Aplikacija za kalkulaciju troškova izrade namještaja

Štiglec, Mislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA KALKULACIJU TROŠKOVA IZRADE NAMJEŠTAJA

Završni rad

Mislav Štiglec

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 14.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Mislav Štiglec
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4287, 26.07.2018.
OIB studenta:	80758295293
Mentor:	Izv. prof. dr. sc. Emmanuel Karlo Nyarko
Sumentor:	dr.sc. Ivana Hartmann-Tolić
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za kalkulaciju troškova izrade namještaja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	14.09.2021.
Datum potvrde ocjene Odbora:	22.09.2021.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 23.09.2021.

Ime i prezime studenta:	Mislav Štiglec
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4287, 26.07.2018.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za kalkulaciju troškova izrade namještaja**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Emmanuel Karlo Nyarko

i sumentora dr.sc. Ivana Hartmann-Tolić

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

Potpis studenta:

SADRŽAJ

1. UVOD

Razvoj informatičkih i računalnih tehnologija već nekoliko desetljeća potpomaže razvoju gotovo svih grana proizvodnje. Jedan takav primjer bit će predstavljen ovim završnim radom.

Izrada namještaja sastoji se od više faza. Jedna od tih faza je proračun dimenzija namještaja, proračun količine materijala potrebnog za izradu određenog namještaja, proračun površine namještaja itd. Ova faza izrade namještaja može se, naravno, realizirati papirom i olovkom osnovnim matematičkim operacijama. Međutim, točnost navedenog postupka je upitna i smanjuje se s povećanjem opsega namještaja za izradu zbog većeg broja matematičkih operacija te veće mogućnosti pogreške. Također, ako se radi o većoj narudžbi u ovakav račun potrebno je uložiti puno vremena i koncentracije.

Zbog tih razloga povoljno je zamijeniti ljudski rad s računalnim programom ili u ovom slučaju računalnom aplikacijom. Računalnom programu povećanje opsega posla ne predstavlja niti vremensko niti fizičko opterećenje dok je vrijeme potrebno za izračun minimalno. Rad će predstaviti aplikaciju koja rješava opisani problem kalkulacije troškova izrade namještaja.

U drugom poglavlju rada navedene su postojeće inačice aplikacija koje rješavaju problem te su uspoređene s mojim rezultatom. Dalje je opisana sama aplikacija i svi njeni dijelovi kao što su grafičko sučelje, matematika izračuna svih potrebnih dimenzija i materijala, struktura koda te brzina aplikacije. Na kraju drugog poglavlja bit će opisano okruženje u kojemu je rad izrađen.

U trećem poglavlju navedena su moguća unaprjeđenja aplikacije i koji sve probleme rješava aplikacija.

1.1 Zadatak završnog rada

Završni rad zahtjeva izradu aplikacije koja će omogućiti kalkulaciju troškova izrade namještaja, prikazati rezultate kalkulacije te ih pohraniti u nekom od mogućih oblika.

2. PREGLED PODRUČJA TEME

U ovom poglavlju aplikacija je uspoređena s aktualnim komercijaliziranim primjerima aplikacija iste namjene. Daje se osvrt na razvojno okruženje, izgled aplikacije, kod aplikacije, ostvarene ciljeve itd.

2.1 Pregled postojećih inačica

U ovom trenutku postoji više aplikacija sa svrhom izračunavanja troškova izrade namještaja, izračunavanja cijene namještaja, veličine namještaja i ostalih matematičkih zahtjeva pri proizvodnji istoga. Većina komercijaliziranih aplikacija rješava sve navedene probleme i uz to nudi dizajn interijera, različite ponude od različitih proizvođača i proizvodnju po vlastitoj želji. Neki od ovakvih softvera su CORPUS [1] i Master-Design Art-Shop X-Lite [2]. Uz programe posebno dizajnirane za ovu upotrebu, svaka od velikih kompanija koja se bavi prodajom i izradom namještaja ima svoj program za njegovo izračunavanje. Primjer je IKEA. Može se napomenuti da bi se sama konstrukcija namještaja mogla napraviti u programima kao što su AutoCAD i SolidWorks koji nisu namijenjeni za matematički izračun ali odlično mogu prikazati element i njegove dimenzije u 3D okruženju.

2.2 Usporedba aplikacije sa sličnim aktualnim implementacijama

Aplikacija CORPUS je, po autorovom subjektivnom mišljenju najatraktivnija i sadrži sve elemente koje softver za izradu namještaja mora sadržavati te je u potpunosti specijaliziran za tu primjenu. Ono što CORPUS čini moćnijim i naprednijim od izrađene aplikacije je 3D predodžba svih rađenih elemenata, njihov međusoban odnos u prostoru te mogućnost izrade namještaja po volji kupca. Postoje još brojne prednosti tako visoko razvijenog programa no nabrojane su one osnovne razlike koje u konačnici odjeljuju vrhunsko optimizirani i komercijalizirani proizvod od ostalih aplikacija. Drugi primjer bi bila aplikacija Master-Design Art-Shop X-Lite. Za razliku od spomenutog CORPUS-a nije toliko vizualno atraktivna ali ima i trodimenzionalni dio u kojemu se može vidjeti konačni raspored izabranih elemenata. Ono što je negativna strana ovog komercijaliziranog programa je težina korištenja program, što je također karakteristika svake aplikacije nebitno koju ona svrhu imala. Naime, potrebno je određeno vrijeme da se savladaju funkcije koje program nudi te bih rekao da je također potrebno određeno znanje matematike i aritmetike kako bi se moglo brzo i jednostavno baratati programom, što na kraju nije dobro svojstvo jer se velika većina korisnika neće prilagoditi. U izrađenoj aplikaciji koja je prezentirana u ovom radu, trudilo se napraviti jednostavno sučelje koje će omogućiti jednostavnu interakciju s korisnikom bez potrebe za upoznavanjem aplikacije i njezinih funkcionalnosti prije korištenja.

3. REALIZACIJA PROBLEMA

Analizirani problem je rastavljen na dijelove kako bi bilo određeno sve ono što se mora implementirati kako bi se zadovoljili uvjeti završnog rada i također kako bi problematični i vremenski zahtjevniji dijelovi bili lakše predviđeni.. Neki od najopćenitijih problema bili su grafičko sučelje, implementacija točnog matematičkog izračuna za elemente te prikaz podataka tih elemenata.

3.1 Okruženje korišteno za izradu aplikacije

Okruženje korišteno za izradu aplikacije je Visual Studio 2019. Inačica Visual Studio okruženja je Visual Studio Community koji je besplatan za razliku od Professional i Enterprise verzije. Visual Studio pruža širok raspon alata koji omogućavaju razvoj različitih tipova aplikacija i programa. Visual Studio podržava razvoj desktop aplikacija (.NET development), mobilnih aplikacija (Xamarin,C++), računalnih igara (Unity,C++), razvoj u linux okruženju (Linux proširenje) i još mnogo različitih inačica računalnog softvera. Za razvoj desktop aplikacije isključivo je korišten .NET Framework [3] te je detaljnije opisan u nastavku prema.

.NET Framework predstavlja infrastrukturu koja omogućava laku izradu desktop aplikacija s već gotovim rješenjima i funkcionalnostima koje uvelike ubrzavaju cjelokupni proces izrade konačnog proizvoda tj. aplikacije. Visual Studio pa tako i .NET je napravljen od strane Microsofta pa je tako prilagođen Windows operacijskom sustavu. .NET podržava 3 programska jezika, C#, F# i Visual Basic. .NET pruža izradu *Windows Form* desktop aplikacije i unutar *Windows Form* predloška olakšana je izrada desktop aplikacije gotovim funkcionalnostima . *Windows Form* koristi C# kao zadani programski jezik koji je detaljnije opisan u nastavku rada [4].

.NET je strukturiran u 4 dijela a to su:

1) *Common Language Runtime* (CLR) – softverski sustav u kojem se kod izvršava, predstavlja posrednika između .NET-a i operacijskog sustava
2) *Framework Class library* (FCL) – standardna biblioteka u kojoj se nalaze sve klase koje su potrebne za izradu aplikacija
3) Moduli (engl. *Windows Forms*, ASP.NET) – tehnologije koje su prilagođene razvoju određenog tipa softvera kao što su na primjer desktop aplikacije ili web aplikacije

.NET je predstavljen 2001. godine i od tada glasi kao jedan od najmoćnijih i najoptimiziranijih infrastruktura za izradu softvera. On se razvija i danas te s Visual Studio okruženjem za koje je namijenjen čini odličan alat za izradu softverskih programa.

3.2 C# programski jezik

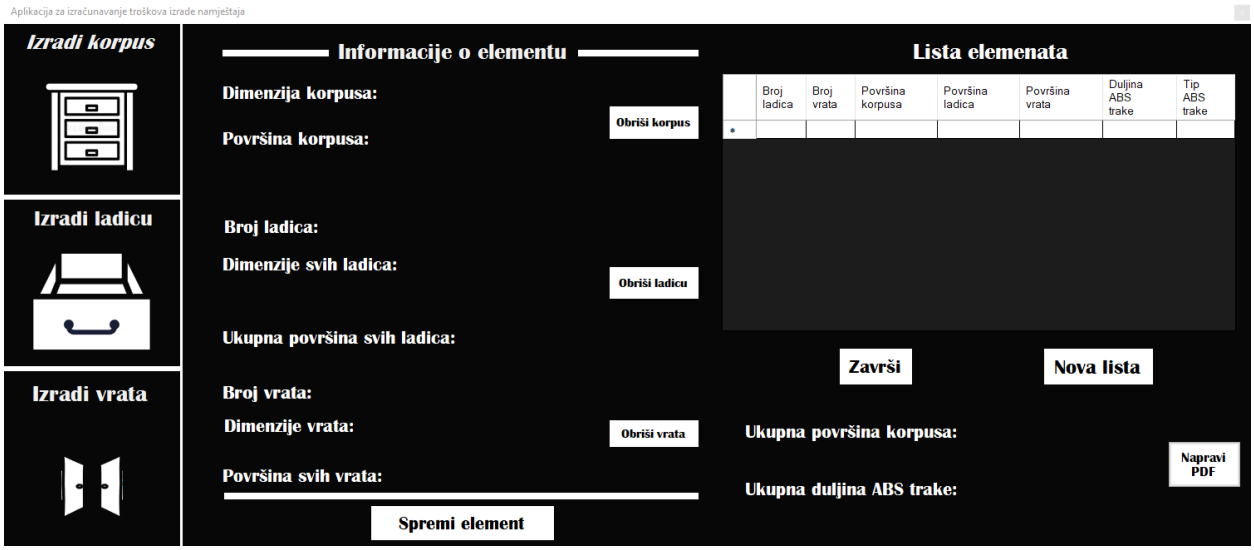
C# je objektno-orijentirani programski jezik predstavljen 2000. godine kao proizvod Microsoft-a. Ponajviše je namijenjen razvijanju aplikacija u .NET infrastrukturi. C# potječe od C i C++ programskih jezika te će osobe koje poznaju dva navedena jezika imati dobre temelje pri radu s C#-om. Neke od značajki C# jezika su:

1) Sakupljač smeća (engl. *Garbage collector*) – oslobađa memoriju koja je zauzeta objektima na koje više nema reference i ne mogu biti korišteni (više nisu u upotrebi)
2) Rukovanje iznimkama (engl. *Exception handling*) – pristup koji omogućava lako rukovanje greškama i lako pronalaženje istih
3) Lambda izrazi – podržavaju tehnike funkcijskog programiranja
4) LINQ – predložak koji omogućava lako rukovanje s podacima iz različitih izvora

Ovdje su nabrojane samo neke najbitnije značajke C# programskog jezika koje ga razlikuju od njegovih prethodnika. C# je objektno-orijentiran te u potpunosti prilagođen takvom tipu programiranja i temelji se na klasama, objektima, sučeljima i podržava sva načela objektno-orijentirane paradigme (nasljeđivanje, polimorfizam…).

3.3 Grafičko (korisničko) sučelje

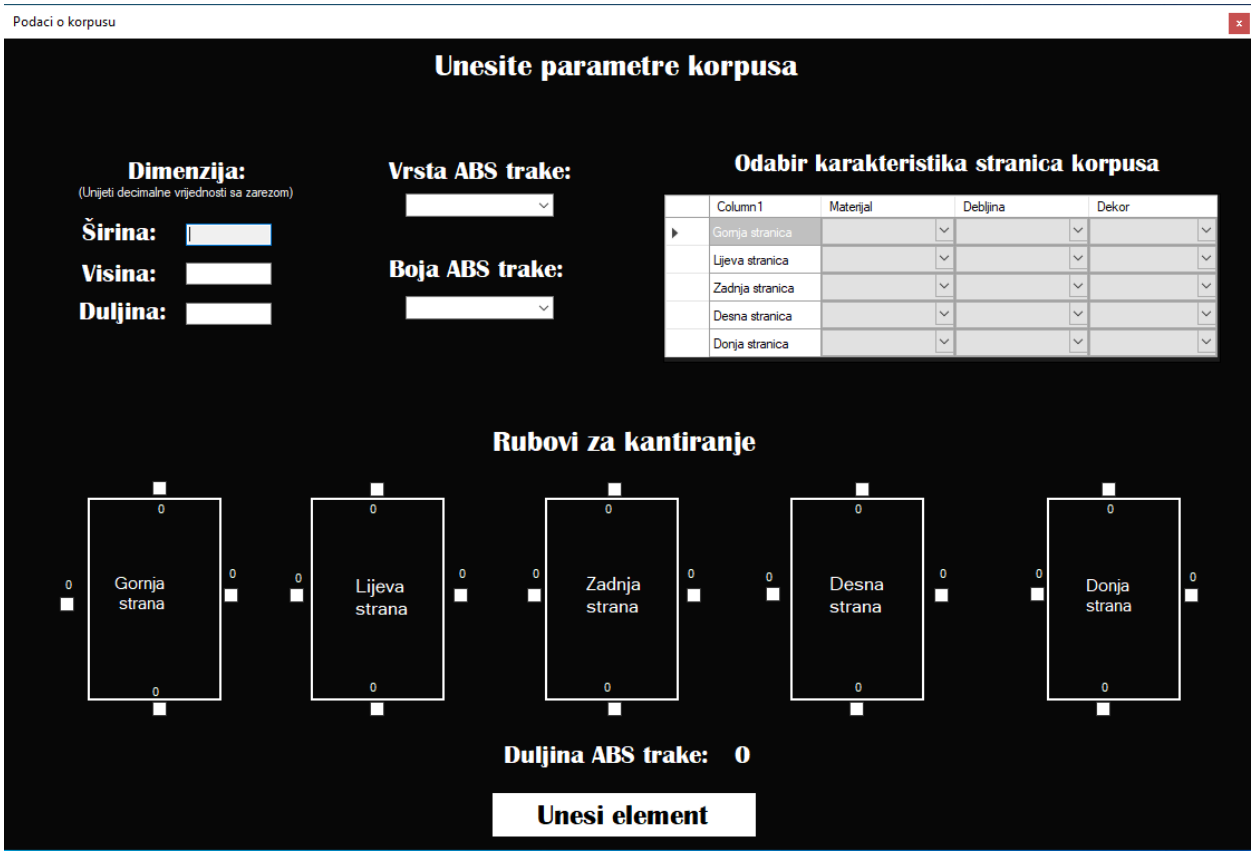
Grafičko sučelje predstavlja poveznicu između korisnika i aplikacije. Grafičko sučelje trebalo bi biti jednostavno i logično za korištenje. U pravilu ako korisnik mora pitati kako se ono koristi ili korištenje aplikacije mora biti objašnjeno, tada možemo govoriti o nedostatku kvalitete korisničkog sučelja. Na prvu izrada korisničkog sučelja izgleda kao najlakši dio posla, međutim izrada korisničkog sučelja može oduzeti najviše vremena i može predstavljati problem.. Korisnik koristi aplikaciju klikanjem na gumbe, upisivanjem teksta u predviđena polja ili klikanjem miša na određena područja aplikacije (sučelja). U nastavku je predočeno grafičko sučelje rada zajedno s njegovim opisom. Ono se sastoji od 4 prozora gdje je prvi prozor ujedno i glavni prozor (engl. *Main Window*) dok su ostala tri prozora pomoćna i služe za odvajanje funkcionalnosti aplikacije na više strana i tako olakšava korištenje aplikacije.



Slika 3.1. Korisničko sučelje (glavni prozor)

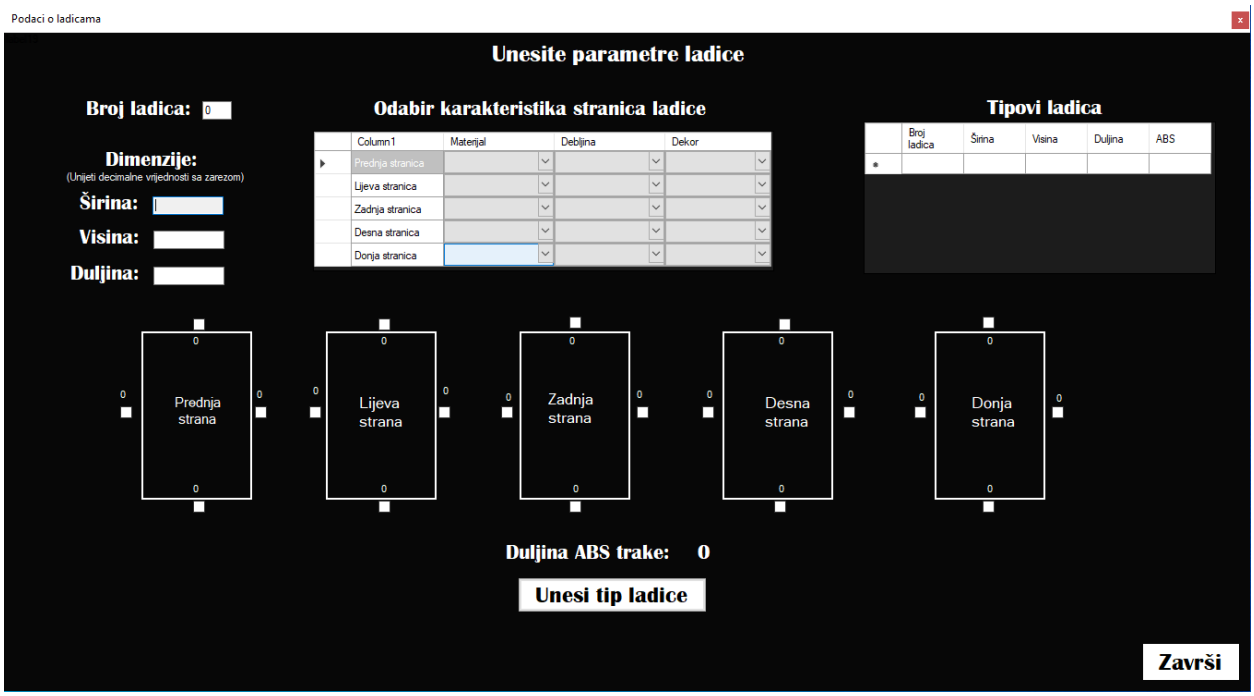
Glavni prozor podijeljen je na 3 logičke cjeline. Prva cjelina služi za odabir elementa koji se želi izraditi i klikom na njih otvara se prozor za izradu odabranog elementa. Elementi su korpus, ladica, i vrata. Druga cjelina predstavlja prostor gdje su navedeni osnovni podaci za određeni element. Informacije o elementu će se ispisati kada je element napravljen ili kada je odabran iz popisa elemenata. Također za svaki element postoji gumb koji omogućava brisanje dijela tog elementa. Treća cjelina se sastoji od tablice koja prikazuje sve dodane elemente. Element se može izbrisati iz tablice ili se može pregledati i urediti. Na kraju postoje gumbi za izračunavanje, brisanje i printanje konačnog popisa elemenata.

Na slici 2.2 je predstavljen prozor koji služi za unos parametara za izradu korpusa. Sve vrijednosti potrebne za izradu elementa unose se u prozor i ti se podaci spremaju za daljnju upotrebu tj. krajnji izračun. Unos svih parametara elemenata reguliran je te ako se unese pogrešna vrijednost aplikacija će izbaciti prozor koji će uputiti korisnika gdje je pogriješio u unosu. Npr. za unošenje broja manjeg od 0 za jednu od dimenzija prikazat će se prozor koji će obavijestiti korisnika o tome da dimenzije nisu dobro unesene. Ispod unosa parametara nalazi se 5 pravokutnika koji predstavljaju sve stranice elementa tj. rubove stranica koji se mogu kantirati. Korisnik prije unosa elementa treba označiti koje stranice će se kantirati te će se njihova duljina ubrojiti u duljinu ABS trake potrebne za kantiranje elementa. Svaka stranica korpusa može biti izrađena od drugačijeg materijala, debljine i dekora. Pri unosu parametara ponuđena je tablica koja prikazuje sve stranice korpusa te korisnik za svaku od njih mora unijeti materijal, debljinu te dekor od kojeg će biti izrađena ta stranica. To omogućava točan izračun količine materijala potrebnog za izradu korpusa.



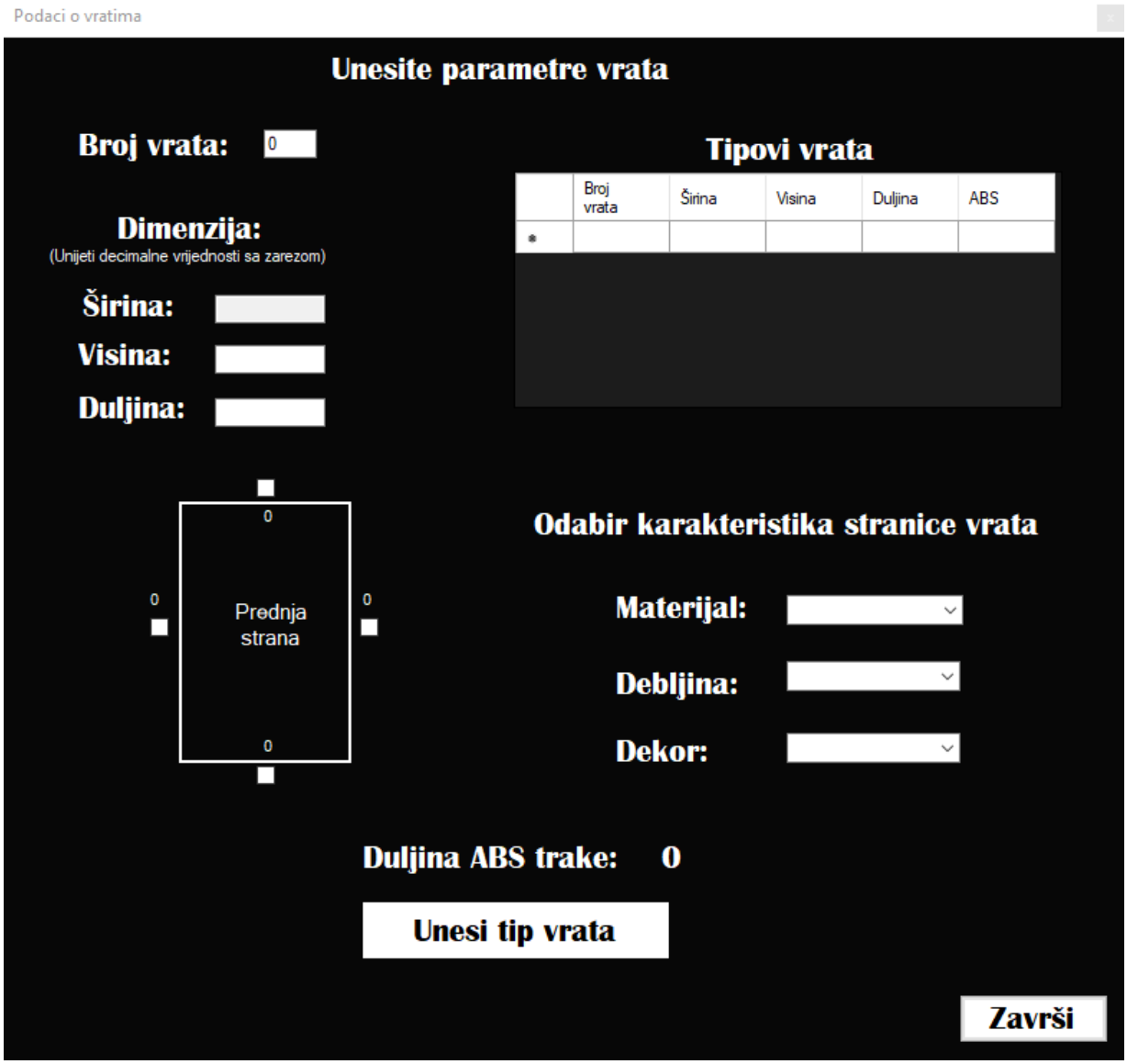
Slika 3.2. Prozor (sučelje) za unos parametara korpusa

Na slikama 3.3 i 3.4 prikazani su prozori za unos ladica i za unos vrata. Ta dva prozora su malo drugačija od prozora za unos korpusa jer postoji mogućnost unosa više tih elemenata različitih dimenzija dok korpus može biti samo jedan. Korisnik može napraviti koliko želi različitih ladica, a njihov broj upisuje u za to predviđeno područje. Za svaki tip ladice korisnik može izabrati dimenzije, rubove stranica koji će biti kantirani, vrstu i boju ABS trake te može za svaki tip ladice odabrati kako će izgledati svaka od stranica te ladice (materijal, debljina, dekor). Svi tipovi ladica s njihovim karakteristikama se ispisuju u posebnu tablicu. Uneseni tip ladice može se obrisati. Redovi tablice sadrže prazno polje u kojem će se pojaviti strelica ako klikom miša označimo bilo koje polje toga reda. Duplim klikom miša na prazno polje označeno strelicom ispred reda prikazuje se prozor s pitanjem o tome želi li se obrisati odabrani tip ladice. Ako je odabrana opcija „Da“ tada će se obrisati taj tip ladice(ako je bilo više ladica tog tipa sve će biti obrisane).



Slika 3.3. Prozor (sučelje) za unos parametara svih ladica

Prozor za izradu vrata razlikuje se od prozora za izradu ladica zbog toga što su vrata sačinjena od samo jedne stranice. Logika unosa vrijednosti je ista samo što kod vrata postoji unos karakteristika samo za jednu stranicu dok kod prozora s ladicama postoji tablica gdje se unose karakteristike svih stranica ladice. Također je omogućeno napraviti više tipova vrata koji će biti ispisani u tablici.



Slika 3.4. Prozor (sučelje) za unos parametara svih vrata

3.4 Struktura programskog koda

Kao što je već rečeno u jednom od ranijih poglavlja, C# je objektno-orijentiran jezik pa su tako svi mogući elementi aplikacije strukturirani u objekte tj. klase koje sadržavaju određene atribute koji te objekte opisuju i metode pomoću kojih objekti dobivaju neku funkcionalnost ili se nad njima vrši neka funkcionalnost. Svi elementi aplikacije strukturirani su kao vlastiti entiteti tj. objekti s obzirom na njihov kontekst unutar aplikacije. Tako su tijekom izrade nastali sljedeći objekti:

1) Korpus (engl. *Corpus*) – predstavlja korpus elementa koji u sebi sadrži atribute koji ga detaljnije opisuju (širina, duljina, visina, tip korpusa,tip ABS trake i materijal)

2) Ladica (engl. *Drawer*) – predstavlja jednu ladicu unutar elementa, unutar koda koristi se lista ladica jer ih može biti više. Svaki tip ladice može se drugačije kantirati i može biti izrađen od drugačijeg materijala
3) Vrata (engl. *Door*) – predstavlja jedna vrata elementa. Kao i kod ladica, vrata može biti više unutar jednog elementa pa se koristi lista pri rukovanju s vratima
4) Stranica (engl. *Page*) – predstavlja jednu stranicu elementa koji se izrađuje. U aplikaciji se radi sa stranicama korpusa, ladice i vrata. Svaka stranica ima svoju debljinu, materijal od kojega je izrađena te dekor. Ovaj objekt je potreban jer korisnik za svaku stranicu elementa može izabrati drugačiji materijal, debljinu i dekor
5) ABS traka (engl. *ABS Tape*) – predstavlja ABS traku elementa. ABS traka je poseban objekt jer sadrži podatke o tipu trake i o duljini trake pojedinih elemenata
6) Element – predstavlja konačni element koji se mora sastojati od korpusa i može imati ladice i vrata pa se tako unutar elementa nalaze objekti korpusa, ladice i vrata. Element ima i objekt ABS trake

Unutar koda postoji klasa koji se naziva *MathManagement* i koja sadrži funkcije za izvođenje matematičkih operacija nad elementima. To su matematičke operacije kao npr. izračun površine korpusa, ladica i vrata, izračun duljine ABS trake, ukupni izračun površine za sve elemente itd. Ova klasa je singleton [5], što znači da je samo jedna instanca ove klase stvorena tijekom cijelog izvođenja programa. To je učinkovito jer nije potrebno više različitih instanci ove klase zbog toga što klasa obavlja matematičke operacije koje se izvršavaju jednako za svaki element. Bez obzira na to što nije stvoreno više različitih instanci ove klase što je cilj objektno-orijentirane paradigme ovo je dobro rješenje zbog odvajanja koda u jedinstvenu cjelinu što je puno preglednije i lakše za modificirati. Primjer *singleton* klase pokazan je na slici 3.5

```
public class MathManagement
{
    private static MathManagement instance = null;
    1 reference
    private MathManagement(){}
    12 references
    public static MathManagement instanceOfMathManagement()
    {
        if (instance == null)
        {
            instance = new MathManagement();
        }
        return instance;
    }
```

Slika 3.5. *MathManagement* klasa

Singleton klasa ima privatni konstruktor i statičku metodu pomoću koje je dana mogućnost za stvaranje objekta ove klase jer on nije dostupan izvana. To je jedan nivo apstrakcije koji pruža „novi“ objekt klase dok se u pozadini ne stvara novi objekt već je dana postojeća instanca tog objekta. Objekt ove klase stvara se samo jednom i to pri prvom pozivu spomenute statičke metode te se sprema u static atribut tipa klase u kojoj se nalazi. Pri svakom drugom pozivu provjerit će se je li atributu pridružena vrijednost i ako je poslat će se ta vrijednost (objekt te klase) i neće doći do stvaranja novog objekta.

Zadnja klasa unutar koda koju je potrebno opisati je klasa *DataBase* koja predstavlja mjesto gdje se čuvaju podaci potrebni za konačni izračun. Zbog toga što je konačni cilj aplikacije ispis površine materijala neke debljine s određenim dekorom, podaci o materijalu, debljini, dekoru i površini se moraju čuvati i povezati. Korisniku je omogućen unos različitih vrijednosti za sve navedene parametre te postoji puno mogućih kombinacija za izradu elementa. Ova klasa omogućava spremanje i međusobno povezivanje tih podataka te rad s istim. Ova klasa je također realizirana kao *singleton* zbog toga što nije potrebno više različitih instanci ove klase. Klasa *DataBase* prikazana je na slici 3.6.

```
class DataBase
{
    private static List<object[]> dataBase;

    private static DataBase instance = null;
    1 reference
    private DataBase() { }
    5 references
    public static DataBase instanceOfDataBase()
    {
        if (instance == null)
        {
            instance = new DataBase();
            dataBase = new List<object[]>();
        }
        return instance;
    }
```

Slika 3.6. *DataBase* klasa

Klasa sadržava jedan atribut koji predstavlja listu u kojoj će biti čuvane moguće kombinacije za izradu elementa. Tijekom dodavanja elemenata klasa će računati površinu za svaku kombinaciju i omogućiti dohvaćanje te vrijednosti pri ispisu. Spomenuti atribut se inicijalizira pri prvom pozivu *instanceOfDataBase* funkcije tj. pri prvom pozivu konstruktora za *DataBase* klasu. Taj atribut bit će obrisan (postavljen na *null* vrijednost) tek kada je završena konačna izrada i kada je gotov konačni proračun.

3.5 Interakcija i prijenos podataka između prozora

U prošlom poglavlju opisani su prozori od kojih se sastoji aplikacija. Desktop aplikacija mora se sastojati od najmanje jednog prozora i princip izgradnje *Windows Form* .NET aplikacije su prozori preko kojih korisnik komunicira s programom. Ako postoji više prozora potrebno je ostvariti komunikaciju među njima jer je svaki prozor posebna klasa koja nasljeđuje klasu *Form*. Klasa *Form* prezentira jedan prozor aplikacije koji predstavlja korisničko sučelje aplikacije.

Klikom na sliku korpusa u glavnom prozoru otvara se novi prozor. Metoda za otvaranje novog prozora prikazana je na slici 3.7.

```
private void corpusPb_Click(object sender, EventArgs e)
{
    //this.Hide();
    CorpusWindow corpusWindow = new CorpusWindow(this);
    //corpusWindow.Closed += (s, args) => this.Show();
    corpusWindow.Show();
}
```

Slika 3.7. Metoda za otvaranje novog prozora

Konstruktor *CorpusWindow* klase tj. prozora koji omogućava unos podataka za izradu korpusa prima jedan parametar tipa MainWindow. Tako unutar klase *CorpusWindow* postoji instanca *MainWindow* objekta i pomoću te instance pristupa se public članovima i metodama *MainWindow* klase. Kada su unešeni svi potrebni podaci u predviđena polja i pritisnuto dugme „Unesi element“ pozvat će se metoda *getDataFromCorpus* iz klase *MainWindow* pomoću instance *mainWindow* koja je proslijeđena kroz konstruktor. Prije samog pozivanja te funkcije stvara se objekt klase *Corpus* koji predstavlja korpus elementa i u njemu se čuvaju svi podaci vezani uz korpus elementa. Taj objekt će kao parametar navedene funkcije biti prenesen u main window i tamo pohranjen te su tako podaci o korpusu iz jednog prozora preneseni u drugi prozor koji nema pristup tim podacima. Navedeni postupak nalazi se na slikama 3.8 i 3.9.

```
Corpus corpus = new Corpus(double.Parse(corpusWidthTb.Text),
                           double.Parse(corpusHeightTb.Text),
                           double.Parse(corpusLengthTb.Text),
                           corpusABSTapeTypeComboBox.Text,
                           corpusPages);

mainWindow.getDataFromCorpus(corpus);
```

Slika 3.8. Poziv metode za dohvaćanje podatak o korpusu iz drugog prozora

```
1 reference
public void getDataFromCorpus(Corpus corpus)
{
    corpusTypeLb.Text = corpus.getCorpusType();

    StringBuilder builder = new StringBuilder();

    builder.Append(corpus.getCorpusWidth()).Append(" x");
    builder.Append(corpus.getCorpusHeight()).Append(" x");
    builder.Append(corpus.getCorpusLength());

    corpusDimLb.Text = builder.ToString();

    corpusSurfaceLb.Text=MathManagement.instanceOfMathManagement().calculateCorpusSurface(corpus).ToString() + " cm^2";

    element.setElementCorpus(corpus);

}
```

Slika 3.9. Metoda za dohvaćanje podataka korpusa

Postupak prijenosa podataka isti je za sve prozore. U aplikaciji se komunikacija odvija između glavnog prozora i 3 pomoćna prozora. Pomoćni prozori međusobno ne komuniciraju, ali kada bi komunicirali postupak bi bio isti.

3.6 Matematika izračunavanja potrebnih vrijednosti

Glavni cilj ove aplikacije tj. završnog rada je omogućiti točno izračunavanje površine elemenata i njihovih dijelova, omogućiti izračunavanje ABS trake elemenata i dati uvid u sve izračunate podatke i elemente. Posao izračunavanja svih spomenutih vrijednosti prebačen je na *MathManagement* klasu koja je opisana u poglavlju 3.4. Metode te klase kao parametre primaju objekte za koje trebaju izračunati površinu predanog elementa ili duljinu ABS trake potrebnu za kantiranje predanog elementa. U nastavku su prikazane neke od metoda kako bi se dobio uvid u implementaciju funkcionalnosti za izračunavanje potrebnih vrijednosti.

Slika 3.10 prikazuje metodu za izračunavanje površine korpusa elementa. Kao parametar prima korpus koji u sebi sadrži sve njegove stranice. Svaka stranica ima svoje dimenzije te se ukupna površina dobiva zbrajanjem površina svih njegovih stranica.

```
5 references
public double calculateCorpusSurface(Corpus corpus)
{
    List<Page> corpusPages = corpus.getCorpusPages();
    double surface=0;
    for(int i = 0; i < corpusPages.Count; i++)
    {
        double pageSurface = corpusPages[i].getPageSideA() * corpusPages[i].getPageSideB();
        surface += pageSurface;
    }

    return surface;
}
```

Slika 3.10. Metoda za izračunavanje površine korpusa

Slika 3.11 prikazuje izračun površine za ladice. Za razliku od izračuna za korpus, gdje možemo imati samo stranice koje čine korpus, ovdje moramo proći kroz sve napravljene ladice i zbrojiti površinu svih stranica za svaku napravljenu ladicu. Tako je ukupna površina zbroj svih površina stranica svih ladica unutar elementa.

```
4 references
public double calculateDrawerSurface(List<Drawer> drawers)
{
    double surface=0;
    for(int i = 0; i < drawers.Count(); i++)
    {
        for (int j = 0; j < drawers[i].getDrawerPages().Count; j++)
        {
            surface += drawers[i].getDrawerPages()[j].getPageSideA() * drawers[i].getDrawerPages()[j].getPageSideB();
        }
    }
    MessageBox.Show("drawer" + surface);
    return surface;
}
```

Slika 3.11. Metoda za izračun površine ladica

Funkcija za izračun površine vrata ista je i kao funkcija za izračun površine ladica jer je i logika spremanja ta dva elementa jednaka.

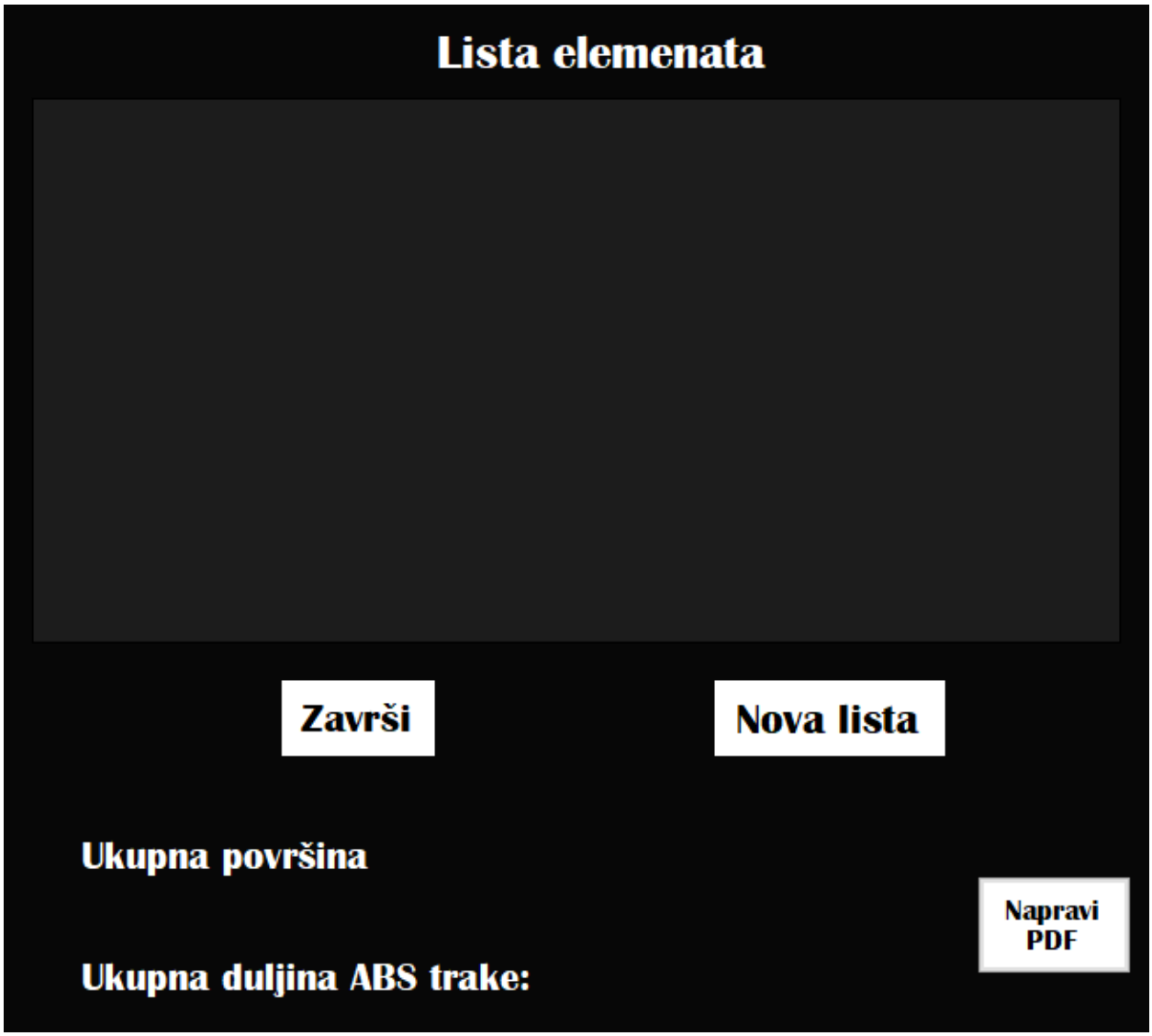
Slika 3.12 prikazuje pristup izračunavanju ukupne površine svih napravljenih elemenata. Ona kao parametar prima listu svih elemenata. Ona unutar sebe na svakom elementu poziva funkcije za izračun površine svih dijelova tog elementa. Što se tiče izračunavanja ukupne duljine ABS trake postupak je isti kao i kod izračuna ukupne površine samo što se nad svakim elementom pozivaju metode za dohvaćanje duljine ABS trake.

```
1 reference
public double calculateFinalSurface(List<Element> allElements)
{

    double surface=0;
    foreach(Element element in allElements)
    {
        surface += this.calculateCorpusSurface(element.getElementCorpus());

        if (element.getElementDrawers() != null)
        {
            surface += this.calculateDrawerSurface(element.getElementDrawers());
        }

        if (element.getElementDoors() != null)
        {
            surface += this.calculateDoorSurface(element.getElementDoors());
        }

    }
    return surface;
}
```

Slika 3.12. Metoda za izračunavanje ukupne površine svih elemenata

3.7 Konačne vrijednosti proračuna i ispis računa

Konačne vrijednosti proračuna dobivaju se klikom na dugme „Završi“. Klikom na dugme izračunavaju se ukupna površina svih elemenata te ukupna duljina ABS trake potrebna za kantiranje svih elemenata. Svi elementi za koje se računaju navedene vrijednosti navedeni su u listi elemenata. Pored konačnih vrijednosti nalazi se dugme „Napravi PDF“ koje daje ispis površine svakog materijala ovisno o njegovoj debljini i dekoru kojeg je potrebno utrošiti za izradu elemenata. Na slici 3.13 prikazano je područje liste elemenata te gumbovi za konačni izračun, izradu nove liste te za prikaz podataka u PDF obliku. Nakon pritiska na gumb „Napravi PDF“ generirat će se PDF dokument na radnoj površini.



Slika 3.13. Prikaz gumbova za završni izračun i izradu PDF dokumenta

U PDF formatu prikazane su površine svakog materijala ovisno o njegovoj debljini i dekoru. Svaka kombinacija prikazana je posebno. Površine su prikazane u metrima kvadratnima.

Ispis površina s obzirom na materijal, debljinu i dekor

Materijal:metal	Debljina: 18 mm	Dekor: hrast sonoma	Površina: 0,83191836 m^2
Materijal:iveral	Debljina: 18 mm	Dekor: hrast sonoma	Površina: 0,38099648 m^2
Materijal:drvo	Debljina: 12 mm	Dekor: hrast sonoma	Površina: 0,966329 m^2

Slika 3.14. Prikaz ispisa elemenata u PDF formatu

3.8 Konačni rezultat (aplikacija)

Kao rezultat rada dobivena je aplikacija koja ima funkcije računanja potrebnih parametara za izgradnju kuhinjskih ili kućanskih elemenata. Također ima svojstvo ispisa tih elemenata u PDF formatu. Aplikacija može biti korištena za sve kućanske elemente takvog tipa tj. oni koji su sačinjeni od korpusa, ladica i vrata. Ona može biti pokrenuta na svim računalima koji koriste Windows operacijski sustav i obavlja sve funkcionalnosti neovisno o korisniku ili nekom drugom vanjskom čimbeniku.

4. TESTIRANJE

Testiranje je postupak pokretanja softvera u kojemu je cilj otkriti moguće greške u programskom kodu ili uvjeravanje da programski kod radi ono što je od njega očekivano. Ako se greške pronađu, potrebno ih je ispraviti, a na taj se način osigurava kvaliteta softvera i olakšava postupak isporuke tog softvera u određenom vremenskom roku. Testiranje je danas jako razvijeno i ima puno različitih načina testiranja i nivoa testiranja. Tri nivoa testiranja su jedinično (engl. Unit testing), integracijsko i sistemsko testiranje. Pri testiranju ove aplikacije korišteno je jedinično testiranje koje je detaljnije opisano u sljedećem poglavlju. Testiranje je obavljeno u istom okruženju u kojem je razvijana sama aplikacija tj. u Visual Studio-u [9].

4.1 Jedinično testiranje (engl. Unit testing)

Jedinično testiranje je testiranje jednog dijela programskog koda. To može biti testiranje jedne klase ili jednog sučelja. Pri tom testiranju, testiraju se samo dijelovi testirane jedinice te se ne testira njihovo međusobno djelovanje ili cjelokupni kod. Jedinica koja se testira osim klase i sučelja može biti i jedna metoda. Za jednu jedinicu može postojati više unit testova koji će ispitati rad te jedinice s obzirom na različite ulazne podatke. U ovom konkretnom slučaju testirana je jedna jedinica tj. klasa koja se bavi matematikom izračunavanja svih vrijednosti koje aplikacija zahtjeva. Na slici 4.1 se nalazi testna metoda koja testira rad metode na slici 3.10.

```
[TestCase(22.234, 34.120, 45.674)]
0 references
public void calculateCorpusSurfaceTest(double width,double height,double length)
{
    //Arrange
    List<Page> pages = createPages(width, height, length);
    Corpus corpus = new Corpus(width,height,length,"2.0cm", pages);
    double expectedCorpusSurface = 5497.554072;
    //Act
    double actualCorpusSurface = mathManagement.calculateCorpusSurface(corpus);
    //Assert
    Assert.AreEqual(expectedCorpusSurface,actualCorpusSurface);
}
```

Slika 4.1. Testna metoda za izračunavanje površine korpusa

Metoda *calculateCorpusSurfaceTest* testira metodu *calculateCorpusSurface*. Jedinični test je podijeljen u 3 dijela koja su ispisana komentarima. U prvom dijelu (engl. Arrange) postavljaju se sve potrebne vrijednosti kako bi se moglo izvršiti testiranje. U ovom slučaju to je objekt korpusa na kojem će se pozvati testirana funkcija i očekivana vrijednost koju funkcija treba vratiti. Prije

kreiranja korpus objekta kreiraju se stranice korpusa zbog toga što su potrebne za stvaranje korpusa. U drugom dijelu (engl. Act) poziva se metoda koja se testira kako bi se dobio rezultat koji ona daje s obzirom na naše ulazne parametre (ulazni parametri nalaze se iznad funkcije (u ovom slučaju 22.234, 34.120 i 45.674). U trećem dijelu (engl. Assert) odvija se usporedba očekivanog rezultata i rezultata koji je vratila testirana funkcija. Kako bi test bio uspješan očekivana i dobivena vrijednost moraju se poklapati. Na slici 4.2 prikazana je još jedna metoda koja testira ukupnu površinu elementa.

```
[TestCase(22.234, 34.120, 45.674)]
0 references
public void calculateFinalCorpusSurfaceTest(double width, double height, double length)
{
    int elementsNum = random.Next(1, 5);
    List<Element> elements = new List<Element>() { };
    List<Page> pages = createPages(width, height, length);
    Corpus corpus = new Corpus(width, height, length, "2.0cm", pages);
    for (int i = 0; i < elementsNum; i++)
    {
        Element element = new Element();
        element.setElementCorpus(corpus);
        elements.Add(element);
    }


    double expectedSurface = 5497.554072 * elementsNum;

    double actualSurface = mathManagement.calculateFinalSurface(elements);

    Assert.AreEqual(expectedSurface, actualSurface, 0.0001);
}
```

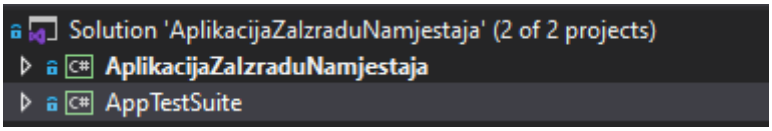
Slika 4.2. Testna metoda za ukupno izračunavanje površine korpusa

Metoda *calculateFinalCorpusSurfaceTest* i *calculateCorpusSurfaceTest* ne razlikuju se samo u metodi koju testiraju već i u različitijem kodu testa. Unutar *calculateFinalCorpusSurfaceTest* kako bi se izvršilo testiranje napravljena je lista elemenata koja će se napuniti objektima tipa *Corpus* zbog toga što metoda koja računa ukupnu površinu korpusa kao parametar prima listu svih elemenata koji su napravljeni.

4.2 Unit testiranje unutar Visual Studio-a

Programsko okruženje Visual Studio nudi osim pisanja koda i njegovo testiranje. U nastavku je opisan postupak kojim je aplikacija testirana unutar ovog okruženja. Unutar jednog *Solution-a* Visual Studio-a mogu se dodati biblioteke (engl. Class Library .NET Framework) koje čine

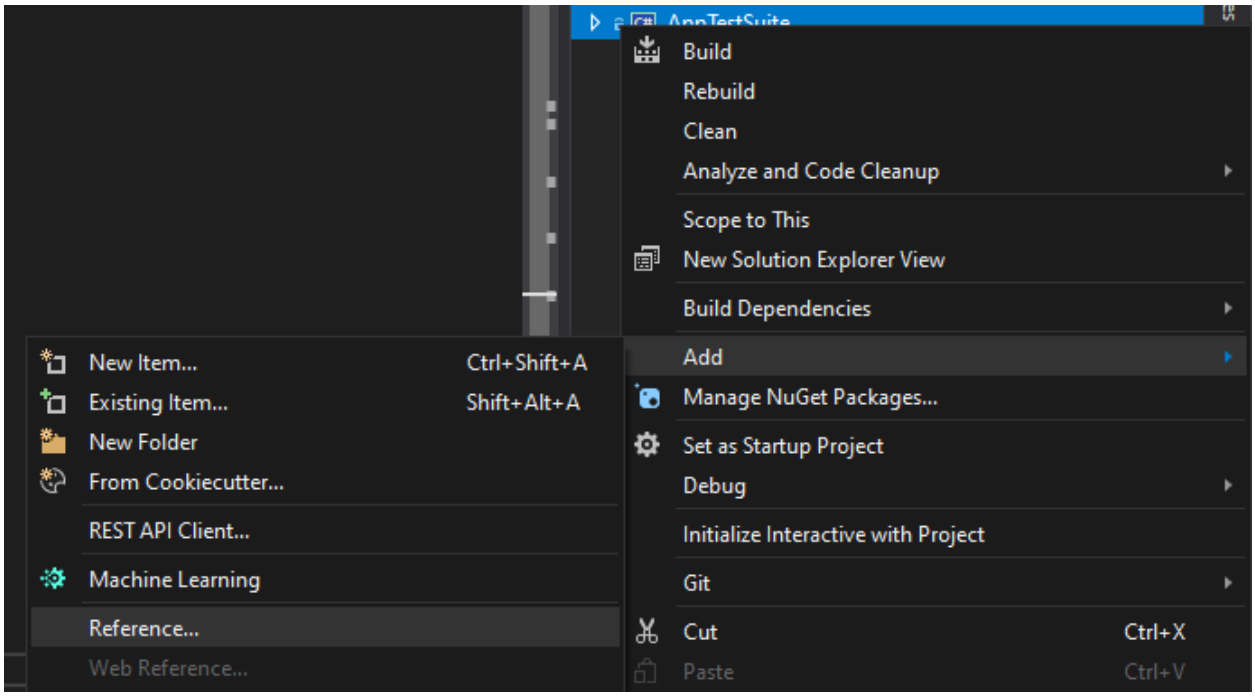
zasebne dijelove *Solution-a* u koji su dodane. Tako su u *Solution* za izradu ove aplikacije dodane dvije biblioteke, prva za pisanje koda aplikacije (AplikacijaZaIzraduNamjestaja) i druga za njegovo testiranje (engl. AppTestSuite). Opisana struktura prikazana je na slici 4.3.



Slika 4.3. Solution s bibliotekama

Nakon dodavanja biblioteke *AppTestSuite* potrebno je dodati ekstenziju (dodatak) pod nazivom *NUint* i *NUit3TestAdapter* koji omogućavaju testiranje za .NET jezike.

Nakon dodavanja navedenih ekstenzija za testiranje koda, jedino je još potrebno referencirati testnu biblioteku na biblioteku s kodom koji se testira. To se može tako da se desnim klikom klikne na testnu biblioteku te se pod izbornikom *Add* pronađe opcija Reference. Nakon toga pojavljuje se izbornik s popisom svih biblioteka u našem *Solutionu* gdje se odabire biblioteka koja se treba testirati. Nakon toga što se tiče okruženja sve je spremno za testiranje



Slika 4.4. Prikaz dodavanja reference na biblioteku

5. ZAKLJUČAK

Rad je napravljen u skladu s njegovim zahtjevima. Rezultat omogućuje izračunavanje parametara potrebnih za izradu namještaja i ispis traženih rezultata. Izrada desktop aplikacije olakšana je već opisanim okruženjem koje sadrži sve opcije i funkcionalnosti za izradu ovakvih aplikacija. Kada se traže složene matematičke operacije ili duge iscrpljujuće matematičke operacije lakše i učinkovitije je taj posao prepustiti računalu koje će posao obaviti brže i točnije. Tako se štedi vrijeme i proizvođač se može usredotočiti na druge poslove. Razvojem programskih okruženja olakšava se i sami postupak implementacije koda tj. proizvoda uz održavanje visoke kvalitete tih proizvoda. Sukladno tome razvija se i proizvodnja desktop aplikacija koja opet može olakšati i unaprijediti proizvodnju drugih grana proizvodnje kao što se u ovom slučaju s desktop aplikacijom olakšava okvirni izračun potrebnog materijala za izradu elemenata. Možemo zaključiti da je razvoj računalnih tehnologija vrlo bitan i da se njegovim razvojem potpomaže razvoju svih ostalih grana djelatnosti. U usporedbi s drugim programskim rješenjima istog tipa postoje elementi aplikacije koji bi mogli biti unaprijeđeni i oni koji bi mogli biti dodani. Ono što je možda najveća razlika između ovog rada i razvijenijih komercijaliziranih aplikacija je 3D prikaz svih elemenata i njihov međuodnos. *Windows Forms* nudi odlično okruženje za izradu desktop aplikacija ali nije specificiran za baratanje 3D objektima jer nije zamišljen kao takav, pa je 3D prikaz jako teško prikazati u tom okruženju. Druga stvar koju je moguće dodati je izrada svih elemenata po vlastitom ukusu kupca. Zadnja od najočiglednijih razlika između ostalih aplikacija i ovog rada je razvijenije korisničko sučelje. Komercijalizirane i bolje aplikacije nude opširnije i atraktivnije korisničko sučelje s velikim brojem mogućnosti. Što se tiče najuže funkcionalnosti i svrhe rada on ispunjuje sve ono što se od njega očekuje i proširenja mogu biti dodana ako postoji potreba za unaprjeđenjem tj. za nekakvom novom funkcionalnosti.

LITERATURA

[1] CORPUS Software, United Themes, 2021, dostupno na: https://corpus.hr/en/pages/, zadnji pristup 11.9.2021

[2] Master-Design Art-Shop X-Lite, Master Design, 2009, dostupno na: https://download.cnet.com/Master-Design-Art-Shop-X-Lite/3000-6677_4-10534771.html , zadnji pristup 11.9.2021

[3] Introduction to .NET, Microsoft, 2020, dostupno na: https://docs.microsoft.com/en-us/dotnet/core/introduction#cross-platform, zadnji pristup 20.8.2021.

[4] C# documentation, Microsoft, 2020, dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/, zadnji pristup 8.8.2021

[5] Implementing the singleton pattern in C#, Jon Skeet, 2006, dostupno na: https://csharpindepth.com/articles/singleton, zadnji pristup 11.9.2021

[6] Sastavljanje korpusa, PraviMajstor, 2011, dostupno na: https://pravimajstor.hr/sastavljanje- korpusa, zadnji pristup 20.7.2021

[7] J. Atwood, J. Spolsky, Stack overflow, Stack Exchange Network, 2008, dostupno na: https://stackoverflow.com/, zadnji pristup 2.9.2021

[8] J. Osivčić, Završni rad – Metode i metodologije u testiranju softvera, Travnik, 2019.

[9] B. Zorić, Prezentacija -Testiranje programske podrške, Osijek, 2019.

SAŽETAK

Naslov: Aplikacija za kalkulaciju troškova izrade namještaja

Tema ovog završnog rada je izrada desktop aplikacije koja će omogućiti izračunavanje troškova izrade namještaja. Ovaj rad tj. aplikacija pruža osnovne operacije koje su potrebne za izradu namještaja, a to su izračun površine materijala koji su potrebni za izradu jednog ili više elemenata i ispis tih podataka u PDF formatu. Osim površine materijala, aplikacija računa i duljinu ABS trake koja je potrebna za kantiranje elemenata. Glavni problem bila je sama izrada aplikacije koja će pružiti pouzdano izračunavanje površine materijala potrebnog za izradu namještaja. Prvo je trebalo osmisliti jednostavno korisničko sučelje kako bi korištenje aplikacije bile što lakše, razumljivije i intuitivnije. Nakon toga trebalo je implementirati računske operacije koje će pružiti korektan izračun i koje će izračunate vrijednosti prikazati na sučelju. Na kraju procesa pružena je mogućnost spremanja svih izračunatih vrijednosti u .pdf formatu koje je tada moguće isprintati.

Ključne riječi: desktop aplikacija, kalkulacija troškova izrade namještaja, C# programski jezik, kantiranje namještaja, ispis površine materijala

ABSTRACT

Title: Furniture production cost application

This topic of this thesis is the development of a desktop application which provides furniture production cost for the user. Some of the basic functionalities included in the application are calculation of material surface that is needed for production of an element as well as the length of ABS tape that is needed for edging the element. The main problem was application development and implementation of reliable application functions which provide correct output values (furniture production cost) depending on user input. The first problem in application development was user interface implementation. The user interface needed to be intuitive and understandable for users. The second problem was implementation of functions that will provide correct output values for different user inputs. The last part of application development was providing the user with the possibility to print the calculated data.

Keywords: desktop application, furniture production cost, C# programming language, furniture edging, material surface printout