

Univerzitet u Sarajevu

Elektrotehnički fakultet Sarajevo

Odsjek za računarstvo i informatiku

# Programska podrška radu sa dokumentima matematičkog sadržaja

---

*Završni rad*

Mentor: Vanr. prof. dr Đonko Dženana

Student: Hidić Adnan

Sarajevo, juli 2013.

**Nastavnik:** Vanr. prof. dr. Đonko Dženana

**Teme za završne radove 1. ciklusa za 2012/2013 studijsku godinu**

**Tema :** Programska podrška radu sa dokumentima matematičkog sadržaja

**Kurs:** Razvoj programskih rješenja

**Student:** Hidić Adnan

**Opis:**

Korisnici desktop računara suočavaju se sa određenim poteškoćama onda kada im je potreban brz i intuitivan unos matematičkih formula, izraza ili određenih karakterističnih simbola. Poteškoće se najčešće ogledaju u činjenici da se korisnici moraju prilagoditi specifičnim programima: equation/formula editorima (uređivačima), ili dodacima editorima teksta koji pružaju potrebne funkcionalnosti. Često je potrebno poznavati posebnu sintaksu u okviru okruženja jer se unos teksta i unos matematičkih formula konceptualno razlikuju. No, za krajnjeg korisnika polazna premissa jeste da se oboje na isti način pišu na papir, pa bi stoga bilo prirodno da se po jednakom principu unose i na računaru. Moderni uređaji sa ekranima osjetljivim na dodir i sa mogućnostima prepoznavanja rukopisa prevazilaze navedeno ograničenje, no ono i dalje važi na desktop računarima. Razvoj desktop aplikacije koja je jednostavna za upotrebu a pruža funkcionalnost intuitivnog unosa matematičkih jednačina, njihovog prikaza i obavljanja prostih izračuna osnovni je cilj ovog završnog rada.

**Ciljevi:**

- Analizirati i specificirati funkcionalne i nefunkcionalne zahtjeve programskog rješenja za podršku radu sa dokumentima matematičkog sadržaja
- Dati pregled postojećih programskih rješenja i usporediti ih
- Dati pregled modela i načina na koji postojeća programska rješenja funkcioniraju
- Razviti vlastiti model i dizajnirati odgovarajuće programsko rješenje
- Implementirati i testirati programsko rješenje i dokumentirati specifičnosti vezane uz korištene tehnologije i programske modele

**Okvirni sadržaj:**

- Uvod - U okviru uvodnog poglavlja dati će se uvod u temu i motivacija za izbor teme, ciljevi rada, kao i struktura rada.
- Opis problema i postojeća rješenja - U okviru ovog poglavlja dati će se opis problema i specificirati će se funkcionalni zahtjevi koje aplikacija treba ispunjavati kako bi se problem riješio. Komparirati će se već postojeća rješenja, njihove osobine, prednosti i nedostaci.
- Dizajn rješenja - U okviru ovog poglavlja opisati će se korištena metodologija analize i dizajna, a na osnovu koje se modelira i dizajnira aplikacija koja će ispuniti zahtjeve postavljene u prethodnom poglavlju. Dati će se i pregled alternativnih modela i obrazložiti će se njihove prednosti i nedostaci.

- Implementacija i testiranje aplikacije - U okviru ovog poglavlja odabrati će se i opisati tehnologije pomoću kojih se aplikacija implementira. Sukladno postavljenom modelu, upotrebom odabranih tehnologija i primjenom odabranih metodologija, opisati će se proces razvoja desktop i web dijela aplikacije. Dokumentirati će se i specifičnosti vezane uz korištene tehnologije. Također, ukratko će biti opisan način na koji je aplikacija testirana, kao i način na koji su ispravljeni eventualni propusti u inicijalnom dizajnu i implementaciji.
- Zaključak- U okviru ovog poglavlja dati će se rezime urađenog i prezentirati će se osvrt na rad, odnosno kreiranu aplikaciju. Dati će se smjernice za budući razvoj aplikacije.

#### **Očekivani rezultati:**

- Specificirani funkcionalni i nefunkcionalni zahtjevi programskog rješenja za podršku radu sa dokumentima matematičkog sadržaja.
- Uspoređena postojeća programska rješenja.
- Uspoređeni modeli postojećih programskih rješenja.
- Razvijen vlastiti model i dizajn programskog rješenja.
- Razvijena odgovarajuća aplikacija.
- Implementirano i testirano programsko rješenje i dokumentirane specifičnosti korištenih tehnologija, komponenti i programskih modela.

#### **Polazna literatura:**

- 1) Željko Jurić, Diskretna matematika za studente tehničkih nauka, Elektrotehnički fakultet u Sarajevu, 2011
- 2) Dženana Đonko, Samir Omanović, Objektno orijentirana analiza i dizajn primjenom UML notacije, Elektrotehnički fakultet u Sarajevu, 2010
- 3) Pavi Sandhu Cengage, The MathML Handbook, Charles River Media, 2003

Mentor

---

Vanr. prof. dr Đonko Dženana

## Sažetak

Korisnici desktop računara suočavaju se sa određenim poteškoćama onda kada im je potreban brz i intuitivan unos matematičkih formula, izraza ili određenih karakterističnih simbola. Poteškoće se najčešće ogledaju u činjenici da se korisnici moraju prilagoditi specifičnim programima: equation/formula editorima (uređivačima), ili dodacima editorima teksta koji pružaju potrebne funkcionalnosti. Često je potrebno poznavati posebnu sintaksu u okviru okruženja jer se unos teksta i unos matematičkih formula konceptualno razlikuju. No, za krajnjeg korisnika polazna premisa jeste da se oboje na isti način pišu na papir, pa bi stoga bilo prirodno da se po jednakom principu unose i na računaru. Moderni uređaji sa ekranima osjetljivim na dodir i sa mogućnostima prepoznavanja rukopisa prevazilaze navedeno ograničenje, no ono i dalje važi na desktop računarima. Razvoj desktop aplikacije koja je jednostavna za upotrebu a pruža funkcionalnost intuitivnog unosa matematičkih jednačina, njihovog prikaza i obavljanja prostih izračuna osnovni je cilj ovog završnog rada.

## Abstract

Desktop computer users are faced with certain difficulties when in need for a fast and intuitive mechanism to input mathematical formulae, expressions or certain special symbols outside the normal alphabet in a computer programme. Those difficulties mostly come from the fact that users need to adapt to certain programmes, "equation/formula editors", or certain addons for text editing software that provide that functionality. Users also need to learn programme-specific syntax because input of maths and input of text are conceptually different. However, end-users' perception is that one can write both plaintext and maths the same way on paper, so it would be only natural to use the same principle when inputting math in a computer programme. Modern touchscreen devices equipped with handwriting recognition abilities overcome formerly stated disadvantages, however, they do remain an issue for desktop users. Primary focus of this paper is to develop an application that is both user-friendly and capable to cope with the problem of simple math input in a desktop computer, its rendering and performing simple calculations.

## 1. Uvod

Problem kojeg se obrađuje u okviru ovog završnog rada jeste problem kreiranja elektronskog zapisa matematičkih izraza i formula, odnosno, u širem smislu, pravljenje dokumenata primarno matematičkog sadržaja. Iako se navedeni problem može relativno jednostavno riješiti (i rješava se) upotrebom nekog paketa uredskih alata poput *Microsoft Office Word*a kroz njegov integrirani *Equation editor*, ili upotrebom *LaTeX*-a ili sličnih *markup* jezika koji omogućavaju posredni *rendering* napisanih matematičkih izraza, oni svi imaju svoje određene nedostatke.

Naime, za upotrebu profesionalnog rješenja poput *MS Office* najčešće je potrebno nabaviti skup (uredskih) alata čiji je *Equation editor* samo jedan dio i nije u fokusu imajući u vidu generičku prirodu takvih proizvoda. S druge strane, *LaTeX* i *markup jezici* zahtijevaju upotrebu posebnih alata, ali i učenje odgovarajućih jezika kako bi ih se moglo koristiti na zadovoljavajući način. U konačnici se problem svodi na to da je formule i matematičke izraze teško "unositi u računar" kao što se to radi sa *običnim* tekstom: bilo da je nužno naučiti sintaksu jezika (bez obzira na mogućnosti koje on pruža, npr. *LaTeX*), bilo da je nužno koristiti gotove grafičke elemente koji se onda "slažu" u formulu i popunjavaju sadržajem što je vizuelno odgovarajuće, ali relativno sporo i najčešće neintuitivno riješeno. Pri tome je potrebno koristiti složeni alat za (na prvi pogled) jednostavan zadatak (primjer *MS Word*a).

Ono što se uradilo kroz ovaj završni rad jeste kreiranje modela i odgovarajućeg tehnološkog demonstratora (aplikacije) koja omogućava što je moguće prirodniji način unosa formula i kreiranja jednostavnih matematičkih dokumenata, ne odveć bogatog opcijama i funkcionalnostima koje su odlika složenijih sistema, ali zato jednostavnog i intuitivnog za korištenje. Pri tome se diskutuje šta je to za korisnika prirodno i jednostavno kod korištenja jedne ovakve aplikacije.

## 1.1. Ciljevi rada

Ciljevi rada su:

- Analizirati i specificirati funkcionalne i nefunkcionalne zahtjeve programskog rješenja za podršku radu sa dokumentima matematičkog sadržaja.
- Dati pregled postojećih programskih rješenja i usporediti ih.
- Dati pregled modela i načina na koji postojeća programska rješenja funkcioniraju.
- Razviti vlastiti model i dizajnirati odgovarajuće programsko rješenje.
- Implementirati i testirati programsko rješenje i dokumentirati specifičnosti korištenih tehnologija, komponenti i programskih modela.

## 1.2. Struktura rada

Rad se sastoji iz pet poglavlja, pri čemu je prvo poglavlje uvodno i u njemu je dat uvod u temu, ciljevi i struktura rada. Ostala četiri poglavlja su:

- **Opis problema i postojeća rješenja-** U okviru ovog poglavlja daje se opis problema i specificiraju se funkcionalni zahtjevi koje aplikacija treba ispunjavati kako bi se problem programski riješio. Kompariraju se već postojeća rješenja, njihove osobine, prednosti i nedostaci.
- **Dizajn rješenja-** U okviru ovog poglavlja opisuje se korištena metodologija analize i dizajna, a na osnovu koje se modelira i dizajnira aplikacija koja će ispuniti zahtjeve postavljene u prethodnom poglavlju. Daje se i pregled alternativnih modela i obrazlažu se njihove prednosti i nedostaci.
- **Implementacija i testiranje aplikacije** - U okviru ovog poglavlja opisane su tehnologije pomoću kojih se aplikacija implementira. Sukladno postavljenom modelu, upotrebom odabranih tehnologija i primjenom odabranih metodologija, opisuje se proces razvoja desktop i web dijela aplikacije. Navode se i specifičnosti vezane uz korištene tehnologije. Također, ukratko je opisan način na koji je aplikacija testirana, kao i način na koji su ispravljeni eventualni propusti u inicijalnom dizajnu i implementaciji.
- **Zaključak-** U okviru ovog poglavlja daje se rezime urađenog i prezentira se osvrt na rad, odnosno kreiranu aplikaciju. Daju se smjernice za budući razvoj aplikacije.

Napomena: rad se sastoji i iz spiska korištenih referenci i vanjskih resursa, uputa za korištenje aplikacije i definicija tuđica i akronima u prilogu.

*Nakošenim slovima* su pisane riječi: koje se posebno naglašava u trenutnom kontekstu, koje predstavljaju tuđice bez adekvatnog prijevoda na službene jezike u BiH, čiji prijevod postoji ali je opisne naravi i zbog toga je nepraktičan za stalnu upotrebu. Font korišten za pisanje rada je Cambria, veličina 11. Naslovi poglavlja pisani su veličinom fonta 13, a naslovi poglavlja i odjeljaka su **podebljani**. Nazivi programskih komponenti i metoda pisani su fontom Segoe UI Light, veličina 10.

## 2. Opis problema i postojeća rješenja

U okviru ovog poglavlja daje se opis problema i specificiraju se funkcionalni zahtjevi koje aplikacija treba ispunjavati kako bi se problem riješio. Uspoređuju se postojeća rješenja za rad sa dokumentima matematičkog sadržaja, njihove osobine, prednosti i nedostaci.

### 2.1. Opis problema

Primarna namjena aplikacije koja se razvija jeste omogućavanje brzog "utipkavanja" matematičkih zadataka i njihovih rješenja u tzv. *light*- perolake matematičke dokumente kao i njihove razmjene između korisnika. Često je autor ovog rada želio tokom *chata* brzo i jednostavno otipkati rješenje zadatka svojim kolegama (a i oni njemu), i uz to pokazati šta je gdje radio tokom rješavanja. Pored prisustva ranije opisanih alata, na kraju je rješenje bilo solomonsko: korišten je ili *MS Paint* da se povlačenjem miša "napiše", odnosno, "nacrtá" zadatak i njegova rješenja, ili je zadatak rađen na papiru pa potom skeniran ili uslikan, ili je korištena npr. integrirana funkcionalnost slikovnih poruka, što je bila funkcionalnost starijih verzija *Windows Live Messengera* kao aplikacije za chat. Treće rješenje jeste pisanje *plaintexta*, odnosno, običnog teksta (kao što je recimo tekst kojeg trenutno čitate) iz kojeg su kolege trebali protumačiti šta im se to tačno šalje (primjerice, "integral od 0 do pi od  $2(x+5)dx$ " - a što bi trebalo biti ekvivalentno sa  $\int_0^\pi 2(x+5)dx$ ).

Dakle, primarni problem ne leži u nedostatku već gotovih rješenja, već u činjenici da korisnici desktop računara podrazumijevaju da za jednostavne stvari (što za korisnika kreiranje dokumenta sa par formula i jeste) moraju postojati jednostavna programska rješenja, bez puno pravila, ograničenja i nevezanih funkcionalnosti. Takva programska rješenja najčešće se nazivaju *apps* (skraćeno od engl. *applications*, u smislu jednostavnih aplikacija).

Rješavanje zadataka i dijeljenje par formula problem je koji je manji po opsegu i djeluje značajno jednostavniji od pravljenja sadržajem bogatog dokumenta ili formalnog matematičkog sadržaja kojeg je onda potrebno *renderirati*.

Dobar primjer *aplikacije* koja je ujedno i *app* u kontekstu koji je ranije naveden jeste Microsoftov *Notepad*- jednostavan program prisutan i integriran u sve verzije operativnog sistema *Windows*, koji korisnicima omogućava upravo ono što mu ime i govori- brz unos teksta, za kojeg nije bitno da li je formatiran, ima li margine, da li je moguće unositi tabele i slično. Zauzvrat, bez puno truda moguće unijeti tekst, spasiti ga i poslati nekome. Bitniji je sadržaj (tzv. *plaintext*) od formatiranja i bogatstva opcijama (moguće je promijeniti globalno font, veličinu fonta, uključiti word wrapping i pretraživati tekst).

Aplikacija koju se razvija u okviru ovog završnog rada po prirodi je veoma slična *Notepadu*- ali koji podržava i unos matematičkih izraza i njihov odgovarajući prikaz, kao i još neke funkcije specifične za područje primjene (npr. izračunavanje nekih jednostavnijih izraza). Štaviše, sâm način unosa izraza je prilagođen načinu korištenja i izgledu (engl. *look-and-feel*) *Notepada*- bez kompleksnih pravila i mnoštva posebnih kontrola, iz *plaintexta* se jednostavno dobija odgovarajuće vizuelne reprezentacije unesenih izraza. Također, korisnici mogu automatski pretvarati jedne u druge nizove simbola uz potvrdu. Primjerice, pi u  $\pi$ .

Dodatno je omogućena jednostavan način razmjene kreiranih sadržaja putem weba po principu sličnom onom za razmjenu slika (*imgur.com*) ili snimaka ekrana (*pokit.org*).

## 2.2. Specifikacija funkcionalnosti aplikacije

Na osnovu prethodno opisanog problema moguće je zaključiti da aplikacija koju se razvija mora zadovoljavati sljedeće funkcionalne zahtjeve:

- 1) ostvariti *look-and-feel* i dati neke od funkcionalnosti jednostavnog editora teksta (kao što su recimo *Microsoft Notepad* ili *Gedit*), što podrazumijeva (pored podrazumijevanog unosa običnog, neformatiranog teksta) i sljedeće:
  - *undo* i *redo* funkcije
  - otvaranje i spašavanje dokumenata sa i na lokalni računar
  - *find & replace* funkcionalnost
- 2) omogućiti intuitivan unos matematičkih formula/jednačina/izraza
- 3) omogućiti odgovarajući prikaz (*rendering*) unesenih matematičkih formula/ jednačina/ izraza
- 4) omogućiti evaluiranje jednostavnih matematičkih izraza u okviru dokumenta
- 5) omogućiti crtanje grafika funkcija jedne promjenjive
- 6) omogućiti funkcionalnosti specifične za potrebe jednostavnog uređivača matematičkih dokumenata:
  - ubacivanje posebnih simbola (npr.  $\pi$ ) na osnovu kratica tj. *autocomplete* funkcionalnost
  - *upload* i *download* dokumenata na i sa servera
  - konfiguriranje programa

Time je zaokružena identifikacija konkretnih funkcionalnosti koje aplikacija implementira.

Pored definicije funkcionalnih zahtjeva, definiraju se i nefunkcionalni zahtjevi, odnosno atributi kvaliteta razvijane aplikacije. Unos i uređivanje teksta trebaju biti zadovoljavajuće brzi za sve realno moguće brzine kucanja teksta. Pri tome je najveća brzina kucanja u prosjeku nešto manja od 10 slova u sekundi, iz čega slijedi da vrijeme odziva pri unosu treba biti manje od 120 milisekundi [1][2]. Funkcije specifične za unos matematičkih izraza trebaju imati vrijeme odziva manje od 1.5 sekundu da bi se održao dojam rada u stvarnom vremenu. Funkcije specifične za *rendering* kreiranog dokumenta trebaju imati vrijeme odziva koji će biti očekivano za korisnika, pri tome se za potrebe ovog rada smatra da je to manje od 3 sekunde. Navedeni zahtjevi odnose se na, za korisnika očekivano velike dokumente. Pri tome se za potrebe ovog rada smatra da su to dokumenti koji sadrže od 3 do 5 hiljada simbola. Konkretna način ostvarivanja funkcionalnosti detaljnije je razrađen i diskutiran u poglavlju 3.



### 2.3. Postojeća rješenja

Autor ovog završnog rada ni u kojem slučaju ne pretenduje napraviti nešto u potpunosti novo: postoji veliki broj uređivača običnog teksta koji svoje funkcije ostvaruju na, za mnoge korisnike, sasvim zadovoljavajući način (npr. *Notepad++*), kao i uređivača za kreiranje stiliziranih naučnih dokumenata pomoću LaTeX-a (npr. *LaTeX Editor*, *WinEdt* i sl.) [3][4][5]. Također, za kreiranje dodatnim sadržajima (slikama, tabelama, referencama, komentarima, crtežima) bogatih dokumenata zasigurno će se koristiti *MS Word* ili npr. besplatni *LibreOffice Writer* [6][7]. Za izračune je moguće koristiti funkcije ugrađenog kalkulatora koji dolazi instaliran uz operativni sistem (bilo Windows, bilo Linux), *Matlab*, *Mathematica*, ili npr. usluge *Wolfram Alpha* tražilice putem weba [8][9][10]. Čak i standardni *Google* pretraživač nudi funkcije izračuna matematičkih izraza nakon što se isti utipkaju u polje tražilice [11]. Po specificiranim funkcionalnostima (a u odnosu na navedeno u 2.2) možda najbliža aplikacija koju je autor ovog rada pronašao jeste *Expresso*, ruskog proizvođača *DeepIT* i namijenjena za *Appleov iOS* operativni sistem [12]. Navedena aplikacija je jednostavni uređivač teksta koji omogućava unos i evaluaciju matematičkih izraza, ali uz određena ograničenja. Za ozbiljne primjene svakako će se iskoristiti kombinacija bilo kojeg od navedenih (ili brojnih drugih postojećih) rješenja koja su dokazana u praksi i koja u kombinaciji (ili samostalno, upotrebom naprednih opcija ili posebnih proširenja) omogućavaju sve što je korisniku potrebno, jasno, uz ulaganje određenog napora.

U suštini, čak i letimična analiza odgovarajućih programskih rješenja pokazuje da postoje dva osnovna i široko rasprostranjena pristupa integraciji matematičkih sadržaja u tekstualne dokumente:

- 1) Upotreba *markup* jezika poput LaTeX-a
- 2) Integracija matematičkih objekata kao posebnih objekata u bogate formate dokumenata

Alternativa navedenim pristupima su raznovrsne improvizacije koje se i ne mogu smatrati sistematičnim rješenjem koje se traži: uređivač *plaintexta* sa mogućnosti odgovarajućeg vizuelnog prikaza (*renderinga*) i jednostavnog računanja u samom dokumentu.

U nastavku se razmatraju osnovna dva navedena pristupa, njihove prednosti i nedostaci, a navode se i primjeri postojećih programskih rješenja koja te pristupe koriste.

### 2.3.1. Upotreba markup jezika

Sâm pojam *markup language* nije jednostavno prevesti na službene jezike u BiH. U suštini označava jezik u kojem se kroz komunikacioni medij prenose ne samo podaci ili sadržaj, već i prezentacijska ili neka druga *uređivačka logika* koja organizuje ili se odnosi na taj sadržaj, a sama je sačinjena od istih simbola koji su konvencijom organizirani tako da ih se može razlikovati od sadržaja. Nešto kraća definicija od prethodne opisne definicije bi bila da se radi o jeziku koji kombinira podatke sa metapodacima- gdje je metapodatak podatak o podatku.

Primjerice, ukoliko razmatramo tekst "*Pozdrav!*" i na određeni način se želi prenijeti informacija da se pozdrav izgovara sporo (recimo uputa glumcu), onda bi se to moglo zabilježiti na sljedeći način "*<sporo> Pozdrav! </sporo>*". Sasvim namjerno je odabrana notacija identična onoj za najpoznatiji markup jezik- XML (skraćenica za *eXtensible Markup Language*). Glumac iz primjera će u svojoj skripti imati napisan cijeli navedeni niz znakova, no ako je ustanovljena odgovarajuća konvencija, "*<sporo>*" i "*</sporo>*" neće izgovoriti, već će govor prilagoditi metapodatku (naredbi) da navedeni tekst izgovara sporo.

Na osnovu datog opisa pojma *markup jezika*, načelno je jasna priroda uređivača teksta koji se zasnivaju na upotrebi nekog *markup* jezika. Pri tome je najpoznatiji *markup* jezik za uređivanje dokumenata jezik TeX kojeg je razvio Donald Knuth 1978. godine. TeX danas predstavlja *de facto* standardni tipografski sistem u kojem je moguće kreirati složene matematičke formule, ali i koristiti raznovrsne simbole i formatiranja specifična za inženjerstvo, hemiju, fiziku, statistiku, matematiku i mnoge druge oblasti. [13]

Pored TeX-a i njegovog najpoznatijeg formata LaTeX (razvio ga je Leslie Lamport 1980. godine), još jedan od *markup* jezika jeste i MathML (*Mathematical Markup Language*) koji predstavlja složen sistem (mada manje moćan od LaTeX-a koji je namijenjen za kreiranje cijelih dokumenata) za pohranu kako načina prezentacije, tako i smisla napisane formule [14]. MathML je dio HTML5 standarda.

Način rada sa LaTeX-om i MathML-om je jednostavan: korisnik će ili ručno ili pomoću nekog za to specijaliziranog alata kreirati dokument odgovarajućeg sadržaja, a potom će se posebni softver, *renderer*, pobrinuti za to da se napisani *markup* kôd korektno interpretira i na odgovarajući način prikaže korisniku. Ukoliko korisnik uoči određene nedostatke, promjene mora napraviti na dokumentu na osnovu kojeg je obavljeno *renderiranje*, budući da sadržaj, jednom *renderiran*, prestaje biti interaktivan za korisnika, tj. moguće ga je samo pregledati.

Već i iz do sada izloženoga, jasno je da su *markup* jezici izuzetno moćni, ali i izuzetno zahtjevni. Pri tome se konkretno misli na sljedeće:

- 1) Korisnik mora naučiti sintaksu jezika. Iako se to u slučaju LaTeX-a isplati na duže staze jer time korisnik stječe potpunu kontrolu nad dokumentima koje definira do u detalj prikaza, to je zahtjevno i neprikladno za jednostavne, neprofesionalne zadaće koje ne zahtijevaju prolazak kroz cijeli proces učenja novog jezika, da bi se potom kreirali i praktično "kompajlirali" dokumenti u formu u kojoj ih je moguće prezentirati.
- 2) Ne postoje *WYSIWYG* (*What You See Is What You Get*) uređivači za TeX derivate budući da su u suprotnosti sa njegovim temeljnim postulatom: "kompajliranje" izvornog kôda u izlazni format [15]. Npr. *MS Word* je WYSIWYG uređivač u smislu da korisnik uči kako kroz program, primjerice, podebljati tekst, ali ne mora znati kako se to interno pohranjuje u *Wordovom* formatu (.docx ili .doc). Rezultat podebljavanja teksta će biti prikazan podebljan tekst na ekranu, odštampan podebljan tekst na papiru, tj. rezultat je uniforman i jednak onome kojeg vidimo u uređivaču. Alternative su *WYSIWYM* (*What You See Is What You Mean*) TeX uređivači poput LyX-a koji daju *otprilike* ono što će biti na izlazu TeX kompajlera, no ne u potpunosti isto budući da se ne koriste isti prezentacijski principi [16]. Dodatno, alternativu predstavlja i rješenje u kojem se unosi TeX kôd, a istovremeno se sa izmjenama u odvojenom radnom prostoru prikazuje dokument onakav kakav jeste nakon kompajliranja. Ovo rješenje ima očito loše performanse (u smislu da se kontinuirano vrši rekompajliranje TeX kôda), no prednost je to što se greške i nedostaci brže uočavaju nego da se prvo napiše dokument, pa tek onda isti pregleda.
- 3) Za MathML situacija je nešto drugačija. Budući da se radi o jeziku namijenjenom za upotrebu u sklopu web dokumenata, očigledno je sasvim pogodno koristiti neki od postojećih alata (primjerice *MathFlow*) koji implementiraju vizuelni/grafički pristup kreiranju matematičkih formula/jednačina/izraza kroz "slaganje" blokova (grafičkih kontrola) iz kojih se onda generira MathML kôd kojeg je moguće ugraditi u npr. HTML5 dokument [17].

Navedeno je moguće vidjeti kroz jednostavan primjer načina zapisa rješenja kvadratne jednačine kroz LaTeX i MathML [18]:

Markup jezik	Markup
LaTeX	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
MathML	<pre> &lt;math mode="display" xmlns="http://www.w3.org/1998/Math/MathML"&gt;   &lt;mrow&gt;     &lt;mi&gt;x&lt;/mi&gt;     &lt;mo&gt;=&lt;/mo&gt;     &lt;mfrac&gt;       &lt;mrow&gt;         &lt;mo form="prefix"&gt;#x2212;&lt;!-- - --&gt;&lt;/mo&gt;         &lt;mi&gt;b&lt;/mi&gt;         &lt;mo&gt;#x00B1;&lt;!-- &amp;PlusMinus; --&gt;&lt;/mo&gt;         &lt;msqrt&gt;           &lt;msup&gt;             &lt;mi&gt;b&lt;/mi&gt;             &lt;mn&gt;2&lt;/mn&gt;           &lt;/msup&gt;           &lt;mo&gt;#x2212;&lt;!-- - --&gt;&lt;/mo&gt;           &lt;mn&gt;4&lt;/mn&gt;           &lt;mo&gt;#x2062;&lt;!-- &amp;InvisibleTimes; --&gt;&lt;/mo&gt;           &lt;mi&gt;a&lt;/mi&gt;           &lt;mo&gt;#x2062;&lt;!-- &amp;InvisibleTimes; --&gt;&lt;/mo&gt;           &lt;mi&gt;c&lt;/mi&gt;         &lt;/msqrt&gt;       &lt;/mrow&gt;       &lt;mrow&gt;         &lt;mn&gt;2&lt;/mn&gt;         &lt;mo&gt;#x2062;&lt;!-- &amp;InvisibleTimes; --&gt;&lt;/mo&gt;         &lt;mi&gt;a&lt;/mi&gt;       &lt;/mrow&gt;     &lt;/mfrac&gt;   &lt;/mrow&gt; &lt;/math&gt; </pre>

Tabela 1- primjer zapisa u *markup* jeziku

Rezultat u oba slučaja je sljedeći prikaz:  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

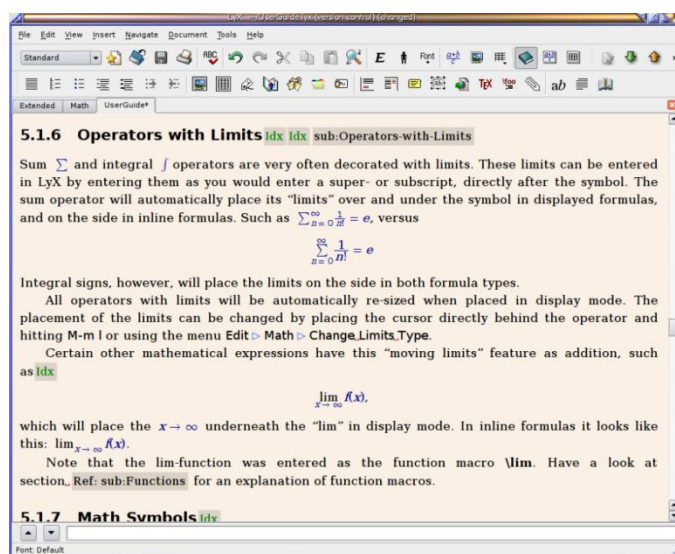
Dodatno je moguće primijetiti da su oba načina zapisa značajno drugačiji od *inline* zapisa koji je prirodniji budući da je direktno izveden iz načina na koji ljudi pišu na papiru. Ipak, on nije strukturiran u mjeri i na način na koji to prethodno opisani *markup* jezici zahtijevaju, niti je na bilo koji način standardiziran. U *inline* zapisu bi to izgledalo ovako:

$$x = (-b \text{ plusminus } \text{korijen}(b^2 - 4 * a * c)) / (2 * a)$$

Pri tome su "plusminus" i "korijen" korišteni jer odgovarajući simboli nisu raspoloživi direktno kao znakovi na tastaturi (iako se primjerice "plusminus" može dobiti držanjem tipke ALT pritisnutom i kucajući 0177 na numeričkom dijelu tastature).

Iako iz navedenih primjera to možda i nije očigledno, *markup* jezici imaju svoje prednosti koje ih i čine popularnima i upotrebljivi su jednom kada ih korisnik nauči koristiti i navikne se na "ekosistem" kojeg čine različiti alati za kreiranje dokumenata upotrebom navedenih *markup* jezika.

Ipak, za ostvarivanje funkcija koje su date u 2.2, upotreba *markup* jezika kao primarnog načina unosa zasigurno otpada, budući da je navedeno da će način biti jednostavan i intuitivan bez mnoštva mogućnosti i stvari koje korisnik treba pamtit i učiti. To ne znači da treba prestati razmatrati npr. LaTeX kao koristan alat kojeg će se potencijalno moći iskoristiti. Naime, *rendering* LaTeX-a integriranog u okviru HTML-a je izuzetno dobro podržan kroz različite skripte koje se izvršavaju u okviru web preglednika (engl. *display engines*) napisanih npr. u *JavaScriptu* (npr. MathJax), što otvara određene interesantne mogućnosti koje će biti razmotrene nešto kasnije [19]. Na slikama br. 1, br. 2, br.3 i br. 4 data su sučelja *LyXa* i *MathFlowa*.



Slika 1- korisničko sučelje LyX WYSIWYM uređivača za LaTeX...

### 5.1.6. Operators with Limits

Sum  $\sum$  and integral  $\int$  operators are very often decorated with limits. These limits can be entered in LyX by entering them as you would enter a super- or subscript, directly after the symbol. The sum operator will automatically place its "limits" over and under the symbol in displayed formulas, and on the side in inline formulas. Such as  $\sum_{n=0}^{\infty} \frac{1}{n!} = e$ , versus

$$\sum_{n=0}^{\infty} \frac{1}{n!} = e$$

Integral signs, however, will place the limits on the side in both formula types.

All operators with limits will be automatically re-sized when placed in display mode. The placement of the limits can be changed by placing the cursor directly behind the operator and hitting M-m l or using the menu Edit > Math > Change Limits Type.

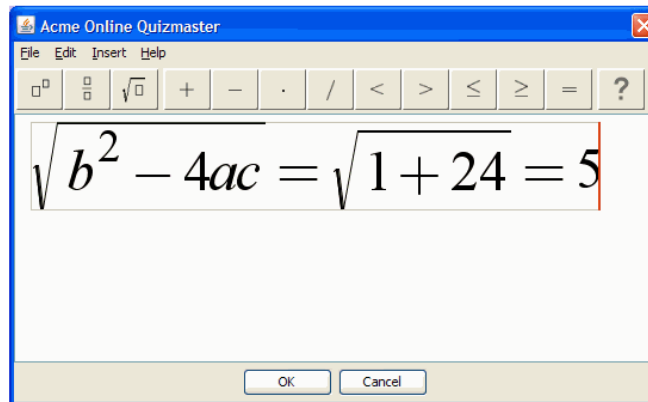
Certain other mathematical expressions have this "moving limits" feature as addition, such as

$$\lim_{x \rightarrow \infty} f(x),$$

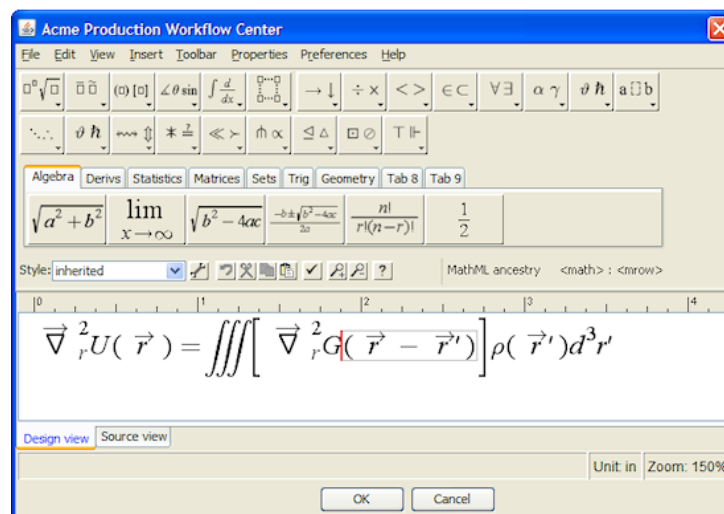
which will place the  $x \rightarrow \infty$  underneath the "lim" in display mode. In inline formulas it looks like this:  $\lim_{x \rightarrow \infty} f(x)$ .

Note that the lim-function was entered as the function macro `\lim`. Have a look at section 5.1.9 for an explanation of function macros.

Slika 2 - ... i rezultat *renderinga* prethodno uredenog dokumenta



Slika 3 - sučelje *Simple Editor MathFlow* komponente (koristi MathML u pozadini)



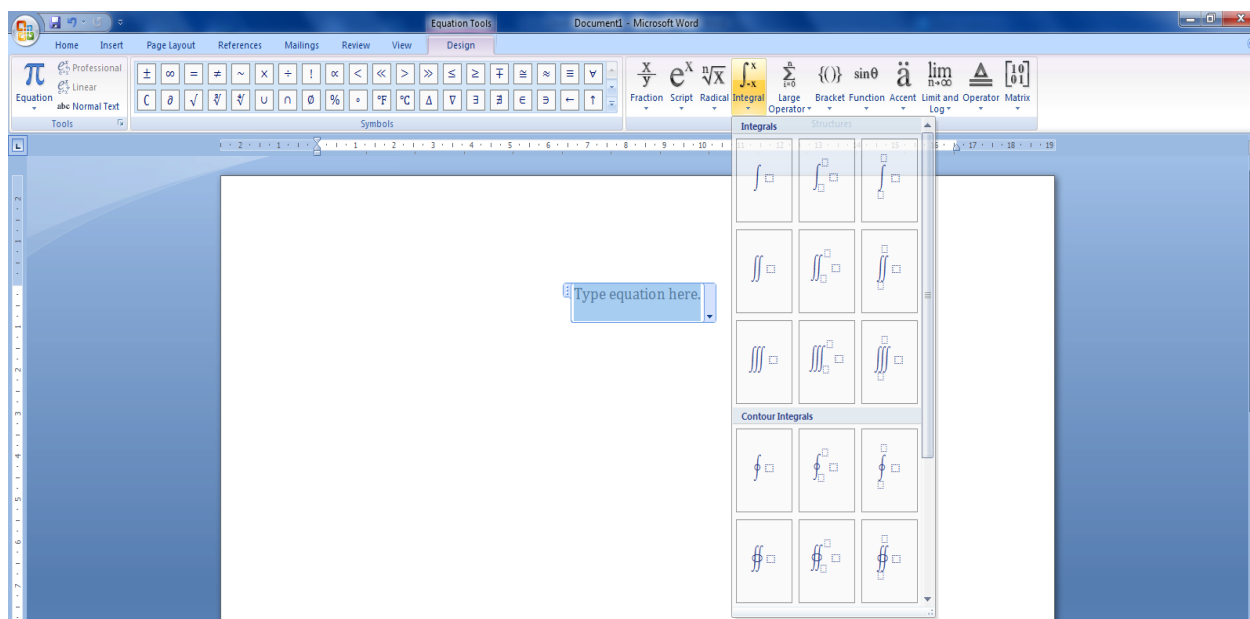
Slika 4 - sučelje *Structure Editor MathFlow* komponente (koristi MathML u pozadini)

### 2.3.2. Upotreba matematičkih objekata u bogatim formatima dokumenata

Alternativa učenju *markup* jezika jeste upotreba WYSIWYG uređivača poput MS Worda koji omogućavaju relativno jednostavno kreiranje dokumenata u kojima se, između ostalog, javljaju i matematički objekti ("*Equations*") koji direktno u dokumentu kojeg se uređuje prikazuju matematičke formule/jednačine/izraze tačno onakvima kakvi trebaju biti. *Equation* objekti se ubacuju kao i bilo koji drugi tip podržanih objekata (poput slika, grafova, *clip arta*, hiperlinkova itd.) i potom se u posebno polje unose željeni izrazi. Ukoliko se korisnik "poigra" i izmijeni predefinirane postavke aplikacije, moguće je i dati korisnički definirane kratice za unos raznih simbola poput "\alpha" za simbol " $\alpha$ " ili primjerice automatski kreirati razlomke u takvom polju pritiskom na "/" i potom unosom razmaka. Moguće je specificirati i neke često korištene formule i spasiti ih, pa ih potom automatski ubacivati kada se za njima ukaže potreba (opcija "*Save selection to Equation Gallery*"). No, najčešća upotreba i u okviru MS Word aplikacije svodi se na kreiranje matematičkog objekta, nakon čega se priloženi objekti (raspoloživi kroz odgovarajući *Equation design* meni: razlomak, integral, dvojni integral, vektor, sinus, kosinus, matrica, itd.) *slažu* i grupiraju kako bi se kreirao željeni izraz. To je relativno spor proces u odnosu na kucanje običnog i neformatiranog teksta (ubacivanje objekta obavlja se upotrebom miša, a unos sadržaja upotrebom tastature), no rezultira izrazima prikazanima onakvi kakvi jesu, a koje je kasnije moguće izmijeniti prostim pozicioniranjem kursora na odgovarajuću poziciju u okviru izraza.

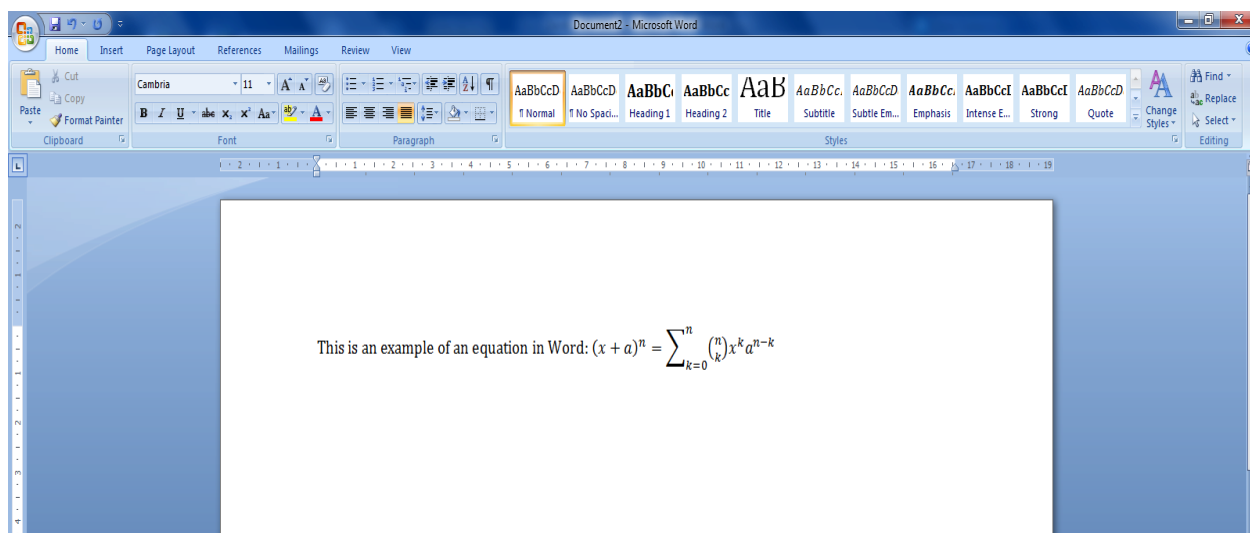
Sudeći temeljem do sada izloženoga, mogao bi se steći dojam da je upotreba programskog rješenja poput MS Worda cjelovito rješenje izloženog problema unosa matematičkih izraza u računar, no to ipak nije u potpunosti slučaj. Naime, zahtjevi postavljeni u 2.1 traže *jednostavnu* aplikaciju namijenjenu *isključivo* kreiranju *plaintext* dokumenata koji imaju matematičke sadržaje i sa mogućnosti obavljanja izračuna, što je stavka koja, iako postoji u MS Word, nije implementirana za ostvarivanje funkcionalnosti na način kako je to zamišljeno za aplikaciju koja se razvija. Naime, MS Word ima ugrađen jednostavan kalkulator i omogućava primjenu određenih formula [20], no u suštini se radi se o kompleksnom programskom rješenju čija je samo jedna od opcija kreiranje matematičkih sadržaja i kao takvog ga treba i posmatrati (nedostaju mu domenski specifične mogućnosti- što je i sasvim prirodno jer se radi o uređivaču tekstualnih dokumenata, a ne naučnom kalkulatoru), posebno imajući u vidu cijenu samog *Office* paketa.

Na slici br. 5 prikazano je sučelje MS Word sa aktiviranom opcijom *Equation Tools* koja služi za uređivanje objekata matematičkog teksta u okviru dokumenta.



Slika 5 - sučelje MS Word, *Equation Tools*

Na slici br. 6 prikazan je izgled jednog objekta matematičkog teksta u okviru MS Word dokumenta. Objekat je izgrađen dodavanjem objekata raspoloživih kroz *Equation Tools*.



Slika 6 - sučelje MS Word, primjer prikaza unesene formule

## 2.4. Zaključak

U ovom poglavlju dat je opis problema za kojeg se kreira odgovarajuće programsko rješenje. Identificirane su funkcije koje će implementirati rezultirajuća aplikacija i koje će na specifičan način riješiti navedeni problem. Dat je pregled postojećih programskih rješenja i opisana su dva glavna načina njihovog funkcioniranja. Također su izložene njihove prednosti i nedostaci.



### 3. Dizajn rješenja

U okviru ovog poglavlja opisuje se korištena metodologija analize i dizajna, a na osnovu koje se modelira i dizajnira aplikacija koja će ispuniti zahtjeve postavljene u prethodnom poglavlju. Daje se i pregled alternativnih modela i obrazlažu se njihove prednosti i nedostaci.

#### 3.1. Korištena metodologija

U okviru ovog odjeljka razmatra se način na koji će se razvijati datu aplikaciju. Stavke logički slijede jedna drugu i prate standardizirani tok razvoja programskih rješenja, odnosno: identifikaciju (analizu) problema, analizu, dizajn, implementaciju i testiranje sistema uz zaključak u kojem se razmatra uspješnost obavljenog projekta [21].

1. Na početku se određuje šta se to želi postići- koji to problem se želi rješavati upotrebom softvera kojeg će se razviti, kao i po čemu se to rješenje razlikuje od eventualno već postojećih rješenja. Nakon identifikacije problema, obavlja se proces objektno orijentiranog dizajna aplikacije.
2. Potom se pristupa opisivanju funkcionalnosti sâmog softvera koje rješavaju identificirani problem. Opis funkcionalnosti daje se na visokom nivou (kroz *dijagram slučajeve upotrebe* ili više njih) kojim se opisuje scenarije interakcije između krajnjeg korisnika i softvera, tj. šta će korisnik moći raditi upotrebom aplikacije.
3. Nakon toga se razvija i dokumentira eventualne eksterne ili vlastite *standarde, procedure i komponente* koje će aplikacija koristiti, implementirati ili poštivati kako bi podržala funkcije navedene u dijagramima slučajeva upotrebe.
4. Temeljem određenih zahtjeva, analizira se arhitekturu sistema. *Dijagramima raspoređivanja* daje se implementacijski pogled na sistem. Prikazuje artefakte koji čine sistem, pa samim time u određenom smislu daje i pogled na logičku strukturu sistema kao cjeline.
5. Na osnovu određene arhitekture sistema kao cjeline, razmatra se komponente zastupljene u okviru svakog od njegovih dijelova (npr. komponente serverskog ili klijentskog dijela sistema i sl.). Detaljniji prikaz komponenti daje se odgovarajućim dijagramom komponenti ili podsistema. U sklopu ove faze moguće je okvirno odrediti osnovne akcije (poruke) koje se razmjenjuju između komponenti. *Dijagramima komponenti* daje se razvojni pogled na sistem koji se bazira na logičkom pogledu i predstavlja vrh logičkog pogleda na sistem.
6. Po određivanju podsistema, razmatraju se ključne aktivnosti koje se odvijaju u sistemu, bilo da su direktno inicirane od korisnika, bilo da se one podrazumijevaju kako bi se omogućila ili adekvatno podržala korisnički inicirana akcija. Aktivnosti u ovoj fazi razmatraju se na nivou podsistema, ali ne i njihovih konkretnih dijelova (klâsa). *Dijagramima aktivnosti* daje se najviši dio procesnog pogleda na sistem.
7. Nakon što su određeni konkretni podsistemi, aktivnosti u kojima sudjeluju i načini njihovog toka, za svaki od podsistema određuju se klase iz kojih su sačinjeni. Klasama su određeni njihovi atributi, metode i načini na koji su povezani ili ostvaruju interakciju sa drugim klasama istog podsistema. *Dijagramima klâsa* daje se niži dio logičkog pogleda na sistem.

8. U ovom koraku upotpunjava se procesni pogled na sistem. Daje se pregled osnovnih algoritama koji se odvijaju u sistemu. *Pregledom algoritama* dat je niži dio procesnog pogleda na sistem.
9. Po završetku dizajna procesa i logike u aplikaciji, specificira se način na koji se osigurava potrebno korisničko iskustvo u radu sa aplikacijom. Također se specificira osnovne elemente i principe kojima se treba voditi pri razvoju *grafičkog korisničkog okruženja* preko kojeg korisnik radi sa aplikacijom.
10. Kada je jednom zaokružen proces analize i dizajna, dobija se potpun logički pogled na sistem kojeg je onda moguće implementirati u različitim programskim jezicima i razvojnim okruženjima. Diskutira se različita razvojna okruženja, nakon čega se opredjeljuje za jedno čije će se specifičnosti uvažiti tokom implementacije sistema. Jasno, suočiti će se sa određenim problemima, ali i olakšicama prilikom "prebacivanja" logičkih specifikacija u neki konkretan jezik ili programski model, što se na odgovarajući način dokumentira. Paralelno sa implementacijom sistema obavlja se i testiranje komponenti koje se razvijaju. U pojedinim slučajevima, prvo se dizajniraju testovi i interfejsi komponenti, pa se tek potom pristupa implementaciji.
11. Nakon što je sistem spreman za funkcioniranje u stvarnim radnim uvjetima, diskutira se koliko implementacija zbilja odgovara specifikaciji, kao i u kolikoj se mjeri uspjelo ostvariti zadate ciljeve. Diskutira se i koja su to eventualna poboljšanja koja je moguće napraviti kako bi se izvršile korekcije sistema ili se sistem usavršio. Prateća dokumentacija ovog završnog rada je i kratko korisničko uputstvo za rad sa aplikacijom.

Napomena: nisu svi navedeni koraci direktno referencirani niti zasebno navođeni u okviru ovog rada, no tok rada kao i ovog dokumenta je usklađen sa prethodno navedenim koracima.

Odabrani način rada u suštini je djelimično modificirani vodopadni model [22]. Opravdanje za upotrebu ovako zastarjelog i nepraktičnog modela razvoja je prilično jednostavno: korisnički zahtjevi se neće mijenjati (ovo je završni rad i "as is" aplikacija), a priroda aplikacije je takva da je unaprijed određeno šta će se i kako raditi. Budući da je zahtijevanih funkcionalnosti svega nekoliko, da ih implementira jedna osoba (autor ovog završnog rada), i da su funkcionalnosti relativno algoritamski komplicirane za realizirati, to je ovakav model (ponegdje uz dodatak elemenata *Test Driven Development*- prvo dizajn testova pa potom implementacija komponente [23]) sasvim prikladan.

## 3.2. Model aplikacije

U sklopu ovog odjeljka razmatraju se načini na koje je moguće osmisliti aplikaciju a da ona ispunjava osnovne zahtjeve koji su ranije izloženi. Pri tome, u odjeljku 3.2.1 autor diskutira neke od načina na koje je pokušao implementirati aplikaciju, ali su završili rezultatom koji je bio neprikladan ili nezadovoljavajući. Iako odjeljak 3.2.1 ne čini glavninu ovog dokumenta, u njega je zasigurno utrošeno višestruko više vremena nego u ostatak rada, budući da su u tom odjeljku opisani različiti načini na koje je autor ovog rada pokušao pristupiti razvoju adekvatnog programskog rješenja za postavljeni problem. Svi isprobani pristupi su bili vrijedno iskustvo i u narednom odjeljku se dokumentuje ono što je naučeno.

### 3.2.1. Diskusija različitih modela

Imajući u vidu funkcionalnosti koje aplikacija treba omogućiti, moguće je osmisliti više različitih načina na koje bi se one mogle realizirati i implementirati. Pri tome se nije ulazilo u odabranu tehnologiju za eventualnu realizaciju. Važno je za napomenuti da je autor sve navedene pristupe isprobao i u određenoj mjeri i realizirao upotrebom *.NET frameworka* i *C# WPF-a (Windows Presentation Foundation)*.

Prvi korak jeste osmisliti kako bi to aplikacija trebala izgledati i funkcionirati jednom kada već bude implementirana, i to iz perspektive korisnika. U te svrhe je nužno odrediti šta je od razmatranih funkcija važnije: da li je to prikaz matematičkih izraza onakvima kako stvarno trebaju izgledati u okviru radnog prostora (rješenje u stilu MS Word), ili su to za područje primjene specifične funkcije kreiranja varijabli i brzih jednostavnih izračuna, vođenje računa o otvorenim i zatvorenim zgradama i tome slično. Naime, ispostavlja se da je naivni pristup u kojem se želi ostvariti i jedno i drugo izuzetno nepraktičan i zahtijeva dugotrajnu implementaciju uz primjenu raznovrsnih obilaznih puteva da bi se nešto ostvarilo ili napravilo (engl. *workaround*).

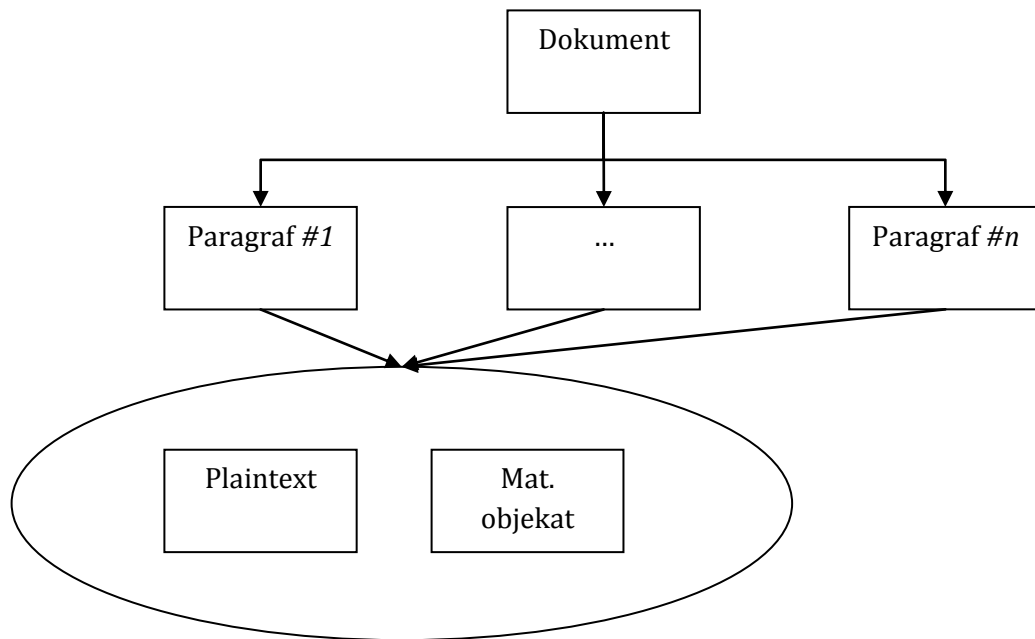
#### 3.2.1.1. Prvi pristup: WYSIWYG upotrebom korisničkih kontrola

Vjerovatno najambiciozniji pristup jeste omogućiti funkcionalnosti na način sličan tome kako je to urađeno u MS Wordu, uz dodavanje mogućnosti deklariranja varijabli, obavljanja evaluacije izraza i *plotanja* grafika. Jasno, to podrazumijeva upotrebu naprednih kontrola za uređivanje teksta kojima se onda proširuju mogućnosti vlastitom implementacijom matematičkih objekata. Iako se na prvi pogled radi o pravocrtnom procesu u kojem je sasvim jasno šta je potrebno učiniti da bi se kreiralo takvo rješenje, već i u konceptualnoj fazi razvoja nailazi se na brojne prepreke.

Prvi korak jeste specificirati način na koji se jedan takav dokument pohranjuje u memoriji. U skladu sa dobrim praksama, odvaja se prezentacijska (grafičke korisničke kontrole) od aplikacijske logike (objekti nositelji podataka i logičke strukture) i specificira se potrebna interakcija između odgovarajućih komponenti. Korisnik ostvaruje interakciju sa prezentacijom i kroz prezentaciju mijenja logičku strukturu, koja se rekonfiguriše i regenerira prezentirani sadržaj.

Jedna moguća struktura takvog dokumenta bi bilo n-arno stablo u čijem korijenu se nalazi dokument, čvorovi-djeca korijena bi bili paragrafi kojih može biti proizvoljno mnogo, a čvorovi-djeca paragrafa bi bili čvorovi tipa običnog teksta ili matematičkog objekta.

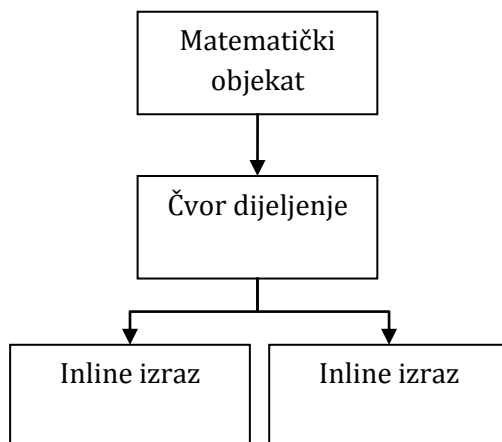
Matematički objekat bi imao strukturu *stabla aritmetičkog izraza* izgrađenog na osnovu prefiksne ili postfiksne notacije [24].



Dijagram 1- prijedlog logičke organizacije dokumenta za WYSIWYG uređivač

No, za WYSIWYG uređivač ispostavlja se da je stablo aritmetičkog izraza izuzetno složeno koristiti kao osnovu iz koje se obavlja *rendering*. Prvi problem jeste činjenica da je stablo aritmetičkog izraza izgrađeno na osnovu *validnog* izraza koji je smislen. Tako je npr. izraz "6+" besmislen i neće rezultirati korektno izgrađenim stablom iz kojeg, slijedeći izloženu logiku, treba kreirati odgovarajuću prezentaciju. No, takav izraz će upravo postojati u jednom trenutku tokom unosa izraza, npr. "6+x". Dakle, potrebno je implementirati određeno "predviđanje" ili uvesti koncept "očekivanog izraza", čime bi pojava validnog operatora automatski ubacivala "očekivani izraz", pa bi onda bilo moguće napraviti i odgovarajuće stablo.

Drugi problem je nešto teže uočiti: ukoliko se pokuša ostvariti 1:1 uvezivanje logičke i prezentacijske strukture, onda bi npr. za čvor koji predstavlja dijeljenje bilo prirodno očekivati da ima dva podčvora koji mogu sadržavati bilo kakav drugi izraz, gdje npr. prvi predstavlja brojnik, a drugi nazivnik. Neka je u matematički objekat unesen samo jedan razlomak. Data struktura odgovarala bi datome na dijagramu br. 2.



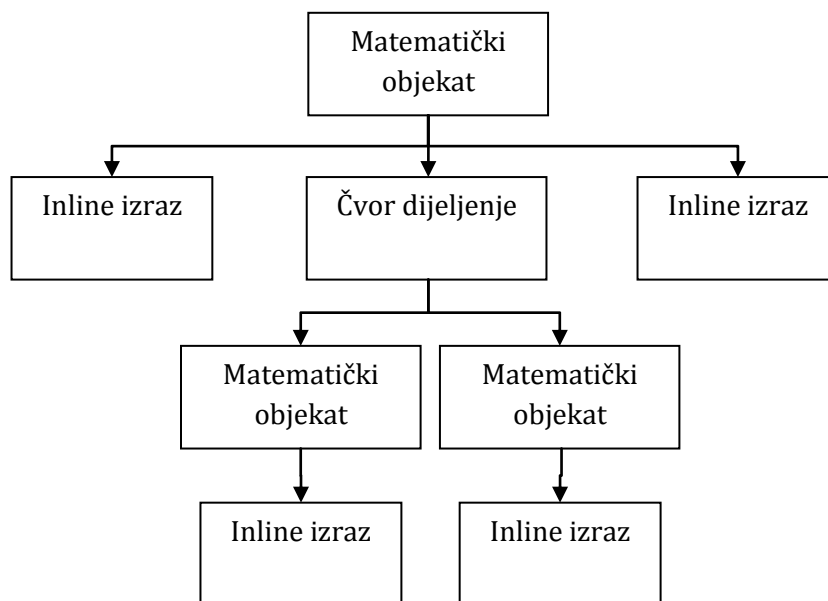
Dijagram 2 - prijedlog logičke organizacije matematičkog objekta "razlomak"

Na dijagramu br. 2 datu strukturu se upotrebom rekurzije može sasvim jednostavno vizualizirati tj. pretvoriti u odgovarajuće grafičke kontrole. No, šta ako korisnik želi "napustiti" razlomak i nastaviti dalje pisati izraz, kao što je recimo  $\frac{x}{y} + 6$ ? U prikazanoj strukturi nigdje nije ostavljen prostor za napuštanje razlomka, što znači da nije moguće direktno uvezati stablo aritmetičkog izraza i odgovarajuću prezentaciju tako da prezentacija osvježava stablo, stablo se rekonfigurira i potom osvježava prezentaciju. Dodatno, kako se prezentacija osvježava, potrebno je pamtit i poziciju kursora u okviru prezentacije i adekvatno je predstaviti u stablu, tako da se javlja potreba za čvorovima-markerima koji će dijeliti čvorove tipa "inline izraz" na dio ispred i iza kursora.

Očigledno je potrebno prilagoditi stablo izraza kako bi ono bilo prikladnije za prezentaciju, i već iz potrebe za postojanjem ovog koraka moguće je zaključiti da će u nastavku rada biti potrebno činiti mnoge kompromise da bi se ovakav program učinio funkcionalnim.

Također, već sada postaje jasno zašto se uređivači teksta koji podržavaju matematičke izraze dijele na one koji nameću stroga pravila na strukturu kako bi se čuvao smisao i značenje izraza (*markup* jezici) i stavljaju ta pravila ispred vizuelno odgovarajućeg prikaza, i one koji samo nude vizuelno odgovarajući prikaz strukture izraza bez da je tokom uređivanja prisutna interpretacija smisla i njegovog značenja (*WYSIWYG*).

Na dijagramu br. 3 dato je poboljšano stablo koje omogućava 1:1 mapiranje i napuštanje npr. razlomka, no pri tome usložnjava mogućnost evaluacije takvog izraza. Rješenje je zasnovano na prilagođavanju stabla logici prikaza, gdje su sada ubačeni "matematički objekti" umjesto *inline* izraza. Primjerice, *objekat* je generička kontrola koja može imati sadržaj proizvoljnog tipa (praktično: kontejnerska kontrola), dok je *inline* izraz npr. `textbox`.



Dijagram 3 - prilagođena struktura stabla

Sada kada je, uz određene kompromise, načelno riješena struktura dokumenta i matematičkih objekata, javlja se niz drugih problema sa ovakvim pristupom.

Prvi problem ovakvog pristupa predstavlja činjenica da je potrebno potpuno samostalno riješiti problem prikaza takvih objekata, što nije nimalo jednostavno za učiniti na iole konzistentan i vizuelno prihvatljiv način. Sasvim reprezentativan primjer jeste prikaz objekta tipa razlomak.

Naivno rješenje bi bilo kreirati novu kontrolu (objekat) koji će u sebi sadržavati dva atributa tipa matematičkog objekta (što može biti npr. neki drugi razlomak, ili bilo kakav izraz) i jednu horizontalnu liniju, gdje će prvi objekat se prikazivati iznad horizontalne linije (brojnik), a drugi ispod te linije (nazivnik). Navedeno se prilično brzo realizira npr. u WPF-u gdje se jednostavno kreira nova korisnička kontrola (komponenta `UserControl`), postavljaju se odgovarajuće mreže (komponenta `Grid`) kao područja u kojima će se nalaziti brojnik i nazivnik, postavlja se horizontalna linija i praktično je napravljen šablon (engl. *template*) kako razlomak treba izgledati neovisno o njegovom sadržaju. Pored razlomka, za početak neka je specificiran i neki *inline* izraz (tipa  $5+x+y$ ) što je očito trivijalno realizirati kao obično jednolinijsko polje za unos teksta (za što također postoji odgovarajuća `UserControl`). No, tu problemi tek počinju: izrazi bi trebali biti promjenjivi, što znači da kreirane kontrole ne smiju imati fiksne veličine. *Inline* izraz treba se širiti kako se unosi tekst, a dodatno, taj izraz treba obavještavati sve ostale kontrole koje ga sadrže da i one trebaju mijenjati svoju veličinu. Ako to ne bi bilo definirano ponašanje, rezultat bi bio razlomačka crta fiksne dužine dok bi brojnik mogao biti proizvoljne dužine što je svakako vizuelno neprihvatljivo.

Sljedeći problem predstavlja činjenica da objekat poput razlomka mora sadržavati podatke o svojoj fizičkoj i logičkoj veličini. Neka je dat inline izraz poput  $a + \frac{b+c}{d}$ . Čak i ukoliko je na prihvatljiv način riješen unos i gniježđenje izraza, javlja se novi problem- poravnanje, koje u prethodnom primjeru ne ide po fizičkoj, već logičkoj sredini razlomka (brojnik ima veću visinu od nazivnika, i razlomak se mora poravnavati po razlomačkoj crti sa ostatkom izraza, a ne po fizičkoj sredini koja u prethodnom primjeru ide kroz slovo "d" u nazivniku razlomka koji se nalazi u brojniku glavnog razlomka u izrazu). Naravno, ni opisani problem nije nerješiv: moguće je kreirati mehanizme reagiranja ovakvih objekata na događaje promjene veličine drugih objekata, kao i specificirati odgovarajuća ponašanja i poravnanja. Ipak, navedeni mehanizmi ovisni su o veličini, tipu korištenog fonta i vertikalnom poravnanju teksta, što često uvodi greške kroz *floating point* aritmetiku i programer je prisiljen koristiti razne "magične" konstante i korekcije izračuna da bi učinio prikaz odgovarajućim i konzistentnim.

Dalje, iz polazne premise jasno je da će se matematičke objekte ubacivati u neku grafičku kontrolu (npr. TextBox) koja je već riješila način na koji prikazuje tekst. Prikaz teksta nije trivijalan problem (što je autor i spoznao nakon nekog vremena provedenog istražujući problematiku): prvi problem predstavlja mogućnost upotrebe raznovrsnih porodica fontova sa različitim stilovima i veličinama, što je donekle moguće izbjeći prostim pozivanjem na tražene funkcionalnosti: program služi za unos *plaintexta* i kao takav može, ali i ne mora nuditi korisniku promjenu fonta, njegovog stila i veličine. No, tu nije kraj problemima: tipografija uvodi i koncepte poravnanja po vertikali i horizontali, što neke naprednije kontrole za uređivanje teksta i nude kao mogućnost kojom je moguće upravljati (npr. WPF-ova RichTextBox korisnička kontrola). Novokreirane matematičke objekte je nužno adekvatno poravnati sa eventualno već postojećim tekstom, što se autoru ovog rada ispostavilo kao nepremostiv problem, čak i kada je sve ostalo bilo već riješeno. Formule su, iako interno konzistentno poravnate, uvijek bile neporavnate u odnosu na vanjski tekst jer principijelno koristeći raspoložive kontrole nije moguće podešavati vertikalna poravnanja van okvira predefiniranih vrijednosti (što se može donekle smatrati ograničenjem korištene konkretne tehnologije). Nakon daljeg eksperimentiranja uočen je još jedan problem koji nije nimalo trivijalan za riješiti: prelazak formule u novi red jednom kada postane predugačka u odnosu na širinu trenutne stranice, što zahtijeva ručno upravljanje načinom prikaza formule i njenim crtanjem na ekranu.

Još jedan važan aspekt za razmotriti jeste da u slučaju da se želi napraviti *WYSIWYG* uređivač, nužno je da se svi matematički objekti ponašaju kao obični tekst, što podrazumijeva selekciju pojedinih objekata i njihovo kopiranje, lijepljenje i brisanje, kao i pokrivanje korak-unazad (engl. *undo*) i korak-unaprijed (engl. *redo*) funkcijama. Pokazuje se da je navedeni problem možda i glavna prepreka implementaciji navedenog pristupa: zahtijeva ručnu obradu svih navedenih akcija, kao i direktno praćenje pomjeranja kursora prilikom selekcije objekata (budući da u svim alatima za kreiranje grafičkih sučelja postoji ograničenje da nije moguće da više objekata istovremeno ima fokus, tj. da budu selektirani- kako su matematički objekti različiti a treba ih moći selektirati, to je potrebno ručno vizuelno simulirati selektiranje, a istovremeno postavljati logičke markere u hijerarhiji objekata da bi se znalo šta je selektirano a šta ne).

Kao zaključak, navedeni pristup jeste izvediv (iako pokušava spojiti nespojive stvari), ali zahtijeva pisanje kôda za sve elementarne akcije (selektiranje, kopiranje, izrazivanje, lijepljenje brisanje), što praktično znači da je potrebno ne samo riješiti specifične probleme (npr. evaluaciju izraza), već i ispočetka definirati i implementirati posebnu prezentacijsku i interakcijsku logiku, uz primjenu raznih trikova (odstupanje od klasičnog oblika aritmetičkog stabla) i obilaznih puteva (upotreba magičnih konstanti i "štimanje" prikaza, "zloupotreba" grafičkih kontrola i njihova primjena na način koji nije predviđen) koji čine implementaciju dužom, složenijom, težom za testiranje i podložnijom greškama, dok sa sobom nosi upitne koristi.

U uvjetima kada postoje gotova rješenja za *rendering* formula/jednačina/izraza, ručno upravljanje svim aspektima prikaza i rada sa matematičkim sadržajima sasvim opravdano se može posmatrati kao izmišljanje *tople vode*. Dodatno, sâm način rada sa jednim ovakvim rješenjem, čak i da je ono sasvim prihvatljivo implementirano, ne razlikuje se u mnogome od načina kako je to urađeno u MS Wordu: ubacuje se poseban objekat i formula se gradi dodavanjem novih objekata, samo što se uz dosta truda tu "ubacio" i jednostavni kalkulator.



### 3.2.1.2. *Drugi pristup: markup jezici*

Nakon što je kroz prethodni odjeljak praktično pokazano da je izuzetno teško napraviti kvalitetan WYSIWYG uređivač koji će istovremeno i poznavati smisao napisanih sadržaja, u ovom odjeljku razmatra se alternativa koja je već ranije navedena: upotreba *markup* jezika.

Konkretni nedostaci u radu sa *markup* jezicima bili su sljedeći:

- korisnik mora poznavati često veoma složenu i nečitku sintaksu
- korisnik ne vidi kreirani sadržaj dok se isti ne *renderira* u odvojenom radnom prostoru

S druge strane, kada *markup* jezici ne bi imali navedene osobine, smanjili bi svoju upotrebljivost i limitirali bi opcije korisnicima koji su ovladali jezikom i njegovim mogućnostima. Stoga se u ovom odjeljku razmatra nekoliko načina prevazilaženja navedenih nedostataka.

Prvi način na koji je moguće korisnicima olakšati rad jeste ponuditi im neku vrstu *autocomplete* funkcionalnosti (samodopunjavanje teksta), gdje je dovoljno da korisnik počne tipkati i programsko rješenje će mu ponuditi izbor između jedne ili više odgovarajućih opcija (npr. *tagova*) koje su pohranjene u programskoj bazi. Npr. kada korisnik ukuca " $\backslash$ al", program bi mu trebao ponuditi izbor opcija među kojima je i " $\backslash$ alpha", čijim izborom će se do tog trenutka unesena sekvenca znakova zamijeniti simbolom  $\alpha$ . Time se omogućava korisniku da ne mora doslovno paziti na sintaksu i smanjuje se mogućnost sintakasnih grešaka koje će trebati naknadno ispravljati, što znači da korisnik sada može brže obaviti svoj zadatak pisanja odgovarajućeg dokumenta. *Autocomplete* funkcionalnost može se dodatno proširiti mogućnošću automatskog ubacivanja određenih cijelih struktura (npr. ispravno napisanog razlomka u LaTeX-u koji ima oblik " $\frac{\ }{\ }$ ") u dokument.

Ipak, *autocomplete* funkcionalnosti nisu bez nedostataka, ovisno o načinu na koji se implementiraju. Prvi način implementacije jeste konstantno uključeni *autocomplete* koji korisniku nudi opcije čim prepozna odgovarajući uzorak trenutno aktivnog teksta. Ovakav način implementacije za neke korisnike može biti zamaraјуći u slučaju kada ne žele unositi nikakve posebne simbole ili *tagove*, već kucaju obični tekst. Ipak, očigledno je da koristi ovakvog pristupa imaju više koristi nego realne štete. Pogodno je rješenje onda kada *markup* čini veći dio dokumenta (relativno u odnosu na *obični* tekst).

Drugi način implementacije jeste *autocomplete* kojeg aktivira neki poseban simbol ili grupa simbola. U razvojnim okruženjima najčešće je to tačka (".") koja nudi izbor jedne od odgovarajućih metoda za dati objekat ili klasu (npr. *intellisense*). Ovo rješenje pogodno je onda kada *markup* čini manji dio dokumenta (relativno u odnosu na *obični* tekst).

Sljedeći način na koji je moguće korisnicima olakšati rad jeste kroz određenu vrstu *živog* pregleda dokumenta kojeg se kreira. To može biti realizirano npr. kroz konstantno renderiranje izmijenjenog dokumenta na odvojenom *threadu(programskoj niti)* koji će se osvježavati što je brže moguće a uz eventualno što manje zauzeće procesorskih resursa. Na taj način korisnik može odmah vidjeti rezultate svog rada i otkloniti greške, mada se poseban trud mora uložiti u identifikaciju grešaka i njihov korektan prikaz u *renderiranom* dokumentu. U slučaju kada tehnologija u kojoj se aplikacija razvija ne podržava višenitnost, isti efekat se ostvaruje tako što se dokument *renderira* tek nakon što se detektuje određena pauza u korisnikovim aktivnostima (primjerice, tristotinjak milisekundi).

Time se sa određenom vjerovatnoćom osigurava da će program imati dovoljno dobro vrijeme odziva za korisnika, a i dalje omogućavati živi pregled dokumenta kojeg se kreira.

Konačno, treća solucija jeste kreirati vlastiti *markup* jezik ili na određeni način pojednostaviti neki već postojeći kroz kreiranje odgovarajućeg konvertera ili prevoditelja. Time se ne postiže nikakvo suštinsko pojednostavljenje: korisnik i dalje mora učiti jezik kojeg je specificirao neko drugi (u ovom slučaju autor ovog završnog rada) i prilagoditi se takvom jeziku. Dodatno, ova solucija također se svodi na kreiranje alternative već provjerenim i značajno moćnijim rješenjima bez donošenja značajnih koristi. Donekle se sva predložena poboljšanja može smatrati i prilično trivijalnim za implementirati u svom osnovnom obliku (*autocomplete*, *rendering uživo*), a i kao takva ne zadovoljavaju postavku zadatka tako da u okvirima ovog rada ovakvo rješenje neće biti dalje razmatrano (iako će se određene njegove elemente iskoristiti u konačnom rješenju).

### 3.2.1.3. Treći pristup: interpretiranje obične plaintext inline notacije

Kroz prethodne odjeljke iskristalizirao se izbor između fokusiranja ili na vizuelnu prihvatljivost uređivanja dokumenta, ili na očuvanje smislenosti dokumenta nauštrb njegovog realnog prikaza. Ono što do sada nije ponuđeno kao alternativa jeste upotreba obične *inline* notacije koja se koristi onda kada je nepraktično razmjenjivati dokumente (npr. na *Haber kutiji- chatu* studenata ETF-a u Sarajevu [25]). Iako *Haber* konkretno podržava LaTeX i korektno ga *renderira*, većina korisnika ne poznaje LaTeX zapis i matematske zapise piše npr. na sljedeći način " $a*x^2+b*x+c=0$ ". Uistinu, za korištenje ovakve notacije nije potrebno nikakvo predznanje nekog *markup* jezika (iako je ekvivalentna LaTeX notacija za ovaj primjer praktično identična).

Autor ovog rada smatra da je navedena notacija najprirodnija od svih navedenih, iako je nepregledna i teška za snalaženje u slučaju složenijih izraza. Također, nije standardizirana i nemoguće ju je adekvatno prebaciti bilo u web preglednike bilo u druge uređivače dokumenata. Nedostatak vezan uz nekompatibilnost može se zanemariti budući da ostvarivanje široke kompatibilnosti ne spada u direktne ciljeve ovog rada (iako je svakako moguće prevazići navedeni nedostatak), dok je izraze date u ovoj notaciji moguće učiniti nešto preglednijima kroz jednostavne tehnike poput bojenja uparenih zagrada.

Navedena notacija je jednostavna za unos, moguće ju je učiniti vizuelno donekle prihvatljivom, no ostaje otvoreno pitanje načina implementacije zahtijevanih funkcionalnosti: adekvatan *rendering* finalnog dokumenta, mogućnosti izračunavanja i slično. Ispostavlja se da je moguće reciklirati pristup koji je podrazumijevao upotrebu *markup* jezika, samo što će se isti u potpunosti "sakriti" od korisnika. *Plaintext* sadržaj dokumenta će se jednim prolazom pretvarati i u odgovarajuću strukturu za prezentaciju (HTML sa LaTeX sadržajem) i u odgovarajuću strukturu za domenske funkcionalnosti (evaluacija izraza). Pri tome za ovu soluciju vrijede svi nedostaci opisani ranije: potrebno je na adekvatan način spriječiti nekorektan prikaz deformiranog LaTeX kôda, budući da korisnik uopće ne mora znati da je ono što mu se prikazuje u biti LaTeX kôd. Isto tako, javlja se problem identifikacije teksta kojeg treba interpretirati kao matematičke izraze u okviru dokumenta. Postoje dvije mogućnosti: prva je da se matematičke strukture nigdje eksplicitno ne najavljaju niti završavaju- programsko rješenje mora samo uočiti gdje se matematičke strukture nalaze, a tek potom pokušati učiniti ih smislenim (odrediti im strukturu), ili da se strukture eksplicitno najavljaju i završavaju čime je proces učinjen lakšim za programera (izbjegava se relativno složena heuristika koja nije otporna na greške), a nešto složenijim za korisnika (mora voditi računa šta želi da se interpretira kao matematiku, a šta ne) [26]. Pri tome se problem sprječavanja prikaza deformiranog LaTeX kôda može riješiti na trivijalan način: naime, ispravno napisan LaTeX moguće je jednostavno 1:1 mapirati u odgovarajuće stablo aritmetičkog izraza [27]. Ukoliko se ne uspije u kreiranju odgovarajućeg stabla na osnovu identificirane matematičke strukture (tehnički, u dokumentu postoji samo tekst kojeg se pokušava interpretirati kao matematički izraz), ona se jednostavno neće uopće ni prebacivati u svoj odgovarajući LaTeX kôd i u renderiranom dokumentu će biti prikazana kao obični tekst. Iz navedenog je jasno da se ovaj pristup u velikoj mjeri zasniva na od korisnika u potpunosti skrivenoj upotrebi *markup* jezika (npr. LaTeX-a, ako izuzmemo nužnost da se *matematički tekst* najavljuje i završava u dokumentu), što je u suštini ono što se ovim radom i željelo postići.

### 3.2.2. Model koji će se koristiti

Model koji se koristi za kreiranje aplikacije u sklopu ovog rada jeste model opisan u odjeljku 3.2.1.3. Kako se nakon odabranog pristupa radu određuje konkretan dizajn aplikacije, u nastavku ovog odjeljka daje se nešto detaljniji opis korištenog pristupa.

Kao što je već ranije rečeno, aplikacija gledano iz korisničke perspektive radi sa običnim tekstom čiji eksplicitno od korisnika specificirani dijelovi se u finalnom *renderu* dokumenta prikazuju upravo onako kako i trebaju izgledati. Pri tome ih se mora moći interpretirati kao korekne matematske izraze.

U svrhe ostvarivanja za domen specifičnih funkcija ipak se moraju uvesti određene konvencije, u ovom slučaju date kao posebni simboli ili kratice. Što se tiče ostvarivanja funkcije samodopunjavanja, ona neće biti uvijek uključena i korisnik će ju morati eksplicitno pozivati. Ipak, kraticu za poziv te funkcije treba učiniti što je moguće jednostavnijom a da to ne bude često korišten simbol. Primjerice, tilda (~) ili kosa crta (\) predstavljaju prikladne kratice. Dodatno, kratica se mora moći koristiti i kao regularan simbol u okviru dokumenta.

Što se tiče *renderinga* finalnog dokumenta, zbog jednostavnosti korištene sintakse i funkcije uljepšavanja teksta (bojenje zagrada i ključnih riječi i simbola), *rendering uživo* finalnog dokumenta je nešto što se može ali i ne mora implementirati.

Što se tiče funkcionalnosti evaluacije unesenih izraza, ona zahtijeva nešto detaljnije obrazloženje prije nego se pristupi konkretnom dizajnu iste. Naime, ono što se želi postići jeste mogućnost kreiranja varijabli u okviru dokumenta (npr. " $x=10$ "), gdje će se simbolu " $x$ " dodijeliti vrijednost 10. Kasnije bi u dokumentu trebalo moći koristiti vrijednost varijable u drugim izrazima.

Također, izrazi se ne bi smjeli automatski evaluirati kako bi se korisniku omogućilo da na proizvoljnim mjestima u dokumentu može imati proizvoljne matematičke izraze, što znači da je evaluacija eksplicitna. U te svrhe bi trebala postojati ključna riječ ili simbol. Rezultat evaluacije trebao bi biti ili konkretna vrijednost, ili neki rezultatni izraz (npr. *eval*< $x+10$ > u situaciji kada simbol  $x$  nema dodijeljenu vrijednost trebao bi se evaluirati u " $x+10$ "). Zarad jednostavnosti ne bi trebalo omogućavati ugnježdivanje evaluacija, budući da je obilazni put za ostvarivanje navedene funkcionalnosti trivijalan i značajno pregledniji za korisnika.

Pored toga, rad korisnika bi bio uveliko olakšan ukoliko bi mu se omogućilo evaluiranje prethodnog izraza (u notaciji npr.  $5+x=?$ ) koja bi u stvarnom vremenu da datom mjestu ubacila rezultat evaluacije.

Nužno je postojanje osnovnih elemenata koji čine *plaintext* čitljivijim, kao što je npr. bojenje uparenih zagrada, otvaranje i zatvaranje matematičkog teksta ili podebljavanje ključnih riječi.

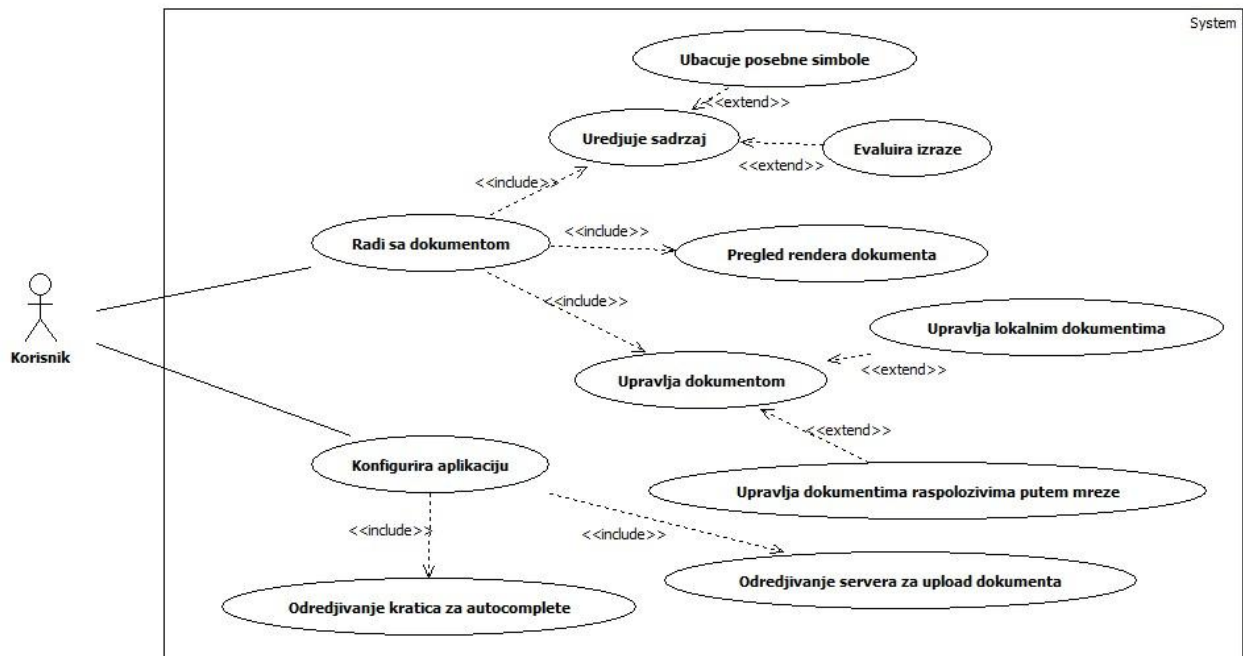
U okviru dizajna se svaka od navedenih funkcija detaljno specificira i određen je način njihove implementacije.

### 3.3. Dizajn aplikacije

Naziv aplikacije koju se razvija je *MathEasy*. U okviru ovog odjeljka specificira se dizajn aplikacije, počevši sa slučajevima upotrebe i kontekstom sistema, preko arhitekture sistema i sistemskih komponenti, aktivnosti u korištenju i radu aplikacije, do identifikacije klasa i korištenih algoritama.

#### 3.3.1. Slučajevi upotrebe

Prije nego se pređe na konkretno specificiranje dizajna aplikacije, korisno je prezentirati šta to aplikacija radi iz perspektive krajnjeg korisnika koristeći se *use case* dijagramom, odnosno, dijagramom slučajeva upotrebe. Na ovom tipu dijagrama se na visokom nivou apstrakcije prikazuje smisao interakcije između korisnika i aplikacije [21].



Dijagram 4 - dijagram slučajeva upotrebe

Slijedi diskusija konkretnih funkcionalnosti uz odgovarajuće opise i obrazloženja:

#### 1. Rad sa dokumentom podrazumijeva:

##### 1.1. Akcije vezane uz dokument kao cjelinu (datoteku), što se svodi na:

1.1.1. **Kreiranje novog praznog dokumenta**, što rezultira otvaranjem novog praznog dokumenta u kojeg je moguće unositi sadržaj.

1.1.2. **Otvaranje postojećeg dokumenta** se može obaviti na dva načina:

1.1.2.1. **Otvaranje lokalnog dokumenta**, tj. dokumenta koji je raspoloživ na korisničkom računaru.

1.1.2.2. **Otvaranje dokumenta sa mreže**, tj. preuzimanje (download) i konsekvantno otvaranje preuzetog dokumenta u radnom okruženju. Za razliku od otvaranja lokalnog dokumenta, potrebna je informacija drugačije prirode (link na mrežni objekat/resurs umjesto putanje na disku):

- 1.1.2.2.1. **Navođenje lokacije**, što znači da korisnik mora aplikaciji proslijediti neki podatak (npr. link) na osnovu kojeg će mu aplikacija sa mreže preuzeti njegov imenovani resurs.
- 1.1.3. **Spašavanje radnog dokumenta** podrazumijeva da se trenutni sadržaj radnog prozora spašava u odgovarajućem formatu u datoteku kako bi mu se kasnije moglo pristupiti. Također se može obaviti na dva načina:
  - 1.1.3.1. **Spašavanje dokumenta na lokalnom računaru** će u odabranom formatu zapisa spasiti trenutni sadržaj radnog dokumenta u datoteku specificiranih atributa na specificiranoj lokaciji. Izbor formata zapisa je sljedeća funkcija:
    - 1.1.3.1.1. **Izbor formata u kojem će se dokument spasiti** podrazumijeva da korisnik može odabrati između TXT i HTML formata zapisa prilikom spašavanja dokumenta. Iako se može smatrati da spašavanje nije ista stvar kao i prebacivanje dokumenta u treći format koji nije *nativni* za aplikaciju, ovo se može okarakterizirati kao *Save as..* funkcionalnost. Jasno, spašavanjem u HTML formatu gubi se mogućnost editiranja dokumenta iz okvira aplikacije (*read-only save*).
    - 1.1.3.2. **Spašavanje dokumenta na mrežu** će trenutno aktivni sadržaj radnog prostora aplikacije *uploadovati* na mrežni servis. Korisnik ne mora aplikaciji davati nikakve instrukcije tokom korištenja ove funkcionalnosti. Kao rezultat spašavanja, korisnik će dobiti odgovarajuće *URL*-ove za pristup i rad sa mrežnim resursom kojeg je upravo spasio na mrežu. Važno je napomenuti da se dokument spašava i u TXT i u HTML formatu, tako da ovisno o načinu pristupa, korisnici mogu sadržaj vidjeti i kroz web preglednik (ako žele čitati dokument) i kroz aplikaciju (ako žele mijenjati dokument).
- 1.2. **Akcije vezane uz sadržaj dokumenta**, što su u suštini *core* funkcionalnosti aplikacije, podrazumijevaju sljedeće:
  - 1.2.1. **Unos teksta**, kao bazična funkcionalnost je jasna za razumjeti. Aplikacija omogućava pravljenje dokumenata koji sadrže i tekst i matematičke formule unutar teksta dokumenta. Nisu dozvoljene bilo kakve dodatne konfiguracije nad fontom ili veličinom fonta. Dodatno, prilikom unosa teksta, koristi se baza *autocomplete* parova, i korisniku se nudi mogućnosti zamjene određene sekvence znakova drugom sekvencom znakova (ukoliko želi).
  - 1.2.2. **Unos matematičkih izraza i formula** se obavlja na način identičan unosu običnog teksta i ni po čemu se ne razlikuju. Pri tome se u izrazima koriste određene ključne riječi koje će biti zasebno definirane. Također, izrazi moraju zadovoljavati određenu strukturu da bi bili korektno interpretirani kao izrazi, a ne kao obični tekst.
  - 1.2.3. **Unos posebnih komandi** što podrazumijeva ključne riječi koje omogućavaju trenutno kopiranje posljednjeg unesenog izraza na trenutnu poziciju i trenutnu evaluaciju i upis rezultata evaluacije posljednjeg unesenog izraza na trenutnu poziciju.

- 1.2.4. **Pregled rendera dokumenta** što podrazumijeva da korisnik u bilo kojem trenutku može pregledati rezultirajući HTML dokument.
  - 1.2.5. **Anuliranje promjena** je samo prijevod engleske riječi *Undo*, i omogućava korisniku da poništi izmjene koje je napravio nad dokumentom.
  - 1.2.6. **Kopira, reže i lijepi sadržaj** podrazumijeva da korisnik ima mogućnost standardne manipulacije nad označenim (selektiranim) *plaintextom*.
2. **Konfiguriranje aplikacije** podrazumijeva da korisnik može mijenjati svoje korisničko iskustvo sa aplikacijom i utjecati na način njenog rada. Konfiguracije mogu biti:
- 2.1. **Konfiguracije vezane uz sadržaj dokumenata**, koje omogućavaju korisniku da:
    - 2.1.1. **Podešava autocomplete parove** oblika (ono što se zamjenjuje, ono čime se zamjenjuje), čime se korisniku omogućava jednostavan način mapiranja posebnih znakova (poput slova starogrčkoga alfabeta). Ovo se ostvaruje uređivanjem aplikacijske konfiguracione datoteke.
  - 2.2. **Konfiguracije vezane uz aplikaciju** omogućavaju korisniku da:
    - 2.2.1. **Podešava rad sa mrežnim dijelom sistema**, što podrazumijeva da korisnik može odrediti server na kojeg će *uploadovati* svoje dokumente. Ovo i nije funkcionalnost namijenjena korisnicima, već jednostavan način da se aplikaciju rekonfiguriše kroz njenu konfiguracionu datoteku u slučaju migriranja web servisa sa jednog na drugi domen.

### 3.3.2. Specifikacija sintakse

Kako se matematski izrazi unose jednako kao i obični tekst, definira se konkretna sintaksa razmatranog programskog rješenja koje pokriva tek nekoliko osnovnih operatora. Osnovni razlog tome je činjenica da su proširenja trivijalna za izvesti. Simbol za označavanje početka/kraja matematskog polja je '\$'. Simbol '\$' bez njegovog podrazumijevanog značenja piše se kao '\\$'.

Značenje	Template zapisa	Primjer	Render	Evaluacija?
<b>Sabiranje</b>	$expr + expr$	$a+b$	$a + b$	DA
<b>Oduzimanje</b>	$expr - expr$	$a-b$	$a - b$	DA
<b>Množenje</b>	$expr * expr$	$a*b$	$a \cdot b$	DA
<b>Dijeljenje</b>	$expr / expr$	$a/(b+c)$	$\frac{a}{b+c}$	DA
<b>Stepenovanje (također superskript)</b>	$expr ^ expr$	$a^2$	$a^2$	DA
<b>Subskript</b>	$a_b$	$x_1$	$x_1$	NE
<b>Logaritmovanje po bazi b</b>	$\log(expr)(b)$	$\log(100)(10)$	$\log_{10} 100$	DA
<b>Prirodni logaritam</b>	$\ln(expr)$	$\ln(100)$	$\ln 100$	DA
<b>Limes</b>	$\lim (a) (expr)$	$\lim (x \rightarrow \infty) (1/x)$	$\lim_{x \rightarrow \infty} \frac{1}{x}$	NE
<b>Neodređeni integral</b>	$\text{integral}(expr)$	$\text{integral}(x^2)$	$\int x^2$	NE

Tabela 2 - operatori koje MathEasy podržava i prepoznaje

Dodatno, MathEasy omogućava sljedeće funkcionalnosti sukladno datoj sintaksi:

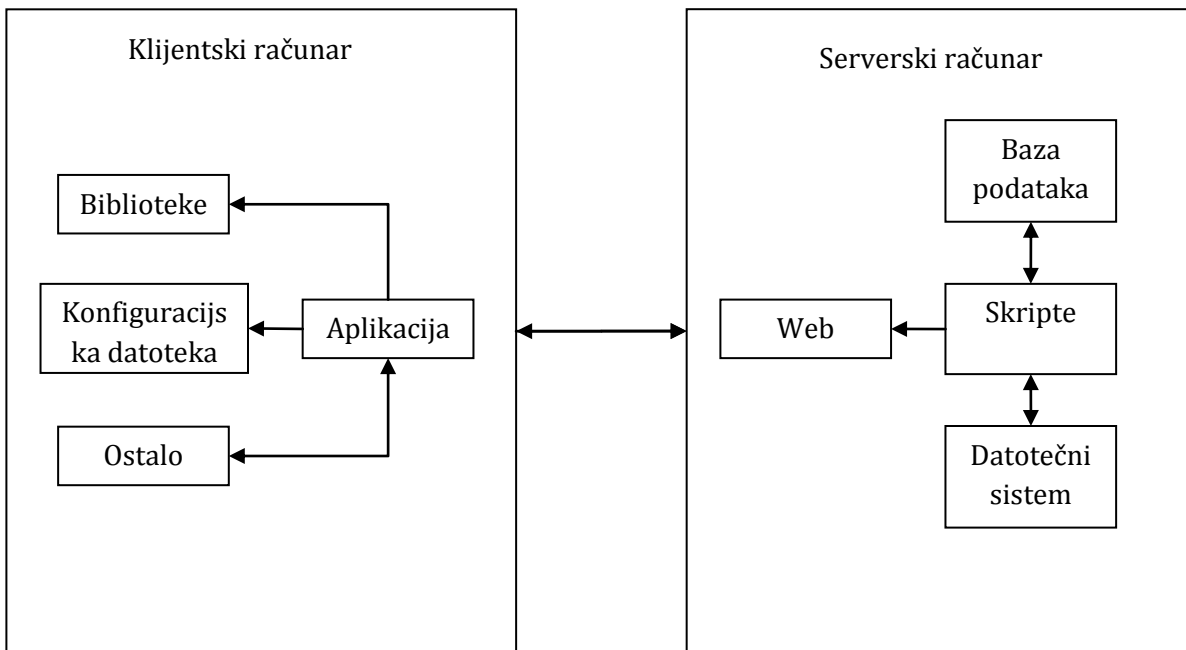
Opis	Sintaksa	Primjer	Rezultat
MathEasy omogućava definiranje simbola i dodjelu odgovarajuće vrijednosti. Pri tome ta vrijednost može biti brojeva konstanta ili izraz. U svim upotrebama u matematskom kontekstu simbol će imati zadatu mu vrijednost.	$\text{sym} = [..=..] expr$	$x=10+5=15$	U nastavku dokumenta, eval<x> će ispisati 15.
MathEasy omogućava trenutnu zamjenu simbola # prvim izrazom ispred prethodećeg mu znaka jednakosti. Ova mogućnost može poslužiti da se izbjegne proces selektiranja, kopiranja i lijepljenja odgovarajućeg plaintexta.	$[..=..] expr = \#$	$x+y=\#$	U trenutku unosa znaka hash (#), isti će biti zamijenjen tako da će u primjeru tekst biti $x+y=x+y$
MathEasy omogućava trenutnu zamjenu simbola ? rezultatom evaluacije prvog izraza ispred prethodećeg mu znaka jednakosti.	$[..=..] expr = ?$	$x=10, y=5, x+y=?$	U trenutku unosa znaka ?, isti će biti zamijenjen tako da će u primjeru tekst biti $x+y=15$
MathEasy omogućava evaluaciju proizvoljnog izraza, ukoliko ga je moguće evaluirati. Rezultat evaluacije vidljiv je isključivo u renderu dokumenta.	$\text{eval}<expr>$	$\text{expr}<10*5>$	U renderu dokumenta će umjesto $\text{eval}<10*5>$ pisati 50.
MathEasy omogućava crtanje grafika funkcija sa jednom promjenjivom, u datom opsegu i sa datim brojem tačaka. Grafik je vidljiv u renderu.	$\text{plot}<expr, od, do, brojačaka>$	$\text{plot} <x+5, 0, 5, 100>$	U renderu dokumenta će biti prikazan odgovarajući grafik.

Tabela 3 - funkcionalnosti koje MathEasy omogućava kroz uređivač



### 3.3.3. Arhitektura i komponente aplikacije

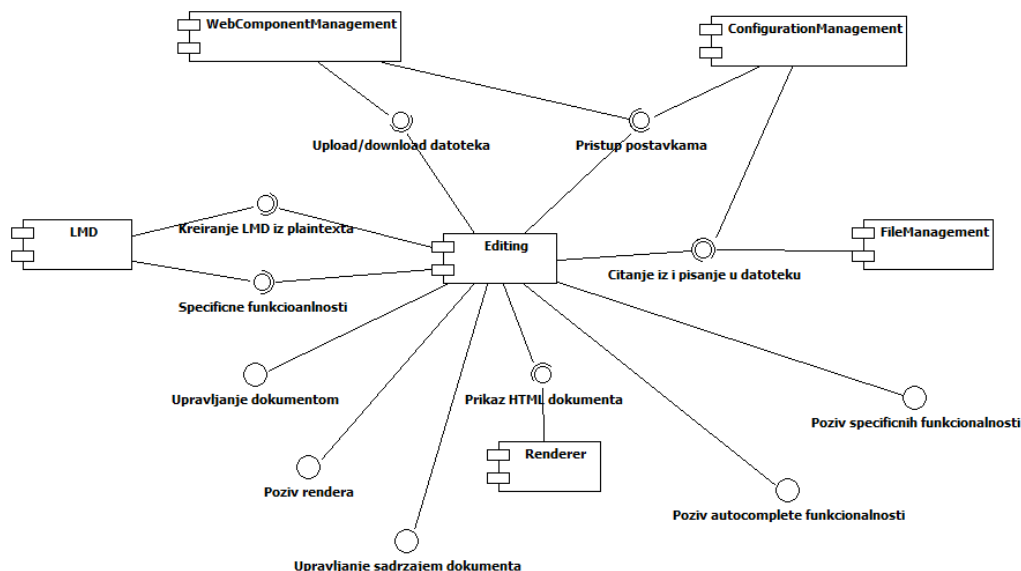
Arhitektura sistema kojeg čini MathEasy desktop aplikacija i odgovarajući web servis relativno je jednostavna i odgovarajući dijagram ne zahtijeva detaljnije obrazloženje:



Dijagram 5 - raspoređivanje komponenti sistema na klijentski i serverski računar

Uloga serverskog računara je da pohranjuje TXT i HTML dokumente koje mu klijenti pošalju uz odgovarajuću identifikaciju, kao i da kao povratnu informaciju šalje odgovarajuće jedinstvene pristupne URL-ove. Kasnije na zahtjeve od strane aplikacije vraća odgovarajući TXT, dok na zahtjeve kroz web preglednike prikazuje *renderirani* HTML dokument i linkove za preuzimanje bilo HTML bilo TXT verzije putem URL-a zatraženog dokumenta.

Osnovne komponente sistema, a na osnovu do sada izrečenog prikazane su na dijagramu br. 6. Nazivi komponenti dati su na engleskom jeziku zarad konzistentnosti, pošto autor rada implementaciju (programski kôd) piše engleskim jezikom.

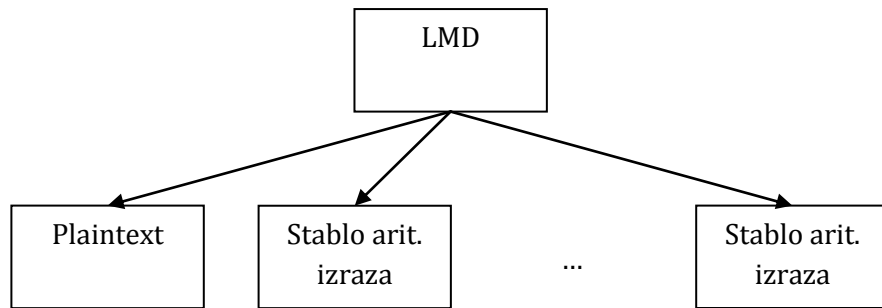


Dijagram 6 - komponente MathEasy aplikacije

Uloga svake od komponenti je jasna i sa dijagrama:

1. ConfigurationManagement komponenta služi za upravljanje postavkama aplikacije, odnosno učitavanje postavki iz konfiguracione datoteke ili učitavanje standardnih (engl. *default*) postavki ukoliko konfiguracione datoteke nema ili joj je sadržaj neispravan, kao i za omogućavanje ostalim komponentama sistema pristup postavkama.
2. WebComponentManagement komponenta služi za komunikaciju sa web servisom i pruža jednostavan interfejs za slanje i preuzimanje datoteka sa web servisa. U svom radu oslanja se na URL servisa kojeg joj omogućava ConfigurationManagement.
3. FileManagement komponenta služi kao omotač oko osnovnih I/O operacija u datotečnom sistemu lokalnog računara, tj. za čitanje i pisanje iz i u datoteke na lokalnom računaru. Konkretno se upotrebljava za učitavanje TXT datoteka i snimanje TXT i HTML datoteka.
4. Renderer komponenta služi za renderiranje i prikaz renderiranog HTML dokumenta.
5. Editing komponenta je komponenta kroz koju korisnik ostvaruje interakciju sa aplikacijom. Upotrebom Editor (Uređivača), korisnik kreira nove, otvara postojeće, spašava i mijenja sadržaj plaintext (TXT) dokumenata. Editing također implementira osnovnog pomoćnika (Helper) za *autocomplete* funkcionalnost. Editing komponenta sposobna je prepoznati ključne simbole i komande koji onda iniciraju interakciju između Editing i LMD komponenti.
6. LMD komponenta koja preuzima sadržaj kojeg joj obezbjeđuje Editing komponenta, pretvara ga u odgovarajući LMD (Lightweight Math Document) format koji će biti specificiran u nastavku, a na osnovu LMD zapisa ili obavlja specifične funkcije (poput evaluacije izraza) i vraća rezultate Editing komponenti, ili pretvara LMD zapis u odgovarajući HTML zapis koji se posredstvom Editing komponente proslijeđuje Rendereru na prikaz, ili FileManagement i WebComponentManagement komponentama na pohranu.

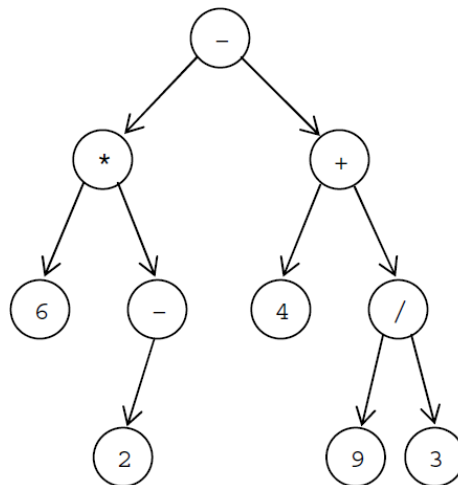
LMD format predstavlja strukturu koja postoji isključivo u radnoj memoriji MathEasy aplikacije tokom njenog rada, i to onda kada se ili obavlja evaluacija, ili obavlja konverzija u odgovarajući HTML dokument. LMD suštinski ne predstavlja ništa drugo do kombinaciju *plaintexta* i stabala aritmetičkih izraza. Pri tome ne postoji koncept paragrafa niti blokova teksta: novi redovi su sadržani u plaintext čvorovima, dok stabla aritmetičkih izraza ignoriraju nove redove:



Dijagram 7 - LMD format

Stablo aritmetičkog izraza je struktura podataka koja opisuje aritmetičke izraze na prirodan način koji omogućava njihovu jednostavnu interpretaciju i efikasno manipuliranje sa njima. Stablo aritmetičkih izraza sadrži različite tipove čvorova. Jedna vrsta čvorova odgovara operandima poput konstanti ili promjenjivih koji se ne mogu razbiti na prostije dijelove [...] Druga vrsta čvorova odgovara operatorima i oni imaju svoje potomke [24].

Primjerice, stablo aritmetičkog izraza za izraz  $6 \cdot (-2) - (4 + \frac{9}{3})$  je dato na dijagramu broj 8 [3]:



Dijagram 8 - primjer stabla aritmetičkog izraza

LMD služi kao posrednik između *plaintexta* sa kojim radi korisnik i odgovarajućeg *rendera* tog *plaintexta* sa prepoznatim svim matematičkim objektima i obavljenim svim naredbama, ali i način na koji se ostvaruju specifične funkcije navedene u tabeli br. 3: trenutna zamjena određenih znakova cijelim prethodnim ili evaluacija cijelog prethodnog izraza.

Nakon specificiranja osnovnog formata zapisa kojeg korisnik kreira (*plaintext*) i posebnih sekvenci znakova koje se zasebno interpretiraju (*tabele br. 2 i 3*), sljedeći korak je specificirati i način na koji će se iz *plaintexta* očitani LMD dokument *renderirati*. Već je rečeno da će se to raditi kroz standardni HTML dokument, no sada će se to detaljno specificirati.

Naime, LMD struktura je pogodna da se upotrebom rekurzije kreira odgovarajući HTML kôd kojeg je onda jednostavno *renderirati*. Očigledno, *plaintext* čvorovi će se direktno ispisivati u HTML uz iznimku da će se eksplicitni prelasci u nove redove prevoditi kao odgovarajući "`<br/>`" tagovi, budući da je već ranije navedeno da LMD, kao ni odgovarajući uređivač ne podržavaju paragrafe kao blokove teksta, tako da se u rezultujućem HTML-u neće pojavljivati paragrafi i odgovarajući im HTML tagovi ("`<p> ... </p>`").

Što se tiče *renderiranja* stabala aritmetičkih izraza, to je jednostavno za izvesti primjenom rekurzije koja će izgraditi odgovarajući LaTeX kôd i ograditi ga najavom početka i kraja tog izraza.

Pri tome je nužno u zaglavlje (engl. *head*) rezultujućeg HTML dokumenta ("`<head> ... </head>`" tagovi) uključiti poveznice na URL-ove datoteka odgovarajućih klijentskih skripti koje su zadužene za *rendering* izraza i grafika u web pregledniku. Također je potrebno obezbijediti da HTML dokument koristi predefinirani font koji nužno mora sadržavati posebne simbole poput često korištenih  $\rightarrow$ ,  $\cdot$ ,  $\alpha$ ,  $\beta$ . Rezultujući HTML mora biti validan.

Da bi se obezbijedilo da se aplikaciju može regularno koristiti i u situaciji kada nije raspoloživa internet konekcija, potrebno je uz aplikaciju priložiti odgovarajući *JavaScript rendering engine* koji će se koristiti prilikom *renderiranja* u okviru same aplikacije.

### 3.3.4. Aktivnosti

Nakon što su date odgovarajuće specifikacije, moguće je preći na detaljnije razmatranje funkcija sistema. Pri tome su neke od funkcija trivijalne i neće im se posvećivati posebna pažnja osim kraćeg tekstualnog opisa. Funkcije su grupirane po njihovoj važnosti i međusobnoj povezanosti, tako da se kreće od temeljnih funkcionalnosti.

#### 1. Učitavanje sadržaja TXT datoteke u informacijski tok

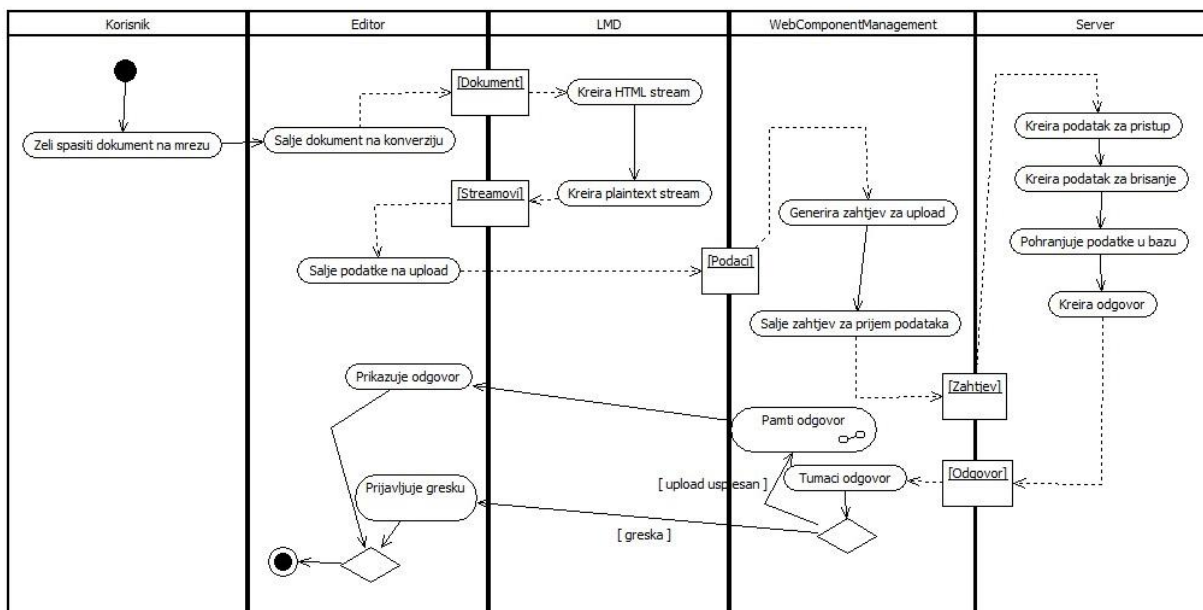
Ova funkcionalnost poziva se onda kada se želi učitati sadržaj datoteke koja se nalazi na lokalnom računaru u program, tj. konkretno za učitavanje sadržaja konfiguracione datoteke (ako ona postoji) ili nekog dokumenta. Funkcionalnost se realizira na način da se detektira nepostojanje datoteke, nemogućnost pristupa istoj ili eventualna narušenost njenog sadržaja, kao i greške u samoj ulaznoj operaciji. Aktivnost ima trivijalan tok pa se stoga neće prikazati odgovarajućim dijagramom.

#### 2. Ispis informacionog toka u TXT ili HTML datoteku

Ova funkcionalnost poziva se onda kada se želi snimiti sadržaj u TXT ili HTML datoteku. U oba slučaja radi se nizovima znakova, jedina razlika je u ekstenziji koju će kreirana datoteka imati. U oba slučaja enkodiranje je UTF-8. Funkcionalnost se realizira na način da se detektira nemogućnost kreiranja datoteke na željenoj lokaciji, obavlja prepisivanje već postojeće istoimene datoteke, kao i greške u samoj izlaznoj operaciji. Aktivnost ima trivijalan tok pa se stoga neće prikazati odgovarajućim dijagramom.

#### 3. Upload TXT i HTML datoteke na mrežni servis

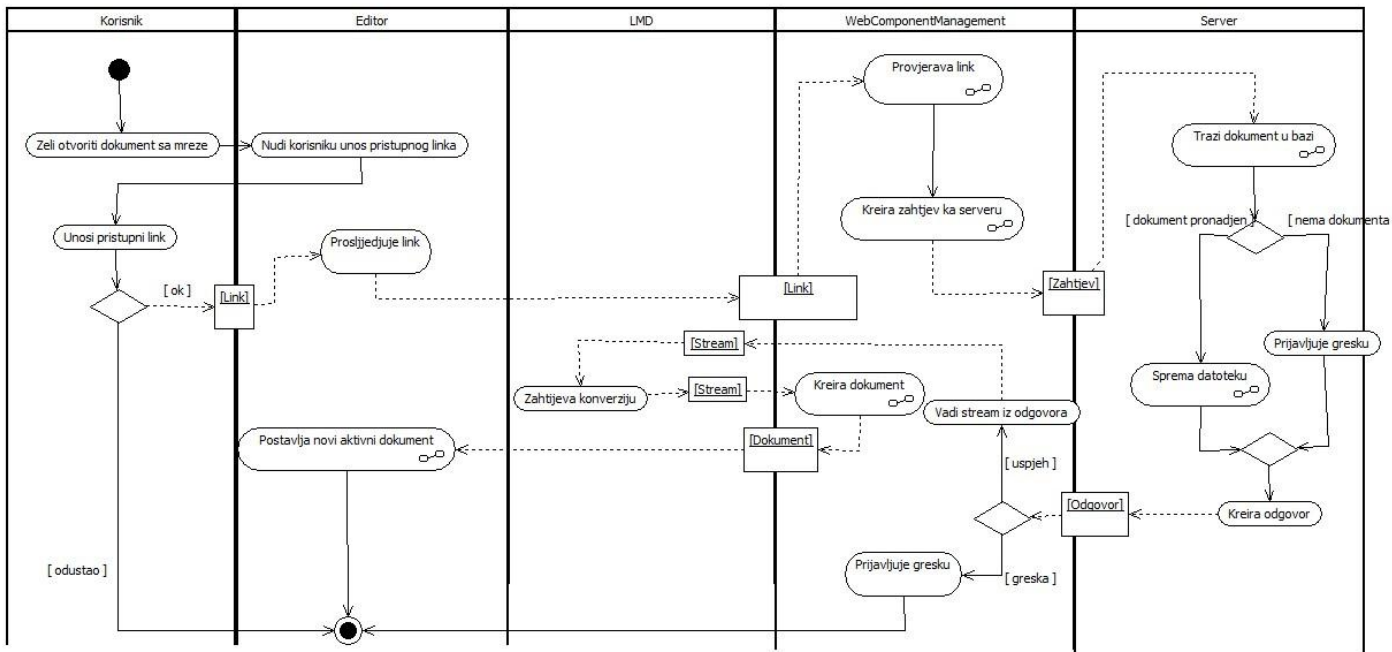
Ova funkcionalnost omogućava korisniku da trenutno aktivni dokument spasi na mrežni servis, kao i da mu kasnije može pristupiti na osnovu javno dostupnog URL-a koji će se korisniku vratiti u slučaju uspješnog *uploada* datoteke na mrežni servis. U suprotnom, aplikacija će prijaviti grešku. Tok aktivnosti dat je na dijagramu br. 9.



Dijagram 9 - upload TXT i HTML datoteke na mrežni servis

#### 4. Download TXT datoteke sa mrežnog servisa

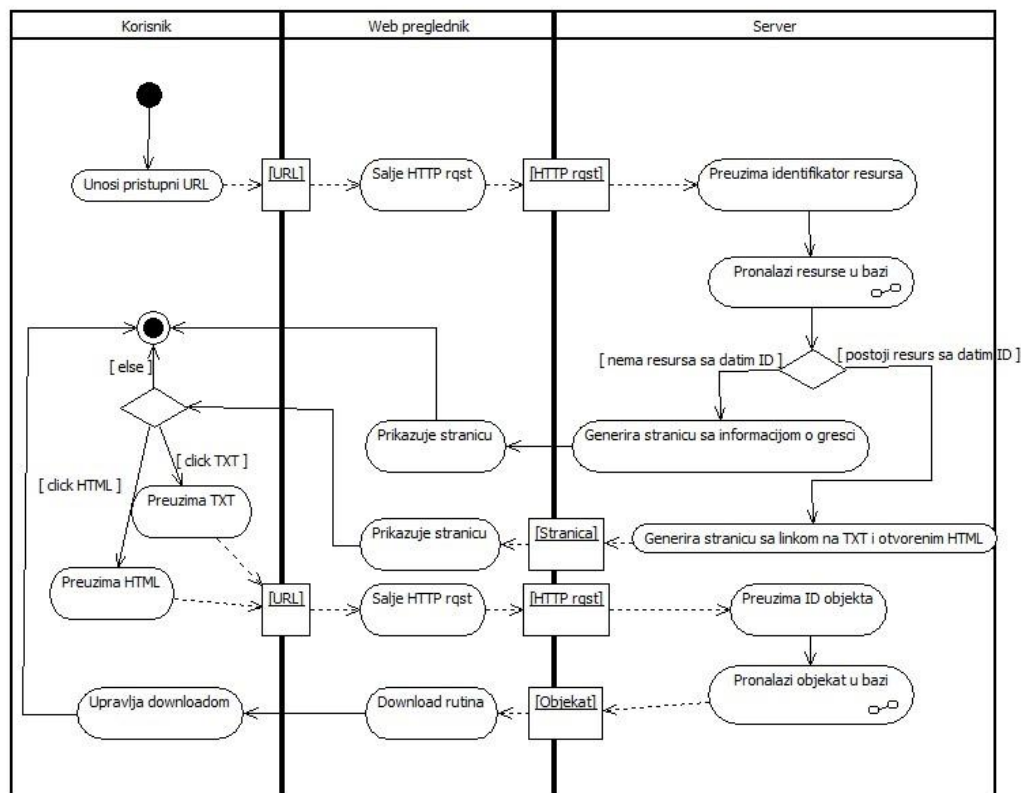
Ova aktivnost omogućava korisniku da unosom odgovarajućeg URL-a u trenutnoj radnoj površini aplikacije dobije dokument sa mrežnog servisa identificiran na osnovu datog URL. Ukoliko URL nije valjan ili servisu nije moguće pristupiti, aplikacija treba prijaviti grešku. Pri tome je nužna podaktivnost koja podrazumijeva upit korisniku šta želi uraditi sa dokumentom trenutno aktivnim u okviru radnog prostora. Tok aktivnosti dat je na dijagramu br. 10.



Dijagram 10 - download TXT datoteke sa mrežnog servisa

#### 5. Pregled dokumenta putem web servisa

Ova aktivnost omogućava korisnicima da pristupe prethodno na servis postavljenom dokumentu bez da na svom računaru moraju imati instaliran MathEasy. Sve što je potrebno jeste posjedovati validan URL i web preglednik, kao i pristup internetu. Postavljeni dokument se prikazuje kao HTML dokument i omogućeno je preuzimanje tog HTML-a ili odgovarajućeg *plaintexta* iz kojeg je HTML generiran. Tok aktivnosti dat je na dijagramu br. 11.



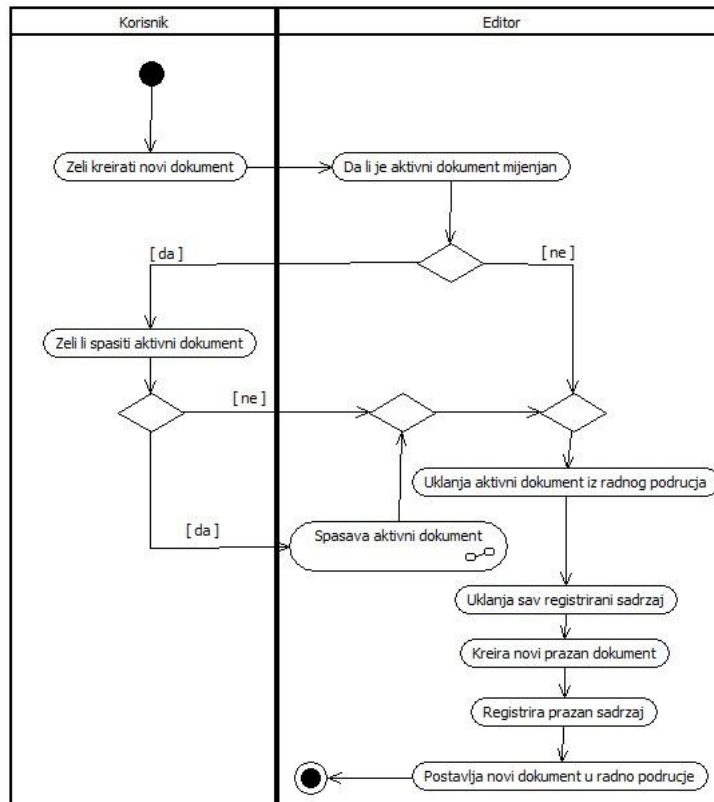
Dijagram 11 - pregled dokumenta putem web servisa

## 6. Upravljanje postavkama aplikacije

Za promjenu postavki aplikacije (simbola koji se ubacuju u dokument i niza znakova koji su im odgovarajući- *autocomplete parovi*, kao i domen iza kojeg se nalazi odgovarajući servis) odgovaran je korisnik koji izmjene obavlja nad UTF-8 enkodiranom konfiguracijskom datotekom koju ili automatski sa predefiniranim vrijednostima kreira i popunjava aplikacija prilikom pokretanja, ili iz postojeće i ispravno formatirane datoteke učitava njen sadržaj kao postavke. Pri tome je format datoteke sljedeći: prvi podatak je domensko ime sa brojem porta i dato je u formatu npr. "matheasy.ba:8080", bez razmaka ispred, u okviru ili iza domenskog imena. Preostali podaci su u obliku u kojem se u svakoj novoj liniji navodi simbol i jednim razmakom odvojen niz znakova (npr. "α alfa"). Razmaci nisu dozvoljeni. Aktivnost ima trivijalan tok (otvoriti datoteku kroz npr. Notepad, izmijeniti njen sadržaj i spasiti promjene) pa se stoga neće prikazati odgovarajućim dijagramom.

## 7. Kreiranje novog dokumenta (New)

Ova funkcionalnost intuitivno je jasna: korisnik ima mogućnost kreirati novi prazan dokument i učitati ga u radni prostor. Pri tome se korisniku mora postaviti upit o željenom toku akcije sa dokumentom koji već postoji u radnom prostoru (npr. spasiti ga u neku datoteku ili ga *uploadovati* na web servis, ili ga jednostavno obrisati bez spašavanja). Tok aktivnosti dat je na dijagramu br. 12.

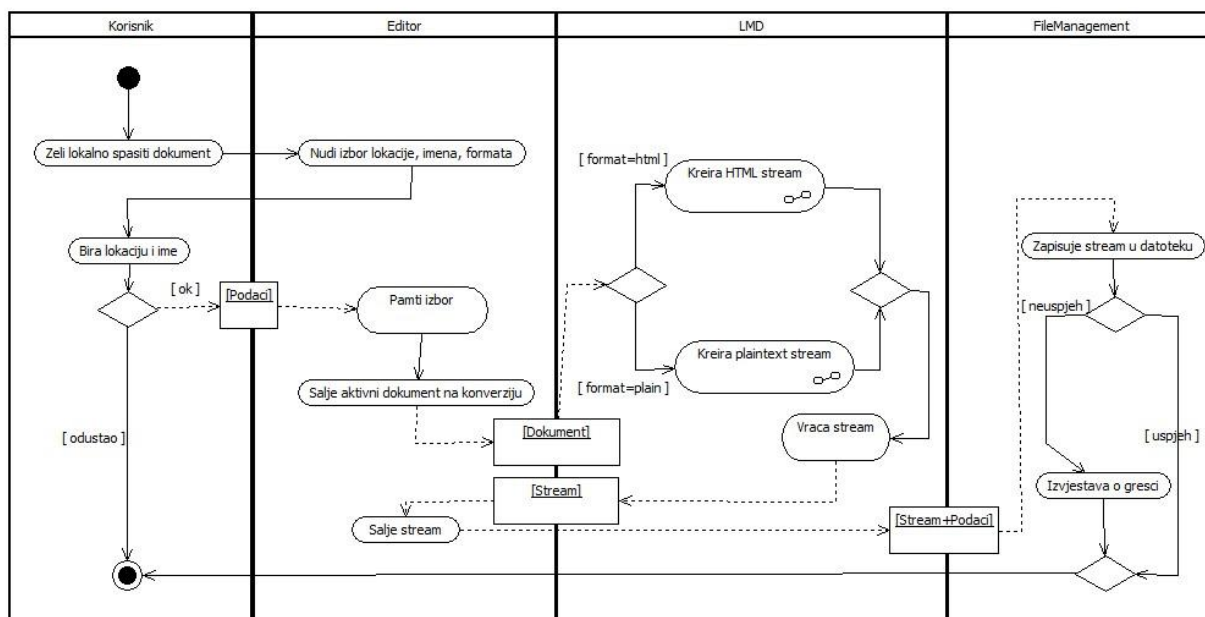


Dijagram 12 - kreiranje novog dokumenta

## 8. Spašavanje trenutnog dokumenta na datu lokaciju i u datom formatu (Save as...)

Navedeni način spašavanja trenutnog dokumenta odnosi se na mogućnost koja se pruža korisniku da spasi dokument iz radnog prostora u specficiranu *plaintext* (TXT) ili HTML datoteku na lokalnom računaru. Nudi se mogućnost određivanja imena, lokacije i tipa datoteke u koju će se spašavanje obaviti. Tok aktivnosti dat je na dijagramu br. 13.





Dijagram 13 - spašavanje trenutnog dokumenta

## 9. Spašavanje trenutnog dokumenta (Save)

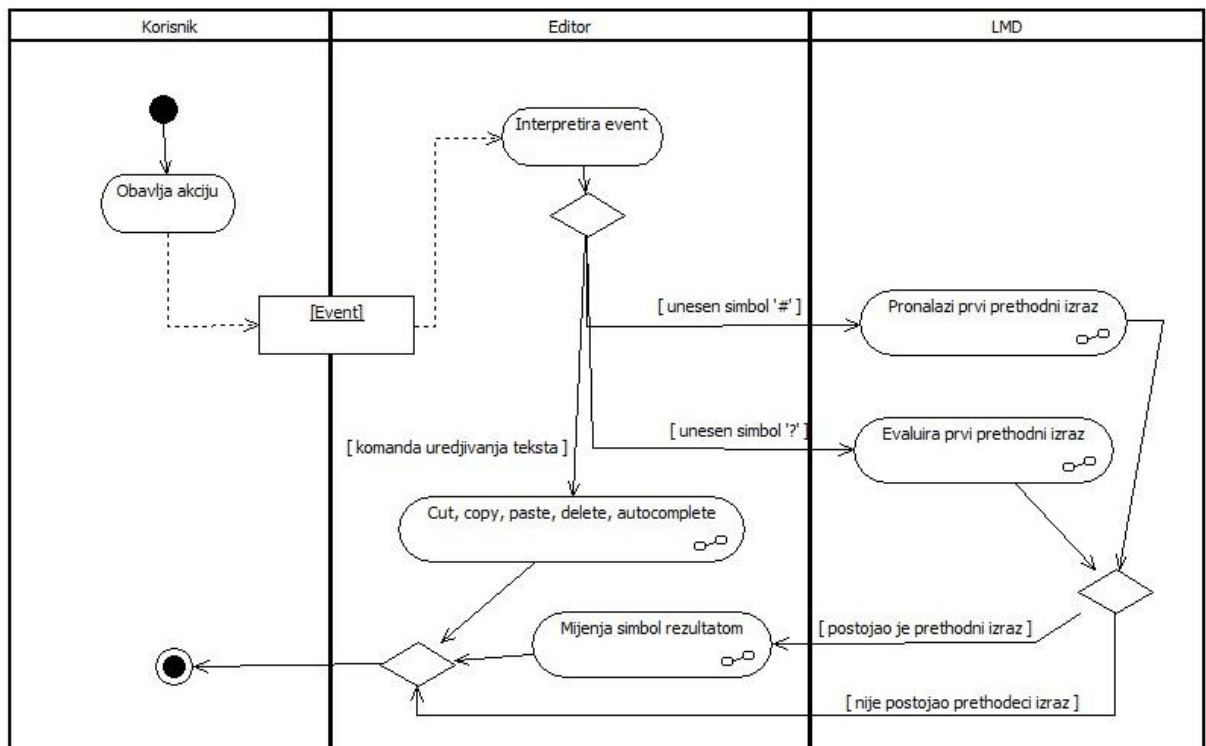
Navedeni način spašavanja trenutnog dokumenta odnosi se na mogućnost koja se pruža korisniku da spasi dokument iz radnog prostora u specificiranu *plaintext* (TXT) ili HTML datoteku na lokalnom računar. Pri tome, ukoliko je trenutni radni dokument već spašen, sljedeća uzastopna spašavanja automatski će se odnositi na istu datoteku koja je inicijalno spašena. Ukoliko trenutni radni dokument nije već prethodno spašen, nudi se mogućnost određivanja imena i lokacije datoteke u koju će se spašavanje obaviti. Ova aktivnost razlikuje se od prethodno navedene (*Save as..*) samo po jednoj dodatnoj podaktivnosti na samome početku gdje se korisniku nude opcije za upravljanje dokumentom koji se trenutno nalazi u radnom prostoru, pa se stoga neće prikazivati odgovarajućim dijagramom.

## 10. Učitavanje dokumenta iz date datoteke (Open)

Navedena aktivnost podrazumijeva omogućavanje upravljanja dokumentom koji se trenutno nalazi u radnom prostoru, kao i učitavanje sadržaja TXT dokumenta kojeg je korisnik odabrao. Aktivnost ima trivijalan tok pa se stoga neće prikazati odgovarajućim dijagramom.

## 11. Uređivanje sadržaja dokumenta

Uređivanje sadržaja dokumenta kroz MathEasy aplikaciju ni po kojem osnovu se ne razlikuje od rada sa najjednostavnijim uređivačima teksta (poput *Notepada* i *Gedita*). Pri tome jedine specifičnosti leže u tome da se ključne riječi i simboli koje MathEasy prepoznaje automatski ističu u odnosu na ostali tekst, da korisnik može inicirati *autocomplete* funkcionalnost, kao i da kucanje simbola "#" i "?" u posebnom kontekstu kako je definirano u tabeli broj 3 omogućava da se ti simboli automatski zamijene prethodnim izrazom ili njegovom evaluacijom. Dodatno, unos simbola "\" aktivira tzv. pomoćnika koji nudi izbor između različitih simbola koji su definirani kroz konfiguracijsku datoteku. Tok jedne osnovne interakcije korisnika sa uređivačem dat je na dijagramu br. 14.



Dijagram 14 - uređivanje sadržaja dokumenta

## 12. Rendering dokumenta

*Rendering* dokumenta podrazumijeva da korisnik želi vidjeti kako dokument izgleda jednom kada je njegov sadržaj unesen. U pozadini se obavlja generiranje HTML-a na osnovu sadržaja dokumenta, a potom se generirani HTML ubacuje kao web stranica u ugrađeni web preglednik. Aktivnost ima trivijalan tok pa se stoga neće prikazati odgovarajućim dijagramom.

### 3.3.5. Klase i algoritmi

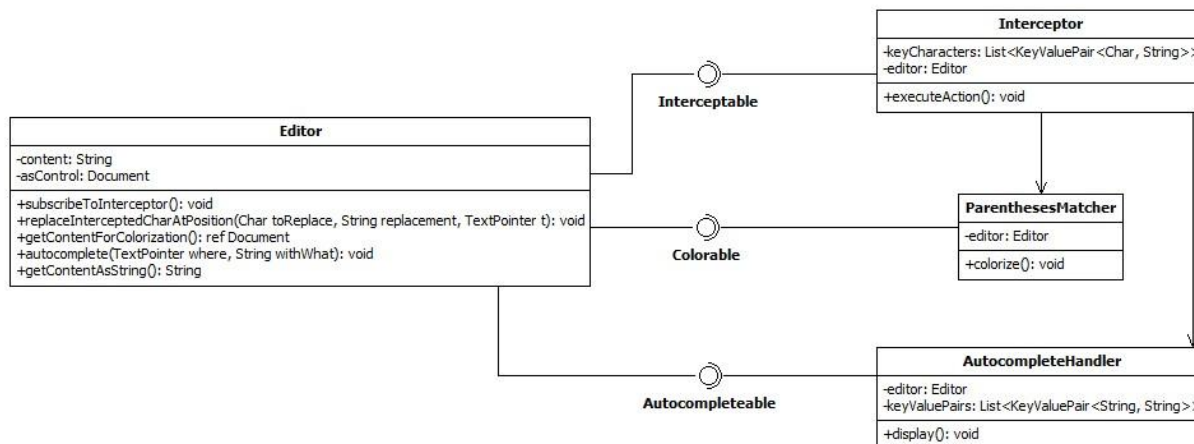
U ovom odjeljku se opisuje logička i procesna (dinamička) struktura MathEasy aplikacije, gdje su najsloženiji dijelovi LMD i Editing komponente aplikacije. Zbog toga se navedenim komponentama posvećuje posebna pažnja. Preostale komponente (FileManagement, WebComponentManagement, ConfigurationManagement, Rendering) se najvećim dijelom oslanjaju na upotrebu već gotovih metoda, objekata ili drugih komponenti (jasno, uz određene izmjene i prilagodbe) koje su prisutne u svim razvojnim okruženjima koja su realan izbor za implementaciju jedne ovakve aplikacije, pa kao takve nisu detaljnije razmatrane niti opisivane. Odgovarajući opisi dati su kroz *dijagrame klasa* sa odgovarajućim obrazloženjima, kao i kroz objašnjenja odgovarajućih algoritama datih u *pseudokodu*. Prva komponenta koja se razmatra je Editing. Upotrebom elemenata navedene komponente korisnik direktno ostvaruje interakciju sa aplikacijom. Sve ostale komponente upravljane su korisničkim akcijama detektiranim kroz elemente Editing komponente, ili funkcionalnosti Editing komponente predstavljaju važan dio u okviru korisničkih akcija na nivou aplikacije (npr. Save funkcionalnost koja sama po sebi nije uređivanje dokumenta, ali zahtijeva sadržaj dokumenta).

Osnovni element (u biti klasa) Editing komponente je uređivač teksta kojeg se realizira kao grafičku kontrolu za unos teksta (naziv koji se najčešće koristi u praksi je TextBox, dok će se za potrebe ovog dokumenta koristiti naziv Editor ili uređivač). Uređivač je u stanju registrirati događaje (korisničke akcije nad elementom), a aplikacija ih je u stanju analizirati i donijeti odgovarajuće odluke.

Sljedeći element kojeg treba razmotriti implementira funkcionalnost analize i donošenja odgovarajuće odluke, kako bi uređivaču ostala jednostavna funkcionalnost implementacije odluke. Odgovarajući element očito služi kao presretač događaja, pa ga se tako i naziva *Interceptor* ili *presretač*. Postavlja se pitanje kako ovaj element radi, tj. šta su to događaji koje presreće? Naime, uređivač jedino što detektira (a da je relevantno za aplikacijsku logiku koja se implementira) jeste izmjena teksta - sadržaja uređivača. Moguće je "snimiti" izmjenu prije nego se ona desi, tj. detektirati namjeru da se izmjena desi ili rezultat tj. izmjenu koja se već desila. Iz tabele br. 3 jasno je da unos određenih karaktera ('?' i '#') treba presresti i obaviti određenu izmjenu nad sadržajem, ukoliko je ona moguća. Dodatno, zahtijeva se i *autocomplete* funkcionalnost koja se inicira unosom karaktera '\'. Presretač ima zadatak detektirati unos navedenih karaktera prije nego se oni dodaju na sadržaj uređivača, a potom inicirati odgovarajuće akcije. Također, unos bilo kakvih zagrada (otvorene i zatvorene male, srednje i velike), kao i rezanje, brisanje ili lijepljenje teksta u okviru uređivača treba rezultirati provjerom uparenosti zagrada u sadržaju dokumenta i njihovim odgovarajućim bojenjem kako bi se korisniku olakšalo snalaženje u okviru *plaintext* dokumenta.

*Autocomplete* funkcionalnost realizira se kroz odgovarajući element koji se aktivira po detekciji unosa specifičnog karaktera. Oslanja se na preuzimanje baze *autocomplete* parova koji su prisutni u postavkama aplikacije. Korisnik može ignorirati ponuđene opcije ukoliko je samo želio unijeti karakter '\' u dokument, ili odabrati jednu od opcija (zamjenskih karaktera). Izborom opcije sprječava se unos karaktera '\' i svih naknadno unesenih karaktera u dokument, i umjesto njih se na datu poziciju u dokumentu ubacuje odabrani zamjenski karakter (ili više njih).

Prethodno navedeno grafički je prikazano na dijagramu br. 15.



Dijagram 15 - klase Editing komponente

Klasa Editor implementira interfejse Interceptor, Colorable i Autocompleteable, što efektivno označava da je akcije nad Editorom moguće presretati, da je pojedine dijelove teksta (u ovom slučaju konkretno zagrade, ali i otvaranja/zatvaranja matematičkih konteksta simbolom \$) moguće bojiti po određenim pravilima, kao i da određeni unosi u Editor pokreću *autocomplete* funkcionalnost. Editor se sa odgovarajućim klasama spaja na način da se prilikom svog instanciranja pretplaćuje na njihove usluge preko presretača koji kao svoj atribut dobija referencu na instancirani Editor. Dodatno se uvezuju odgovarajući događaji (akcije) koje je Editor u stanju prepoznati i automatski proslijediti presretaču.

Nakon što je uvezivanje završeno, svaka korisnička akcija nad Editorom automatski se prosljeđuje Interceptoru na tumačenje, gdje će Interceptor uvezati Editor sa odgovarajućom komponentom koja će izvršiti potrebne akcije. Ono što nije prikazano na dijagramu br. 15 je povezivanje Interceptora sa LMD funkcionalnostima, a koje je nužno da bi funkcionalnosti iz tabele br. 3 bile implementirane. Razlog za to je činjenica da još nije opisana LMD komponenta.

Druga komponenta koju se razmatra je LMD komponenta koja čini jezgro aplikacije. Elementi (klase) LMD komponente mogu se podijeliti u dvije grupe (paketa ili imenska prostora):

- prva, čije klase su sadržaj LMD dokumenta (dokument, *plaintext*, *mathtext* kao *plaintext* protumačen u matematičkom kontekstu, *commandtext* za specifične eval i plot funkcije)
- druga, čije klase čine strukturu stabla aritmetičkog izraza (tokeni koji se organiziraju u stablo)

Jasno, svaka grupa pored osnovnih sadrži i pomoćne klase koje implementiraju određene specifične funkcionalnosti. Grupe su nazvane LMD.Core (prva) i LMD.Math (druga) i prva ovisi o drugoj.

Prije dalje razrade logičke strukture LMD komponente potrebno je prvo definirati kako će se i za što će se ova komponenta koristiti. Na osnovu date specifikacije, jasno je da se LMD komponenta koristi u dva slučaja:

- kada se želi *renderirati* finalni izgled dokumenta. Tada se *plaintext* dokument pretvara u *posrednički* LMD dokument. Pretvaranje *plaintexta* u LMD podrazumijeva i obavljanje eventualnih izračuna komandi. Iz rezultirajućeg LMD jednostavno je kreirati odgovarajući HTML kôd kojeg je moguće *renderirati* kroz web preglednik.
- kada korisnik pozove specifičnu funkcionalnost, kao što je evaluacija prvog prethodnog izraza ili lijepljenje prvog prethodnog izraza. Tada se iz *plaintexta* generira LMD, obavlja se specifična akcija i njen rezultat se vraća na mjesto poziva.

Time su i praktično određene dvije osnovne funkcionalnosti LMD dokumenta: generiranje adekvatne strukture iz *plaintexta* i obavljanje specifičnih funkcionalnosti poput evaluacije i lijepljenja izraza.

Kreiranje LMD dokumenta iz običnog neformatiranog teksta nije trivijalan zadatak: potrebno je prepoznati odgovarajuće uzorke u tekstu koji odgovaraju onome što je specifično za mogućnosti aplikacije vezane uz matematiku: deklariranje simbola, izrazi i njihova struktura i slično, a potom prepoznate uzorke "razbiti" u stabla aritmetičkih izraza kako bi se sa njima moglo raditi (evaluirati ih).

Dodatno, potrebno je održavati određeni vid tabele u kojoj će se javljati svi simboli sa njihovim vrijednostima- gdje vrijednosti mogu biti konkretne (npr.  $x=15$ ) ili simbol može stajati za cijeli izraz (npr.  $x=y^2+7$ ). Na taj način će biti moguće korektno evaluirati izraze kada se za tim pokaže potreba.

Kreiranje LMD dokumenta iz neformatiranog teksta odvija se u više faza, dok je pretvaranje u odgovarajući HTML dokument jednostavno izvesti upotrebom rekurzije za izraze, a običnim mapiranjem za tekst.

1. Prvi korak jeste na ulaz prevoditelja dobiti niz znakova koji predstavlja *plaintext* dokument.
2. U sljedećem koraku prevoditelj u okviru datog niza znakova obavlja prepoznavanje uzoraka teksta koje je moguće protumačiti kao matematičke izraze ili posebne naredbe (tabela br. 3). Odgovarajući uzorci pohranjuju se u objekte LMD dokumenta.
3. Nakon što su uzorci prepoznati, pokreće se faza izgradnje odgovarajućih stabala aritmetičkih izraza za objekte LMD dokumenta koji predstavljaju (tj. za koje je prepoznato da odgovaraju uzorku) matematičkih izraza ili posebnih naredbi. Kreće se od prvog ka posljednjem objektu. Ukoliko je uspješno izgrađeno stablo izraza, stablo se dodjeljuje kao atribut odgovarajućeg objekta. U suprotnom, izraz se konvertuje u *plaintext* objekat jer neispravno zapisan izraz nije moguće na odgovarajući način *renderirati* bez sintaksnih grešaka.
4. Nakon što su sva validna stabla izgrađena, pokreće se izgradnja tabele simbola sa pripadajućim vrijednostima. Simboli se sa vrijednostima povezuju znakom '='. Pri tome, ukoliko je vrijednost simbola data izrazom, vrši se njegova evaluacija.

5. Nakon što su svi simboli zamijenjeni odgovarajućim vrijednostima ili evaluiranim izrazima, obavljaju se finalne evaluacije i izvršavanje zadatih naredbi koje se interpretiraju ako se nalaze u odgovarajućem kontekstu (primjerice, '?' kao dio rečenice neće se interpretirati, dok '?' kojem prethodi izraz praćen znakom '=' biti interpretiran tako da će se odrediti evaluirana vrijednost prethodnog izraza).

Konkretna način izvođenja navedenih koraka može se vidjeti na sljedećem primjeru:

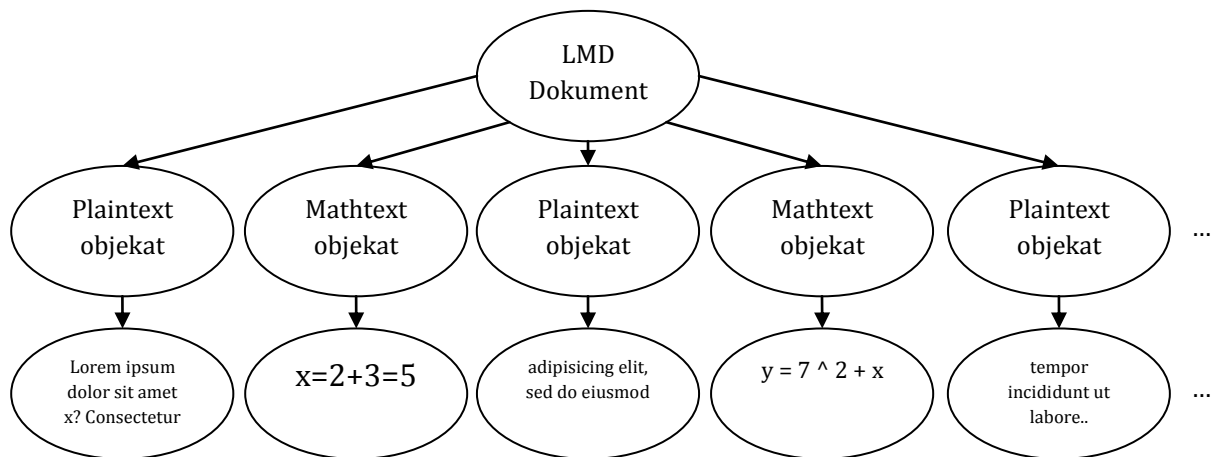
1. *Plaintext* oblik:

Lorem ipsum dolor sit # amet x? Consectetur \$x=2+3=5\$ adipisicing elit, sed do eiusmod \$y = 7 ^ 2 + x\$ tempor incididunt ut labore et dolore \$z = x + y\$ magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

2. Prepoznavanje uzoraka:

Lorem ipsum dolor sit amet x? Consectetur
 $x=2+3=5$ 
adipisicing elit, sed do eiusmod
 $y = 7 ^ 2 + x$   
 $2 + x$ 
tempor incididunt ut labore et dolore
 $z = x + y$ 
magna aliqua. Ut enim ad minim  
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Odgovarajući LMD dokument:



Dijagram 16 - dio strukture LMD dokumenta za dati primjer

3. Gradnja stabala aritmetičkih izraza

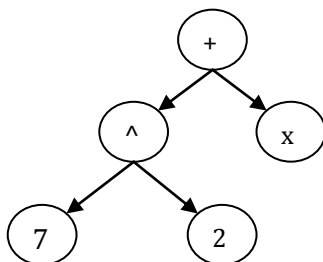
U ovom koraku od interesa su samo *Mathtext* objekti. Svaki objekat se zasebno analizira. Kao primjer će se uzeti " $x=2+3=5$ ".

U prvoj fazi prvo se identificiraju znaci jednakosti kako bi se *plaintext* dekomponirao u sastavne dijelove (izraze):

$x = 2 + 3 = 5$

Potom se obavlja *tokenizacija* ili finalni korak dekomponiranja izraza u njegove sastavne dijelove (varijable, konstante i operatore). U konkretnom primjeru, *tokenizacijom* izraza ' $2+3$ ' dobijaju se sljedeći tokeni: ' $2$ ' tipa konstantne vrijednosti 2, ' $+$ ' tipa operatora sabiranja i ' $3$ ' tipa konstantne vrijednosti 3:

Konačno, kada je izvršena *tokenizacija* svakog od (u ovom primjeru) tri izraza, kreće se sa izgradnjom odgovarajućih stabala aritmetičkih izraza. Gradnja stabala obavlja se *Shunting-yard* algoritmom modificiranim za potrebe elegantnog rješavanja ispisa rezultirajućeg LaTeX kôda. Objašnjenje modifikacija dato je u nastavku ovog odjeljka. Rezultirajuće stablo npr. za izraz  $7^2 + x$  ima sljedeću strukturu:



Dijagram 17 - stablo aritmetičkog izraza za dati primjer

#### 4. Gradnja tabele simbola

U ovom koraku se posjećuju svi matematički objekti i prepoznaju se simboli i njihove vrijednosti. Simbol je elementarni izraz (atom) kojeg čini niz znakova bez razmaka i obavezno počinje slovom. U datom primjeru, tabela simbola ima sljedeći oblik nakon prolaska kroz cijeli dokument:

Simbol	Vrijednost
x	5
y	$7^2 + x = 49 + 5 = 54$
z	$x + y = 5 + 54 = 58$

Tabela 4 - tabela simbola i vrijednosti za dati primjer

#### 5. Finalna evaluacija i oblik dokumenta

Pod posljednjom evaluacijom podrazumijeva se da se tamo gdje se zahtijeva evaluacija izraza bilo za *rendering* (eval), bilo za potrebe rada na dokumentu ('?' komanda). U evaluiranom izrazu simboli se zamjenjuju vrijednostima i izraz se ponovo evaluira dok dvaput uzastopno rezultat evaluacije ne bude identičan. Isto se odnosi i na funkciju crtanja grafa, s tim da se provjerava da li konačni izraz sadrži maksimalno jedan simbol bez vrijednosti. Ako je to slučaj, uzimaju se ostali parametri i generiraju parovi tačaka koje će biti iscrtane jednom kada se dokument *renderira*.

Od interesa su algoritmi za prepoznavanje uzoraka izraza i naredbi, *tokenizaciju* i gradnju stabla aritmetičkog izraza.

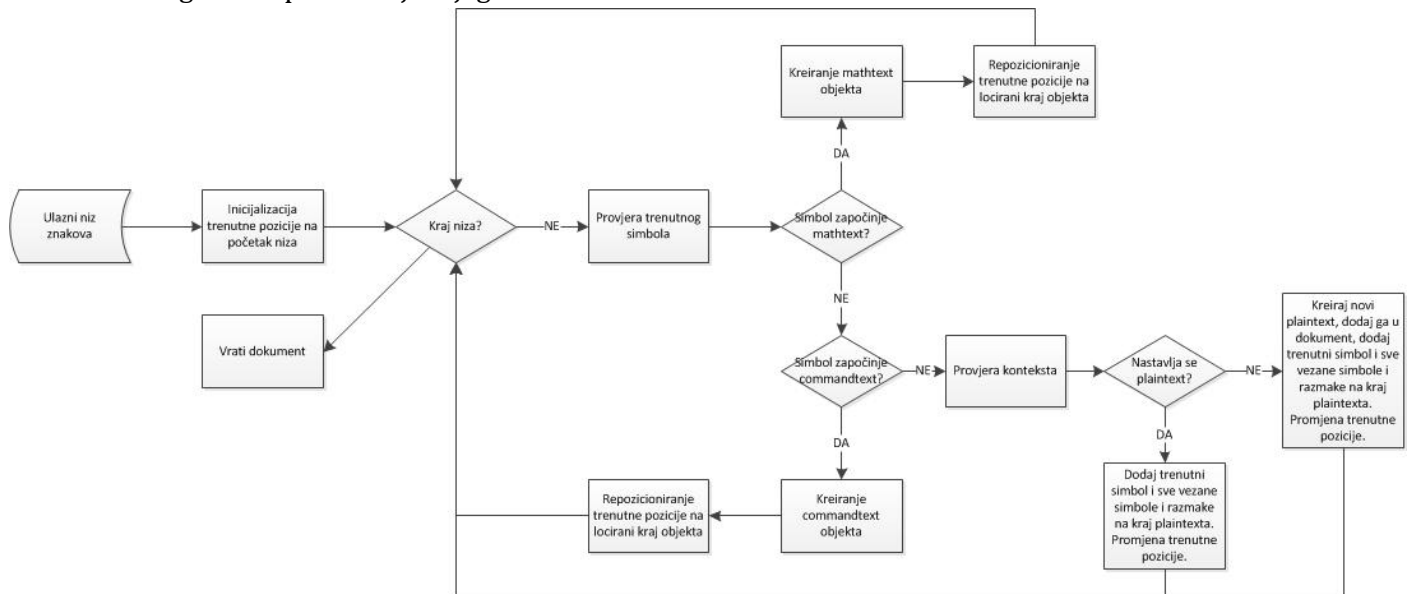
1. Algoritam za prepoznavanje uzoraka kao svoj ulaz prima obični neformatirani tekst, a na svom izlazu daje LMD dokument sa identificiranim dijelovima teksta koje treba tretirati kao obični tekst, dijelovima koje treba tretirati kao matematičke izraze (*mathtext*) i dijelovima koji predstavljaju specijalne naredbe (*eval*, *plot*) . Pri tome se prepoznaje samo *uzorak*, ali ne i to da li je izraz smislen. Rezultirajući LMD dokument sastoji se iz plaintext, mathtext i commandtext LMD objekata.

Postoje i određene pretpostavke kojima se ovaj algoritam vodi:

- o korisnik je eksplicitno označio područja u tekstu koja želi da se interpretiraju i adekvatno *renderiraju*. Sukladno specifikaciji datoj ranije, to se obavlja simbolima '\$' na početku i kraju jednog takvog područja. Ukoliko se ispred simbola '\$' nalazi simbol '\', onda se simbol ne tumači kao eksplicitna oznaka područja interpretiranog teksta (" \$x=10 \$ " u odnosu na " *ovaj \\$ se neće interpretirati* ").
- o funkcije *eval()* i *plot()* se ne moraju posebno označavati, kao ni izrazi ili simboli koji im se prosljeđuju kao parametri.

Način na koji algoritam radi je jednostavan: prolazi se kroz primljeni tekst od početka prema kraju. Ukoliko se uoči simbol '\$' kojem prvi prethodni simbol (uključujući razmake) nije simbol '\' (\$ koji kojem nije izbjegnuto značenje koje ima u aplikaciji- engl. *escape*), cjelokupan sljedeći tekst do prvog sljedećeg simbola '\$' kojem nije izbjegnuto značenje se tretira kao mathtext. Ukoliko se uoče ključne riječi *eval* ili *plot*, cjelokupan tekst između prve sljedeće otvorene i njoj odgovarajuće zatvorene male zagrade tretira se kao commandtext.

Tok algoritma prikazan je dijagramom br. 18:



Dijagram 18 - tok algoritma za prepoznavanje uzoraka



Kao što je moguće vidjeti sa dijagrama br. 18, algoritam prolazi kroz ulazni niz znakova i obavlja provjeru simbol po simbol. `MathText` i `CommandText` LMD objekti se kreiraju kada se detektiraju specifičan jedan simbol ('\$' sukladno sintaksnim pravilima) ili niz simbola (eval, plot)), nakon čega se automatski traži kraj odgovarajućih objekata, smješta se njihov sadržaj, a trenutna pozicija koja se provjeravala u ulaznom nizu simbola pomjera jedan simbol iza posljednjeg simbola datog objekta. Ukoliko nije prepoznat početak ni `mathText` ni `commandText` objekata, trenutni simbol i svi simboli koji mu slijede a nisu '\$', kao i svi razmaci nakon njega dodaju se već postojećem ili novom `plainText` objektu. Trenutna pozicija se pomjera na posljednji obrađeni simbol, a petlja prelazi u svoju novu iteraciju. U kreiranim `mathText` i `commandText` objektima ignoriraju se razmaci i vrši se razlaganje teksta oko znaka jednakosti u više zasebnih izraza čiji redoslijed je očuvan.

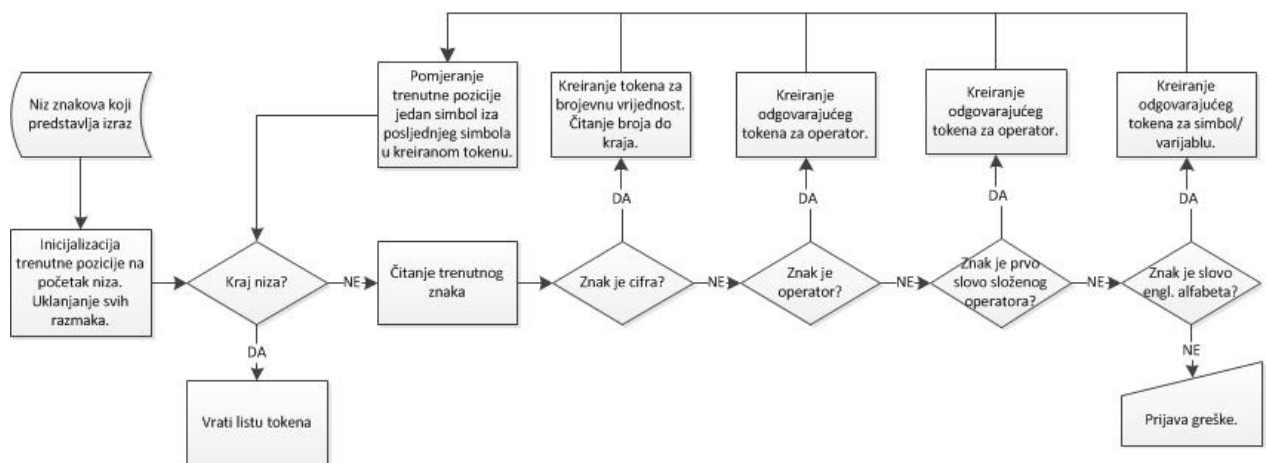
2. Algoritam za tokenizaciju na svom ulazu prima niz znakova za koji je ranije utvrđeno da ga *treba* interpretirati kao korektno napisan matematički izraz (što *ne znači* da izraz sam po sebi *jeste* korektan), a kao izlaz daje niz objekata (*tokena*) za koje je identificirano da li su konstanta, simbol/varijabla ili operator. Ako se radi o operatoru, onda se određuje koji je to operator, čime se određuje i njegov tip i njegov prioritet u odnosu na ostale operatore.

U procesu kreiranja LMD dokumenta iz *plaintexta*, ovaj algoritam izvršava se nakon identifikacije uzoraka, a prije gradnje stabla aritmetičkog izlaza. Pri tome, ukoliko nije moguće *tokenizirati* ulazni niz znakova ili dođe do makar jedne greške prilikom izvršavanja algoritma (neočekivani simbol), ulazni niz znakova se smatra nevalidnim i ne pristupa se gradnji stabla aritmetičkog izraza koje je pretpostavka za *rendering* i evaluaciju. Zbog toga ga se *renderira* kao obični tekst.

Konstante se prepoznaju kao decimalni brojevi. Za zapis se koristi decimalna tačka ('.'), a ne decimalni zarez (','). Nije dozvoljen zapis brojeva poput '.9', već se mora koristiti zapis '0.9'. Imena varijabli/simbola počinju slovom, a u okviru njih se mogu koristiti i arapski brojevi. Korištena slova moraju biti slova engleskog alfabeta. Osim slova engleskog alfabeta i arapskih brojeva, nikakvi drugi znakovi nisu dozvoljeni i rezultirati će neuspjehom *tokenizacije*. Operatori koje je moguće prepoznati su iz tabele br. 2. Prepoznaju se i otvorene i zatvorene male, srednje i velike zagrade.

Važno je za napomenuti da se ni nakon izvršenja ovog algoritma *ne garantuje* da je ulazni niz znakova *korektan*, već samo da su u njemu sve konstante, varijable/simboli i operatori *ispravno napisani*.

Tok algoritma dat je dijagramom br. 19. Potrebno je napomenuti da kreiranje svakog od *tokena* predstavlja zaseban algoritam koji podrazumijeva čitanje jednog ili niza simbola i kreiranje odgovarajućih *tokena*. Zasebni algoritmi kreiranja *tokena* mogu također naići na greške, čime cjelokupan proces *tokenizacije* pada, tj. dolazi do bacanja izuzetka. *Tokeni* su nositelji informacija: za konstante odgovarajući *tokeni* nose vrijednost konstante, za varijable nose njihova imena, a za operatore nose njihov tip i prioritet.



Dijagram 19 - tok algoritma za *tokenizaciju* izraza

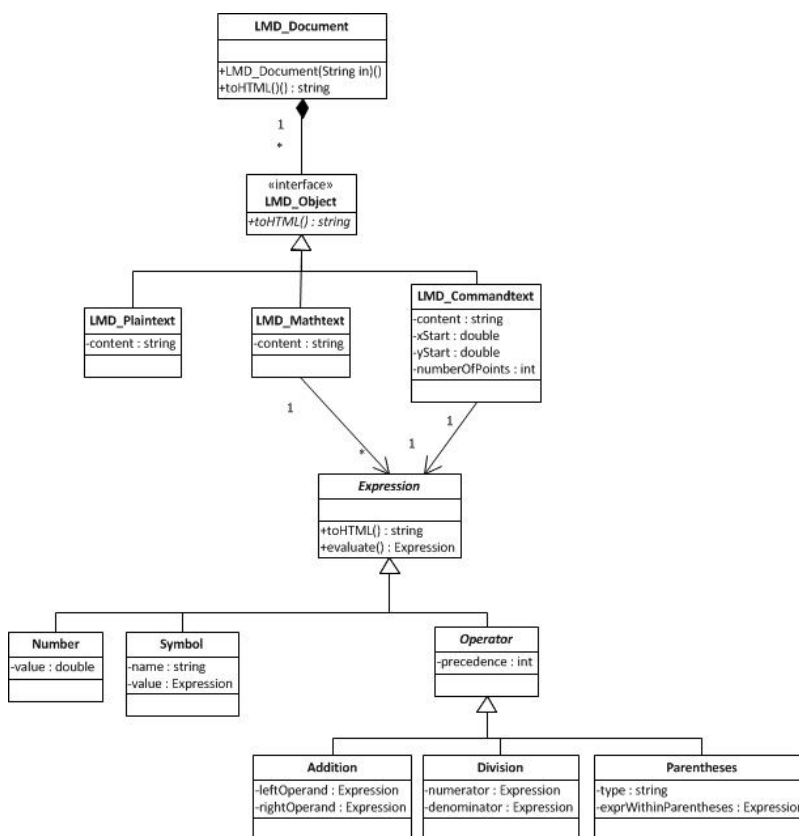
- Algoritam za gradnju stabla aritmetičkog izraza na svom ulazu prima listu ranije identificiranih *tokena*, a na svom izlazu po svom završetku izvršavanja daje *stablo aritmetičkog izraza*, jasno, ukoliko je izraz korektan.

Za prvu fazu koristi se modificirani *Shunting-yard* algoritam [28][29]. To je metod parsiranja matematičkih izraza datih u *infiksnoj* notaciji (koju koristi MathEasy, npr. izraz " $2+5*(3-1)$ " je zapisan u infiksnoj notaciji). Kao rezultat daje PN (prefiksni) ili RPN (postfiksni) zapis ([*Reverse*] *Polish Notation*- zapis u [obrnutoj] poljskoj notaciji) [24]. Prefiksni zapis za prethodni primjer bio bi "+ 2 \* 5 - 3 1". (Prefiksna) poljska notacija je način zapisivanja matematičkih izraza u kojem svakog operatora *sljede* njegovi operandi. Kod postfiksne notacije svakom operatoru *prethode* njegovi operandi. Algoritam je razvio Edsger Dijkstra i zasnovan je na upotrebi FIFO strukture podataka zvane stek (*stack*) [28].

Kako bi bilo moguće zadržati zagrade i na odgovarajući način ih *renderirati*, odstupa se od osnovnog oblika algoritma pa se i zagrade smatra posebnom vrstom operatora. Time se omogućava kreiranje objekata tipa zagrada koji se ponaša kao unarni operator, a evaluira se u evaluaciju izraza koji je sadržan u okviru zagrada. Također, kako *Shunting-yard* algoritam u svom osnovnom obliku kao izlaz daje postfiksni zapis, mijenja ga se tako da izlaz bude prefiksni zapis koji je razumljiviji i pogodniji za dalji rad.

Osnovni način rada *Shunting-yard* algoritma je sljedeći (zasnovan na originalnom Dijkstrinom opisu algoritma): algoritam uzima jedan po jedan *token* iz liste *tokena* koja je došla kao ulaz u algoritam, i obrađuje ih na sljedeći način: ako se radi o *tokenu* koji nosi brojevu vrijednost ili se radi o varijabli/simbolu, stavlja ga se u izlazni red. Ako je *token* operator  $O_1$ , tada dokle god postoji operator  $O_2$  koji je na vrhu steka i ili je  $O_1$  lijevo asocijativan operator sa prioritetom jednakim ili manjim od  $O_2$ , tada se  $O_2$  skida sa steka i stavlja na izlazni red, inače se  $O_1$  stavlja na stek. Ako je *token* otvorena zagrada, stavlja ju se na stek.

Ako je token zatvorena zagrada, dokle god je *token* na vrhu steka otvorena zagrada skidaju se operatori sa steka na izlazni red. Potom se skida zatvorena zagrada sa steka, i stavlja ju se u izlazni red. Ako se stek isprazni tražeći odgovarajuću otvorenu zagradu, to znači da je došlo do greške i da zagrade nisu uparene kako treba. Kada više nema nepročitanih *tokena*, ukoliko je još uvijek prisutan makar jedan operator na steku i ukoliko je operator na vrhu steka zagrada, tada se radi o pogrešno uparenim zagradama i potrebno je prijaviti grešku, u suprotnom se operator stavlja u izlazni red, čime algoritam završava [28]. Modifikacija koju se implementira jeste da se kreće od posljednjeg prema prvom *tokenu* u listi *tokena* koja je data kao ulaz u algoritam, promjenom izlaznog reda u izlazni stek te zamjenom ponašanja otvorenih i zatvorenih zagrada (otvorena zagrada bi trebala izbacivati elemente sa steka dok ne pronade zatvorenu zagradu) [30]. Nakon što je izvođenje *Shunting-yard* algoritma rezultiralo ili greškom (što signalizira da se izraz treba ponašati kao obični tekst) ili prefiksnim zapisom (uređenjem liste *tokena*), kreće se u izgradnju stabla izraza. Svaki *token* pretvara se u odgovarajući čvor stabla, a čvorovi se međusobno povezuju granama na osnovu svog tipa. Ova faza algoritma je trivijalna za realizirati upotrebom rekurzije.



Dijagram 20 - organizacija MathEasy klase

Na dijagramu br. 20 prikazana je osnovna šema organizacije klâsa u LMD komponenti aplikacije. Da bi se dijagram očuvao preglednim, prikazane su samo neke klase i samo neke njihove najznačajnije metode, pa se može smatrati opisnim.

### 3.3.6. Baza podataka

Baza podataka nalazi se na serverskom računar i za svaki unos (jedini entitet u bazi) čuva njegov ID, vrijeme postavljanja na server, tekstualnu datoteku i odgovarajući HTML zapis. Kako se u ovom slučaju radi o razvoju tehnološkog demonstratora a ne komercijalno upotrebljive aplikacije, to je zaobiđen sigurnosni aspekt tj. sprječavanje raznovrsnih napada na sistem (recimo, *upload* HTML-a koji nije generiran kroz MathEasy i problem njegovog prikaza kroz web preglednik kada mu se pristupa putem jedinstvenog URL, preopterećenje resursa servera itd.). Također, periodično uklanjanje starih dokumenata sa servera nije implementirano.

Baza podataka je samim time izuzetno jednostavna, kao i odgovarajuće skripte koje će se izvršavati na serveru.

## 3.4. Zaključak

U ovom poglavlju izrađen je dizajn aplikacije čime su stvorene pretpostavke za implementaciju aplikacije. Opisane su funkcionalnosti, arhitektura, komponente, klase i korišteni algoritmi MathEasy aplikacije. Dizajn je jezički i platformski neovisan, iako podrazumijeva upotrebu objektno orijentiranog programiranja i upotrebu *neke* razvojne platforme koja nudi određene gotove grafičke kontrole sa osobinama koje proizilaze iz opisanog u ovom poglavlju.

## 4. Implementacija i testiranje aplikacije

U okviru ovog poglavlja opisane su tehnologije pomoću kojih se aplikacija implementira. Sukladno postavljenom modelu, upotrebom odabranih tehnologija i primjenom odabranih metodologija, opisuje se proces razvoja desktop i web dijela aplikacije. Navode se i specifičnosti vezane uz korištene tehnologije. Također, ukratko je opisan način na koji je aplikacija testirana, kao i način na koji su ispravljeni eventualni propusti u inicijalnom dizajnu i implementaciji.

### 4.1. Razvojno okruženje i korištena tehnologija

U 3. poglavlju istaknuta je potreba za korištenjem gotovih komponenti koje samostalno rješavaju problem unosa teksta (osnova za uređivač), kao i potreba za postojanjem jednostavnog sistema za upravljanje događajima koje inicira ili kreira korisnik za potrebe ostvarivanja funkcionalnosti poput zamjene simbola '?' evaluiranim prvim prethodnim izrazom. Osim toga, uređivač bi trebao dolaziti sa ugrađenom mogućnosti da se na jednostavan način promijeni boja pojedinim dijelovima teksta, kao i da se u uređivač mogu ugraditi složenije komponente kao što je *autocomplete* pomoćnik.

Imajući navedeno u vidu, da se radi o desktop aplikaciji, da ne postoje nikakvi zahtjevi na portabilnost aplikacije na npr. Linux operativne sisteme, kao i da se autor ovog rada tokom studija susreo sa C# programskim jezikom, izbor se prirodno svodi na C# (Visual Studio 2010) aplikaciju. Između ostalog, Visual Studio IDE (razvojno okruženje) omogućava kreiranje WPF (Windows Presentation Foundation) aplikacija upotrebom C# i XAML (*Extensible Application Markup Language*, zasnovan na XML-u) jezika. WPF ne podrazumijeva samo grafički podsistem za *rendering* korisničkih sučelja u aplikacijama za Windows operativne sisteme, već potpuno zaseban i značajno drugačiji programski model u odnosu na standardni Windows Forms model. To podrazumijeva niz klása, korisničkih kontrola i programskih mogućnosti, a nudi i značajno drugačije iskustvo u razvoju aplikacija u odnosu na Windows Forms.

Pored autorove želje da, između ostalog, kroz ovaj završni rad nauči raditi u WPF-u, jedan od razloga za WPF je i činjenica da WPF dolazi sa nizom korisničkih kontrola čije ponašanje i koncept funkcioniranja su, iako imaju iste nazive kao njihovi pandani u Windows Forms, drugačiji i napredniji. Jedna od tih kontrola je WPF varijanta kontrole `RichTextBox` koja na elegantan način rješava rad sa *bogatim* tekstualnim dokumentom (primjenu raznovrsnih formatiranja na pojedine dijelove teksta, ubacivanje objekata u dokument i sl.) što je upravo ono što je i potrebno za MathEasy aplikaciju (bojiti će se uparene zagrade), a što nije na pogodan način riješeno u Windows Forms.

Također, WPF nudi znatno napredniji i kompleksniji način rada sa događajima (*events*) nego Windows Forms, što daje programeru bolju kontrolu nad procesiranjem događaja i što se koristi za presretanje posebnih komandi i lakšu implementaciju *autocomplete* funkcionalnosti.

Mogućnostima bogatije korisničke kontrole i drugačiji način upravljanja događajima nisu jedine razlike WPF i Windows Forms modela, no radi se o mogućnostima koje su najvažnije za ovaj rad.

Pored navedenih, dalje razlike su u tome da se WPF aplikacije razvijaju kroz poseban dizajner koji omogućava precizno podešavanje načina na koji će izgledati korisničko sučelje, da se cjelokupno sučelje gradi kroz *Gridove*- mreže koje mogu dinamički mijenjati svoju veličinu i da se sučelje specifikira posebnim jezikom XAML koji je veoma sličan XML-u (i HTML-u) pa se pisanje sučelja ne razlikuje u mnogo čemu od pravljenja web stranice.

Što se tiče serverskog dijela aplikacije, koriste se PHP skripte koje će biti odgovorne za korektnu interpretaciju URL-ova i obradu tj. smještanje dokumenata u bazu podataka i pristup istima bilo iz aplikacije, bilo kroz web preglednik putem odgovarajućeg URL-a. Baza podataka koju se koristi je MySQL.

Za *rendering* LaTeX sadržaja koristi se *MathJax JavaScript rendering engine* [19]. LaTeX sadržaj jednostavno se integrira u obični HTML prostim navođenjem delimitera (graničnika) koji su oblika `'\('` i `')\'` za *inline* sadržaje, ili `'\[` i `]\'` za višelinijске sadržaje. Sve ostalo na sebe preuzima MathJax. Naravno, skripti je potrebno određeno vrijeme da se izvrši kako bi se stranica na odgovarajući način *renderirala*.

The Cauchy-Schwarz Inequality

$$\left(\sum_{k=1}^n a_k b_k\right)^2 \leq \left(\sum_{k=1}^n a_k^2\right) \left(\sum_{k=1}^n b_k^2\right)$$

Slika 7 - prikaz HTML dokumenta uz upotrebu *MathJax rendering enginea*

Za prikaz grafika koristi se jqPlot dodatak za *jQuery Javascript framework* [31][32]. Jednostavno se koristi uz osnovno poznavanje jQuery frameworka: odredi se *div* element koji će sadržavati grafik, a potom se upotrebom odgovarajuće metode proslijedi niz koji sadrži uređene parove vrijednosti. Iscrtavanje grafika počinje onog trenutka kada je dokument spreman.

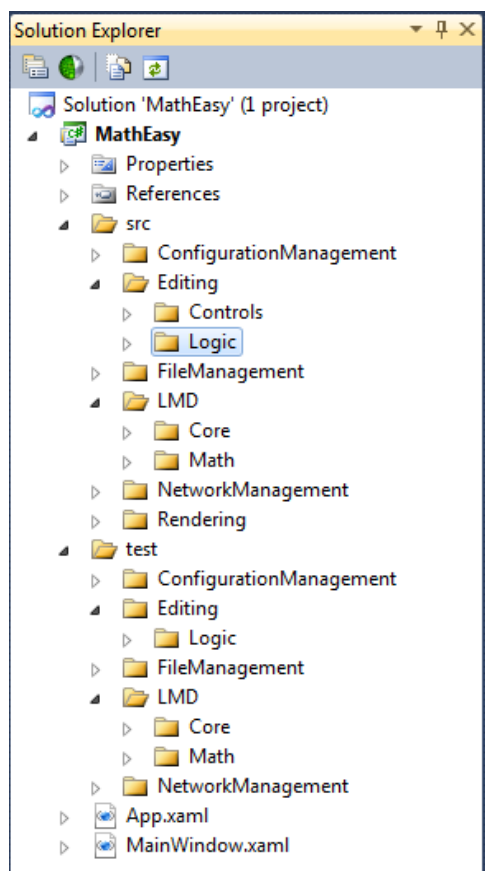
Sve navedeno detaljnije se diskutuje u odgovarajućim odjeljcima koji se bave konkretnim razvojem MathEasy aplikacije. Pri tome se specifičnosti WPF programskog modela navode u odnosu na rasprostranjeni Windows Forms model.

## 4.2. Specifičnosti i razvoj WPF desktop aplikacije

### 4.2.1. Struktura projekta

Desktop dio MathEasy aplikacije razvijen je kroz Visual Studio 2010 razvojno okruženje kao Windows Presentation Foundation aplikacija. WPF je podržan od treće verzije .NET frameworka (izašao 2006. godine) i WPF aplikacije je moguće pokretati na svim računarima pod Windows operativnim sistemima sa instaliranom navedenom ili novijom verzijom .NET frameworka.

Struktura projekta WPF aplikacije identična je kao i kod "običnih" Windows Forms aplikacija, kao što je dato na slici br. 8:



Slika 8 - struktura MathEasy projekta

Sa slike br. 8 može se zaključiti da se projekat sastoji iz dva osnovna dijela, a to su *src* i *test*. *src* folder sadrži *source* kôd, dok *test* folder sadrži *unit* testove kojima se provjerava ispravnost funkcioniranja (verifikacija) pojedinih komponenti. Izvorni kôd aplikacije sastoji se iz 6 imenskih prostora (namespaces) koji se automatski kreiraju s obzirom na to da postoje odgovarajući folderi u projektnoj strukturi. Imenski prostori odgovaraju komponentama koje su specificirane dizajnom aplikacije. "*test*" folder sadrži identične imenske prostore, s razlikom da imenski prostori koji sadrže komponente koje nije praktično ili moguće *unit* testirati nisu prisutni (npr. *src.Editing.Controls*). Sadržaj tih imenskih prostora čine *TestFixture* klase, tj. klase koje služe za *unit* testiranje.

#### 4.2.2. XAML i WPF kontrole

Sa slike br. 7 moguće je uočiti da se na korijenskom nivou projekta nalaze dvije datoteke sa ekstenzijom ".xaml". Radi se o datotekama koje su pisane u ranije spomenutom *markup* jeziku koji se zove XAML. XAML služi za specificiranje korisničkog sučelja na način veoma sličan načinu na koji se to radi sa HTML web dokumentima. Sve izmjene napravljene u XAML kôdu automatski se manifestiraju u dizajnerskom pogledu na napisani kôd, tj. XAML se interpretira i Visual Studio automatski generira odgovarajući C# kôd. U nastavku (listing br. 1) je dat jednostavan "Hello World" XAML kôd koji generiše prozor sa jednom labelom postavljenom u *grid* (mrežu). Važno je primijetiti da XAML specifikacija dozvoljava da element prozora može imati *samo* jedno dijete - u ovom slučaju jedan grid. Za kreiranje složenijih sučelja, potrebno je napraviti korijenski grid (ili neki drugi kontejner koji podržava postavljanje više kontrola kao svoje djece) koji će sadržavati djecu-druge gridove koji će sadržavati korisničke kontrole.

```
<Window x:Class="MathEasy.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Label Content="Hello world!"/>
        <Button Width="60" Height="30" Margin="86,0,0,0" HorizontalAlignment="Left"
VerticalAlignment="Top" Content="Click me!" Click="button1_Click"/>
    </Grid>
</Window>
```

Listing 1 - primjer specifikacije jednostavnog prozora u XAML

Svakoj kontroli moguće je odrediti njenu poziciju u okviru Grid objekta (ili njegovih redova ili kolona koje je moguće specificirati kao attribute objekta), visinu i širinu (moguće je specificirati ponašanje gdje se veličina kontrole dinamički povezuje sa veličinom roditeljske kontrole (u ovom slučaju Grid objekta), pa promjenom veličine prozora se sasvim transparentno (sa aspekta programera) mijenja veličina kontrole. Pri tome više ne postoje ograničenja iz Windows Forms modela: kontrola tipa Label više nema sadržaj koji je nužno tekst, već možemo vidjeti i sa listinga br. 1 da postoji atribut labele koji se zove "Content", a koji može biti obični tekst ili URI na npr. sliku, ili neki statički resurs (npr. vrijednost atributa neke klase - to znači da se sadržaj labele uvezao sa klasom i da se onda mijenja tokom izvršenja programa kako se mijenja vrijednost tog atributa). Metode za obradu događaja (engl. *Event handler*) u WPF predstavljaju samo još jedan atribut u XAML (u listingu br. 1: "Click").

Još jedno interesantno svojstvo WPF-a je da je moguće manipulirati svim aspektima prikaza i ponašanja kontrole (ili njenih atributa!) kroz kreiranje šablona (engl. *templating*). U listingu br. 2 dat je primjer korisničke kontrole koja nasljeđuje RichTextBox korisničku kontrolu (važna napomena: nije moguće naslijediti kontrolu koja je već naslijedila neku drugu WPF kontrolu. Ograničenje i nema neko jako opravdanje, barem ga autor ovog rada nije uspio pronaći. Izgleda da se jednostavno radi o mogućnosti koju su autori WPF-a odlučili zabraniti.).



```

<RichTextBox x:Class="MathEasy.src.Editing.Controls.LMD_Editor"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" FocusManager.IsFocusScope="True" d:DesignWidth="300"
    SelectionChanged="RichTextBox_SelectionChanged" PreviewKeyDown="hendlanjeIntellisensea"
    KeyUp="RichTextBox_KeyUp" LostFocus="RichTextBox_LostFocus"
    GotFocus="RichTextBox_GotFocus" AutoWordSelection="False" AllowDrop="False"
    IsDocumentEnabled="True" HorizontalScrollBarVisibility="Auto" HorizontalAlignment="Center"
    VerticalAlignment="Center">
    <RichTextBox.Resources>
        <Style TargetType="{x:Type Paragraph}">
            <Setter Property="Margin" Value="0"/>
        </Style>
    </RichTextBox.Resources>
</RichTextBox>

```

Listing 2 - primjer naslijeđene WPF korisničke kontrole

Iz listinga br. 2 može se vidjeti da je određen niz atributa za samu kontrolu, kao i da je u okviru njenog sadržaja postavljen atribut `RichTextBox.Resources`. U okviru njega definiran je stil koji se odnosi ("TargetType") na sve elemente `RichTextBoxa` koji su tipa `Paragraph`. Definira se statički *Setter* (postavljač) koji atribut "Margin" objekta tipa `Paragraph` postavlja na vrijednost "0". Naime, WPF daje izuzetno napredan model dokumenta bogatog sadržajima ("FlowDocument") koji je zasnovan na upotrebi blokovskih elemenata (kao što su paragrafi, tabele, sekcije, liste), a kojima se kao djecu dodaju *Inlines* tj. kontrole izvedene iz klase `Inline` a koje predstavljaju atome u okviru bloka (npr. slijed identično formatiranog izmjenjivog ili neizmjenjivog teksta ili kontejner koji može sadržavati proizvoljnu korisničku kontrolu). Kako paragrafi prilikom svog kreiranja automatski dobijaju prored između paragrafa u vrijednosti 10, specificiranim stilom spriječeno je vizuelno razlikovanje odvojenih paragrafa.

Primjer `FlowDocument` modela dat je kroz listing br. 3:

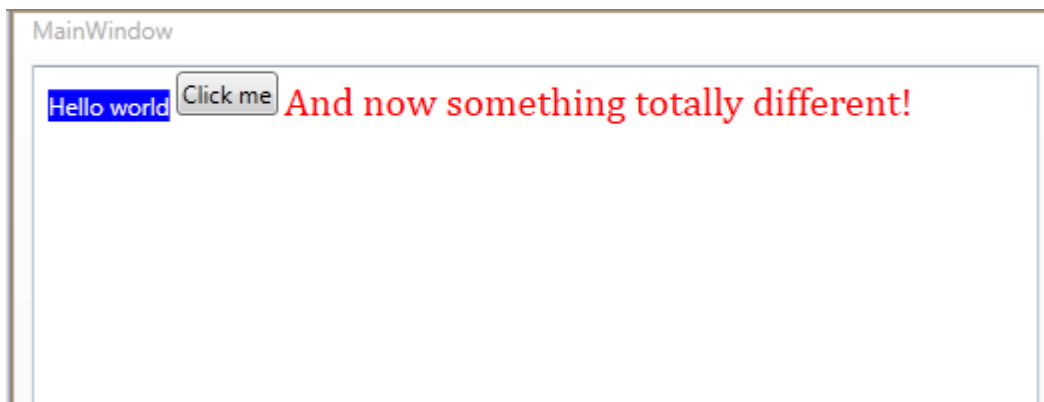
```

<RichTextBox IsDocumentEnabled="True">
    <FlowDocument>
        <Paragraph>
            <Run Background="Blue" Foreground="White">Hello world</Run>
            <InlineUIContainer>
                <Button Content="Click me"></Button>
            </InlineUIContainer>
            <Run FontFamily="Cambria" FontSize="20" Foreground="Red">And now
something totally different!</Run>
        </Paragraph>
    </FlowDocument>
</RichTextBox>

```

Listing 3 - primjer strukture `FlowDocument` objekta

Rezultat listinga br. 3 u Visual Studio dizajneru izgleda kako je dato na slici br. 9.



Slika 9 - *rendering* kôda datog u listingu br. 3

Objekat tipa `Run` predstavlja izmjenjivi slijed teksta na kojeg je primijenjeno određeno formatiranje, u ovom slučaju data su dva *Runa*, gdje je prvom podešena plava pozadinska boja i bijela boja teksta, tok je drugom boja teksta crvena, font koji se koristi više nije automatski dodijeljeni i promijenjena mu je i veličina. `InlineUIContainer` je *inline* objekat koji omogućava da mu se kao sadržaj postavi jedna korisnička kontrola (u ovom slučaju `Button`). `InlineUIContainer` dodaje se kao sadržaj u okvir odgovarajućeg paragrafa u dokumentu. Ukoliko se odgovarajućom opcijom na nivou `RichTextBoxa` to omogući (`IsDocumentEnabled=true`), kontrola sadržana u `InlineUIContainer` postaje aktivna (u ovom slučaju, dugme je moguće kliknuti i vezati ga uz neku metodu za obradu događaja, tako da se klikom npr. prikaže poruka).

Još jedna specifičnost WPF-a koja nije pokazana kroz prethodne primjere je činjenica da objekti specificirani kroz XAML *ne dobijaju* automatski svoja imena (npr. "`richtextbox_1`") kao što je to slučaj u Windows Forms, već je to potrebno eksplicitno specificirati. Naime, bezimenim objektima *nije moguće pristupiti* iz C# programskog kôda. Zbog toga se (ukoliko se ručno uređuje XAML kôd i ne radi se uređivanje sučelja putem dizajnera) mora eksplicitno navesti atribut *Name* za objekat.

Navedenim primjerima izloženi su samo neki osnovni principi uređivanja grafičkog korisničkog sučelja u WPF-u i osnovni pregled sintakse XAML-a. Detaljan pregled osobina WPF-a poput definisanja predložaka/šablona, stiliziranja, uređivanja imenskih prostora kroz XAML i definiranja ponašanja kontrola u zavisnosti od korisničkih akcija zauzeo bi dovoljno prostora da bi slobodno mogao predstavljati zaseban rad. Princip uvezivanja kontrola i izvora podataka kojima se kontrole popunjavaju (engl. *binding*) je značajno drugačiji nego što je to bio slučaj u Windows Forms. On omogućava relativno komplikovan sistem automatske validacije podataka kao i složenih interakcija između korisničkih kontrola međusobno, no ova mogućnost nije bila potrebna za ovaj rad. I u slučaju da jeste, ne bi je bilo moguće realizirati zbog upotrebe `RichTextBoxa` za kojeg je u WPF modelu procijenjeno da je previše složen i da omogućava interakcije koje čine povezivanje besmislenim (npr. situacija kada je određeni `Run` teksta povezan sa sadržajem nekog izvora podataka- korisnik pritisne CTRL+A i obriše čitav sadržaj `RichTextBoxa`, što povlači uklanjanje *Runa* iz memorije. Kako to preslikati na izvor podataka?).

#### 4.2.3. Model upravljanja događajima u WPF

Kada je moguće napraviti korisničko sučelje, potrebno ga je učiniti upotrebljivim. Slično kao i u Windows Forms, konstruktor odgovarajuće kontrole poziva `InitializeComponent()` metodu koja kreira i inicijalizira sve elemente kontrole (samo što navedena metoda u WPF ima dodatni korak, a to je pretvaranje XAML kôda u C# kôd).

Već pri kreiranju prvog *event handlera* tj. metode klase koja obrađuje događaje i akcije uočava se još jedna značajna razlika između WPF i Windows Forms: sistem događaja u WPF-u je nadograđen i događaje se sada naziva *rutirajućim događajima (routed events)*. Radi se o događajima koji se sada usmjeravaju kroz vizuelno stablo (lako ga je uočiti kroz odgovarajući XAML kôd- svaka kontrola je dijete neke druge kontrole čime se formira odgovarajuće vizuelno stablo). Usmjeravanje kroz vizuelno stablo odvija se na osnovu odabrane strategije, koja može biti tunelirajuća (*tunneling*), poput mjehura- strategija širenja (*bubbling*) ili direktna.

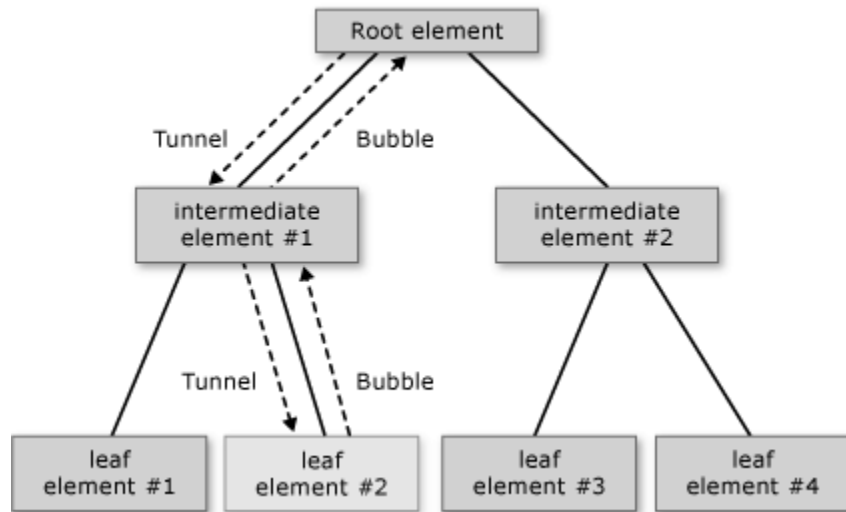
Na osnovu strategija, svi rutirajući događaji dolaze u parovima. Prvi događaj u paru ima reprezentativno ime formulirano na način jednak kao u Windows Forms (npr. `MouseDown`). Drugi događaj u paru dobija prefiks *Preview*, pa se time dobija npr. `PreviewMouseDown`. *Preview* događaji koriste tunelirajuću strategiju: kada se inicira događaj, on putuje od aplikacijskog korijena prema elementu (važno je napomenuti da svi vizuelni elementi se sada nasljeđuju iz klase `UIElement`) koji je inicirao događaj. Ovi događaji dešavaju se *prije* odgovarajućih *bubbling* događaja. Pri tome, svi rutirajući događaji imaju atribut `Handled`. Vrijednost ovog atributa je *false* tokom cijelog životnog ciklusa događaja sve dok ga se u nekom *event handleru* eksplicitno ne postavi na *true*, tj. metoda za obradu događaja po svom završetku izvršenja ostavi događaj takav da `Handled` atribut ima vrijednost *true*. Ukoliko se postavi navedena vrijednost, događaj se dalje ne rutira tj. ne silazi niz vizuelno stablo.

*Bubbling* događaji kreću od elementa i putuju (penju se kroz stablo) prema korijenskom elementu. I za ovu strategiju rutiranja događaja vrijedi da ukoliko bilo koja metoda za obradu događaja izmijeni događaj tako da on postane *handled*, događaj se dalje ne rutira.

Strategija direktnog rutiranja identična je načinu na koji rade događaji u Windows Forms: događaj se inicira i obrađuje samo i isključivo na izvornom elementu.

Iako koncept rutirajućih događaja djeluje kao bespotrebna komplikacija nečega što korektno radi po principu direktnog rutiranja (tj. bez rutiranja kao takvog), upotreba rutirajućih događaja omogućava značajno jednostavniji rad sa složenim interakcijama između kontrola grafičkog korisničkog sučelja a koje su inicirane sistemskim ili korisnički definiranim događajima. Jednostavnim pretplaćivanjem na određene vrste rutirajućih događaja, kontrola postaje potpuno integrirana sa svojim okruženjem i u stanju je na za programera transparentan način sudjelovati u složenim interakcijama i biti obaviještena o namjeri za događaj (*preview* događaj) i spriječiti ga prije nego se on uopće desi, ili spriječiti propagaciju događaja prema kontrolama - djeci te kontrole.

Primjer obrade događaja i toka rutiranja dat je na primjeru događaja vezanih uz pritisak tipke miša nad nekom korisničkom kontrolom [33]:



Slika 10 - primjer vizuelnog stabla WPF aplikacije i putanja događaja [33]

Element koji inicira događaj je leaf element #2 dat na slici br. 10. Tok procesiranja događaja je dat kako slijedi:

- a) *PreviewMouseDown* (tunelirajući) na *Root element*
- b) *PreviewMouseDown* (tunelirajući) na *intermediate element #1*
- c) *PreviewMouseDown* (tunelirajući) na *leaf element #2* (inicijator događaja)
- d) *MouseDown* (bubble) na *leaf element #2* (inicijator događaja)
- e) *MouseDown* (bubble) na *intermediate element #1*
- f) *MouseDown* (bubble) na *Root element*

Moguće je primijetiti da se presretanjem *Preview* događaja prije nego isti dođe do svog izvora (najkasnije u koraku c) ) može postići da izvorišni element nikada ne dobije informaciju da je događaj uopće bio iniciran. Time se na jednostavan način realizira potiskivanje određenih neželjenih ponašanja korisničkih kontrola.

Osim konceptualnih razlika i činjenice da se koriste drugačije klase za događaje nego što je to slučaj u Windows Forms, još jedna razlika je i u načinu na koji se kreiraju (i iniciraju) korisnički definirani događaji. Kako navedena funkcionalnost nije potrebna u MathEasy aplikaciji, to se neće diskutirati no više informacija moguće je dobiti putem reference [34].

#### 4.2.4. Model komandi u WPF

Još jedan koncept koji je uveden u WPF su *komande*. Komande omogućavaju rad sa korisničkim akcijama na prirodniji način nego što je to slučaj sa događajima koji su vezani uz specifične uređaje (npr. tastaturu i miš) [35]. Primjer komandi su standardne cut, copy i paste operacije. Osnovni zadatak komandi je razdvojiti semantiku i objekat koji inicira komandu od logike koja se bavi izvršavanjem komande. Na ovaj način je omogućeno da više odvojenih objekata može izazvati izvršavanje identične programske logike.

Ranije navedene komande može se inicirati kraticama putem tastature kao što su CTRL+C za kopiranje, CTRL+V za lijepljenje i CTRL+X za rezanje sadržaja, no isti efekat se može dobiti i upotrebom kontekstnog menija kojeg se dobija desnim klikom miša i izborom odgovarajuće opcije. Treći način ostvarivanja identične funkcionalnosti je selekcijom pomoću miša, a potom izborom opcije u standardnom meniju aplikacije. Dakle- više različitih načina ostvarivanja identične funkcionalnosti indikator je da se u dizajnu aplikacije treba razmisliti o upotrebi WPF komandi. Za upotrebu komandi potrebno je razumjeti i precizno odrediti: koja je komanda u pitanju, šta je izvor komande (command source), šta je objekat komande tj. objekat nad kojim se komanda izvršava (command target) i kako se za komandu specificira njena programska logika (command binding).

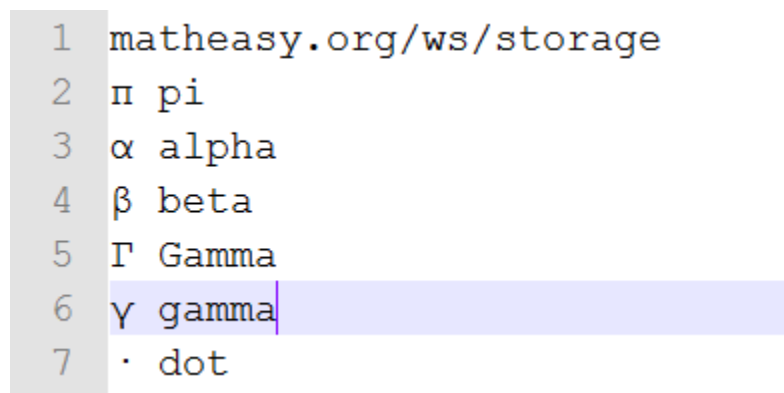
WPF dolazi sa više skupova ugrađenih komandi (MediaCommands, ApplicationCommands, NavigationCommands, ComponentCommands, EditingCommands), no svakako je moguće specificirati vlastite komande i koristiti ih na način jednak ugrađenima. Implementacija vlastitih komandi je jednostavna (implementira se ICommand interfejs koji specificira dvije metode: Execute i CanExecute). Execute izvršava akcije komande, dok CanExecute određuje da li je komandu moguće izvršiti nad odgovarajućim ciljnim (*target*) objektom. Npr. ukoliko u polju za unos teksta trenutno nema selekcije tj. nije selektiran niti jedan simbol ili niz simbola, CanExecute metoda će se pozvati prije Execute metode i zaključiti da nisu stvoreni uvjeti za izvršenje komande. Kako WPF automatski povezuje pozicije sa kojih se inicira komandu (npr. kontekstni meni) sa ciljnim objektom (polje za unos teksta) i odgovarajućom logikom, to će odgovarajuća opcija u kontekstnom meniju po automatizmu biti blokirana, tj. neće biti moguće uopće inicirati komandu sve dok se ne selektira određeni tekst.

U kontekstu razvoja MathEasy aplikacije pokazuje se da su komande izuzetno korisne (za uvezivanje funkcija poput kreiranja novog dokumenta, otvaranja/downloada postojećeg ili spašavanja/uploada novog dokumenta), ali i opasne (na primjeru RichTextBox kontrole koja nudi mogućnosti i uvezana je sa komandama koje je potrebno zabraniti).

#### 4.2.5. Detalji implementacije

U okviru ovog odjeljka razmatra se način na koje su implementirane određene funkcionalnosti u okviru WPF aplikacije. Pri tome se razmatranja odnose na specifičnosti WPF-a, ali i detalje načina ostvarivanja nekih od specificiranih funkcionalnosti.

Prva stvar koju se razmatra jeste način rada sa konfiguracijom aplikacije. Specifikacijom je određeno da je konfiguracija sadržana u konfiguracijskoj datoteci koja se nalazi u istom folderu u kojem se nalazi i izvršna datoteka MathEasy aplikacije. Zarad jednostavnosti, datoteka će biti u obliku običnog teksta u formatu opisanom datom specifikacijom. Naziv datoteke je "*matheasy.config*" i sadržaj je kodiran kao UTF-8. Na slici br. 11 dat je primjer sadržaja. U prvom redu data je putanja do servisa odgovornog za *upload* (šalju se PUT HTTP zahtjevi) i *download* (šalju se GET HTTP zahtjevi). Ispod prve linije, u svakoj narednoj nalazi se simbol kojeg se želi imati kao izbor kroz *autocomplete* jednim praznim mjestom odvojen od niza znakova (bez razmaka) koji služe kao kratica za uneseni simbol.



```
1 matheasy.org/ws/storage
2 π pi
3 α alpha
4 β beta
5 Γ Gamma
6 γ gamma
7 · dot
```

Slika 11 - sadržaj konfiguracijske datoteke

Potrebno se pobrinuti i za to da se onemogući upotreba uređivača dokumenta u svrhe kreiranja web stranica što može biti i maliciozan sadržaj. U te svrhe se svi HTML specifični karakteri mijenjaju svojim ekvivalentima (npr. za '<' je to '&lt;') kako bi se spriječio unos HTML tagova u LMD dokument.

Sljedeća specifičnost vezana je uz Editor. Budući da se u okviru aplikacije koristi WPF RichTextBox, javlja se problem vezan uz činjenicu da navedena kontrola omogućava različite vrste blokova (npr. *bullets & numbering*, tabele i sl.), raznovrsne načine formatiranja teksta (podebljavanje, nakošena slova, podvučena slova itd.), kao i automatski omogućeno ubacivanje slika u dokument kroz sistemski međuspremnik (engl. *clipboard*).

Prva stvar koju je potrebno napraviti jeste prilagoditi sadržaj RichTextBox kontrole LMD modelu. Kao što je već specificirano dizajnom, LMD zarad jednostavnosti ne podržava paragrafe, pa ih je potrebno ručno kreirati dvostrukim *enterom*. Navedeno će se realizirati presretanjem događaja pritiska tipke *enter* i potiskivanjem standardne akcije kreiranja novog reda. Umjesto toga, ručno će se ubaciti znak novog reda na datu poziciju.

Kostur kôda koji obavlja navedenu funkciju je u suštini metoda za obradu događaja za PreviewKeyDown događaj i data je listingom br. 4:

```
private void editor_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        editor.Selection.Start.InsertLineBreak();
        var newPointer =
editor.Selection.End.GetNextInsertionPosition(LogicalDirection.Forward);
        editor.Selection.Select(newPointer, newPointer);
        e.Handled = true;
    }
}
```

Listing 4 - primjer kôda za potiskivanje kreiranja novog paragrafa

Navedeno rješenje je jednostavnije nego da se dozvoli kreiranje paragrafa i da se potom njihovi sadržaji dodaju jedan na drugi kako bi se kreirala jedinstvena znakovna reprezentacija unesenog sadržaja.

Dalje, potrebno je zabraniti bilo kakva formatiranja teksta i unos objekata poput slika. To se radi blokiranjem WPF-ovih ugrađenih komandi na način da se napravi metoda klase koja automatski vraća vrijednost *false* u kontekstu provjere da li je komandu moguće izvršiti (CanExecute metoda).

```
<RichTextBox.CommandBindings>
    <CommandBinding Command="{x:Static EditingCommands.ToggleNumbering}"
CanExecute="zabrani" />
    <CommandBinding Command="{x:Static EditingCommands.ToggleBold}"
CanExecute="zabrani" />
    <CommandBinding Command="{x:Static EditingCommands.ToggleItalic}"
CanExecute="zabrani" />
    <CommandBinding Command="{x:Static EditingCommands.ToggleUnderline}"
CanExecute="zabrani" />
    <CommandBinding Command="{x:Static EditingCommands.ToggleSuperscript}"
CanExecute="zabrani" />
    <CommandBinding Command="{x:Static EditingCommands.ToggleSubscript}"
CanExecute="zabrani" />
</RichTextBox.CommandBindings>
```

Listing 5 - primjer kôda za potiskivanje aplikacijskih komandi

XAML kôdom datim u listingu br. 5 potisnute su aplikacijske komande za promjenu stila kojim je tekst napisan (npr. zabranjeno je podebljavanje koje je inače moguće aktivirati pritiskom na CTRL+B), kao i za ubacivanje listi (*numbering*). Na analogan način rješava se i problem ubacivanja tabela, proreda i slično. Zabrana ubacivanja objekata poput slika rješava se u dva koraka: prvo se zabranjuju *Drag&Drop* operacije postavljanjem atributa RichTextBox objekta "AllowDrop" na *false*, a potom je potrebno napraviti vlastiti omotač oko ugrađene funkcije za lijepljenje sadržaja. Prvo se definira odgovarajući CommandBinding (kao u listingu 5), samo što sada CanExecute metoda provjerava da li se u sistemskom međuspremniku (iz kojeg se vrši lijepljenje) nalazi slika.

```

        <RichTextBox.CommandBindings>
            <CommandBinding Command="{x:Static ApplicationCommands.Paste}"
CanExecute="CustomPasteCanExecute"></CommandBinding>
        </RichTextBox.CommandBindings>

        ...

        private void CustomPasteCanExecute(object sender, CanExecuteRoutedEventArgs e)
        {
            e.CanExecute = !Clipboard.ContainsImage();
            e.Handled = true;
        }

```

Listing 6 - primjer kôda za potiskivanje lijepljenja slike u dokument

Kako je potisnuto kreiranje novih paragrafa, dobijanje stringa koji predstavlja sadržaj dokumenta postaje trivijalan problem za riješiti: dokument garantovano sadrži jedan paragraf u okviru kojeg se nalazi proizvoljan broj *runova* (podsjećanja radi, *runove* se koristi npr. kod bojenja uparenih zagrada, posebnih simbola i ključnih riječi). Svaki *Run* sadrži svoj *plaintext* sadržaj u vidu atributa *Text*. Da bi se poboljšale performanse koristi se *StringBuilder* klasa koja sadržaj nadovezuje na kraj privremenog stringa, a ne obavlja se obično "sabiranje" stringova koje je sporije jer se kreira potpuno novi string.

```

private String flowdoc2string(FlowDocument flowdoc)
{
    StringBuilder str = new StringBuilder();
    foreach (Paragraph p in flowdoc.Blocks)
    {
        foreach (Inline i in p.Inlines)
        {
            Run r = i as Run;
            str.Append((r != null) ? r.Text : "");
        }
    }
    return str.ToString();
}

```

Listing 7 - primjer kôda za preuzimanje sadržaja *FlowDocument* objekta kao stringa

U kôdu datom kroz listing br. 7 podrazumijeva se da je sadržaj dat u vidu niza *runova*, a ukoliko se iz nekog razloga u okviru paragrafa pojavi bilo koji objekat koji nije *Run*, tretirati će se kao prazan string (upotreba ternarnog operatora).

*Autocomplete* funkcionalnost realizira se kao poseban *popup* prozor čiji se prikaz inicira kada se presretne unos simbola '\', što je sukladno specifikaciji. *Popup* prozor ima osobinu da je transparentan kako bi se što jednostavnije uklopio u okruženje, kao i da se uklanja ili unosom razmaka ili pritiskom na escape dugme. U suprotnom se filtrira baza simbola iz konfiguracijske datoteke sukladno korisničkom unosu, a kako bi se ponudio odgovarajući izbor koji se potvrđuje pritiskom na dugme *enter*.

Ono što zavrjeđuje pažnju jeste način na koji se ostvaruje transparentnost bilo kojeg dijela ili cijele kontrole, što je relativno jednostavno za izvesti u WPF-u, a praktično nemoguće u okviru Windows Forms.



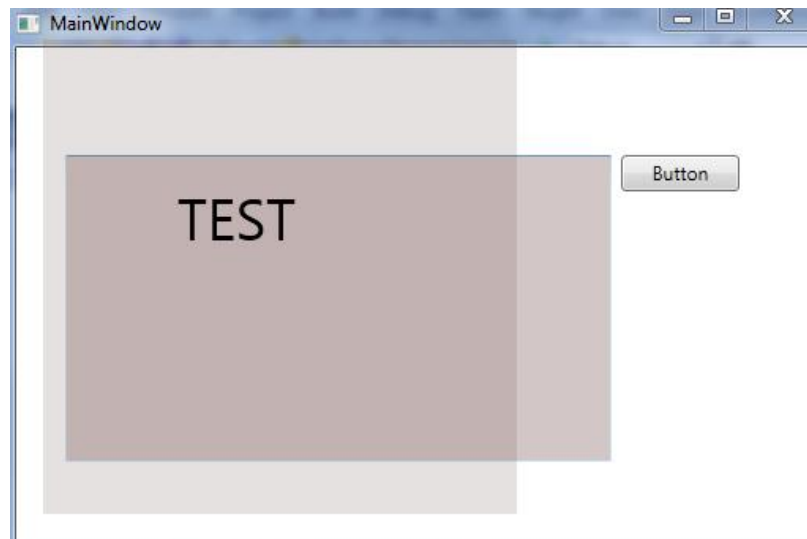
Listingom br. 8 dat je primjer prozora koji je djelimično transparentan i sadrži pozdravnu poruku.

```
<Window x:Class="MathEasy.src Editing.Controls.AutoComplete"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Autocomplete" Height="300" AllowsTransparency="True" Width="300"
        WindowStartupLocation="CenterOwner" WindowStyle="None" Background="#2D705353"
        Foreground="#2D705353">
    <Grid>
        <Label Content="HELLO" Height="93" HorizontalAlignment="Left" Margin="80,83,0,0"
        Name="label1" VerticalAlignment="Top" Width="143" FontSize="36" />
    </Grid>
</Window>
```

Listing 8 - primjer transparentnog prozora u WPF

Kao što se može vidjeti, potrebno je postaviti vrijednost atributa `AllowsTransparency` na `true`, a potom bilo kroz dizajner, bilo ručno, dati kôd pozadinske boje (u ovom slučaju želi se postići da cijeli prozor bude transparentan i da nema granice tj. obrub). Pri tome je kôd boje dat u formatu ARGB (redom vrijednosti date u heksadecimalnom brojevnom sistemu u rasponu od 0 do 255, označavaju transparentnost od potpune do nepostojeće, intenzitet crvene, zelene i plave boje, respektivno).

Prozor specificiran XAML kôdom datim u listingu br. 8 moguće je vidjeti na slici br. 12. Prozor je transparentan- vidi se sadržaj prozora koji je "iza" njega i koji nije fokusiran. Tekst "TEST" dio je transparentnog prozora - kontrole koje su dio prozora ne naslijeđuju transparentnost (osim pozadinske boje).



Slika 12 - transparentni prozor u WPF

Jedina specifičnost implementacije WPF aplikacije koja preostaje jeste `WebBrowser` kontrola koja predstavlja ugrađeni web preglednik u aplikaciji. WPF donosi osnovnu varijantu ove kontrole koja je u suštini direktno vezana uz trenutnu verziju *Internet Explorera* instaliranu uz operativni sistem.

Kontrola se može koristiti bilo za standardnu navigaciju kroz web, bilo za prikaz HTML-a koji se kontroli prosljeđuje kao *string* upotrebom specifične `navigateToString` metode. Pri tome je važno napomenuti probleme sa kojima se suočava prilikom upotrebe ove kontrole.

Prvi problem `WebBrowser` kontrole predstavlja činjenica da se radi o maskiranom *Internet Exploreru* (u daljem tekstu: 'IE'). Još uvijek se često koristi verzija IE6 koja se pokazala izuzetno ranjivom i sa brojnim sigurnosnim propustima - a ukoliko se na korisničkom računar u koristi IE6, odgovarajuća kontrola u aplikaciji je IE6. Drugi problem u korištenju navedene kontrole su sigurnosne postavke IE-a koje sprječavaju preusmjerenje na lokalne sadržaje, što se manifestira kroz prijavljivanje besmislenih grešaka o neispravnosti *JavaScript* kôda korištenog na stranici, ili kroz nemogućnost preglednika da pristupi eksternim datotekama u kojima se taj kôd nalazi. Potrebno je eksplicitno skinuti navedenu zabranu kroz sučelje IE-a, kako bi preglednik počeo izvršavati *JavaScript*.

Dodatni problem predstavlja činjenica da je WPF osmišljen tako da ne podržava višenitnost (*threading*) u radu sa grafičkim korisničkim sučeljem (barem ne na bilo kakav način koji ne podrazumijeva primjenu trikova). Tačnije rečeno, postoji samo jedna programska nit koja se bavi iscrtaivanjem sučelja. Jasno, višenitnost je regularno podržana što se tiče rada sa podacima.

Također, *rendering* HTML-a u web pregledniku zahtijeva određeno vrijeme. To vrijeme iznosi par stotina milisekundi čak i za relativno male dokumente (posebno kada se ima u vidu činjenica da se moraju izvršiti skripte za prikaz matematičkih izraza i crtanje grafika). Ispostavlja se da je nepraktično napraviti aplikaciju koja bi mogla u stvarnom vremenu (bez zastoja) omogućiti korisniku unos sadržaja i njegov pravovremeni *render* u okviru web preglednika kao kontrole. S druge strane, to ne predstavlja osobit problem ukoliko se prihvati model prema kojem će se *rendering* obavljati tek na eksplicitan zahtjev korisnika.

Naravno, ovo nije tako značajno ograničenje jer ga se može na relativno *jeftin* način zaobići-limitirati brzinu kojom će se *render* osvježavati. Npr. trivijalna implementacija bi bila da se svakom izmjenom sadržaja dokumenta aktivira procedura *rendering*a novog dokumenta. To podrazumijeva utrošak određenog vremena da bi se cijeli *plaintext* konvertovao u LMD, a potom LMD pretvorio u HTML. Jedna od mogućih optimizacija bi bila da se pamte *delte* tj. izmjene dokumenta, što bi omogućilo brže procesiranje i generiranje novog HTML-a kojeg treba prikazati. Druga optimizacija je određivanje trenutka u kojem se *rendering* uopće aktivira- recimo, ukoliko osoba dovoljno brzo tipka, *rendering* bi trebalo aktivirati tek kada osoba napravi pauzu (kako ne bi zablokirao unos teksta). To naravno podrazumijeva određivanje kada prepoznati pauzu i gdje smjestiti proceduru kreiranja HTML-a. Najbrža varijanta je da se promjenom teksta šalje zahtjev za *renderingom*, pa kada se procijeni da je vrijeme za *rendering*, dodatna nit će pokrenuti generiranje HTML-a, i tek na kraju njegov *rendering*.

Još jedna specifičnost jeste i iritantni (i za korisnika neočekivani) zvuk "klika" kada se pokrene prikaz HTML-a u `WebBrowser` kontroli. Istog je potrebno isključiti ili kroz postavke IE-a, ili programski na način kako je dato listingom br. 9.

```

#region remove clicking sound
    private const int FEATURE_DISABLE_NAVIGATION_SOUNDS = 21;
    private const int SET_FEATURE_ON_THREAD = 0x00000001;
    private const int SET_FEATURE_ON_PROCESS = 0x00000002;
    private const int SET_FEATURE_IN_REGISTRY = 0x00000004;
    private const int SET_FEATURE_ON_THREAD_LOCALMACHINE = 0x00000008;
    private const int SET_FEATURE_ON_THREAD_INTRANET = 0x00000010;
    private const int SET_FEATURE_ON_THREAD_TRUSTED = 0x00000020;
    private const int SET_FEATURE_ON_THREAD_INTERNET = 0x00000040;
    private const int SET_FEATURE_ON_THREAD_RESTRICTED = 0x00000080;

    [DllImport("urlmon.dll")]
    [PreserveSig]
    [return: MarshalAs(UnmanagedType.Error)]
    static extern int CoInternetSetFeatureEnabled(
        int FeatureEntry,
        [MarshalAs(UnmanagedType.U4)] int dwFlags,
        bool fEnable);
#endregion

...
int feature = FEATURE_DISABLE_NAVIGATION_SOUNDS;
CoInternetSetFeatureEnabled(feature, SET_FEATURE_ON_PROCESS, true);

```

Listing 9 - kôd za ukidanje zvukova pri navigaciji u *webbrowser* kontroli

Listingom br. 9 dat je isječak kôda kojim se isključuju zvukovi (klik) pri navigaciji u *WebBrowser* kontroli. Konkretno, ovo će spriječiti nepredviđeni zvuk prilikom iniciranja *renderinga* dokumenta, i to na nivou procesa (trenutnog). Priložen je i niz konstanti kojima je moguće regulirati nivo na kojem će se blokiranje zvuka aplicirati (nit, proces, sistemski *registry* i sl.). Isključivanje se obavlja pozivom odgovarajuće metode koja je uvezena iz sistemskog DLL-a.

Još jedan od problema sa performansama javlja se i kod ozbiljnije upotrebe *RichTextBox* kontrole koja iz autoru neobjašnjivih razloga pokazuje slabije performanse pri određenim postavkama ponašanja aplikacije. Pri nešto bržem kucanju od običnog, pojavljuje se primijetno zastajkivanje (reda do 200 milisekundi) ukoliko je aplikacija konfigurirana na način da se može izvršavati na bilo kojoj od procesorskih jezgri. Ukoliko se eksplicitno naredi izvršavanje na jednoj jezgri (npr. prvoj), performanse se značajno poboljšavaju i problematično zastajkivanje se više ne pojavljuje. Listingom br. 10 dat je odgovarajući kôd koji rješava problem zastajkivanja programa tokom bržeg unosa teksta u WPF *RichTextBox* kontrolu.

```

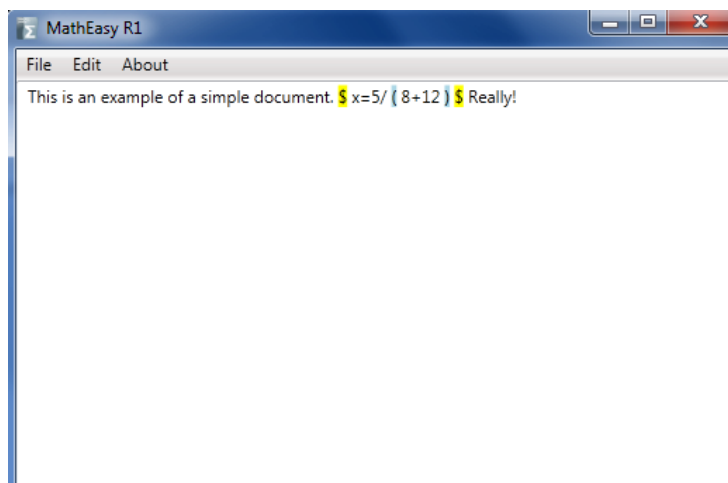
System.Diagnostics.Process.GetCurrentProcess().ProcessorAffinity = (System.IntPtr)1;

```

Listing 10 - kôd za podešavanje procesorskog afiniteta

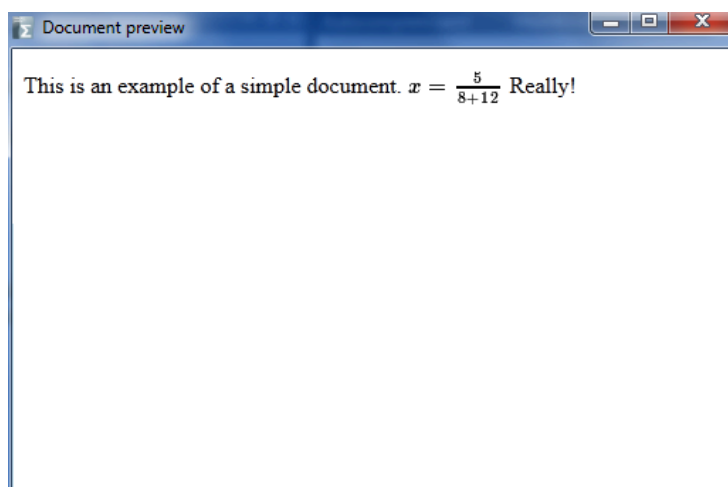
Dizajn korisničkog sučelja je minimalistički i ne zahtijeva objašnjenja detaljnija od objašnjenja datih u okviru korisničkih uputa (poglavlje 7, odjeljak 1). U nastavku su na slikama 13, 14 i 15 date slike specifičnih ekrana.

Na slici br. 13 data je slika radnog prostora. Na izboru korisniku nalazi se glavni meni aplikacije i polje za unos teksta. Automatski se obavlja se dizajnom određene operacije (npr. bojenje zagrada).



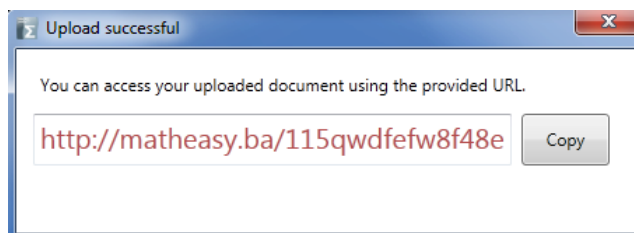
Slika 13 - glavni radni prostor MathEasy aplikacije

Na slici br. 14 data je slika prozora u kojem se prikazuje *renderirani* dokument. *Renderiranje* mora eksplicitno zahtijevati sâm korisnik kroz opciju File→Preview:



Slika 14 - *renderirani* dokument

Na slici br. 15 data je slika prozora koji daje URL za pristup *uploadovanom* dokumentu:

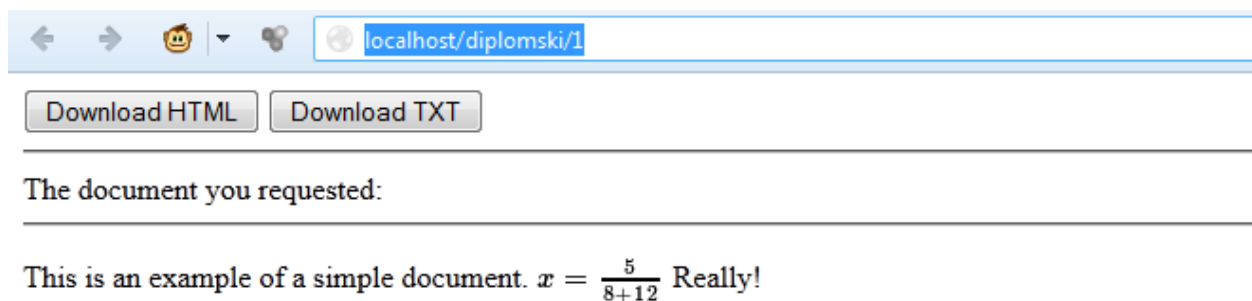


Slika 15 - rezultat uploada dokumenta (primjer)

### 4.3. Specifičnosti i razvoj serverskog dijela aplikacije

Osnovica serverskog dijela aplikacije su jednostavne skripte koje služe za obradu zahtjeva sa strane desktop aplikacije (HTTP GET, HTTP POST) i obradu zahtjeva koji dolaze putem web preglednika (HTTP GET). Osnovne akcije koje navedene skripte obavljaju su dakle pisanje (*upload dokumenta*) i čitanje (bilo iz aplikacije- *download* ili putem preglednika). Svaki dokument identificiran je svojim jedinstvenim URL-om koji sadrži identifikator na osnovu kojeg se pristupa dokumentu. Pri tome se koristi i maskiranje URL-ova kako bi se pristup ostvarivao uniformno (*Apache* modul *mod\_rewrite* [36]).

Dizajn stranice je bazičan i sastoji se od linkova za preuzimanje HTML ili TXT datoteke i prikaza odgovarajućeg dokumenta, kao što je i prikazano na slici br. 16:



Slika 16 - pristup dokumentu putem web preglednika

Drugih napomena u vezi sa serverskom aplikacijom nema budući da se koriste osnovna znanja stečena tokom studija (predmet *Web tehnologije* sa pripadajućim predavanjima i tutorijalima u akademskoj 2012/2013 godini [37]).

### 4.4. Testiranje

Kao što je već navedeno u poglavlju 3.2 koje se bavi razmatranjem koji pristup koristiti za dizajn aplikacije, aplikacija i model koji su prezentirani u okviru ovog rada su rezultat iskustva autora koji je problem rada sa dokumentima matematičkog sadržaja pokušao riješiti na više načina tokom šest mjeseci koliko je radio na ovom radu. Pri tome se u posljednjoj iteraciji koja se ovdje prezentira autor vodio stečenim iskustvom da je dobra praksa prvo specificirati testove za komponentu, pa je tek potom implementirati.

Zadržalo se na najjednostavnijem načinu testiranja- *unit* testiranju koje se bavi testiranjem pojedinačnih funkcionalnih jedinica sistema (npr. metoda klása). Kako je aplikacija koju se razvija najvećim dijelom i zasnovana na logici pretvaranja običnog teksta u određeni format i obradi aritmetičkih izraza, to se *unit* testiranje pokazuje sasvim dovoljnim. Korisničko sučelje ne sadrži mnogo funkcionalnosti niti složenih ili slabo strukturiranih interakcija, tako da se testiranje sučelja nema potrebe obavljati na sistematičan način (tj. obavljalo se po principu pokretanja programa i pokušavanja da se program krahira ili izazove nepredviđeno ponašanje).

Iz elemenata grafičkog korisničkog sučelja nastojalo se ukloniti izvršnu logiku kako bi se navedene komponente moglo testirati.

Pri tome su elementi LMD komponente najdetaljnije testirani (praktično, razvoj LMD komponente bio je upravljan unaprijed dizajniranim testovima), dok preostale komponente su pokrivene samo osnovnim testovima, i to nakon što su implementirane.

Za testiranje je korišten nUnit framework. Sve testne klase bile su pohranjene u zasebnom testnom folderu [38]. Svaki put su se pokretali svi testovi kako bi se provjerilo ponašanje sistema nakon izmjena, što se može smatrati na nesistematičan način provođenim testiranjem na regresiju.

nUnit framework poslužio je i za testiranje vremena odziva pojedinih komponenti sistema u testnim slučajevima, a na osnovu čega su uočene mogućnosti za poboljšanje i optimizaciju programskog kôda kako bi se ispunili nefunkcionalni zahtjevi aplikacije. Navedeno se na jednostavan način ostvaruje primjenom sljedećeg isječka kôda datog listingom br. 11.

```
Assert.That(očekivano, Is.EqualTo(rezultat).After(broj_sekundi, interval_provjere));
```

Listing 11 - kôd koji pored ispravnosti testira i vrijeme odziva

Pri tome metoda `After` služi kako bi se provjerilo da li je rezultat jednak očekivanoj vrijednosti nakon datog broja milisekundi (prvi parametar metode `After`), dok se navedena provjera obavlja svakih  $x$  milisekundi, gdje je  $x$  dato kao drugi parametar metode `After`.

## 4.5. Zaključak

U ovom poglavlju obrazložen je izbor razvojnog okruženja i tehnologija koje su korištene u razvoju aplikacije. Opisani su, prema autorovom mišljenju, najkarakterističniji detalji iz razvoja WPF aplikacije, a u odnosu na Windows Forms programski model. Navedeno može poslužiti kao jedan od dodatnih rezultata ovog rada.

Također, razmotren je način na koji se verificiralo da aplikacija radi korektno, tj. da su sve funkcionalnosti aplikacije realizirane u skladu sa specifikacijom. Pri tome je testiranje zbog obima rada svedeno na *unit* testiranje (*white-box testiranje*) najvažnijih komponenti, dok validacija i funkcionalno testiranje (*black-box testiranje*) nisu strukturirano testirani.

Razvoj najsloženije komponente u aplikaciji (LMD komponenta) bio je upravljan unaprijed definiranim testovima. Već u fazi dizajna testova naišlo se na ozbiljne probleme u dizajnu LMD komponente i njenih elemenata. Najčešći problemi su bili 1) loše osmišljena raspodjela poslova 2) nedostajuće funkcionalnosti. Upotrebom navedenog pristupa testiranju autor je uštedio vrijeme implementacije od početka loše osmišljenih komponenti.

Testiranjem su uočeni i nedostaci u smislu performansi metode kojom se boje uparene zgrade u dokumentu. Nedostaci su uklonjeni implementacijom strategije po kojoj se bojenje zgrada obavlja dio po dio, a ne odjednom, čime je značajno poboljšan odziv aplikacije.

## 5. Zaključak

### Pregled urađenog

U okviru ovog završnog rada urađeno je sljedeće:

- Analiziran je problem rada sa dokumentima matematičkog sadržaja na desktop računarima.
- Prezentirana su postojeća programska rješenja, njihove prednosti i nedostaci.
- Prezentirani su mogući pristupi u kreiranju vlastitog programskog rješenja.
- Specificirani su funkcionalni i nefunkcionalni zahtjevi.
- Odabran je i modificiran jedan od prezentiranih pristupa, nakon čega se pristupilo dizajnu odgovarajućeg programskog rješenja.
- Odabrani su alati koji služe kao ispomoć za rješavanje specifičnih problema (*rendering* matematičkih izraza, *plotanje* grafika).
- Za implementaciju je odabran programski jezik C# i *Windows Presentation Foundation* programski model.
- Prezentirane su određene i za ovaj rad relevantne opće specifičnosti WPF programskog modela.
- Opisan je način na koji je obavljeno testiranje aplikacije.
- Razvijena je aplikacija koja odgovara postavljenim funkcionalnim i nefunkcionalnim zahtjevima.

Autor rada smatra posebno važnim sljedeće rezultate rada:

- Autor se po prvi put tokom svog studija susreo sa problemima koji su specifični za razvoj aplikacije koja nije poslovna po prirodi, tj. ne rješava zadati poslovni problem već služi kao *utility* ili pomoćna- sistemska aplikacija.
- Autor se također po prvi put susreo sa problemima specifičnima za razvoj aplikacije koju se može klasificirati kao generički softver. Poznata je podjela softvera na generički (poput *MS Office*, operativnih sistema i sl.) i prilagođenog (jedan svoje vrste, za specifičnu primjenu i jedinstven poslovni ili tehnički problem), gdje generičkom softveru funkcije određuje razvojni tim, a prilagođenom klijent [22].
- Autor prije ovog završnog rada nije u potpunosti poznao sve složenosti programskih rješenja za uređivanje teksta. Tek kada je pokušao napraviti svoje programsko rješenje, uvidio je koliko se funkcija i procesa posmatra "zdravo za gotovo" i koliko su *utility* aplikacije složene za napraviti.
- Autor je naučio koristiti LaTeX.
- Autor je naučio koristiti MathML.
- Autor je naučio specifičnosti WPF programskog modela.

## Budući razvoj

Desktop i serverski dio sistema razvijeni za potrebe ovog rada mogu se smatrati isključivo demonstratorom, a nikako potpunim i završenim programskim rješenjem. Pri tome za *ozbiljnu* upotrebu ovakvog rješenja treba minimalno uraditi sljedeće:

- Zaštititi serverski dio sistema od *uploada* malicioznih sadržaja.
- Implementirati osvježavanje prikaza *renderiranog* dokumenta u stvarnom vremenu.
- Implementirati mogućnost promjene fonta i veličine slova.
- Omogućiti učitavanje sadržaja iz HTML dokumenta kako bi se isti mogao uređivati.
- Omogućiti upravljanje konfiguracijom aplikacije kroz korisničko sučelje.
- Poboljšati performanse metode za bojenje zagrada u dokumentu.

Svi prijedlozi ili samostalne implementacije poboljšanja MathEasy aplikacije svakako su dobrodošli.



## 6. Prilozi

### 6.1. Upute za korištenje aplikacije

U nastavku su date sve informacije potrebne za korištenje funkcija koje pruža MathEasy. Informacije i upute su logički organizirane u odjeljke vezane uz specifične funkcionalnosti ili osobine.

#### 6.1.1. Instalacija i konfiguracija

MathEasy aplikacija dolazi u vidu instalacijskog programa koji će odgovarajuće datoteke smjestiti na željeno mjesto na lokalnom disku i ukoliko to korisnik želi, kreirati prečac na desktopu. Aplikacija se smješta u folder /MathEasy i u njemu se nalaze dvije datoteke: izvršna (.exe) datoteka MathEasy aplikacije i konfiguracijska (.config) tekstualna datoteka koja sadrži postavke aplikacije.

Ukoliko korisnik želi dodati vlastite simbole kako bi ih mogao ubacivati u dokument primjenom *autocomplete* funkcionalnosti, dovoljno je da otvori konfiguracijsku datoteku u bilo kojem uređivaču teksta i doda za svaki simbol novi red u kojem je prvi znak simbol kojeg se želi koristiti, potom jedan razmak, i nakon toga niz znakova koji odgovara tom simbolu i koji je smislen za korisnika.

#### 6.1.2. Meni opcije

Aplikacija se sastoji od glavnog menija i polja za unos teksta (aktivni dokument- radni prostor).

Glavni meni sadrži opcije *File*, *Edit* i *About*.

Meni *File* sadrži opcije *New*, *Open*, *Download*, *Save*, *Upload*, *Preview* i *Exit*.

- *New* - otvara novi prazan dokument. Ukoliko trenutno aktivni dokument nije prazan, korisnika se pita da li želi spasiti stari dokument i nastaviti sa kreiranjem novog praznog dokumenta. Kratica na tastaturi je CTRL+N.
- *Open* - otvara dijaloški okvir za izbor lokacije sa koje se želi učitati/otvoriti .TXT dokument. Ukoliko trenutno aktivni dokument nije prazan, korisnika se pita da li želi spasiti stari dokument i nastaviti sa otvaranjem novog dokumenta. Kratica na tastaturi je CTRL+O.
- *Download*- otvara dijaloški okvir za unos pristupnog linka za datoteku koja se nalazi na serveru. Ukoliko trenutno aktivni dokument nije prazan, korisnika se pita da li želi spasiti stari dokument i nastaviti sa otvaranjem novog dokumenta. Ukoliko nema konekcije sa serverom, ili je link neispravan, prijavljuje se greška. Kratica na tastaturi je CTRL+D.
- *Save* - otvara dijaloški okvir za izbor lokacije i imena datoteke koju se želi spasiti. Također se određuje format u kojem se trenutni dokument spašava- izbor je moguć između HTML i TXT formata. Pri tome dokument spašen kao HTML nije moguće kasnije učitati i uređivati kroz MathEasy . Kratica na tastaturi je CTRL+S.
- *Upload*- izborom ove opcije automatski počinje *upload* trenutnog dokumenta iz radnog prostora na server. Ukoliko dođe do problema prilikom postavljanja dokumenta bit će prijavljena greška, u suprotnom se korisniku prikazuje dijaloški okvir koji sadrži jedinstveni pristupni link pomoću kojeg je moguće pristupiti *uploadovanom* dokumentu. Kratica na tastaturi je CTRL+u.
- *Preview*- izborom ove opcije u ugrađenom web pregledniku se prikazuje kreirani dokument. Kratica na tastaturi je CTRL+P.

- Exit- izborom ove opcije izlazi se iz aplikacije. Ukoliko trenutno aktivni dokument nije prazan, korisnika se pita da li želi spasiti dokument prije nego napusti aplikaciju. Kratica na tastaturi je ALT+F4.

Meni *Edit* sadrži opcije: *Undo, Redo, Cut, Copy, Paste, Delete, Find & Replace*.

- Undo - Obavlja povratak jedan korak unazad kroz historiju uređivanja dokumenta, ukoliko je to moguće učiniti. Historija uređivanja dokumenta čuva se na nivou unosa pojedinih znakova, ili cijele grupe znakova kada se radi o upotrebi komandi poput # i ?. Kratica na tastaturi je CTRL+Z.
- Redo - Obavlja kretanje jedan korak unaprijed kroz historiju uređivanja dokumenata, ukoliko je to moguće učiniti. Historija uređivanja dokumenta čuva se na nivou unosa pojedinih znakova, ili cijele grupe znakova kada se radi o upotrebi komandi poput # i ?. Kratica na tastaturi je CTRL+Y.
- Cut - Izrezuje trenutno selektirani dio dokumenta, ukoliko je to moguće učiniti. Kratica na tastaturi je CTRL+X.
- Copy - Kopira trenutno selektirani dio dokumenta, ukoliko je to moguće učiniti. Kratica na tastaturi je CTRL+C.
- Paste - Lijepi sadržaj iz sistemskog međuspremnika, ukoliko je to moguće učiniti, na dato mjesto u dokumentu. Sadržaj mora biti obični tekst. Sve formatiranje teksta se uklanja prilikom lijepljenja. Kratica na tastaturi je CTRL+V
- Delete - Briše trenutno selektirani dio dokumenta, ukoliko je to moguće učiniti. Kratica na tastaturi je DELETE.
- Find & Replace - Pokreće dijaloški okvir u kojega se unosi tekst kojeg treba zamijeniti i tekst kojim se navedeni tekst treba zamijeniti. Kratica na tastaturi je CTRL+F.

Meni opcija *About* prikazuje osnovne informacije o aplikaciji i njenom autoru.

### **6.1.3. Uređivanje dokumenta**

Rad sa tekstualnim dokumentom osnovna je namjena MathEasy aplikacije. Sveukupan rad s dokumentom odvija se putem radnog prostora- polja za unos i rad sa tekstom. Predefinirana veličina fonta kojeg se koristi je 12. Podržani su svi UTF-8 karakteri. Font niti njegovu veličinu u ovoj verziji MathEasy aplikacije nije moguće mijenjati.

Obični tekst unosi se kao u bilo kojem drugom uređivaču teksta. Ipak, MathEasy se razlikuje od ostalih aplikacija po svojim mogućnostima specifičnima za rad sa dokumentima matematičkog sadržaja. Opis i način ostvarivanja funkcionalnosti dat je u sljedećim pododjeljcima. Svi posebni simboli (npr. uparene zagrade), ključne riječi i matematička područja teksta automatski se boje odgovarajućim bojama, gdje je boja tamnija što je ugnježđenje dublje.

#### **6.1.3.1. Autocomplete**

Autocomplete omogućava korisniku da brzo i jednostavno pronađe nestandardni simbol i ubaci ga u dokument. Nestandardne simbole specificira se kroz konfiguracijsku datoteku- korisnik može jednostavno pronaći željeni simbol i odrediti kraticu koju želi koristiti za njega, ubaciti ih u vlastitu konfiguracijsku datoteku i nakon ponovnog pokretanja MathEasy je spreman ponuditi korisniku unos novog simbola.

Autocomplete funkcionalnost moguće je pokrenuti u bilo kojem trenutku unosom simbola '\ ' (Alt Gr + Q pri bosanskohercegovačkim postavkama tastature). Pojaviti će se prozor koji nestaje pritiskom na tipku Escape, ili unosom razmaka. U tom slučaju simbol '\ ' ostaje kao sadržaj dokumenta. U suprotnom, na samom početku prikazani su svi raspoloživi simboli. Moguće ih je filtrirati daljim unosom slova, npr. '\ ' i potom 'a' će ostaviti prikazanim samo one simbole čija kratica počinje slovom 'a'. Kroz prikazane simbole moguće se kretati kursorskim tipkama (gore, dole, lijevo, desno), ili odabrati željeni simbol pomoću miša. Na poziciji u dokumentu u kojoj je unesen simbol '\ ' pojaviti će se odabrani simbol. Od izbora je u svakom trenutku moguće odustati.

#### 6.1.3.2. Rad sa matematičkim izrazima

Dijelovi teksta koje se želi interpretirati kao matematička područja potrebno je na odgovarajući način otvoriti i zatvoriti upotrebom simbola '\$'. Ukoliko korisnik želi unijeti simbol '\$' koji se ne interpretira kao otvaranje ili zatvaranje matematičkog područja, mora ga navesti sa kosom crtom ispred: '\\$'. Pri tome se određeno područje može odnositi samo na jedan izraz ili jednačinu. U tabeli br. 5 dati su odgovarajući primjeri. Također, posebne komande eval() i plot() nije dozvoljeno koristiti u okviru matematičkog područja dokumenta. Pri tome korisnik treba voditi računa da komande ? i # koristi isključivo u okviru pravilno ograničenog matematičkog područja, u suprotnom neće imati nikakvog efekta.

Primjer	Značenje i efekat	Render
<b>\$ x = 50 / 5 = 10 \$</b>	Deklariran je simbol x sa vrijednosti 10.	$x = \frac{50}{5} = 10$
<b>\$ 50 / 5 \$</b>	Prosti ispis matematičkog izraza.	$\frac{50}{5}$
<b>\$x=5\$      \$y=10\$ \$z=x+y\$</b>	Prvo je deklariran simbol x sa vrijednosti 5. Potom je deklariran simbol y sa vrijednosti 10. Na kraju je deklariran simbol z čija vrijednost je jednaka zbiru vrijednosti simbola x i y.	$x = 5 \quad y = 10 \quad z = x + y$
<b>\$x = a b\$</b>	Neispravan matematički izraz, tretira se kao obični tekst.	$\$x=a b\$$
<b>\$x = a. Ovo je neispravno.</b>	Neispravno zatvoreno matematičko područje - sav tekst tretira se kao matematika, i to neispravna, pa je kao rezultat ispisan obični tekst.	$\$x = a. Ovo je neispravno.$
<b>\$ x = 5 \$. \\$ je simbol za USD.</b>	Deklariran je simbol x sa vrijednosti 5.	$x = 5. \$$ je simbol za USD.

Tabela 5 - neki primjeri matematičkih izraza, sintaksa, značenje i render

### 6.1.3.3. Brzo kopiranje prvog prethodnog izraza

U okviru ispravno otvorenog i zatvorenog područja teksta koji se interpretira kao matematika moguće je koristiti funkcionalnost brzog kopiranja prvog izraza ispred prethodećeg znaka jednakosti. Ova funkcionalnost predstavlja samo jednostavniji i brži način kopiranja dijela teksta, a ostvaruje se onog trenutka kada se unese simbol '#' u okviru ispravno ograničenog područja matematičkog teksta, iza znaka '='.

Primjer	Rezultat nakon unosa simbola #	Primjedba
$x=5+7*3= \# \$$	$x=5+7*3= 5+7*3\$$	Na mjesto simbola # ubacuje se prvi prethodni izraz.
$x=5++++\# \$$	$x=5++++\# \$$	Ispred simbola # nije bio znak jednakosti, pa njegov unos nije imao efekta.
$\# \$$	$\# \$$	Ispred simbola # nije bio znak jednakosti, pa njegov unos nije imao efekta.
$\$=\# \$$	$\$=\$$	Ispred simbola # nalazi se simbol =, a prethodi mu prazan izraz koji se ubacuje na njegovo mjesto.

Tabela 6 - neki primjeri upotrebe komande za brzo kopiranje '#'

### 6.1.3.4. Evaluacija prvog prethodnog izraza

U okviru ispravno otvorenog i zatvorenog područja teksta koji se interpretira kao matematika moguće je koristiti funkcionalnost evaluacije prvog prethodnog izraza ispred prethodećeg znaka jednakosti. Ova funkcionalnost ostvaruje se onog trenutka kada se unese simbol '?' u okviru ispravno ograničenog područja matematičkog teksta, iza znaka '='.

Primjer	Rezultat nakon unosa simbola ?	Primjedba
$x=5+7*3= ? \$$	$x=5+7*3= 26 \$$	Na mjesto simbola ? ubacuje se evaluacija prvog prethodnog izraza.
$x=5++++? \$$	$x=5++++? \$$	Ispred simbola ? nije bio znak jednakosti, pa njegov unos nije imao efekta.
$? \$$	$? \$$	Ispred simbola ? nije bio znak jednakosti, pa njegov unos nije imao efekta.
$\$=? \$$	$\$=? \$$	Ispred simbola ? nalazi se simbol =, a prethodi mu prazan izraz koji se ubacuje na njegovo mjesto.

Tabela 7 - neki primjeri upotrebe komande za evaluaciju izraza '?'

### 6.1.3.5. Funkcije koje se izvršavaju prilikom renderinga dokumenta

U okviru običnog (nematematskog) teksta moguće je koristiti funkcije `eval(expr)` i `plot(expr, od, do, brojtačaka)` da bi se prilikom *renderinga* na odgovarajućim mjestima u dokumentu prikazao rezultat evaluacije izraza, odnosno, grafik funkcije sa jednom varijablom datom kao izraz, u zadanom opsegu i sa zadatim brojem tačaka.

Primjer	Render
Iz toga slijedi da je <code>x=eval&lt;10*5/4*2&gt;</code> .	Iz toga slijedi da je $x=25$ .
Još jedan primjer je <code>eval&lt;(5+3)/z&gt;</code>	Još jedan primjer je $\frac{8}{z}$
Neispravan <code>eval:</code> <code>eval&lt;5,5&gt;</code>	Neispravan eval: eval<5,5>
Evo i grafika: <code>plot&lt;x+5,0,5,10&gt;</code>	<p>Evo i grafika:</p>
Još jedan grafik <code>plot&lt;x^2+y^2,0,5,1000&gt;</code>	Još jedan grafik <code>plot&lt;x^2+y^2,0,5,1000&gt;</code>
Grafik sa neispravnim parametrima: <code>plot&lt;x+5&gt;</code>	Grafik sa neispravnim parametrima: <code>plot&lt;x+5&gt;</code>

## 6.2. Rječnik pojmova i skraćenica

U sklopu ovog odjeljka daju se direktni prijevodi ili opisne definicije pojmova i skraćenica koje su često korištene u radu.

Pojam	Prijevod ili opisna definicija
<b>Rendering</b>	Generiranje prikaza (slike) na osnovu nekog modela, a pomoću računarskih programa. Renderira se npr. slika 3D objekta na osnovu kreiranog modela, ili prikaz web stranice na osnovu zadatog HTML-a.
<b>Evaluacija</b>	Određivanje vrijednosti ili izvršavanje naredbe/uvažavanje operatora
<b>Plotanje</b>	Crtanje grafika na osnovu datih ulaznih podataka pomoći računarskog programa
<b>Plaintext</b>	Obični tekst bez bilo kakvog primijenjenog formatiranja
<b>Upload</b>	Proces slanja informacijskog toka sa lokalnog računara na destinaciju na mreži
<b>Download</b>	Proces preuzimanja informacijskog toka sa izvora na mreži na lokalni računar
<b>HTML</b>	Hypertext Markup Language - markup jezik za kreiranje web dokumenata i ostalih sadržaja koje je moguće prikazivati u web pregledniku
<b>Markup jezik</b>	Jezik koji odvaja podatke od metapodataka kojima se npr. definira logička struktura ili način prikaza
<b>WYSIWYG</b>	What You See Is What You Get- akronim koji se koristi kada se želi istaći da ono što korisnik uređuje i vidi 1:1 odgovara onome što će biti odštampano ili prikazano na ekranu, bez razlike.
<b>WYSIWYM</b>	What You See Is What You Mean- akronim koji se koristi kada se želi istaći da ono što korisnik uređuje i vidi <i>otprilike</i> odgovara onome što će biti odštampano ili prikazano na ekranu, sa određenim razlikama u stilu i načinu prikaza.
<b>Test Driven Development</b>	Razvoj softverskih komponenti koji je upravljan unaprijed spremljenim testovima. Implementacija se svodi na programiranje dok svi spremljeni testovi ne prolaze.
<b>.NET framework</b>	Softverski framework kojeg je razvio Microsoft za Windows operative sisteme. Uključuje obimnu biblioteku komponenti i omogućava razvoj Windows aplikacija u više programskih jezika.
<b>C#</b>	Programski jezik zasnovan na C++, jedan od jezika koje promoviše i održava Microsoft i koji je najpopularniji jezik za razvoj .NET aplikacija
<b>Web browser (preglednik)</b>	Aplikacija koja renderira HTML dokumente
<b>PHP</b>	PHP Hypertext Preprocessor, jedan od najpopularnijih skriptnih jezika koji se izvršava na serveru.
<b>URL</b>	Uniform Resource Locator, niz znakova koji u cjelini čini referencu na resurs raspoloživ putem mreže.
<b>Inline</b>	"U liniji"- odnosi se na logički sadržaj jednog paragrafa. U standardnom formatu dokumenata, dokument je sačinjen iz poglavlja, poglavlja su sačinjena od sekcija ili odjeljaka, a svaki odjeljak sačinjen iz više blokova (npr. paragraf je blok), a gdje svaki paragraf sadrži inlines: tekst, slike, reference i sl.

Tabela 8 - rječnik pojmova

## 7. Korištena literatura i vanjski resursi

- 1) Brown, CM (1988). Human-computer interface design guidelines. Norwood, NJ: Ablex Publishing
- 2) Ayres, Robert U; Martinás, Katalin (2005), "120 wpm for very skilled typist", On the Reappraisal of Microeconomics: Economic Growth and Change in a Material World, Cheltenham, UK & Northampton, Massachusetts: Edward Elgar Publishing, p. 41, ISBN 1-84542-272-4, 22 Novembar 2010
- 3) Notepad++, napredni uređivač običnog teksta. Više informacija na web stranici projekta: <http://notepad-plus-plus.org/>
- 4) Uređivač LaTeX dokumenata. Više informacija na web stranici: <http://www.latexeditor.org/>
- 5) Uređivač LaTeX dokumenata. Više informacija na web stranici: <http://www.winedt.com/>
- 6) MS Word dio je uredskog paketa alata kojeg izdaje Microsoft pod nazivom Microsoft Office. Više informacija na web stranici: <http://office.microsoft.com/en-us/>
- 7) Writer je dio uredskog paketa alata otvorenog kôda (uz to i besplatnog) kojeg održava The Document Foundation. Više informacija na web stranici: <https://www.libreoffice.org/>
- 8) MATLAB je interaktivno okruženje za numeričke izračune, vizualizaciju i programiranje proizvođača. Više informacija na web stranici proizvođača MATLAB-a, firme Mathworks: <http://www.mathworks.com/products/matlab/>
- 9) Mathematica je softver koji obavlja izračune, a koristi se u svim područjima u kojima je potrebno tehničko izračunavanje. Više informacija na web stranici proizvođača, firme Wolfram Research: <http://www.wolfram.com/mathematica/>
- 10) Wolfram Alpha tražilica ("computational knowledge engine"), moguće joj je pristupiti putem sljedeće adrese <http://www.wolframalpha.com/>
- 11) Kroz Google tražilicu i nju ugrađeni kalkulator moguće je izvršavati jednostavne izračune: <http://www.google.com/help/features.html>
- 12) Espresso je jednostavan uređivač teksta koji omogućava obavljanje izračuna u okviru dokumenta. Više informacija na web stranici proizvođača, firme DeepIT: <http://deepitpro.com/en/mac/products/Espresso/info/index.shtml>
- 13) TeX je sistem za obradu i prijelom teksta. Iako na internetu postoji veliki broj izvora podataka o TeX-u, najkonkretniji je članak na Wikipediji, uz konsultaciju na Wikipediji navedenih referenci: <https://en.wikipedia.org/wiki/TeX>
- 14) Pavi Sandhu Cengage, The MathML Handbook, Charles River Media, 2003
- 15) Objašnjenje arhitekture LaTeX-a (a posredno i TeX-a ) sa foruma posvećenog TeX-u: <http://tex.stackexchange.com/questions/3274/latex-architecture-how-does-it-all-work>
- 16) LyX, WYSIWYM obrađivač (La)TeX dokumenata. Više informacija na web stranici: <http://www.lyx.org/>
- 17) Uređivač MathML sadržaja, dolazi u više varijanti. Više informacija na web stranici: <http://www.dessci.com/en/products/mathflow/>
- 18) Primjer je preuzet sa Wikipedijinog unosa o MathML-u, odjeljak sa usporedbama: [http://en.wikipedia.org/wiki/MathML#Example\\_and\\_comparison\\_to\\_other\\_formats](http://en.wikipedia.org/wiki/MathML#Example_and_comparison_to_other_formats)
- 19) Više informacija o MathJax rendering engineu: <http://www.mathjax.org/>
- 20) Više informacija o jednostavnom kalkulatoru ugrađenom u MS Word: <http://office-watch.com/t/n.aspx?a=592>

- 21) Dženana Đonko, Samir Omanović, Objektno orijentirana analiza i dizajn primjenom UML notacije, Elektrotehnički fakultet u Sarajevu, 2010
- 22) Ian Sommerville, Software Engineering, Addison Wesley, 2010
- 23) Test Driven Development, [http://en.wikipedia.org/wiki/Test-driven\\_development](http://en.wikipedia.org/wiki/Test-driven_development)
- 24) Željko Jurić, Diskretna matematika za studente tehničkih nauka, Elektrotehnički fakultet u Sarajevu, 2011
- 25) Haber kutija nalazi se na web stranici studenata Elektrotehničkog fakulteta u Sarajevu, [www.etf.ba](http://www.etf.ba)
- 26) Murray Sargent III, Unocode Nearly Plaint-Text Encoding of Mathematics, Office Authoring Services, Microsoft Corporation, 2006.
- 27) LaTeX Command Summary, holandska grupa posvećena (La)TeX-u, raspoloživo putem sljedećeg linka: [www.ntg.nl/doc/biemesderfer/ltxcnib.pdf](http://www.ntg.nl/doc/biemesderfer/ltxcnib.pdf)
- 28) Shunting-yard algoritam, osnovne informacije: [http://en.wikipedia.org/wiki/Shunting-yard\\_algorithm](http://en.wikipedia.org/wiki/Shunting-yard_algorithm)
- 29) Implementacija modificiranog Shunting-yard algoritma raspoloživa je putem sljedećeg linka: [http://en.literateprograms.org/Shunting-yard\\_algorithm\\_%28C%29](http://en.literateprograms.org/Shunting-yard_algorithm_%28C%29)
- 30) Dr. E.W. Dijkstra, ALGOL-60 Translation, ALGOL Bulletin Supplement nr 10 (*Shunting-yard* algoritam)
- 31) jqPlot je dodatak za jQuery koji omogućava jednostavno plotanje grafika u okviru web dokumenata. Više informacija moguće je dobiti putem sljedećeg linka: <http://www.jqplot.com/>
- 32) jQuery je multiplatformska JavaScript biblioteka. Više informacija moguće je dobiti putem sljedećeg linka: <http://jquery.com/>
- 33) Više o usmjeravanim događajima (routed events) putem MSDN (microsoftova stranica za pomoć developerima): <http://msdn.microsoft.com/en-us/library/ms742806.aspx>
- 34) Upute kako kreirati korisnički definirani routed event moguće je dobiti putem MSDN-a: <http://msdn.microsoft.com/en-us/library/ms752288.aspx>
- 35) Više informacija o WPF komandama moguće je dobiti putem MSDN-a: <http://msdn.microsoft.com/en-us/library/ms752308.aspx>
- 36) Detaljnije o mod\_rewrite modulu za Apache moguće je dobiti putem sljedećeg linka: [http://httpd.apache.org/docs/current/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/current/mod/mod_rewrite.html)
- 37) Web tehnologije, u trenutku pisanja ovog rada predmet u 6. semestru studija na odsjeku za RI na Elektrotehničkom fakultetu u Sarajevu. Materijali za predmet, a koji su autoru poslužili prilikom implementacije aplikacije raspoloživi su putem sljedećeg linka: <http://c2.etf.unsa.ba/course/view.php?id=119>
- 38) NUnit framework omogućava unit testiranje aplikacija koje se razvija kroz Visual Studio upotrebom C# programskog jezika. Više informacija moguće je dobiti putem sljedećeg linka: <http://www.nunit.org/>

Napomena: sve web reference su citirane po njihovom sadržaju na dan 3.7.2013. godine.



## Indeks dijagrama

Dijagram 1- prijedlog logičke organizacije dokumenta za WYSIWYG uređivač .....	19
Dijagram 2 - prijedlog logičke organizacije matematičkog objekta "razlomak" .....	20
Dijagram 3 - prilagođena struktura stabla .....	21
Dijagram 4 - dijagram slučajeva upotrebe .....	28
Dijagram 5 - raspoređivanje komponenti sistema na klijentski i serverski računar .....	32
Dijagram 6 - komponente MathEasy aplikacije .....	33
Dijagram 8 - primjer stabla aritmetičkog izraza .....	34
Dijagram 7 - LMD format .....	34
Dijagram 9 - upload TXT i HTML datoteke na mrežni servis .....	36
Dijagram 10 - download TXT datoteke sa mrežnog servisa .....	37
Dijagram 11 - pregled dokumenta putem web servisa .....	38
Dijagram 12 - kreiranje novog dokumenta .....	39
Dijagram 13 - spašavanje trenutnog dokumenta .....	40
Dijagram 14 - uređivanje sadržaja dokumenta .....	41
Dijagram 15 - klase Editing komponente .....	43
Dijagram 16 - dio strukture LMD dokumenta za dati primjer .....	45
Dijagram 17 - stablo aritmetičkog izraza za dati primjer .....	46
Dijagram 18 - tok algoritma za prepoznavanje uzoraka .....	47
Dijagram 19 - tok algoritma za <i>tokenizaciju</i> izraza .....	49
Dijagram 20 - organizacija MathEasy klasa .....	50

## Indeks tabela

Tabela 1- primjer zapisa u <i>markup</i> jeziku .....	11
Tabela 2 - operatori koje MathEasy podržava i prepoznaje .....	31
Tabela 3 - funkcionalnosti koje MathEasy omogućava kroz uređivač .....	31
Tabela 4 - tabela simbola i vrijednosti za dati primjer .....	46
Tabela 5 - neki primjeri matematičkih izraza, sintaksa, značenje i render .....	74
Tabela 6 - neki primjeri upotrebe komande za brzo kopiranje '#' .....	75
Tabela 7 - neki primjeri upotrebe komande za evaluaciju izraza '?' .....	75
Tabela 8 - rječnik pojmova .....	77

## Indeks slika

Slika 1 - korisničko sučelje LyX WYSIWYM uređivača za LaTeX.....	12
Slika 2 - ... i rezultat <i>renderinga</i> prethodno uređenog dokumenta .....	12
Slika 3 - sučelje <i>Simple Editor MathFlow</i> komponente (koristi MathML u pozadini) .....	13
Slika 4 - sučelje <i>Structure Editor MathFlow</i> komponente (koristi MathML u pozadini) .....	13
Slika 5 - sučelje MS Word, <i>Equation Tools</i> .....	15
Slika 6 - sučelje MS Word, primjer prikaza unesene formule .....	15
Slika 7 - prikaz HTML dokumenta uz upotrebu <i>MathJax rendering enginea</i> .....	53
Slika 8 - struktura MathEasy projekta.....	54
Slika 9 - <i>rendering</i> kôda datog u listingu br. 3.....	57
Slika 10 - primjer vizuelnog stabla WPF aplikacije i putanja događaja [33] .....	59
Slika 11 - sadržaj konfiguracijske datoteke.....	61
Slika 12 - transparentni prozor u WPF .....	64
Slika 13 - glavni radni prostor MathEasy aplikacije .....	67
Slika 14 - <i>renderirani</i> dokument .....	67
Slika 15 - rezultat uploada dokumenta (primjer).....	67
Slika 16 - pristup dokumentu putem web preglednika.....	68

## Indeks listinga

Listing 1 - primjer specifikacije jednostavnog prozora u XAML.....	55
Listing 2 - primjer naslijeđene WPF korisničke kontrole.....	56
Listing 3 - primjer strukture FlowDocument objekta.....	56
Listing 4 - primjer kôda za potiskivanje kreiranja novog paragrafa .....	62
Listing 5 - primjer kôda za potiskivanje aplikacijskih komandi .....	62
Listing 6 - primjer kôda za potiskivanje lijepljenja slike u dokument .....	63
Listing 7 - primjer kôda za preuzimanje sadržaja FlowDocument objekta kao stringa .....	63
Listing 8 - primjer transparentnog prozora u WPF .....	64
Listing 9 - kôd za ukidanje zvukova pri navigaciji u <i>webbrowser</i> kontroli .....	66
Listing 10 - kôd za podešavanje procesorskog afiniteta.....	66
Listing 11 - kôd koji pored ispravnosti testira i vrijeme odziva.....	69

## Sadržaj

Sažetak.....	3
Abstract.....	3
1. Uvod.....	4
1.1. Ciljevi rada.....	5
1.2. Struktura rada.....	5
2. Opis problema i postojeća rješenja.....	6
2.1. Opis problema.....	6
2.2. Specifikacija funkcionalnosti aplikacije.....	7
2.3. Postojeća rješenja.....	8
2.3.1. Upotreba markup jezika.....	9
2.3.2. Upotreba matematičkih objekata u bogatim formatima dokumenata.....	14
2.4. Zaključak.....	15
3. Dizajn rješenja.....	16
3.1. Korištena metodologija.....	16
3.2. Model aplikacije.....	18
3.2.1. Diskusija različitih modela.....	18
3.2.2. Model koji će se koristiti.....	27
3.3. Dizajn aplikacije.....	28
3.3.1. Slučajevi upotrebe.....	28
3.3.2. Specifikacija sintakse.....	31
3.3.3. Arhitektura i komponente aplikacije.....	32
3.3.4. Aktivnosti.....	36
3.3.5. Klase i algoritmi.....	42
3.3.6. Baza podataka.....	51
3.4. Zaključak.....	51
4. Implementacija i testiranje aplikacije.....	52
4.1. Razvojno okruženje i korištena tehnologija.....	52
4.2. Specifičnosti i razvoj WPF desktop aplikacije.....	54
4.2.1. Struktura projekta.....	54
4.2.2. XAML i WPF kontrole.....	55
4.2.3. Model upravljanja događajima u WPF.....	58
4.2.4. Model komandi u WPF.....	60

4.2.5.	Detalji implementacije .....	61
4.3.	Specifičnosti i razvoj serverskog dijela aplikacije .....	68
4.4.	Testiranje .....	68
4.5.	Zaključak .....	69
5.	Zaključak .....	70
	Pregled urađenog .....	70
	Budući razvoj .....	71
6.	Prilozi .....	72
6.1.	Upute za korištenje aplikacije .....	72
6.1.1.	Instalacija i konfiguracija .....	72
6.1.2.	Meni opcije .....	72
6.1.3.	Uređivanje dokumenta .....	73
6.2.	Rječnik pojmova i skraćenica .....	77
7.	Korištena literatura i vanjski resursi .....	78
	Indeks dijagrama .....	80
	Indeks tabela .....	80
	Indeks slika .....	81
	Indeks listinga .....	81