Konstrukcija grafičkog sučelja u biblioteci Qt5

Ibriks, Edi

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Edi Ibriks

KONSTRUKCIJA GRAFIČKOG SUČELJA U BIBLIOTECI QT5

Diplomski rad

Voditelj rada:
Prof. dr. sc. Mladen Jurak

Zagreb, rujan 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Qt je programska okolina za konstrukciju aplikacija u programskom jeziku C++, koja omogućava jednostavnu konstrukciju korisničkog grafičkog sučelja. Programiranje grafičkog sučelja zahtijeva niz specifičnih programskih tehnika i oblikovnih obrazaca. Cilj ovog diplomskog rada je prezentirati i evaluirati programsku podršku koju Qt verzije 5 nudi za konstrukciju korisničkog grafičkog sučelja, kao i konstruirati složenu aplikaciju po vlastitom izboru.

Glavni dio rada podijeljen je u dva poglavlja. U prvom poglavlju dan je kratak pregled povijesti Qt-a, alata koji sudjeluju pri izgradnji aplikacija i značajke poput mehanizma signala i slotova. U drugom poglavlju dan je opis izgleda, funkcionalnosti i izrade web preglednika pomoću Qt5 biblioteke, pisanog u C++ programskom jeziku.

Poglavlje 1

Qt

1.1 Povijest Qt-a

Qt softverski okvir osmislila su dva norveška programera, Haavard Nord (CEO kompanije Trolltech) i Eirik Chamble-Eng(predsjednik Trolltech-a). Ideja je začeta 1990. godine kada su Haavard i Eirik radili zajedno na C++ aplikaciji s bazama podataka za ultrazvučne slike, koja je trebala imati mogućnost pokretanja na Unix, Macintosh i Windows računalima. Za vrijeme kolaboracije nastala je ideja o objektno-orijentiranom sofrverskom okviru s grafičkim sučeljem.

Haavard počinje s izradom prvih klasa Qt-a 1991. godine, dok se Eirik fokusira na dizajn. Sljedeće godine Eirik uvodi ideju signala i slotova, jednostavnu ali moćnu paradigmu u programiranju grafičkih sučelja karakterističnu za Qt. Godinu kasnije proizvode Qt-ovu prvu grafičku jezgru (*kernel*) i implementiraju vlastite *widget*-e. U ožujku 1994. godine osnivaju vlastitu kompaniju Quasar Technologies, koja kasnije mijenja ime u Trolltech, te smišljaju naziv za svoj proizvod (Qt). Odabiru slovo "Q" zbog njegova izgleda u Emacs fontu i slovo "t" iz riječi *toolkit* (skup alata).

U travnju 1995. godine, mjesec dana prije izlaska na tržište, sklapaju ugovor s norveškom kompanijom Metis za razvoj programa temeljenog na Qt biblioteci. U tom periodu zapošljavaju Arnta Gulbrandsena koji je u narednih šest godina u Trolltechu razvio i implementirao inovativan sustav dokumentacije te pridonio razvoju Qt-a.

20. svibnja 1995. na tržiste izlazi prva javna verzija Qt-a(Qt 0.90) dostupna pod dvije vrste licenci: komercijalnoj i besplatnoj za *open-source* aplikacije. Godinu kasnije Europska svemirska agencija kupnjom deset komercijalnih licenci postaje drugim kupcem Qt

biblioteke. Potom izlazi Qt 1.0, a do kraja godine Qt je imao osam kupaca, svaki iz druge države, s ukupno 18 kupljenih licenci. Te godine je i Matthias Ettrich osnovao KDE projekt, grafičko sučelje za Unix sustave koje je izgrađeno korištenjem Qt softverskog okvira.

1999. objavljena je verzija Qt 2.0, a iste godine dobivaju LinuxWorld nagradu za najbolju biblioteku/alat. Qt 3.0 izlazi 2001. kao prva verzija Qt-a dostupna za Mac OS X, s puno drugih noviteta od kojih su neki potpora za lokalizaciju i Unicode, poboljšani widgeti za uređivanje teksta i Qt Designer, alat za editiranje grafičkog sučelja koji funkcionira na principu *drag-and-drop*.

U ljeto 2005. objavljuju Qt 4.0, prvu verziju Qt-a koja je bila dostupna pod obje licence za sve operacijske sustave koje je podržavala. Qt 4 je s oko 500 klasa i više od 9000 funkcija bila najveća i najbogatija verzija Qt-a, zbog čega je morala biti podijeljena u više biblioteka kako bi se pri pokretanju učitavali samo dijelovi potrebni za ispunjenje zahtjeva konkretne aplikacije. Ovom verzijom Qt je od skupova alata za izradu grafičkog sučelja postao softverski okvir za razvoj aplikacija s grafičkim sučeljem [1].

Nokia otkupljuje Trolltech 2008. i mijenja mu naziv u Qt Software at Nokia. Godinu kasnije izlazi Qt 4.5 koji po prvi puta sadrži Qt Creator, integrirano razvojno okruženje (*integrated development environment*, IDE) koje ujedinjuje uređivač koda i Qt Designer. Godinu kasnije izlazi Qt 4.7 čija je glavna značajka potpora za Symbian uređaje te Qt Quick aplikacijski okvir za izradu aplikacija, koji uključuje deklarativni jezik QML.

U ožujku 2011. Nokia prodaje komercijalno licenciranje Qt-a tvrtki Digia, koja sljedeće godine otkupljuje sva prava na Qt i izbacuje Qt 5.0 s brojnim novitetima poput Qt Quick 2, te poboljšane grafike i brzine rada. Od tada, nove verzije Qt 5 izlaze svakih šest mjeseci s novim značajkama i podrškom za sve veći broj platformi.

Konačno, 2016. godine, The Qt Company osnovana 2014. kao podružnica Digie koja je bila zadužena za razvoj Qt-a, odvaja se od Digie i počinje djelovati samostalno. Danas je za razvoj Qt-a, osim The Qt Company, zadužan i Qt Project, skupina developera i tvrtki koji rade na poboljšanju Qt-a [2,3].

1.2 Apstrakcija grafičkog sučelja

Kada je prvi put objavljen, Qt je koristio vlastiti stroj za crtanje (*paint engine*) i kontrolu, oponašajući izgled različitih platformi pri stvaranju svojih *widgeta*. To je olakšalo prijenos aplikacija na druge platforme budući da je vrlo malo klasa u Qt-u ovisilo o ciljanoj platformi. Međutim, da bi se smanjila pogreška pri oponašanju, u novijim inačicama Qt-a uvodi se upotreba sučelja za razvoj aplikacija (*application programming interface*, API) ciljane platforme. Na nekim platformama (kao što su MeeGo i KDE) Qt je izvorni API, dok neki drugi prijenosni grafički alati kao i Qt, upotrebljavaju skup alata ciljane platforme pri implementaciji grafičkog sučelja [3].

1.3 qmake

qmake je alat koji sudjeluje u izgradnji aplikacija, biblioteka i drugih komponenti Qt-a, te pohranjuje potrebne informacije u *Makefile*. *Makefile* je datoteka koja sadrži skup instrukcija potrebnih za izgradnju programa i povezivanje datoteka projekta (na linux sustavima).

Osnovne informacije o projektu, potrebne *qmake*-u pri izgradnji *Makefile*-a, sadrži projekt (.pro) datoteka. Projektna datoteka se uglavnom sastoji od imena projekta, predloška projekta, konfiguracije koja se koristi te popisa modula, izvornih datoteka, zaglavlja, formi grafičkog sučelja i ostalih datoteka koje projekt koristi.

Osim što sadrži informacije o projektu, *qmake* automatski generira pravila za meta objektni *compiler* (*moc*) i *compiler* grafičkog sučelja (*uic*). *moc* opskrbljuje klase objekata s potrebnim kodom za međusobno povezivanje, dok *uic* pretvara XML format grafičkog sučelja u C++ datoteku [7].

1.4 Meta objektni sustav

Meta objektni sustav omogućava mehanizam signala i slotova za komunikaciju među objektima. Baziran je na tri stvari:

- QObject klasa koja pruža baznu klasu za objekte koji koriste meta objektni sustav.

- *Q_OBJECT* makronaredba unutar deklaracije klase koja omogućava upotrebu značajki meta objektnog sustava poput dinamičkih svojstva, signala i slotova.
- *Meta-Object Compiler (moc)* koji opskrbljuje svaku podklasu klase QObject s potrebnim kodom za korištenje značajki meta objektnog sustava.

moc pri čitanju C++ izvornih datoteka traži klase koje u deklaraciji sadrže *Q_OBJECT* makronaredbu te stvara nove C++ datoteke koje sadrže meta objektni kod za svaku od prethodnih klasa. Te datoteke se uključuju u izvorne datoteke klasa ili, češće, kompiliraju i povezuju s implementacijom klasa. Sustav meta objekata nudi i usluge za provjeru imena klasa, provjeru nasljeđivanja, dohvaćanje svojstava objekata te stvaranje novih instanci klasa [4].

QObject je bazna klasa svih Qt objekata. Implementirana je koristeći *Composite design pattern* koji omogućava uniformno tretiranje individualnog objekta i kompozicije objekata, kao i organizaciju objekata u objektna stabla. Kod kreiranja QObjekt-a s drugim objektom kao roditeljem, taj objekt će automatski dodati kreirani QObject u listu svoje djece. Roditelj preuzima dijete u svoje vlasništvo te ga automatski briše u svom destruktoru [4,5].

1.5 Signali i slotovi

Signali i slotovi se koriste za komunikaciju među objektima. Mehanizam signala i slotova jedno je od glavnih značajki Qt-a koje ga razlikuje od većine drugih softverskih okvira.

Kod stvaranja grafičkog sučelja, pri promjeni sadržaja jednog *widgeta* često želimo da drugi *widget*-i budu obaviješteni, tj. želimo da objekti, neovisno o tipu, imaju mogućnost međusobne komunikacije. Stariji skupovi alata ostvaruju ovakvu vrstu komunikacije pomoću tehnike zvane *callback*, pokazivača na funkciju. Ako želimo da nas funkcija koja se izvodi obavijesti o nekom događaju, proslijedimo joj pokazivač na drugu funkciju (*callback*), koja se poziva po potrebi. Međutim, ova tehnika ima dva nedostatka. *Callback* funkcija je usko povezana s funkcijom koja obrađuje događaj i nije osiguran poziv funkcije s ispravnim argumentima, odnosno nisu osigurani ispravni tipovi podataka. Drugi način je upotreba *Observer design pattern*, kod kojeg objekt koji se naziva subjekt sadrži listu objekata koji ovise o njemu, te pri mijenjanju stanja obavještava objekte s liste, najčešće koristeći njihove fukcije.

Signali i slotovi su Qt-ova alternativa *callback* tehnici i generalnije rješenje od *Observer design pattern*-a, ali zahtijevaju *moc*. Signal se emitira pri svakom novom događaju. Slot je funkcija koja se poziva kao reakcija na određeni signal. Qt-ovi *widget*-i imaju puno preddefiniranih signala i slotova ali postoji i mogućnost dodavanja vlastitih.

Kod mehanizam signala i slotova, osiguran je poziv funkcija s ispravnim argumentima, odnosno ispravnim tipovima podataka. S obzirom na to da su signature obje funkcije kompatibilne, uređivač koda nam pomaže pri pisanju koda da ne bi došlo do nepodudaranja u tipu. Signali i slotovi ne ovise jedni o drugima, odnosno nisu čvrsto povezani. Klasa koja emitira signal ne zna, niti joj je bitno, koji slot prima signal, ali imamo garanciju da ako povežemo signal sa slotom, taj slot će biti pozvan s parametrima signala u pravo vrijeme.

Signale i slotove mogu sadržavati sve klase koje nasljeđuju klasu QObject ili jednu od njenih podklasa poput QWidget-a. Objekti uglavnom emitiraju signale pri promjeni stanja da bi obavijestili druge objekte kojima je ta promjena od značaja. Slotovi služe za primanje signala, ali su i funkcije danog objekta. Kao što objekti ne znaju postoje li drugi objekti koji primaju njegov signal, tako ni slot ne zna postoji li signal povezan s njime. Također, jedan slot može biti povezan s velikim brojem signala i jedan signal može biti povezan s velikim brojem slotova. Time je omogućeno jednostavno povezivanje potpuno neovisnih komponenti Qt-a [6].

Primjer upotrebe mehanizma signala i slotova demostriran je sljedećim kodom.

```
#include <QObject>

class Brojac : public QObject
{
    Q_OBJECT

public:
    Brojac() { m_broj = 0; }

public slots:
    void postaviBroj(int broj);

signals:
    void brojPromijenjen(int noviBroj);

private:
    int m_broj;
};
```

Slika 1.1: Klasa *Brojac*

Klasa *Brojac* sadrži signal *brojPromijenjen()* kojim može poručiti drugim objektima da je došlo do promjene stanja njene varijable *m_broj*. Također sadrži slot kojem drugi objekti mogu poslati signal. Sve klase koje sadrže signale i slotove moraju imati makronaredbu *Q_OBJECT* u deklaraciji. Usto, trebaju nasljeđivati klasu QObject (direktno ili indirektno).

```
void Brojac::postaviBroj(int broj)
{
    if (broj != m_broj) {
        m_broj = broj;
        emit brojPromijenjen(broj);
    }
}
```

Slika 1.2: Implementacija slota *postaviBroj()*

emit funkcija emitira signal svaki put kada se promijeni vrijednost varijable *m_broj* klase *Brojac*.

U sljedećem isječku koda stvorena su dva objekta klase *Brojac*, te je povezan signal prvog objekta sa slotom drugog objekta pomoću funkcije *connect()*.

```
Brojac a, b;
QObject::connect(&a, &Brojac::brojPromijenjen,
                 &b, &Brojac::postaviBroj);

a.postaviBroj(15);     // a.m_broj = 15, b.m_broj = 15
b.postaviBroj(30);     // a.m_mroj = 15, b.m_broj = 30
```

Slika 1.3: Datoteke aplikacije

Pozivanjem funkcije *a.postaviBroj(15)* emitira se signal *brojPromijenjen(15)* objekta *a*. Time se aktivira slot *postaviBroj()* objekta *b*, točnije poziva se *b.postaviBroj(15)*. Tada se pak emitira signal *brojPromijenjen(15)* objekta *b*, ali se ignorira budući da nije povezan niti s jednim signalom. Veza između signala i slota može se prekinuti pozivom funkcije *disconnect()*.

1.6 QML aplikacije

QML je deklarativni jezik koji omogućava kreiranje sučelja pomoću vizualnih komponenti, te njihove interakcije i međusobnog odnosa. Stvoren je da bi se omogućilo dinamičko povezivanje komponenti i njihova ponovna upotreba i prilagodba unutar sučelja. Izgradnja sučelja koristeći QML dostupna je kroz Qt Quick modul pomoću kojeg je moguće povezati grafičko sučelje s C++ bibliotekama. Podršku za QML osigurava Qt QML modul koji definira i implementira QML jezik i strukturu sistema, te pruža API koji omogućuje proširivanje jezika s vlastitim klasama i integraciju QML koda sa JavaScript i C++ kodom. Dok Qt QML pruža jezik i strukturu QML aplikacija, Qt Quick modul omogućuje korištenje vizualnih komponenti, podršku za *model-view*, softverski okvir za animacije i druge funkcionalnosti pri izgradnji grafičkog sučelja [8,9].

1.7 Qt Creator

Qt Creator je C++, JavaScript i QML integrirano razvojno okruženje (*integrated development environment*, IDE) koje je dio Qt-ovog SDK-a (*software development kit*) za razvoj grafičkog sučelja. Značajke Qt Creatora [10]:

- Upravljanje projektima.
- Alati za dizajniranje grafičkog sučelja Qt Creator i Qt Quick.
- Uređivač koda koji nudi funkcionalnosti poput isticanja sintakse izvornog koda, automatskog kompletiranja koda i semantičke navigacije.
- Mogućnost izgradnje i pokretanja projekta pomoću *qmake* alata.
- Omogućeno testiranje i korištenje programa za pronalaženje pogrešaka (*debugger*).
- Mogućnost kreiranja instalacijskih paketa za mobilne uređaje prikladnih za objavljivanje.

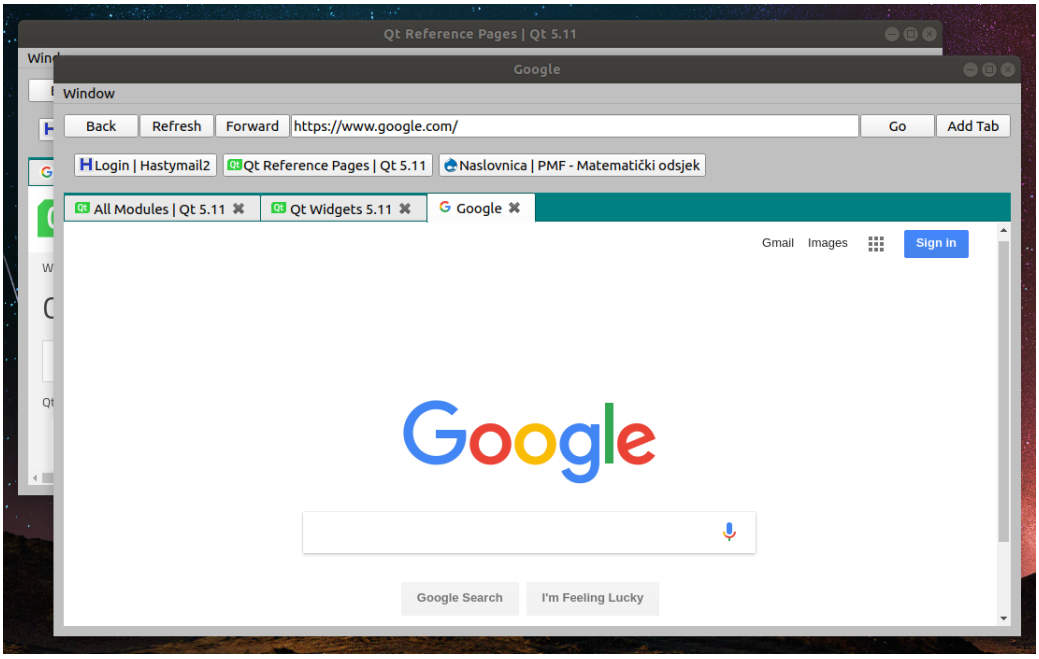
Qt Designer se koristi za izradu i dizajn grafičkog sučelja pomoću Qt Widget-a. Funkcionira na principu *what-you-see-is-what-you-get (WYSIWYG)*, odnosno „*što vidiš, to i dobiješ*". *Widget*-i i forme integrirani su s programskim kodom za upotrebu mehanizma signala i slotova, čime se lako definira njihovo ponašanje. Sve vrijednosti postavljene u Qt Designeru mogu biti promijenjene pomoću koda [11].

Poglavlje 2

Web preglednik

2.1 Izgled i funkcionalnosti

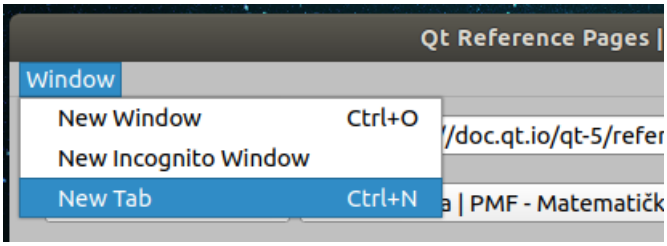
U ovome radu u svrhu demonstracije Qt biblioteke izrađen je i prikazan primjer web preglednika pisanog u C++ programskom jeziku. Aplikacija je napravljena pomoću Qt Creator-a, odnosno uporabom ugrađenog uređivača koda i Qt Designer-a, vizualnog uređivača za korisničko sučelje. Aplikacija se sastoji od izbornika, navigacijske trake, spremljenih web adresa (*bookmarks*) i tab *widget*-a s prikazom web stranica. Navigacijska traka sadrži pet dugmadi i linijski tekstualni editor za upis web adrese. Korisniku je omogućen pristup traženoj web stranici, mogućnost privatnog pristupa web sadržaju te korištenje tabova i više prozora.



Slika 2.1: Izgled web preglednika

Unutar prozora aplikacije kroz menu Window moguće je:

- dodati novi tab u postojeći prozor; fokus se prebacuje na novi tab
- otvoriti novi prozor aplikacije; fokus se prebacuje na novi prozor aplikacije
- otvoriti novi incognito prozor; fokus se prebacuje na novi prozor aplikacije koji ima naslov Incognito



Slika 2.2: Izgled Window menu-a

Navigacijskom trakom muguće je:

- klikom na dugme Back otvoriti prethodnu web stranicu u trenutnom tabu,
- klikom na dugme Refresh ponovno učitati trenutnu web stranicu u trenutnom tabu,
- klikom na dugme Forward otvoriti sljedeću web stranicu u trenutnom tabu,
- upisati željenu web adresu u linijski tekstualni editor te pritiskom na Enter otvoriti traženu web stranicu u trenutnom tabu,
- upisati željenu web adresu u linijski tekstualni editor te klikom na dugme Go otvoriti traženu web stranicu u trenutnom tabu,
- klikom na dugme Add Tab dodati novi tab u postojeći prozor; fokus se prebacuje na novi tab trenutno otvorenog prozora.

Interakcijom s dugmadi spremljenih web adresa moguće je:

- klikom na dugme s ikonom i naslovom spremljene web adrese otvoriti spremljenu web stranicu u novom tabu trenutno otvorenog prozora.

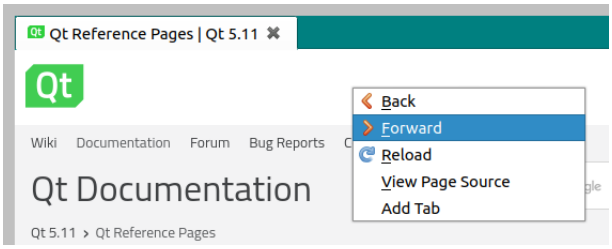
Kroz zaglavlje tab *widget*-a moguće je:

- klikom na zaglavlje taba koji trenutno nije prikazan promijeniti fokus na njega; prikazuje se web stranica tog taba,
- klikom na x ikonu u zaglavlju taba zatvoriti tab trenutno otvorenog prozora aplikacije, bilo da je fokus trenutno na njemu ili ne,
- pritiskom na zaglavlje taba i povlačenjem promijeniti redoslijed tabova u trenutno otvorenom prozoru,
- dvostrukim klikom na zaglavlje tab *widget*-a dodati novi tab u trenutno otvoreni prozor; fokus se prebacuje na novi tab.

Desnim klikom na otvorenu web stranicu moguće je:

- otvoriti prethodnu web stranicu u trenutnom tabu,
- otvoriti sljedeću web stranicu u trenutnom tabu,

- ponovno učitati trenutnu web stranicu u trenutnom tabu,
- otvoriti izvorni kod trenutne web stranice u trenutnom tabu,
- otvoriti novi tab u trenutno otvorenom prozoru aplikacije; fokus se prebacuje na novi tab trenutno otvorenog prozora.



Slika 2.3: Izgled izbornika prilikom desnog klika na stranicu

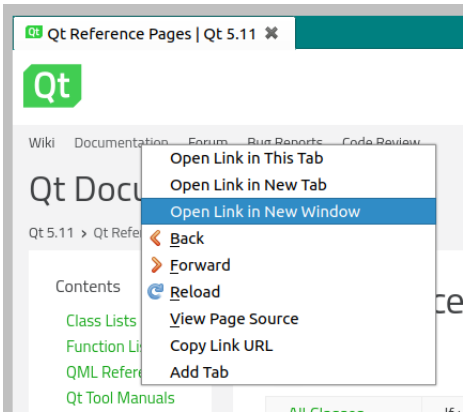
Lijevim klikom na poveznicu unutar otvorene web stranice moguće je:

- otvoriti web stranicu poveznice u trenutnom tabu.

Desnim klikom na poveznicu unutar otvorene web stranice moguće je:

- otvoriti web stranicu poveznice u trenutnom tabu,
- otvoriti web stranicu poveznice u novom tabu trenutno otvorenog prozora aplikacije; fokus se prebacuje na novi tab s prikazom web stranice poveznice,
- otvoriti web stranicu poveznice u novom prozoru aplikacije; fokus se prebacuje na novi prozor aplikacije, točnije na tab s prikazom web stranice poveznice u novom prozoru aplikacije,
- otvoriti prethodnu web stranicu u trenutnom tabu,
- otvoriti sljedeću web stranicu u trenutnom tabu,
- ponovno učitati trenutnu web stranicu u trenutnom tabu,
- otvoriti izvorni kod trenutne web stranice u trenutnom tabu,

- kopirati web adresu poveznice,
- otvoriti novi tab u trenutno otvorenom prozoru aplikacije; fokus se prebacuje na novi tab trenutno otvorenog prozora.



Slika 2.4: Izgled izbornika prilikom desnog klika na poveznicu

Automatizirane radnje aplikacije:

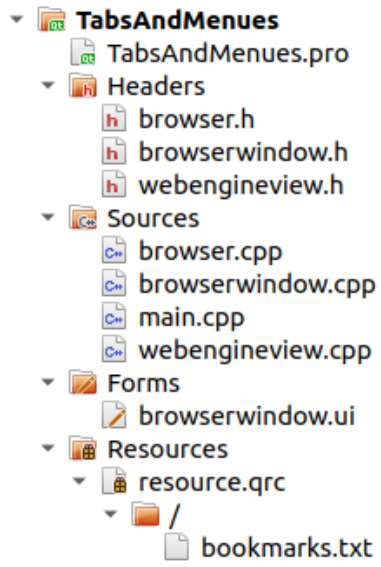
- u zaglavlju trenutnog taba prikazuju se ikona i naslov trenutno prikazane web stranice; mijenja se pri promjeni web stranice u trenutnom tabu,
- linijski tekst editor za unos web adrese uvijek sadrži web adresu stranice prikazane u trenutno otvorenom tabu prozora aplikacije; mijenja se pri promjeni web stranice u trenutnom tabu, odabirom nekog taba na kojemu trenutno nije fokus, te otvaranjem poveznice u novom tabu, otvaranjem spremljene web stranice u novom tabu ili dodavanjem novog taba s obzirom na to da se fokus u tom trenutku stavlja na novo otvoreni tab,
- naslov prozora aplikacije sadrži naslov web stranice u trenutno otvorenom tabu prozora; mijenja se pri promjeni web stranice u trenutnom tabu, odabirom nekog taba na kojemu trenutno nije fokus, te otvaranjem poveznice u novom tabu, otvaranjem spremljene web stranice u novom tabu ili dodavanjem novog taba s obzirom na to da se fokus u tom trenutku stavlja na novo dodani tab,
- naslov incognito prozora aplikacije ima nepromjenjiv naslov „Incognito”,
- prilikom zatvaranja zadnjeg taba prozora otvara se novi tab u prozoru.

2.2 Izrada

Pri izradi aplikacije korišteni su sljedeći Qt moduli i skupovi C++ klasa:

- Qt Core; modul Qt-a koji sadrži osnovne funkcije potrebne za rad aplikacija (QFile, QTextStream, QUrl, QVariant i QVector)
- Qt Gui; modul za integraciju grafičkog sučelja s aplikacijom (QContextMenuEvent i QCloseEvent)
- Qt Widgets; nadopunjuje Qt Gui s C++ klasama za prikaz widgeta u grafičkom sučelju (QAction, QAplication, QLineEdit, QMainWindow, QMenu, QMenuBar, QPushButton, QTabBar, QTabWidget i QWidget)
- Qt WebEngine Widgets; nadopunjuje Qt Widgets s C++ klasama za funkcionalnosti i prikaz web sadržaja (QWebEngineContextMenuData, QWebEnginePage, QWebEngineProfile, QWebEngineSettings i QWebEngineView)

Projekt aplikacije sastoji se od sljedećih datoteka:



Slika 2.5: Datoteke aplikacije

- TabsAndMenues.pro – projektna datoteka koja sadrži popis modula koji se koriste, popis izvornih datoteka i njihovih datoteka zaglavlja, popis datoteka grafičkog sučelja, resurse te dodatne postavke projekta,
- main.cpp – izvorna datoteka glavne funkcije,
- browser.cpp i browser.h – datoteke s implementacijom klase Browser koja upravlja prozorima aplikacije,
- browserwindow.cpp i browserwindow.h – datoteke s implementacijom klase BrowserWindow koja predstavlja prozor aplikacije
- webengineview.cpp i webengineview.h – datoteke s implementacijom klase WebEngineView koja je zadužena za prikaz web sadržaja,
- browserwindow.ui – datoteka koja generira kod grafičkog sučelja kreiranog u Qt Designer-u,
- bookmarks.txt – tekstualna datoteka s popisom spremljenih web adresa (*bookmarks*).

Module Qt-a koje koristimo potrebno je navesti u datoteci TabsAndMenues.pro. Varijabla *core* uključuje Qt Core, model Qt-a koji sadrži osnovne funkcije potrebne za rad aplikacija. Varijabla *gui* uključuje Qt Gui, modul koji sadrži bazne klase za komponente korisničkog grafičkog sučelja, točnije za sustav prozora, 2D i 3D grafike (OpenGL), slike, fontove, tipografiju i upravljanje događajima. Varijable *widgets* i *webenginewigets* uključuju modul Qt Widgets koji sadrži klase widgeta za prikaz u grafičkom sučelju (odvojio se od Qt Gui s verzijom Qt 5) i modul Qt WebEngine zaslužan za funkcionalnosti i prikaz web sadržaja.

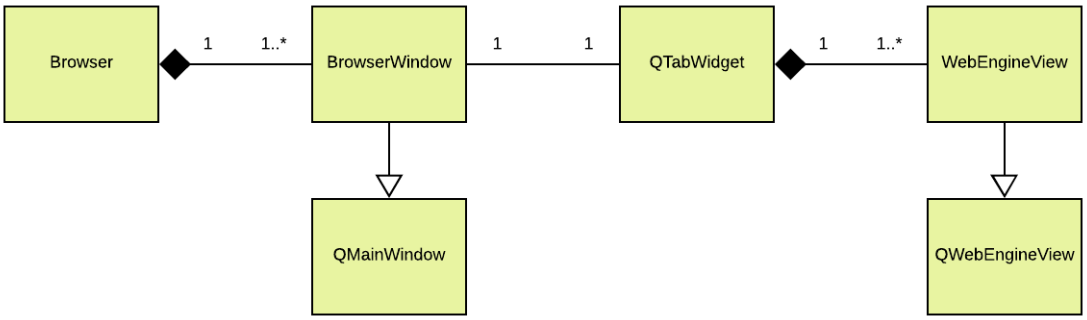
```
QT   += core gui widgets webenginewidgets

TARGET = TabsAndMenues
TEMPLATE = app

CONFIG += c++11

SOURCES += \
    main.cpp \
    browser.cpp \
    browserwindow.cpp \
    webengineview.cpp

HEADERS += \
    browser.h \
    browserwindow.h \
    webengineview.h

FORMS += \
    browserwindow.ui

RESOURCES += \
    resource.qrc
```

Slika 2.6: TabsAndMenues.pro

Varijablom *TARGET* definiramo ime projekta, dok *TEMPLATE* definira predložak koji se koristi pri generiranju projekta, u našem slučaju predložak aplikacije s grafičkim sučeljem. Neki od primjera drugih predložaka su *lib* za stvaranje biblioteka i *vcapp* za stvaranje Visual Studio projekta. Varijablom *CONFIG* specificiramo konfiguraciju projekta i opcije uređivača koda. Varijable *SOURCES*, *HEADERS*, *FORMS* i *RESOURCES* sadrže popis izvornih datoteka, zaglavlja, formi grafičkog sučelja i resursa korištenih u aplikaciji.

Klase

Odnos između klasa aplikacije prikazan je sljedećom slikom.



Slika 2.7: Klase aplikacije

Glavna klasa ove aplikacije je klasa Browser, a sastoji se od privatnog QWebEngineProfile-a koji sadrži postavke, skripte, *cookie*-je i listu posjećenih stranica koju dijeli sa svim stranicama koje pripadaju danom profilu. Također, sadrži privatni QVector klase BrowserWindow (lista otvorenih prozora), konstruktor i javnu funkciju za pokretanje novog prozora aplikacije.

```
class Browser
{
public:
    Browser();
    BrowserWindow *createWindow(bool offTheRecord = false);

private:
    QVector<BrowserWindow*> m_windows;
    QWebEngineProfile m_otrProfile;
};
```

Slika 2.8: Klasa Browser

Klasa BrowserWindow predstavlja jedan otvoreni prozor aplikacije. Sadrži privatne varijable QWebEngineProfile i Browser (pokazivač na roditelja), konstruktor, javnu funkciju za dohvačanje web stranice trenutno prikazanog taba (WebEngineView), prikazane slotove i funkciju *closeEvent()* zaduženu za brisanje prozora nakon zatvaranja. Nasljeđuje klasu QMainWindow koja predstavlja glavni prozor aplikacije u koji se mogu dodati drugi QWidget-i, kao i QToolBar, QDockWidget, QMenuBar i QStatusBar. QWidget je bazna klasa svih objekata korisničkog grafičkog sučelja. On prima i upravlja događajima nastalih

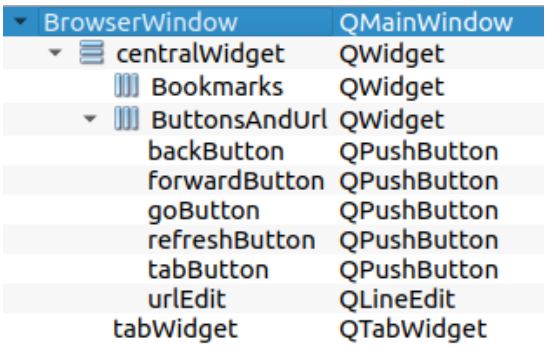
mišem ili tipkovnicom i crta svoju reprezentaciju na ekran. Widget koji nije ugrađen u drugi widget nazivamo prozor. u Qt-u, QMainWindow je najčešći tip prozora.

```
class BrowserWindow : public QMainWindow, public Ui::BrowserWindow
{
    Q_OBJECT

public:
    BrowserWindow(Browser *browser, QWebEngineProfile *profile);
    WebEngineView* currentTab();

protected:
    void closeEvent(QCloseEvent *event) override;

public slots:
    void handleUrlClicked();
    void handleUrlBack();
    void handleUrlRefresh();
    void handleUrlForward();
    void handleUrlChanged();
    void handleNewTab();
    void openInNewTab();
    void openInNewWindow();
    void handleBookmarks();
    void handleNewWindowTriggered();
    void handleNewIncognitoWindowTriggered();

private:
    Browser *m_browser;
    QWebEngineProfile *m_profile;
};
```

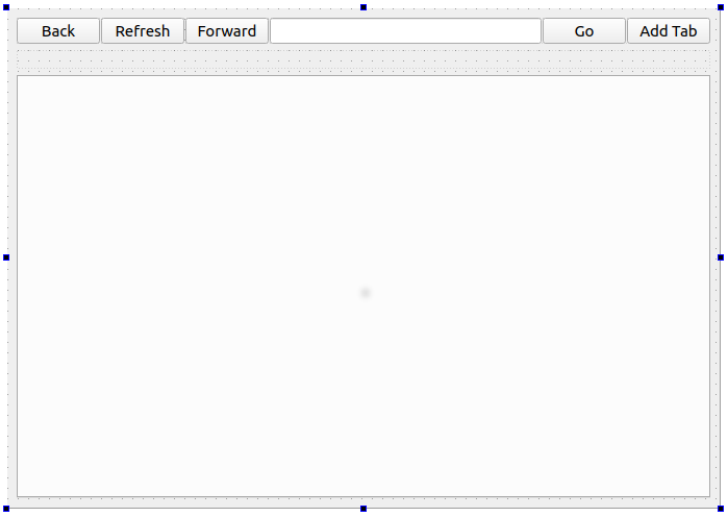
Slika 2.9: Klasa BrowserWindow

Objekte konstruirane u Qt Designer-u nasljeđuje od forme browserwindow.ui. Na ovaj način možemo pristupiti svim objektima korisničkog grafičkog sučelja (definiranih u formi) direktno iz klase, kao i koristiti njihove signale i slotove. Kreirana forma sastoji od navigacijske trake, okvira namijenjenog prikazu spremljenih web stranica i praznog QTabWidgeta. QTabWidget klasa pruža pregled widgeta na hrpi preko tabova. Sastoji se od zaglavlja s tabovima i područja za prikaz widgeta. Svaki tab je povezan s nekim widgetom, a prikazuje se samo widget povezan s trenutnim tabom. Navigacijska traka je predstavljena okvirom klase QWidget u koji je umetnuto pet dugmadi klase QPushButton i linijski tekstualni editor za upis web adrese klase QLineEdit. Okvir za spremljene adrese klase QWidget inicijalizira se pri otvaranju novog prozora aplikacije čitanjem tekstualne datoteke te stvaranjem

dugmadi klase QPushButton povezanih s pročitanim web adresama. Također, otvaranjem novog prozora aplikacije inicijalizira se prvi tab QTabWidgeta s prikazom web sadržaja.



Slika 2.10: Objekti grafičkog sučelja



Slika 2.11: Izgled grafičkog sučelja u Qt Designer-u

Svaki postojeći tab sastoji se od klase WebEngineView koja nasljeđuje klasu QWebEngineView. QWebEngineView je glavni widget Qt WebEngine modula koji omogućuje prikaz web stranica. Svaki QWebEngineView sadrži QWebEnginePage, klasu koja se satoji od glavnog okvira odgovornog za web sadržaj, povijest otvorenih web strnica i akcija pri interakciji s web sadržajem. Također, svaki QWebEnginePage koristi pripadni QWebEngineProfile za postavke, listu posjećenih web stranica itd. WebEngineView, osim što nasljeđuje klasu QWebEngineView, sadrži konstruktor, privatnu varijablu BrowserWindow (pokazivač na roditelja) i funkciju *contextMenuEvent()* zaduženu za stvaranje vlastitog menu-a prilikom interakcije s prikazanom web stranicom.

```
class WebEngineView : public QWebEngineView
{
    Q_OBJECT

private:
    BrowserWindow *w;

public:
    WebEngineView(BrowserWindow *w);

protected:
    void contextMenuEvent(QContextMenuEvent *event) override;
};
```

Slika 2.12: Klasa WebEngineView

Klase BrowserWindow i WebEngineView u sebi sadrže makronaredbu *Q_OBJECT* kako bi bila omogućena međusobna komunikacija, kao i komunikacija s objektima definiranim u grafičkom sučelju koji po definiciji nasljeđuju QWidget, a time i QObject klasu.

Implementacija funkcionalnosti

Funkcionalnosti menu-a Window u prozoru aplikacije definiramo unutar konstruktora klase BrowserWindow. S obzirom da nasljeđuje klasu QMainWindow omogućena je upotreba QMenuBar-a. Korištenjem opcija menu-a Window pozivaju se slotovi klase BrowserWindow koji se ovdje koriste kao javne funkcije klase. Slot *handleNewTab* dodaje WebEngineView s zadanom web stanicom kao novi tab u QTabWidget. Slotovi *handleNewWindowTriggered* i *handleNewIncognitoWindowTriggered* dodaju novi BrowserWindow u listu

otvorenih prozora klase Browser. U slučaju incognito prozora QWebEnginePage sadržan u WebEngineView-u ne koristi QWebEngineProfile klase Browser već prazan, novostvoreni profil.

```
QMenu *menu = new QMenu(tr("Window") );
menu->addAction(tr("New Window"), this, &BrowserWindow::handleNewWindowTriggered, QKeySequence::Open);
menu->addAction(tr("New Incognito Window"), this, &BrowserWindow::handleNewIncognitoWindowTriggered);
menu->addAction(tr("New Tab"), this, &BrowserWindow::handleNewTab, QKeySequence::New);
menuBar()->addMenu(menu);
```

Slika 2.13: Stvaranje menu-a glavnog prozora

Funkcionalnosti navigacijske trake također definiramo unutar konstruktora klase BrowserWindow spajanjem signala koje emitiraju widgeti navigacijske trake sa slotovima klase. Signali koji se koriste su *QPushButton::clicked* i *QLineEdit::returnPressed*. Slotovi *handleUrlBack*, *handleUrlRefreshed* i *handleUrlForward* pristupaju trenutno otvorenom tabu na kojem se koriste preddefinirane funkcije klase QWebEngineView. Slot *handleUrlClicked* postavlja web adresu, koja je napisanu u linijskom tekstualnom editoru, trenutno prikazanog taba.

```
connect(goButton, &QPushButton::clicked, this, &BrowserWindow::handleUrlClicked);
connect(urlEdit, &QLineEdit::returnPressed, this, &BrowserWindow::handleUrlClicked);

connect(backButton, &QPushButton::clicked, this, &BrowserWindow::handleUrlBack);
connect(refreshButton, &QPushButton::clicked, this, &BrowserWindow::handleUrlRefresh);
connect(forwardButton, &QPushButton::clicked, this, &BrowserWindow::handleUrlForward);
connect(tabButton, &QPushButton::clicked, this, &BrowserWindow::handleNewTab);
```

Slika 2.14: Povezivanje signala navigacijskog bara sa slotovima glavnog prozora

Izvedba dugmadi sa spremljenim web adresama realizirana je pomoću slota *handleBookmarks*. U konstruktoru klase BrowserWindow pristupa se sadržaju datoteke bookmarks.txt te se stvara novi WebEngineView za svaku pročitanu web adresu. Time se aktivira slot *handleBookmarks*, koji stvara dugme poveznice s ikonom i naslovom spremljene web adrese. Prilikom klika na dugme otvara se novi tab s prikazom spremljene web stranice.

Definiranje funkcionalnosti zaglavlja tab widgeta također je realizirano unutar konstruktora klase BrowserWindow. Koristimo klasu QTabBar i postavljamo mogućnost pomi-

canja i zatvaranja tabova. Povezujemo signal QTabBar-a *tabBarDoubleClicked* sa slotom *handleNewTab* klase BrowserWindow. Prilikom emitiranja signala QTabBar-a *tabCloseRequested* briše se kliknuti tab i WebEngineView. Brisanjem zadnjeg taba prozora otvara se novi tab.

```
QTabBar *tabBar = tabWidget->tabBar();
tabBar->setTabsClosable(true);
tabBar->setSelectionBehaviorOnRemove(QTabBar::SelectPreviousTab);
tabBar->setMovable(true);

connect(tabBar, &QTabBar::tabBarDoubleClicked, [this](int index) {
    if (index == -1)
        this->handleNewTab();
});
```

Slika 2.15: Definiranje funkcionalnosti zaglavlja tab widgeta

```
connect(tabWidget->tabBar(), &QTabBar::tabCloseRequested, [=](int index) {
    if (tabWidget->count() == 1) {
        this->handleNewTab();
        tabWidget->widget(index)->deleteLater();
    }
    else {
        tabWidget->widget(index)->deleteLater();
    }
});
```

Slika 2.16: Definiranje zatvaranja taba

QWebEngineView sadrži preddefinirane funkcionalnosti pri interakciji s web sadržajem. WebEngineView nasljeđuje spomenute funkcionalnosti, a dodavanje vlastitih omogućeno je kroz funkciju *contextMenuEvent()*. Korištenjem opcija kontekstnog menu-a pozivaju se slotovi klase BrowserWindow. Slot *openInNewTab* u trenutno otvoreni prozor dodaje tab s prikazom web stranice poveznice. Slot *openInNewWindow* otvara novi prozor aplikacije te dodaje tab s prikazom web stranice poveznice.

```
QAction *inNewTab;
inNewTab = new QAction("Open Link in New Tab", this);
inNewTab->setData(QVariant(page()->contextMenuData().linkUrl()));
menu->insertAction(before, inNewTab);
connect(inNewTab, &QAction::triggered, w, &BrowserWindow::openInNewTab);
QAction *inNewWindow;
inNewWindow = new QAction("Open Link in New Window", this);
inNewWindow->setData(QVariant(page()->contextMenuData().linkUrl()));
menu->insertAction(before, inNewWindow);
connect(inNewWindow, &QAction::triggered, w, &BrowserWindow::openInNewWindow);
```

Slika 2.17: Umetanje funkcionalnosti interakcije s web stranicom

Prilikom promjene taba fokus se prebacuje na kliknuti tab, a QLineEdit i naslov prozora prilagođavaju se s ozbirom na trenutno prikazanu web stranicu. To je omogućeno povezivanjem signala QTabWidget-a *currentChanged* sa slotom QTabWidget-a *setCurrentIndex* i slotom BrowserWindow-a *handleUrlChanged* koji prilagođava tekst QLineEdit-a i naslov trenutno otvorenog prozora.

```
connect(tabWidget, &QTabWidget::currentChanged, tabWidget, &QTabWidget::setCurrentIndex);
connect(tabWidget, &QTabWidget::currentChanged, this , &BrowserWindow::handleUrlChanged);
```

Slika 2.18: Umetanje funkcionalnosti interakcije s web stranicom

Kod promjene web stranice prikazane u trenutnom tabu potrebno je promjeniti ikonu i naslov koji se prikazuju u zaglavlju taba. To se postiže izvođenjem sljedećih linija koda aktiviranih pri emitiranju signala klase QWebEngineView *titleChanged* i *iconChanged*;

```
connect(m_webview, &WebEngineView::titleChanged, [this](const QString &title) {
    int index = tabWidget->currentIndex();
    tabWidget->tabBar()->setTabText(index, title);
    if (this->windowTitle() != "Incognito") this->setWindowTitle(title);
});
connect(m_webview, &WebEngineView::iconChanged, [this](const QIcon &icon) {
    int index = tabWidget->currentIndex();
    tabWidget->tabBar()->setTabIcon(index, icon);
});
```

Slika 2.19: Umetanje funkcionalnosti interakcije s web stranicom

Poglavlje 3

Zaključak

U ovome radu predstavljena je biblioteka Qt5. Ukratko je opisana povijest Qt-a, alati koji se koriste pri kreiranju aplikacija i osnovne značajke poput mehanizma signala i slotova. U svrhu demonstracije Qt biblioteke izrađen je i prikazan primjer web preglednika pisanog u C++ programskom jeziku. Kroz rad su opisani izgled i funkcionalnosti aplikacije napravljene pomoću Qt Creator-a, odnosno uporabom ugrađenog uređivača koda i Qt Designer-a, vizualnog uređivača za korisničko sučelje. Za dodavanje funkcionalnosti aplikacije korišteni su Qt moduli Qt Core, Qt Gui, Qt Widgets i Qt WebEngine Widgets. Korisniku je kroz aplikaciju omogućen pristup traženoj web stranici, mogućnost privatnog pristupa web sadržaju te korištenje tabova i više prozora.

Literatura

[1] M. Blanchette, M. Summerfield, C++ GUI Programming with Qt 4, Second Edition, Prentice Hall, 2008.

[2] Qt History, https://wiki.qt.io/Qt_History (kolovoz 2018.)

[3] Qt (software), https://en.wikipedia.org/wiki/Qt_(software) (kolovoz 2018.)

[4] The Meta-Object System, http://doc.qt.io/qt-5/metaobjects.html (kolovoz 2018.)

[5] QObject Class, http://doc.qt.io/qt-5/qobject.html (kolovoz 2018.)

[6] Signals & Slots, http://doc.qt.io/qt-5/signalsandslots.html (kolovoz 2018.)

[7] qmake Manual, http://doc.qt.io/qt-5/qmake-manual.html (kolovoz 2018.)

[8] Qt QML, http://doc.qt.io/qt-5/qtqml-index.html (kolovoz 2018.)

[9] QML Applications, http://doc.qt.io/qt-5/qmlapplications.html (kolovoz 2018.)

[10] IDE Overview, http://doc.qt.io/qtcreator/creator-overview.html (kolovoz 2018.)

[11] Qt Designer Manual, http://doc.qt.io/qt-5/qtdesigner-manual.html (kolovoz 2018.)

Sažetak

Konstrukcija grafičkog sučelja u biblioteci Qt5

U diplomskom radu predstavljena je biblioteka Qt5, točnije njena podrška pri izradi korisničkog grafičkog sučelja. Ukratko je opisana povijest Qt-a, alati koji se koriste pri kreiranju aplikacija i osnovne značajke poput mehanizma signala i slotova. U svrhu demonstracije Qt biblioteke izrađen je i prikazan primjer web preglednika pisanog u C++ programskom jeziku. Aplikacija je napravljena pomoću Qt Creator-a, odnosno uporabom ugrađenog uređivača koda i Qt Designer-a, vizualnog uređivača za korisničko sučelje.

Ključne riječi: Qt, Qt Creator, Qt Designer, aplikacija, web preglednik, grafičko sučelje

Summary

Construction of graphical user interface using Qt5

This thesis describes a Qt5 framework and its support for creating a graphical user interface. An insight is given into the brief history of Qt, tools used while crating an application and its features like signal-slot mehanism. For the purpose of demonstrating the Qt library, an example of a web browser written in the C ++ programming language was created and displayed. The application was created using Qt Creator, more specifically its built-in code editor and Qt Designer, a visual user interface editor.

Key words: Qt, Qt Creator, Qt Designer, application, web browser, graphical user interface

Životopis

Edi Ibriks rođen je 5. studenog 1990. u Rijeci. 2005. godine završava Osnovnu školu Podmurvice u Rijeci i iste godine upisuje se u Gimnaziju Andrije Mohorovičića Rijeka u Rijeci, prirodoslovno - matematički smjer. 2010. godine upisuje Preddiplomski sveučilišni studij matematike na Prirodoslovno - matematičkom falkultetu u Zagrebu, Matematičkom odsjeku kojeg završava 2015. godine. Iste godine upisuje Diplomski sveučilišni studij matematike na spomenutom fakutetu; smjer Računarstvo i matematika.

Prilozi

Uz elektroničku verziju diplomskog rada u obliku PDF datoteke i sažetka u latex formatu, na priloženom CD-u nalazi se i cjelokupan izvorni kod programa predstavljenog u drugom poglavlju.