Sustav za upravljanje medicinskim snimkama

Koščak, Kristijan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni preddiplomski studij računarstva

SUSTAV ZA UPRAVLJANJE MEDICINSKIM SNIMKAMA

Završni rad

Kristijan Koščak

Osijek,2019

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 21.09.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Kristijan Koščak
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3939, 26.09.2018.
OIB studenta:	90363650383
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Naslov završnog rada:	Sustav za upravljanje medicinskim snimkama
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2019.
Datum potvrde ocjene Odbora:	25.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2019.

Ime i prezime studenta:	Kristijan Koščak
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3939, 26.09.2018.
Ephorus podudaranje [%]:	3%

Ovom izjavom izjavljujem da je rad pod nazivom**: Sustav za upravljanje medicinskim snimkama**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. POSTOJEĆI FORMATI ZAPISA MEDICINSKIH SNIMKI 2

2.1. Općenito o medicinskim snimkama 2

2.2. DICOM (Digital Imaging and Communications in Medicine) 4

2.3. MetaImage (mha i mhd ekstenzije) 6

2.4. VTK file format 7

3. GRAFIČKA SUČELJA U PYTHONU 9

3.1. Tkinter 9

3.2. PyQt 10

3.3. Kivy 11

4. DIZAJN APLIKACIJE 14

4.1. Prezentacijski dio 14

4.2. Logički dio 19

4.3. ER dijagram baza 28

5. ZAKLJUČAK 29

LITERATURA 30

SAŽETAK 32

ABSTRACT 33

ŽIVOTOPIS 34

1. UVOD

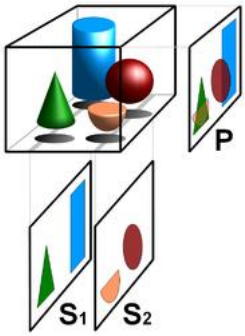
Razvoj računalne tehnologije i mrežne komunikacije unaprijedio je dokumentiranje medicinskih snimaka u digitalnom obliku. Samim time, snimke su postale pristupačnije ,preglednije i dugoročnije[1]. Lakši pristup, spremanje bez oštećenja i kasnije korištenje samo su neke od prednosti koje donose ovakve snimke. Medicinske snimke rezultat su nekoliko različitih tehnologija(X-zrake, *CT*, *MRI*, ultra-zvuk) koje se koriste za prikaz ljudskog tijela s ciljem dijagnoze, nadziranja ili poduzimanja određene akcije vezane za medicinsko stanje. Svaka od tehnologija daje različite informacije o području tijela koje se proučava ili o kojem se poduzimaju nekakve akcije. Takve informacije su dobivene iz podataka na pojedinim uređajima koje se sistematiziraju korištenjem odgovarajućeg softvera. Softver razvijaju tvrtke za medicinske ustanove te ga prilagođavaju njihovim potrebama. Velik broj naprednih funkcionalnosti u takvim softverima za sobom povlači obrazovanje osoblja za rukovanje takvim softverima.

2. POSTOJEĆI FORMATI ZAPISA MEDICINSKIH SNIMKI

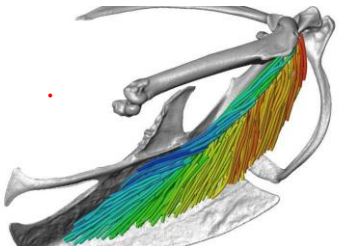
U ovome poglavlju će se spomenuti medicinske snimke te objasniti pojedini pojmovi koji su bitni za sve snimke. Nakon toga će se spomenuti nekoliko formata medicinskih snimaka koji se danas upotrebljavaju širom svijeta u medicinskim ustanovama.

2.1. Općenito o medicinskim snimkama

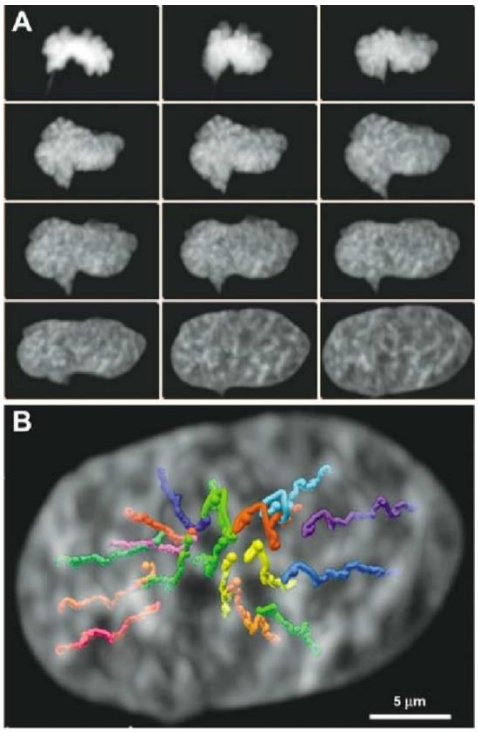
Primjena računalnih programa u medicini dovela je do velikog opsega zapisa ili formata medicinskih snimaka. Nastaju kao rezultat stvaranja vizualnog prikaza unutrašnjosti tijela[2]. Formati slikovnih datoteka pružaju jednostavan način za pohranu informacija opisujući sliku kao računalnu datoteku. Medicinska snimka sastoji se od jedne ili više slika koje predstavljaju projekciju volumena na ravninu slike što se još naziva projekcija ili planarna slika (prikazano „P“ oznakom na slici 2.1.), serije slika koje predstavljaju tanke slojeve ili kriške kroz volumen (tomografska ili višeslojna dvodimenzionalna slika) što se vidi na slici 2.1. oznakama „S1“ i „S2“ ,skup podataka iz volumena (volumen ili 3D slikanje) što je vidljivo na slici 2.2. ili višestruko snimanje iste tomografske ili volumne slike tijekom vremena da bi se proizveo dinamični niz što se još naziva i 4D slika(Sl. 2.3.). Format datoteke opisuje organizaciju podataka unutar slike i kako se pikseli trebaju interpretirati pomoću softvera za ispravno učitavanje i vizualizaciju.



Sl. 2.1. Prikaz planarne slike (P) i tomografskih slika(S1,S2). Izvor: [3]



Sl. 2.2. Prikaz Volumena. Izvor: [4]



Sl. 2.3. Prikaz 4D slike. Izvor: [5]

Medicinska snimka predstavlja unutarnju strukturu u obliku niza elemenata koji se nazivaju pikseli. Broj piksela od kojih je sastavljena slika definiran je razlučivošću. Izražavanje piksela ovisi o snimanju, protokolu prikupljanja, rekonstrukciji i naknadnoj obradi. Svi formati snimaka imaju zajedničke pojmove kao što su dubina piksela, fotometrijska interpretacija, meta-podaci i podaci piksela[2].

Dubina piksela je broj bitova koji se koriste za kodiranje informacija svakog piksela. Svaka se slika pohranjuje u datoteku i čuva u memoriji računala kao grupa bajtova. Bajtovi su skupina od 8 bitova i predstavljaju najmanju količinu koja se može pohraniti u memoriju računala.

Fotometrijska interpretacija određuje kako se podaci piksela trebaju tumačiti za ispravan prikaz slike kao jednobojna ili slika u boji. Da bi se odredilo jesu li informacije o boji ili nisu pohranjene u vrijednostima piksela slike, uvodi se koncept uzoraka po pikselu. Jednobojne slike imaju po jedan uzorak po pikselu i nema podataka o boji pohranjenih na slici. Za prikaz slika koristi se ljestvica sivih tonova od crne do bijele. Slike nuklearne medicine, kao što je pozitronska emisijska tomografija (PET) i jednofotonska emisijska tomografija (engl. *SPECT*), obično se prikazuju mapom boja ili paletom boja. U tom slučaju, svaki piksel slike je povezan s bojom u unaprijed definiranoj karti boja, ali boja se odnosi samo na prikaz te je informacija koja je povezana s vrijednostima piksela i nije stvarno pohranjena u njima.

Metapodaci su informacije koje opisuju sliku. Bez obzira na format datoteke uvijek postoji informacija povezana sa slikom izvan podataka o pikselima. Te se informacije obično pohranjuju na početku datoteke kao zaglavlje i sadrže barem dimenzije matrice slike, prostornu razlučivost, dubinu piksela i fotometrijsku interpretaciju. Zahvaljujući metapodacima, softverska aplikacija može prepoznati i ispravno otvoriti sliku u podržanom formatu datoteke. U slučaju medicinskih slika, metapodaci imaju širu ulogu. Slike koje dolaze iz dijagnostičkih liječenja obično imaju informacije o tome kako je slika nastala. Slika magnetske rezonance imat će parametre koji se odnose informacije o vremenu, kutu zakretanja, pacijentu itd. Metapodaci su korisni za bilježenje i iskorištavanje informacija vezanih uz slike u kliničke i istraživačke svrhe te za organiziranje i dohvaćanje arhivskih slika i povezanih podataka.

Podaci piksela su odjeljak koji sadrži numeričke vrijednosti piksela. Prema tipu podatka , podaci piksela se pohranjuju kao cijeli brojevi ili brojevi s pomičnim zarezom. Slike generirane tomografskim načinom snimanja, radiološke slike kao *CT* i *MR* pohranjuju 16 bita za svaki piksel kao cijeli broj.

Podaci piksela se nalaze na fiksnoj poziciji nakon preskakanja duljine zaglavlja kod formata s zaglavljem fiksne veličine. Kod formata s zaglavljem promjenjive duljine, početna lokacija podataka piksela označena je oznakom ili pokazivačem. Veličina podataka piksela računa se kao umnožak broja redova, broja kolona i dubine piksela(u bajtovima).

$$Broj\ redova\ \cdot Broj\ kolona\ \cdot Dubina\ piksela\ (\cdot Broj\ okvira)$$

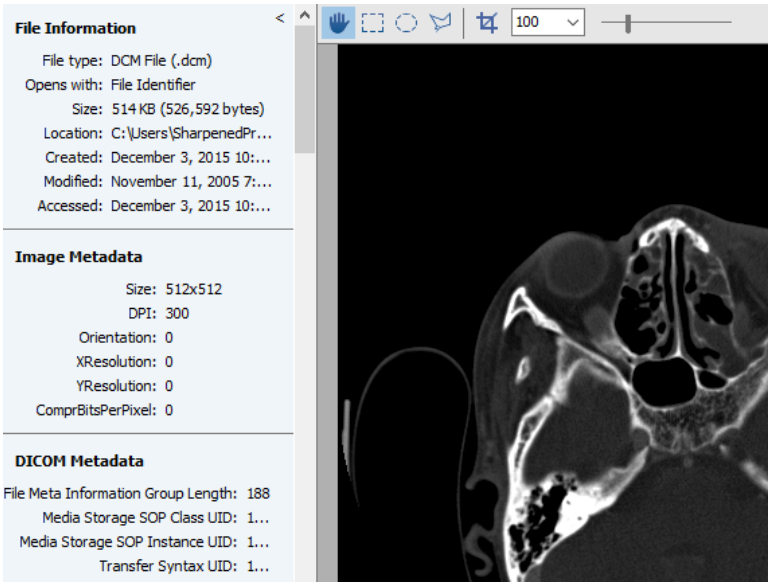
Konačno veličina datoteke slike je zbroj veličine zaglavlja i veličine podataka piksela.

$$Veličina\ zaglavlja + veličina\ podataka\ piksela$$

Oba izraza vrijede u slučaju nekomprimiranih podataka. Slikovni podaci mogu se također komprimirati kako bi se smanjili zahtjevi za pohranu i prijenos, pri čemu se veličina datoteke smanjuje za faktor koji ovisi o usvojenoj tehnici kompresije. Općenito govoreći, kompresija može biti reverzibilna (bez gubitaka) ili nepovratna (gubitak).

2.2. DICOM (engl. *Digital Imaging and Communications in Medicine*)

Prilikom slanja, primanja, spremanja, obrađivanja i prikazivanja medicinskih snimaka, koristi se međunarodni standard za medicinske snimke koji se naziva DICOM [6]. Definira formate za medicinske snimke koje se mogu razmjenjivati u klinikama. Upotreba DICOM-a je velika. Može se pronaći na snimkama u radiologiji, kardiologiji, uređajima za radioterapiju te uređajima za oftalmologiju i stomatologiju. DICOM funkcionira na način da grupira podatke u skupove podataka npr. datoteka rendgenske snimke prsnog koša sadrži id, ime, spol, dob, težinu i visinu pacijenta unutar datoteke što je vidljivo primjerom na slici 2.5. Ukoliko se pojavi greška te informacije se ne mogu odvojiti od slike. Na sličan način JPEG slike mogu imati ugrađene oznake za prepoznavanje i opisivanje slike. Osim navedenih atributa sadrži i atribut koji se odnosi na podatke piksela slike. Jedan objekt može imati samo jedan atribut koji sadrži podatke piksela i to odgovara jednoj slici [2]. Međutim atribut može imati višestruke okvire (engl. *frame*) koji omogućuju pohranu niza digitalnih slika s ultrazvučnog pregleda(cine-petlje) ili druge podatke s više okvira. U slučaju višedimenzionalnih slika, trodimenzionalni i četverodimenzionalni podaci mogu biti učahureni u jedan objekt. Za prikaz vrijednosti niza znakova, ako je kodirano više podatkovnih elemenata, sljedeći podatkovni element razdvojen je s kosom crtom („\“). Koristeći razne standarde možemo komprimirati podatke piksela, ali i cijeli skup podataka(rijetko se provodi). Sve aplikacije koriste isti osnovni format, uključujući korištenje mreže i datoteke. Kada se zapiše u datoteku dodaje se pravo zaglavlje koje sadrži kopije nekoliko ključnih atributa i detalja o aplikaciji.



Sl. 2.4. Prikaz DICOM slike s metapodacima i informacijama o datoteci. Izvor: [7]

Sl. 2.5. Primjer informacija o pacijentu(ime,spol,itd…) Izvor: [8]

Koncept DICOM snimaka leži u tome da je snimka povezana sa svojim metapodacima, neki od njih su vidljivi na slici 2.4.(lijevi stupac, središnji dio). Metapodaci i podaci piksela su spojeni u jedinstvenu datoteku, a zaglavlje osim informacija o slici sadrži i podatke o samome pacijentu kao što je, već ranije spomenuto. Stoga veličina zaglavlja ovisi o tim parametrima. Podaci o pikselima, mogu pohraniti vrijednosti piksela samo kao cijele brojeve. DICOM trenutno ne može spremiti podatke piksela u pomičnim zarezima dok podržava različite vrste podataka, uključujući i plutajuće podatke, za pohranu metapodataka. Za pregled snimaka mogu se koristiti software-i kao što su *DICOM File Viewer*, *Free DICOM Viewer*, *Sante DICOM Viewer*, *MicroDicom* itd. Mogu biti spremljene na CD-u,DVD-u ili USB-u [9]. Također mogu se ispisivati pomoću pisača u obliku filma X-zraka. DICOM datoteke sadrže ekstenziju .dcm dok je sam naziv datoteke ograničen na 8 znakova. Unutar datoteke nalazi se slika sa podacima o slici te podacima o pacijentu i podacima o pregledu(datum pregleda, instanca studija itd.). Svi ti podaci u datoteci su zapisani prema DICOM standardu koji sadrži znakove kao što su otvorena i zatvorena zagrada te zarez [10]. Unutar zagrada nalaze se dva četveroznamenkasta heksadecimalna broja koji predstavljaju oznaku (engl. *tag*). Prvi četveroznamenkasti broj se odnosi na oznaku grupe dok je drugi oznaka elementa. Primjeri nekih oznaka i njihovih značenja prikazana su tablicom 2.1. Svaka ta oznaka posjeduje svoj VR(engl. *value representation*) koji predstavlja prikaz vrijednosti u određenom obliku.

Tab. 2.1. Prikaz nekih od najčešće korištenih oznaka. Izvor: [11]

Tag(oznaka)	Značenje
(0008,0020)	Datum pregleda
(0008,0030)	Vrijeme pregleda
(0010,0020)	ID pacijenta
(0020,0010)	ID pregleda
(0020,0011)	Serijski broj
(0008,0050)	Datum pristupa
(0008,0090)	Ime liječnika
(0010,0030)	Datum rođenja pacijenta
(0010,0040)	Spol pacijenta
(0020,0013)	Broj instance
(0020,000D)	Instanca studija
(0020,000E)	Instanca serije
(0002,0001)	Meta informacije o datoteci, verzija
(0002,0000)	Meta informacije o datoteci, dužina grupe
(0002,0013)	Naziv verzije implementacije
(0008,1030)	Opis pregleda
(0009,1010)	Komentar

2.3. MetaImage (mha i mhd ekstenzije)

MetaImage je tekstualni format datoteke s oznakama (engl. *tag*) za medicinske snimke [12]. Format datoteke je proširen radi podržavanja različitih objekata koji se pojavljuju u medicini kao što su cijevi za posude i igle itd [13]. Biblioteka koja sadržava to se naziva MetaIO. Nekoliko godina se, već široko primjenjuje za istraživanja zbog prilično stabilnog središnjeg koda. MHD i MHA su ekstenzije datoteke koje se odnose na MetaImage medicinski slikovni format koji se koristi u softveru ITK (engl. *Insight Toolkit*). ITK je softverski alat otvorenog koda za slikovnu analizu [14]. Obavlja dvije operacije, registraciju i segmentaciju. Segmentacija je proces identificiranja i klasificiranja podataka koji se nalaze u digitalnom prikazu kao što je slika dobivena iz medicinskih aparata kao što su *CT,MRI* ili ultra-zvučni skeneri. Registracija je postupak usklađivanja podudaranosti između podataka. ITK je implementiran u *C++* programskom jeziku.

Razlika između MHA i MHD ekstenzije je u tome što datoteka s MHA ekstenzijom sadrži zaglavlje i podatke slike u istoj datoteci dok datoteke s MHD ekstenzijom sadrže samo zaglavlje i poveznicu na datoteku s RAW ekstenzijom koja sadrži podatke o slici.[15]

Podaci u MetaImage formatu mogu biti kompresirani i nekompresirani. Pretvaranje nekompresiranih podataka u MetaImage format ovisi o specificiranju zaglavne datoteke koja opisuje i upućuje na datoteku koja sadrži naše podatke. Nekompresirani podaci su podaci u neobrađenom formatu, a mogu biti s zaglavljem(DICOM,PNG, itd..) Kod kompresiranih podataka prvo ih moramo pretvoriti u nekompresirani format. Koristeći ITK softverski alat stvara se MetaImage zaglavna datoteka koja upućuje na našu datoteku. Komadići bajtova (engl. *brick of bytes*) su volumen slikovnih podataka pohranjenih u jednoj datoteci. Volumen može biti bilo koje dimenzije. Na slici 2.6. vidimo RAW datoteku koja sadrži podatke o slici kao što su dimenzija, veličina dimenzija, vrsta podataka te sam naziv datoteke.

```
ObjectType = Image
NDims = 3
DimSize = 256 256 64
ElementType = MET_USHORT
ElementDataFile = image.raw     (this tag must be last in a MetaImageHeader)
```

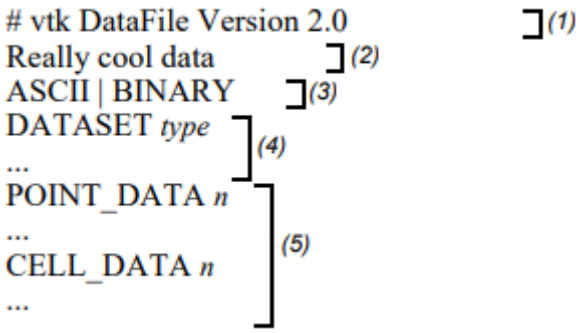
Sl. 2.6. Podaci o slici. Izvor: [14]

2.4. VTK file format

VTK(engl. *Visualization Toolkit*) je biblioteka velike primjene. Koristi se za *3D* računalnu grafiku, modeliranje, procesiranje slika, prikazivanje volumena, znanstvenu vizualizaciju i *2D* crtanje [16]. Cilj stvaranja je neovisnost o platformi, tj. pokretanja na bilo kojoj platformi (*Linux, Windows, Mac, Web*, mobilni uređaji). Učinkovitost je povećana tako što je osnovna funkcionalnost VTK biblioteke pisana u *C++*. Neke od značajki su filteri, podatkovni model, interakcija među podacima, *2D* crtanje itd. Filteri manipuliraju podacima. Primljeni podaci se pregledavaju i od njih svaki filter stvara izvedene podatke. Međusobno povezani filteri stvaraju mrežu podataka koja pretvara sirove podatke u vizualno razumljive formate. Podatkovni model ima mogućnost predstavljanja svakog problema iz stvarnog svijeta u području fizičke znanosti. Temeljne strukture podataka posebno su prikladne za medicinsko snimanje koji uključuje razlike i rješenja. Interakcija među podacima pomaže nam razumjeti sadržaj, oblik i značenje podataka. VTK ima cijeli skup *2D* dijagrama i vrsta grafikona za tablične podatke.

VTK formati datoteka sastoje se od 5 osnovnih dijelova: [17]

1. Prvi dio odnosi se na identifikaciju i verziju datoteke. Primjer izgleda prvog dijela: „# vtk DataFile Version x.x“
2. Ovaj dio je zaglavlje. Zaglavlje se sastoji od niza znakova koji završava znakom „\n“ koji označava kraj linije. Može biti maksimalno 256 znakova te se može koristiti za opis podataka ili uključiti neke potrebne informacije.
3. Treći dio predstavlja format datoteke odnosno opisivanje vrste datoteke. Može biti ASCII(„ASCII“) ili binarni(„BINARY“) tip.
4. Ovim dijelom predstavljena je struktura skupa podataka. Geometrijski dio opisuje topologiju i geometriju skupa podataka. Započinje sa linijom koja sadrži ključnu riječ „DATASET“ te riječ koja opisuje vrstu skupa podataka. Ovisno o vrsti skupa podataka ostale kombinacije ključnih riječi /podataka definiraju stvarne podatke
5. Peti dio opisuje atribute skupa podataka. Započinje s ključnom riječi „POINT_DATA“ ili „CELL_DATA“ te sadrži cijeli broj koji označava broj točaka ili ćelija. Ostale kombinacije ključnih riječi/podataka definiraju stvarne vrijednosti atributa skupa podataka.



Sl. 2.7. Primjer VTK file formata. Izvor: [17]

Pregled dijelova datoteke ovoga formata prikazan je na slici 2.7. Prva tri dijela su obavezni dok su druga dva neobavezni. Time je omogućena fleksibilnost kombiniranja i podudaranja atributa skupa podataka i geometrije.

3. GRAFIČKA SUČELJA U PYTHONU

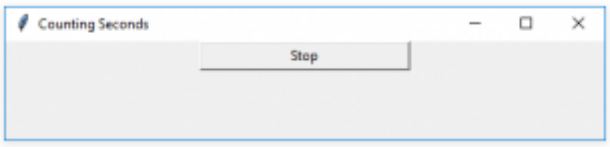
Ovo poglavlje opisuje neke od grafičkih sučelja koje se koristi u kombinaciji s programskim jezikom *Python*-om. Navest će se prednosti i mane pojedinih te detaljnije opisati *Kivy* grafičko sučelje budući da je ono dio ovoga završnog rada. Radi lakšeg razumijevanja potrebno je objasniti neke od pojmova koji će se koristiti. *Widget* je mali grafički element. Međusobnim povezivanjem više takvih, nastaju korisnička sučelja. Neki od grafičkih elemenata koji se spominju su tekstualna oznaka(engl. *label*) koji omogućava prikaz teksta na ekranu, dugme (engl. button) koji može služiti kao okidač za izvršenje neke radnje, polje za unos teksta (engl. *textbox*) itd.

3.1. Tkinter

Tk i Tcl su sastavni dijelovi *Python*-a. Predstavljaju snažni i platformski neovisni alat za izgradnju sučelja koji je dostupan programerima kroz Tkinter paket. Tcl (engl. *Tool Command Language* ili komandni jezični alat) je dinamički programski jezik prilagodljiv za svakakvu upotrebu od računalnih do internetskih aplikacija [18]. Tk je grafičko korisničko sučelje za mnoge dinamičke programske jezike koje se koristi za razvoj računalnih aplikacija. Tkinter paket je usko objektno orijentirani sloj koji leži na Tk/Tcl-u. Za korištenje ne treba pisati Tcl kod, već proučiti dokumentaciju Tk-a, a po potrebi i Tcl dokumentaciju [19]. Predstavlja set omotača koji implementiraju Tk grafičke elemente kao *Python* klase. Unutrašnji modul _tkinter pruža mehanizam za zaštitu koji omogućava interakciju između Python-a i Tcl-a. Glavna prednost su mu brzina i to što obično dolazi zajedno s Pythonom. Materijali su lako dostupni, uključujući tutorijale, knjige i sl. Za kreiranje Tkinter-a potrebno je uključiti modul tkinter, kreirati glavni prozor, dodati grafički element te na njega zakačiti okidač za nekakav događaj. Na slici 3.1. vidljivo je pozivanje tkinter-a s komandom import te se kreira glavni prozor odmah ispod. Naslov prozora se mijenja, dodaje se dugme (engl. *button*) te ga se oblikuje i postavlja okidač za event koji gasi prozor. Komandom pack dugme se organizira u blok. Na slici 3.2. vidljiv je rezultat pokretanja aplikacije.

```
1 import tkinter as tk
2 r = tk.Tk()
3 r.title('Counting Seconds')
4 button = tk.Button(r, text='Stop', width=25, command=r.destroy)
5 button.pack()
6 r.mainloop()
7
```

Sl. 3.1. Prikaz tkinter koda. Izvor: [20]



Sl. 3.2. Prikaz sučelja. Izvor: [20]

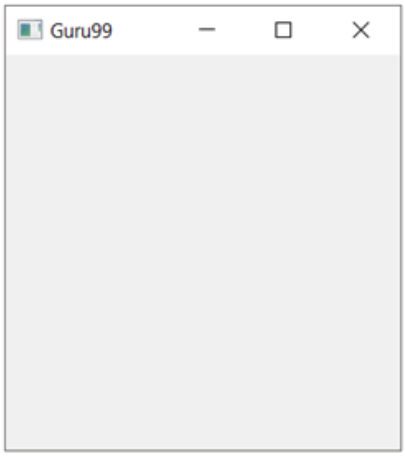
3.2. PyQt

PyQt je skup *Python* v2 i v3 povezivanja za Qt aplikacijsku razvojnu cjelinu firme Qt. Pokreće se na svim platformama uključujući *Windows, OS X, Linux, iOS* i *Android* [21]. Povezivanja su implementirana kao skup *Python* modula i sadrže više od tisuću klasa. PyQt4 je starija verzija stoga više ne pružaju podršku za istu niti će biti novijih izdanja. Preporuča se korištenje PyQt5 za bilo kakav razvoj aplikacija. PyQt također predstavlja opsežnu *C++* razvojnu cjelinu za razvoj aplikacija. Da bi to bilo moguće koristi se tzv. „sip“ koji predstavlja alat za stvaranje *C++* biblioteka kao *Python* klasa. Osim što je grafičko korisničko sučelje, omogućava i korištenje *Unicodea, SQL* baze podatka, *XML,* potpuno funkcionalan internetski preglednik, multimedijsku razvojnu cjelinu kao i mnogo grafičkih elemenata. Posjeduje Qt dizajnera koji predstavlja grafičko korisničko dizajnersko sučelje iz kojega *Python* može generirati kod. Na slici 3.3. vidljiv je kod koji predstavlja realizaciju grafičkog korisničkog sučelja u PyQt-u.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
if __name__ == "__main__":
    app = QApplication(sys.argv)
    w = QWidget()
    w.resize(300,300)
    w.setWindowTitle('Guru99')
    w.show()
    sys.exit(app.exec_())
```

Sl. 3.3. Prikaz PyQt5 koda. Izvor: [22]

Uključuje se Qapplication i QWidget iz biblioteke PyQt5.QtWidgets. Zatim glavni korak je kreiranje ulazne točke aplikacije, a to je objekt app. W predstavlja objekt klase QWidget koji je ujedno i prozor aplikacije. Zatim se postavlja veličina i naslov prozora te ga se prikazuje. Na slici 3.4 vidljiv je rezultat.



Sl. 3.4. Prikaz PyQt5 sučelja. Izvor: [22]

3.3. Kivy

Kivy je besplatna *Python* biblioteka otvorenog koda koja omogućava brzo i jednostavno razvijanje interaktivnih platformski neovisnih aplikacija [23]. Ima ogromne prednosti pokretanja na različitim platformama, kao *HTML5*. Razlika je ,što se izvodi bolje jer se ne oslanja na težak preglednik. Mnoge njegove komponente su implementirane u *C* programskom jeziku koristeći *Cython* biblioteku tako da se većina grafičke obrade odvija izravno na procesoru. Grafički pokretač je izrađen preko *OpenGL ES 2* tako da je procesor ubrzan. Bez obzira na hardversko i softversko okruženje postiže odličnu ravnotežu između prenosivosti i performansi. Pruža podršku za mnoge uređaje. Dizajniran je s ciljem što većeg fokusiranja na razvijanje interaktivnih i prilagođenih aplikacija što je lakše i brže moguće. Pokreće se na različitim platformama kao što je *Linux*, *Windows*, *OS X*, *Android*, *iOS* i *Raspberry Pi* [24]. Isti kod se može pokrenuti na svim podržavajućim platformama što je ujedno i ideja samog *Kivy*-a. Može koristiti većinu ulaza, protokola i uređaja uključujući *WM_Touch, WM_Pen, Mac OS X Trackpad* i *Magic Mouse, Mtdev, Linux Kernel HID, TUIO*. Razvojna cjelina posjeduje dobro dokumentirano grafičko sučelje za programiranje aplikacija(engl. *application programming interface*) kao i vodič kroz programiranje za pomoć pri započinjanju s radom. Sadrži više od dvadeset grafičkih elemenata kao što su tekstualna oznaka (engl. *label*), dugme (engl. *button*), polje za unos teksta (engl. *textbox*) itd. Kao što je napomenuto, mnogi dijelovi su napisani u *C* programskom jeziku pomoću *Cythona* te su ujedno i dobro testirani.

Kako bi se koristio potrebno ga je instalirati. *Kivy* razdvaja logički dio od same prezentacije aplikacije. Na taj način održava jednostavan kod. Osim što je zasebna biblioteka za *Python*, nudi svoj vlastiti jezik kako bi osim, već spomenutog i povezao elemente sučelja.

```
14. # File name: hello2.py
15. from kivy.app import App
16. from kivy.uix.button import Label
17.
18. class Hello2App(App):
19.     def build(self):
20.         return Label()
21.
22. if __name__=="__main__":
23.     Hello2App().run()
```

Sl. 3.5. Prikaz logičkog dijela. Izvor: [23]

Na slici 3.5. vidljiva je struktura koda u *Python*-u. Prvo što se primjećuje je imenovanje same *Python* datoteke, zatim se uključuje App iz kivy.app biblioteke što predstavlja prazni prozor gdje se dodaju druge *Kivy* komponente. Nasljeđivanje je bitna stavka objektno orijentiranog programiranja koja se koristi na liniji 18 što znači da sve metode i atribute koje posjeduje klasa App posjeduje i Hello2App. Sama klasa posjeduje metodu build(self) koja prilikom stvaranja vraća objekt klase Label koja je ujedno i grafički element.

```
24. # File name: hello2.kv
25. #:kivy 1.9.0
26. <Label>:
27.     text: 'Hello World!'
```

Sl. 3.6. Prikaz prezentacijskog dijela. Izvor: [23]

Svaki grafički element posjeduje svojstva ili atribute koji mu mogu promijeniti sadržaj, prikaz ili ponašanje. Slika 3.6. prikazuje jednu takvu promjenu gdje se tekstualnoj oznaci mijenja tekst. Slika 3.5. i 3.6. predstavljaju dvije zasebne datoteke, a to su hello2.py i hello2.kv. Bitno je naglasiti da ukoliko se želi stvoriti povezanost koja je ujedno i bitna ,da sve funkcionira ispravno, potrebno je da se klasa na liniji 18 (prije dijela App) identično zove kao *Kivy* datoteka tj. „hello2“. U suprotnom aplikacija će raditi, ali se na ekranu neće ispisivati „Hello World“. Slika 3.7. prikazuje rezultat ispravno povezanih *Python* i *Kivy* datoteka.

Sl. 3.7. Rezultat. Izvor: [23]

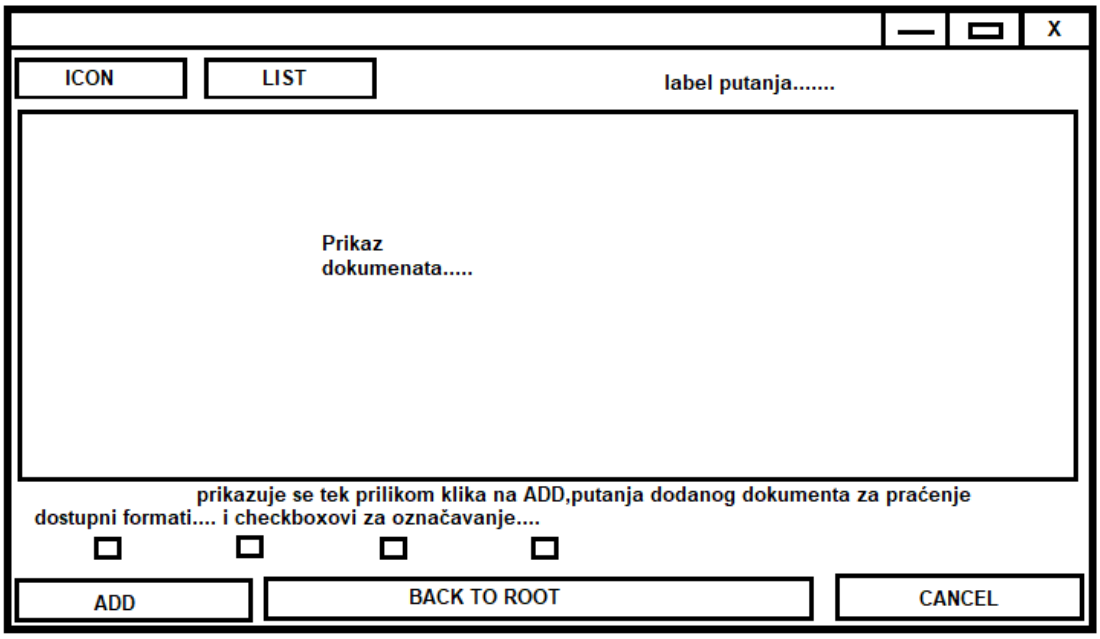
Instrukcije koje započinju sa „#:“ u zaglavlju *Kivy* datoteke zovu se smjernice ili direktive [23]. Na slici 3.6. navodi se verzija koja se koristi tj. 1.9.0. Linija 26. govori nam da će se izmijeniti klasa za prikaz tekstualnog sadržaja(label). *Kivy* jezik je izražen kao slijed pravila. Pravila su dijelovi koda koji definiraju sadržaj, ponašanje i prikaz *Kivy widget* klase. Pravilo uvijek započinje sa otvorenom šiljastom zagradom „<“ te slijedi ime i zatvorena šiljasta zagrada popraćena dvotočkom „>:“ Unutar pravila svojstvo teksta se postavlja na „Hello World!“. Ovakvo razvijanje aplikacije pruža mnoge prednosti, a jedna od njih je i uredniji pregled nad datotekama aplikacije. Nema potrebe za pisanjem čistog koda u *Python*-u i uključivanja pojedinih *Kivy* biblioteka, ali i to ima svojih prednosti u slučaju da treba dodati dinamičke komponente.

4. DIZAJN APLIKACIJE

Sama ideja aplikacije je bila omogućiti korisniku jednostavnije pronalaženje datoteka odgovarajućih ekstenzija u pojedinim dokumentima te pohranjivanje pojedinih podataka o slikama u bazu podataka. Ovo poglavlje objašnjava kompletnu aplikacija.

4.1. Prezentacijski dio

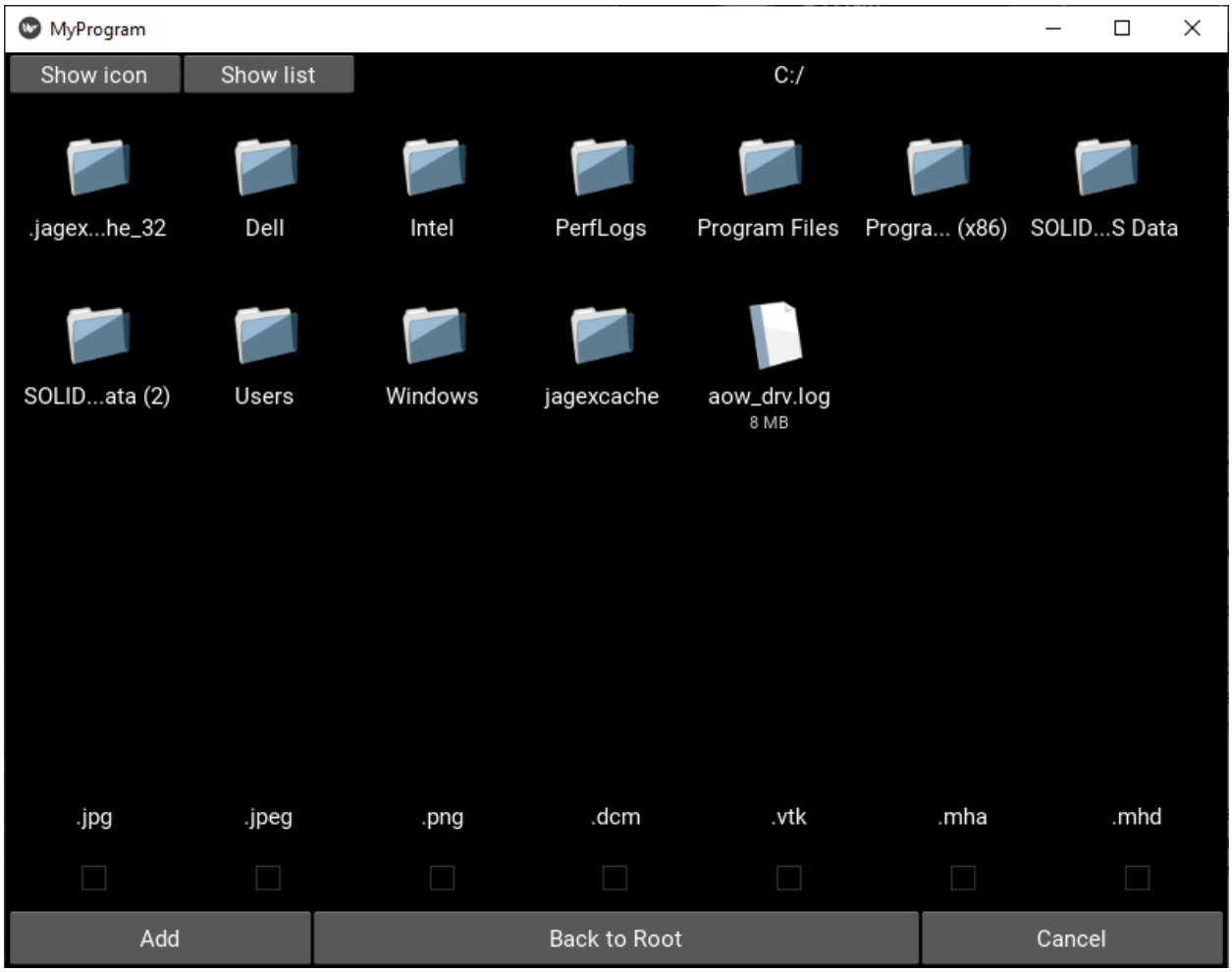
Skica prozora za dodavanje dokumenata koji se žele pratit s odgovarajućim ekstenzijama se vidi na slici 4.1. Korisniku aplikacije se pruža mogućnost prikaza dokumenata kao ikona ili kao liste. U podnožju sučelja također ima izbor za odabir ekstenzija slika koje želi pratiti. Također posjeduje tri dugmića za dodavanje dokumenata za praćenje s ekstenzijama (dugme ADD), vraćanja na početak izbornika te dugme(dugme BACK TO ROOT) te za vraćanje na početni ekran (dugme CANCEL).



Sl. 4.1. Prikaz skice sučelja AddFolderScreen.

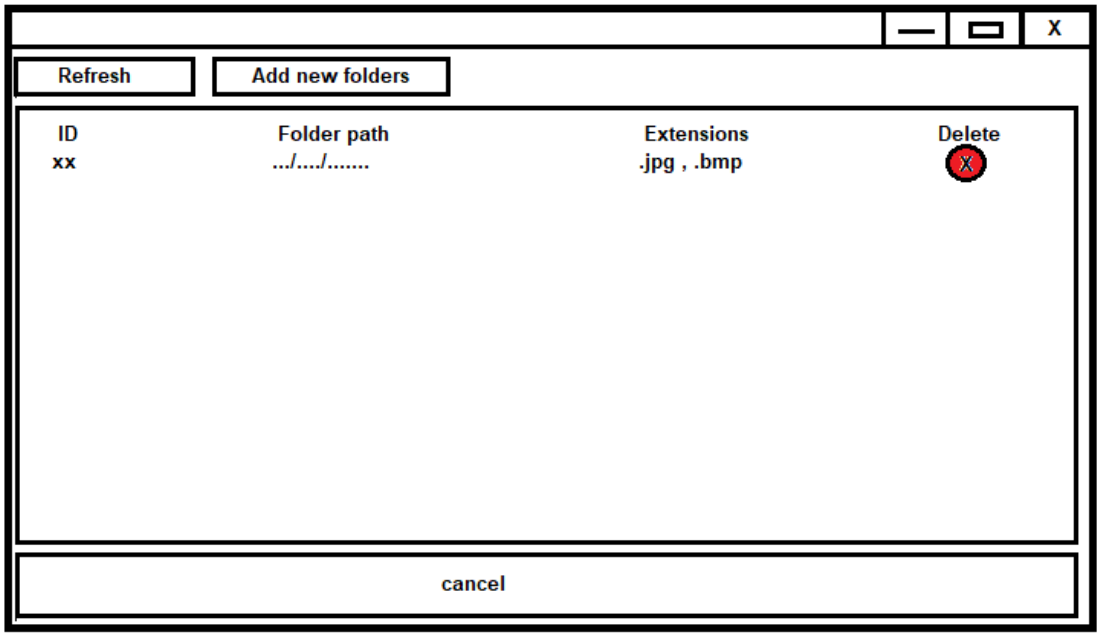
Čitav izgled sučelja je organiziran pomoću *BoxLayouta* koji je ujedno i podklasa *widget*-a te je orijentiran vertikalno i služi kao spremnik za grafičke elemente. Služi za prikaz grafičkih elemenata jednog pored drugog(horizontalno) ili jednog ispod drugog(vertikalno). Unutar se nalazi još jedan spremnik koji je orijentiran horizontalno i on se odnosi na dva dugmića (ICON i LIST) te tekstualna oznaka za prikaz putanje.

Zatim slijedi *RelativeLayout* za prikaz dokumenata koji sadrži gotovi grafički element *FileChooser* spreman za korištenje, prikazivanje i prolazak kroz datoteke sustava. *RelativeLayout* je također podklasa *widget*-a čiji su unutrašnji grafički elementi smješteni relativno u odnosu na njega. Ispod je vidljiva tekstualna oznaka za prikaz dodanih dokumenata za praćenje i još tri spremnika grafičkih elemenata. Jedan sadrži nazive ekstenzija, drugi sadrži dugmiće za odabir i treći sadrži tri dugmića(ADD, BACK TO ROOT, CANCEL). U konačnici to izgleda kao na slici 4.2.



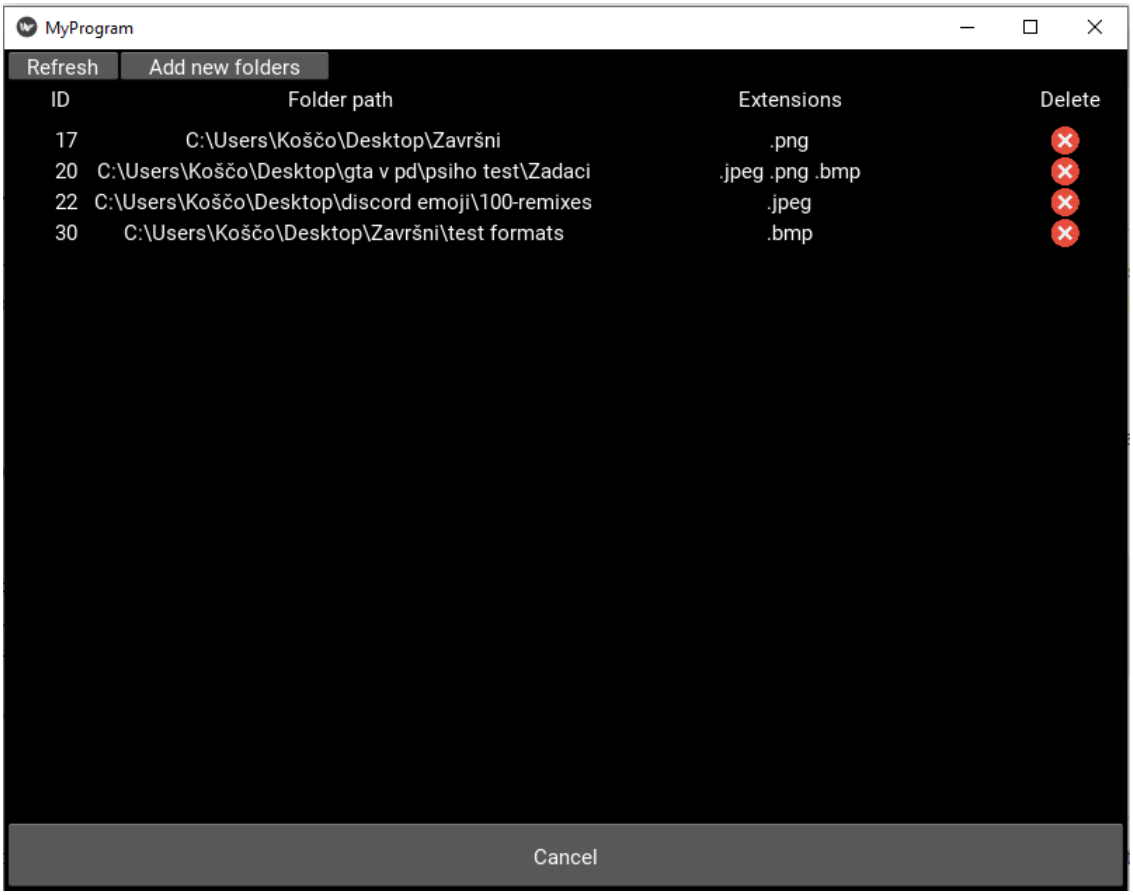
Sl. 4.2. Prikaz sučelja AddFolderScreen.

Izgled prozora SettingsScreen za prikaz putanja dokumenata koji se prate s pripadajućim ekstenzijama je također prvo smišljen. Zamišljen je prema skici ispod (Sl. 4.3.).



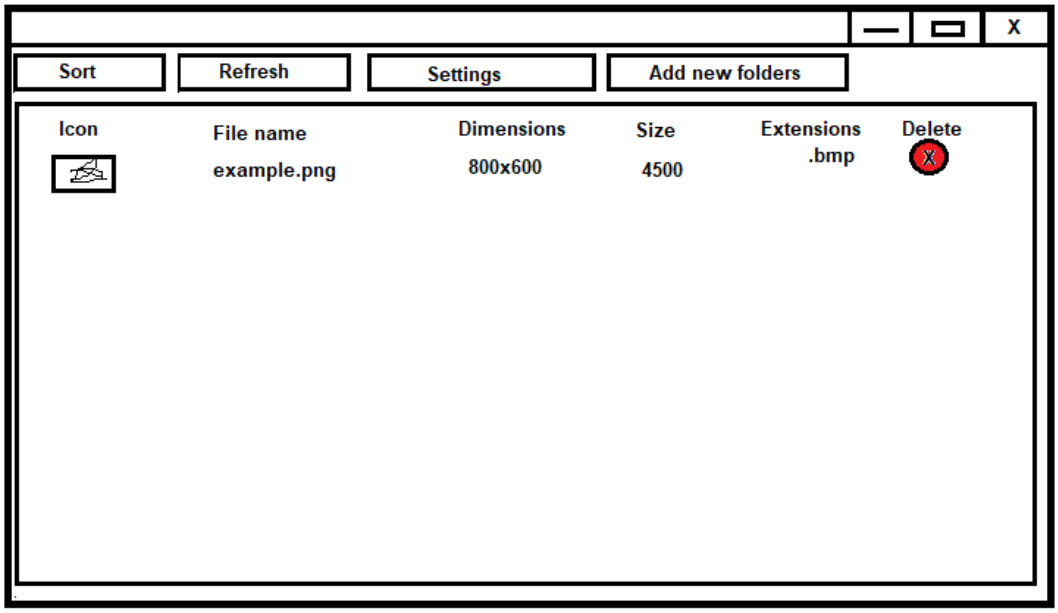
Sl. 4.3. Prikaz skice sučelja SettingsScreen.

Korisnik ima mogućnost osvježiti prikaz kako bi povukao podatke iz baze settings.db u koju su prethodno dodani dokumenti za praćenje s pripadajućim ekstenzijama. Nakon ispisivanja bilo koji od ispisanih može ukloniti klikom na crveni dugmić. Također se klikom na dugmić cancel može vratiti na početni ekran. Struktura je organizirana slično kao i kod sučelja AddFolderScreen. Jedan glavni spremnik grafičkih elemenata koji je orijentiran vertikalno sadrži dva spremnika i dva *GridLayouta*. *GridLayout* se koristi za mrežasti prikaz grafičkih elemenata tj. koristi se kao tablica. Prvi spremnik sadrži dva dugmića (Refresh i Add new folders), zatim slijedi prva tablica koja sadrži 4 kolone te one predstavljaju zaglavlje tablice(ID, Path, Extensions i Delete). Poslije toga slijedi *ScrollView* koji sadrži drugu tablicu. *ScrollView* je grafički element i omogućava da se grafički elementi koji se nalaze unutar njega kreću gore dolje radi lakše preglednosti kod velikog broja podataka. Posljednji spremnik sadrži dugmić(cancel) za povratak na početni ekran. Na slici 4.4. vidljiv je rezultat sučelja za prikaz datoteka koje se prate sa pripadajućim ekstenzijama.



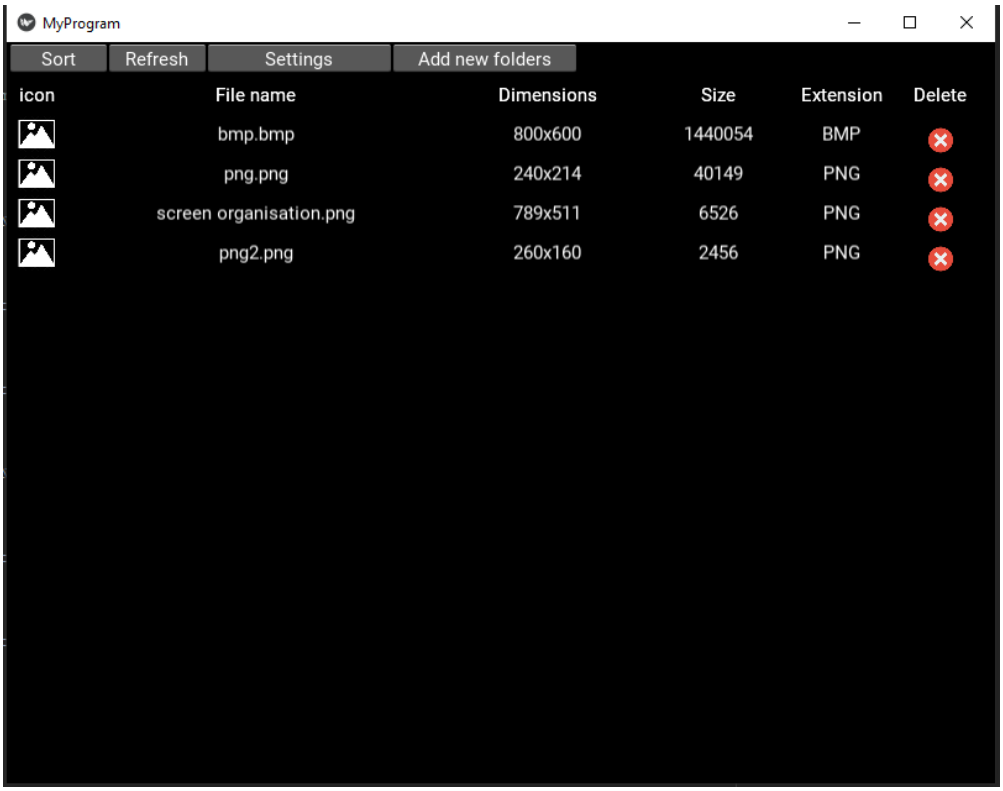
Sl. 4.4. Prikaz sučelja SettingsScreen.

Glavno sučelje ili MainScreen služi za prikaz datoteka koje odgovaraju datim putanjama i pripadajućim ekstenzijama povućenim iz baze settings.db. Korisniku pruža mogućnost sortiranja pronađenih datoteka(slika) i osvježavanje kako bi pronašao nove slike ukoliko je došlo do dodavanja/brisanja dokumenata za praćenje sa svojim ekstenzijama. Klikom na dugmić za osvježavanje podaci o pronađenim slikama se spremaju u bazu files.db. Ostale mogućnosti su prelazak na SettingScreen sučelje te AddScreen sučelje koji su ranije opisani. Na slici 4.5. vidljiva je skica spomenutog sučelja.



Sl. 4.5. Prikaz skice sučelja MainScreen.

Slično kao i prethodna dva sučelja , glavni ekran sadrži jedan glavni spremnik orijentiran vertikalno. Unutar njega se nalaze spremnik i dvije tablice. Prvi spremnik sadrži dugmiće za sortiranje(sort), osvježavanje(refresh), prelazak na sučelje SettingScreen(settings) i prelazak na sučelje za dodavanje novih dokumenata i ekstenzija(Add new folders) . Prva tablica ima 6 kolona koji ujedno predstavlja i zaglavlje tablice (Icon, File name, Dimensions, Size, Extensions i Delete). Druga tablica je unutar *ScrollView* grafičkog elementa te on predstavlja glavni dio tablice i sadrži podatke o slikama povućene iz baze. Na slici 4.6. vidljiv je izgled našeg sučelja za prikaz pronađenih dokumenata i njegovih pojedinih vrijednosti(ime, dimenzija, veličina, ekstenzija).



Sl. 4.6. Prikaz sučelja MainScreen.

4.2. Logički dio

Aplikacija kreće od klase ScreenHandler(Sl.4.7.) čija je svrha da brine samo o ScreenManager-u. ScreenManager je klasa koja je uključena u program.py budući da je potreban za prolazak između ekrana. Kako je, već ranije navedeno da postoji tri sučelja (ekrana) MainScreen, SettingsScreen i AddFolderScreen njegova svrha je brine kad je koji ekran pozvan. Kada se stvori objekt klase ScreenHandler poziva se funkcija create_screens koja preko ScreenManager-a dodaje tri ekrana. Također posjeduje metodu(get_scrmanager) koja vraća taj upravitelj ekranima.

```
class ScreenHandler():
    sm = ScreenManager()
    def __init__(self):
        self.create_screens()
    def create_screens(self):
        self.sm.add_widget(MainScreen(name='main'))
        self.sm.add_widget(SettingsScreen(name='settings'))
        self.sm.add_widget(AddFolderScreen(name='addfolders'))
    def get_scrmanager(self):
        return self.sm
```

Sl. 4.7. Klasa ScreenHandler.

Klasa MyProgramApp(App) predstavlja ulaznu točku aplikacije koja prilikom pokretanja vraća ScreenManagera preko objekta klase ScreenHandler i metode koja je ranije spomenuta(get_scrmanager). Kod je vidljiv na slici 4.8.

```
class MyProgramApp(App):
    def build(self):
        return ScreenHandler().get_scrmanager()
```

Sl. 4.8. Ulazna točka aplikacije.

Za korištenje baze podatka u Pythonu uključuje se sqlite3 baza podataka koja predstavlja lokalnu bazu podataka. Radi lakšeg rukovanja podacima aplikacija posjeduje dva upravitelja bazama. Jedna se odnosi na bazu settings.db dok je druga u funkciji baze files.db. Klasa SettingsDataBaseHandler(Sl. 4.10.) prilikom kreiranja objekta poziva metodu create_db koja stvara tablicu settings_table ukoliko ne postoji. Osim navedene metode, posjeduje i metodu za dohvaćanje podataka iz baze, metodu za brisanje podataka iz baze prema odgovarajućoj identifikacijskoj oznaci(engl. *ID*), metodu koja slaže primljene argumente u jedan skup znakova te metodu koja primljene argumente ubacuje u bazu podataka(Sl.4.9. i Sl.4.10.).

```
class SettingsDataBaseHandler():
    def __init__(self):
        self.create_db()

    def __connect_db(self):
        connection = sqlite3.connect('settings.db')
        return connection

    def __disconnect_db(self,connection):
        connection.commit()
        connection.close()

    def get_data(self):
        connection = self.__connect_db()
        pointer = connection.cursor()
        results = pointer.execute("SELECT * from settings_table")
        results = results.fetchall()
        self.__disconnect_db(connection)
        return results

    def remove_data(self,text):
        connection = self.__connect_db()
        pointer = connection.cursor()
        exc = "DELETE FROM settings_table WHERE settings_id=" + text
        pointer.execute(exc)
        self.__disconnect_db(connection)
```

Sl. 4.9. Klasa SettingsDataBaseHandler. 1.dio.

```
def build_exts(self,chb_jpg, chb_jpeg, chb_png, chb_dcm, chb_vtk, chb_mha, chb_mhd):
    exts = ''
    if (chb_jpg == True):
        exts = exts + ' .jpg'
    if (chb_jpeg == True):
        exts = exts + ' .jpeg'
    if (chb_png == True):
        exts = exts + ' .png'
    if (chb_dcm == True):
        exts = exts + ' .dcm'
    if (chb_vtk == True):
        exts = exts + ' .vtk'
    if (chb_mha == True):
        exts = exts + ' .mha'
    if (chb_mhd == True):
        exts = exts + ' .mhd'
    return exts

def insert_db(self,path,chb_jpg,chb_jpeg,chb_png,chb_bmp):
    connection = self.__connect_db()
    pointer = connection.cursor()
    newexts = self.build_exts(chb_jpg, chb_jpeg, chb_png, chb_bmp)
    newpath = path
    pointer.execute("INSERT INTO settings_table (paths,extensions) VALUES (?,?)",
                    ([newpath, newexts]))
    self.__disconnect_db(connection)

def create_db(self):
    connection = self.__connect_db()
    pointer = connection.cursor()
    pointer.execute('''CREATE TABLE IF NOT EXISTS settings_table
           (settings_id INTEGER PRIMARY KEY AUTOINCREMENT, paths TEXT , extensions TEXT )''')
    self.__disconnect_db(connection)
```

Sl. 4.10. Klasa SettingsDataBaseHandler. 2.dio.

Za bilo kakav pristup bazi potrebno je prvo stvoriti konekciju prema odgovarajućoj bazi. Na bazu se povezuje metodom __conect_db što je ujedno i privatna metoda. Nakon toga treba postaviti pokazivač(kursor) na tu konekciju. Pokazivačem se dalje izvršavaju odgovarajuće naredbe. Bitno je zatvoriti konekciju prema bazi podataka kako ne bi bila uzaludno otvorena nakon obavljenog posla što se izvršava metodom __disconnect_db.

Klasa AddFolderScreen nasljeđuje klasu Screen te u sebi sadrži objekt klase SettingsDataBaseHandler, a predstavlja klasu koja upravlja sučeljem AddFolderScreen. Klikom na dugme Add na slici 4.2. povlači se putanja te odabrane ekstenzije. Zatim se objektom klase SettingsDataBaseHandler i metode insert_db povučeni podaci spremaju u bazu settings.db što je vidljivo na slici 4.11.

```
class AddFolderScreen(Screen):
    settingsdbhandler = SettingsDataBaseHandler()

    def btn_add(self,path,chb_jpg,chb_jpeg,chb_png,chb_bmp):
        self.settingsdbhandler.insert_db(path,chb_jpg,chb_jpeg,chb_png,chb_bmp)
```

Sl. 4.11. Upravitelj sučeljem AddFolderScreen.

Klasa SettingsScreen u sebi ima objekt klase SettingsDataBaseHandler i objekt klase SettingsHandler te ujedno upravlja svojim sučeljem. Odgovornost klase SettingsHandler leži u upravljanju podacima. Kao što je vidljivo na slici 4.12. ,posjeduje objekt koji upravlja podacima u bazi settings.db i posjeduje posebnu listu u koju se spremaju podaci iz baze. Metodom clear_table čisti se tablica koja se predaje kao argument te posebna lista. Čišćenje liste je bitno, budući da se želi izbjeći nagomilavanje istih podataka u listi dok je za ispisivanje podataka potreban čist layout.

```
class SettingsHandler():
    dbhandler = SettingsDataBaseHandler()
    folders = []

    def clear_table(self,grtable):
        grtable.clear_widgets()
        self.folders.clear()

    def grab_data(self):
        results = self.dbhandler.get_data()
        self.folders.clear()

        for result in results:
            self.folders.append({
                "id:": result[0],
                "path:":result[1],
                "exts:":result[2]
            })

    def get_folders(self):
        return self.folders

    def get_extensions(self):
        return [x["exts:"] for x in self.folders]

    def get_paths(self):
        return [x["path:"] for x in self.folders]
```

Sl. 4.12. Klasa SettingsHandler

Također posjeduje i metodu grab_data koja preko objekta(SettingsDataBaseHandler) ,dohvaća podatke metodom get_data, ponovo čisti listu te zatim podatke iz baze pohranjuje u obliku koji olakšava prolazak kroz listu. Metodom get_folders vraća se lista kao rezultat poziva iste. Kako bi se olakšalo dohvaćanje putanja i ekstenzija postoje dvije metode koje vraćaju traženo, a to su get_extensions i get_paths. Klikom na dugme Refresh (Sl.4.4.) poziva se metoda btn_refresh iz klase SettingsScreen. Na slici 4.13. vidljivo je kako se obavljaju tri funkcije. Prvo se poziva metoda istoimene klase refresh_table kojoj se kao argument predaje tablica te se preko objekta za upravljanje podacima poziva metoda clear_table i predaje taj isti widget. Zatim se preko istog objekta(klase SettingsHandler) poziva metoda grab_data i u konačnici se poziva metoda za ispis podataka na ekran (fill_table). Metoda fill_table kao argumente prima tablicu i preko objekta za upravljanje podacima(i njegove metode get_folders) listu podataka. Prolaskom kroz predanu listu podataka, u tablicu se dodaju tri tekstualne oznake(id, putanju i ekstenziju) te ikonica za uklanjanje spomenutih oznaka. Kako bi to bilo moguće ,potrebno je svaku tu ikonicu povezati s metodom remove_button. Spomenuta metoda se poziva svaki puta kada se klikne na ikonicu za brisanje. Također detektira id ikone(identičan je id-u podataka u bazi) koja je kliknuta te preko objekta(klase SettingsDataBaseHandler) poziva metodu remove_data te mu predaju id kao argument.

```
class SettingsScreen(Screen):
    dbhandler = SettingsDataBaseHandler()
    settingshandler = SettingsHandler()

    def btn_refresh(self):
        self.refresh_table(self.ids.settings_gridl)
        self.settingshandler.grab_data()
        self.fill_table(self.ids.settings_gridl,self.settingshandler.get_folders())

    def refresh_table(self,grl_table):
        self.settingshandler.clear_table(grl_table)

    def fill_table(self,grtable,folders):

        for folder in folders:
            label_id = Label(text=str(folder["id:"]), size_hint_x=0.1, size_hint_y=None, height=20)
            grtable.add_widget(label_id)
            label_path = Label(text=folder["path:"], size_hint_x=0.4, size_hint_y=None, height=20)
            grtable.add_widget(label_path)
            label_ext = Label(text=folder["exts:"], size_hint_x=0.4, size_hint_y=None, height=20)
            grtable.add_widget(label_ext)
            icon_remove = IconButton(size_hint_x=0.1, size_hint_y=None, height=20,
                                     id="icon_remove_"+str(folder["id:"]),source="delete_icon.png")
            icon_remove.bind(on_press=self.remove_button)
            grtable.add_widget(icon_remove)

    def remove_button(self,event):
        str=event.id.split('_')      #   ['icon','remove','number']
        self.dbhandler.remove_data(str[2])
        self.btn_refresh()
```

Sl. 4.13. Klasa SettingsScreen.

Rukovatelj bazom podataka za pronađene datoteke (files.db) je sličan bazi podataka u koju se spremaju putanje dokumenata sa pripadajućim ekstenzijama (settings.db). Razlika je što posjeduje dvije dodatne metode, za brisanje pronađenih podataka(clear_db) i za sortiranje baze podataka prema veličini pronađenih datoteka(sort_data). Navedene dvije metode su vidljive na slici 4.14.

```
def clear_db(self):
    connection = self.__connect_db()
    pointer = connection.cursor()
    pointer.execute("DELETE FROM files_table")
    self.__disconnect_db(connection)

def sort_data(self):
    connection = self.__connect_db()
    pointer = connection.cursor()
    results = pointer.execute("SELECT * FROM files_table ORDER BY size DESC;")
    results = results.fetchall()
    self.__disconnect_db(connection)
    return results
```

Sl. 4.14. Klasa FilesDataBaseHandler, dodatne metode.

Sučelje glavnog ekrana funkcionira na sličan način kao i SettingsScreen. Također posjeduje dva objekta, jedan od klase FilesHandler i jedan od FilesDataBaseHandler. Klasa FilesHandler posjeduje objekt klase SettingsHandler, objekt klase FilesDataBaseHandler i tri liste za putanje, ekstenzije i pronađene datoteke. Metodom grab_data se preko objekta settingshandler i njegove metode grab_data dohvaćaju podaci iz baze settings.db. Zatim se preko istog objekta i metoda get_paths i get_extensions dohvaćaju putanje i ekstenzije, koje se zatim razdvajaju na zarezu.

```
class FilesHandler():
    settingshandler = SettingsHandler()
    paths = []
    exts = []

    filesdbhandler = FilesDataBaseHandler()
    files = []

    def grab_settings(self):
        self.settingshandler.grab_data()
        self.paths = self.settingshandler.get_paths()
        self.exts = self.settingshandler.get_extensions()
        self.exts = [x.split() for x in self.exts]

    def find_files(self):
        self.filesdbhandler.clear_db()
        for i in range(len(self.paths)):
            for root, dirs, files in os.walk(self.paths[i]):
                for f in files:
                    for ext in self.exts[i]:
                        if f.endswith(ext):
                            self.process_images(os.path.join(root, f))
```

Sl. 4.15. Klasa FilesHandler 1.dio.

Na slici 4.15 uočljiva je i metoda find_files kojom se prvo preko objekta filesdbhandler čisti tablica s podacima o pronađenim datotekama. Dohvaćene putanje predstavljaju putanje odabranih dokumenata koji se žele pratiti s odgovarajućim ekstenzijama dokumenata radi filtriranja. Nakon brisanja , slijedi prolazak kroz spomenute putanje te ukoliko se na njoj pronađu datoteke koje odgovaraju spremljenim ekstenzijama poziva se metoda process_image (Sl.4.16.) kojoj se kao argument predaje putanja od pronađene datoteke. Kako bi se spremili određeni podaci o pronađenom dokumentu koristi se PIL biblioteka. Spremaju se podaci o dimenziji datoteke(slike), ekstenzija, veličina te ime. Nakon toga, preko objekta klase FilesDataBaseHandler i njegove metode insert_db spremamo podatke u bazu files.db. Metoda grab_dbfiles kao argument prima listu, kojom zatim prolazi i u listu za pronađene datoteke(Sl. 4.15. ,files) sprema podatke iz baze podataka u posebnom obliku radi lakšeg prolaska kroz istu. Također uočljive su metode clear_table kojom se čisti danu tablicu te liste za putanje, ekstenzije i datoteke te metoda get_files kojom se predaje lista datoteka.

```
    def process_images(self, path):
        with Image.open(path) as image:
            width, height = image.size
            extension = image.format
        size = os.path.getsize(path)
        name = os.path.basename(path).strip('.' + extension)
        self.filesdbhandler.insert_db(name, path, str(width) + "x" + str(height), size, extension)

    def grab_dbfiles(self,results):

        for result in results:
            self.files.append({
                "id:": result[0],
                "name:": result[1],
                "path:": result[2],
                "dimensions:": result[3],
                "size:": result[4],
                "extension:": result[5]
            })

    def clear_table(self, grtable):
        grtable.clear_widgets()
        self.files.clear()
        self.paths.clear()
        self.exts.clear()

    def get_files(self):
        return self.files
```

Sl. 4.16. Klasa FilesHandler 2.dio.

Klikom da dugme Refresh (Sl. 4.6.) u klasi glavnog sučelja poziva se metoda btn_refresh(Sl. 4.17.) koja vrši nekoliko funkcija. Prvo se brišu grafički elementi u danoj tablici metodom clear_table ,objekta klase FilesHandler. Zatim se na istom objektu pozivaju metode grab_settings, find_files i

grab_dbfiles. Kao argument metodi grab_dbfiles predaje se rezultat izvođenja metode get_data objekta klase FilesDataBaseHandler, tj. lista s podacima o datoteci.

U konačnici metodom fill_table ispisuju se podaci na ekranu na identičan način kao i kod sučelja SettingsScreen. Kao argumenti predaju se tablica koja se popunjava podacima i podaci u obliku liste koje treba ispisati.

```
class MainScreen(Screen):
    fileshandler = FilesHandler()
    filesdbhandler = FilesDataBaseHandler()
    def btn_refresh(self):
        self.fileshandler.clear_table(self.ids.folders_gridl)
        self.process_files()
        self.fill_table(self.ids.folders_gridl,self.fileshandler.get_files())
    def process_files(self):
        self.fileshandler.grab_settings()
        self.fileshandler.find_files()
        self.fileshandler.grab_dbfiles(self.filesdbhandler.get_data())
    def fill_table(self, grtable, files):
        for file in files:
            img = Img(source="image_icon.png", size_hint_x=0.05, size_hint_y=None, height=30)
            grtable.add_widget(img)
            label_name = Label(text=file["name:"], size_hint_x=0.4, size_hint_y=None, height=30)
            grtable.add_widget(label_name)
            label_dimensions = Label(text=file["dimensions:"], size_hint_x=0.2, size_hint_y=None, height=30)
            grtable.add_widget(label_dimensions)
            label_size = Label(text=str(file["size:"]), size_hint_x=0.15, size_hint_y=None, height=30)
            grtable.add_widget(label_size)
            label_extension = Label(text=file["extension:"], size_hint_x=0.1, size_hint_y=None, height=30)
            grtable.add_widget(label_extension)
            icon_remove = IconButton(size_hint_x=0.1, size_hint_y=None, height=20,
                                     id="icon_remove_" + str(file["id:"]),
                                     source="delete_icon.png")
            icon_remove.bind(on_press=self.remove_button)
            grtable.add_widget(icon_remove)
```

Sl. 4.17. Klasa MainScreen 1.dio.

Sučelja SettingsScreen i MainScreen su jako slična. Razlika leži u broju funkcija koje se obavljaju u metodi btn_refresh, metodi remove_button i dodatnoj metodi za sortiranje podataka.

```
    def remove_button(self, event):
        str = event.id.split('_')  # ['icon','remove','number']
        self.filesdbhandler.remove_data(str[2])
        self.fileshandler.clear_table(self.ids.folders_gridl)
        self.fileshandler.grab_dbfiles(self.filesdbhandler.get_data())
        self.fill_table(self.ids.folders_gridl, self.fileshandler.get_files())
    def btn_sort(self):
        self.filesdbhandler.sort_data()
        self.fileshandler.clear_table(self.ids.folders_gridl)
        self.fileshandler.grab_dbfiles(self.filesdbhandler.sort_data())
        self.fill_table(self.ids.folders_gridl, self.fileshandler.get_files())
```

Sl. 4.18. Klasa MainScreen 2.dio.

Razlika remove_button metode klase SettingsScreen i MainScreen leži u tome da se nakon brisanja, prvo čisti tablica(GridLayout) preko objekta klase FilesHandler, zatim se metodom grab_dbfiles na istom objektu dohvaćaju podaci o datotekama iz baze i konačno metodom fill_table se tablica ponovno popunjava. Sve to je vidljivo na slici 4.18. U slučaju da je umjesto navedenih metoda stavljena metoda btn_refresh prilikom brisanja bilo kojeg podatka izmjena ne bi bila vidljiva. Metoda btn_refresh ima nekoliko funkcija više koje se izvršavaju iz razloga što ukoliko u međuvremenu dođe do promjena (npr. brisanje ili dodavanje putanja dokumenata koji se prate) promjene moraju biti vidljive. Svakim osvježavanjem iznova se dohvaćaju putanje i ekstenzije iz baze settings.db te se proces ponavlja iznova kao što je gore navedeno. Dodatna metoda btn_sort preko objekta klase FilesDataBaseHandler sortira podatke u bazi prema veličini same datoteke te ih ponovno ispisuje u sučelju.

Kako bi sve funkcioniralo kako spada, aplikacija, tekstualna oznaka, upravitelj ekranima itd. potrebno je navedene dijelove uključiti iz pojedinih biblioteka. Navedeno vidimo na slici 4.19.

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.core.window import Window
from kivy.uix.image import Image as Img
from kivy.uix.button import ButtonBehavior
from PIL import Image
```

Sl. 4.19. Bitne biblioteke i njihovi dijelovi.

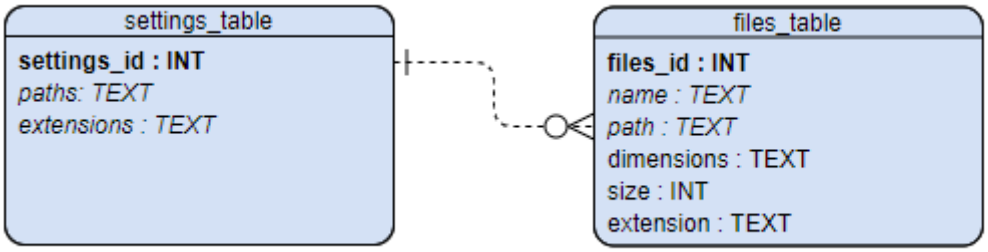
```
import sqlite3
import os

Window.size = (800, 600)

Builder.load_file('myprogram.kv')
Builder.load_file('settings.kv')
Builder.load_file('addfolders.kv')
```

Sl. 4.20. Bitni dijelovi aplikacije.

Kao što je, već spomenuto, za korištenje *sqlite* baze podataka u Python-u, potrebnu ju je uključiti u samu aplikaciju komandom import. Također vidljivo je (Sl. 4.20.) kako se koristi modul os koji omogućava korištenje operacijskog sustava. Postavljena je fiksna veličina prozora. Kako bi se odvojila kompleksnost aplikacije i stvorila bolja preglednost koristimo uzorak graditelja(engl. *builder*) koji to omogućava. Tako se sama logika aplikacije razdvaja od njene prezentacije ,ali opet uključuje u samu logiku kako bi izgled bio zadovoljavajući putem graditelja.

4.3. ER dijagram baza

Na slici 4.21. uočava se ERD(engl. *Entity Relationship Diagram*) odnosno dijagram odnosa s entitetima [25]. Služi za stvaranje strukturnog izgleda same baze podataka. Crtanjem baze podataka u obliku ovakvih dijagrama omogućava pronalaženje grešaka i ispravljanje istih prije izrade baze podataka. Dijagram se sastoji od entiteta, atributa i veza. Entitet je neka određena stvar kao npr. osoba, vozilo, knjiga itd. Atributi su svojstvo koje opisuje svaki entitet kao npr. u slučaju osobe to bi bilo spol, visina, težina itd. Sastoji se od imena koje opisuje svojstvo i vrste koja opisuje tip podatka. Veza između dva entiteta govori kako među njima postoji nekakav odnos. Postoji veza jedan naprema jedan, jedan naprema više i više naprema više.



Sl. 4.21. ERD baza.

Na slici 4.21. može se primijetiti kako entitet settings_table koji predstavlja dokument koji se prati posjeduje tri atributa. Posjeduje settings_id čija je vrsta podataka cjelobrojni broj, putanju u koju se spremaju podaci u tekstualnom formatu te ekstenziju s istom vrstom podataka. Tablica files_table predstavlja pronađeni dokument. On posjeduje atribute kao files_id i veličinu datoteke cjelobrojnog tipa podataka te ime, putanju, dimenziju i ekstenziju tekstualnog tipa podataka. Veza između njih je jedna naprema više što znači da dokument koji se prati može imati više datoteka određenih ekstenzija, dok se ta pronađena datoteka može nalaziti u samo jednom dokumentu.

5. ZAKLJUČAK

Svrha ovog završnog rada bila je stvoriti sustav za upravljanje medicinskim snimkama. Prilikom izrade rada predstavljene su moguće medicinske snimke kao što su planarna slika, tomografska slika, 3D slika i 4D slika. Također bitna stavka za razumijevanje rada su teorijski pojmovi kao što su dubina piksela, fotometrijska interpretacija, meta-podaci i podaci piksela. Ti pojmovi su objašnjeni budući da su bitni za formate snimaka koji su u radu spomenuti. Zbog velike rasprostranjenosti i pogodnosti u različitim područjima medicine, DICOM je najviše opisan. Bitno obilježje su mu oznake (engl. *tag*) koje sadrže različite podatke vezane za samu sliku. Razlika DICOM-a i MetaImage-a leži u tome što DICOM predstavlja međunarodni standard dok je MetaImage samo format. Osim spomenuta dva formata, spominje se i biblioteka VTK (engl. *Visualization Toolkit*) koja obavlja razne funkcije koje su spomenute u radu.

Nakon uvoda u sam rad i upoznavanja s pojedinim formatima medicinskih snimkama, dizajniran je i sam izgled aplikacije. Prije same realizacije izgleda, spomenuta su pojedina grafička sučelja koja se u programskom jeziku *Python*-u mogu koristiti. Među spomenutima vidljivo je kako im je zajednička platformska neovisnost odnosno mogućnost pokretanja na različitim platformama. Međutim, *Kivy* kao grafičko sučelje odudara od ostalih zbog jednostavnosti pisanja, mogućnosti pokretanja istog koda na svim platformama te razdvajanja logičkog dijela od prezentacijskog. Prezentacijski dio je napravljen koristeći *Kivy* grafičko korisničko sučelje što je ujedno bila i ideja ovog završnog rada. Izrada svakog od tri navedena sučelja je objašnjena radi lakšeg razumijevanja kako je izgled organiziran. Detaljnije je objašnjena logika same aplikacije budući da je vrlo bitna za razumijevanje svrhe ovog rada. Predstavljene su slike uz opis svake pojedine klase i njihovih metoda. Naknadno je objašnjeno i dodavanje bitnih elemenata za ovaj rad iz pojedinih biblioteka.

Budući da sama aplikacija koristi bazu podataka, istu je bilo potrebno unaprijed osmisliti kako bi se izbjegli mogući problemi prilikom povezivanja istih. ER dijagramom je predstavljena povezanost između baza podataka koje se u radu koriste.

LITERATURA

[1] U.S. Food & Drug Administration, dio Medical Imaging (https://www.fda.gov/radiation-emitting-products/radiation-emitting-products-and-procedures/medical-imaging)

[2] US National Library of Medicine National Institutes of Health (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3948928/)

[3] Wikipedia, Tomografija, slika (https://en.wikipedia.org/wiki/Tomography)

[4] PHYS.org ,slika (https://3c1703fe8d.site.internapcdn.net/newman/csz/news/800/2019/1-new3dimaging.jpg)

[5] Semantic Scholar, slika (https://www.semanticscholar.org/paper/Four-dimensional-imaging-of-chromatin-dynamics-the-Manders-Visser/0108ec88592366b3b54c30bd36b6d7afca5b42f4)

[6] DICOM (https://www.dicomstandard.org/about/)

[7] File Viewer Lite, slika (https://cdn.windowsfileviewer.com/images/types/open_dicom_with_file_viewer_lite.png)

[8] Web stranica, Micro Dicom, slika (http://www.microdicom.com/images/screenshots/MicroDicom.png)

[9] DICOM wiki (https://en.wikipedia.org/wiki/DICOM)

[10] Data Structures and Encoding (http://dicom.nema.org/medical/dicom/current/output/pdf/part05.pdf)

[11] Required DICOM tags (https://www.pclviewer.com/help/required_dicom_tags.htm)

[12] File-Extensions.org (https://www.file-extensions.org/mha-file-extension)

[13] MetaImage (https://itk.org/Wiki/ITK/MetaIO/Documentation)

[14] ITK (https://itk.org/ITK/project/about.html)

[15] ImageMagick (https://www.imagemagick.org/discourse-server/viewtopic.php?t=20226)

[16] VTK(The Visualization Toolkit) (https://vtk.org/about/#overview)

[17] PDF dokument, file-formats.pdf (https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf)

[18] TCL and TK (http://www.tcl.tk/)

[19] Graphical User Interfaces with Tk (https://docs.python.org/3/library/tk.html)

[20] Primjer za Tkinter (https://www.geeksforgeeks.org/python-gui-tkinter/)

[21] PyQt (https://riverbankcomputing.com/software/pyqt/intro)

[22] Primjer za PyQt (https://www.guru99.com/pyqt-tutorial.html)

[23] Roberto Ulloa Kivy – Interactive Applications and Games in Python (Second edition) (2015), Chapter 1.GUI Basics – Building an Interface (https://gitlab.com/timofonic/bkkkk/blob/master/kivy-interactive-applications-roberto-ulloa(www.ebook-dl.com%202nd%20ed).pdf)

[24] Kivy internetska stranica (https://kivy.org/#home)

[25] ERD (https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/)

SAŽETAK

Ideja ovog završnog rada bila je stvoriti softversko rješenje koje će omogućiti lakše upravljanje sustavom medicinskih snimaka. Uobičajeno traženje određenih medicinskih snimaka je otežano u slučaju velikog broja istih, spremljenih na nekom računalu. Što je veća memorija pohranjenih snimaka to je postupak traženja veći, a samim time i kompliciraniji bez obzira bile one uredno dokumentirane ili razbacane po memoriji. Problem je riješen koristeći *Python* programski jezik te *Kivy* grafičko korisničko sučelje, kroz tri sučelja i dvije baze podataka u koje se pohranjuju podaci. Jedno sučelje omogućava označavanje dokumenata koji se prate(preko putanja) s pripadajućim ekstenzijama te ujedno i spremanje istih u prvu bazu podataka. Drugo sučelje omogućava prikaz podataka iz prve baze podataka te ujedno uklanjanje pojedinih podataka. Trećim, odnosno glavnim sučeljem omogućen je prikaz podataka pronađenih datoteka (slika) koji se dohvaćaju iz druge baze podataka. Kako bi se datoteke pronašle koriste se putanje dokumenata koji se prate i ekstenzije te se prema njima filtriraju datoteke i njihovi podaci se dodaju u drugu bazu podataka. Uvidom u završni rad, može se zaključiti kako ovaj sustav za upravljanjem medicinskim snimkama je primjenljiv u raznim područjima, promjenom ekstenzija koje se prate.

Ključne riječi: medicinska snimka, Python, Kivy, sustav za upravljanjem

ABSTRACT

MANAGEMENT SYSTEM FOR MEDICAL RECORDINGS

The idea of this final work was to create a software solution that would more easily manage the medical recordings system. The usual searching for certain medical recordings has been difficulted in the case of a large number of it being stored on a computer. The greater the memory of stored recordings is that the searching process is larger and complicated whether files were stored orderly or scattered across the memory. The problem was solved using Python's programming language and Kivy graphical user interface, through three interfaces and two databases where data is stored. First interface simply allows to tag documents for tracking (using path of document) with associated extensions and finally store those two in the first database. The other one allows the display of data from first database and removing some of them. The third or main interface allows the display of data of found files (images) that are retrieved from second databases. To find files, paths of documents which are tracked and extensions are used to filter files and then add its data to second database. An insight into the final work, it can be concluded that this medical recordings management system is applicable in various fields, by changing extensions which are tracked.

Keywords: medical recordings, Python, Kivy, management system

ŽIVOTOPIS

Kristijan Koščak rođen je 16.veljače 1998. godine u Vinkovcima. Osnovno obrazovanje započeo je 2004. godine u Osnovnoj školi Antuna Gustava Matoša, Vinkovci. Po završetku osnovne škole, 2012. nastavlja obrazovanje u Tehničkoj školi Ruđera Boškovića u Vinkovcima gdje upisuje smjer elektrotehnika. 2016. godine upisuje preddiplomski studij, smjer računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku koji je u sklopu Sveučilišta Josipa Jurja Strossmayera.