UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

VOICECOMMANDER

PROPUSĂ DE:

ANDREI RÎȘCANU

SESIUNEA: IULIE 2018

COORDONATOR ȘTIINȚIFIC:

LECT. DR. ANCA IGNAT

UNIVERSITATEA „ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

VoiceCommander

Andrei Rîșcanu

Sesiunea: Iulie, 2018

Coordonator științific

Lect. dr. Anca Ignat

Avizat,

Îndrumător lucrare de licență

Lect.dr. Ignat Anca

28 iunie 2018

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Subsemntatul Rîșcanu Andrei cu domiciliul în Iași, Județul Iași, născut la data de 4 mai 1996, identificat prin CNP 1960504226731, absolvent al Universităţii ”Alexandru Ioan Cuza” din Iaşi, Facultatea de Informatică specializarea Informatică, promoţia 2018, declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: ”VoiceCommander” elaborată sub îndrumarea d-na Ignat Anca, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diplomă sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Iași, 28 iunie 2018

Rîșcanu Andrei

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul ”VoiceCommander”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iași, 28 iunie 2018

Rîșcanu Andrei

Cuprins

Introducere **6**

1 Contribuții **7**

2 Obiectiv **8**

2.1 Soluția propusă 8

2.2 Soluții existente 8

3 Fundamente **10**

3.1 C# 10

3.2 WPF (Windows Presentation Foundation) 10

3.3 Fody 11

4 Analiză și proiectare **12**

4.1 Model 12

4.2 View 13

4.3 ViewModel 14

4.4 Structuri de date 16

5 Detalii de implementare **19**

5.1 Pornirea aplicației 19

5.2 Interacțiunea cu aplicația 20

5.2.1 Interfața grafică 21
5.2.2 Linia de comandă 21
5.2.3 Recunoașterea vorbirii 23
5.3 Execuția comenzilor 24

6 Manual de utilizare **25**

Concluzii **28**

Bibliografie **29**

Introducere

Un sistem de fișiere controlează modul în care sunt salvate datele. Fără acesta nu am putea delimita care este începutul și care este sfârșitul unei informații salvate. Prin urmare, datele trebuie separate în grupuri și fiecărui grup îi trebuie atribuit un nume după care va fi identificat. Astfel, un grup de date este numit un *fișier*. Structura și regulile după care sunt administrate grupurile de fișiere formează un *sistem de fișiere*.

Modul prin care o persoană creează, citește, modifică sau șterge un fișier dintr-un sistem diferă în funcție de cunoștințele și preferințele acestuia. Observând cu atenție persoanele din jur și modul în care acestea operează calculatorul, se poate ajunge la concluzia că utilizatorii pot fi clasificați în una din 3 categorii:

- standard: cei ce folosesc în general o interfață grafică pentru a-și îndeplini sarcinile;
- avansat: în această categorie se află persoanele ce pot manevra un sistem de fișiere folosind linia de comandă;
- invalid: persoane ce nu pot opera fizic un calculator datorită unui handicap.

Analizând piața, realizăm că pentru fiecare tip de persoană din aceste categorii există o gamă largă de aplicații ce îi pot ajuta să opereze calculatorul cu ușurință, dar nu există o aplicație universală și intuitivă totodată. Sau, dacă există, aceasta este greu de găsit și, prin urmare, nu oricine va avea acces la ea.

Capitolul 1

Contribuții

Principalele contribuții aduse aplicației *VoiceCommander* sunt următoarele:

- implementarea *design pattern*-ului *MVVM*;
- crearea interfeței grafice;
- reprezentarea elementelor unui sistem de fișiere în cod și în interfață;
- implementarea sistemului de recunoaștere a vorbirii folosind biblioteca *System.Speech.Recognition* din *C#*;
- implementarea comenzilor folosite de interfața grafică, linia de comandă și de sistemul de recunoaștere a vorbirii.

Capitolul 2

Obiectiv

2.1 Soluția propusă

Se dorește realizarea unei aplicații ce poate fi utilizată în 3 moduri: folosind interfața grafică, linia de comandă și/sau recunoaștere a vorbirii. Utilizatorul, indiferent de experiența anterioară în operarea calculatorului, trebuie să poată folosi cu ușurință oricare din cele 3 moduri descrise. În plus, în cazul în care utilizatorul dorește să schimbe modul de operare al aplicației, tranziția va fi foarte ușoară și oricând posibilă.

Aplicația trebuie sa fie modularizată, astfel încât să poată fi extinsă cu ușurință în viitor. Principalele adiții plănuite pe viitor ar fi adăugarea mai multor comenzi și portarea aplicației către alte platforme. Aplicația va fi *open-source*, astfel încât ea va putea fi îmbunătățită de orice persoană doritoare.

2.2 Soluții existente

Pentru a putea construi o aplicație folositoare a fost necesară studierea pieței. În urma acestei investigări, *VoiceCommander* a fost modelat în funcție de avantajele și dezavantajele celorlalte aplicații ce servesc același scop: manevrarea sistemului de fișiere.

Principala sursă de inspirație a fost aplicația *Total Commander*. Influența acestuia este reflectată mai ales în interfața grafică a *VoiceCommander*, având aceeași structură de afișare a fișierelor/directoarelor. În partea de jos a ambelor aplicații este prezentă linia de comandă, dar implementată diferit si cu funcționalități diferite. Ce lipsește însă aplicației *Total Commander* este componenta de recunoaștere vocală și simplitatea, având tendința de a părea copleșitoare pentru un utilizator neexperimentat.

Sistemul de operare *Windows* are propiul asistent personal numit *Cortana* pentru a putea executa diferite acțiuni folosind recunoaștere a vorbirii, dar acesta nu este specializat pentru operații cu fișiere și nu are o interfață grafică. O alternativă ar fi *Windows Speech Recognition*, ce poate fi folosit drept *mouse* pentru *File Explorer*, dar nici acesta nu este specializat în manevrarea fișierelor.

Nu în ultimul rând, un rol foarte important îl are și linia de comandă din sistemele de operare *UNIX*, majoritatea comenzilor din *VoiceCommander* fiind inspirate de aceasta. Deși foarte complexă și utilă, linia de comandă nu conține o interfață grafică și nici funcționalitatea de recunoaștere a vorbirii. Pe deasupra, aceasta este mult prea copleșitoare pentru majoritatea oamenilor, în special pentru utilizatorii standard ai unui calculator.

Pe lângă soluțiile prezentate până acum, există și alte aplicații ce îndeplinesc același scop, dar majoritatea nu au implementată funcționalitatea de recunoaștere a vorbirii, iar cele care au această funcționalitate, în general nu au cel puțin una din celelalte 2 componente prezente la *VoiceCommander* (interfață grafică și linie de comandă).

Capitolul 3

Fundamente

VoiceCommander este o aplicație creată pe platforma *.NET* pentru a fi folosit pe dispozitive ce folosesc sistemul de operare *Windows*, urmând ca mai târziu sa se facă trecerea și către *UNIX*. Aplicația folosește următoarele tehnologii din cadrul *.NET*:

3.1 C#

Acest limbaj este relativ ușor de înțeles și utilizat, dezvoltat de către *Microsoft* pentru a fi folosit la o gamă largă de aplicații. O caracteristică importantă este compatibilitatea sa cu *WPF* privind crearea aplicațiilor *desktop*, motiv pentru care *VoiceCommander* a fost creat folosind acest limbaj.

Pentru partea de recunoaștere a vorbirii, aplicația folosește biblioteca *SpeechRecognitionEngine* din *C#*. Aceasta oferă acces la funcționalitatea de recunoaștere a vorbirii prin primirea unui flux audio ca intrare ce îl transformă apoi într-o transcriere de text.

3.2 WPF (Windows Presentation Foundation)

Folosit pentru redarea interfețelor utilizatorilor în aplicațiile bazate pe *Windows, WPF* este un subsistem grafic de la *Microsoft* care unește o serie de elemente comune ale interfeței

cu utilizatorul, cum ar fi redarea 2D / 3D, documente fixe și adaptive, tipografie, grafică vectorială, animație în timp real și medii *pre-rendered*. Aceste elemente pot fi apoi legate și manipulate pe baza diferitelor evenimente, interacțiuni ale utilizatorilor și legări de date.

WPF este folosit în cadrul *VoiceCommander* pentru a oferi utilizatorului o interfață cu care să poată interacționa, modificările putând fi observate aproape instant datorită procesului de *Binding*. Un alt motiv pentru care aplicația folosește *WPF* este datorită libertății pe care o oferă programatorului în a crea o interfață grafică.

3.3 Fody

Este o bibliotecă extensibilă care poate genera și manipula codul intermediar (*IL*) în timpul construcției. Motivul pentru care a fost aleasă această bibliotecă este de a reduce semnificativ codul necesar stabilirii unei legături între *View* și *Model*. Utilizarea sa poate fi observată în clasa *BaseViewModel*, unde apare următoarea secvență de cod:

```
[AddINotifyPropertyChangedInterface]
public class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged = (sender, e) => { };
}
```

În continuare, această clasa este moștenită de alte 4 clase, în care există un număr total de 9 variabile ce sunt legate de diverse elemente din interfața grafică. Folosind *Fody*, timpul necesar implementării aplicației și întreținerii acesteia a fost redus semnificativ.

Capitolul 4

Analiză și proiectare

Pentru a separa interfața grafică de datele utilizatorului, aplicația implementează *architecture pattern*-ul MVVM (Model-View-ViewModel). În continuare vom descrie modul prin care s-a realizat acest obiectiv și ce structuri de date au fost folosite în cadrul *VoiceCommander*.

4.1 Model

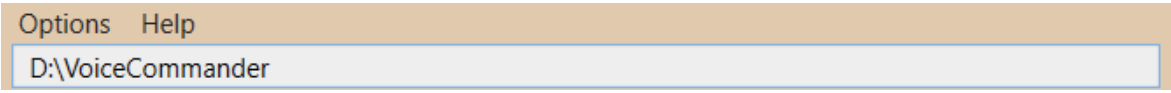
În cadrul VoiceCommander, Model-ul este constituit de reprezentarea abstractă a următoarelor elemente din cadrul unui calculator: *drive*, *folder* și *file*. În *enum*-ul DirectoryItemType se găsesc tipurile declarate mai sus, iar în clasa DirectoryItem se află diverse informații despre un element, precum:

- Type: este un element de tipul DirectoryItemType;
- FullPath: calea absolută a elementului respectiv;
- Name: numele cu care este salvat un element în memorie.

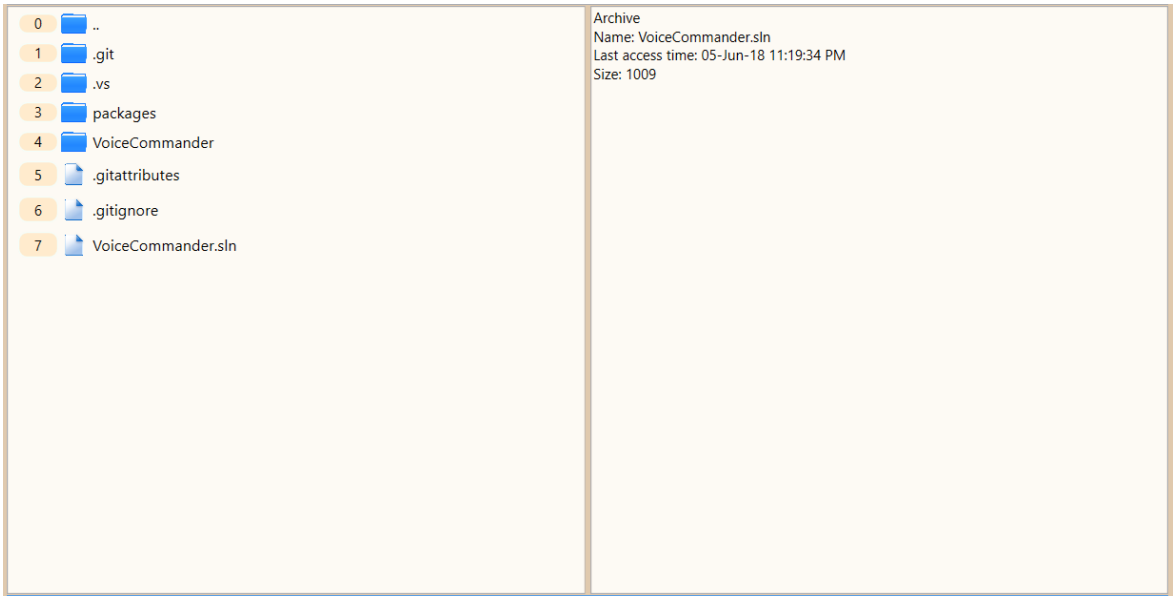
4.2 View

Interfața grafică este formată din 3 componente majore: bara de meniu, poziționată în partea de sus a ferestrei, linia de comandă, în partea de jos, și ferestrele de afișare care sunt plasate în mijlocul interfeței. Acestea sunt împachetate în fereastra aplicației folosind *control*-ul *DockPanel*.

Bara de meniu este la rândul ei formată din 2 componente suprapuse: bara de opțiuni și un *TextBox* de tipul *readonly* ce conține calea către directorul curent.



Ferestrele de afișare sunt despărțite de un separator (componenta *GridSplitter*) pentru a permite utilizatorului să poată selecta dimensiunea unei ferestre in raport cu cealaltă. În partea stângă este amplasată fereastra de afișare a sistemului de fișiere, fiecare element conținând indicele cu care poate fi accesat din linia de comandă sau utilizând recunoaștere vocală, imaginea ce reprezintă tipul elementului și numele cu care acesta este salvat. În partea dreaptă se află fereastra ce afișează utilizatorului rezultatul unei comenzi vocale sau date prin linia de comandă.



Linia de comandă este implementată în fișierul *CommandLineView.xaml* și are ca scop permiterea introducerii comenzilor sub formă de text de către utilizator.

4.3 ViewModel

Componenta *ViewModel* se diferențiază de componenta *Controller* din cadrul MVC (Model-View-Controller) prin faptul că aceasta se bazează foarte mult pe interfața grafică a aplicației, folosind un *binder* pentru a face legătura între interfață si date.

În cadrul VoiceCommander, *ViewModel*-ul este reprezentat de 4 clase ce moștenesc clasa de bază *BaseViewModel*: *CommandLineViewModel, DirectoryItemViewModel, DirectoryStructureViewModel* și *OutputStringItemViewModel*. Clasa părinte folosește atributul *AddINotifyPropertyChangedInterface* din cadrul Fody pentru a simplifica scrierea și întreținerea codului.

CommandLineViewModel este clasa ce interpretează comanda dată de utilizator și o execută în cazul în care este validă. Linia de comandă poate apela funcția *ExecuteCommand* din cadrul acestei clase în 4 moduri:

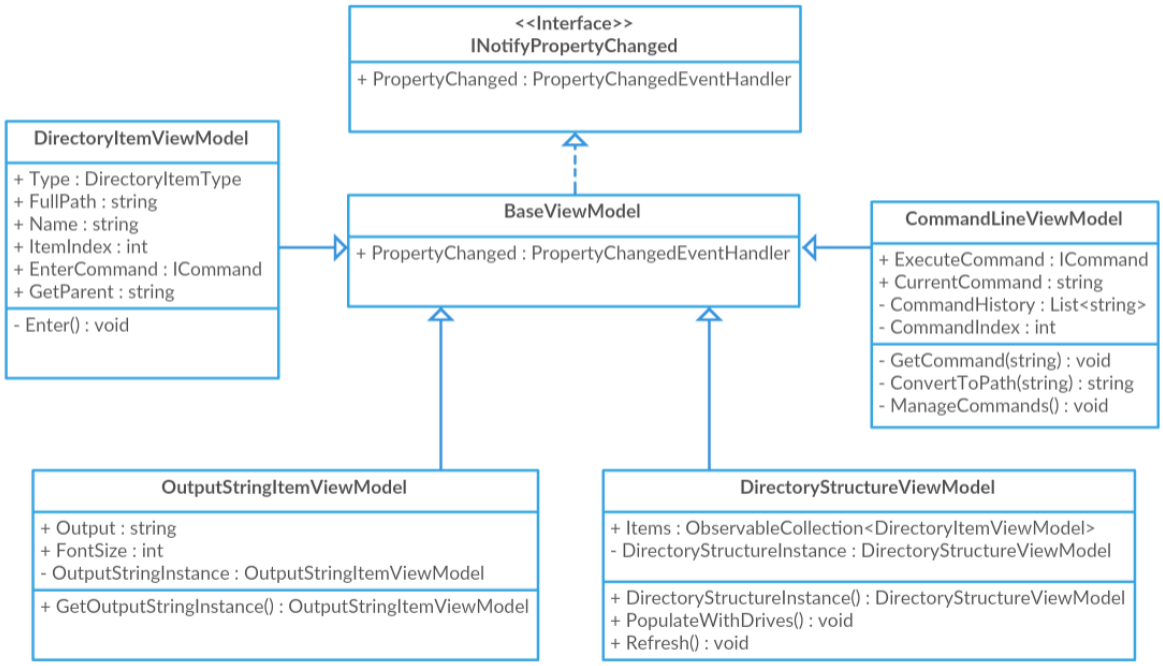
- apăsarea tastei *Enter*, caz în care programul va prelua textul scris până în acel moment și îl va da ca parametru funcției unde va fi verificată validitatea comenzii date;
- apăsarea tastei *Up* (săgeată sus), pentru ca aplicația să preia ultima comandă executată și s-o plaseze în spațiul destinat tastării comenzilor;
- apăsarea tastei *Down* (săgeată jos), pentru ca aplicația să preia următoarea comandă executată față de cea selectată (când este posibil) și s-o plaseze în spațiul destinat tastării comenzilor;
- apăsarea tastelor *Ctrl* și *F* concomitent, pentru ca aplicația sa atașeze la sfârșitul textului din spațiul destinat scrierii comenzilor calea către directorul curent.

Clasa *CommandLineViewModel* memorează un număr limitat de comenzi (în ordinea în care au fost executate) într-o listă pentru a putea fi ușor accesate de utilizator în cazul în care acesta dorește sa repete o comandă sau să vadă ultimele comenzi executate. În plus, pentru a simplifica munca utilizatorului, în momentul în care funcția de parcurgere a textului introdus de utilizator întâlnește o sintaxă de forma \n (unde n = număr cuprins între 0 si numărul total de elemente din directorul curent), aplicația va înlocui această sintaxă cu calea către elementul al cărui index corespunde acelui număr.

DirectoryItemViewModel este clasa a cărei instanță reprezintă un element din calculator. Aceasta reține tipul obiectului (*drive, folder, file*), calea absolută către acesta, numele, indexul și părintele. În cazul în care obiectul nu este de tipul drive, părintele acestuia va fi afișat ca primul element din interfață cu numele "..". Pe lângă variabile, clasa pune la dispoziție și funcția *Enter* ce verifică dacă este posibilă accesarea elementului (în cazul în care este un drive sau folder) și execută acțiunea, repopulând lista cu elemente.

În clasa *DirectoryStructureViewModel* sunt salvate elementele din calea curentă, într-un recipient de tipul *ObservableCollection*, și sunt implementate funcțiile necesare repopulării recipientului și reîmprospătării interfeței grafice. Clasa implementează design pattern-ul Singleton și este setată ca fiind *DataContext*-ul aplicației.

OutputStringItemViewModel are ca rol setarea textului din fereastra de afișare și mărimea acestuia.



4.4 Structuri de date

O componentă esențială din cadrul *VoiceCommander* este sistemul de fișiere. Pentru a le putea reprezenta, aplicația folosește un recipient de tipul *ObservableCollection* în care salvează elementele curente (*drive, folder, file*). Motivul pentru care este folosit acest tip de recipient este ca interfața să poată stabili o legătură cu acesta (*Data Binding*).

Pentru a salva comenzi și a le putea apela în funcție de un cuvânt cheie ales, aplicația

folosește un dicționar de tipul *IDictionary*, ce primește un *string* drept cuvânt cheie și o funcție de tip *Action* ca valoare. A fost adoptată această metodă deoarece astfel pot fi adăugate comenzi noi cu ușurință în cadrul aplicației și pot fi folosite de toate funcționalitățile (interfață, linie de comandă, recunoaștere a vorbirii). În plus, este foarte ușor de modificat cuvântul cheie prin care o funcție este apelată.

Pe lângă structurile de date descrise până acum, *VoiceCommander* folosește un număr de structuri de clase ce ajută la îndeplinirea diferitelor funcționalități, grupate în funcție de sarcinile pe care le îndeplinesc:

- directorul *CommandLine* conține clasele *Command, LineCommands* și *Register*; acestea sunt folosite în a implementa comenzile folosite de utilizator și a oferi un mod de a le stoca;
- directorul *Commands* conține clasele *ExecuteCommand* și *RelayCommand*; acestea au ca scop punerea la dispoziție a unei comenzi la care se poate face *Binding* pentru a putea fi apelată atunci când un anumit eveniment are loc în *View* (spre exemplu, apăsarea unei taste);
- directorul *Converters* conține clasele *GridWidthToListViewWidthConverter, HeaderToImageConverter* și *ImageSizeToFontSizeConverter*; acestea sunt folosite pentru a putea modifica diferite componente ale interfeței în funcție de Model (și invers), atunci când o variabilă este modificată (spre exemplu, în funcție de tipul elementului, va fi afișată imaginea corespunzătoare);
- directorul *Helpers* conține clasa *OutputStrings*, unde sunt memorate diferite constante de tip *string*;
- clasa *DirectoryStructure*, în care sunt salvate variabile și funcții ajutătoare precum numărul total de elemente afișate, extragerea numelui unui element in funcție de calea

sa absolută, extragerea unui element de la o anumită poziție din recipient, extragerea conținutului unui director sau obținerea *drive*-urilor;

- clasa *SpeechFunctionality*, unde este implementată funcționalitatea de recunoaștere a vorbirii; această clasă este instanțiată la pornirea aplicației și conține un dicționar extensibil în care sunt salvate cuvintele cheie pentru care sunt executate diferite operațiuni.

Capitolul 5

Detalii de implementare

Aplicația VoiceCommander a fost implementată astfel încât să fie optimă din punct de vedere al timpului de execuție și ușor de extins. Modul prin care s-au atins aceste obiective va fi explicat în cele ce urmează.

5.1 Pornirea aplicației

La pornirea aplicației, clasa *MainWindow* execută 3 comenzi:

```
public MainWindow()
{
    InitializeComponent();

    SpeechFunctionality.InitializeSpeechFunctionality();

    this.DataContext = DirectoryStructureViewModel.GetDirectoryStructureInstance();
}
```

Prima comandă este apelul funcției *InitialiazeComponent*, în care este inițializată interfața grafică a aplicației. De asemenea, în acest moment sunt făcute legăturile dintre *View* și *Model*. O legătură arată astfel:

```
<TextBox Text="{Binding Items[0].GetParent, Mode=OneWay}"/>
```

Folosind sintaxa *Binding Items[0].GetParent*, textul din partea de sus a aplicației ce reprezintă calea către directorul curent va fi modificat de fiecare dată când primul element din recipientul *Items* se va schimba. *GetParent* este un *string* ce returnează calea întreagă către componenta selectată. Motivul pentru care legătura se face la primul element este că întotdeauna acesta va fi directorul părinte al celorlalte elemente. *Mode=OneWay* indică vizibilitatea modificărilor ale acelui element. În acest caz, *OneWay* înseamnă că modificările din *Model* vor fi vizibile în interfață, dar nu și invers.

A doua comandă inițializează clasa ce implementează funcționalitatea de recunoștere a vorbirii prin apelarea funcției ce setează un dicționar de cuvinte accesibile utilizatorului. În momentul în care aplicația detectează unul din cuvintele existente în dicționar, este apelată o funcție ce împarte textul primit în cuvinte, primul reprezentând comanda ce se dorește a fi executată iar restul cuvintelor fiind parametrii, urmând ca apoi să fie apelată funcția corespunzătoare din clasa *Command*.

A treia comandă setează *DataContext*-ul aplicației ca fiind instanța *Singleton*-ului *DirectoryStructureViewModel*. În momentul în care aplicația este pornită, variabila ce salvează unica instanță a acestei clase va apela constructorul, în interiorul căruia este apelată funcția *PopulateWithDrives*. La rândul ei, această funcție inițializează recipientul public *Items* cu o reprezentare a *drive*-urilor din calculator, salvate ca *DirectoryItemViewModel*.

5.2 Interacțiunea cu aplicația

După pornirea aplicației, utilizatorul va putea interacționa cu aceasta prin unul din cele 3 moduri disponibile: interfață grafică, linie de comandă, recunoaștere a vorbirii. Modurile enumerate sunt implementate diferit, dar au și lucruri in comun.

5.2.1 Interfața grafică

Utilizatorul poate interacționa cu aplicația doar prin folosirea unui mouse. Câteva din acțiunile pe care le poate executa acesta sunt schimbarea locației curente, schimbarea unor opțiuni din cadrul aplicației și mărirea sau micșorarea ferestrelor de afișare.

În momentul în care utilizatorul execută comanda de a intra într-un director, aplicația, ce are fiecare reprezentare a *drive*-urilor și *folder*-urilor din interfața grafică ”legată” la o variabilă de tipul *ICommand* din clasa *DirectoryItemViewModel*, va verifica în primul rând ca elementul selectat să nu fie un fișier, după care va verifica dacă utilizatorul a dorit să iasă din directorul curent sau să intre într-unul nou. Următorul pas este golirea recipientului *Items* și repopularea lui cu elementele necesare. Cum aplicația folosește MVVM, schimbările sunt imediat vizibile în interfață.

Dacă utilizatorul mișcă separatorul ferestrelor de afișare în stânga sau dreapta, acesta va remarca faptul că imaginea, indexul și titlul elementelor din fereastra ce afișează sistemul de fișiere se vor micșora sau mări până la un anumit punct. Acest lucru este posibil datorită procesului de *Data Binding* disponibil în *WPF*. Mai precis, fiecare din cele 3 elemente (imagine, index, titlu) are mărimea stabilită ca un procent din mărimea totală a ferestrei. În momentul în care utilizatorul schimbă mărimea ferestrelor de afișare, componentele ale căror proprietăți sunt legate de aceasta vor fi de asemenea modificate.

5.2.2 Linia de comandă

Dacă utilizatorul dorește sa opereze sistemul de fișiere utilizând doar tastatura, aplicația îi pune la dispoziție o linie de comandă în cadrul căruia acesta poate executa diferite operații. Pentru a putea realiza acest lucru, utilizatorul trebuie doar să introducă în linia de comandă cuvinte cheie precum *cd, mv, info* etc. urmate de parametrii necesari (unde este cazul).

Pentru a putea folosi această funcționalitate, aplicația instanțiază clasa *Register* la pornirea aplicației. Această clasă conține doar un constructor, unde este apelată funția *Register* din clasa *Command*. Constructorul apelează această funcție astfel:

```
Command.Register("count", LineCommands.CountElements);
Command.Register("clear", LineCommands.Clear);
Command.Register("info", LineCommands.ShowItemDetails);
Command.Register("fontsize", LineCommands.ChangeFontSize);
Command.Register("cd", LineCommands.ChangeLocation);
Command.Register("read", LineCommands.ReadContent);
```

Primul parametru este un *string* ce reprezintă numele unei comenzi, iar al doilea este funcția ce urmează să fie apelată folosind primul parametru. În clasa *Command*, acești parametri constituie o intrare într-un recipient de tip *Dictionary*, primul reprezentând cheia iar al doilea fiind valoarea, salvat ca funcție din clasa *LineCommands*.

Pentru ca aplicația să detecteze când un utilizator dorește să execute o comandă, a fost stabilită o legătură între cheia *Enter* și variabila *ExecuteCommand* din *CommandLineViewModel*. *ExecuteCommand* este de tipul *ICommand*, și primește ca parametru la inițializare o funcție ce va fi salvată într-o variabilă internă. În momentul în care un utilizator apasă tasta *Enter*, programul apelează funcția *GetCommand* cu textul introdus de utilizator dat ca parametru, unde va fi extras primul cuvânt din text și examinat în scopul de a vedea dacă acesta reprezintă o comandă existentă în dicționar. Dacă răspunsul este unul afirmativ, comanda va fi apelată împreună cu restul cuvintelor ce vor fi interpretate ca parametri ai comenzii. În funcție de scopul comenzii executate, ferestrele de afișare își vor schimba conținutul.

Având în vedere modul în care funcțiile sunt salvate și apelate, extinderea comenzilor posibile este foarte simplă. Metoda trebuie implementată în *LineCommands* (sau se poate crea o altă clasă dacă se dorește) și salvată în dicționar, având ca și cheie cuvântul cu care va fi apelată.

5.2.3 Recunoașterea vorbirii

După ce aplicația este pornită și funcționalitatea de recunoaștere a vorbirii este inițializată, utilizatorul poate folosi comenzi vocale pentru a apela funcții folosite și de linia de comandă. Atunci când un cuvânt (sau o serie de cuvinte) din dicționar este detectat, aplicația va verifica acuratețea cu care a fost recunoscut cuvântul, iar daca aceasta este destul de mare, va verifica ce funcție se dorește a fi apelată din clasa *Command*. Toate funcțiile ce pot fi apelate sunt salvate în dicționarul din clasa *Command*, așadar, în cazul în care se va dori ca aplicația să primească îmbunătățiri la comenzile existente sau vor fi implementate noi comenzi, acestea vor fi vizibile în ambele funcționalități ale aplicației (linie de comandă și recunoaștere a vorbirii).

```
public static SpeechRecognitionEngine recognizer = new SpeechRecognitionEngine();
private static Choices Commands = new Choices(new string[] { "count", "back", "clear"
    "enter", "fontsize", "info", "read", "file", "folder", "delete" });
private static Choices Numbers = new Choices(PopulateWithNumbers(100));
private static Choices Alphabet = new Choices(new string[] {"a", "b", "c", "d", "e",
```

În imagine se observă faptul că există 3 variabile ce salvează opțiunile disponibile utilizatorului. În variabila *Commands* sunt salvate cuvintele cheie ce reprezintă comenzile accesibile utilizatorului. În variabila *Numbers* este apelată la inițializare o funcție ce returnează un vector de *string*-uri, conținând numere de la 0 la parametrul dat (în cazul de față, 100). În ultima variabila, *Alphabet*, sunt salvate toate literele alfabetului englez. Având în vedere modul în care sunt stocate aceste chei, pentru a adăuga noi cuvinte ce vor apela o comandă sau pentru a le modifica pe cele existente, este de ajuns să modificăm acești vectori.

Motivul pentru care există 3 variabile ce salvează diferite *string*-uri este că utilizatorul poate apela 3 tipuri de comenzi:

- comenzi ce nu au nevoie de parametri, precum *back* sau *count*;
- comenzi ce au nevoie de un număr dat ca parametru, precum *fontsize*, pentru a modifica mărimea textului din fereastra de afișare, sau *enter*, pentru a intra în directorul cu

indexul egal cu numărul dat ca parametru;

- comenzi ce au nevoie de un *string*, precum *file* sau *folder*, pentru a putea crea un fișier (sau director) cu numele ales de utilizator prin pronunțarea fiecărei litere pe rând.

5.3 Execuția comenzilor

În momentul în care utilizatorul dorește să execute o comandă, aplicația va apela funcția *GetCommand*, unde va verifica întâi dacă utilizatorul a dorit să vadă o comandă precedentă (prin apăsarea tastei *Up/Down*). În caz afirmativ, aplicația va introduce comandă precedentă, ce a fost salvată într-o listă, în linia de comandă. În caz negativ, aplicația va continua prin a verifica dacă utilizatorul a dorit să adauge calea curentă în linia de comandă (prin apăsarea simultană a tastelor *Ctrl* și *F*). Dacă utilizatorul nu a intenționat să facă acest lucru, înseamnă că acesta a dorit să execute o comandă. Așadar, aplicația va verifica existența comenzii în dicționar, iar în cazul în care aceasta există, va apela funcția *ConvertToPath* cu restul parametrilor pentru a-i converti într-o cale validă, urmând ca apoi să apeleze funcția corespunzătoare comenzii împreună cu parametrii dați. În interiorul funcției, pentru comenzi precum *cd* ce au nevoie de parametri, se verifică în primul rând dacă aceștia au fost dați, după care sunt executate comenzile folosind funcții disponibile în limbajul *C#*.

După ce o comandă a fost executată, modificările din interfață devin imediat vizibile datorită *Data Binding*-ului. Acestea pot fi modificări în fereastra de afișare a elementelor (ștergerea unui element, adăugarea unui nou element etc.), caz în care este apelată funcția *Refresh* din *DirectoryStructureViewModel* de repopulare a recipientului *Items*, sau modificări în fereastra de afișare a rezultatelor, unde va fi afișat un text ce reprezintă fie rezultatul comenzii executate (numărul de elemente, informații despre un element etc.), fie un text ce va îi va transmite utilizatorului faptul că execuția comenzii nu a putut fi dusă la bun sfârșit. Cum și mărimea textului afișat de această fereastră este legat de o variabilă de tip *int*, aceasta poate fi modificată folosind comanda *fontsize n* (unde *n* este un număr cuprins între 10 și 100).

Capitolul 6

Manual de utilizare

VoiceCommander este o aplicație intuitivă, astfel încât și cei mai neexperimentați utilizatori o pot folosi cu ușurință. După pornirea aplicației, aceștia pot începe sa opereze sistemul de fișiere al calculatorului în funcție de preferințe.

Folosind interfața grafică, utilizatorii pot executa următoarele operații:

- mărirea/micșorarea ferestrelor de afișare prin mișcarea separatorului în stânga sau dreapta;
- schimbarea locației curente prin dublu click pe un director (sau selectând un director și apăsând tasta *Enter* pentru a intra într-un director sau *Back* pentru a ieși din directorul curent);
- aplicarea diferitelor operații asupra fișierelor/directoarelor apăsând click dreapta pe ele;
- schimbarea unor opțiuni din cadrul aplicației apăsând butonul *Options* din bara de meniu;
- citirea unui text ajutător din cadrul aplicației apăsând butonul *Help* din bara de meniu.

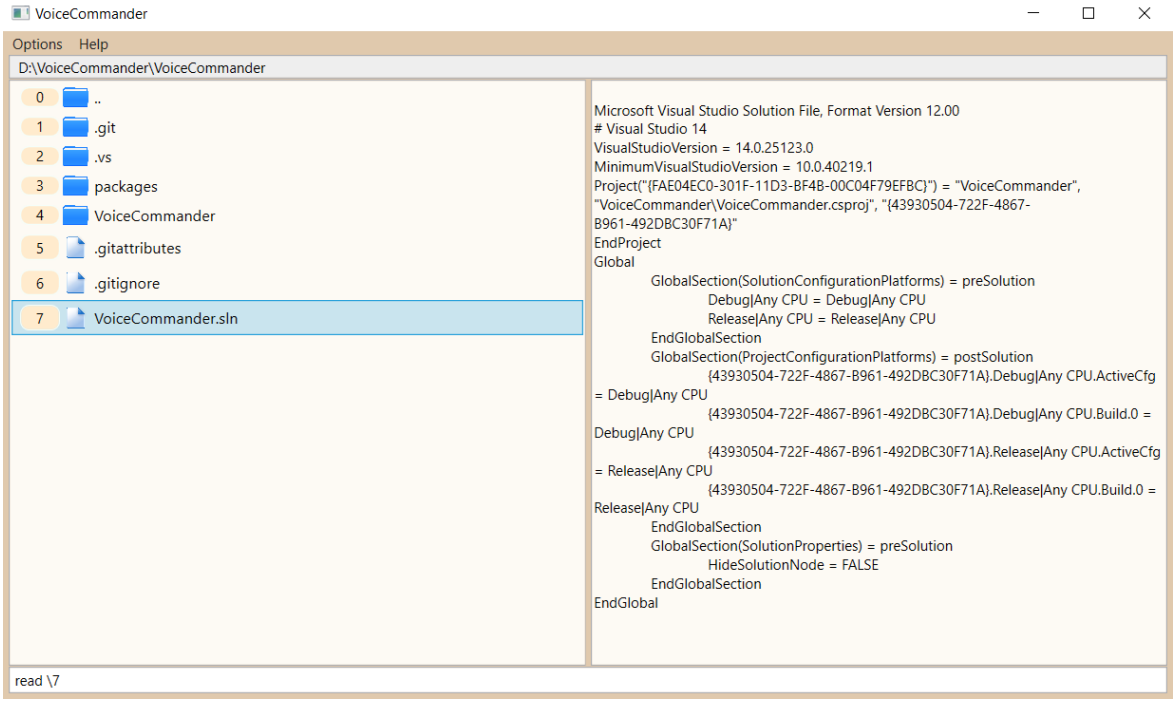
Pentru a utiliza linia de comandă, aplicația pune la dispoziție un număr de comenzi, o parte din acestea fiind:

- *count*, pentru a afișa numărul de elemente din directorul curent;
- *cd* urmat de calea dorită sau indicele directorului, pentru a intra în acel director;
- *read* urmat de numele fișierului, pentru a afișa conținutul (text) al acestuia;
- *mkfile/rmfile* urmat de numele fișierului, pentru a crea/șterge un fișier;
- *mkdir/rmdir* urmat de numele directorului, pentru a crea/șterge un director;
- *mv* urmat de numele fișierului si destinație, pentru a muta un fișier;

Pentru a executa comenzi folosind recunoaștere vocală, utilizatorul va pronunța numele comenzii împreună cu indicele elementului asupra căruia se execută o acțiune (atunci când este cazul). O parte din acestea sunt:

- *back*, pentru a ieși din directorul curent;
- *enter*, pentru a intra într-un director;
- *info*, pentru a afișa detalii despre un element;
- *fontsize* urmat de un număr cuprins între 10 si 200, pentru a schimba mărimea fontului din fereastra de afișare a rezultatelor;
- *clear*, pentru a goli fereastra de afișare a rezultatelor;
- *file* urmat de un șir de caractere, pentru a crea un nou fișier;
- *delete* urmat de un număr ce reprezintă indexul fișierului sau directorului ce se dorește a fi șters (operație ireversibilă).

Următoarea imagine arată ce va afișa aplicația la execuția comenzii *read* (recunoaștere a vorbirii sau linie de comandă) sau prin dublu click pe un fișier:



Pe lângă acestea, aplicația mai dispune de următoarele ”trucuri”:

- apăsarea tastelor *Ctrl* și *F* simultan va adăuga la sfârșitul liniei de comandă calea către directorul curent;
- apăsarea tastelor *Ctrl* și *H* simultan va afișa un scurt text ajutător pentru a-i arăta utilizatorului comenzile disponibile în caz că acesta le-a uitat;
- apăsarea tastelor *Up/Down* va parcurge prin ultimele comenzi executate și le va scrie în linia de comandă;

Concluzii

Aplicația a fost creată cu scopul de a oferi oricărui utilizator posibilitatea de a manipula cu ușurință propriul sistem de fișiere. Pentru a îndeplini acest obiectiv, am decis să implementăm *design pattern*-ul *MVVM* folosindu-ne de *framework*-ul *Fody*, ceea ce s-a dovedit a fi folositor în a construi o aplicație eficientă și modularizată. Având în vedere arhitectura aplicației, tehnologiile folosite și structurile de date implementate ce au fost prezentate în capitolele anterioare, sperăm că obiectivul a fost atins în cele din urmă.

Ne propunem ca pe viitor să extindem funcționalitatea aplicației, astfel încât să oferim utilizatorului mai multă libertate, prin adăugarea de comenzi noi. Totodată, aplicația trebuie să își păstreze modul intuitiv prin care poate fi operată, pentru a fi accesibilă și celui mai nou neexperimentat utilizator.

Bibliografie

- Adam Nathan, *WPF 4.5 Unleashed*, 2013
- Laurent Bugnion, *MVVM - Commands, RelayCommands and EventToCommand*, https://msdn.microsoft.com/en-us/magazine/dn237302.aspx
- Microsoft, *Data Binding Overview*, https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview
- Microsoft, *C# Guide*, https://docs.microsoft.com/en-us/dotnet/csharp/
- Microsoft, *Getting Started (WPF)*, https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/
- Microsoft, *Microsoft Speech Programming Guide*, https://msdn.microsoft.com/en-us/library/hh378466(v=office.14).aspx
- Shamlia, *Understanding the basics of MVVM design pattern*, https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/
- Simon Cropp, *Fody*, https://github.com/Fody/Fody