

Математички факултет
Универзитет у Београду

Развој графичког корисничког интерфејса за пројекат отвореног кода *QLab*

Мастер рад

Никола Миленковић

Новембар 2011

Ментор:

Др Мирослав Марић



Садржај

1. Увод	4
1.1. Шта је <i>QLab</i> ?	4
1.2. Циљеви <i>QLab</i> -а.....	4
1.3. Графички кориснички интерфејс.....	5
2. Захтеви	6
2.1 Пословни захтеви.....	6
2.2 Функционални захтеви	7
2.2.1 Креирање и учитавање пројеката.....	7
2.2.2 Креирање и брисање датотека	7
2.2.3 Конзола.....	8
2.2.4 Историја команди.....	8
2.2.5 Подршка <i>MATLAB</i> кода.....	9
2.2.6 Листа променљивих.....	9
2.2.7 Претраживач функција	11
2.2.8 Уређивање кода	11
2.2.9 Покретање кода	15
2.2.10 Отклањање грешака у коду.....	15
2.2.11 Приказ графика.....	16
2.2.12 Мени.....	17
2.2.13 Флексибилност интерфејса.....	17
2.2.14 Језик.....	18
2.2.15 Позивање помоћи за одабрани део кода	18
2.2.16 Помоћ/документација.....	20
3. Архитектура.....	21
3.1 Технологија	21
3.2 Обрасци за развој.....	22
3.2.1 Предности коришћења образаца.....	22
3.2.2 <i>Model-View-Controller</i>	22
3.2.3 <i>Model-View-Presenter</i>	24
3.3 Имплементација.....	26
3.3.1 <i>Prism</i>	26
3.3.2 Сервиси.....	27
3.3.3 Главна маска	28
3.3.4 <i>Ribbon</i>	29
3.3.5 Део за контролу датотека	29
3.3.6 Уређивање кода	31

3.3.7	Историја команди	33
3.3.8	Списак променљивих	33
3.3.9	Документа.....	34
4.	Финални изглед.....	35
4.1.1	Прибављање кода апликације.....	35
5.	Закључак и даљи развој	36
6.	Референце.....	37

1. Увод

QLab је апликација отвореног кода која служи за нумеричка израчунавања. Израђује се у најновијим технологијама компаније *Microsoft*, као што су *.NET Framework 4*, *C#* и *WPF*. Због свог обима и сложености, апликација је подељена на неколико великих целина. У овом излагању ће бити приказана архитектура и дизајн графичког корисничког интерфејса, који због специфичности апликације има јединствене захтеве.

Интерфејс је модуларан па је компонентама омогућено да се независно развијају без утицаја једне на друге. Свака компонента интерфејса имплементира *Model-View-Presenter* шаблон за развој. На тај начин се добијају компоненте које су концептуално подељене на три дела којима се тачно зна улога, што је за пројекат овакве сложености веома значајно. Архитектура и дизајн интерфејса *QLab*-а се такође може користити и у другим сложеним апликацијама.

Један од најпознатијих програмских пакета за нумеричка израчунавања јесте *MATLAB*.

MATLAB се такође користи и у индустријске сврхе, помаже при различитим симулацијама система и помоћу њега се лако добијају графички прикази резултата.

Једна од мана *MATLAB*-а је то што није бесплатан. Из тог разлога су настале бројне алтернативе отвореног кода као што су: *Octave*, *FreeMat* и *Scilab*. Те апликације циљају да у потпуности подрже *MATLAB* језик, тј. да се код извршава у њима без потребе за модификацијама.

Алтернативе такође имају своје мане, као што су нпр. непостојање корисничког интерфејса, компликовано коришћење, лоша документација, и слично.

Из тих разлога је настао пројекат под називом *QLab*.

1.1. Шта је *QLab*?

Укратко, *QLab* је бесплатна алтернатива комерцијалној апликацији *MATLAB*. У првој фази развоја је у плану потпуна компатибилност са *MATLAB*-ом, док ће у наредним фазама бити додате функционалности које не постоје у *MATLAB*-у. *QLab* је апликација отвореног кода, тј. свако ко жели може да скине изворни код и да га прилагоди својим потребама.

Развој овог пројекта проистекао је из сарадње Математичког факултета и *Microsoft*-а на иницијативу др Мирослава Марића.

1.2. Циљеви *QLab*-а

- **Потпуна подршка за *MATLAB* језик** – као и друге бесплатне алтернативе, и *QLab* има за циљ апсолутну подршку *MATLAB* језика. Све апликације које су писане за *MATLAB* би требало да се извршавају без модификација у *QLab*-у
- **Графички интерфејс лаган за коришћење** – интерфејс дизајниран тако да буде исти или бољи од тренутног решења у *MATLAB*-у.
- **Коришћење најновијих развојних технологија** – овим учесници у развоју стичу искуство које ће им бити од значаја у каријери.

- **Тимски рад** – један од главних разлога настанка *QLab*-а. Студенти на Математичком факултету ретко имају прилике да се опробају у тимском раду, а овај пројекат то успешно превазилази.
- **Места за све студенте** – с обзиром на математичку природу проблема, сви студенти са свих смерова могу учествовати у развоју, од програмирања, преко писања тест кода, па све до писања документације.

1.3. Графички кориснички интерфејс

QLab је пројекат из неколико целина које могу независно да се развијају:

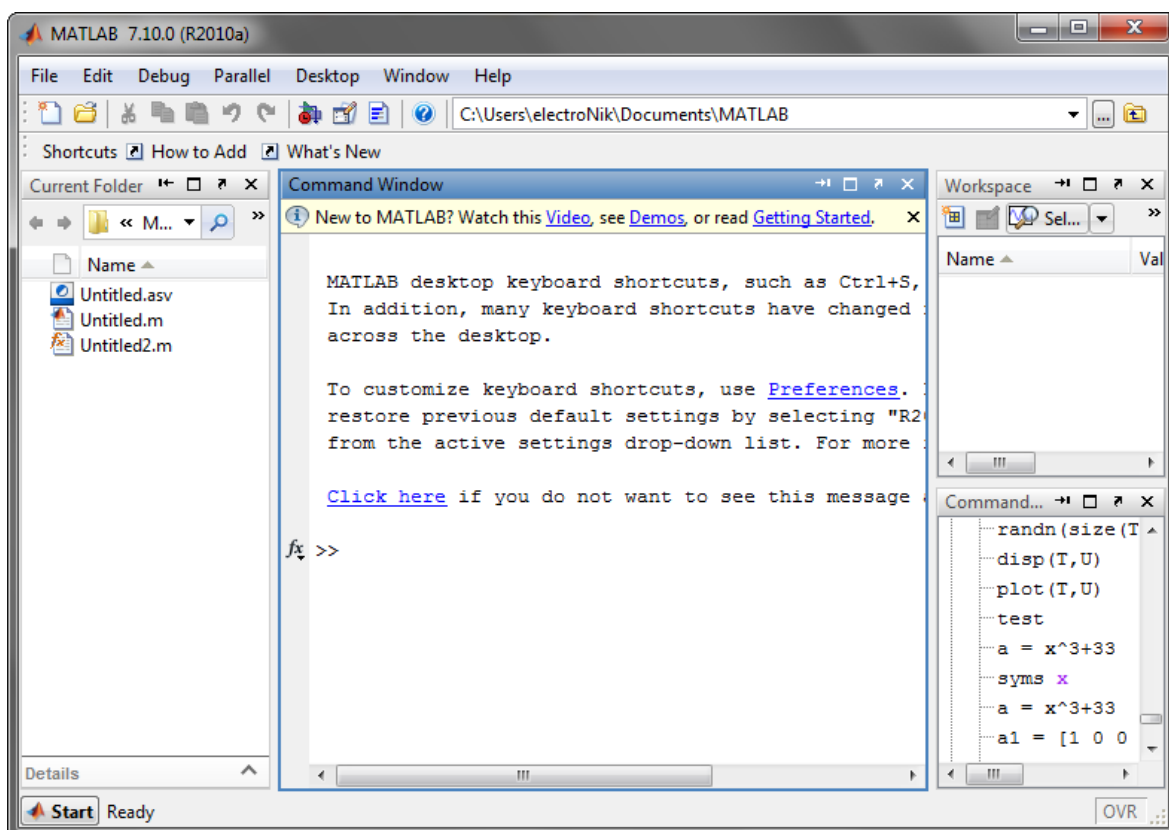
- *Engine* – врши нумеричка израчунавања
- *Parser* – преводи кориснички код у формат који је препознатљив *Engine*-у
- *GUI* – графички кориснички интерфејс
- *Help* – документација и упутство за коришћење

Тема овог рада јесте развој графичког корисничког интерфејса. У овом раду ће бити описана његова архитектура и имплементација, као и технологије које се користе у развоју.

2. Захтеви

За програмирање нумеричких алгоритама на Математичком факултету тренутно се користи *MATLAB* апликација која није бесплатна. Факултетима и другим лицима која желе да се баве нумериком, цена може да буде огромна препрека. Из тог разлога настаје потреба за пројектом као што је *QLab*.

Као што је већ напоменуто, *QLab* се састоји из више целина које могу да се развијају независно. С обзиром да је тема овог рада графички кориснички интерфејс, то ће бити предмет дискусије у наставку.



Слика 1: Изглед *MATLAB* апликације

2.1 Пословни захтеви

QLab апликација мора да испуњава следеће захтеве:

Лака прилагодивост корисника на *QLab* – Потребно је да постоји извесна сличност између *QLab* и *MATLAB* апликација како би се корисницима олакшало привикавање на ново окружење. Такође је потребно да све опције које постоје у интерфејсу *MATLAB*-а постоје и у *QLab*-у.

Примена најновијих трендова у дизајну корисничког интерфејса – Један од главних циљева *QLab*-а јесте примена најсавременијих програмерских технологија, а то такође важи и за кориснички интерфејс. Захтева се да интерфејс буде модеран, и да прати најновије трендове дизајна и функционалности корисничких интерфејса.

2.2 Функционални захтеви

У наставку рада детаљно су изложени функционални захтеви које би *QLab* апликација требало да задовољи.

2.2.1 Креирање и учитавање пројеката

Требало би да се рад у *QLab*-у заснива на пројектима. Када корисник жели да напише неки програм, покрене *QLab* и док не изабере опцију „креирај нови пројекат“ или „отвори постојећи пројекат“ неће моћи да пише програме.

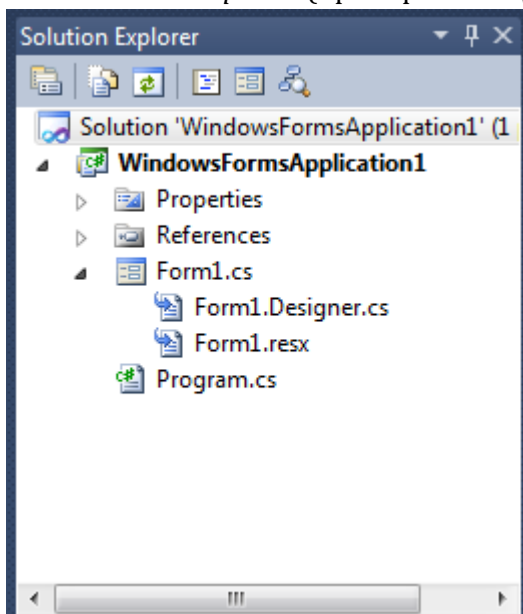
Да би корисник дефинисао пројекат мора да унесе име пројекта и директоријум у коме се тај пројекат налази. Дефинисање директоријума је неопходно да би *QLab* знао где да тражи датотеке укључене у пројекат. Датотеке не морају да буду у истом директоријуму где и пројекат, тј. могу да се налазе произвољно дубоко у неком од поддиректоријума.

При креирању пројекта, требало би направити манифест датотеку. То је датотека која садржи списак путања до свих датотека, укључених у пројекат.

Отварање пројекта подразумева читање манифест датотеке и на основу њеног садржаја приказ датотека које су укључене у пројекат у облику дрвета.

2.2.2 Креирање и брисање датотека

Као што је већ напоменуто, путање до свих датотека у неком пројекту треба да буду унете у манифест. Компонента апликације која буде приказивала структуру датотека пројекта ће се звати: *solution explorer* (пример на слици 2).



Слика 2: пример *solution explorer*-а

QLab ће вршити аутоматску детекцију датотека у пројекту и укључивати их у пројекат. Ова функционалност је корисна у случају када корисник жели да убаца датотеке у пројекат ван апликације.

Корисник ће имати опцију да креира нову датотеку из *QLab*-а. Када изабере ову опцију, од корисника ће се захтевати да унесе име и тип те нове датотеке. По креирању датотеке отвориће се прозор за уређивање кода.

За брисање ће се користити мени који се отвори када се кликне десним тастером миша на датотеку и кад корисник изабере опцију „обриши датотеку“. Такође ће бити подржана пречица преко тастатуре. Пре брисања датотеке обавезно треба да се прикаже упозорење о брисању.

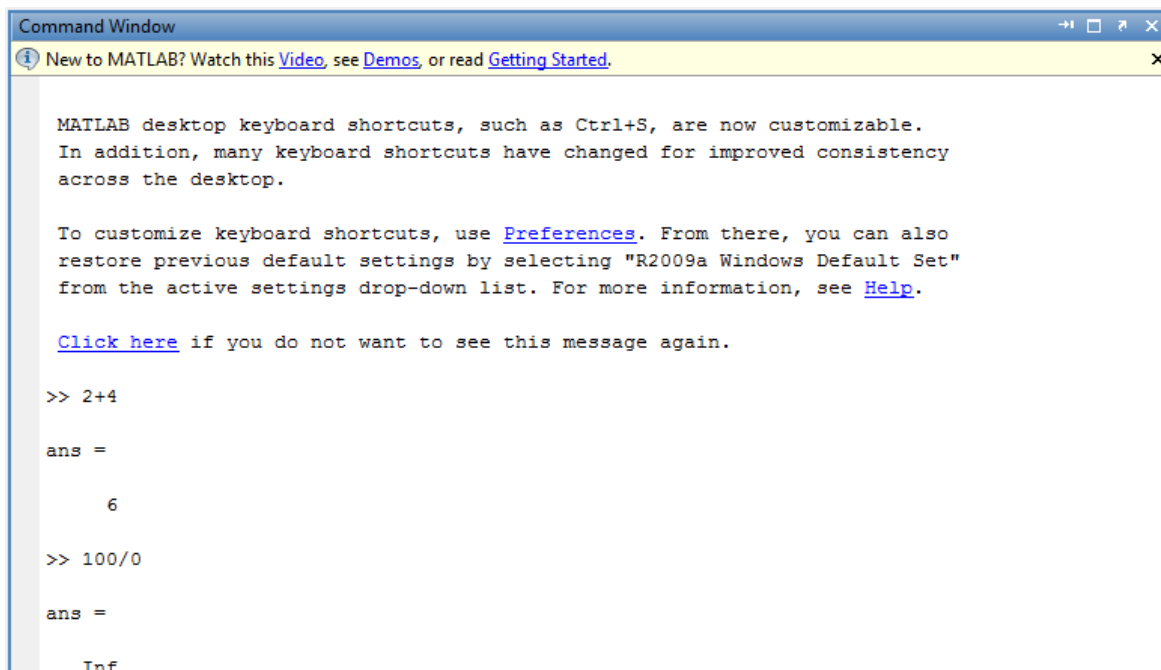
QLab ће подржавати копирање датотека као и могућност превлачења датотека из других апликација.

2.2.3 Конзола

Конзола ће бити једна од најчешће коришћених функционалности *QLab*-а. Преко конзоле ће се позивати уграђене функције *QLab*-а, као и функције које корисник сам напише.

Понашање конзоле треба да буде исто као и у *MATLAB* програмском пакету, а то подразумева следеће:

- Свака тренутно актуелна наредба треба да почиње са знаком за унос „>>“.
- Корисник може да мења само текст који је после знака за унос.
- Када корисник изврши команду, испише се резултат после којег се поново појави знак за унос.
- Конзола би требало да подржава различите боје текста. На пример, плава боја за линкове, црвена за испис грешака.
- Подразумевани фонт ће бити фонт једнаке ширине (*monospace*).
- Фонт, као и његова величина биће променљиви од стране корисника.
- Конзола ће подржавати стандардне могућности текстуалних поља као што су: копирање текста, превлачење текста, исецање текста и сл. При том је неопходно водити рачуна где је дозвољено мењање текста.
- Притиском на тастер *ENTER* тренутно унета команда се извршава и њен резултат испишује.
- Корисник може притиском на тастере горе (↑) и доле (↓) да пролази кроз историју унетих команди. Притиском на тастер „горе“ аутоматски се уписује последња унета команда. Поновним притиском на тастер „горе“ уписује претпоследња команда, итд., све док се не дође до прве команде. Слично за тастер „доле“.



```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customizable.
In addition, many keyboard shortcuts have changed for improved consistency
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a Windows Default Set"
from the active settings drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

>> 2+4

ans =

     6

>> 100/0

ans =

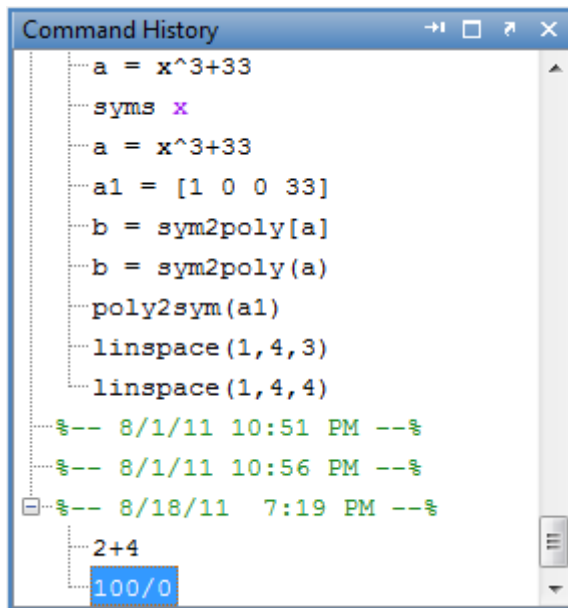
   Inf
```

Слика 3: Изглед конзоле у *MATLAB*-у

2.2.4 Историја команди

У одељку 2.2.3 је речено да је потребно запамтити сваку извршену команду ради побољшања употребљивости програма. Поред тога што ће кроз историју команди моћи

да се пролази стрелицама „горе“ и „доле“ на тастатури, моћи ће и преко посебног прозора који приказује све извршене команде. На слици 4 је приказано како изгледа прозор са историјом команди.



Слика 4: Историја команди

Поред тога што има функцију приказа извршених команди, прозор историје команди такође има и следеће функционалности:

- Извршене команде ће бити приказане у облику дрвета које ће груписати команде по датуму покретања апликације:
 - Унутрашњи чворови приказују датум покретања апликације.
 - Листови приказују извршену команду.
- Могућност одабира више команди у исто време.
- Одабране команде могу да се:
 - Копирају
 - Бришу
 - Изврше
 - Претоворе у .m датотеку
- Постоји опција за пражњење историје команди.

2.2.5 Подршка *MATLAB* кода

Да би *QLab* био примамљив за коришћење, неопходно је да постојећи *MATLAB* код може без измена да се извршава у *QLab*-у.

За увоз *MATLAB* кода у *QLab* пројекат биће довољно само прекопирати датотеке са кодом у *QLab* пројекат.

2.2.6 Листа променљивих

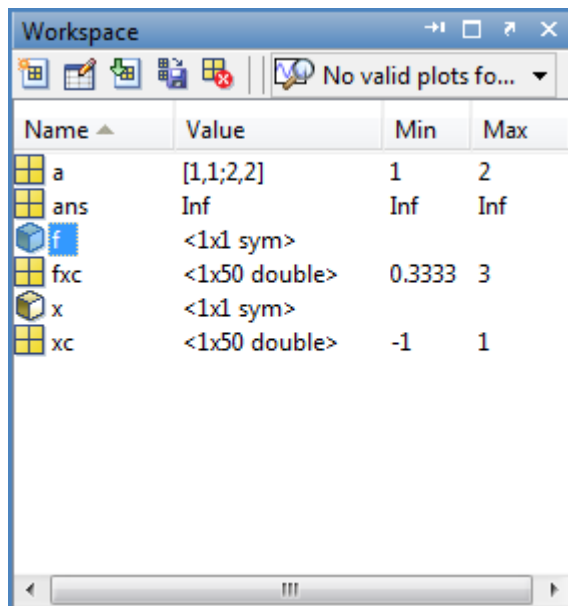
Све декларисане променљиве је неопходно видети на неки начин. За то ће да се постара прозор са списком свих декларисаних променљивих (слика 5).

Унос променљивих је могуће извршити на неколико начина. Први начин је декларацијом променљиве у конзоли. Други начин је извршењем кода који у себи садржи декларацију променљиве. Трећи начин јесте кликом на икону „креирај променљиву“ у оквиру прозора са списком променљивих.

Додатни начин за унос променљиве је преко опције за увоз података. Када корисник одабере ову опцију, приказаће му се прозор за одабир жељене датотеке са

подацима. Подржани типови су слике (векторске и растерске), текстуални подаци, табеле, звукови и видео.

Измена вредности променљивих се врши избором жељене променљиве и одабиром опције „измени вредност променљиве“. Ова акција отвара нови прозор са табелом у којој је неопходно унети нове или изменити постојеће вредности. Алтернативно, измена променљивих се врши кроз извршење кода, било у конзоли, било у датотеци са кодом.



Name	Value	Min	Max
a	[1,1;2,2]	1	2
ans	Inf	Inf	Inf
f	<1x1 sym>		
fxc	<1x50 double>	0.3333	3
x	<1x1 sym>		
xc	<1x50 double>	-1	1

Слика 5: Списак променљивих

Списак променљивих омогућује и брисање декларисаних променљивих. То се врши на тај начин што се одабере променљива (или више њих) и изабере опција „обриши променљиву/променљиве“. После ове акције обрисане променљиве више нису доступне за коришћење и изазивају грешку уколико се покуша читање њихових вредности.

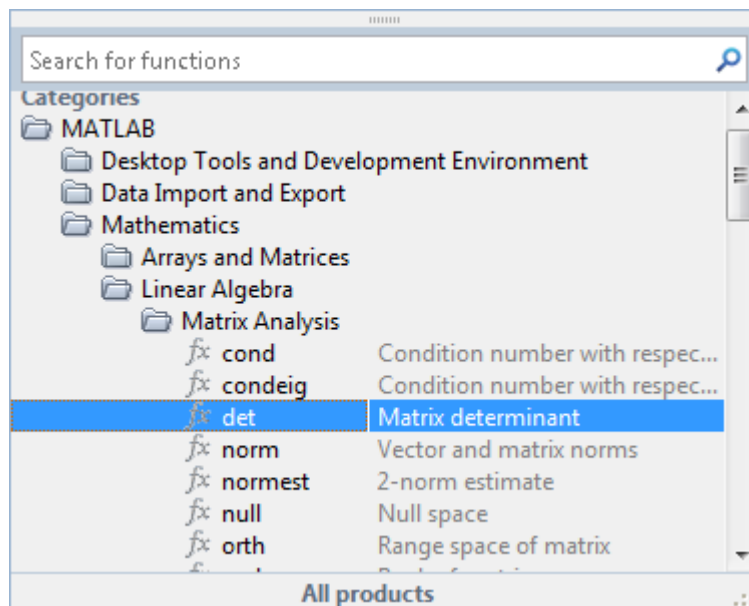
Прозор са списком променљивих ће имати и опцију за извоз вредности променљиве у датотеку. При одабиру ове опције кориснику се прикаже прозор за чување датотека, и када корисник одабере име датотеке и њену локацију, подаци ће се сачувати у наведеној датотеци.

За приказ променљивих, листа треба да подржи следеће колоне, које ће корисник моћи да приказује и скрива по свом нахођењу:

- Назив променљиве
- Вредност уколико је могуће приказати. Ако није, приказати величину променљиве
- Минимум
- Максимум
- Величину променљиве
- Колико бајтова заузима
- Класа (*double*, *symbolic*, итд.)
- Опсег променљиве (нпр. ако је у матрицу смештено: 1, 2, 3, 4, опсег је 3)
- Аритметичка средина
- Медијана
- Модус
- Варијанса
- Стандардна девијација

2.2.7 Претраживач функција

Када корисник жели брз преглед свих уграђених функција у *QLab*-у, користиће претраживач функција. Изглед тог прозора ће бити исти као што је приказано на слици 6.



Слика 6: Претраживач функција

Функције би требало да буду приказане у облику дрвета, где сваки чвор представља једну грану математике. Листови дрвета су конкретни називи функција које ће бити уграђене у *QLab*. Поред имена функције, претраживач функција ће обезбедити и опис функције.

Ради лакшег и ефикаснијег коришћења, претраживач функција ће имати поље за претрагу у које корисник може унети жељени упит.

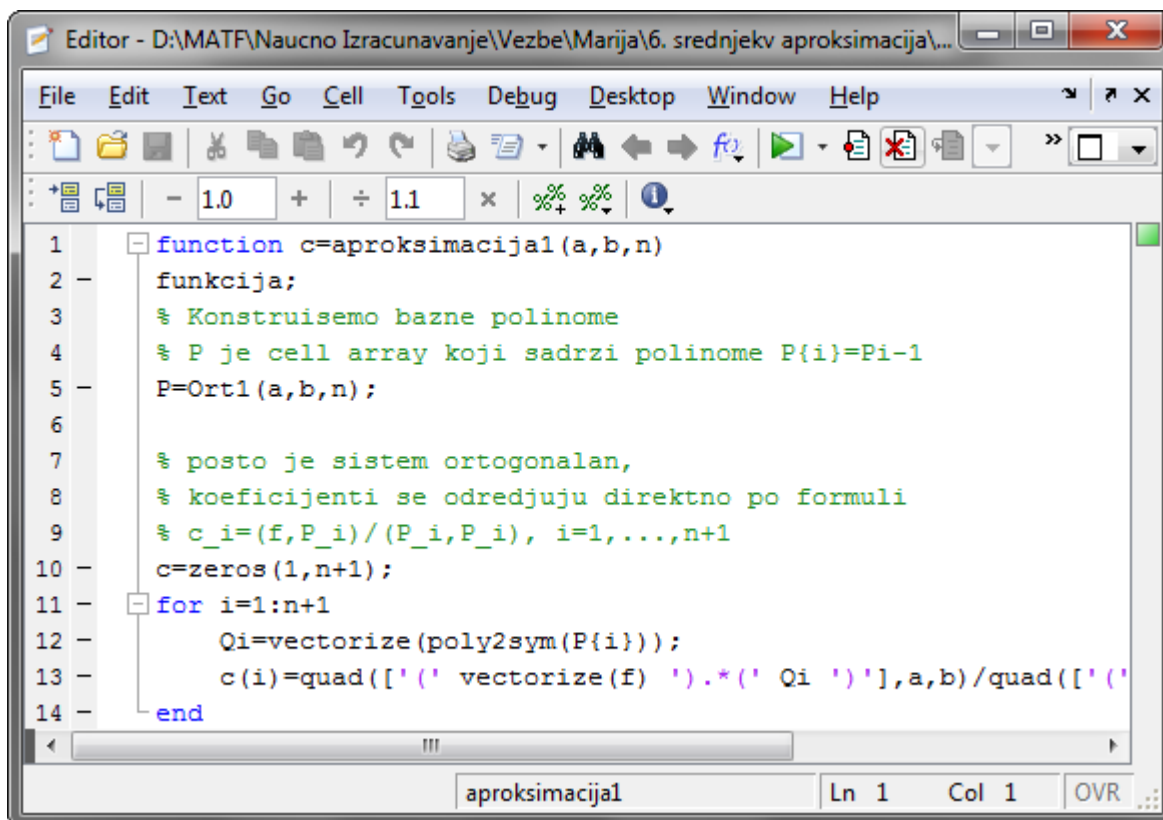
Када корисник унесе упит, и када притисне тастер ENTER, упит се изврши. Критеријуми по којима је потребно имплементирати претрагу су: назив категорије (чвор дрвета), назив функције (лист дрвета) и опис функције. Ако се тражени појам налази било где у ове три ставке, ставка ће бити враћена као резултат.

Резултат ће бити груписан у два дела: категорија и функција. Део са категоријама ће приказати све категорије које су враћене као резултат претраге. Део са функцијама ће излистати све функције које у називу или опису имају тражен појам.

Да би корисник убацио неку функцију у свој код на место где се налази курсор, довољно је да кликне два пута на жељену функцију.

2.2.8 Уређивање кода

С обзиром да ће *QLab* подржавати *MATLAB* програмски језик, неопходно је обезбедити алат унутар *QLab*-а који ће помагати програмерима да развијају програме.

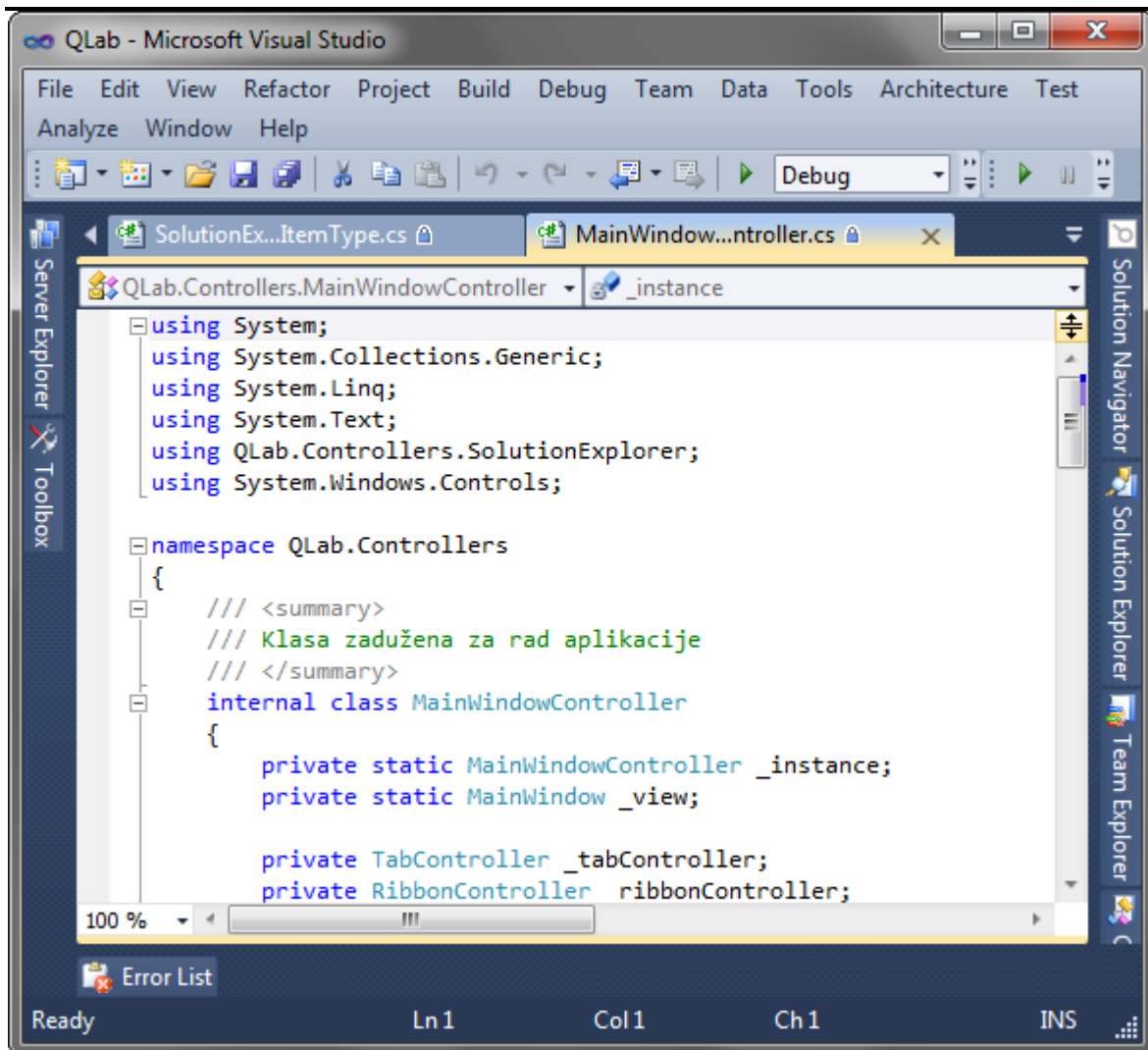


Слика 7: Прозор за уређивање кода у MATLAB-у

На слици 7 види се имплементација прозора за уређивање кода у програмском пакету *MATLAB*. Када се одабере опција „отвори датотеку“, добије се прозор као на слици. Овај прозор је релативно независан од остатка *MATLAB*-а. Сви неопходни алати везани за код налазе се у овом прозору.

За разлику од *MATLAB*-а, *Visual Studio 2010* има другачији приступ (слика 8). Све је у једном главном прозору. Све функционалности *Visual Studio*-а су смештене у језичке (енг. *tab*), а ту је и између осталог и прозор за уређивање кода. Овим је постигнута униформност апликације.

С обзиром да ће *QLab* интерфејс имати елементе *Visual Studio*-а, уређивање кода ће доста личити на *Visual Studio*. А пошто се ради о уређивању *MATLAB* кода, интерфејс ће имати све функционалности које има и *MATLAB* програмски пакет.

Слика 8: Уређивање кода у програмском пакету *Visual Studio 2010*

Функционалности које ће имати *QLab* везане за уређивање кода су следеће:

Преламање текста – Када дужина линија кода пређе тренутну ширину прозора за уређивање кода, текст ће се преломити у нову линију. Овим се обезбеђује да се у сваком тренутку види цео код, без потребе корисника да шета прозор лево-десно. Ова функционалност ће моћи да се подешава од стране корисника, тј. корисник ће моћи да подеси да се текст приказује непреломљен уколико то жели.

Број линије – Лево од сваке линије кода је потребно да се испишује редни број те линије. То ће кориснику рећи две ствари: колико тачно има линија кода у тренутној датотеци, и код откривања грешака, на којој линији се десила грешка. Уколико је линија преломљена, испред дела где је настао прелом не би требало да пише број линије. Уколико корисник не жели да види бројеве линија, моћи ће да их искључи у подешавањима апликације.

Бојење текста – Текст унутар прозора за уређивање кода треба аутоматски да се боји подешеним бојама (као на слици 7 и на слици 8) - кључне речи једном бојом, коментари другом, итд. Те боје ће корисник моћи да мења.

Локација курсора – У статусној линији би требало да се испишује тренутна локација курсора, у виду броја реда и броја колоне¹. Како корисник помера курсор, тако се и вредности ова два поља у статусној линији ажурирају. Бројење почиње од 1.

Начин уноса (уметање или преписивање) – Постоје два начина уноса текста: уметање које се најчешће користи у свим уређивачима текста, и преписивање

¹ Колона у овом контексту је редни број карактера од почетка реда

које када корисник уноси текст, мења текст десно од курсора. У статусној линији ће бити приказана индикација који је тренутни мод уноса, тј. *INS* за уметање и *OVR* за преписивање.

Скривање делова кода – Испред почетка било ког блока кода (*if*, *for*, *function*) ће стајати индикатор да тај цео блок може да се сакрије. На слици 7 се то види испред кључних речи *function* и *for*. Када корисник кликне на тај индикатор (знак минус) цео блок кода се сведе у једну линију. У тој једној линији ће бити исписан текст који је и био раније, с тим што ће се на крај додати три тачке. Испред те линије ће се појавити индикатор да линија може да се прошири (знак плус) и кад корисник кликне на индикатор, прикажу се све линије кода које су биле скривене. Ова функционалност ће памтити стања угњеждених скривених/откривених делова кода.

Фонт – Подразумевани фонт је monospace². Корисник ће у подешавањима програма моћи да промени фонт, као и величину фонта. Такође, моћи ће да промени дебљину фонта и моћи ће да постави искошена или подвучена слова. Треба напоменути да ће, између осталог, и подразумевана боја фонта моћи да се промени.

Одабир текста – корисник ће моћи да одабира текст било коришћењем миша било коришћењем тастатуре. Изабрани текст ће подразумевано бити плаве боје, али ће боја маркера који означава изабрани текст зависити од боје фонта, као и од боје позадине. Ако корисник жели да поништи избор, довољно је да кликне било где унутар прозора за уређивање кода. Иако постоје програми (1) (2) у којима је могуће одабрати два или више несуседна дела текста, то овде неће бити случај.

Копирање, сечење, налепљивање текста – Ове опције се подразумевају, али није на одмет напоменути их. Копирање смешта одабрани текст у меморију (*clipboard*). Сечење текста ради исто што и копирање с том разликом што сечење додатно обрише одабрани текст. Налепљивање умеће текст из меморије на одговарајуће место (тамо где је курсор). При налепљивању треба водити рачуна о моду уноса текста.

Аутоматско увлачење текста – Одређени блокови кода аутоматски се увлаче за један таб. Ти блокови су: *if*, *for*, *while*. Када корисник унесе неки ред који почиње са наведеним кључним речима и када притисне тастер *ENTER*, тада се нови ред аутоматски увлачи за један таб више од претходног реда. Ова функционалност помаже при читљивости кода. Корисник ће моћи да искључује и укључује ову функционалност по жељи.

Дужина таба – Корисник ће моћи у подешавањима апликације да промени дужину таба карактера. То значи да свако појављивање таба заузима подешену дужину карактера. Подразумевана вредност је 4 карактера.

Предлагање завршетка речи – Када корисник започне унос кључне речи или назива променљиве, приказаће се прозор који има понуђене све променљиве и кључне речи. Када корисник изабере жељену ставку и притисне тастер *ENTER*, одабрана реч се аутоматски унесе/доврши.

Подршка за више отворених датотека у исто време – *QLab* ће подржавати вишеструко отварање датотека у исто време. Пример ове функционалности се види на слици 8, где су отворене две датотеке у исто време. Свака датотека има свој засебан језичак у коме се налази прозор за уређивање текста.

² Фонт код кога су ширине карактера идентичне

Постављање тачака прекида³ (енг. *breakpoints*) – С обзиром да ће уређивач кода бити уско везан за део апликације који се бави отклањањем грешака (одељак 2.2.10), потребно је да подржи неке његове функционалности, као што су тачке прекида. Кад корисник постави тачку прекида, десно од броја линије ће се појавити црвена тачка која означава да ће се на тој линији догодити прекид у извршавању апликације. Корисник може да постави/склони тачку прекида тако што ће кликнути на празан део између броја линије и самог текста. Други начин је кроз меније.

Извршавање одабраног дела кода – Код који корисник одабере може да се изврши као да је прекопиран у конзолу и тамо извршен. Ова функционалност то ради у једном клику.

Отварање одабраног дела кода – Уколико корисник одабере функцију за коју постоји изворни код у некој другој датотеци, овом командом се та датотека отвара као нови језичак у *QLab*-у.

Рад са коментарима – Потребно је да постоји опција за коментарисање одабраног дела кода, као и опција за уклањање коментара. Ако је курсор унутар коментара, корисник може једним кликом да прошири избор на цео коментар.

Контекстуални мени – У контекстуалном менију постоји списак свих најчешће коришћених функционалности прозора за уређивање кода. Те функционалности су следеће:

- Изврши одабрани део кода
- Отвори одабрани део кода
- Помоћ за одабрани део кода
- Претраживач функција
- Потпис одабране функције
- Исеци
- Копирај
- Налепи
- Одабери цео коментар
- Искоментариши одабрани део кода
- Уклони коментар са одабраног дела кода
- Постави/уклони тачку прекида
- Постави/уклони условну тачку прекида
- Омогући/онемогући тренутну тачку прекида

2.2.9 Покретање кода

Кад корисник напише функцију у облику `.m` датотеке, треба на неки начин и да је изврши. То ће урадити тако што ће из конзоле позвати жељену функцију са одговарајућим аргументима, слично као кад позива уграђене функције *QLab*-а.

2.2.10 Отклањање грешака у коду

У процесу развијања алгоритама није редак случај да се догоди грешка. Грешке које нису синтаксне није лако уочити у току програмирања. *QLab* ће проверавати синтаксне грешке у току куцања кода, и све неправилности одмах приказати. Грешка ће бити означена подвлачењем црвене линије испод дела кода који није исправан. Када корисник пређе мишем преко подвученог дела кода, приказаће се нотификација са описом грешке и, ако је могуће, са описом начина исправке грешке.

³ Тачка прекида означава место где ће се извршавање програма зауставити. Ово помаже код уклањања грешака јер је у тренутку прекида могуће видети стања свих променљивих које су учитане у меморију.

Код семантичких грешака ситуација је другачија. Корисник уноси синтаксно исправан код, али при извршавању тог кода се добија неочекивани резултат. Без помоћних алата корисник би морао да исправља грешке тако што би у сваком кораку алгоритма морао да позива функцију за испис да би видео шта се тачно дешава у току извршења апликације.

Код оваквих проблема алати за отклањање грешака могу много да помогну кориснику, и да увелико смање време неопходно за проналазак и отклањање грешака.

Ови алати су зависни од прозора за уређивање кода (одељак 2.2.8). Тачније речено, алати за отклањање грешака у коду не могу да раде без прозора за уређивање кода и сви алати раде у режиму прекида, тј. када је паузирано извршавање програма.

Постоје две врсте прекида:

- 1) Безусловни прекид – Кад се наиђе на ову врсту прекида приликом извршавања програма, извршавање се зауставља (осим уколико тачка прекида није онемогућена).
- 2) Условни прекид – Извршавање програма се зауставља ако је задовољен задати услов. Примери услова су следећи: $x > 20$; $x > y$; $p = 'ABC'$

У алате за отклањање грешака спадају:

Индикатор извршавања – Приказује линију кода која се следећа извршава.

Посматрач (енг. *watch*) – Ово је прозор који за задати израз приказује резултат. На пример, ако корисник унесе име променљиве као израз који треба да се прати, у сваком кораку извршења кода ће се приказивати вредност посматране променљиве.

Следећи корак – Овом акцијом се изврши следећа линија кода, и извршавање се поново паузира.

Уђи у – Овом акцијом се улази у дефиницију функције уколико њен код постоји. Ова акција отвара датотеку са имплементацијом функције и приказује је као тренутно одабрани језичак. Индикатор извршавања се постави на прву линију кода унутар функције.

Изађи – Ако је индикатор извршења унутар неке функције, овом акцијом излазимо из тренутне функције, и индикатор извршења постављамо на линију која позива претходну функцију.

Настави – Наставља извршавање кода.

2.2.11 Приказ графика

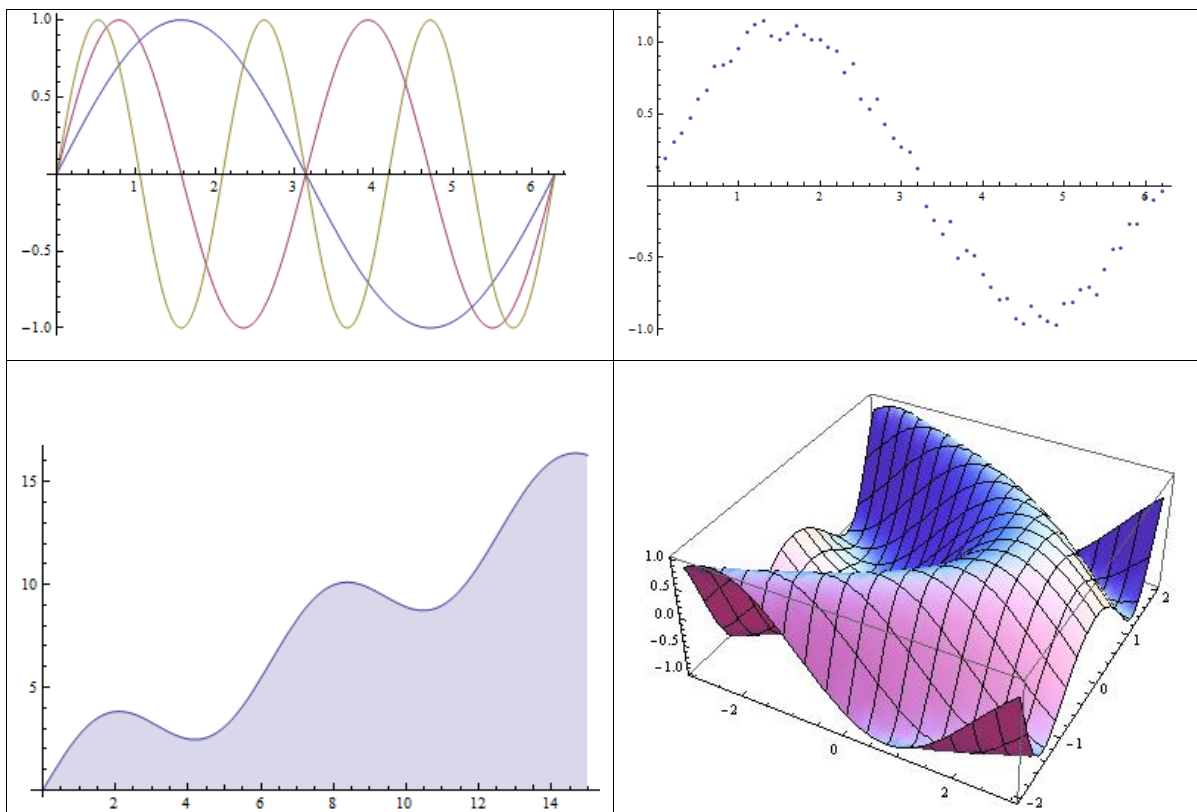
QLab ће имати могућност приказивања резултата у виду графика. Ово је врло важна функционалност с обзиром да визуелни приказ података може дати другачију перспективу на резултате. На слици 9 приказани су примери како графички прикази треба да изгледају.

Сваки графикон се приказује у засебном језичку. То се догађа када корисник позове неку од функција за исцртавање, независно од тога да ли је позвана из конзоле или извршењем корисничког кода.

Функционалности које је потребно имплементирати су:

Извоз слика – Сваки графикон је могуће извести у неки од стандардних формата као што су: *PNG, JPG, GIF, BMP*. Када корисник изабере опцију за извоз, појавиће се дијалог за чување датотека где се бира локација за чување, као и формат слике.

Промена величине слике – Корисник ће кликом на слику добити опцију за промену величине слике. Димензије слике се мењају повлачењем ивица или ћошкова слике.

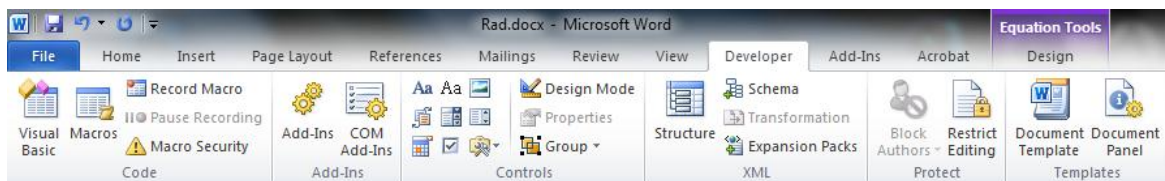


Слика 9: Пример приказа графика

2.2.12 Мени

Мени обезбеђује приступ свим функционалностима програма. У зависности од тренутно активног језичка, неке функционалности ће бити скривене, а неке приказане.

С обзиром да *QLab* прати актуелне трендове у области програмирања, мени неће бити „класичан“, већ тзв. *ribbon*. На слици 10 је приказан пример апликације са *ribbon*-ом.

Слика 10: *Ribbon* у програмском пакету *Microsoft Word 2010*

За разлику од класичних менија, функционалности ће бити распоређене по групама. Групе садрже сродне функционалности. Изузетак је „*File*“ група, која је у ствари класичан мени и садржи функционалности као што су: „Нова датотека“, „Сачувај датотеку“, „Подешавања апликације“, и сл.

Групе које је потребно имплементирати у мени *QLab*-а су следеће:

- *File*
- Основно
- Развој
- Прозори

2.2.13 Флексибилност интерфејса

Неопходно је обезбедити могућност подешавања изгледа апликације онако како кориснику највише одговара. Због тога је неопходно да интерфејс буде флексибилан у следећем смислу:

Величина прозора – Сваком прозору унутар апликације је могуће променити величину и по хоризонтали и по вертикали. Промена величине једног прозора може да изазове промену величине другог прозора.

Скривање прозора – Сваки прозор је могуће по жељи искључити или укључити. Сваки прозор који је укључен је могуће привремено сакрити као што се види на слици 8. Скривени прозор се поново прикаже када се мишем пређе преко његове иконице. Да се прозор не би аутоматски скривао, потребно је „прикачити га“ опцијом која ће се налазити на врху сваког прозора. Угашене прозоре је могуће поново упалити из менија.

Премештање прозора – Сваки прозор је могуће преместити на жељену локацију, онако како то кориснику највише одговара. За премештање прозора је довољно мишем ухватити насловну линију прозора, и вући на жељено место. Док корисник вуче прозор, апликација ће му нудити места где прозор може да се смести.

Прозори ван апликације – Неопходно је да прозори одају утисак да су интегрални део апликације. То значи да ће сви прозори бити унутар *QLab* главног прозора, као што је приказано на слици 1. Ако корисник жели да одвоји неки прозор од главне апликације, имаће могућност да то уради. На пример, ако жели да му прозор за уређивање кода буде одвојен, моћи ће да превуче ван главног прозора.

2.2.14 Језик

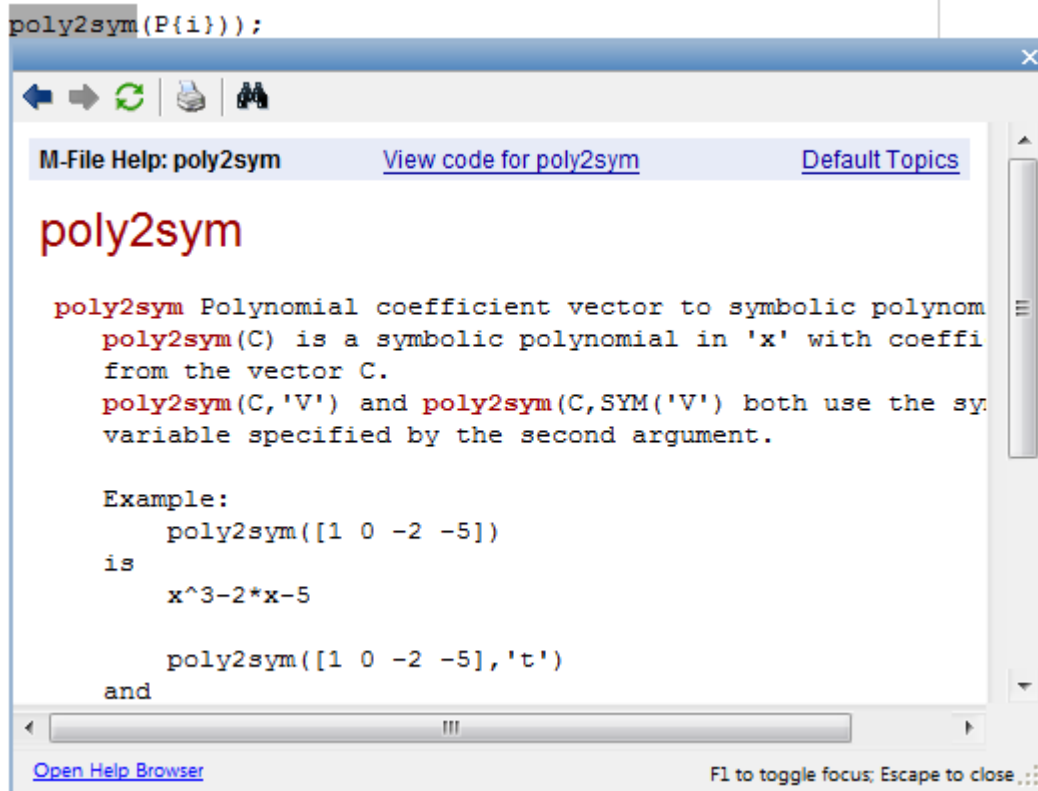
Претпоставља се да ће *QLab* апликација бити коришћена у више различитих земаља, и као таква неопходно је да подржава више језика. Примарни језик је енглески па потом српски и на крају сви остали. Редослед језика је битан при расподели времена за развој апликације.

Апликација ће имплементирати механизме за промену језика у току рада, тј. када корисник изабере други језик, да му се у том тренутку сви текстови у апликацији пребаце на тај нови језик. Под текстове спадају називи свих опција у менију, облачићи са описом дугмића и функција, текстови порука, као и помоћ у апликацији.

Језик ће бити могуће променити из подешавања апликације и није неопходно да постоји функционалност интерфејса преко које ће бити могуће брзо мењати језике.

2.2.15 Позивање помоћи за одабрани део кода

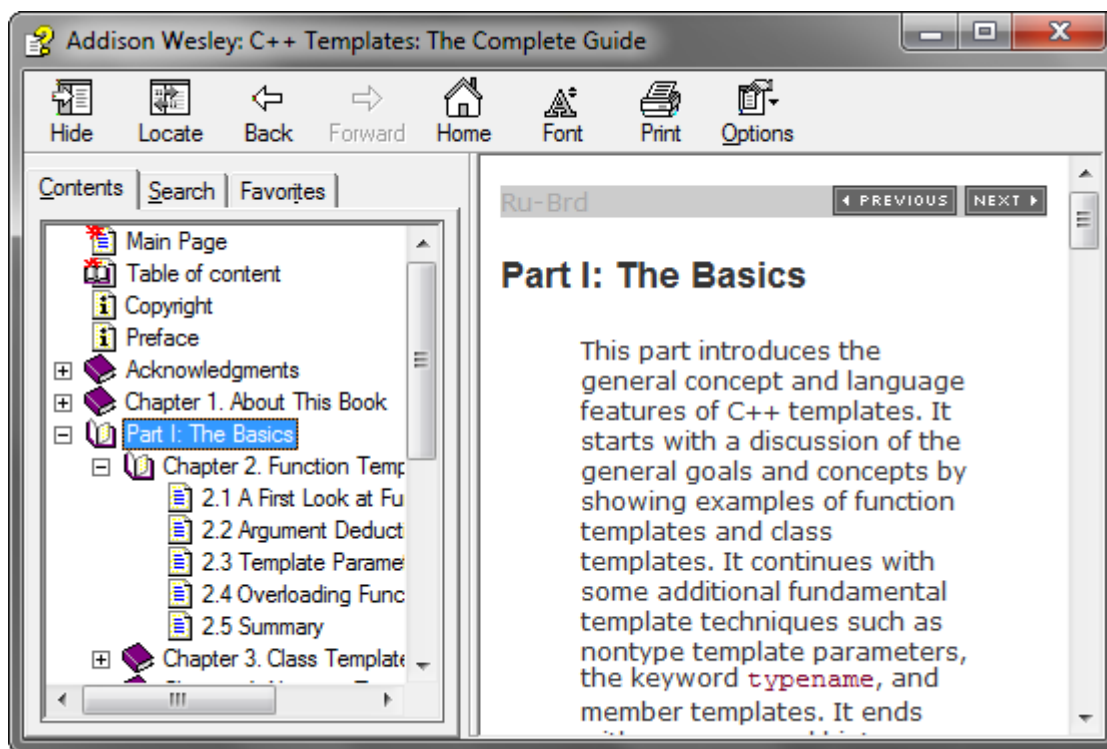
Када корисник жели да види документацију за одређену уграђену функцију у *QLab*-у, потребно је да одабере жељену функцију у коду и да позове помоћ, нпр. преко тастатуре тастером *F1*. Пример те функционалности се види на слици 11, где је корисник одабрао функцију *poly2sum* у коду и позвао помоћ.

Слика 11: MATLAB помоћ за функцију *poly2sym*

Прозор помоћи за функције има следеће карактеристике:

- Текст помоћи** – Текст написан од стране тима који се бави исписом помоћи и документације за *QLab*. Потребно је да текст буде у стандардизованој форми.
- Навигација** – Текстови ће садржати линкове ка другим текстовима и кликом на неки од линкова приказаће се нови текст. Ово је понашање које је стандардно за претраживаче интернета. Када корисник оде на нови текст, приказаће се икона „назад“ помоћу које може да се врати на претходни текст. Слично важи и за икону „напред“.
- Штампа** – Корисник ће моћи да одштампа текст који се види у прозору за помоћ кликом на одговарајућу икону. Пре саме штампе појавиће се дијалог за подешавање штампе где корисник може да одабере број копија, штампач на коме ће се штампати, оријентацију папира, и сл.
- Претрага** – корисник ће моћи позивањем одговарајуће функционалности да претражи текст помоћи. С обзиром да текст може да буде дугачак, претрага може кориснику да уштеди доста времена.
- Приказ изворног кода функције** – Свака функција ће имати свој изворни код, и уколико корисник жели да види како је функција имплементирана, искористиће ову функционалност прозора за помоћ.
- Одабир и копирање текста** – Корисник ће моћи да изабере произвољни део текста помоћи и да га копира у радну меморију.
- Линк ка индексу** – Уколико корисник жели да види све функције које *QLab* нуди, моћи ће да оде на индекс страну где су пописане све функције *QLab*-а. На сваку функцију је могуће кликнути и приказати њен текст помоћи.
- Величина прозора** – Као и сви остали прозори у *QLab*-у, и прозор за помоћ ће моћи да се повећава/смањује у зависности од жеље корисника. За ову акцију корисник ће мишем доћи до ивице коју жели да повећа, кликнути и вући до жељене величине.

2.2.16 Помоћ/документација



Слика 12: Прозор за помоћ у апликацији

Помоћ је важан део апликације јер омогућава корисницима да сазнају тачно сваку функционалност коју апликација пружа. Због тога је важно да апликација има део за помоћ који ће лако, брзо и ефикасно да се користи. Главни циљ је да корисник, уколико има проблема са радом у апликацији, може за што краће време да нађе решење за свој проблем.

Функционалности помоћи су следеће:

Садржај – Прозор за претрагу поседује садржај преко кога ће корисник моћи логички да пронађе тему која га интересује. Садржај је приказан у облику дрвета, слично као леви део слике 12. Кликом на неку тему приказаће се текст.

Текст – Када корисник пронађе тему која га занима и изабере је, у десном делу прозора приказаће се текст помоћи. Прозор подржава текст са напредним форматирањем, које подразумева: произвољну величину, боју и фонт суседног текста, наслове, слике, линкове, итд. Тачније, потребно је да текст буде *HTML* компатибилан.

Претрага – Када корисник зна за одређени појам, а не зна где се налази, искористиће претрагу да нађе текст са жељеним појмом. При претрази корисник уноси жељени појам у поље за унос текста. Кад изврши претрагу, добиће списак свих тема које садрже жељени појам, било у наслову, било у тексту.

Навигација – При преласку са једне теме на другу памте се све теме које су отворене да би корисник могао да иде напред/назад кроз теме које је посетио, као у претраживачу интернета.

Штампа – Корисник има могућност да одштампа жељену тему, јер то може неким корисницима значајно да олакша читање.

Промена величине текста – Прозор за помоћ има уграђене функционалности за повећање фонта. То може олакшати читање особама са слабијим видом.

Омиљене теме – Кориснику је омогућен бржи приступ темама које најчешће чита. Томе служе омиљене теме. Корисник може било коју тему да означи као омиљену, и она ће се трајно наћи у овом списку. Такође, може и теме које се налазе у овом списку да избаци кад год пожели.

3. Архитектура

У овој секцији рада биће речи о архитектури, дизајну и имплементацији графичког корисничког интерфејса на основу наведених захтева.

3.1 Технологија

При изради *QLab*-а су коришћене најновије технологије. Једна технологија која се активно развија, која је бесплатна за коришћење, и која има квалитетне алате за развој јесте *.NET Framework*. Тренутно актуелна верзија је 4, и та верзија се користи за развој свих делова *QLab*-а, а између осталог и за развој графичког интерфејса.

Оно што актуелна верзија *.NET Framework*-а може да понуди као огромну предност за развој графичког интерфејса јесте *Windows Presentation Foundation*. То је графички подсистем за исцртавање корисничких интерфејса за апликације које су писане за *Windows* оперативни систем. За разлику од претходне технологије која се базирала на *GDI* подсистему, *Windows Presentation Foundation* користи *DirectX*⁴. Мотивација за настанак *WPF*-а је жеља да се направи конзистентан програмерски модел за развој корисничких интерфејса као и раздвајање корисничког интерфејса од пословне логике.

Предности које *Windows Presentation Foundation* пружа су:

Извршавање на графичкој картици – Интерфејси писани у *WPF* технологији се извршавају на графичкој картици. Тиме се добија на брзији апликације јер процесору остаје више времена за извршавање пословне логике апликације.

Везивање података – *WPF* има уграђени скуп функционалности за рад са подацима који омогућава програмерима да „везују“ и манипулишу подацима унутар апликације. Ово представља важан концепт који одваја дизајнирање и развој корисничког интерфејса од развоја пословне логике.

Векторска графика – Компоненте у *WPF*-у су засноване на векторима, и самим тим квалитет графике се не губи при различитим нивоима увећања слике. Квалитет слике такође се не мења ни при различитим резолуцијама екрана.

Multimedia – *WPF* има уграђену подршку за уобичајене мултимедијалне садржаје као што су векторске или растерске слике, аудио и видео. Такође подржава *2D* и *3D* графику и анимације, што значи да је могуће имати тродимензионалне графичке корисничке интерфејсе.

⁴ Мајкрософтов *DirectX* (*Microsoft DirectX*) је скуп АПИ-ја, чија је намена манипулисање хардвером. Посебну примену налази у програмима везаним за рачунарске игре на Мајкрософтовим платформама.

3.2 Обрасци за развој

Образац за развој је опште, поново употребљиво решење за честе проблеме који се срећу приликом пројектовања софтвера. За већину архитектуралних програмерских проблема већ постоје решења која се сматрају оптималним и њих зовемо обрасцима.

Да би *QLab* био оптимално дизајниран, коришћени су обрасци за развој. У свакој од целина апликације је коришћен бар један образац.

Постоји велики број образаца за развој, и за сваки се зна шта му је намена и где се користи. Постоје обрасци за базе података, структуре података, комуникацију између компоненти, за интерфејс, итд.

3.2.1 Предности коришћења образаца

Главна предност се огледа у томе што развој апликације постаје олакшан. Могуће је јасно дефинисати делове апликације, и тачно се зна који део је задужен за коју функционалност.

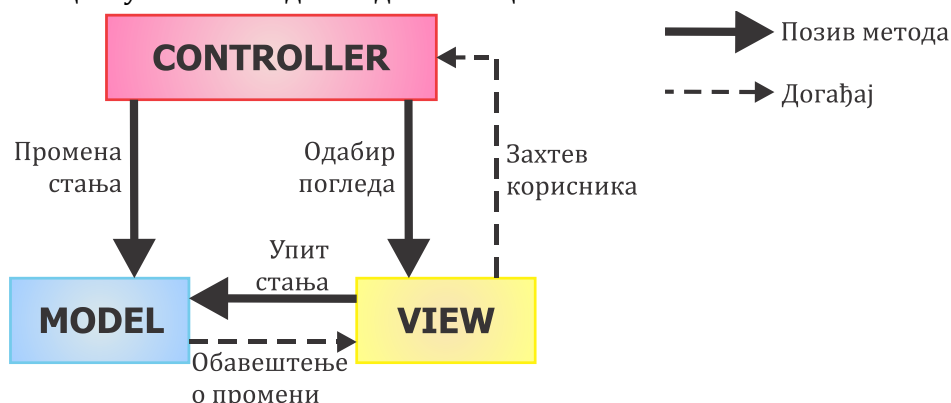
Обрасци за развој апликација представљају стандард. С обзиром да велики део програмера зна обрасце, довођење нових програмера у тим неће одузети много времена за њихово навикавање на архитектуру апликације.

3.2.2 Model-View-Controller

За имплементацију интерфејса постоји више добрих решења. Развојем нових технологија за развој интерфејса (*HTML 5*, *Silverlight*, *WPF*, и др.) развијају се и нови обрасци за развој.

С обзиром да су апликације које се праве већином пословне, и да се углавном састоје из више делова, развијени су обрасци који подржавају овакву архитектуру. Под архитектуром послове апликације се мисли на архитектуру која се састоји из података, дела апликације која обрађује те податке и дела апликације који приказује податке и даје кориснику контролу над истим.

Најпознатији је *MVC* (*Model-View-Controller*) који апликацију раздваја на три дела, а то су: *Model* (подаци), *View* (приказ или интерфејс) и *Controller* (пословна логика). Концептуални изглед се види на слици 13.

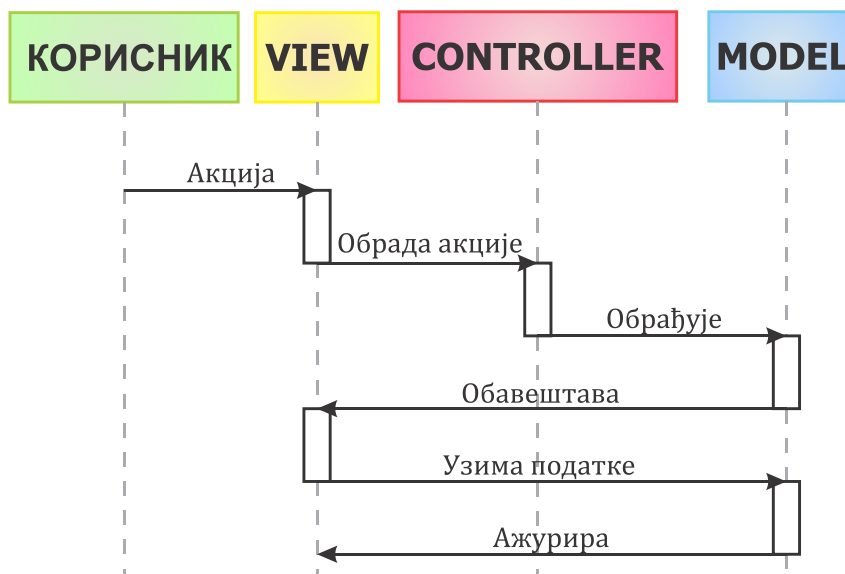


Слика 13: MVC образац за развој

Model се бави понашањем и подацима апликације. Неки од његових метода су упити који дозвољавају кориснику да добије информације о тренутном стању података. Такође има и методе које омогућавају да се стање података промени. *Model* има могућност да региструје погледе, и да их све оавести кад дође до промене стања.

View је одговоран за изглед и приказ података, сортирање, валидацију уноса, итд. *View* је потпуно изолован од било какве манипулације подацима. Такође обезбеђује и интерфејс за интеракцију са системом. Предност *MVC* приступа је то што подржава сваки вид интерфејса. Колико год да има приказа над подацима, њих контролише *Controller*. Приказ може да буде у виду *ASPX*, *PHP* или *HTML* странице, *Silverlight* или *Windows* апликације, као и апликација за мобилне телефоне.

Controller интерпретира корисников унос и, ако је неопходно, обавештава *Model* да промени стање. Он имплементира пословну логику апликације у којој је дефинисано кад, како и под којим условима треба мењати стање *Model*-а.



Слика 14: Секвенчни дијаграм *MVC* обрасца за развој

Ток података обично иде овако (види слику 14):

1. Корисник одради неку акцију на корисничком интерфејсу.
2. *Controller* обради ту акцију на основу догађаја са корисничког интерфејса.
3. *Controller* модификује стање *Model*-а у зависности од корисникове акције и у зависности од пословне логике апликације.
4. *View* користи *Model* индиректно да изгенерише одговарајући приказ података. *View* добија податке од *Model*-а. *Model* и *Controller* немају представу о приказу података.
5. Кориснички интерфејс чека даље инструкције, чиме се понавља читав процес.

Предности:

- Пошто се код *MVC*-а користе различити погледи на *Model* који је увек исти, олакшано је одржавати, тестирати и унапређивати апликацију.
- Олакшано је додавање нових приказа података. Довољно је само имплементирати њихов *Controller* и *View*.
- С обзиром да је *Model* у потпуности одвојен од приказа, имамо доста флексибилности при дизајнирању и имплементацији *Model*-а. Могуће је дизајнирати *Model* имајући за циљ поновну употребу (енг. *reusability*) и модуларност. *Model* је могуће додатно унапређивати.
- Могуће је паралелно развијати и *Model* и *View* и *Controller*.
- Апликације постају лако надоградиве и скалабилне⁵.

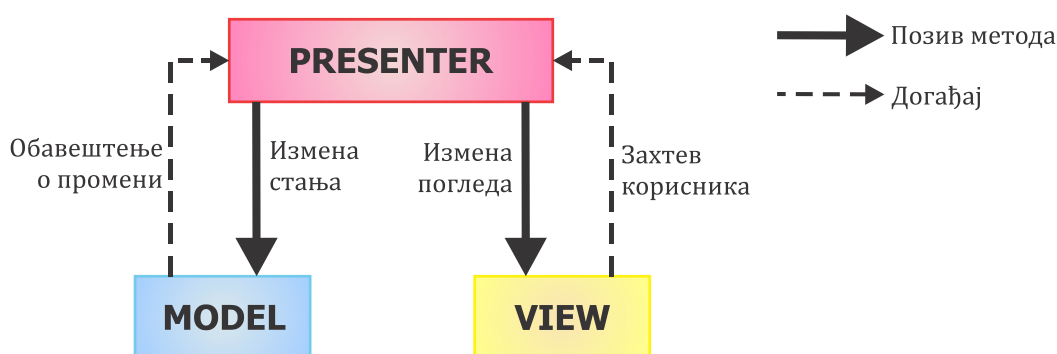
⁵ У телекомуникацијама и развоју софтвера, скалабилност (енг. *scalability*) је пожељно имати као својство система, мреже или процеса. Скалабилност значи да се са растом оптерећења систем или избори без проблема или да га је лако надоградити да би се изборио са додатним оптерећењем.

Мане:

- Захтева искусног пројектанта који може да на основу захтева испројектује све компоненте *MVC* обрасца за развој.
- Захтева више времена за анализу и дизајн.
- Дизајн није погодан за мале апликације.

3.2.3 Model-View-Presenter

Model-View-Presenter је варијација *MVC* обрасца за развој, специјално намењена моделу догађаја. Главна разлика у односу на *MVC* је то што *Presenter* имплементира *Observer* дизајн, али главна идеја остаје: *Model* чува податке, *View* приказује те податке на неки начин, а *Presenter* се бави пословном логиком и комуникацијом између друга два слоја.



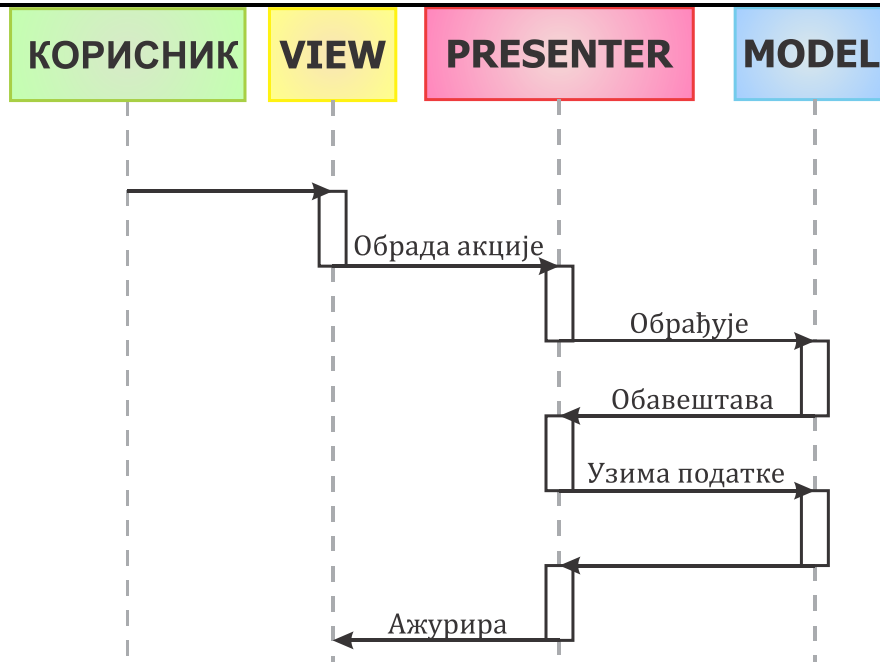
Слика 15: Концептуални изглед *MVP* обрасца за развој

MVP имплементира *Observer* методологију, на тај начин што *Presenter* интерпретира догађаје које му *View* прослеђује. У зависности од догађаја, *Presenter* модификује стање *Model*-а, и ако је потребно, промени *View*. *MVP* раздваја одговорности визуелног приказа и обрађивања догађаја у различите класе. *View* класа је задужена за приказ података, тј. задужена је за својства контрола корисничког интерфејса и за прослеђивање догађаја *Presenter* класи. *Presenter* садржи логику која одговара на догађаје и ажурира *Model* и *View* ако је потребно. Код *MVP*-а *Model* и *View* немају директну везу (слика 15).

Model је исти као већ описани *Model* код *MVC* обрасца за развој.

View се ажурира када се и *Model* ажурира. Ажурирање иде индиректно и искључиво преко *Presenter*-а. Овакав приступ се назива *Passive View*.

Presenter је задужен за обраду корисникових акција и за измену стања *Model*-а у зависности од корисникових акција. Корисникове акције прима *View* и он их прослеђује *Presenter*-у ради обраде. По промени стања *Model*-а, *Presenter* ажурира *View* у складу са променама и чека на даља упутства од стране корисника. *Presenter* мора да зна *Model* и бар један одговарајући *View*.



Слика 16: Секвенци дијаграм MVP обрасца за развој

Ток података обично иде овако (види слику 16):

1. Корисник изврши неку акцију у корисничком интерфејсу.
2. *Presenter* је обавештен о акцији преко механизма догађаја.
3. *Presenter* захваљујући функционалности која је у њега уграђена одради акцију чији исход може да представља промену стања *Model*-а.
4. *Presenter* је обавештен о промени стања *Model*-а, и на основу тога мења одговарајући приказ користећи податке из *Model*-а.
5. Кориснички интерфејс чека даље инструкције, чиме се понавља читав процес.

Предности:

- Као и код *MVC* обрасца за развој, *MVP* разграничава код корисничког интерфејса од кода пословне логике. Не само што смо извукли податке и преселили их у *Model*, него смо и скоро сву комплексну логику корисничког интерфејса пребацили у *Presenter*.
- Код *View* дела, скоро да немамо код, осим за исцртавање корисничког интерфејса. То теоретски значи да можемо врло лако да заменимо делове корисничког интерфејса, целе форме или чак цео кориснички интерфејс.
- *Unit* тестирање постаје много лакше с обзиром да је довољно тестирати само *Presenter* у коме се налази цела логика програма.

Мане (сличне као код *MVC*-а):

- Захтева искусног пројектанта који може да на основу захтева испројектује све компоненте *MVP* обрасца за развој.
- Захтева доста времена за анализу и дизајн.
- Дизајн није погодан за мале апликације.
- Отклањање грешака код модела са догађајима може да буде тешко.
- Код за правилно имплементирање обрасца за развој може да буде компликован, па се због тога не препоручује користити га у малим апликацијама.

3.3 Имплементација

Спајањем *Model-View-Presenter*-а, *C#*-а и *Windows Presentation Foundation*-а добија се имплементација графичког корисничког интерфејса за програм отвореног кода *QLab*.

QLab ће бити модуларна апликација где ће сваки модул бити једна *Model-View-Presenter* тројка, тј. сваки модул ће се састојати из делова као на слици 15. Тиме се постиже боља независност модула и могуће је модуле независно развијати што је важно због отворене природе кода.

3.3.1 *Prism*

Prism је скуп архитектуралних смерница дизањан да помогне да се лакше направи богат, флексибилан и лак за одржавање графички кориснички интерфејс. Поддржава технологије *Windows Presentation Foundation*, *Silverlight* и *Windows Phone 7*. Користећи обрасце за развој који обухватају важне принципе архитектуре апликације, као што су раздвајање брига (енг. *separation of concerns*) и лабаво повезивање (енг. *loose coupling*), *Prism* помаже у дизањирању и прављењу апликација чије су компоненте „лабаво“ повезане и које могу независно да се развијају али се такође лако и глатко уклапају у целокупну апликацију. Такве апликације се обично зову композитне апликације.

Prism је намењен програмерима који праве *WPF* или *Silverlight* апликације које обично имају много екрана, богату интеракцију са корисницима и визуализацију података и које у себи садрже доста логике за приказ података и доста пословне логике. Овакве апликације обично комуницирају са више позадинских система и сервиса и, користећи слојевиту архитектуру, могу бити физички раздвојене у више делова. Очекивано је да ће апликација временом еволуирати у одзиву на нове захтеве и пословне прилике. Укратко, овакве апликације су „направљене да трају“ и „направљене да се мењају“. Апликацијама које немају овакве захтеве се може десити да немају користи од *Prism*-а.

Смернице су дизањиране тако да помажу архитектама и програмерима да постигну следеће циљеве:

- Креирање апликације од модула који могу бити направљени, искомпјилирани и, опционо, пуштени у промет од стране различитих тимова, користећи притом *WPF* или *Silverlight*.
- Минимизовање међутимске зависности и омогућавање тимовима да се специјализују у различитим областима као што су: дизањн корисничких интерфејса, имплементација пословне логике, и развој инфраструктуре.
- Коришћење архитектуре која промовише поновну употребљивост кода.
- Повећање квалитета апликација апстрахујући сервисе који су доступни свим тимовима.
- Инкрементално интегрисање нових могућности.

Због лакоће коришћења, квалитетне документације (3) и свих осталих наведених предности, *Prism* јесте идеалан скуп смерница за тимски оријентисан пројекат као што је *QLab*.

3.3.1.1. Региони

Региони су концепт у *Prism*-у који дозвољавају корисницима да одређене делове интерфејса означе као регион у смислу да у регион може динамички да се смешта било какав приказ.

Региони су класе које имплементирају *IRegion* интерфејс. Сваки регион може да садржи нула или више контрола за приказ садржаја које се врло лако могу додати динамички.

Компонента која контролише регионе, и која иде уз *Prism* библиотеке се назива *Region Manager*. Он може да креира регионе у коду или у *XAML*-у. Апликације могу да садрже једну или више инстанци *Region Manager*-а, али ће се у *QLab*-у користити само једна.

3.3.1.2. Глобални догађаји

С обзиром да *Prism* промовише независност између компоненти апликације, јавља се питање како да те компоненте међусобно комуницирају. Један од начина који не нарушава концепт независности јесу глобални догађаји.

Компонента која се контролише догађаје се назива *EventAggregator*, и компоненте које желе да је користе могу добити референцу на инстанцу интерфејса *IEventAggregator*.

Да бисмо дефинисали догађај неопходно је да знамо који ће се тип објекта прослеђивати методама које слушају тај догађај, као и назив догађаја. Листинг 1 приказује пример дефиниције једног глобалног догађаја који као аргумент прослеђује објекат типа *String*.

```
using Microsoft.Practices.Prism.Events;

namespace QLab.Events
{
    public class BeforeCommandExecuting :
        CompositePresentationEvent<string> { }
}
```

Листинг 1: Пример дефиниције једног глобалног догађаја

За регистровање на глобални догађај је потребно дефинисати методу која ће се извршити при окидању догађаја, која као аргумент прима објекат типа који је дефинисан при креирању дефиниције догађаја. На листингу 2 се може видети комплетан пример регистровања на догађај.

```
[ImportingConstructor]
public ConsoleController(IEventAggregator EventAggregator)
{
    _eventAggregator = EventAggregator;
    _eventAggregator
        .GetEvent<CommandExecuted>()
        .Subscribe(OnCommandExecuted);
}

//метода која се позива када се окине догађај OnCommandExecuted
public void OnCommandExecuted(string Result)
{
    //обрада резултата
}
```

Листинг 2: Пример регистровања на глобални догађај

3.3.2 Сервиси

Комуникација између компоненти *QLab*-а ће се вршити преко сервиса који су доступни преко јавних интерфејса. Сваки сервис ће имплементирати одређени интерфејс који се односи на одређени део функционалности апликације. Сервиси ће

имплементирати *Singleton* образац за развој⁶. С обзиром да ће сервиси бити прослеђивани у виду референци ка инстанцама интерфејса, моћи ће се лако унапређивати или чак заменити новим, без утицаја на остатак кода апликације.

Да би компоненте интерфејса исправно радиле, мораће да при инстанцирању добију референцу ка једном или више сервиса.

Компонентама *QLab*-а ће бити доступни следећи сервиси:

Сервис за извршавање кода (енг. code execution service) – Један од главнијих сервиса, чији ће задатак бити да комуницира са engine делом апликације. Компонентама које користе овај сервис нуди методу за извршење произвољног кода. Кад се заврши извршавање кода, резултат се прослеђује преко глобалног догађаја који компоненте могу да ослушкују.

Сервис за рад са датотекама (енг. solution explorer service) – Овај сервис ће вршити контролу над свим датотекама које потпадају под неки пројекат. Компонентама које користе овај сервис нуди методе за додавање, брисање, преименовање и читање датотека, директоријума и њихових садржаја. При позивању било које од метода, резултат се одмах памти у дуготрајној меморији корисника апликације.

Сервис за обраду историје команди (енг. command history service) – Сервис који се бави историјом извршених команди. Под тиме се подразумева праћење које су команде извршене, брисање и чување старих команди. Компонентама које користе овај сервис нуди могућност да прочитају које су команде раније извршене, као и да позову поново неку од раније извршених команди. Комуницираће са сервисом за извршење кода.

Сервис за контролу отворених докумената (енг. tab service) – Сервис који се бави контролом отворених докумената. Компонентама које користе овај сервис нуди методе за отварање новог документа, као и затварање неког од постојећих. Сервис води рачуна о томе да ли су измене сачуване, и ако нису, пита корисника да ли жели да сачува измене пре затварања.

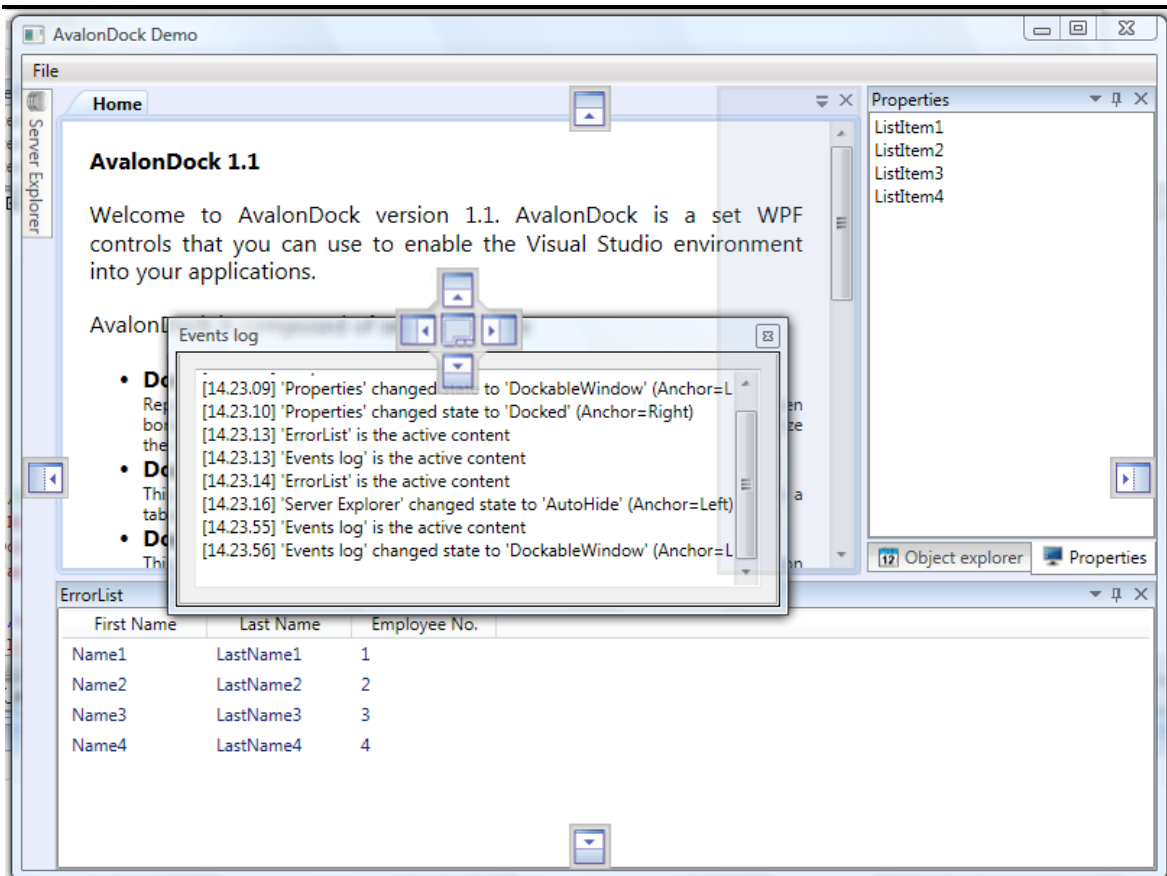
3.3.3 Главна маска

Главна маска (енг. shell) ће бити део *QLab* интерфејса који ће водити рачуна о другим модулима и биће први модул који се учита у меморију при покретању апликације.

Поред тога што ће у меморији организовати изглед модула, такође ће то радити и у приказу. Сваки приказани модул ће изгледати као на Слика 7. Биће у ствари део коју може да се независно помера. За ту функционалност је задужена контрола која се зове *Avalon Dock* (3), која је део већег пројекта отвореног кода *SharpDevelop* (4).

Avalon Dock је контрола која дозвољава велику флексибилност интерфејса што је и тражено у захтевима (одељак 2.2.13). На Слика 17 се види контрола у тренутку превлачења једног дела (модула) интерфејса. *Avalon Dock* организује модуле у прозору, језичке и видљивост модула.

⁶ *Singleton* је образац за развој који дефинише како се класе инстанцирају тако да увек буде креирана највише једна инстанца одређене класе. Углавном се користи код класа које врше неку контролу или управљају одређеним ресурсима.



Слика 17: Avalon Dock контрола за модуларан интерфејс

С обзиром да је контрола бесплатна и отвореног кода, могуће је њено коришћење у *QLab*-у. Потребно је скинути изворни код или компајлиране библиотеке и укључити их у пројекат да би биле видљиве интерфејсу *QLab*-а.

Модел података са којим ће радити главна маска јесу сами модули. За сваки модул ће бити сачувана његова позиција, у ком панелу се налази и да ли је видљив или скривен.

3.3.4 Ribbon

Овај део интерфејса спада под главну маску, али се води као модул који је независан од *Avalon Dock*-а. Овај део апликације приказује дугмад за већину команди које се налазе у *QLab*-у.

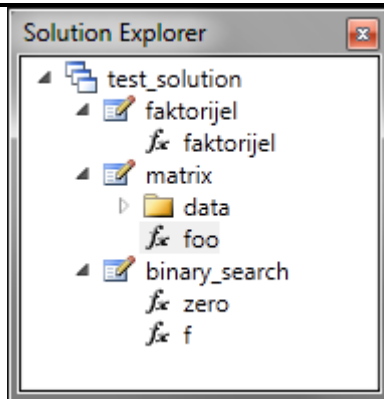
Ова контрола се налази у све више и више програма које испоручује компанија *Microsoft*, али и други произвођачи. Тренутно се не налази у стандардном пакету контрола које иду уз *.NET Framework 4* и *Visual Studio 2010*. Компанија *Microsoft* је изbacила *Ribbon* као бесплатну контролу отвореног кода (4) која може да се скине са сајта и користи у апликацијама.

Контролер ће обезбеђивати команде на које дугмад из приказа могу да се везују и позивају. Преко команди ће се одређивати која дугмад је активна, а која није.

3.3.5 Део за контролу датотека

Да би било могуће радити са датотекама у апликацији, потребно је имати део интерфејса који контролише и приказује датотеке.

Та компонента апликације ће бити имплементиран тако да подржава рад са дрвенастим структурама неограничене дубине.



Слика 18: Приказ датотека у пројекту

Модел података ће такође подржавати дрвенасту структуру, и то на следећи начин:

- Корен дрвета мора да буде развојни пакет (енг. *solution*)
- Директоријуми и пројекти могу да буду гране дрвета
- Датотеке морају да буду листови дрвета
- Сваки елемент дрвета треба да има листу своје деце

Контрола датотека ће на основу захтева бити у могућности да физички делује на структуру фајлова на диску. Моћи ће да креира,брише и мења имена датотекама, а такође и директоријумима.

При свакој измени структуре датотека и директоријума контрола ће чувати измене у виду *XML* датотеке која је због своје подршке за дрвенасту структуру и интеграције у *.NET Framework* идеална за коришћење. Информације о развојном пакету у пројектима унутар њега ће бити записане у *XML* датотеци са екстензијом *.qlsln* (листинг 3). Информације о пројектима унутар развојног пакета ће бити записане у *XML* датотеци са екстензијом *.qlproj* (листинг 4). Датотеке са изворним кодом ће имати екстензију *.q*.

```
<?xml version="1.0" encoding="utf-8"?>
<Solution
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" Name="Solution1.qlsln">
  <Project Name="Test1.qlproj" Folder="Test1">
    <File Name="test.q">
      <Buffer />
    </File>
  </Project>
</Solution>
```

Листинг 3: Пример сачуваног програмског пакета

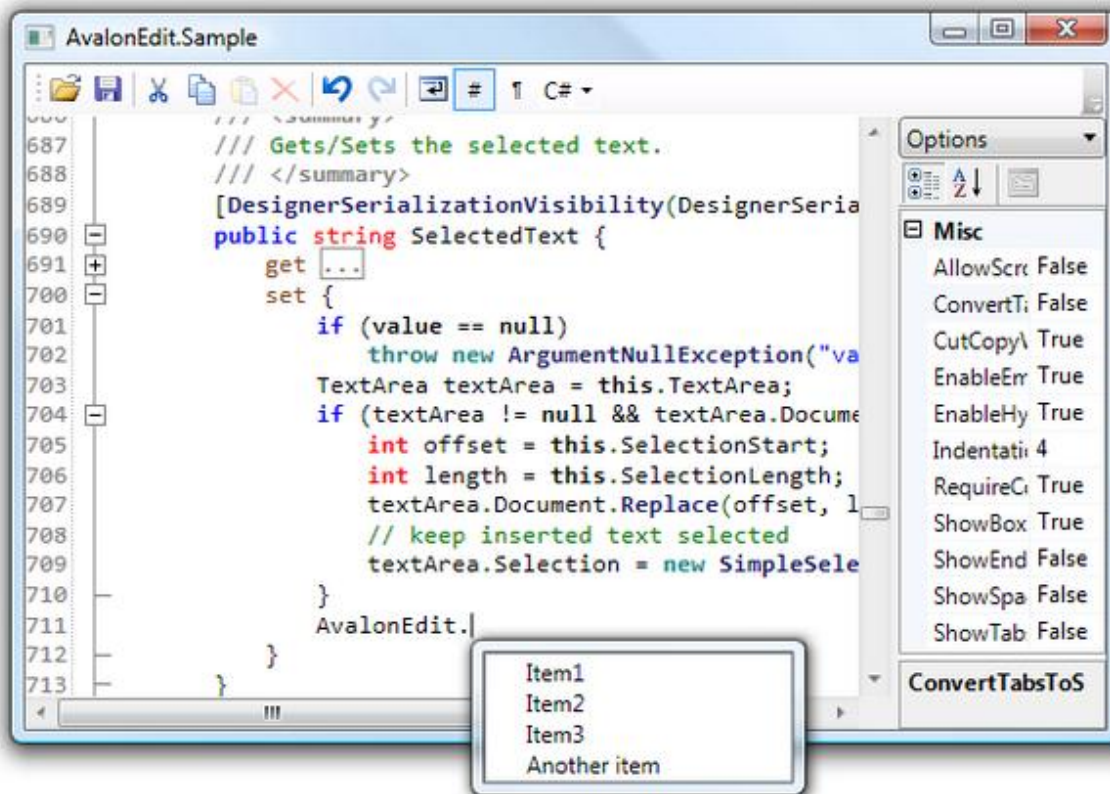
```
<?xml version="1.0" encoding="utf-8"?>
<Solution
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
Folder="Test1" Name="Test1.qlproj">
  <File Name="test.q">
    <Buffer />
  </File>
</Solution>
```

Листинг 4: Пример сачуваног пројекта

3.3.6 Уређивање кода

Контрола која се бави уређивањем кода јесте један део апликације који се највише користи. Из тог разлога треба обратити посебну пажњу на контролу за уређивање кода и обезбедити да ради брзо и ефикасно.

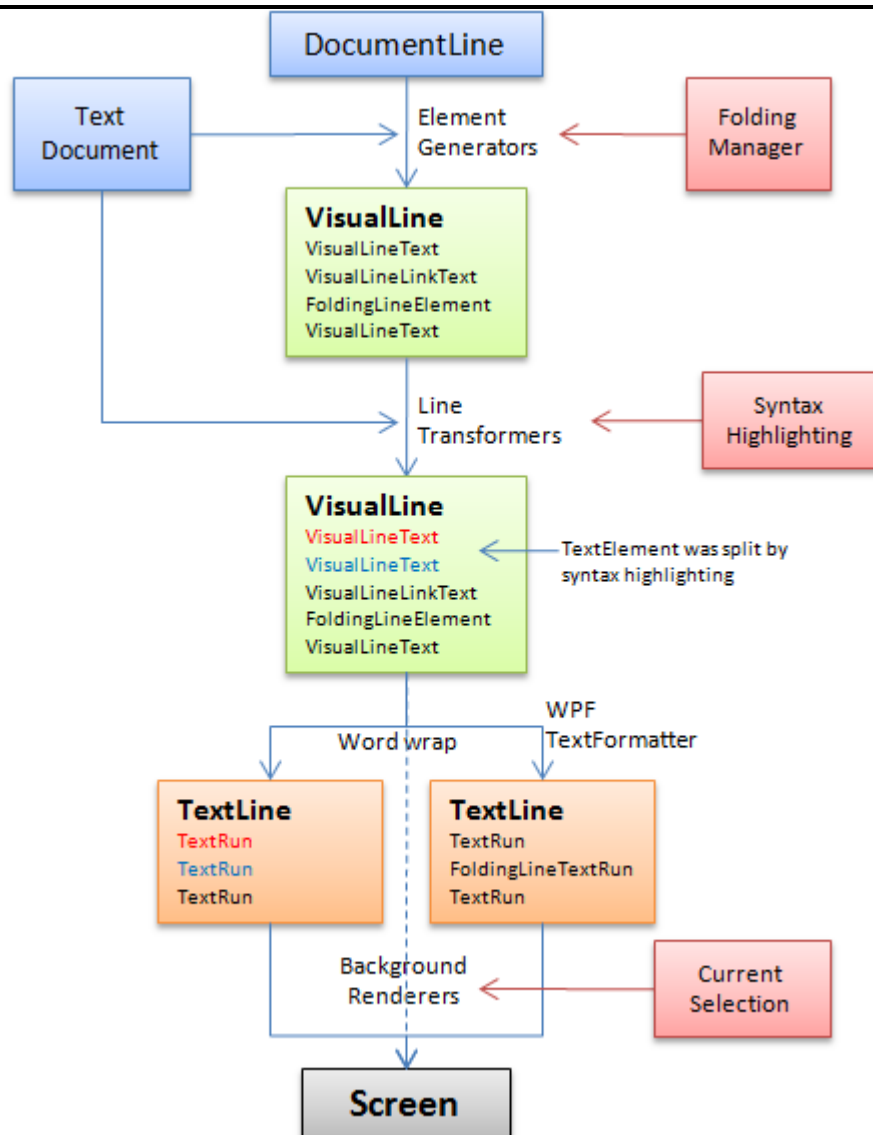
За *Windows Presentation Foundation* постоје нека решења отвореног кода која се баве овим проблемом, али решење које се највише истиче јесте *AvalonEdit* (5). Подржава све могућности наведене у захтевима и још много више. Такође има и врло добру документацију (6) (7), што је веома битно ради лакшег коришћења. Као и *AvalonDock*, *AvalonEdit* је део пројекта отвореног кода *SharpDevelop*.



Слика 19: *AvalonEdit* контрола приказује *C#* код

На слици 19 се види како изгледа *AvalonEdit* контрола подешена да ради са језиком *C#*. Такође се види и приказивање прозора који нуди кориснику шта да унесе.

Важно је напоменути процес кроз који унети текст пролази пре него што се обојен и форматиран прикаже, и то се може видети на слици 20.



Слика 20: Процес бојења и форматирања кода код *AvalonEdit* контроле

За потребе *QLab*-а неопходно је дефинисати елементе текста које ће *AvalonEdit* да боји. *AvalonEdit* подржава два начина за дефинисање језика. Један начин је кроз код, а други начин који ће се користити у овом пројекту је преко *XAML*-а. Листинг 5 приказује пример како се дефинишу коментари, кључне речи, ниске карактера и бројеви погодни за *AvalonEdit* контролу.

```

<SyntaxDefinition name="MATLAB"
xmlns="http://icsharpcode.net/sharpdevelop/syntaxdefinition/2008">
  <Color name="Comment" foreground="Green" />
  <Color name="String" foreground="Purple" />

  <!-- Ово је главни скуп правила. -->
  <RuleSet>
    <Span color="Comment" begin="%" />
    <Span color="Comment" multiline="true" begin="%{" end="%" />

    <!-- Дефиниција ниске карактера -->
    <Span color="String">
      <Begin>'</Begin>
      <End>'</End>
    </Span>
  
```



```

<!-- Кључне речи су плаве боје -->
<Keywords fontWeight="bold" foreground="Blue">
  <Word>if</Word>
  <Word>else</Word>
  <Word>while</Word>
  <!-- ... -->
</Keywords>

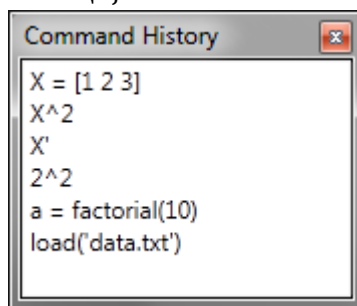
<!-- Бројеви -->
<Rule foreground="DarkBlue">
\b ( \d+(\.[0-9]+)? #цифра са опционим разломљеним делом
  | \.[0-9]+ #број који почиње са тачком
  )
  ([eE][+-]?[0-9]+)? #опциони експонент
</Rule>
</RuleSet>
</SyntaxDefinition>

```

Листинг 5: Дефиниција коментара, ниски карактера, кључних речи и бројева за *MATLAB* код

3.3.7 Историја команди

Ова компонента приказује списак раније позваних команди. За функционисање ће се ослањати на сервис за рад са историјом команди. На слици 21 се види историја команди одвојена од главне апликације.



Слика 21: Имплементација историје команди

Приказ се састоји од листе која је везана за колекцију команди коју контролише сервис. Сервис при извршавању команди пуни колекцију, а при додавању/брисању команди из колекције, приказ се аутоматски мења.

Контролер прати корисникове акције над листом команди, и у зависности од акције делује на одговарајући начин. Корисник може да:

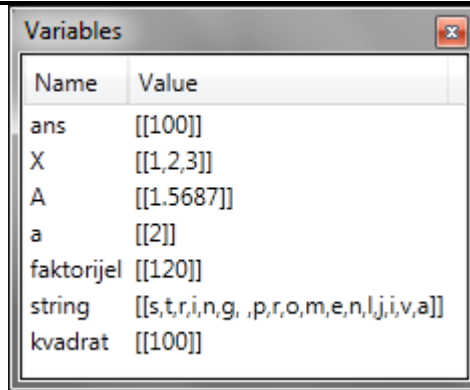
- Изврши команду из листе
- Обрише команду из листе
- Изабере једну или више команди
- Копира одабрану команду у меморију

Модел података у овом случају су саме команде, као и време извршења. Те информације је неопходно чувати да би историја команди могла правилно да се чува на диску.

3.3.8 Списак променљивих

У овој компоненти се приказује списак свих променљивих које су тренутно инстанциране, као и њихове вредности. За функционисање користи податке које добија од сервиса за извршење кода.

На слици 22 се види дизајн компоненте за приказ списка променљивих одвојен од главне апликације. За приказ се користи мрежа (*DataGrid*) која има колоне за име променљиве и за вредност променљиве.



Name	Value
ans	[[100]]
X	[[1,2,3]]
A	[[1.5687]]
a	[[2]]
faktorijel	[[120]]
string	[[s,t,r,i,n,g, ,p,r,o,m,e,n,l,j,i,v,a]]
kvadrat	[[100]]

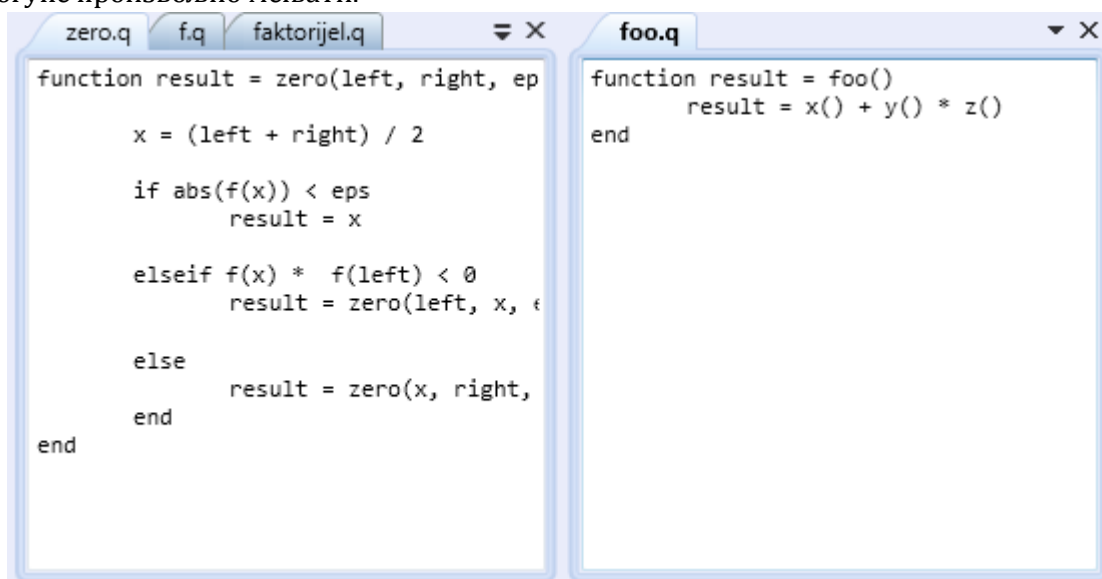
Слика 22: Имплементација списка променљивих

Контролер од сервиса за извршење кода добија списак променљивих које су тренутно инстанциране и смешта их у локалну колекцију. С обзиром да је приказ (односно мрежа) везан за ту колекцију, при измени колекције се аутоматски мења и приказ.

Модел података је у овом случају пар кључ-вредност где је кључ назив променљиве, а вредност текстуална репрезентација вредности променљиве.

3.3.9 Документа

Ова компонента приказује отворене датотеке у виду језичака. Сваки језичак у себи садржи компоненту за уређивање кода. Функционалност ове компоненте је уско везана за сервис за контролу отворених докумената. На слици 23 се виде четири отворена документа распоређених тако да су три са лево а један десно. Распоред је могуће произвољно мењати.



```

function result = zero(left, right, eps)
    x = (left + right) / 2
    if abs(f(x)) < eps
        result = x
    elseif f(x) * f(left) < 0
        result = zero(left, x, eps)
    else
        result = zero(x, right, eps)
    end
end

function result = foo()
    result = x() + y() * z()
end

```

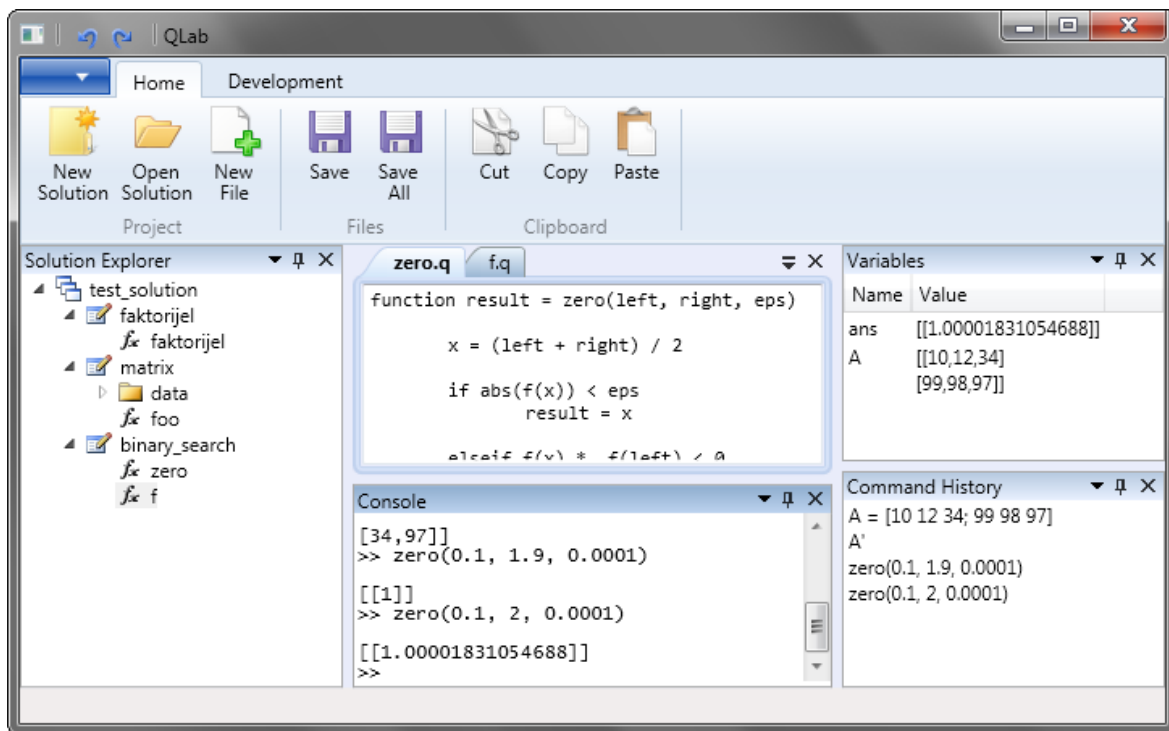
Слика 23: Приказ отворених датотека подељен на два дела

Приказ користи могућности контроле *AvalonDock* који симулира рад контроле у *Visual Studio*-у 2010.

Модел података јесте колекција тренутно отворених докумената.

4. Финални изглед

На слици се види изглед апликације у коме су имплементиране све функционалности из претходне секције.



Слика 24: Финални изглед апликације

Подразумевано, распоред компоненти у апликације је такав да је ribbon на врху, solution explorer са леве стране, датотеке са кодом се отварају у средини, конзола је у дну, а списак променљивих и историја команди десно. Наравно, могуће је преместити компоненте по жељи, осим *ribbon*-а који је фиксиран на врху (њего је могуће смањити тако да се види само први ред са називима језичака).

4.1.1 Прибављање кода апликације

Код апликације је јавно доступан и сви заинтересовани га могу скинути са следеће локације: qlab.codeplex.com. За покретање је неопходно имати инсталиран *.NET Framework 4*.

Компајлирање је препоручљиво вршити из *Visual Studio*-а 2010.

5. Закључак и даљи развој

Академској заједници су врло корисни пројекти отвореног кода које организују факултети, због којих факултети постају светски познати, а студенти који учествују у развоју преко потребну праксу и искуство.

Основни циљ *QLab* пројекта јесте да у једном тренутку преузме примат међу апликацијама сличне намене. За остарење тог циља важно је да свака целина апликације буде квалитетна, што укључује и интерфејс. Оно у чему се огледа квалитет интерфејса јесу следеће чињенице:

- Архитектура интерфејса је модуларна, односно подељена је у целине које су логички и имплементационо независне. Оваква организација пројекта гарантује квалитетан код и лакше касније унапређивање.
- Сваки од модула имплементира *Model-View-Presenter* образац за развој који додатно унапређује квалитет кода интерфејса. Овај образац је само једна у низу добрих пракси и решења примењених у интерфејсу *QLab*-а.
- За развој интерфејса се користе најновије технологије и прате се стандарди за дизајн који обезбеђују ефикасно извршавање, подршку за више платформи, као и угодан и ефикасан рад.

Иако интерфејс омогућава корисницима угодан и ефикасан рад, постоји простор за даље унапређење. Места где је могуће додатно развијати апликацију се могу сврстати у две категорије: усавршавање тренутних функционалности и подршка за нове функционалности имплементираних у другим подпројектима *QLab*-а.

У даљем развоју ће и студенти са других факултета дати свој допринос с обзиром да влада велика заинтересованост студената техничких наука за укључењем у пројекат.

6. Референце

- [1] Pash, Adam. Select Multiple Lines of Text in Firefox 3. *Lifehacker*. [На мрежи] [Цитирано: 20 Август 2011.] <http://lifehacker.com/396379/select-multiple-lines-of-text-in-firefox-3>
- [2] Одабир текста у апликацији Word 2007. *Microsoft Office web help*. [На мрежи] Microsoft. [Цитирано: 20 Август 2011.] <http://office.microsoft.com/en-us/word-help/select-text-NA010096402.aspx#BM1>
- [3] Microsoft Corporation. Developer's Guide to Microsoft Prism. *MSDN*. [На мрежи] Microsoft, Novembar 2010. <http://msdn.microsoft.com/en-us/library/gg406140.aspx>.
- [4] Avalon Dock. *Codeplex*. [На мрежи] <http://avalondock.codeplex.com/>
- [5] SharpDevelop. *SharpDevelop*. [На мрежи] ic#code. <http://www.icsharpcode.net/OpenSource/SD/>
- [6] Ribbon Overview. *MSDN*. [На мрежи] Microsoft
- [7] Grunwald, Daniel. AvalonEdit. *SharpDevelop*. [На мрежи] <http://wiki.sharpdevelop.net/AvalonEdit.ashx>.
- [8] *AvalonEdit Documentation*. 2011.
- [9] Using AvalonEdit (WPF Text Editor). *CodeProject*. [На мрежи] <http://www.codeproject.com/KB/edit/AvalonEdit.aspx>
- [10] Поређење апликација за вођење пројеката. *Wikipedia*. [На мрежи] [Цитирано: 5 Децембар 2010.] http://en.wikipedia.org/wiki/Comparison_of_project_management_software