Izvedba grafičkog korisničkog sučelja aplikacije za Windows platformu pomoću C++ Buildera

Kovačević, Danijel

Sveučilište
Sjever

Završni rad br. 13/RINF/2024

Izvedba grafičkog korisničkog sučelja aplikacije za Windows platformu pomoću C++ Buildera

Danijel Kovačević, 2443000648

Đurđevac, rujan 2024. godine

Sveučilište
Sjever

Računarstvo i informatika

Završni rad br. 13/RINF/2024

Izvedba grafičkog korisničkog sučelja aplikacije za Windows platformu pomoću C++ Buildera

Student:

Danijel Kovačević, 2443000648

Mentor:

mr.sc Vladimir Stanisavljević, dipl. ing.

Đurđevac, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Računarstvo i informatika

STUDIJ Računarstvo i informatika

PRISTUPNIK Danijel Kovačević | MATIČNI BROJ 2443000648

DATUM 15.09.2024. | KOLEGIJ Uvod u objektno orijentirano programiranje

NASLOV RADA Izvedba grafičkog korisničkog sučelja aplikacije za Windows platformu pomoću C++ Buildera

NASLOV RADA NA ENGL. JEZIKU Implementation of Grafical User Interface of an Windows application by using C++ Builder

MENTOR Vladimir Stanisavljević | ZVANJE viši predavač

ČLANOVI POVJERENSTVA

1. doc.dr.sc. Tomislav Horvat - predsjednik
2. Leon Horvat, predavač - član
3. mr.sc. Vladimir Stanisavljević, v. pred. - mentor
4. pred. Dražen Crčić, dipl.ing. - zamjenski član
5.

Zadatak završnog rada

BROJ 13/RINF/2024

OPIS

Izgradnja grafičkih korisničkih sučelja (GUI) stolnih aplikacija je složenen zadatak. Brojna razvojna okruženja i platforme pri izgradnji prefereiraju jezike koji podržavaju Objektno orijentirane paradigme (OOP) kojima se lakše izvode složene hijerarhije komponenti korisničkog sučelja. Embracadero C++ Builder je jedna od takvih platformi koja nudi komletnu kolekciju klasa za izgradnju korisničkog sučelja na više platformi. Za demonstracjiu mogućnosti platforme potrebno je za proizvoljnu platformu izraditi složeniju aplikaciju koja koristi C++ i OOP i na konkretnim implementacijskim detaljima prikazati prednosti OOP za razvoj GUI aplikacija.
U radu je potrebno:
* razmotriti prednosti i nedostatke primjene OOP za izgradnju stolne aplikcije s grafičkim korisničkim sučeljem
* detaljno opisati mogućnosti Embracader C++ Buildera i ukratko usporediti njegove mogućnosti s konkurentskim platformama
* izraditi složeniju korisničku aplikaciju koja demonstrira većinu mogućnosti platforme

ZADATAK URUČEN 16. 9. 2024 | POTPIS MENTORA

SVEUČILIŠTE SJEVER

Sažetak

Ovaj završni rad bavi se objektno orijentiranim razvojem grafičkih korisničkih sučelja aplikacija koristeći C++ Builder, razvojno okruženje alata RAD Studio tvrtke Embarcadero. Ovaj alat je poznat po svojoj jednostavnosti u izradi aplikacija za Windows platformu iako omogućuje izradu aplikacija i za druge operativne sustave. Uspješnost pojedine aplikacije uvelike ovisi o kvalitetno izrađenom korisničkom sučelju, a ovo razvojno okruženje baš tu pruža velike mogućnosti. U ovom radu je kroz izradu jedne složenije aplikacije demonstrirano kako se uz korištenje programske kolekcije vizualnih komponenti i programskog jezika C++, mogu izraditi aplikacije s funkcionalnim i intuitivnim korisničkim sučeljima.

Namjena same aplikacije je pomoć u organizaciji i upravljanju osobnim financijama. Pored mogućnosti evidentiranja svih prihoda i troškova koji su tablično prikazani, u aplikaciji je moguće kreirati i ispisivati razna izvješća kako bi se pratile transakcije po računima korisnika. Spajanjem na vanjsku pristupnu točku, aplikacija nudi prikaz važeće tečajne liste, ima kalkulator za pretvaranje vrijednosti valuta, te kalkulator za provjeru osobnog identifikacijskog broja (ili OIB). U ovoj aplikaciji uz pomoć kompletne kolekcije klasa koje nudi C++ Builder, demonstrirano je izvođenje složene hijerarhije komponenti korisničkog sučelja koju primjena objektno orijentirane paradigme uvelike olakšava.

Ključne riječi: objektno orijentirano programiranje, grafičko korisničko sučelje, C++ Builder, GUI, VCL, OOP, korisničko iskustvo, dizajn aplikacija, C++ programski jezik, vizualne komponente

Summary

This thesis focuses on the object-oriented development of graphical user interfaces for applications using C++ Builder, a development tool from Embarcadero's RAD Studio. Known for its ease of use in creating applications for the Windows platform, C++ Builder also supports the development of applications for other operating systems. The success of an application largely depends on a well-designed user interface, and this development tool provides excellent opportunities in that regard. In this thesis, through the creation of a complex application, it is demonstrated that by utilizing the Visual Component Library (VCL) and the C++ programming language, it is possible to develop applications with functional and intuitive user interfaces.

The purpose of the application is to assist in organizing and managing personal finances. In addition to allowing the user to record all income and expenses, which are displayed in a table, the application can generate and print various reports to monitor transactions across user accounts. By connecting to an external access point, the application also provides access to the current exchange rate list and includes both a currency conversion calculator and a personal identification number (OIB) validation tool. This application demonstrates the construction of a complex hierarchy of user interface components, significantly simplified by employing the Object-Oriented Paradigm (OOP) and the full set of classes provided by C++ Builder.

Keywords: object-oriented programming, graphical user interface, C++ Builder, GUI, VCL, OOP, user experience, application design, C++ programming language, visual components

Popis korištenih kratica

API Application Programming Interface
Skup pravila, protokola i alata koji omogućuju različitim softverskim aplikacijama međusobnu komunikaciju

ERD Entity-Relationship Diagram
Dijagram koji vizualno prikazuje strukturu baze podataka

GPS Global Positioning System
Globalni navigacijski satelitski sustav

GUI Graphic User Interface
Grafičko korisničko sučelje

HNB Hrvatska narodna banka

IDE Integrated Development Environment
Integrirano razvojno okruženje

JSON JavaScript Object Notation
Format za razmjenu podataka

REST Representational State Transfer
Način komunikacije između klijenta i poslužitelja putem HTTP protokola

SQL Structured Query Language
Strukturirani jezik koji se koristi u relacijskim bazama podataka.

STL Standard Template Library
Standardna biblioteka predložaka u C++ programskom jeziku

URL Uniform Resource Locator
Jedinstveni lokator resursa na internetu

UI User Interface
Korisničko sučelje

UX User Expirience
Korisničko iskustvo

VCL Visual Component Library
Biblioteka vizualnih komponenti

XML Extensible Markup Language
Format koji se koristi za strukturiranje, pohranu i prijenos podataka u tekstualnom obliku

Sadržaj

1. Uvod

Razvojem informatike i digitalnih uređaja, a u svrhu što jednostavnijeg njihovog korištenja, pojavila se potreba i za razvojem aplikacija preko kojih se vrši interakcija između uređaja i korisnika. U informatici, za razliku od sustavskih, uslužnih i pomoćnih programa te programskih jezika, aplikacija predstavlja primjenski ili korisnički program, odnosno skup naredbi i uputa koji omogućuje izvršenje neke zadaće. Aplikacija [1] može biti djelomično specijaliziran program poput onih za obradu teksta ili upravljanje bazom podataka, a često se pod time podrazumijeva i program načinjen po mjeri samo jednoga posla za potrebe nekog poduzeća ili konkretnog sustava.

Aplikacije s grafičkim sučeljem su najčešći tipovi aplikacija. Pored funkcionalnosti, *korisničko iskustvo* [2] (ili eng. User Experience – UX) igra veliku ulogu u određivanju njihove kvalitete. Intuitivno i kvalitetno dizajnirano *korisničko sučelje* [3] (ili eng. User Interface – UI) često čini razliku između uspješnog i neuspješnog softverskog rješenja. Izradom grafičkih korisničkih sučelja bavit ćemo se i u ovom završnom radu.

Na početku rada prvo ćemo se upoznati sa C++ programskim jezikom kojeg ćemo koristiti unutar razvojnog okruženja C++ Builder. Slijedeće treće poglavlje nam donosi detaljniji opis ovog razvojnog okruženja u kojem ćemo prikazati njegove mogućnosti kao snažnog alata za razvoj *grafičkih korisničkih sučelja* [4] (ili GUI – engl. Graphical User Interface) na Windows platformi. Opisat ćemo njegove glavne značajke, a posebno *programsku kolekciju vizualnih komponenti* (ili VCL - eng. Visual Component Library) koja omogućuje brzu izradu aplikacija s vrlo malo pisanja programskog koda.

Spomenut ćemo i *FireMonkey*, razvojni okvir koji omogućava stvaranje multiplatformskih aplikacija te uz pomoć vektorske grafike omogućava programerima dijeljenje kôda između različitih operativnih sustava kao što su Windows, macOS-a, iOS-a i Android. Tu je i *Firebird Database Access Components* (ili FireDAC), alat koji pruža jednostavan pristup različitim vrstama baza podataka, uključujući Oracle, Microsoft SQL Server, MySQL, SQLite i mnoge druge. Opisat ćemo i RAD Server koji omogućava izradu skalabilnih i sigurnih RESTful web servisa koristeći XML i JSON formate, a koji podržava autentifikaciju korisnika, push obavijesti, geolociranje i pohranu podataka. U tom poglavlju još ćemo navesti nekoliko integriranih razvojnih okruženja koji se koriste za izradu aplikacija s grafičkim korisničkim sučeljima te ih usporediti s C++ Builderom. Spomenut ćemo prednosti i nedostatke C++ Buildera u usporedbi s tim konkurentskim platformama.

Četvrto poglavlje nam donosi pregled samog sučelja razvojnog alata RAD Studio, opis alatnih traka i dijaloga. Opisat ćemo način kreiranja i pokretanja novog projekta, stvaranje novih

dijaloga, postavljanje komponenti na dijaloge i podešavanje njihovih postavki. Bit će prikazan i način pisanja koda i dodavanje funkcija komponentama ovisno o događajima koji se mogu izvršiti na pojedinoj komponenti.

Na kraju ćemo u petom poglavlju kroz izradu aplikacije za upravljanje osobnim financijama, prikazati praktične primjere razvoja grafičkog korisničkog sučelja korištenjem programske kolekcije vizualnih komponenti ovog razvojnog alata te C++ programskog jezika. Prikazat ćemo primjer izrade korisničkih sučelja, koristeći osnovne elemente poput prozora, dijaloga, listi, gumba, okvira itd. Ključne značajke svih tih komponenti su njihova svojstva, metode i događaji, te njihova jednostavna implementacija.

Namjena same aplikacije je, kako smo već napomenuli, pomoć u organizaciji i upravljanju osobnim financijama i sastoji se od nekoliko dijaloga. Ti dijalozi nam nude tablično prikazane podatke o svim računima korisnika i svim izvedenim transakcijama. Postoji mogućnost kreiranja izvještaja za obavljene transakcije, spremanje tih izvještaja u pdf obliku, a isto tako postoji i mogućnost ispisa tih izvještaja. Na izvještaju je moguće ispisati i bar kod iz transakcije uz pomoć kojeg je moguće plaćanje računa obaviti putem mobilnog bankarstva ili sl. Aplikacija sadrži funkcionalnost prikaza aktualne tečajne liste Hrvatske narodne banke koju putem REST servisa dohvaća sa njihove pristupne točke. Na istom dijalogu implementiran je i kalkulator valuta kojim je moguće svaku odabranu valutu iz tablice s tečajem preračunati u euro. Aplikacija podržava višejezičnost, odnosno nudi mogućnost odabira prikaza sučelja na tri jezika.

U uvodu sam ukratko najavio sadržaj ovog rada, a u nastavku ćemo se detaljnije pozabaviti najavljenim te pokušati ovaj rad kroz teorijski i praktični dio pretvoriti u jednu cjelinu koja bi čitatelju mogla dati bar mali uvod u širok spektar mogućnosti objektno orijentiranog programiranja, C++ programskog jezika i C++ Buildera kao alata za izradu kvalitetnih stolnih aplikacija.

2. Programski jezik C++

C++ programski jezik [5] je jezik opće namjene i srednje razine koji je razvijen kao proširenje programskog jezika C. Ova evolucija jezika C donijela je niz novih mogućnosti i koncepta, čineći C++ jednim od važnijih jezika u svijetu računalstva. Jezik je prvotno razvijen od strane Bjarna Stroustrupa u Bell Laboratories tijekom 1980-ih godina, a od tada je postao ključan alat za razvoj različitih softverskih sustava.

Tih godina najvažniji programski model bio je proceduralno strukturirano programiranje. Proceduralno programiranje zasniva se na promatranju programa kao niza jednostavnih programskih odsječaka, odnosno procedura. Svaka procedura je konstruirana tako da obavlja jedan manji zadatak, a cijeli se program sastoji od niza procedura koje sudjeluju u rješavanju zadatka. Kako bi koristi od ovakve podjele programa bile što izraženije, smatralo se dobrom programerskom taktikom odvojiti proceduru od podataka koje ona obrađuje. Time je bilo moguće pozvati proceduru za različite ulazne podatke i na taj način iskoristiti je na više mjesta.

Strukturirano programiranje je samo dodatak na proceduralni model. Ono definira niz osnovnih jezičnih konstrukcija, kao što su petlje, grananja i pozivi procedura, koje unose red u programe i čine samo programiranje daleko jednostavnijim, a napisani kôd čitljivijim. Ovakav programski pristup je bio vrlo uspješan do kasnih osamdesetih godina dvadesetog stoljeća, kada su njegovi nedostaci postajali sve očitiji. Naime, odvajanje podataka i procedura čini programski kôd težim za čitanje i razumijevanje. Prirodnije je o podacima razmišljati preko operacija koje možemo obaviti nad njima.

Nadalje, pokazalo se složenim istodobno razmišljati o problemu i odmah strukturirati rješenje. Umjesto rješavanja problema, programeri su mnogo vremena provodili pronalazeći načine da programe usklade sa zadanom strukturom. Također, današnji programi se pokreću pomoću miša, prozora, izbornika i dijaloga. Programiranje je pogonjeno događajima za razliku od starog, sekvencijalnog načina. Proceduralni programi su korisniku, u trenutku kada je bila potrebna interakcija korisnika (npr. zahtjev za ispis na pisaču) prikazivali ekran nudeći mu opcije. Ovisno o odabranoj opciji, izvođenje kôda se usmjeravalo na određeni programski odsječak. Pogonjeno događajima znači da se program ne odvija po unaprijed određenom slijedu, već se programom upravlja pomoću niza događaja. Događaja ima raznih: pomicanje miša, pritisak na tipku, izbor stavke iz izbornika i slično. Sada su sve opcije dostupne istodobno, a program postaje interaktivan, što znači da promptno odgovara na korisnikove zahtjeve.

Kako bi se takvi zahtjevi jednostavnije proveli u praksi, razvijen je objektni pristup programiranju. Osnovna ideja je „razbiti“ program u niz zatvorenih cjelina (objekata) koje međusobno surađuju u rješavanju problema. Umjesto specijaliziranih procedura koje

manipuliraju proizvoljnim podacima, radimo s objektima koji objedinjuju podatke i operacije nad tim podacima. Objekti u programskom kôdu su apstrakcije kojima predstavljamo objekte iz stvarnosti. Pri tome je važno što objekt radi, a ne kako on to radi. To omogućava da se pojedini objekt može po potrebi izbaciti i zamijeniti drugim koji će istu zadaću obaviti bolje.

Četiri su važna svojstva C++ jezika koja ga čine objektno orijentiranim:

- apstrakcija
- enkapsulacija
- nasljeđivanje
- polimorfizam

U nastavku ćemo ćemo ukratko opisati sve značajke objektno orijentiranog programiranja.

2.1. Apstrakcija

Apstrakcija podataka [6] jedna je od važnijih značajki objektno orijentiranog programiranja u C++. Apstrakcija znači prikazivanje samo bitnih informacija i skrivanje detalja. Apstrakcija podataka odnosi se na pružanje samo bitnih informacija o podacima vanjskom svijetu, skrivanje pozadinskih detalja ili implementacije. Razmotrimo primjer iz stvarnog života čovjeka koji vozi automobil. Čovjek samo zna da će pritiskom na papučicu gasa povećati brzinu automobila ili kočenjem zaustaviti auto, ali ne zna na koji način se pritiskom na papučicu gasa brzina zapravo povećava. To je ono što je apstrakcija.

Apstrakciju u C++ možemo implementirati koristeći klase. Klasa nam pomaže grupirati podatke i funkcije koristeći dostupne specifikatore pristupa. Klasa može odlučiti koji će član podataka biti vidljiv vanjskom svijetu, a koji ne. Još jedna vrsta apstrakcije u C++ mogu biti datoteke zaglavlja. Na primjer, razmotrimo metodu pow() prisutnu u datoteci zaglavlja math.h. Kad god trebamo izračunati potenciju nekog broja, jednostavno pozovemo funkciju pow() koja se nalazi u datoteci zaglavlja math.h i prosljeđujemo joj broj kao argument bez poznavanja temeljnog algoritma prema kojem funkcija zapravo izračunava potenciju tog broja.

2.2. Enkapsulacija

U objektno orijentiranom programiranju enkapsulacija [7] se definira kao povezivanje podataka i funkcija koje njima upravljaju u jednu cjelinu. Razmotrimo primjer enkapsulacije iz stvarnog života. Unutar neke tvrtke postoje različiti odjeli poput odjela za računovodstvo, odjela za financije, odjela za prodaju itd. Odjel za financije obrađuje sve financijske transakcije i vodi evidenciju svih podataka povezanih s financijama. Isto tako, odjel prodaje bavi se svim poslovima vezanim uz prodaju i vodi evidenciju svih prodaja. Može se pojaviti situacija da iz

nekog razloga djelatnik odjela financija treba sve podatke o prodaji u pojedinom mjesecu. U tom slučaju nije mu dopušten izravan pristup podacima odjela prodaje. Prvo će se morati obratiti nekom drugom službeniku u odjelu prodaje, a zatim od njega tražiti određene podatke. To je ono što je enkapsulacija. Ovdje su podaci prodajnog odjela i zaposlenici koji njima mogu manipulirati enkapsulirani pod jednim imenom “prodajni odjel”.

2.3. Nasljeđivanje

Sposobnost klase da izvede svojstva i karakteristike iz druge klase zove se nasljeđivanje [8]. Nasljeđivanje je također jedna od važnijih značajki objektno orijentiranog programiranja. Klasa koja nasljeđuje svojstva od druge klase naziva se podklasa ili izvedena klasa. Klasa čija svojstva nasljeđuje podklasa naziva se osnovna klasa ili nadklasa. Nasljeđivanje podržava koncept "ponovnog korištenja", tj. kada želimo stvoriti novu klasu, a već postoji klasa koja uključuje dio kôda koji želimo, možemo izvesti našu novu klasu iz postojeće klase. Čineći to, ponovno koristimo atribute i metode postojeće klase.

2.4. Polimorfizam

Polimorfizam [9] možemo definirati kao sposobnost nekog objekta da bude prikazan u više oblika. Npr. neka osoba u isto vrijeme može biti student, zaposlenik, sportaš i sl., čime iskazuje različito ponašanje u različitim situacijama. Isto tako i objekti i funkcije mogu pokazivati različito ponašanje u različitim slučajevima. Ponašanje ovisi o vrstama podataka koji se koriste u tim funkcijama. U objektno orijentiranom programiranju to se zove preopterećenje operatora i preopterećenje funkcija. Preopterećenje funkcija je korištenje jednog naziva funkcije za izvođenje različitih vrsta zadataka. Polimorfizam se intenzivno koristi u implementaciji nasljeđivanja.

2.5. Klasa

Klasa [10] je još jedna značajka objektno orijentiranog programiranja koja omogućuje grupiranje podataka i metoda unutar jedne cjeline. To je korisnički definiran tip podataka koji sadrži vlastite podatke i funkcije članova kojima se može pristupiti i koristiti stvaranjem instance te klase. Npr. razmotrimo automobil kao neku klasu. Može postojati mnogo automobila s različitim imenima i markama, ali svi će dijeliti neka zajednička svojstva: svi će imati 4 kotača, maksimalnu brzinu, prijeđene kilometre itd. Dakle, ovdje je automobil klasa, a kotači,

ograničenje brzine i kilometraža su njihova svojstva. Možemo reći da je klasa nacrt koji predstavlja grupu objekata koji dijele neka zajednička svojstva i ponašanja.

Objekt je instanca klase, entitet koji se može identificirati s nekim karakteristikama i ponašanjem. Definiranje klase ne zauzima mjesto u memoriji, ali kada se instancira, tj. kada se kreira objekt, tom objektu se alocira lokacija u memoriji.

2.6. Ostale značajke

Još jedna od značajki C++ programskog jezika je i podrška za pokazivače, što omogućuje izravno upravljanje memorijom i pristup nižoj razini računalnog sustava. No, s tom slobodom dolazi odgovornost programera jer upravljanje memorijom može biti izvor grešaka ako nije pažljivo provedeno.

Isto tako treba spomenuti i *Standardnu biblioteku* C++-a (ili eng Standard Template Library - STL) [11] koja pruža bogat skup gotovih funkcionalnosti, uključujući kontejnere poput vektora i lista, algoritme za sortiranje i pretraživanje, te različite vrste iteratora. STL često olakšava i ubrzava proces razvoja softvera, omogućujući programerima korištenje testiranih i optimiziranih implementacija određenih funkcionalnosti. Svaki objekt svojoj okolini pruža isključivo podatke i operacije koji su neophodni da bi okolina objekt mogla koristiti. Ti javno dostupni podaci zajedno s operacijama koje ih prihvaćaju ili vraćaju čine sučelje (engl. interface) objekta. Programer koji će koristiti taj objekt više se ne mora zamarati razmišljajući o načinu na koji objekt iznutra funkcionira, on jednostavno preko sučelja traži od objekta određenu uslugu.

C++ je široko korišten programski jezik za razvoj operativnih sustava, igara, sustava za upravljanje bazama podataka, algoritama umjetne inteligencije, mobilnih aplikacija i mnogo čega drugoga. Njegova svestranost i performanse čine ga jednim od bitnijih jezika u svijetu programiranja.

Sva ove značajke C++-a, odnosno objektno orijentirane paradigme, igraju i veliku ulogu u radu s C++ Builderom, čineći razvoj kompleksnih, interaktivnih aplikacija jednostavnijim i intuitivnijim. Primjer korištenja apstrakcije u C++ Builderu imamo kada se iza jednostavnog sučelja razvojnog okruženja skriva složena implementacija raznih vizualnih komponenata. Na primjer, komponente kao što su gumbi, polja za unos teksta ili dijalozi predstavljeni su kao objekti s kojima programer može raditi jednostavnim pozivanjem njihovih metoda ili postavljanjem svojstava. Iza tih objekata krije se kompleksna implementacija grafičkog sučelja i komunikacije s operativnim sustavom, ali programer se fokusira samo na ono što je bitno za aplikaciju.

Enkapsulaciju C++ Builder koristi kako bi povezao podatke i funkcije koje njima upravljaju unutar komponenti. Na primjer, svaka vizualna komponenta, poput gumba ili prozora, ima vlastita svojstva i metode koje su enkapsulirane unutar same komponente. Programer može postaviti i čitati svojstva poput veličine ili boje objekta, a da ne mora razumjeti detalje kako su ti elementi implementirani u pozadini.

Nasljeđivanje je posebno korisno u C++ Builderu jer omogućava proširivanje postojećih klasa komponenti bez potrebe za ponovnim pisanjem koda. Na primjer, programer može kreirati vlastite komponente nasljeđujući standardne komponente kao što su prozori, okviri ili kontrole, te dodavati nove funkcionalnosti specifične za aplikaciju. Ovaj mehanizam omogućava ponovnu uporabu koda i lako prilagođavanje komponenti novim zahtjevima.

Polimorfizam je isto tako ključan za rad s različitim komponentama u C++ Builderu. Kroz polimorfizam, objekti različitih klasa (npr. različite vrste vizualnih komponenti) mogu biti tretirani kao instance osnovne klase. To omogućava, na primjer, kreiranje generičkih funkcija koje mogu raditi s bilo kojom komponentom, bez obzira na to o kojoj se konkretnoj vrsti radi [9].

C++ Builder koristi snagu C++-a kako bi razvio interaktivne i događajima vođene aplikacije. U slijedećem ćemo poglavlju detaljnije opisati značajke ovog razvojnog okruženja i uporabu tih značajki kroz principe objektno orijentiranog programiranja.

3. Razvojno okruženje RAD Studio

RAD Studio C++ Builder je integrirano razvojno okruženje razvijeno od strane tvrtke Embarcadero Technologies, a namijenjeno je za brzi razvoj aplikacija koristeći C++ programski jezik. C++ Builder je posebno usmjeren na jednostavno stvaranje vizualno kvalitetnih i funkcionalnih Windows aplikacija s naglaskom na objektno-orijentirano programiranje.

3.1. Svojstva i značajke razvojnog okruženja

Jedna od bitnijih značajki C++ Buildera je Visual Component Library, koja pruža bogat skup vizualnih komponenata koje se lako mogu povlačiti i ispuštati na radnu površinu, čime se ubrzava proces izrade grafičkih korisničkih sučelja. Pri izradi aplikacija najčešće je dovoljno koristiti vizualne komponente koje u svojoj implementaciji sadrže sve funkcionalnosti potrebne za izvršavanje određenih tipova zadataka. Ova vizualna orijentiranost pomaže programerima da, umjesto trošenja vremena na pisanje velike količine programskog kôda, više vremena posvete izradi kvalitetnog sučelja, odnosno bržem razvoju aplikacije.

Pored VCL-a, C++ Builder također koristi *FireMonkey* [13], razvojni okvir koji omogućava stvaranje multiplatformskih aplikacija. FireMonkey koristi vektorsku grafiku i omogućava programerima da dijele kôd između Windowsa, macOS-a, iOS-a i Androida. Ova fleksibilnost čini C++ Builder prikladnim za razvoj aplikacija za različite operativne sustave. Ovaj razvojni okvir ima ugrađenu podršku za efekte (kao što su zamućenja, sjaj i drugi) i animaciju, što omogućuje jednostavnu izgradnju modernih fluidnih sučelja.

Budući da je FireMonkey kompatibilan s više platformi, isti izvorni kôd i dizajn formi može se koristiti za implementaciju na različite platforme. Izvorno podržava 32-bitne i 64-bitne izvršne datoteke na Windowsima, 32-bitne izvršne datoteke na macOS-u, 32-bitne i 64-bitne izvršne datoteke na iOS-u te 32-bitne i 64-bitne izvršne datoteke na Androidu. FireMonkey sadrži servise koji prilagođavaju korisničko sučelje ispravnom ponašanju i izgledu na svakoj ciljanoj platformi. Od uvođenja u C++ Builder XE2 verzije, razvojni okvir je doživio brojna poboljšanja u mnogim područjima, a i dalje se razvija i poboljšava. Na primjer, dodane su brojne komponente kao što su razni senzori i podrška za GPS, posebno korisno za one koji razvijaju mobilne aplikacije. Došlo je i do značajnih poboljšanja performansi i temeljnih tehničkih poboljšanja.

C++ Builder olakšava rad s bazama podataka kroz *FireDAC* [14] koji pruža jednostavan pristup različitim vrstama baza podataka, uključujući Oracle, Microsoft SQL Server, MySQL,

SQLite i mnoge druge. FireDAC također podržava različite načine povezivanja i omogućava povezivanje s lokalnim i udaljenim bazama podataka.

C++ Builder podržava LiveBindings, što omogućava povezivanje vizualnih komponenata izravno s podacima iz različitih izvora, čime se pojednostavljuje i ubrzava proces povezivanja podataka.

Uz ove značajke C++ Builder se integrira i sa *RAD Serverom* [15] koji omogućava izradu skalabilnih i sigurnih RESTful web servisa koristeći XML i JSON formate. Ovo je korisno za razvoj aplikacija koje trebaju komunicirati s udaljenim sustavima ili mobilnim uređajima. RAD Server također podržava autentifikaciju korisnika, push obavijesti, geolociranje i pohranu podataka.

3.2. Hijerarhija klasa i komponenata

C++ Builder je razvojni alat koji kombinira snagu C++ jezika s vizualnim razvojem aplikacija. Mnoge komponente i klase u ovom razvojnom okruženju su organizirane u hijerarhijske strukture koristeći principe objektno orijentiranog programiranja. Ovdje su neki primjeri hijerarhije klasa [16]:

TComponent (Osnovna klasa za sve komponente)

```
TComponent → TControl → TWinControl → TForm
TComponent → TControl → TGraphicControl → TLabel
```

TControl (Osnovna klasa za vizualne komponente)

```
TControl → TWinControl → TButton
TControl → TWinControl → TEdit
```

TForm (Osnovna klasa za prozore)

```
TForm → TCustomForm → TForm
```

TWinControl (Interaktivne kontrole s mogućnošću upravljanja unosom)

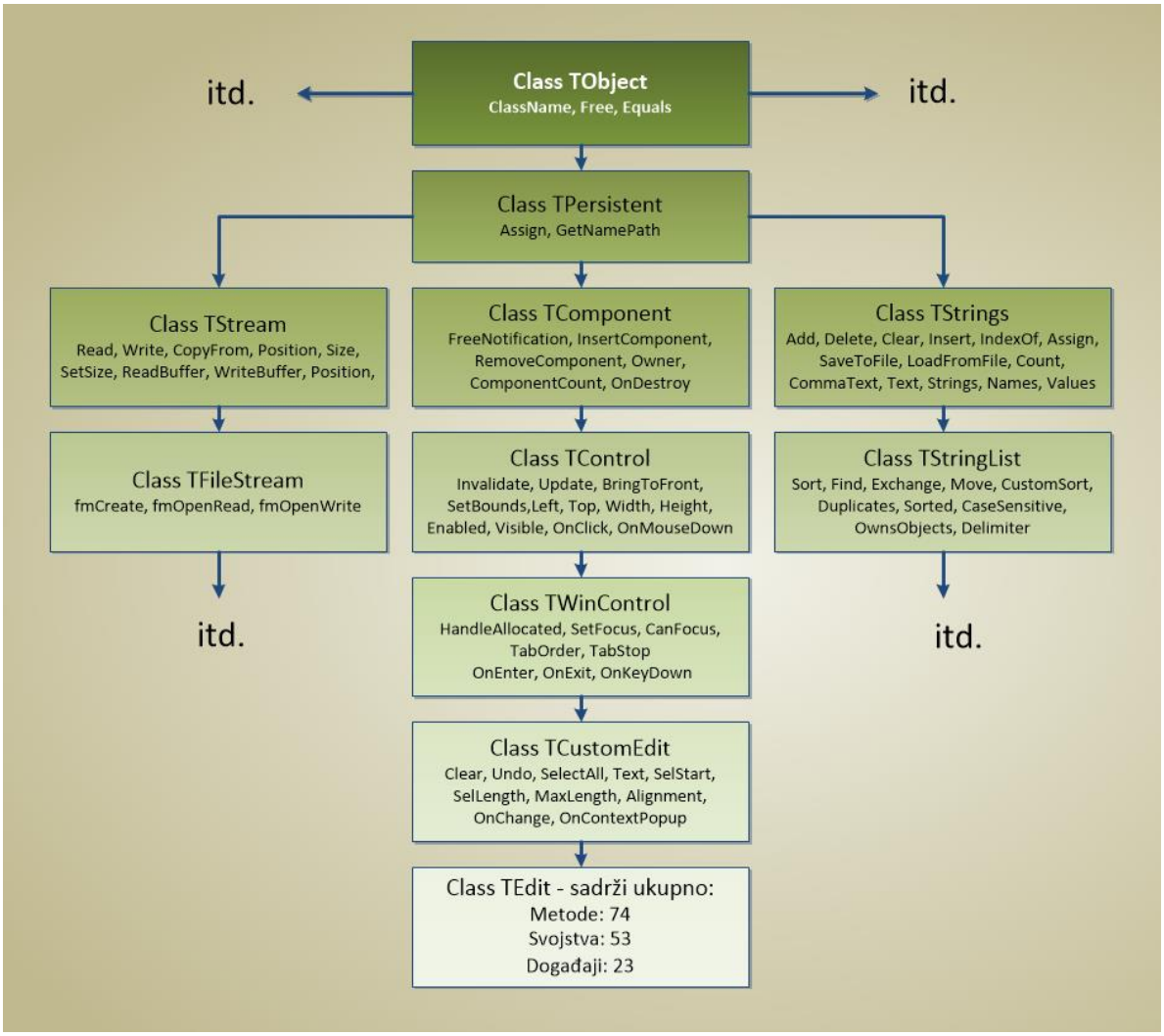
```
TWinControl → TEdit
TWinControl → TComboBox
```

TListControl (Osnovna klasa za kontrole koje prikazuju popise)

```
TListControl → TListBox
TListControl → TComboBox
```

Ono što ovdje možemo primijetiti da klase poput *TEdit* nasljeđuju svojstva i ponašanje iz drugih klasa koristeći višeslojnu hijerarhiju nasljeđivanja, tj. *TEdit* može nasljeđivati i od *TCustomEdit* (koji upravlja tekstualnim funkcijama) i od *TWinControl* (koji upravlja prozorskim kontrolama) zahvaljujući višeslojnom nasljeđivanju. *TEdit* je vizualna komponenta koja služi za

unos i uređivanje teksta i koja se koristi u gotovo svim aplikacijama koje zahtijevaju interakciju s korisnikom putem unosa tekstualnih podataka.



Slika 3.1 – Hijerarhijska struktura klasa u C++ Builderu

Slika 3.1 nam prikazuje stablo, odnosno kako izgleda hijerarhija nasljeđivanja u slučaju TEdit vizualne komponente, a koju ćemo ukratko i opisati:

- Sve klase u VCL-u nasljeđuju osnovnu klasu *TObjekt*. Ona pruža osnovne metode koje omogućuju funkcioniranje svih objekata, poput `ClassName()`, `Free()`, itd.
- Klasa *TPersistent* nasljeđuje `TObject` i uvodi mogućnost serijalizacije (spremanja i učitavanja) svojstava komponenti. To omogućava pohranu i vraćanje stanja objekata.
- *TComponent* je osnova za sve VCL komponente. Ona omogućuje komponentama da postoje unutar roditeljskog objekta, npr. forme, i podržava mehanizam vlasništva i streaming sistema u VCL-u.

- *TControl* je osnovna klasa za sve vizualne kontrole. Nasljeđuje `TComponent` i uvodi osnovne vizualne značajke, poput veličine, položaja, prikaza i rukovanja korisničkim unosom.
- *TWinControl* nasljeđuje osnovna svojstva iz TControl, ali dodaje podršku za rad s Windows prozorskim kontrolama. Time omogućava interakciju s korisnikom preko miša i tipkovnice.
- *TCustomEdit*, kao podklasa *TWinControl*, proširuje te osnovne vizualne kontrole dodavanjem funkcionalnosti za unos i uređivanje teksta.
- i na kraju *TEdit* nasljeđuje funkcionalnosti iz *TCustomEdit*, čime dobiva osnovu za upravljanje tekstom, ali dodaje dodatne metode i svojstva potrebne za stvaranje komponente koja se koristi u aplikacijama. Na kraju, klasa *TEdit* sadrži 74 metode, 53 svojstva i 23 događaja.

3.3. Primjena svojstava OOP u C++ Builderu

Nastavno na prikaz hijerarhije među klasama prikazat ćemo i osnovna svojstva objektno orijentiranog programiranja na stvarnim komponentama [17]:

Nasljeđivanje – koristi se za kreiranje novih komponenti temeljenih na postojećim klasama.

Primjer: *TButton* nasljeđuje iz klase *TControl*, što znači da nasljeđuje sve metode i svojstva vizualne komponente, kao što su `Position`, `Size`, ili `OnClick` događaji, ali dodaje i specifične funkcionalnosti gumba.

Polimorfizam - omogućava korištenje objekata različitih klasa kroz zajedničko sučelje. Npr. sve kontrole nasljeđuju iz *TControl*, pa ih možemo tretirati kao objekte te osnovne klase.

Primjer: Funkcija može prihvatiti argument tipa *TControl*, ali se može pozvati i s objektima poput *TButton, TLabel* ili *TEdit*, jer su sve te klase izvedene iz *TControl*.

Apstrakcija – postiže se skrivanjem detalja implementacije. U C++ Builderu, mnoge klase pružaju samo bitne informacije putem javnih sučelja, dok su detalji implementacije sakriveni.

Primjer: Kada koristimo komponentu *TEdit* za unos teksta, programer se bavi samo vanjskim svojstvima poput `Text`, `SetFocus`, `Clear`, dok su unutarnji mehanizmi za prikazivanje i obradu teksta apstraktirani.

Enkapsulacija – koristi se kako bi podaci i metode bili organizirani unutar klase i kako bi se ograničio pristup internim elementima klase.

Primjer: U komponenti *TButton*, svojstva kao što su `Caption` ili funkcije kao `OnClick` su javni i dostupni programeru, dok su detalji o tome kako se klik gumba obrađuje interno skriveni.

Ovi primjeri pokazuju kako C++ Builder koristi osnovne principe objektno orijentiranog programiranja za olakšavanje razvoja kompleksnih i dinamičnih aplikacija. Nasljeđivanje, polimorfizam, apstrakcija i enkapsulacija omogućavaju modularnost, ponovnu upotrebu koda i jednostavno održavanje, čineći rad s hijerarhijskim klasama intuitivnim i efikasnim.

Sve ove značajke prikazuju C++ Builder kao kvalitetan alat za izradu aplikacija. Međutim, na tržištu postoji još nekoliko alata sličnih njemu pa ćemo u nastavku opisati njegove prednosti i mane u odnosu na konkurentne alate.

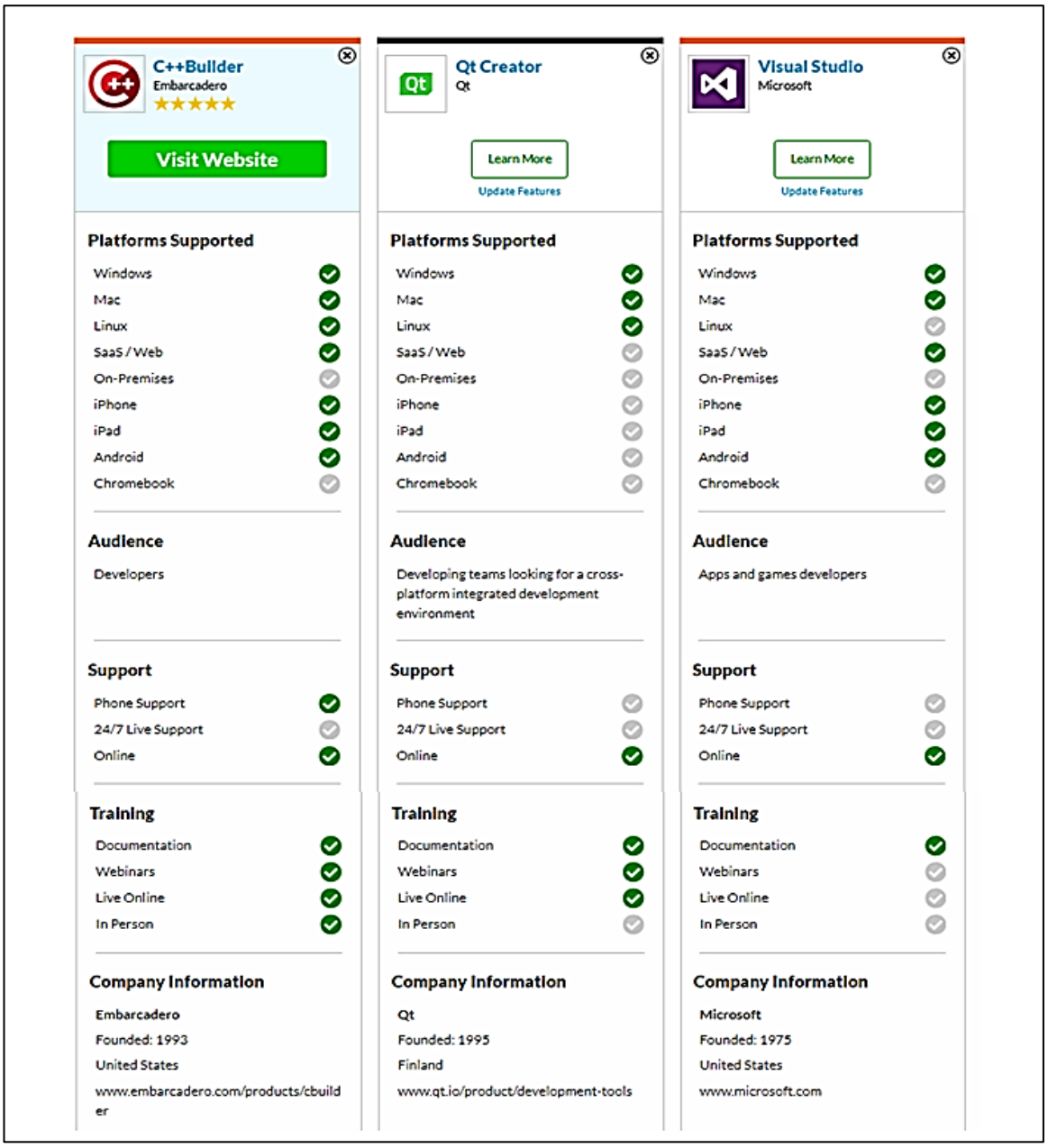
3.4. Konkurentna razvojna okruženja

C++ Builder predstavlja integrirano razvojno okruženje koje olakšava izradu Windows aplikacija koristeći C++ programski jezik. Međutim, osim C++ Buildera, postoje i drugi alati koji se mogu koristiti za razvoj Windows aplikacija.

Visual Studio - sa C++ podrškom pruža izvrsno integrirano razvojno okruženje za C++ programiranje. Ono uključuje bogat set alata za razvoj, debugiranje, dizajniranje korisničkih sučelja, te integraciju s raznim tehnologijama i servisima. Kao i kod C++ Buildera moguća je izrada *Windows Forms aplikacija* [18] pomoću vizualnih komponenti i ima podršku za drag-and-drop. Osim C++, podržava i druge jezike kao što su C#, VB.NET i druge. Besplatan je Community Edition, a postoji i u komercijalnim verzijama koje nude dodatne značajke.

Qt [19] - moćan alat za razvoj multiplatformskih aplikacija s bogatim grafičkim korisničkim sučeljima. Njegova glavna prednost je fleksibilnost i prenosivost aplikacija na različite operativne sustave uključujući Windows, macOS, Linux, pa čak i mobilne platforme poput Androida i iOS-a. Koristi *Qt Designer* za vizualno kreiranje sučelja, što omogućuje lako prototipiranje. Qt pruža bogat set biblioteka za rad s GUI-em, mrežnim komponentama, multimedijom i bazama podataka. Nudi open-source licencu, ali i postoje i komercijalne verzije.

Eclipse CDT (C/C++ Development Tooling) - popularno otvoreno razvojno okruženje koje se može proširiti dodacima. CDT je njegov modul za razvoj u programskom jeziku C i C++, pružajući alate za programiranje, debugiranje i analizu kôda. Eclipse može biti korišten s različitim dodacima za vizualno programiranje, kao što su SWT ili JavaFX Scene Builder [20]. SWT je toolkit koji omogućava razvoj grafičkih korisničkih sučelja za Java aplikacije. Razvijen je od strane Eclipse Foundation i koristi prilagođene GUI komponente, što znači da aplikacije izgledaju i ponašaju se kao aplikacije prilagođene baš tom operativnom sustavu.



Slika 3.2 – Usporedba C++ Buildera s konkurentnim alatima

(Izvor: https://sourceforge.net/software/compare)

Svaki od ovih alata ima svoje prednosti i nedostatke kao što možemo vidjeti na slici 3.2, pa je prilikom odabira razvojnog okruženja, važno uzeti u obzir specifične potrebe projekta, integraciju s određenim tehnologijama i platformama, te afinitete programera.

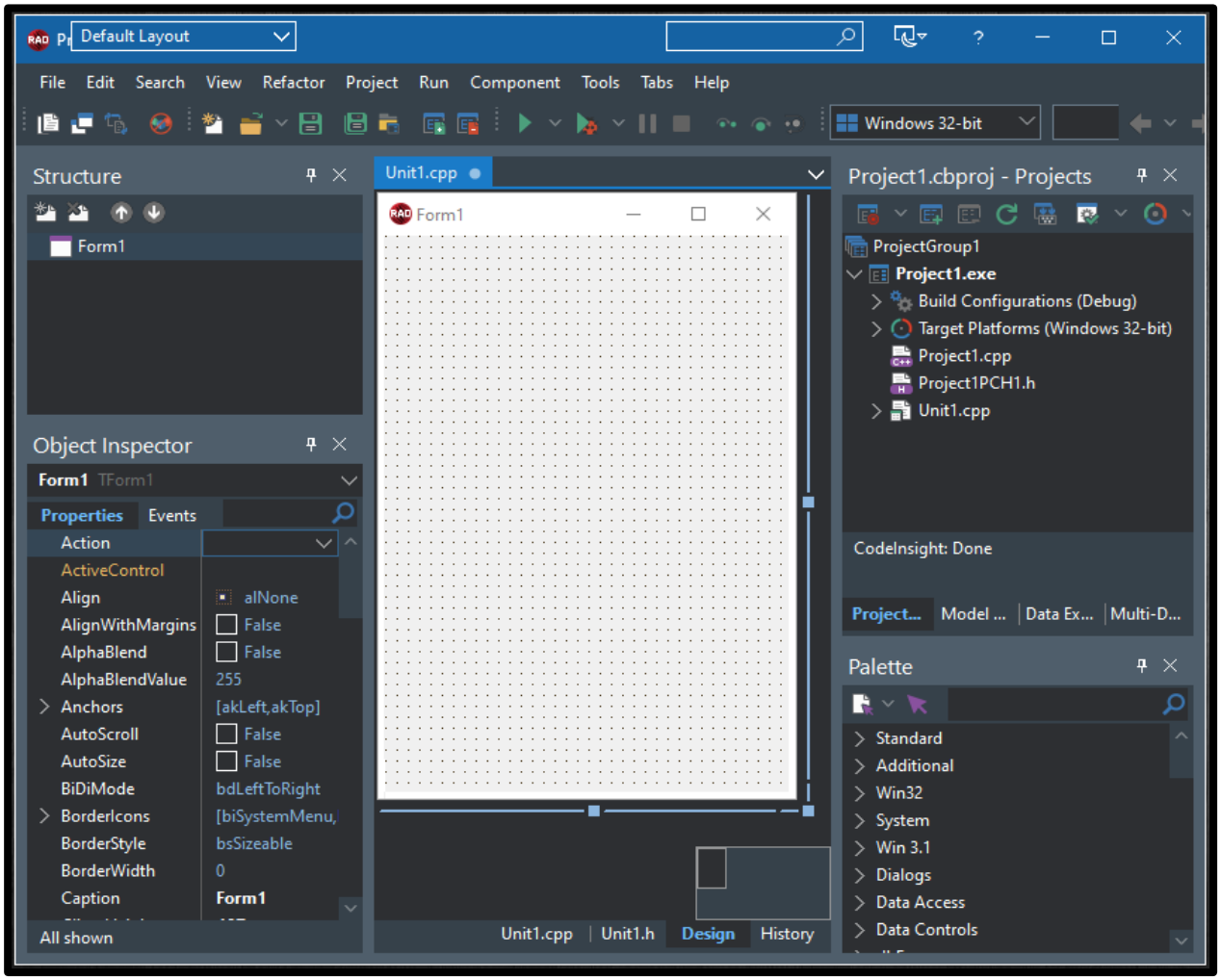
U sljedećem poglavlju ćemo opisati postupak izrade aplikacije u C++ Builderu. Opisat ćemo sučelje ovog razvojnog okruženja, način kreiranja dijaloga te način postavljanja komponenti. Prikazat ćemo i editor kôda u kojem ćemo pomoću C++ dodjeljivati funkcije nekim događajima, a sve to će biti popraćeno slikama radi lakšeg snalaženja.

4. Izrada projekta u C++ Builderu

U ovom poglavlju opisat ćemo način izrade VCL aplikacije u C++ Builderu [21]. Počet ćemo od osnovnog postavljanja radnog okruženja, a zatim ćemo proći kroz dizajniranje korisničkog sučelja. Nakon toga, istražit ćemo kako integrirati logiku u aplikaciju. Važno je napomenuti i upravljanje događajima u C++ Builderu te ćemo saznati kako povezati korisničke akcije s funkcijama u aplikaciji.

4.1. Sučelje C++ Buildera

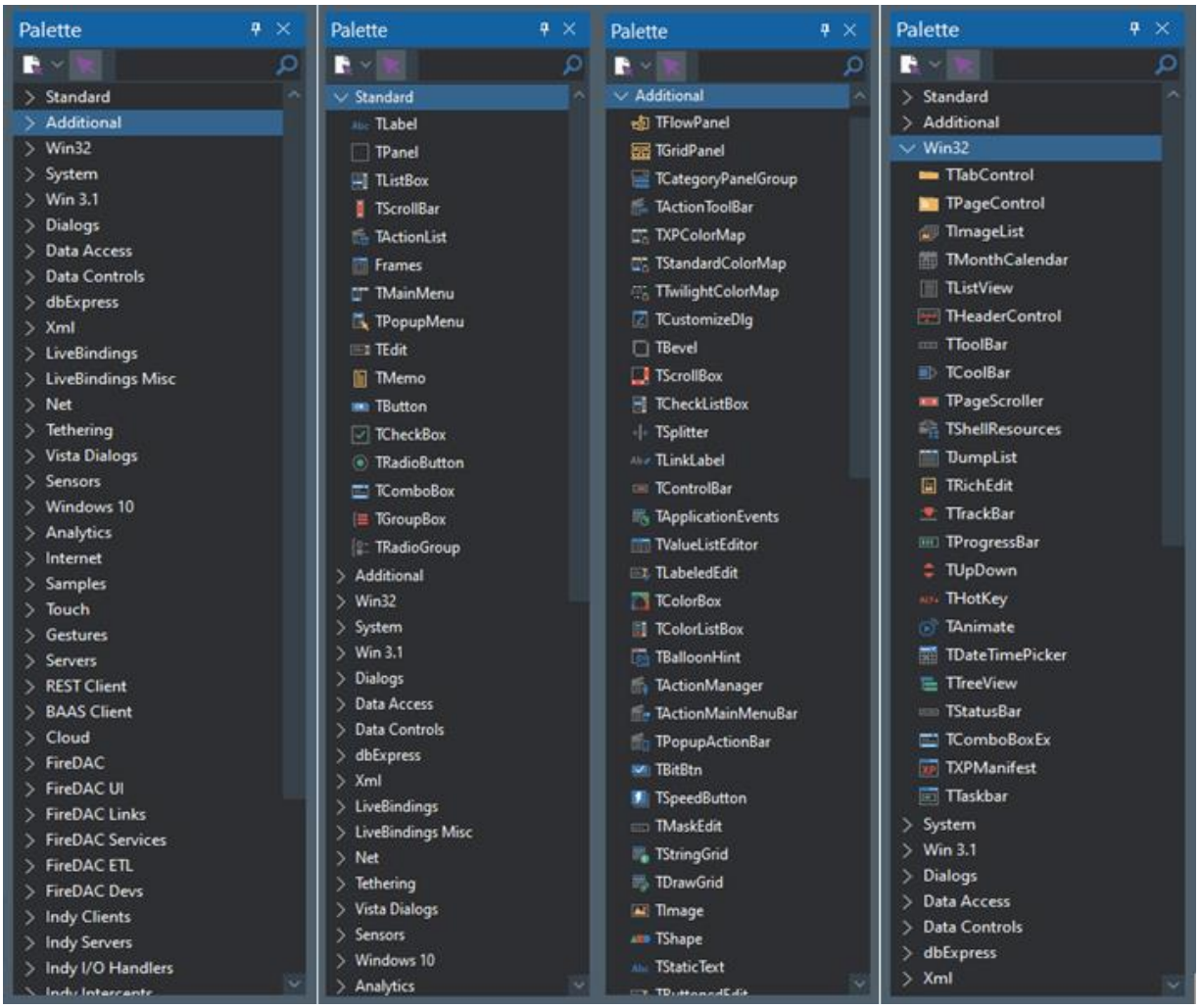
Na samom početku potrebno je kreirati novi projekt i iz izbornika izabrati *Windows VCL Application*. Kreiranjem nove Windows VCL aplikacije pokreće se razvojno okruženje i kreiraju se tri datoteke: datoteka projekta (Projekt1.cbproj), glavna datoteka izvornog kôda (Project1.cpp), datoteka zaglavlja (Project1.h) te datoteke dijaloga koji predstavlja glavno sučelje aplikacije.



Slika 4.1 – Razvojno okruženje C++ Buildera

Za svaki novi dodani dijalog također se kreiraju te datoteke od kojih datoteka zaglavlja s ekstenzijom *.h* sadrži definicije zaglavlja te prototipe metoda, datoteka s ekstenzijom *.cpp* sadrži programsku implementaciju dok je datoteka s ekstenzijom *.dfm* opisna datoteka i sadrži opise svih komponenti tog dijaloga.

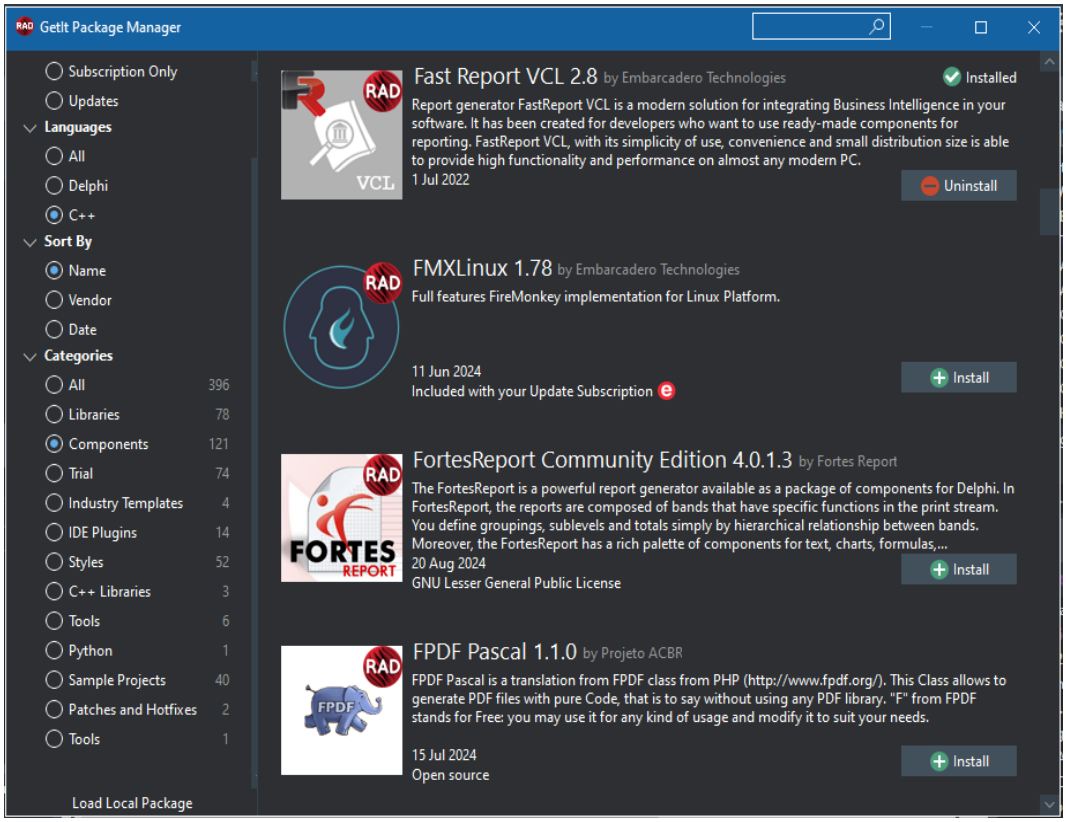
Uz standardne alatne trake unutar sučelja tu se još nalazi i nekoliko dijaloga razvojnog okruženja (Slika 4.1). S lijeve strane u gornjem dijelu je dijalog *Structure* koji će nam prikazivati sve komponente koji se nalaze na dijalogu naše aplikacije (*Form1*). Ispod njega se, nakon odabira komponente, prikazuju svojstva i događaji odabrane komponente koje možemo prilagoditi, odnosno dodati prema potrebama aplikacije. S desne strane se nalazi dijalog s prikazom stabla projekta, te dijalog *Palette* u kojem su ponuđene sve vizualne i nevizualne komponente koje možemo koristiti pri izradi aplikacije.



Slika 4.2 - Dio komponenti koje sadrži VCL biblioteka

Iako slika 4.2 prikazuje samo dio, vidi se da se radi o velikom broju, točnije nekoliko stotina komponenti. Komponente su podijeljene po grupama radi lakšeg pronalaženja, a postoji i mogućnost pretrage pomoću tražilice. Osim komponenti koje RAD Studio inicijalno sadrži,

postoji i mogućnost preuzimanja i instalacije dodatnih paketa komponenti putem repozitorija (Slika 4.3).



Slika 4.3 – Repozitorij s dodatnim paketima

Sami dijalog glavnog sučelja nove aplikacije nalazi se u centralnom dijelu i prilagodljiv je po veličini ovisno o potrebama. Sadrži i mrežu pomoću koje se lakše pozicioniraju i poravnavaju komponente. Postavljanje komponente vrši se jednostavno povlačenjem odabrane komponente sa palete na mjesto koje nam odgovara unutar dijaloga aplikacije koju izrađujemo.

Svaka komponenta na dijalogu, u kôdu je predstavljena pokazivačem odgovarajućeg tipa (*TGroupBox**, *TRadioButton**, *TButton** itd.), a ime pokazivača je zapravo vrijednost svojstva *Name* pojedine komponente. Metode klase dijaloga najčešće predstavljaju dijelove kôda koji se trebaju izvršiti kao posljedica određenog događaja (npr. *Button1Click*), a po potrebi moguće je dodati i vlastite metode.

Kada se korištenjem dizajnera u dijalog dodaju nove komponente, sadržaj datoteke zaglavlja automatski se nadopunjuje odgovarajućim deklaracijama pokazivača. Slično vrijedi i prilikom praćenja novog događaja, a tada se prototip metode za obradu tog događaja automatski dodaje u datoteku zaglavlja.

4.2. Opis komponenti i pisanje koda

U nastavku ćemo navesti nekoliko komponenti koje se najčešće koriste i opisat ćemo njihova svojstva i primjenu [21]:

TLabel – na dijalogu aplikacije omogućuje prikazivanje teksta korisnicima. Tekst može biti statičan ili dinamički postavljen putem kôda. Iako *TLabel* komponenta sama po sebi ne podržava interakciju korisnika, može se koristiti kao dio kompleksnijih sučelja u kombinaciji s drugim komponentama koje omogućuju interaktivnost, poput *TButton* ili *TEdit* komponenti.

TEdit - omogućuje korisnicima unos teksta pomoću tipkovnice. Tekst se može unositi ručno ili se može postaviti putem programskog kôda. Moguće je postaviti različite parametre ograničenja za unos teksta, kao što su maksimalna duljina teksta, tipovi dopuštenih znakova ili format unosa. Koristeći događaje i metode, mogu se implementirati mehanizmi za provjeru ispravnosti unesenih podataka, pružajući korisnicima povratne informacije o ispravnosti njihovog unosa. *TEdit* podržava događaje kao što su *OnChange, OnEnter, OnExit* i sl. omogućujući praćenje i reakciju na promjene u sadržaju polja.

TButton - omogućuje definiranje akcija koje se pokreću kada korisnik pritisne gumb. To može uključivati pokretanje određene funkcije, navigaciju na drugi dijalog ili izvršavanje neke druge operacije. Podržava različite događaje poput *OnClick, OnMouseDown, OnMouseUp*, što omogućuje definiranje ponašanja gumba.

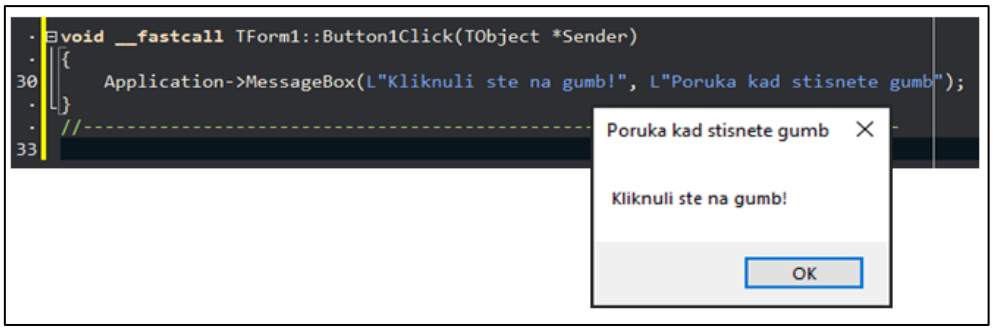
TDateTimePicker - omogućuje korisnicima odabir datuma, vremena ili i datuma i vremena zajedno, ovisno o postavkama komponente. Pruža vizualni prikaz kalendara unutar korisničkog sučelja, olakšavajući korisnicima pregled i odabir željenog datuma ili vremena. Moguće je postaviti različite parametre ograničenja za odabir datuma i vremena, kao što su minimalni i maksimalni dopušteni datum ili vremenski interval. *TDateTimePicker* podržava različite događaje kao što su *OnChange, OnCloseUp, OnDropDown* itd.

Osim navedenih komponenti često se tu još koriste:

- *TListBox* / *TComboBox* - prikazivanje liste opcija koje korisnici mogu odabrati.
- *TPanel* / *TGroupBox* - organizacija i grupiranje drugih komponenti unutar forme.
- *TCheckBox* / *TRadioButton* - omogućavanje odabira između više opcija.
- *TMainMenu* / *TPopupMenu* - stvaranje glavnog izbornika ili kontekstnog izbornika s opcijama koje korisnici mogu odabrati.
- *TCheckBoxList* - prikazivanje liste opcija u obliku više potvrdnih okvira, gdje korisnici mogu odabrati jednu ili više opcija.
- *TTreeView* - prikazivanje hijerarhijskih podataka u obliku stabla, što je korisno za organizaciju i prikazivanje struktura podataka poput direktorija i datoteka.

- *TListView* - prikazivanje podataka u obliku liste s različitim prikazima, uključujući prikaz ikona, detalja ili podataka s jednostavnim tekstom.
- *TImageList* - upravljanje skupom slika koje se koriste unutar aplikacije, olakšavajući njihovo učitavanje i prikazivanje unutar različitih komponenti poput *TListView*-a ili *TTreeView*-a.
- *TProgressBar* - prikazivanje napretka ili statusa izvođenja određenih zadataka unutar aplikacije.
- *TChart* / *TDBChart* - stvaranje različitih vrsta grafova i dijagrama kako bi se vizualizirali podaci unutar aplikacije.
- *TTabControl* / *TPageControl* - organizacija više skupina komponenti unutar jednog prozora pomoću kartica, olakšavajući korisnicima navigaciju kroz različite dijelove aplikacije.
- *TScrollBar* - dodavanje klizača koji omogućuje korisnicima navigaciju kroz velike skupove podataka, poput dugih tekstova ili sl.
- *TBitBtn* - dodavanje gumba s ikonom koji obično koristi manje prostora u odnosu na standardni TButton, što ga čini pogodnim za kompaktna korisnička sučelja.
- *TMaskEdit* - unos podataka s određenim formatom, poput brojeva telefona ili poštanskih brojeva, omogućujući programerima definiranje i provjeru formata unosa [18].

Pisanje kôda [21], odnosno dodavanje funkcionalnosti korištenim komponentama vrši se na način da se prvo, nakon postavljanja komponente na formu aplikacije, iz izbornika *Object Inspector/Events* odabere događaj. Dvostrukim klikom na prazno polje pokraj imena tog događaja automatski se kreira implementacija metode koja će se izvršavati kada se događaj desi, te se kôd otvara u uređivaču. Npr., za komponentu *Button1* i odabrani događaj *OnClick*, implementacija metode će se otvoriti u uređivaču, a kôd koji ćemo upisati unutar te metode pokrenut će se svaki puta kada kliknemo na taj gumb. Na primjeru sa slike 4.4 vidi se kôd koji će se pokrenuti klikom na gumb, a kao posljedica izvršenja kôda pojavit će se *MessageBox* s navedenom porukom.



Slika 4.4 - Implementacija metode za odabrani događaj

RAD Studio unutar svog uređivača kôda sadrži i *IntelliSense* koji nudi dovršavanje pisanja kôda te prikazuje dostupne klase, metode, parametre i sl. Naravno, uređivač kôda moguće je dograditi dostupnim ažuriranjima iz prije spomenutog repozitorija (Slika 4.3).

U bilo kojem trenutku izrade aplikacije, aplikaciju je moguće pokrenuti klikom na gumb *Run* ili odabirom stavke iz glavnog izbornika čime će C++ Builder automatski izvršiti kompilaciju kôda i pokrenuti aplikaciju ukoliko nije pronađena niti jedna pogreška.

Aplikaciju je moguće pokrenuti i u načinu rada za otkrivanje grešaka, te je moguće aplikaciju izvršavati korak po korak ukoliko su prije pokretanja definirane prekidne točke unutar kôda aplikacije. Time je lakše pronaći razloge, odnosno greške u kôdu, zbog kojih dolazi do nepravilnog rada ili rušenja aplikacije.

U direktoriju projekta nakon pokretanja aplikacije kreirat će se izvršna *.exe* datoteka. Međutim, takva datoteka će moći ispravno pokrenuti aplikaciju samo na računalu na kojem je instaliran RAD Studio jer prema inicijalnim postavkama u *.exe* datoteku neće biti uključene biblioteke i vanjski paketi komponenti. Da bi se aplikacija mogla pokrenuti i uspješno izvršavati na drugim računalima potrebno je u postavkama projekta isključiti dinamičko povezivanje s bibliotekama (*„Use dynamic RTL“* i *„Build with runtime packages*“). Time će sve te potrebne biblioteke i vanjski paketi biti uključeni u *.exe* datoteku.

Kako bismo se bolje upoznali s mogućnostima i načinom izrade VCL aplikacije u C++ Builderu, u slijedećem poglavlju ćemo opisati praktični dio izrade aplikacije čija će namjena biti pomoć u organizaciji i vođenju osobnih financija. Cilj je izraditi aplikaciju koja će korisnicima olakšati raspolaganje vlastitim financijama, praćenje prihoda i izdataka i to sve pomoću jednostavnog i učinkovitog sučelja aplikacije na Windows platformi.

5. Praktični rad

Veliku pažnju pri planiranju i izradi aplikacija potrebno je posvetiti korisničkom iskustvu. Korisničko iskustvo [22] u razvoju desktop aplikacija s grafičkim sučeljem obuhvaća sve aspekte interakcije korisnika s aplikacijom, a ključno je za osiguranje učinkovitosti i zadovoljstva korisnika. Dizajn sučelja trebao bi biti intuitivan i jednostavan, omogućavajući korisnicima da lako razumiju kako koristiti aplikaciju bez dugog razmišljanja. Naglašava se važnost provođenja testiranja korisnika, što pomaže dizajnerima da identificiraju i isprave probleme u sučelju prije nego što aplikacija dospije do krajnjih korisnika.

Osim toga, važnost estetskog dizajna [23] ne smije se zanemariti. Dobro dizajnirani proizvodi ne samo da izgledaju dobro, već i funkcioniraju na način koji korisnicima omogućava lako razumijevanje i korištenje. Performanse i stabilnost aplikacije značajno utječu na korisničko iskustvo. Spore ili nestabilne aplikacije mogu frustrirati korisnike i potaknuti ih da odustanu od njihovog korištenja.

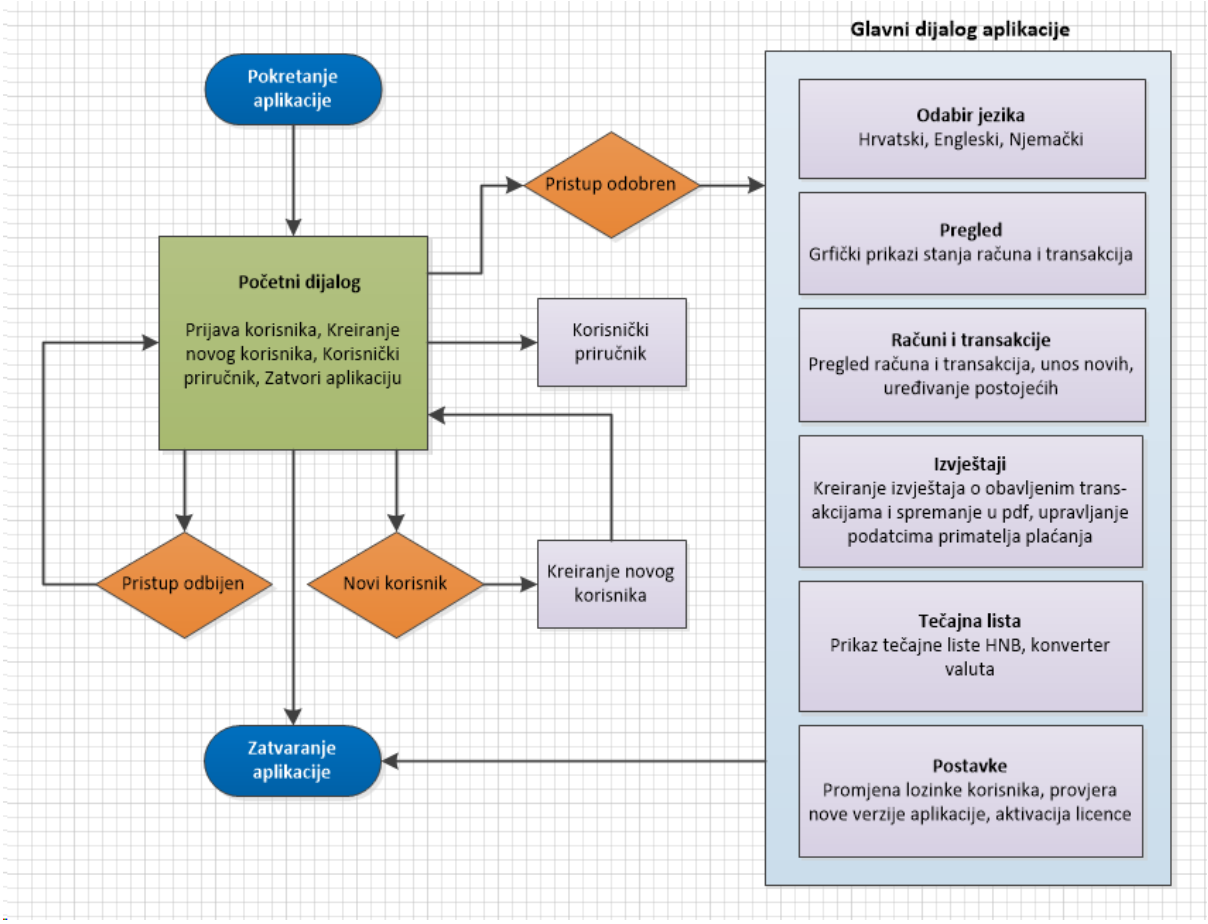
Kontinuirano prikupljanje povratnih informacija i iteracija ključni su za poboljšanje UX-a. Razumijevanje stvarnih korisničkih potreba i njihovih zahtjeva značajno se može poboljšati konačni proizvod [24].

5.1. Planiranje i priprema

Uzimajući u obzir sve navedeno, pokušao sam izraditi aplikaciju koja bi zadovoljila barem nekakve minimalne uvijete svojom namjenom, funkcionalnostima i izgledom grafičkog korisničkog sučelja. U moru aplikacija koje se danas pojavljuju teško je bilo odabrati nešto što već nije viđeno, ali na kraju sam odabrao aplikaciju za upravljanje osobnim financijama.

Na početku je bilo potrebno odabrati koje će sve funkcionalnosti imati aplikacija i tu je svakako trebalo izraditi nekoliko dijaloga koji će odvojiti pojedine funkcionalnosti. Osnovna funkcionalnost je tablično prikazivanje podataka, mogućnost kreiranja novih, uređivanja i brisanja postojećih, njihovo filtriranje i sortiranje. Trebala bi postojati mogućnost prikaza tih podataka na izvještajima, te ispis tih izvještaja na pisaču ili mogućnost spremanja u nekom digitalnom obliku, npr. u PDF obliku. Pristup aplikaciji bi trebao biti ograničen samo registriranim korisnicima, te bi morala biti mogućnost prikaza određenih podataka samo određenim korisnicima, npr. da svaki korisnik može vidjeti stanja samo svojih računa. Aplikacija bi trebala biti višejezična, sadržavati uputu za korištenje i mogućnost zapisivanja logova.

Nakon izrade dijagrama tijeka aplikacije (slika 5.1), mogao sam kreirati bazu podataka i započeti sa samom izradom u razvojnom okruženju.



Slika 5.1 - Dijagram tijeka aplikacije

5.2. Baza podataka

Za rad aplikacije pored korisničkog sučelja potrebna je i baza podataka. Baze podataka koristimo radi perzistencije što nam omogućava da podaci ostanu sačuvani i dostupni čak i nakon što se aplikacija zatvori ili sustav ponovno pokrene. U aplikacijama koje koriste baze podataka, perzistencija podataka oslanja se na sloj za pohranu podataka, poznat kao sloj perzistencije [25][26] (eng. persistence layer), koji omogućava spremanje, dohvaćanje i upravljanje podacima. Njegova je uloga osigurati učinkovitu interakciju s bazom podataka, omogućujući aplikaciji spremanje i dohvaćanje podataka na standardizirani i optimizirani način. Dobro dizajniran sloj perzistencije povećava čitljivost, održavanje i performanse aplikacije. Tehnike poput indeksiranja i optimizacije baze podataka mogu znatno poboljšati učinkovitost aplikacije. Osim u bazama, podatci se mogu pohranjivati i u različitim formatima datoteka, npr. CSV, JSON, XML i sl..

Još jedna bitna stvar kod izrade aplikacija s bazom podataka je i mapiranje tipova podataka. Mapiranje tipova podataka odnosi se na proces povezivanja ili preslikavanja jednog tipa podataka u drugi, obično radi kompatibilnosti, uštede memorije ili pojednostavljenja logike

unutar programa. Mapiranje omogućuje osiguranje da se prilikom pretvorbe podataka ili prenošenja podataka između sustava različitih tipova podataka neće izgubiti ili oštetiti informacije.

U tu svrhu koriste se alati za objektno relacijsko mapiranje (ili eng Object-Relational Mapping – ORM) koji omogućuju programerima rad s bazama podataka koristeći objektno-orijentirani pristup. To znači da se programeri mogu fokusirati na rad s objektima, a ne na pisanje izravnih SQL upita, čime se pojednostavljuje razvoj i povećava produktivnost. U C++ Builderu, iako nema izvorne ORM biblioteke s punom podrškom kao u nekim drugim jezicima, postoje opcije za rad s relacijskim bazama podataka kroz biblioteke poput FireDAC koje omogućuju mapiranje i rad s podacima na visokoj razini.

Uzimajući to sve u obzir, pri izradi ove aplikacije koristio sam SQLite bazu podataka [27] u obliku *.sdb* datoteke spremljene u direktoriju same aplikacije. SQLite sustav za upravljanje bazom podataka je biblioteka koja je samostojeća, neovisna o drugim komponentama, biblioteka kojoj nije potreban server, niti ikakve dodatne konfiguracije. Sami kôd SQLite-a se nalazi na javno dostupnoj domeni, tj. izvorni kôd je dostupan svima i to besplatno u bilo koje svrhe. SQLite biblioteka je male veličine, ali može upravljati podacima sve do 281 TB što je i više nego dovoljno za bilo koju aplikaciju.

S obzirom da C++ Builder za rad s bazama podataka koristi FireDAC dodatak, uz vrlo male izmjene unutar koda aplikacije, moguće je bazu postaviti i na server kako bi bila dostupna na više računala. Kako smo i prije naveli, FireDAC dodatak pruža jednostavan pristup različitim vrstama baza podataka, uključujući Oracle, Microsoft SQL Server, MySQL, SQLite i mnoge druge.

U svrhu testiranja, kreirao sam i bazu na localhost poslužitelju uz pomoć programskog paketa XAMPP [28]. To je besplatni softverski paket koji omogućuje jednostavno postavljanje i upravljanje lokalnim razvojnim okruženjem za aplikacije. XAMPP je izuzetno koristan za web developere jer omogućuje pokretanje i testiranje web aplikacija lokalno na računalu prije nego što se one postave na produkcijski poslužitelj. Pruža jednostavno sučelje za upravljanje servisima (npr. Apache i MySQL) i omogućuje brz razvoj bez potrebe za kompleksnim postavljanjem okruženja.

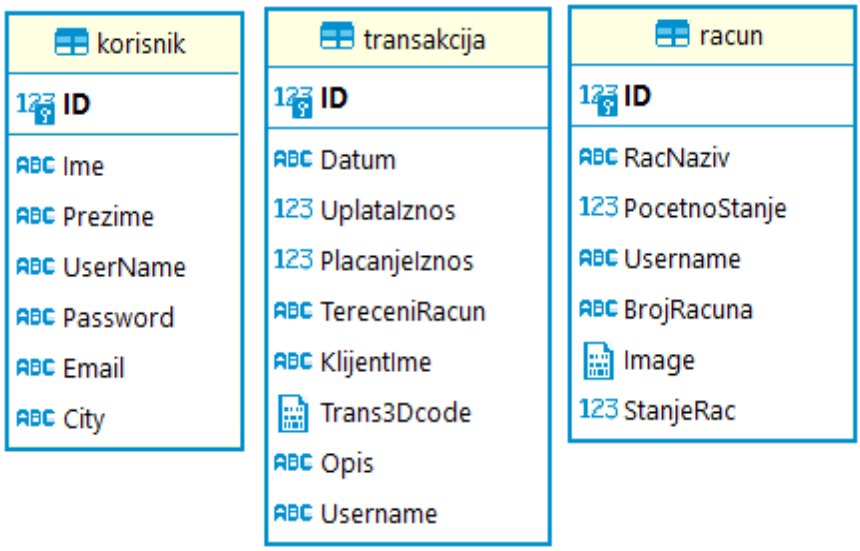
Za kreiranje i upravljanje bazom podataka za vrijeme razvoja aplikacije korištena je aplikacija DBeaver Community Edition [29] koja se koristi kao SQL klijent i alat za administraciju baza. Aplikacija ima uređivač kôda koji podržava automatsko dovršavanje kôda i isticanje sintakse. Također nudi dodatke koji omogućuju korisnicima prilagodbu velikog dijela funkcionalnosti specifičnih za pojedine vrste baza podataka. Community izdanje (CE) DBeavera

je besplatno i otvorenog kôda koji se distribuira pod Apache licencom. Zatvoreno Enterprise izdanje DBeavera distribuira se pod komercijalnom licencom.

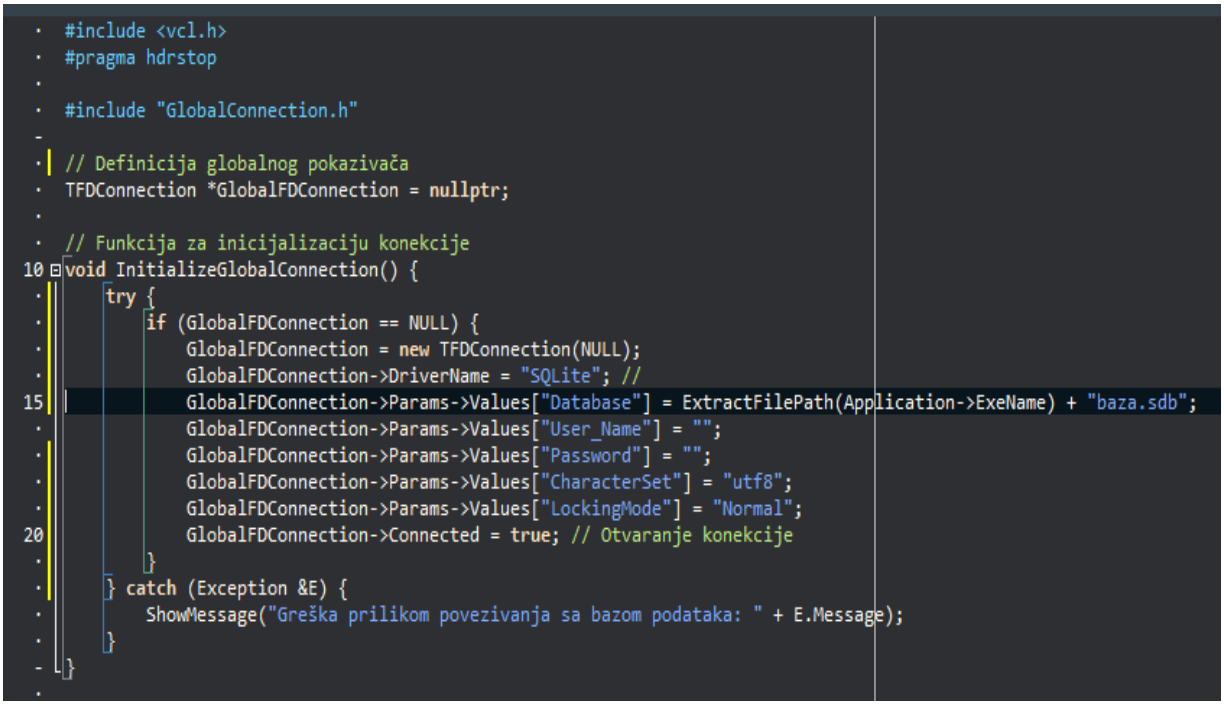
Kreirana baza podataka za potrebe ove aplikacije sadrži 3 tablice:

- *korisnik* – sadrži podatke o korisnicima aplikacije koji imaju pravo pristupa (ime, prezime, email, grad, korisničko ime i lozinka)
- *transakcija* – sadrži podatke o svakoj učinjenoj transakciji, odnosno o uplatama i isplatama sa računa (datum, iznos uplate, iznos isplate, terećeni račun, naziv primatelja kojem su isplaćena sredstva, opis transakcije itd.)
- *racun* – sadrži podatke o nazivu i stanju svih tekućih i žiroračuna korisnika, te podatke o njihovim karticama

Na slici 5.2 prikazan je dijagram entiteta i odnosa (ili eng. Entity-Relationship Diagram – ERD) baze na kojem se vide polja koja sadrže tablice i tipovi varijabli koje se koriste u tablicama.



Slika 5.2 - Prikaz baze podataka



```
#include <vcl.h>
#pragma hdrstop

#include "GlobalConnection.h"

// Definicija globalnog pokazivača
TFDConnection *GlobalFDConnection = nullptr;

// Funkcija za inicijalizaciju konekcije
void InitializeGlobalConnection() {
    try {
        if (GlobalFDConnection == NULL) {
            GlobalFDConnection = new TFDConnection(NULL);
            GlobalFDConnection->DriverName = "SQLite"; //
            GlobalFDConnection->Params->Values["Database"] = ExtractFilePath(Application->ExeName) + "baza.sdb";
            GlobalFDConnection->Params->Values["User_Name"] = "";
            GlobalFDConnection->Params->Values["Password"] = "";
            GlobalFDConnection->Params->Values["CharacterSet"] = "utf8";
            GlobalFDConnection->Params->Values["LockingMode"] = "Normal";
            GlobalFDConnection->Connected = true; // Otvaranje konekcije
        }
    } catch (Exception &E) {
        ShowMessage("Greška prilikom povezivanja sa bazom podataka: " + E.Message);
    }
}
```

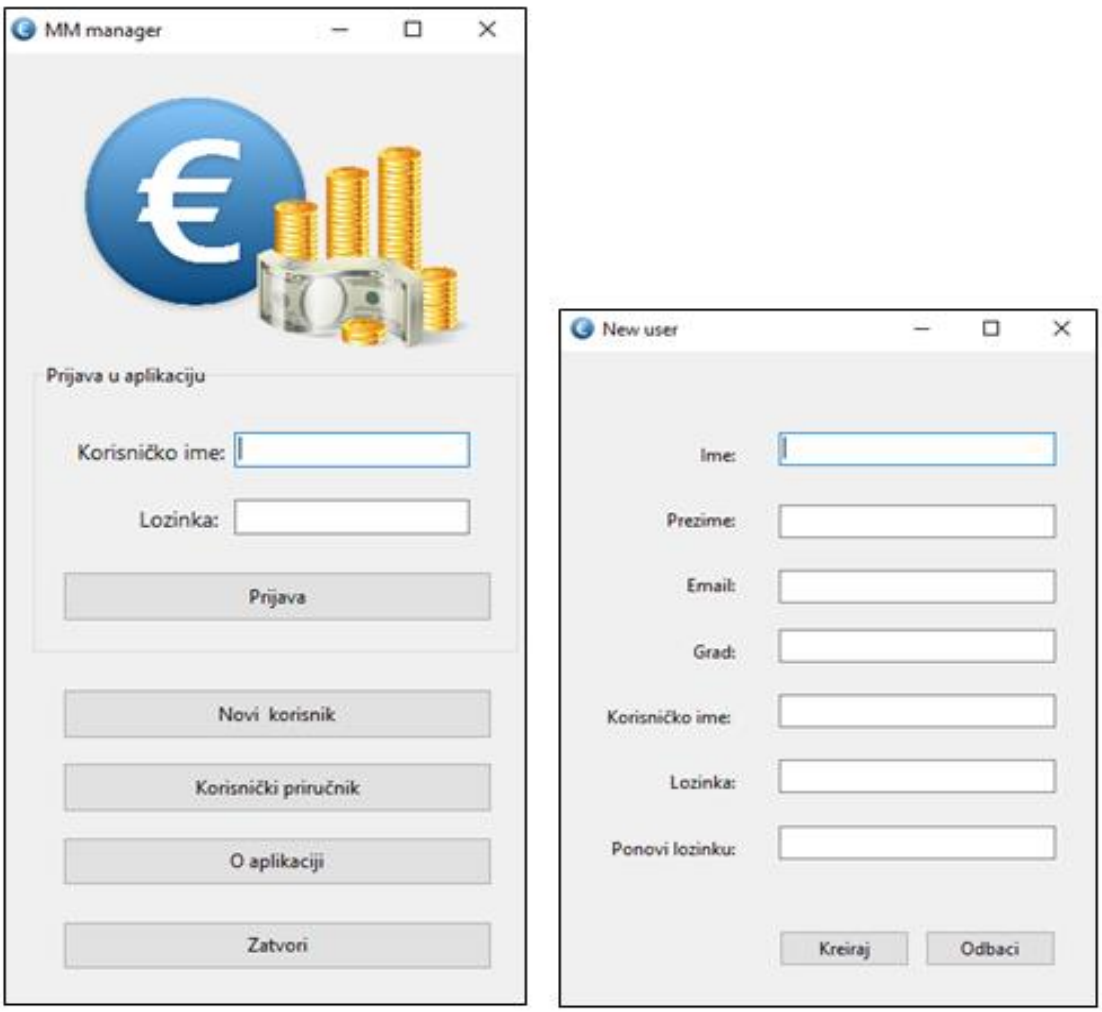
Slika 5.3 - Programski kôd funkcije za spajanje na bazu

Povezivanje aplikacije s bazom vrši se pomoću nekoliko parametara, a sami izgled kôda prikazan je na slici 5.3. Ovaj kôd implementira funkciju *InitializeGlobalConnection* koja služi za inicijalizaciju globalne konekcije prema bazi podataka. Funkcija koristi *try-catch* blok za rukovanje potencijalnim iznimkama (greškama) koje mogu nastati tijekom povezivanja. Funkcija najprije provjerava je li pokazivač *GlobalFDConnection* jednak *NULL*. Ako jest, to znači da konekcija još nije inicijalizirana. Zatim kreira novi objekt *TFDConnection*, koji je klasa za rad s FireDAC konekcijama u C++ Builderu. Nakon što su svi parametri postavljeni, konekcija se aktivira postavljanjem svojstva *Connected* na *true*. Ako dođe do greške prilikom inicijalizacije ili otvaranja konekcije, iznimka se hvata unutar *catch* bloka. U tom slučaju, prikazuje se poruka s opisom greške koristeći *ShowMessage* funkciju, koja kombinira tekst "*Greška prilikom povezivanja sa bazom podataka:"* s opisom greške. Inicijalizacija spoja na bazu vrši se odmah nakon pokretanja aplikacije, a prije prikaza početnog dijaloga kako bi baza bila spremna za sve daljnje upite koje će aplikacija vršiti za vrijeme rada.

5.3. Izrada dijaloga aplikacije

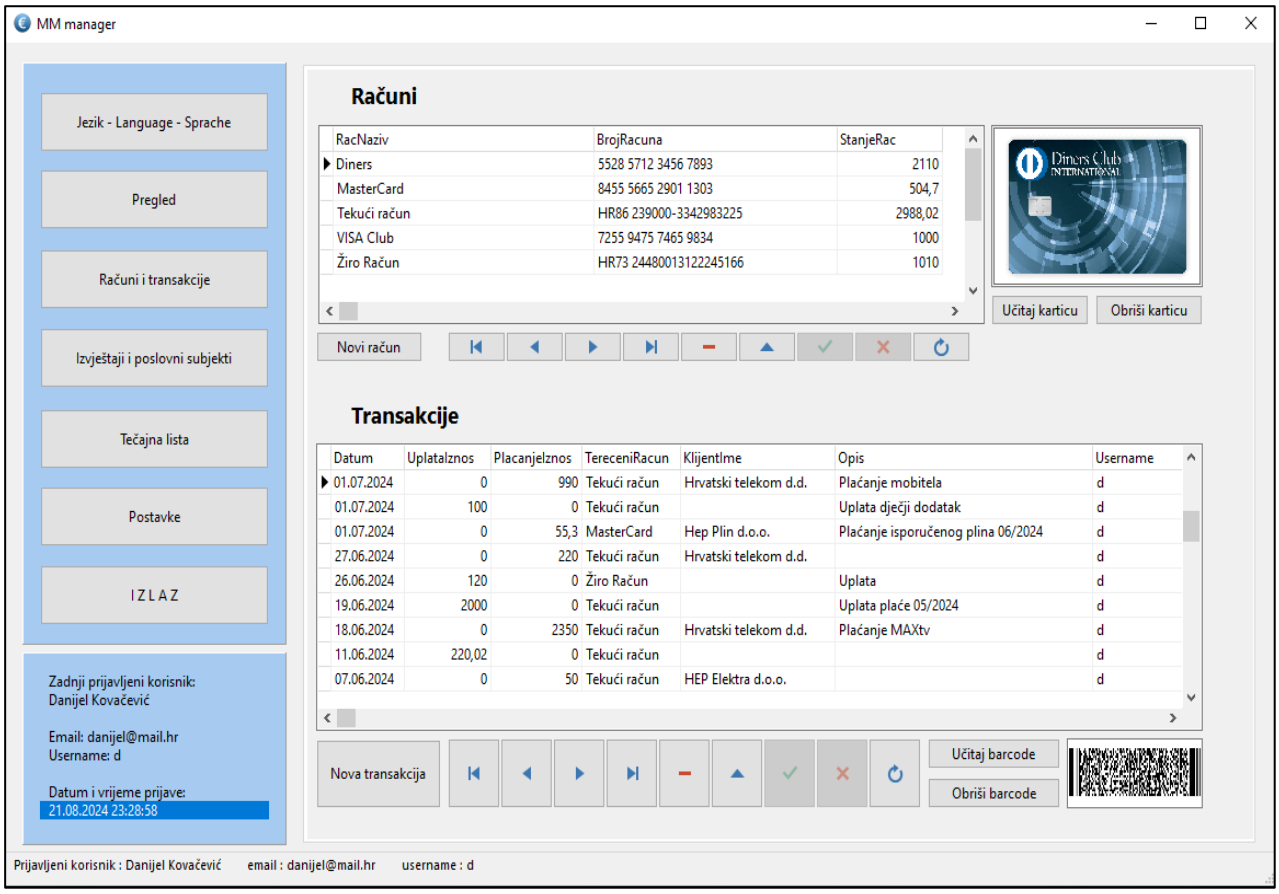
Nakon definiranja i kreiranja baze podataka pristupio sam izradi dijaloga aplikacije. Ideja je da se pokretanjem aplikacije pojavi početni dijalog gdje je za prijavu u aplikaciju potrebno unijeti korisničko ime i lozinku. Pritiskom na gumb „Prijava“ vrši se provjera korisnika u bazi podataka, a ako korisnik ne postoji, o tome se obavještava porukom. Prilikom prvog korištenja

aplikacije potrebno je kreirati novog korisnika, odnosno upisati ga u bazu. To se vrši odabirom gumba *Novi korisnik* čime se otvara dijalog za unos podataka o tom korisniku (slika 5.4).



Slika 5.4 – Početni dijalog aplikacije i dijalog za unos podataka o novom korisniku

Nakon kreiranja novog korisnika i uspješne prijave u aplikaciju korisničko ime prijavljenog korisnika spremamo u globalnu varijablu *_prijavljeniKorisnik* koju ćemo do zatvaranja aplikacije koristiti za personalizirani prikaz podataka, odnosno svaki korisnik će moći vidjeti samo svoje račune i njihovo stanje, te u postavkama moći promijeniti samo svoju lozinku. Aplikacija je zamišljena kao višekorisnička u smislu da će pojedinačni korisnici vidjeti samo svoje račune dok će svi korisnici moći vidjeti sve obavljene transakcije. Npr. svi članovi jednog kućanstva će koristit aplikaciju i svaki će vidjeti samo svoje račune, dok će imati mogućnost vidjeti sve transakcije jer plaćaju zajedničke račune, odnosno zajednički dijele troškove kućanstva.



Slika 5.5 – Glavni dijalog aplikacije s tabličnim prikazom računa i transakcija

Slika 5.5 nam prikazuje glavni dijalog aplikacije na kojem su tablični prikazani podatci o svim računima prijavljenog korisnika i sve transakcije koje su do sada učinjene kroz aplikaciju. Za tablični prikaz podataka pri izradi aplikacije koristi se vizualna komponenta *TDBGrid*. *TDBGrid* prikazuje podatke u tabličnom formatu, gdje svaki red predstavlja jedan zapis iz baze podataka, a svaki stupac predstavlja polje u tom zapisu. *TDBGrid* omogućuje korisnicima uređivanje podataka izravno unutar tabličnog prikaza, čime se olakšava ažuriranje podataka bez potrebe za otvaranjem posebnih prozora ili obrazaca za uređivanje. Komponenta omogućuje sortiranje podataka po određenim stupcima, kao i primjenu filtera za prikazivanje samo određenih zapisa koji zadovoljavaju određene uvjete. *TDBGrid* se obično koristi u kombinaciji s *DataSource* komponentom, koja služi kao veza između grafičkog korisničkog sučelja i npr. *TFDQuery* komponente. Glavna uloga *TFDQuery* komponente je izvršavanje SQL upita. Programer unutar svojstva *SQL* definira tekstualni upit koji se šalje bazi podataka. Ovaj upit može biti jednostavan `SELECT * FROM racuni`, kao što je jedan od upita iz naše aplikacije, ili složeniji s filtrima, pridruživanjem tablica, grupiranjem i drugim SQL funkcionalnostima. Nakon izvršavanja upita, podatci se pohranjuju u interni dataset unutar *TFDQuery* komponente, koji se može koristiti za prikaz ili daljnju obradu podataka.

Jedna od korisnih značajki *TFDQuery* komponente je i podrška za rad s parametrima unutar SQL upita. Na primjer, upit `SELECT * FROM racuni WHERE user = :user` omogućava korištenje parametra `:user`, čija se vrijednost dinamički postavlja u kôdu. U našoj aplikaciji njemu se dodjeljuje vrijednost varijable `_prijavljeniKorisnik` i koristi se za dohvaćanje podataka o računima samo korisnika koji je prijavljen u aplikaciju.

Svojstvo `Active` kontrolira je li upit aktivan ili ne. Kada se postavi na `true`, upit se izvršava, a rezultati su dostupni u dataset-u. Ova funkcionalnost omogućava jednostavno rukovanje podatcima bez potrebe za dodatnim kôdom za upravljanje vezama ili osvježavanjem podataka. To možemo vidjeti i na primjeru kôda iz aplikacije koji je prikazan na slici 5.6. Funkcija *FormCreate* se automatski poziva prilikom stvaranja (učitavanja) forme, odnosno kada se forma prvi put prikazuje korisniku. U ovom slučaju uloga funkcije je postavljanje veza i parametara za različite SQL upite koji će se koristiti na formi. Svaki upit koristi istu vezu prema bazi (`GlobalFDConnection`) i parametar `username`, što znači da su svi podaci na formi specifično vezani za trenutno prijavljenog korisnika (`_prijavljeniKorisnik`). Ova struktura omogućava da forma prikazuje personalizirane informacije i podatke o računu, transakcijama i izračunima.

```
20
   void __fastcall TFormRacuni::FormCreate(TObject *Sender)
   {
       // Postavljanje vrijednosti parametra upita iz varijable _prijavljeniKorisnik
       FDQueryRacuni->Connection = GlobalFDConnection;
       FDQueryRacuni->ParamByName("username")->AsString = _prijavljeniKorisnik;
       FDQueryRacuni->Active = true;

       FDQueryCalculated->Connection = GlobalFDConnection;
       FDQueryCalculated->ParamByName("username")->AsString = _prijavljeniKorisnik;
30     FDQueryCalculated->Active = true;

       FDQueryTrans->Connection = GlobalFDConnection;
       FDQueryTrans->Active = true;
   }
```

Slika 5.6 – Postavljanje parametara za TFDQuery komponente

Ako se vratimo na okvir za dizajn glavnog dijaloga (slika 5.5), ispod svake tablice možemo vidjeti i tipke za navigaciju, odnosno tipke za kretanje kroz zapise u tablici te njihovo uređivanje i brisanje. Za kreiranje navigacijskih tipki koristimo komponentu *TDBNavigator*. *TDBNavigator* omogućuje korisnicima navigaciju kroz zapise u povezanoj bazi podataka. Ova komponenta unutar aplikacije olakšava pristup i manipulaciju podacima. Sadrži gumbe za navigaciju kroz zapise u bazi podataka, omogućujući korisnicima da npr. prelaze na prvi, prethodni, sljedeći i posljednji zapis.

Unutar korisničkog sučelja aplikacije moguće je i prikazivanje slika pohranjenih u bazi podataka. Moguće je implementirati funkcionalnosti za dodavanje, uređivanje, i brisanje slika iz baze podataka, čime se omogućuje fleksibilno upravljanje slikama unutar aplikacije. U tu svrhu

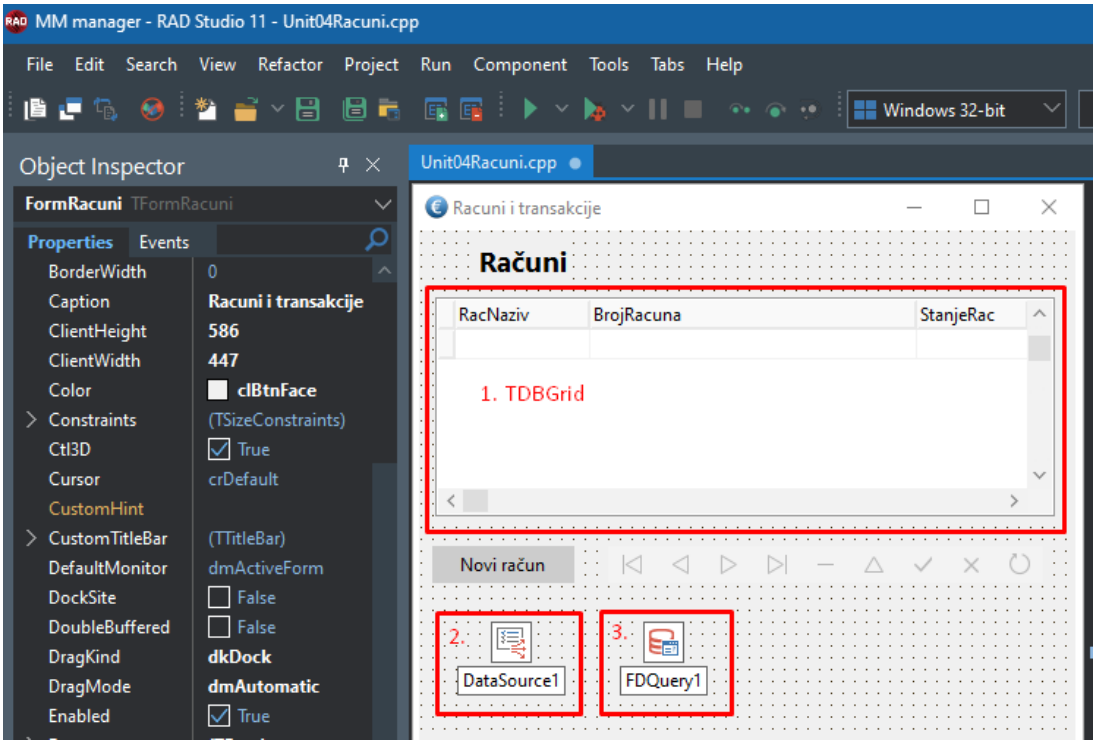
koristimo komponentu *TDBImage* koja podržava različite formate slika, uključujući popularne formate kao što su *jpeg*, *png*, *bmp* i *gif*. Isto kao i sve ostale komponente koje koriste podatke iz baze, tako se i *TDBImage* koristi u kombinaciji s *DataSource* komponentom, koja služi kao veza između baze podataka i grafičkog korisničkog sučelja. U našoj aplikaciji *TDBImage* koristimo za prikaz slika kartica tekućih i žiro računa, te bar kodova koji se mogu koristiti za plaćanja, a sadrže podatke o samoj transakciji.

5.4. Primjena vizualnih komponenti

Sve ovo do sada opisano prikazat ćemo kroz primjer kako bismo demonstrirali koliko je zapravo jednostavna izrada aplikacije pomoću vizualnih komponenti u C++ Builderu. Izdvojit ćemo jedan dio izrade naše aplikacije u kojem ćemo kreirati novi dijalog za prikaz podataka iz naše baze podataka.

5.4.1. Primjer 1 – Tablični prikaz podataka iz baze

Za prikaz ćemo koristit tablicu koju ćemo dobiti dodavanjem komponente *TDBGrid*. Da bi *TDBGrid* „znao“ koje će podatke prikazivati, moramo dodati još dvije komponente, a to su *TDataSource* i *TFDQuery*.

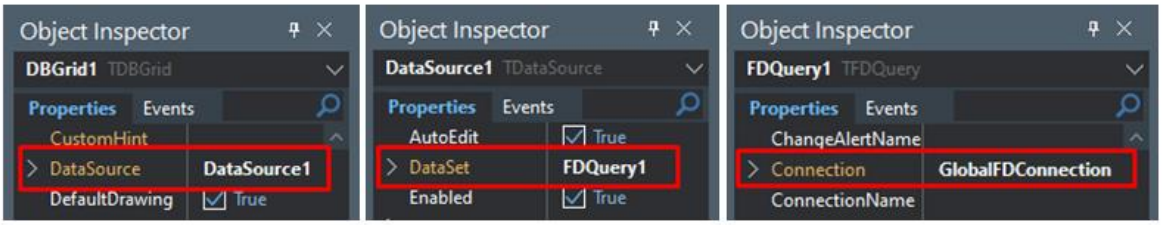


Slika 5.7 – Dodavanje komponenti na dijalog aplikacije

Ovdje bi nam bila potrebna još jedna komponenta, a to je *TFDConnection* koja se koristi za definiranje konekcije i spajanje na bazu podataka. Mi ju ovdje nećemo koristiti jer spoj na bazu imamo programski implementirano u *GlobalConnection.cpp* datoteci (slika 5.3). Sve potrebne komponente povlačimo mišem iz dijaloga *Palette* unutar razvojnog sučelja. Postavljanjem spomenutih komponenti u razvojnom sučelju aplikacije dobit ćemo prikaz kao na slici 5.7.

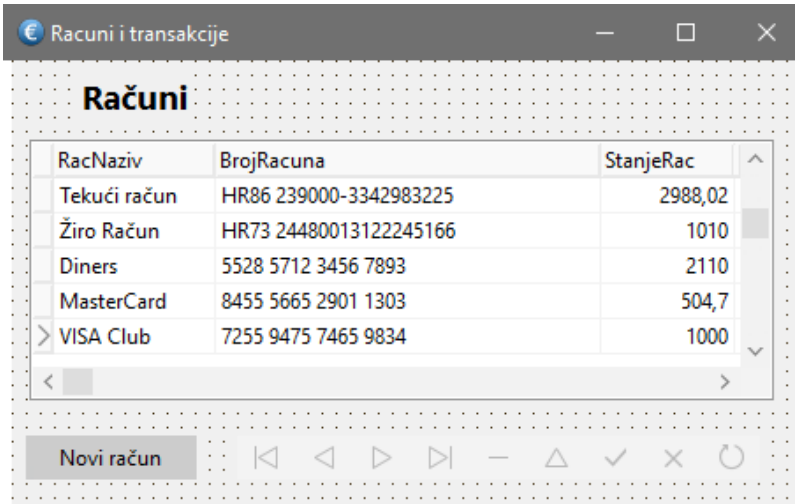
Da bi naša aplikacija ispravno radila potrebno je još samo definirati par svojstava dodanih komponenata (slika 5.8):

- *TDBGrid* – podesiti svojstvo *DataSource* na *DataSource1*
- *DataSource1* – podesiti svojstvo *DataSet* na *FDQuery1*
- *FDQuery1* – podesiti svojstvo *Connection* na *GlobalFDConnection* i SQL tekst:
  *SELECT * FROM racun WHERE username = :username*



Slika 5.8 – Podešavanje svojstava komponenti

Podešavanjem ova tri svojstva i postavljanjem svojstva *Active* na *true* za komponente *GlobalFDConnection* i *FDQuery1*, naša aplikacija je spremna za pokretanje. Pritiskom na tipku *F9* ili stavku *Run* iz glavnog izbornika razvojnog okruženja, te nakon uspješnog kompajliranja, u tablici dijaloga naše aplikacije prikazat će nam se podaci iz baze podataka točno prema upitu koji smo definirali u *FDQuery1* komponenti (slika 5.9).



Slika 5.9 – Izgled aplikacije nakon pokretanja

I najvažnija činjenica koju bismo sada trebali istaknuti, a koja je ujedno i tema ovog završnog rada jest da smo s ovih par jednostavnih koraka u izradi aplikacije, postavljanjem ove tri vizualne komponente zamijenili više desetaka linija programskog kôda i vrlo jednostavno izradili grafičko korisničko sučelje aplikacije. Za izradu ovog dijela naše aplikacije bilo nam je potrebno samo nekoliko minuta i par klikova čime smo korištenjem VCL komponenti C++ Buildera dobili jednostavnu, ali funkcionalnu aplikaciju.

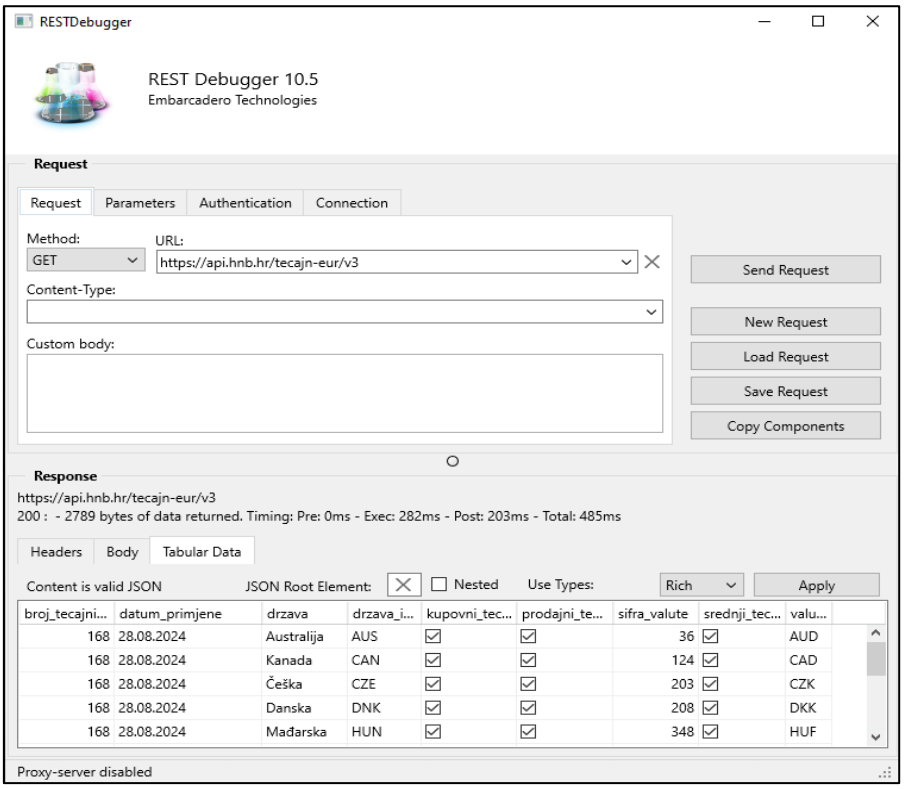
U prethodnom poglavlju ovog završnog rada spomenuli smo da C++ Builder sadrži nekoliko stotina vizualnih i ne vizualnih komponenti, pa ako razmotrimo koliko malo komponenti nam je bilo potrebno da napravimo funkcionalnu aplikaciju, možemo reći da su mogućnosti C++ Buildera u izradi Windows aplikacija skoro neograničene.

5.4.2. Primjer 2 – Izrada REST servisa

U nastavku ćemo prikazati još jedan primjer jednostavnosti korištenja vizualnih komponenti u kombinaciji sa alatom *RESTDebugger* [30]. *RESTDebugger* je alat koji se u sklopu razvojnog okruženja RAD Studio koristi za dohvaćanje podataka sa udaljenih REST API servisa, preuzimanja tih podataka u npr. JSON formatu i parsiranje, odnosno prevođenje istih radi lakšeg prikaza na grafičkom sučelju aplikacije.

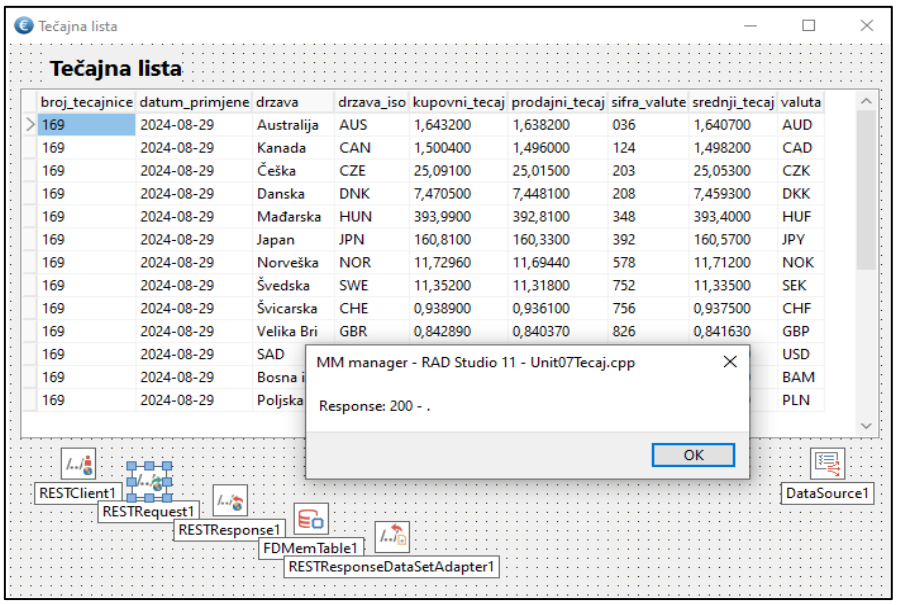
Za ovaj primjer ćemo upotrijebiti dijalog *Tečajna lista* iz naše aplikacije na kojem se nalazi tablica kreirana pomoću komponente *DBGrid.* U tablici ćemo prikazivati važeću Tečajnu listu na određeni datum preuzetu sa API servisa Hrvatske narodne banke [31].

Da bismo izradili ovaj dio aplikacije potrebno je na razvojnom sučelju dodati novi dijalog te postaviti i konfigurirati potrebne komponente. Komponente koje nam trebaju su *DBGrid*, *RESTClient*, *RESTRequest*, *RESTResponse*, *RESTResponseDataSetAdapter*, *FDMemTable* i *DataSource*. Od svih navedenih komponenata, odabirom iz dijaloga *Palette,* na dijalog aplikacije ćemo postaviti samo *DBGrid* i *DataSource*. Sve ostale komponente kreirat ćemo pomoću alata *RESTDebugger* [32]. Na slici 5.10 vidimo korisničko sučelje alata *RESTDebugger* u koji unosimo poveznicu API-a te odabiremo metodu koju ćemo koristiti pri slanju zahtjeva. Mi odabiremo *GET* metodu koja služi za dohvat sadržaja sa navedene poveznice. S obzirom da taj API za dohvat tečaja ne koristi nikakve parametre niti autorizaciju, te podatke nismo morali unositi. Sada je potrebno kliknuti na gumb *Send Request* i u tabličnom prikazu na kartici *TabularData* će nam se pojaviti sadržaj odgovora sa udaljene lokacije na koju smo poslali upit. Nama će se pojaviti trenutna tečajna lista HNB-a.



Slika 5.10 - Korisničko sučelje alata REST Debugger

Nakon ispravno obavljenog upita, potrebno je kliknuti na *Copy Components* i tada će nam *RESTDebugger* sam izraditi ostatak potrebnih komponenti za našu aplikaciju i spremiti ih u međumemoriju [29]. Kopirane komponente potrebno je zalijepiti na dijalog *Tečajna lista* (iz glavnog izbornika razvojnog sučelja odabrati *Edit -> Paste*) i zatim *DataSource1* komponentu povezati sa *FDMemTable1* komponentom. Desnim klikom na komponentu *RESTRequest* i odabirom *Execute* izvršit će se upit, te ćemo dobiti odgovor kao na slici 5.11.



Slika 5.11 - Izgled dijaloga „Tečajna lista“ nakon izvršenog upita

I ovim primjerom smo pokazali na koji način je uz pomoć vizualnih komponenti RAD Studio alata vrlo jednostavno aplikaciji dodati još jednu funkcionalnost.

5.4.3. Primjer 3 – Izrada i ispis izvještaja iz aplikacije

Postupak izrade izvještaja u C++ Builderu je također jednostavan postupak. Međutim, da bismo mogli generirati izvještaje prvo je potrebno skinuti dodatak *FastReport* iz repozitorija razvojnog okruženja.

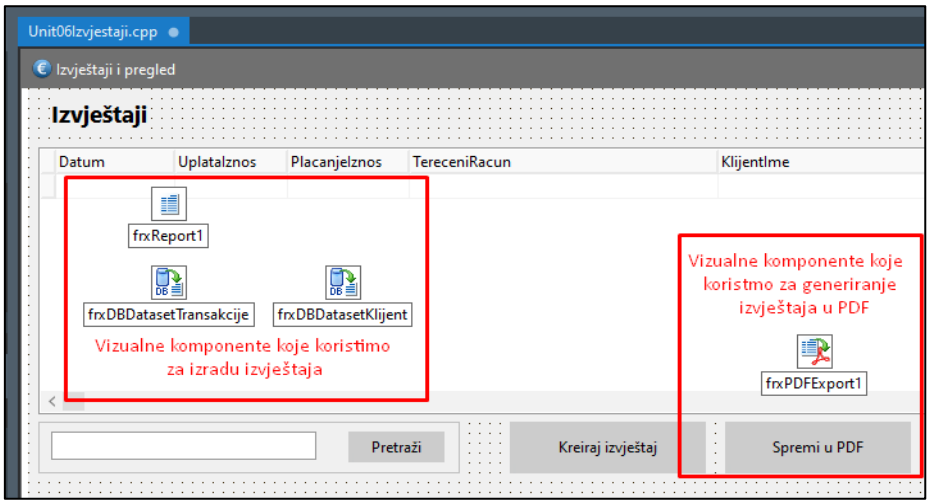
FastReport [33] je napredni alat za izradu izvještaja koji se koristi u raznim programskim okruženjima, uključujući i C++ Builder. Ovaj dodatak nudi bogat skup značajki za generiranje, ispis i izvođenje izvještaja omogućujući programerima brzo i jednostavno kreiranje profesionalnih izvještaja uz minimalno programiranje.

Jedna od karakteristika *FastReport*-a je njegov grafički dizajner koji omogućuje korisnicima postavljanje vizualnih elemenata izvještaja samo povlačenjem i ispuštanjem. Ovaj dizajner nudi intuitivno sučelje koje olakšava dodavanje komponenti kao što su tekstualna polja, grafovi, slike i druge vizualne komponente bez potrebe za dubokim poznavanjem kodiranja.

FastReport podržava razne izvore podataka uključujući baze podataka, XML datoteke i objekte u memoriji. Ovo omogućava fleksibilnost u načinu na koji se podaci dohvaćaju i koriste u izvještajima. Također podržava različite formate izvoza uključujući PDF, Excel, RTF, HTML i druge što olakšava dijeljenje izvještaja s krajnjim korisnicima.

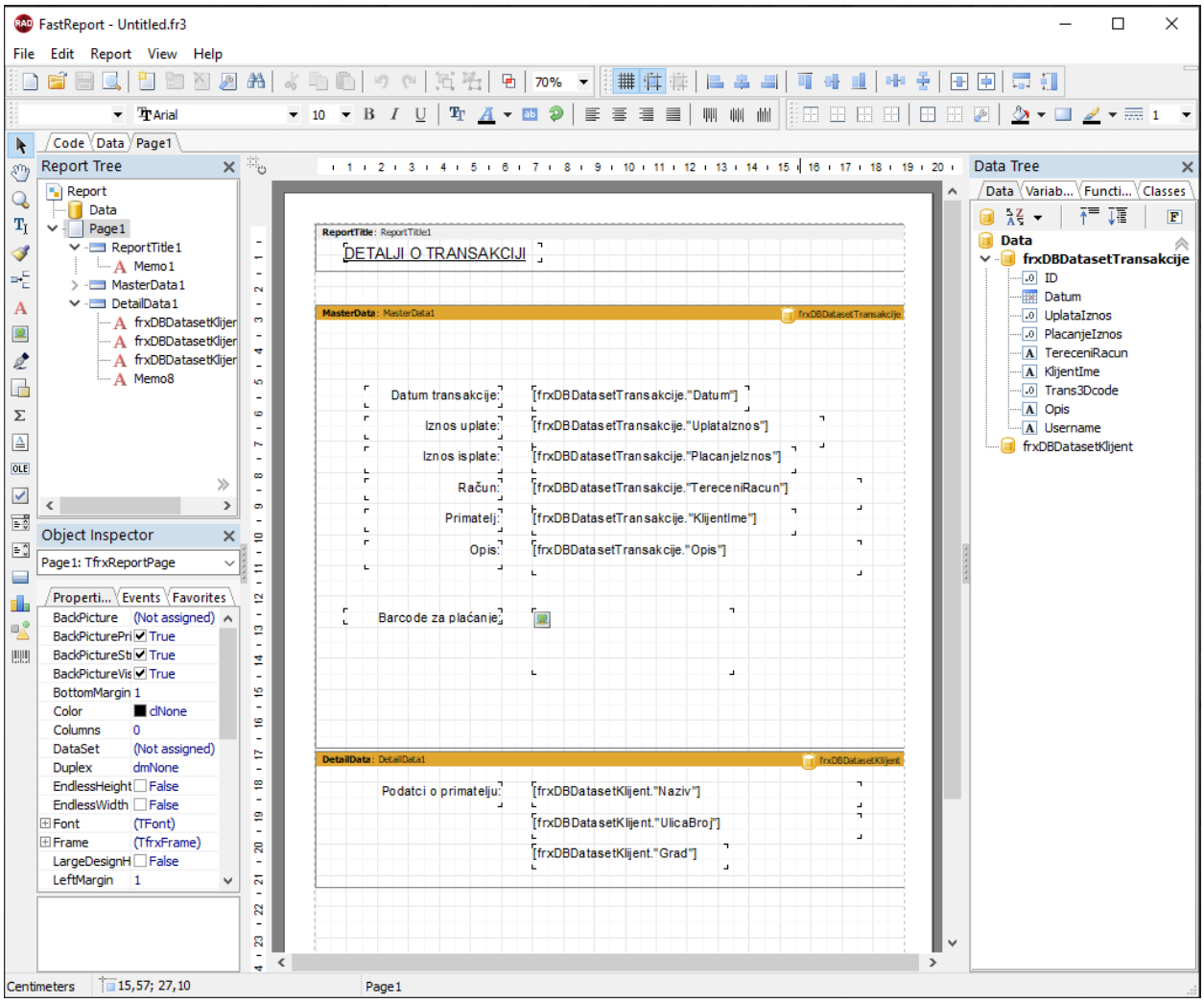
FastReport se koristi kroz nekoliko ključnih komponenti. Prva je `TfrxReport`, glavna komponenta za kreiranje i generiranje izvještaja. Ova komponenta povezuje podatke i upravlja prikazom izvještaja. Za povezivanje s bazom podataka koristi se `TfrxDBDataset` komponenta koja omogućuje da podaci iz baze budu dostupni unutar izvještaja.

U našoj aplikaciji kreiranje izvještaja ćemo izraditi na dijalogu na kojem su u tabličnom prikazu prikazane sve dosada obavljene transakcije. Izvještaj će nam služiti da iz tablice, odnosno baze podataka, prikažemo samo one podatke koji su nam potrebni. Moguće je izraditi i više izvještaja ovisno o potrebama za podacima koje želimo koristiti za daljnju upotrebu. Prvi korak u izradi izvještaja je postavljanje potrebnih vizualnih komponenti unutar dijaloga (slika 5.12). Tako ćemo sa palete komponenti iz grupe *FastReport VCL* odabrati komponentu *frxReport* i *frxDBDataset*. Mi ćemo koristiti dvije dataset komponente jer ćemo u izvještaju koristiti podatke iz dvije tablice u bazi i za svaku od njih nam je potrebna posebna komponenta. U postavkama komponenti pod svojstvom *DataSource* odabrat ćemo vezu koju ćemo koristiti za dohvat podataka, a to su u našem slučaju *DataSourceTransakcija* i *DataSourceKlijent* komponente koje koristimo i za prikaz podataka u tablici unutar dijaloga „Izvještaji“.



Slika 5.12 – Dijalog za izradu izvještaja s postavljenim komponentama

Kada se postave komponente i podese njihove postavke dvostrukim klikom na komponentu *frxReport* otvara se *FastReport* dizajner, grafičko sučelje koje omogućuje postavljanje vizualnih elemenata izvještaja. Ovdje se mogu dodati zaglavlja, podnožja, tekstualna polja i grafovi.



Slika 5.13 – Korisničko grafičko sučelje FastReport dizajnera

U izborniku s lijeve strane imamo alatnu traku s komponentama koje nam služe za dodavanje raznih naslova, teksta, grafičkih oblika i slično, poput onih koji se koriste u uređivačima teksta. S desne strane imamo stablo s prikazom polja podataka u tablicama koje smo komponentama *frxDBDatasetTransakcije* i *frxDBDatasetKlijent* prethodno povezali s bazom podataka. Sada je samo potrebno određeni podatak sa tog stabla koji želimo koristiti u izvještaju, mišem prevući na naš predložak i na tom mjestu će nam nakon kreiranja izvještaja taj podatak ovdje biti prikazan. Ukoliko želimo tijekom izrade izvještaja pogledati kako će nam izvještaj izgledati, iz glavog izbornika možemo odabrati opciju *File → Preview*. Nakon što smo posložili izgled izvještaja, potrebno ga je spremiti u korijensku mapu našeg projekta i zatvoriti *FastReport* dizajner.

Da bi kompletirali ovu funkcionalnost potrebno je još na neki način u aplikaciji pokrenuti kreiranje izvještaja. To možemo implementirati postavljanjem gumba na dijalogu kojem ćemo dodijeliti funkciju kreiranja izvještaja. Kada postavimo gumb na dijalog, otvorimo uređivač koda na *OnClick* događaj i jednostavno komponenti *frxReport1* pozovemo metodu *ShowReport()* koja je dio klase same komponente (slika 5.14).

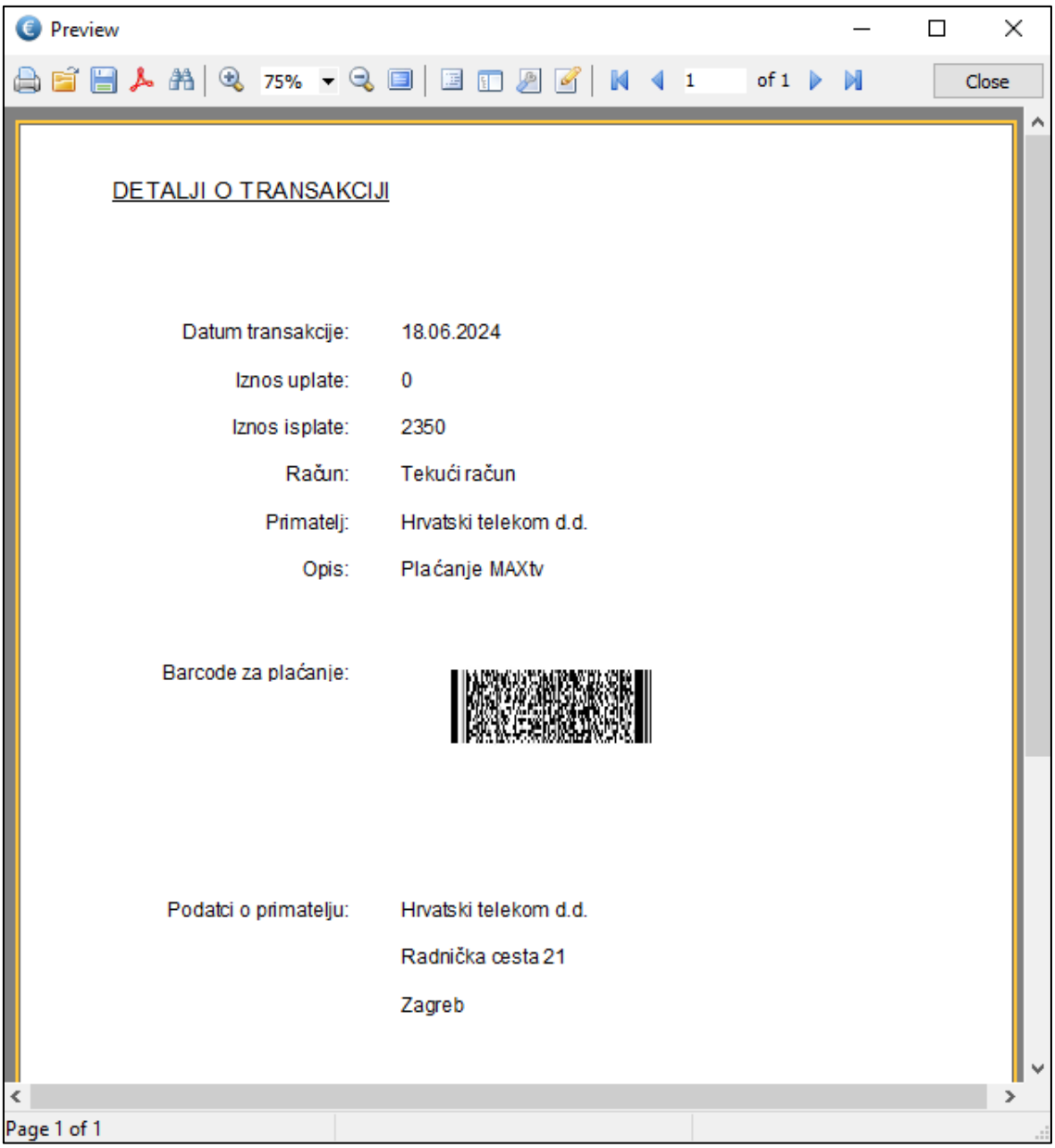
```
void __fastcall TFormIzvjestaji::ButtonIzvjestajClick(TObject *Sender)
{
    frxReport1->ShowReport();
}
//---------------------------------------------------------------------------


void __fastcall TFormIzvjestaji::ButtonSavePdfClick(TObject *Sender)
{
    //frxPDFExport1->FileName = "Detalji_transakcije.pdf";
    // Definirano u vizualnoj komponenti

    frxReport1->PrepareReport(true);
    frxReport1->Export(frxPDFExport1);
}
//---------------------------------------------------------------------------
```

Slika 5.14 – Dodjeljivanje funkcija gumbima za kreiranje i prikaz izvještaja

Alat FastReport, kao što smo već spomenuli, nudi i generiranje izvještaja u raznim formatima datoteka. U našoj aplikaciji ćemo izvještaj koji smo izradili generirati u PDF-u. Za to nam je potrebna još jedna vizualna komponenta koja sadrži sve potrebno da bismo implementirali ovu funkcionalnost. Iz palete komponenti na naš dijalog ćemo postaviti komponentu *frxPDFExport* i u editor koda samo dodati dvije linije koda koje će pozvati metodu za pripremu izvještaja i izvoz izvještaja u PDF (slika 5.14).

I kod ove funkcionalnosti možemo vidjeti koliko nam C++ Builder nudi mogućnosti za jednostavnu izradu aplikacija gdje smo uz pomoć nekoliko komponenti i par linija koda dodali

mogućnost izrade izvještaja. Naravno, svaki generirani izvještaj moguće je i ispisati na pisač koji je instaliran na računalu na kojem je aplikacija pokrenuta. U našem slučaju izvještaj sadrži detalje o transakciji.



Slika 5.15 – Izvještaj generiran u PDF

Na slici 5.15 vidimo izvještaj koji sadrži datum, iznos i opis plaćanja, te podatke o primatelju. Prikazan je i QR kod pomoću kojeg je moguće izvršiti plaćanje putem mobilnog bankarstva. Na način opisan u ovoj cjelini možemo izraditi razne izvještaje, a sadržaj određujemo slaganjem komponenti na samom predlošku.

5.5. Dovršetak izrade aplikacije

Kada su svi elementi aplikacije razvijeni i integrirani, dolazi trenutak završne obrade. Prvi korak u ovom procesu je detaljno testiranje aplikacije. Jedna od najvećih prednosti desktop aplikacija je njihova brzina i performanse. Budući da se pokreću lokalno, desktop aplikacije imaju izravni pristup resursima uređaja, što omogućava brže izvršavanje složenih zadataka i obradu velikih količina podataka. To je posebno važno za softver koji zahtijeva visok stupanj računalne snage ili rad u offline okruženju, dok recimo web aplikacije često ovise o brzini internetske veze, što može rezultirati kašnjenjem i slabijom izvedbom u intenzivnim zadacima.
Iz tih razloga testiranje treba uključivati provjeru svih funkcionalnosti, kao i otkrivanje i ispravljanje potencijalnih grešaka.

Nakon što su svi problemi riješeni, sljedeći korak je optimizacija aplikacije. Ovaj proces može uključivati poboljšanje brzine izvođenja, smanjenje veličine datoteka i optimizaciju korisničkog sučelja. Također je važno osigurati da aplikacija ispunjava sve zahtjeve vezane uz sigurnost, kako bi se zaštitili korisnički podaci.

Podaci pohranjeni lokalno na uređajima mogu pružiti viši stupanj privatnosti jer nisu izloženi rizicima online napada, poput hakerskih upada u udaljene servere. S druge strane, sigurnost desktop aplikacija više ovisi o samom korisniku koji mora redovito ažurirati softver i održavati sigurnost sustava.

Nakon završetka testiranja i optimizacije priprema se instalacijski paket. C++ Builder nudi mogućnosti izrade instalacijskih datoteka koje olakšavaju distribuciju aplikacije. Tijekom ovog procesa važno je uključiti sve potrebne komponente i knjižnice koje aplikacija zahtijeva.

Uzimajući u obzir sve navedeno i koristeći se opisanim načinom izrade aplikacije u C++ Builderu izrada cijele aplikacije je bila puno jednostavnija nego što bi to bilo pisanjem koda za svaku funkcionalnost. Vizualne komponente, njihova predefinirana svojstva i funkcije omogućuju programerima da više vremena posvete izgledu i funkcionalnosti aplikacija. Aplikacija koju sam izradio, iako je još uvijek u razvojnoj fazi, sadrži dvadesetak raznih funkcionalnosti i bila mi je dobra osnova za izradu ovog završnog rada.

1. Zaključak

U ovom završnom radu istaknuo sam važnost kvalitetnog dizajna korisničkog sučelja u razvoju aplikacija. Funkcionalnost i korisničko iskustvo bitni su u određivanju uspješnosti softverskih rješenja. Kroz praktične primjere prikazao sam proces dizajniranja korisničkog sučelja koristeći C++ Builder, ističući prednosti i fleksibilnost ovog razvojnog alata.

Ovaj razvojni alat koristi C++ programski jezik koji je dizajniran tako da može riješiti širok spektar problema i koristiti se za različite vrste aplikacija. To uključuje razvoj operativnih sustava, jezgrenih modula, aplikacija za stolna računala, igara, mrežnih aplikacija, kao i softvera za ugradbene sustave. C++ kombinira karakteristike jezika niske razine (bliske strojnom jeziku) i jezika visoke razine (koji su više apstraktni i bliži ljudskom jeziku). On omogućava pristup osnovnim operacijama na hardverskoj razini poput direktnog upravljanja memorijom pomoću pokazivača, ali istovremeno pruža visokorazinske apstrakcije kao što su objekti, klase i generičko programiranje.

Korištenjem C++ Buildera vidjeli smo i koje mogućnosti nam pruža VCL biblioteka i kako iskoristiti njene komponente za brzo i efikasno razvijanje korisničkih sučelja za Windows aplikacije. Prikazali smo da je korištenje ugrađenih komponenti, koje već sadrže potrebne funkcionalnosti, često efikasnije od pisanja velike količine programskog kôda. Demonstrirali smo to i prilikom izrade aplikacije kroz upotrebu osnovnih elemenata poput prozora, dijaloga, tablica, gumba i okvira. Njihova svojstva, metode i događaji jednostavni su za implementaciju.

Međutim, meni kao početniku koji tek ulazim u svijet programiranja, izrada jednog ovakvog projekta nije bila nimalo jednostavna. Upoznavanje s bazama podataka, objektno orijentiranim programiranjem i C++ programskim jezikom zahtjeva puno vremena provedenog na pisanju koda, traženju i ispravljanju pogrešaka, a to nije nimalo lak zadatak. Izrada aplikacije i sama priprema trajala je nekoliko mjeseci, ali na kraju mogu samo zaključiti da uz naporan rad i korištenje alata kao što je C++ Builder izrada aplikacija može biti puno jednostavnija. Nadam se isto tako, iako je ovaj završni rad samo zagrebao po površini OOP-a, C++ programskog jezika i razvojnog okruženja RAD Studio, da će svakome tko je zainteresiran za svijet razvoja aplikacija biti barem mala pomoć, makar u odabiru razvojne platforme i programskog jezika kojeg će koristiti.

2. Literatura

[1] Hrvatska enciklopedija, mrežno izdanje, Leksikografski zavod Miroslav Krleža, 2013.-2024., (https://www.enciklopedija.hr/clanak/aplikacija), dostupno 19.8.2024.
[2] https://dir.hr/sto-je-ux-ui-dizajn, dostupno 23.9.2024.
[3] https://www.ditdot.hr/ux-i-ui-dizajn-osnove-korisnickog-iskustva, dostupno 23.9.2024.
[4] Jansen, Bernard J. "The graphical user interface."ACM SIGCHI Bulletin 30.2 (1998): 22-26.
[5] Julijan Šribar i Boris Motik: Demistificirani C++: dobro upoznajte protivnika da biste njime ovladali - 5. dopunjeno izdanje usklađeno sa standardom C++11/C++14., Zagreb, Element, 2018.
[6] https://www.geeksforgeeks.org/abstraction-in-cpp, dostupno 23.8.2024.
[7] https://www.geeksforgeeks.org/encapsulation-in-cpp, dostupno 23.8.2024.
[8] https://www.geeksforgeeks.org/inheritance-in-c, dostupno 23.8.2024.
[9] https://www.geeksforgeeks.org/cpp-polymorphism, dostupno 23.8.2024.
[10] https://www.geeksforgeeks.org/c-classes-and-objects, dostupno 23.8.2024.
[11] David R. Mausser, Atul. Saini: The STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library, Addison Wesley Longman Publishing Co., Inc., 1995.
[12] https://learn.saylor.org/mod/page/view.php?id=29934, dostupno 13.09.2024.
[13] https://en.wikipedia.org/wiki/FireMonkey, dostupno 13.9.2024.
[14] https://docwiki.embarcadero.com/RADStudio/Athens/en/FireDAC, dostupno 15.9.2024.
[15] https://www.embarcadero.com/products/rad-server, dostupno 15.9.2024.
[16] https://docwiki.embarcadero.com/RADStudio/Athens/en/Index, dostupno 15.9.2024.
[17] https://docwiki.embarcadero.com/Libraries/Alexandria/en/System.Classes, dostupno 16.9.2024.
[18] https://www.google.hr/ books/edition/Windows _Forms_Programming_in_C, dostupno 17.9.2024.
[19] Ray Rischpater: Application Development with Qt Creator - Second Edition, Packt Publishing Ltd, 27. stu 2014
[20] Scarpino, M. Holder, S., Ng, S., & Mihalkovic, L. SWT/JFace, Manning Publications Co, 209 Bruce Park Avenue, Greenwich, 2004
[21] dr. sc. Ž. Kovačević: Napredne tehnike programiranja u C++ BUILDERU, Tehničko veleučilište u Zagrebu, (https://www.tvz.hr/wp-content/uploads/2022/11/Napredne-tehnike-programiranja-u-C-Builderu.pdf, dostupno 21.9.2024.)
[22] 19-1. Krug, S. (2000). *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders Press.]
[23] Norman, D. A.: The Design of Everyday Things, Basic Books, 2013
[24] Cooper A., Reimann R., & Cronin D.: About Face: The Essentials of Interaction Design, Wiley, 2007
[25] https://dzone.com/articles/mastering-persistence-why-the-persistence-layer-is, dostupno 17.9.2024.
[26] https://www.mongodb.com/resources/basics/databases/data-persistence, dostupno 17.9.2024.
[27] https://www.sqlite.org, dostupno 22.8.2024.
[28] https://www.apachefriends.org/ 23.09.2024
[29] https://dbeaver.com/docs/dbeaver, dostupno 17.9.2024.
[30] https://corneliusconcepts.tech/trestresponsedatasetadapter-and-hidden-gems-rest-debugger, dostupno 22.9.2024.
[31] https://api.hnb.hr/tecajn-eur/v3, dostupno 24.8.2024.
[32] https://blogs.embarcadero.com/everything-you-need-to-use-rest-in-your-apps-now, dostupno 23.9.2024.
[33] https://origin2.cdn.componentsource.com/sites/default/files/resources/fast-reports/888146/frvclusermanual-en-jun2024.pdf, dostupno 25.09.2024.

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, **Danijel Kovačević**, pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor završnog rada pod naslovom **„Izvedba grafičkog korisničkog sučelja aplikacije za Windows platformu pomoću C++ Buildera“** te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student:
Danijel Kovačević

(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.