Факултет за Информатички Науки и Компјутерско Инженерство

Визуелно Програмирање

Проектна задача

BackUpSync

Изработиле:

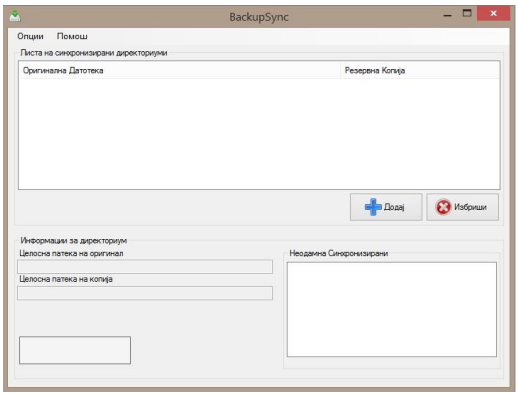
Томе Иванов 121147

Теодор Индов 121164

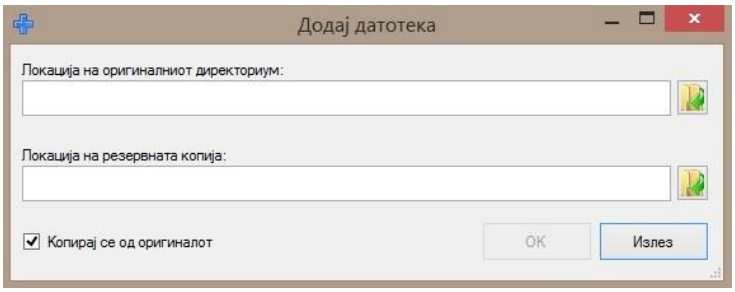
2015г.

1.Објаснување на проблемот (опис на апликација, функции, упатство)

Идејата на проектната задача е десктоп апликацијата BackUpSync која претставува позадинска синхронизација на два или повеќе директоруми зададени од корисникот.



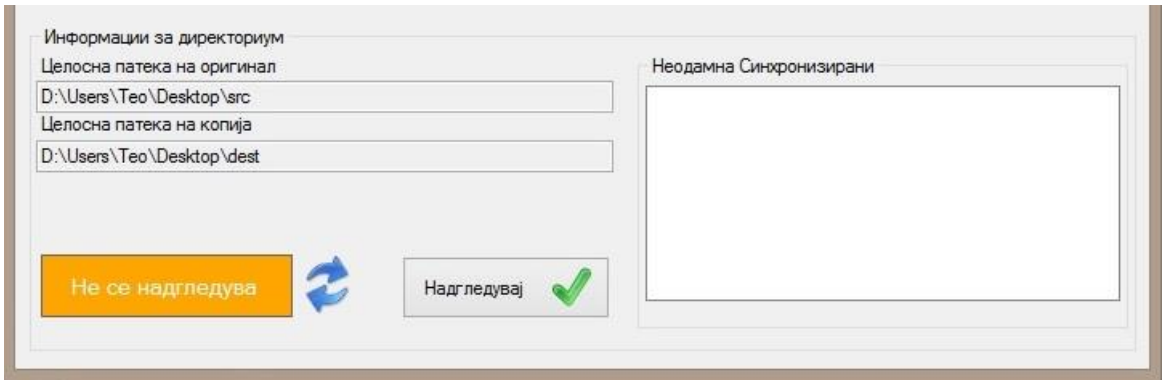
При првиот поглед дадени се листата на синхронизирани директориуми, опциите за додавање и бришење на дадена врска помеѓу два директориума, информации за секоја врска при нејзино селектирање и неодамна синхронизирани фајлови за секоја врска.

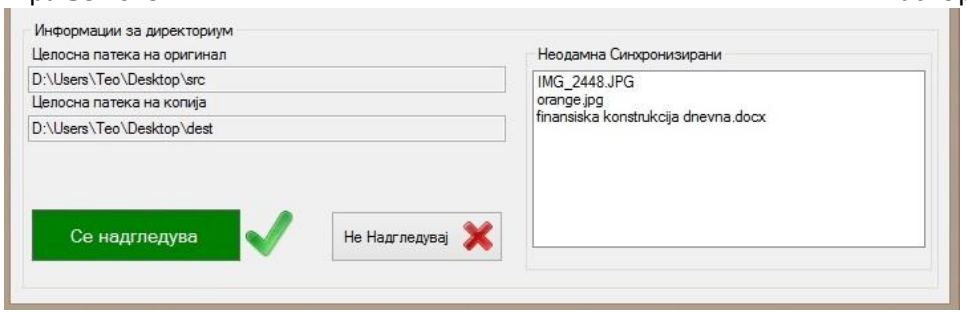


Со кликање на копчето “Додај“, се отвара нова форма која прашува кој фолдер сакате да биде извор (оригинална датотека) и дестинација (резервна датотека). Ги одбирате саканите директориуми и при кликнување на “ОК“ формата се исклучува по што во позадина продолжува синхронизирањето на фајловите.

Ова може да се повторува повеќе пати за повеќе датотеки при што дозволено е синхронизирање на еден извор во повеќе дестинации и повеќе извори во една дестинација. Додека се извршува синхронизацијата при селектирање на врската ќе видите дека има статус “Не се надгледува“ и икона која покажува дека во моментот се извршува синхонизирање и ја има при секоја синхронизација. Кога ќе заврши синхронизацијата, врската почнува да се надгледува. Во информациите имате опција да го исклучите надгледувањето доколку сакате.

Потоа, со секое наредна промена на било каква датотека во изворниот директориум, BackUpSync во позадина ја детектира и автоматски го менува дестинацискиот директориум без вие да правите ништо. Доколку е исклучено набљудувањето, ова нема да се случува. Со минимизарање на апликацијата, таа продолжува да работи и да ги детектира сите промени кај сите врски доколку е вклучено набљудувањето.





2.Опис на решението

- MainForm - класа во која е дефиниран целиот интерфејс и интеракцијата со корисникот. Во неа се дефинирани сите настани на компонентите на апликацијата и менаџирањето со врските помеѓу директориумите.
- AddEntryForm - класа во која е дефинирана формата за додавање на нова врска. Во неа се вршат проверките околу валидноста на избраните директориуми.
- SyncEntry - класа која претставува една единечна врска помеѓу два директориуми. Во неа се сместени сите настани и проверки за промените во директориумите и ги чува сите информации потребни за таа врска.
- FileOperations – класа во која се дефинирани функции за копирање, преместување и бришење на датотеки од дадените директориму или самите директориуми.
- SyncEntry - класа која претставува една единечна врска помеѓу два директориуми. Во неа се сместени сите настани и проверки за промените во директориумите и ги чува сите информации потребни за таа врска.
- AboutForm - форма која се појавува при кликање на “Помош -> За Програмата“ и содржи краток текст кој ја објаснува апликацијата.

3.Опис на класата SyncEntry

Секоја врска помеѓу два директориума претставува објект од класата SyncEntry. При иницијализација на класата се дефинираат двата директориуми, се повикува функција за подесување FileSystemWatcher-от и се почнува со набљудување. Исто така доколку е потребно се копираат сите датотеки од оригиналниот фолдер во дестинацискиот.

```
public SyncEntry(string sourceDir, string destDir, bool shouldCopy){
            this.recentSync = new Queue<string>(10);
            this.sourceDir = sourceDir;
            this.destDir = destDir;
            SetupWatcher();
            firstCopyDone = true;
            if (shouldCopy) {//ako treba da se kopira postavi BackgroundWorker
                firstCopyDone = false;
                worker = new BackgroundWorker();
                worker.DoWork += new DoWorkEventHandler(worker_DoWork);
                worker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
                worker.RunWorkerAsync();
            }
            //ako ne se osteteni pocni so nadgleduvanje
            else if (!DirsDamaged())
                StartWatching();
        }
```

Надгледувањето го врши објектот од класата FileSystemWatcher и кој предизвикува настани кои ги преземаат потребните акции. Подесувањето на FileSystemWatcher-от се одвиве во функцијата SetupWatcher и дополнително се дефинирани функции кои што го запираат или го стартуваат надгледувањето. Дефинирани се и справувачи со настаните за промена, бришење, креирање или промена на името на директориумите и дадотеките кои се набљудуваат.

```
        internal void SetupWatcher() {
            try {
                watcher = new FileSystemWatcher();
                //patekata koja treba da se nadgleduva
                watcher.Path = sourceDir;
                //postavuvanje na filtri za eventi
                watcher.IncludeSubdirectories = true;
                watcher.NotifyFilter = System.IO.NotifyFilters.DirectoryName;
                watcher.NotifyFilter = watcher.NotifyFilter | System.IO.NotifyFilters.FileName;
                watcher.NotifyFilter = watcher.NotifyFilter | System.IO.NotifyFilters.Attributes;
                //postavuvanje na event handlers
                watcher.Changed += new FileSystemEventHandler(eventChangeRaised);
                watcher.Created += new FileSystemEventHandler(eventCreateRaised);
                watcher.Deleted += new FileSystemEventHandler(eventDeleteRaised);
                watcher.Renamed += new RenamedEventHandler(eventRenameRaised);
            } catch (ArgumentException ex) {
                if (watcher != null)
                    watcher.Dispose();
                watcher = null;
            }
        }
        public void StopWatching() {
            if (watcher != null) //onevozmozuvanje na watcherot
                watcher.EnableRaisingEvents = false;
            IsWatched = false;
        }
        public void StartWatching() {
                if (!DirsDamaged() && firstCopyDone) {//ovozmozuvanje na watcherot
                    if (watcher == null)
                        SetupWatcher(); //kreiranje na watcherot
                    watcher.EnableRaisingEvents = true;
                    IsWatched = true;
                }
                else
                    IsWatched = false;
        }
        private void eventCreateRaised(object sender, FileSystemEventArgs e){
            string source = e.FullPath;
            string dest = destDir + source.Replace(sourceDir, @"\");
            IsCopying = true;
            try {
                FileOperations.Copy(source, dest);
            } catch (Exception ex) {
                HasError = true;
                ErrorMsg = ex.Message;
            }
                IsCopying = false;
                addToRecent(e.FullPath.Remove(0, e.FullPath.LastIndexOf("\\") + 1));
        }
```

Дефинирани се и функции за BackgroundWorker-от.

```
        private void worker_DoWork(object sender, DoWorkEventArgs e)
        {
            IsCopying = true;
            FileOperations.DirectoryCopy(sourceDir, destDir, true);
        }

        private void worker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
        {
            IsCopying = false;
            firstCopyDone = true;
            if (!DirsDamaged())
                StartWatching();
        }
```

Исто така има и функција која проверува дали директориумите се оштетени или избришани и се ги поставува соодветните променливи.

```
        public bool DirsDamaged()
        {
            SourceDamaged = !Directory.Exists(SourceDirFullPath);
            DestDamaged = !Directory.Exists(DestDirFullPath);
            bool dirsDamaged = SourceDamaged || DestDamaged;
            if (dirsDamaged && IsWatched) StopWatching();
            return dirsDamaged;
        }
```