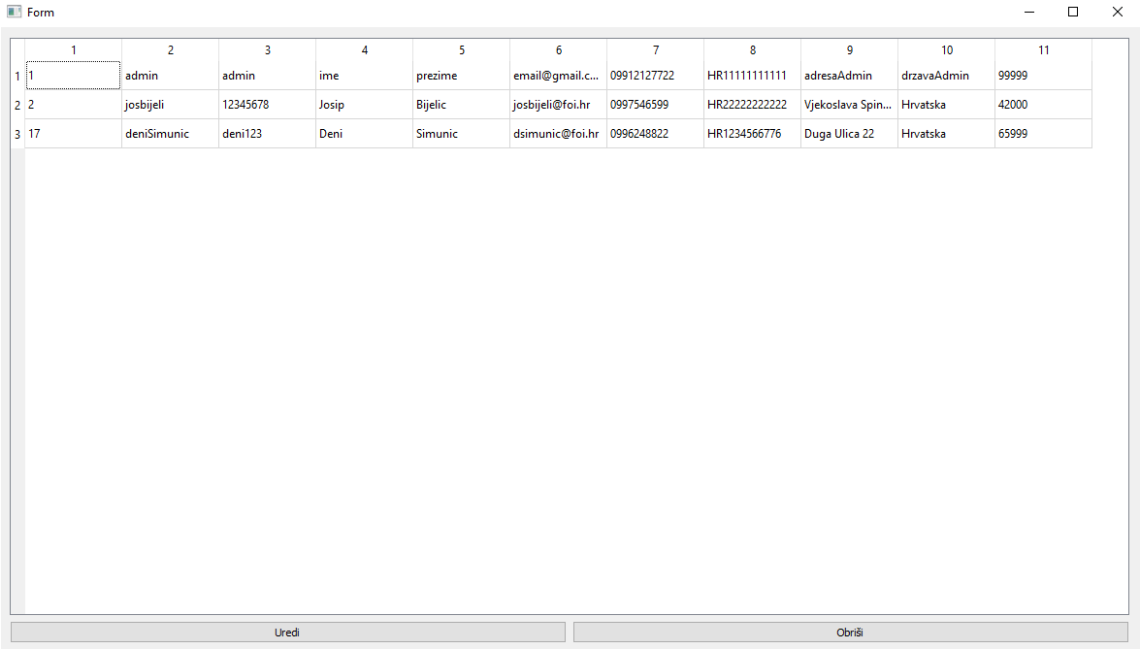
```
def kupi(self):
    if(self.listWidget.count() != 0):
        odjeca = Odjeca.Odjeca()
        self.kupljeno()
        odjeca.unesiKupovinu(self.korisnikID)
        self.izbrisiSve()
        self.osvjeziAplikaciju()
    else:
        self.praznaKosarica()
```



Slika 9. Kupljena odjeća

7.3.4. Administracija korisnika

Administrator nakon prijave u aplikaciju ima mogućnost uređivanja i brisanja korisnika koji su kreirali svoje račune. Ako je prijavljeni korisnik administrator kako vidimo na slici 6, administrator može klikom na gumb „Korisnici“ otvoriti sljedeću formu:



Slika 10. Administracija korisnika

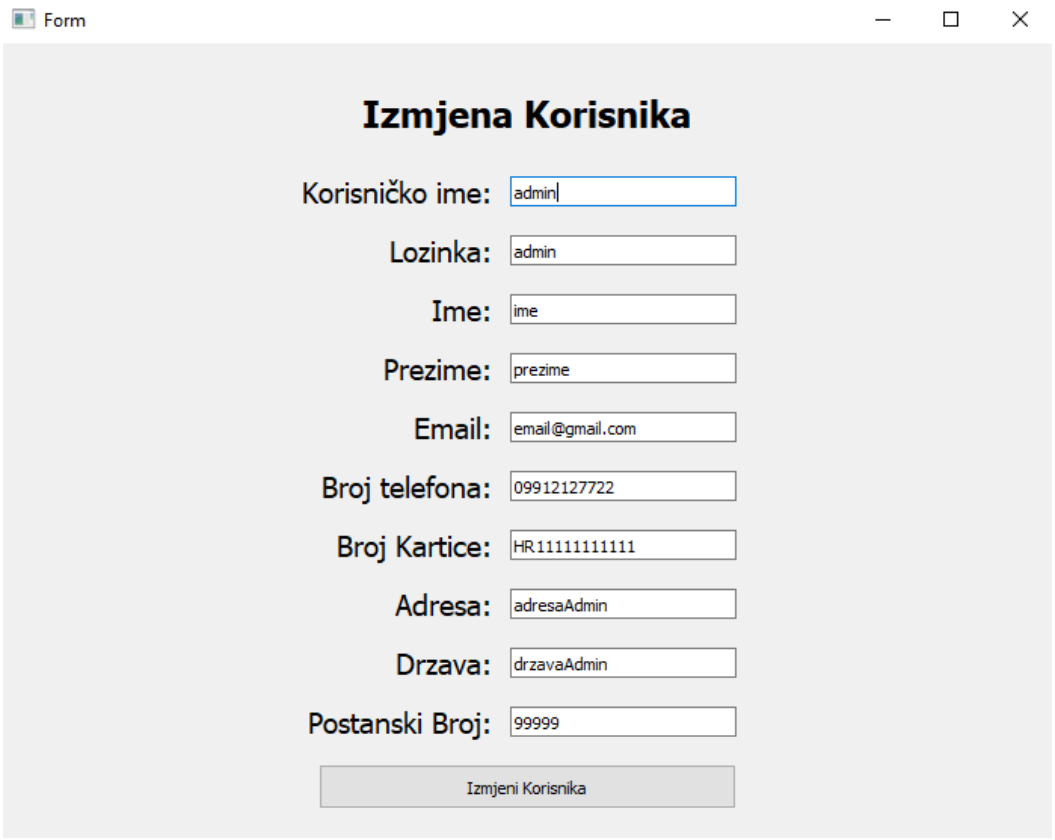
Na sljedećoj formi administrator vidi popis svih registriranih korisnika i sve njihove podatke. Ukoliko administrator želi izbrisati nekog korisnika mora ga odabrati na listi i pritisnuti gumb „Obriši“.

```
def obrisiKorisnika(self):
    korisnik = Korisnik.Korisnik()
    lista = self.dohvatiKorisnike()
    korisnik.obrisiKorisnika(int(lista[0]))
    self.ispisKorisinka()
```

```
def obrisiKorisnika(self, korisnikID):
    upit = "UPDATE korisnik set korisnikIzbrisan = 1 WHERE IDkorisnik =
"+str(korisnikID)+";"
    self.konekcijaNaBazu.Izbrisi(upit)
```

Svaki podatak se ne briše direktno iz baze podataka već ostaje radi evidencije i radi integriteta baze podataka pa je iz tog razloga kod korisnika dodan stupac „korisnikIzbrisan“ koji se postavlja na 1, ako su podaci „izbrisani“.

Administrator ima ovlasti izmjene podataka o korisnicima. On može izmijeniti podatke o korisniku tako što klikom na gumb „Izmjeni“ otvara sljedeću formu:



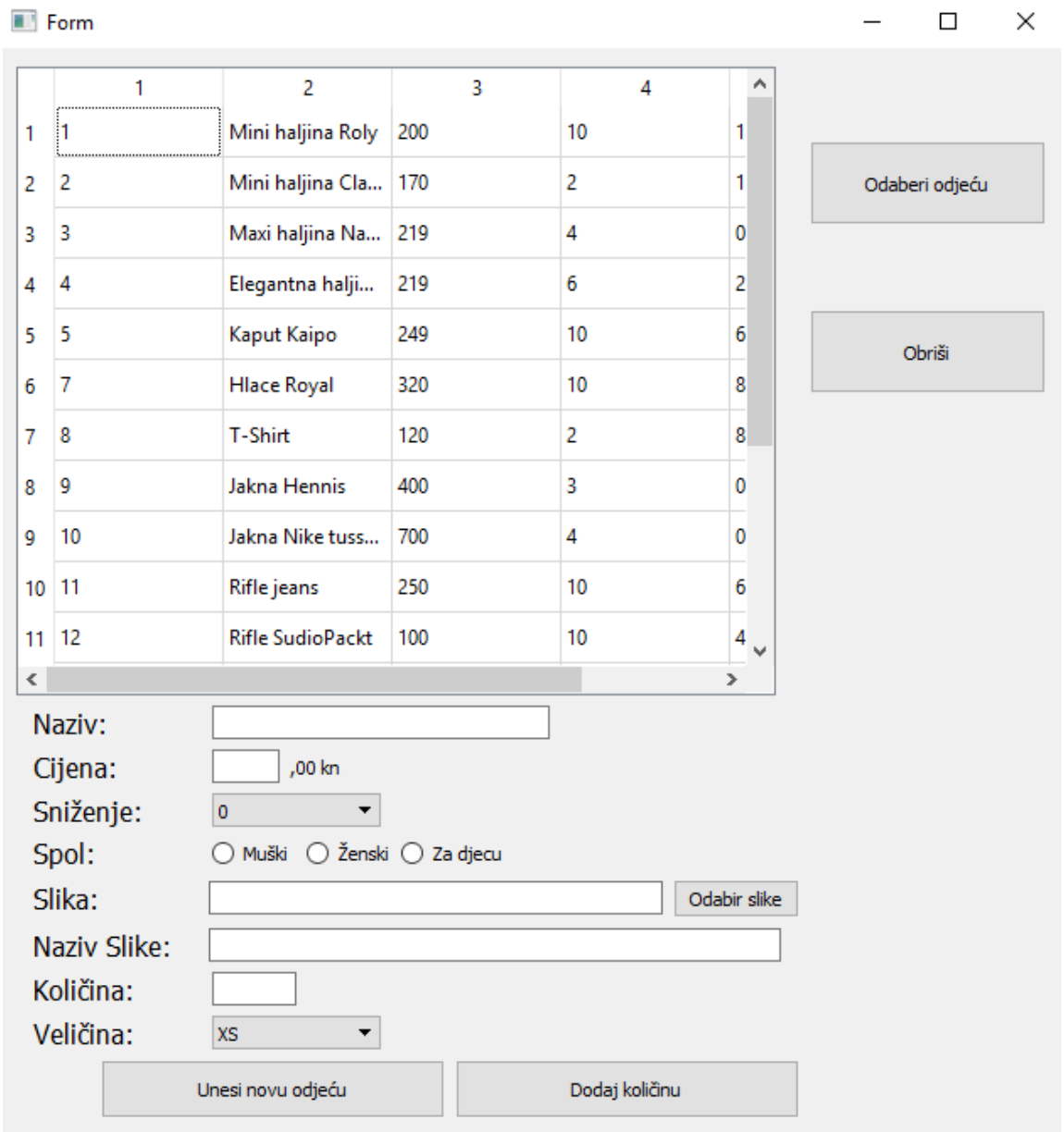
Slika 11. Izmjena korisnika

Na slici 11. prikazana je forma za izmjenu korisnika gdje su u tekstualno polje upisani podaci odabranog korisnika za izmjenu. Nakon što administrator izmjeni željene podatke oni se spremaju u bazu i forma se zatvara.

```
def izmjeniKorisnika(self, korisnikID, korisnickoIme, lozinka, ime,
prezime, email, brojTelefona, brojKartice, adresa, drzava, postanskiBroj):

  upit = 'UPDATE korisnik SET korisnickoIme ='+'"'+str(korisnickoIme)+'",'\
       + 'lozinka = '+'"'+str(lozinka)+'",'\
       + 'ime = '+'"'+str(ime)+'",'\
       + 'prezime = '+'"'+str(prezime)+'",'\
       + 'email = '+'"'+str(email)+'",'\
       + 'brojTelefona = '+'"'+str(brojTelefona)+'",'\
       + 'brojKartice = '+'"'+str(brojKartice)+'",'\
       + 'adresa = '+'"'+str(adresa)+'",'\
       + 'drzava = '+'"'+str(drzava)+'",'\
       + 'postanskiBroj = '+'"'+str(postanskiBroj)+'" ' + ' WHERE
IDkorisnik = '+str(korisnikID)+';'
    self.konekcijaNaBazu.UnesiPodatke(upit)
```

7.3.5. Administracija odjeće

Administrator također ima mogućnost dodavanja nove odjeće u bazu podataka. Klikom na gumb odjeća administratoru se otvara sljedeća forma:

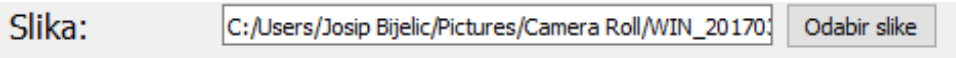


Slika 12. Administracija odjeće

U tablici s podacima o odjeći na slici 12, vidljivi su svi podaci o odjeći uključujući ukupnu količinu raspoloživu na skladištu. Administrator unosi novu odjeću tako što unosi sve potrebne podatke kao što su naziv, cijena i količina te odabire spol i popust. Da bi administrator odabrao sliku treba kliknuti na gumb „Odabir slike“ gdje odabire sliku s računala. Naziv slike označava naziv same slike koja se sprema u datoteku sa slikama. Nakon odabira slike administrator odabire gumb „Unesi novu odjeću“.

```
def odabirSike(self):
    putanjaDoSlike = QtWidgets.QFileDialog.getOpenFileName(self, 'Picture',
'', '*.png *.jpeg *.jpg')
    self.lineEditSlika.setText(putanjaDoSlike[0])

def unesiOdjecu(self):

    odjeca = Odjeca.Odjeca()

    naziv = self.lineEditNaziv.text()
    cijena = self.lineEditCijena.text()
    kolicina = self.lineEditKolicina.text()
    snizenje = self.comboBoxSnizenje.currentText()
    velicina = self.comboBoxVelicina.currentText()
    spol = self.odaberiSpol()
    snizenjeID = self.izracunajSnizenjeID(snizenje)
    velicinaID = self.odaberiVelicinu(velicina)

    if self.lineEditSlika.text():
        prethodnaPutanja = self.lineEditSlika.text()
        datoteka, ekstenzija = os.path.splitext(prethodnaPutanja)
        novaPutanja = os.path.dirname(__file__)
        novaPutanja = os.path.join(novaPutanja, 'Slikeodjece' + '/' +
self.lineEditNazivSlike.text() + ekstenzija)
        slikaBaza = 'Slikeodjece' + '/' + self.lineEditNazivSlike.text() +
ekstenzija
        shutil.move(prethodnaPutanja, novaPutanja)
    else:
        slikaBaza = " "
    odjeca.unesiOdjecu(cijena, naziv, snizenjeID, spol, slikaBaza ,
velicinaID, kolicina)
    self.ispisOdjeca()
```

Nakon što korisnik odabere sliku, putanja do slike se ispisuje u tekstualni okvir pokraj gumba za odabir slike. Ovako izgleda primjer ako odaberemo neku sliku iz datoteke „Pictures“ na „C“ particiji diska.

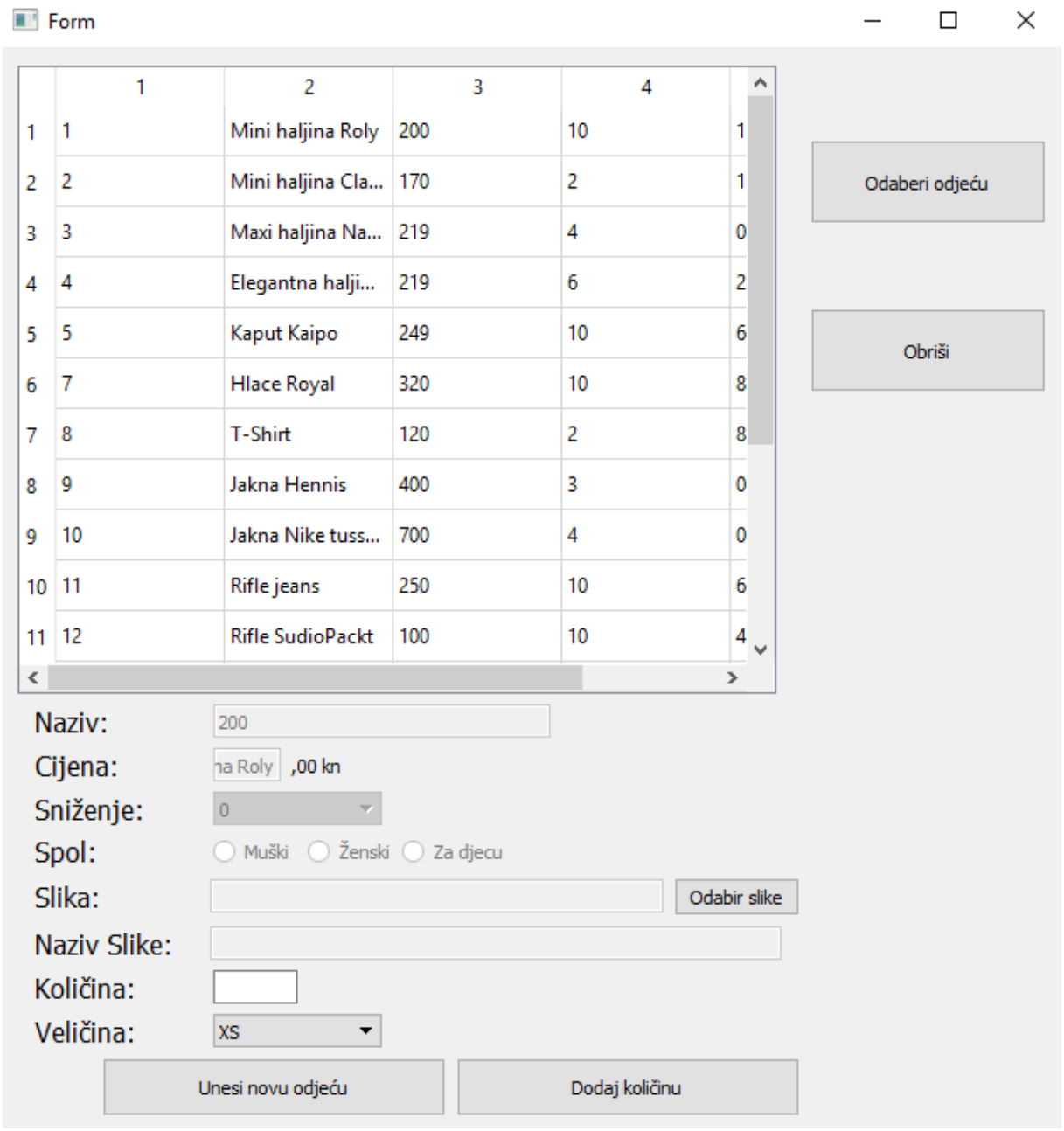


Slika 13. Odabir slike

Metoda klase QFileDialog getOpenFileName() vraća putanju do lokacije gdje se preuzeta slika nalazi. Kao što se vidi na slici 13, datoteke su odijeljene uzlaznom kosom crtom(eng. *front slash*). Ovdje je zanimljivo istaknuti kako ova funkcionalnost radi unatoč tome što Windows koristi silaznu kosu crtu(eng. *back slash*) jer na novijim Windowsima možemo koristiti oba znaka kao što je navedeno na službenim stranicama Microsoft-a:

I/O funkcije sadržane u Windows API-u konvertiraju znak „/“ u „\“ tijekom konvertiranja naziva datoteke u „NT-style name“ [12].

Ako administrator želi dodati samo količinu za već postojeću odjeću, to čini tako što odabere odjeću klikom na gumb „Odaberi odjeću“. Nakon prethodne aktivnosti forma izgleda ovako:



Slika 13. Dodavanje količine

Kao što se vidi na slici 13, svi elementi forme su postavljeni na neaktivan(eng. disabled) osim količine i veličine. Administrator tada može unijeti proizvoljnu količinu i odabrati veličinu za već postojeću odjeću. Na kraju administrator unos treba potvrditi klikom na gumb „Dodaj količinu“ i količina se sprema u bazu podataka.

8. Zaključak

Programiranje multiplatformskih aplikacije je mukotrpan i težak posao. Programer mora od samog početka razvoja do implementacije i testiranja paziti na svaki detalj koji bi mogao utjecati na portabilnost programskog koda. Iako se čini neisplativo, današnje aplikacije sve više traže svojstvo portabilnosti radi proširenja tržišta i ostalih pogodnosti.

Prije nego što se započne pisati aplikacija namijenjena za više operacijskih sustava potrebno je detaljno analizirati svaki operacijski sustav , njegova ograničenja i slično. Na sreću razvili su se brojni moderni alati kao što su Haxe, Qt, Electron koji programeru olakšavaju izradu portabilnog koda. Iako postoje alati koji generiraju kod „instant“ portabilan treba biti oprezan jer se ne mora samo u kodu kriti pogreška pri razvoju aplikacije.

U ovom radu se može vidjeti koliko je jednostavno raditi multiplatformsku aplikaciju s grafičkim korisničkim sučeljem kad je u pitanju Pyhton i PyQt biblioteka. Python ima isti interpreter na većini operacijskih sustava a PyQt biblioteke omogućuju grafičko korisničko sučelje. Aplikacija je pokrenuta na 2 operacijska sustava. Prvi je Windows 10, a drugi Ubuntu 16.04. Na oba operacijska sustava aplikacija radi identično, jedine male razlike su u dizajnu i izgledu same aplikacije što je za ovaj rad zanemarivo jer je ovdje bio naglasak na samu funkcionalnost aplikacije.

Popis literature

[1] Hook .B (2005) *Write Portable Code : An Introduction to Developing Software for Multiple Platforms* (No Starch Press)

[2] Rischpater Ray (2013) *Application Development with Qt Creator*. Pact Publishing Ltd.

[3] Summerfield Mark (2007) *Rapid GUI Programming with Python and Qt*

[4] PyQt5 Reference Guide(2017) http://pyqt.sourceforge.net/Docs/PyQt5/

[5] Haxe The cross-platform toolkit(2017) https://haxe.org/#learn-more

[6] Haxe Compiler – Haxe(2017) http://old.haxe.org/doc/compiler

[7] Qt Documentation http://doc.qt.io/

[8] MySQLdb User's Guide http://mysql-python.sourceforge.net/MySQLdb.html

[9] B4X – Wikipedia https://en.wikipedia.org/wiki/B4X

[10] About Electron | Electron https://electron.atom.io/docs/tutorial/about/

[11] Christian Engvall, *Electron Hello World 06/08/2016* https://www.christianengvall.se/electron-hello-world/

[12] Naming Files, Paths, and Namespaces (Windows) *2017* https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx

Popis slika.

Popis tablica.

Prilog

Cijeli programski kod nalazi se na GitHub repozitoriju kojem možete pristupiti na sljedećem linku:
https://github.com/belivk1982/ShoppingApp