# Gestione pacchetti Nuget

**Titolo del progetto:** Gestione pacchetti Nuget
**Alunno:** Alessandro Colugnat
**Classe:** Info I4AC
**Anno scolastico:** 2018/2019
**Docente responsabile:** Ugo Bernasconi

1 Introduzione ... 3
1.1 Informazioni sul progetto ... 3
1.2 Abstract ... 3
1.3 Scopo ... 3
2 Analisi ... 4
2.1 Analisi del dominio ... 4
2.2 Analisi dei costi ... 4
2.3 Analisi e specifica dei requisiti ... 4
2.4 Use case ... 7
2.5 Pianificazione ... 8
2.6 Analisi dei mezzi ... 9
2.6.1 Software ... 9
2.6.2 Hardware ... 9
3 Progettazione ... 9
3.1 Design della interfaccia utente ... 9
3.2 Design delle cartelle ... 11
4 Implementazione ... 12
5 Test ... 20
5.1 Protocollo di test ... 20
5.2 Risultati test ... 21
6 Consuntivo ... 22
7 Conclusioni ... 23
7.1 Sviluppi futuri ... 23
7.2 Considerazioni personali ... 23
8 Bibliografia ... 23
8.1 Sitografia ... 23
9 Allegati ... 23

# 1 Introduzione

## 1.1 Informazioni sul progetto

- Allievo: Alessandro Colugnat
- Docente: Ugo Bernasconi
- Scuola arti e mestieri di Trevano
- Informatica I4AC
- 01.09.2018 – 12.12.2018

## 1.2 Abstract

In the past, the person had to install the Nuget package directly to the project, this is a problem because if you do not have internet you cannot install any Nuget package, my work is to create the program for download the Nuget package from internet, and use in the case the computer is offline.

## 1.3 Scopo

Lo scopo del progetto è quello di creare una interfaccia che permetta di scegliere in quale percorso si deve scaricare oppure aggiornare i pacchetti Nuget, oppure se si vuole creare un PDF con tutte le informazioni dei pacchetti precedentemente scaricati dal programma, oppure eliminare il pacchetto con tutte le dipendenze create insieme al pacchetto Nuget nel momento della installazione.

# 2 Analisi

## 2.1 Analisi del dominio

Il problema dei pacchetti Nuget scaricati su Visual Studio a scuola e che non possono essere scaricati per via dei permessi del proxy forniti dalla scuola, questo progetto permette di scaricare i vari pacchetti Nuget senza blocchi causati dal proxy, a differenza del programma di Visual Studio, questo programma permette di scaricare in un percorso a scelta il pacchetto che si vuole e infine si può installare su Visual Studio. Questo prodotto non esiste in vendita su internet ma ci sono vari siti che mostrano il codice per scaricare i vari pacchetti Nuget. Su Visual studio 2017 si possono installare ma non scaricare i pacchetti Nuget e viene bloccato per quando c'è il proxy.

## 2.2 Analisi dei costi

| Componenti | Prezzo |
|---|---|
| 1 Lavoratore | 62 CHF/ ora |

Prezzo totale per 150 ore: 9300 CHF

## 2.3 Analisi e specifica dei requisiti

Questa documentazione serve a introdurre gli utenti alla creazione del programma insieme al suo scopo, il prodotto serve a scaricare, eliminare e aggiornare i pacchetti Nuget che serviranno da estensione per Visual Studio 2017, il prodotto è un programma eseguibile che si può usare quando c'è una connessione internet per scaricare i pacchetti Nuget, l'utente può cercare il prodotto con la barra di ricerca e scaricare il pacchetto che verrà selezionato dall'utente, si può scegliere il percorso dove mettere il pacchetto scaricato. Per la sicurezza del download è affidata al sito ufficiale di Nuget.

| ID: REQ-001 | |
|---|---|
| **Nome** | Creazione di una interfaccia grafica |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Devono essere presenti 4 pagine |
| **002** | La prima pagina deve essere per scaricare i pacchetti |
| **003** | La seconda deve essere per le impostazioni del programma |
| **004** | La terza pagina deve essere per la visualizzazione dei pacchetti Nuget già installati |
| **005** | La quarta pagina deve essere per le informazioni riguardanti il copyright |

| ID: REQ-002 | |
|---|---|
| **Nome** | Inserire dentro la barra di ricerca |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Prendere i dati dentro la barra di ricerca |
| **002** | Controllare sul database di Nuget se i pacchetti contengono quella scritta |
| **003** | Inserire dentro la ListBox i primi 14 pacchetti trovati nella ricerca |

| ID: REQ-003 | |
|---|---|
| **Nome** | Salvare il pacchetto |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Cliccare un pacchetto all'interno della ListBox |
| **002** | Visualizzare nella casella di fianco tutti i dati del pacchetto selezionato |
| **003** | Cliccare su tasto Salva |

| ID: REQ-004 | |
|---|---|
| **Nome** | Cambiare percorso |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Andare nella finestra Settings |
| **002** | Cliccare il bottone Browse… |
| **003** | Scegliere il percorso desiderato |

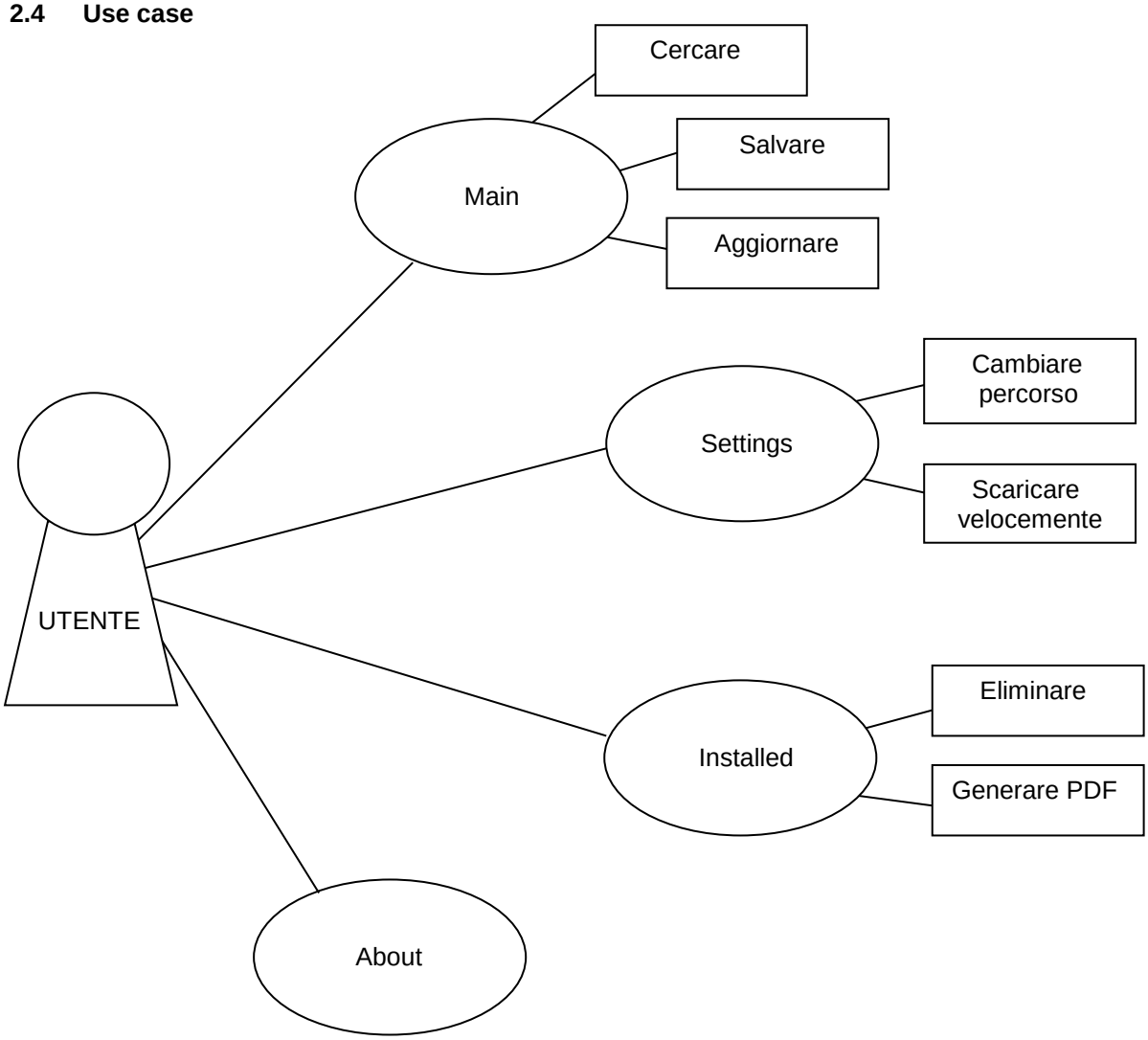
| ID: REQ-005 | |
|---|---|
| **Nome** | Eliminare pacchetto Nuget |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Andare nella finestra Installed |
| **002** | Selezionare un pacchetto Nuget |
| **003** | Cliccare il bottone Elimina |

| ID: REQ-006 | |
|---|---|
| **Nome** | Creare file PDF |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Nessuna nota |
| **Sotto requisiti** | |
| **001** | Andare nella finestra Installed |
| **002** | Cliccare il tasto Generate PDF |

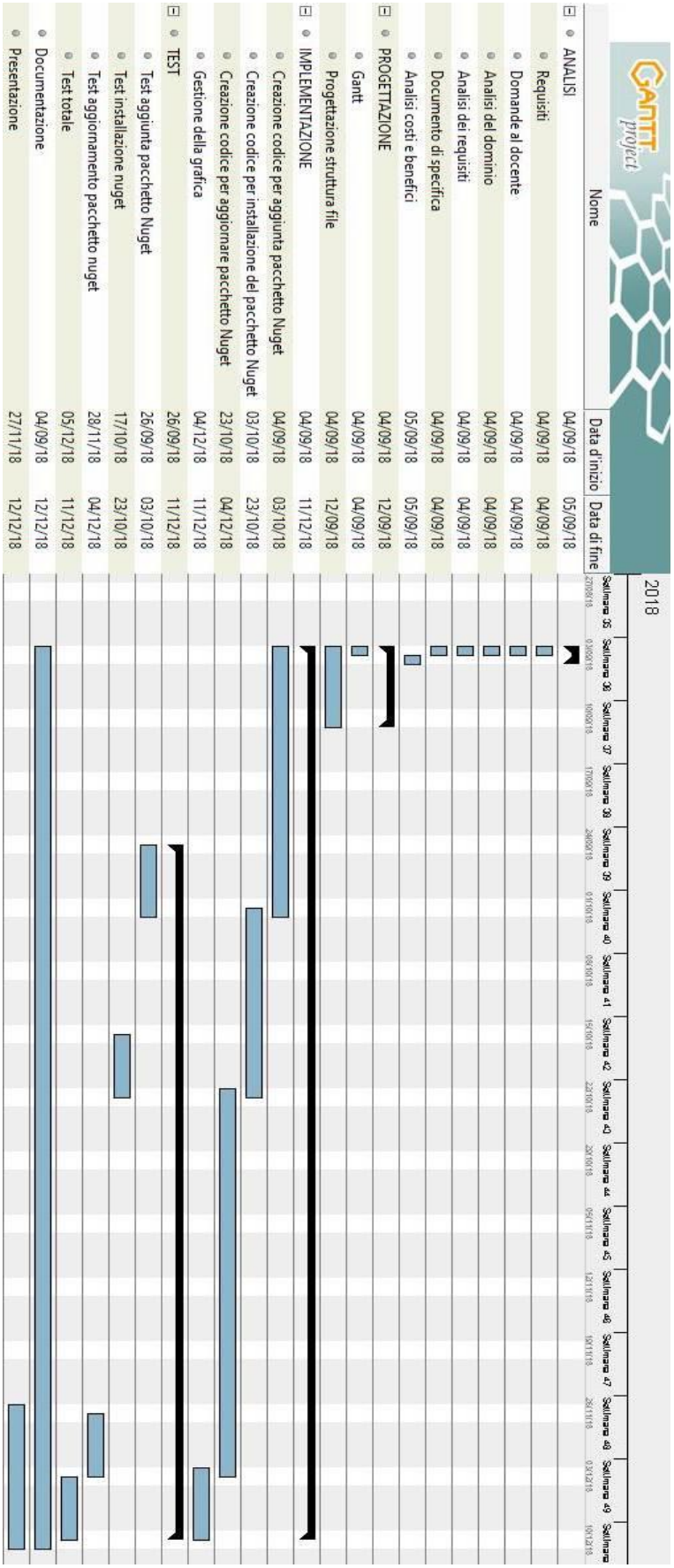
**Spiegazione elementi tabella dei requisiti:**

**ID**: identificativo univoco del requisito
**Nome**: breve descrizione del requisito
**Priorità**: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.
**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.
Sulla documentazione apparirà solamente l'ultima versione, mentre le vecchie dovranno essere inserite nei diari.
**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.
**Sotto requisiti**: elementi che compongono il requisito.

### 2.4 Use case


Cercare
Salvare
Main
Aggiornare
Cambiare percorso
Settings
Scaricare velocemente
UTENTE
Eliminare
Installed
Generare PDF
About

## 2.5 Pianificazione



### 2.6 Analisi dei mezzi

#### 2.6.1 Software

- Visual studio 2017

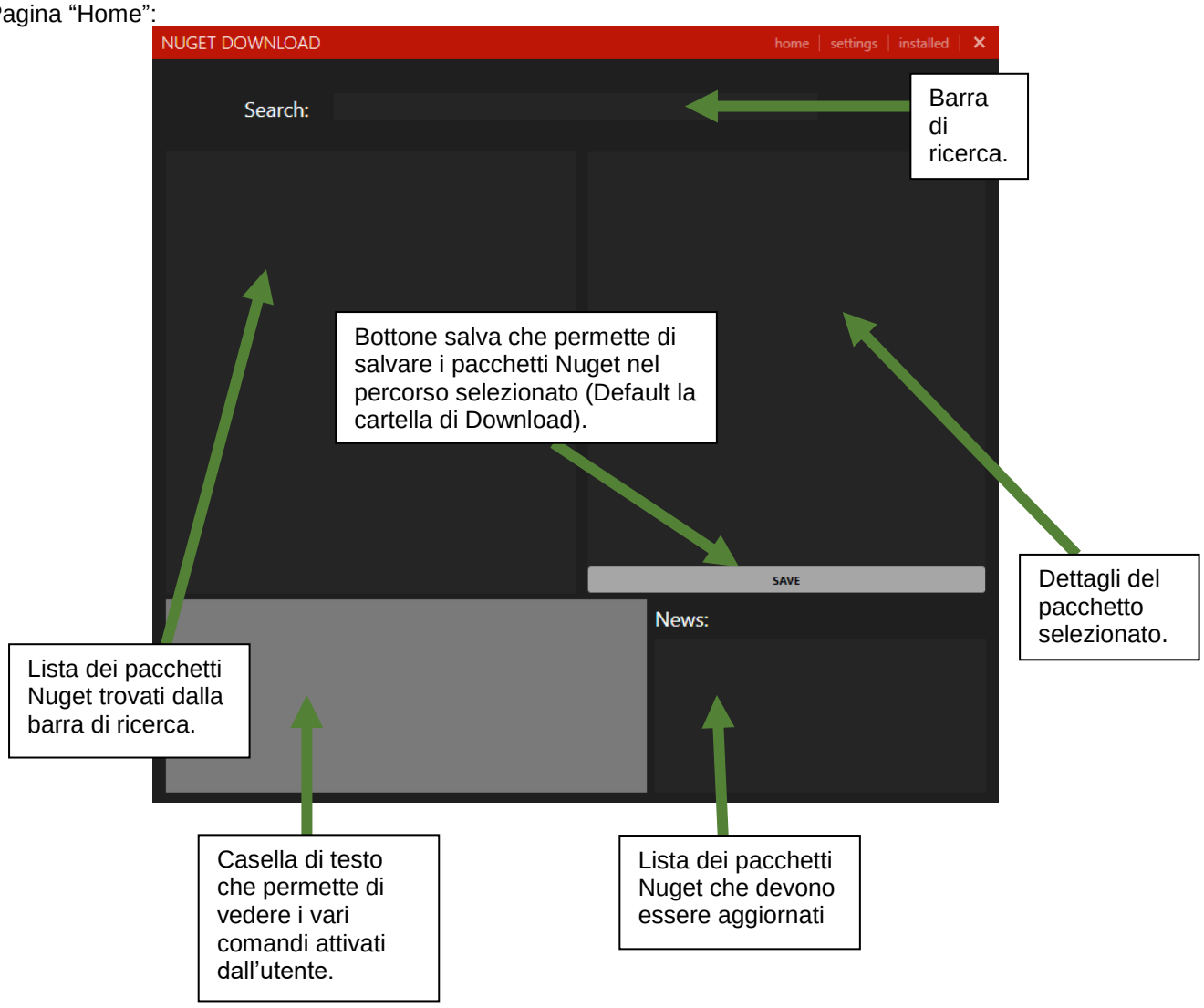
#### 2.6.2 Hardware

- 1 Computer

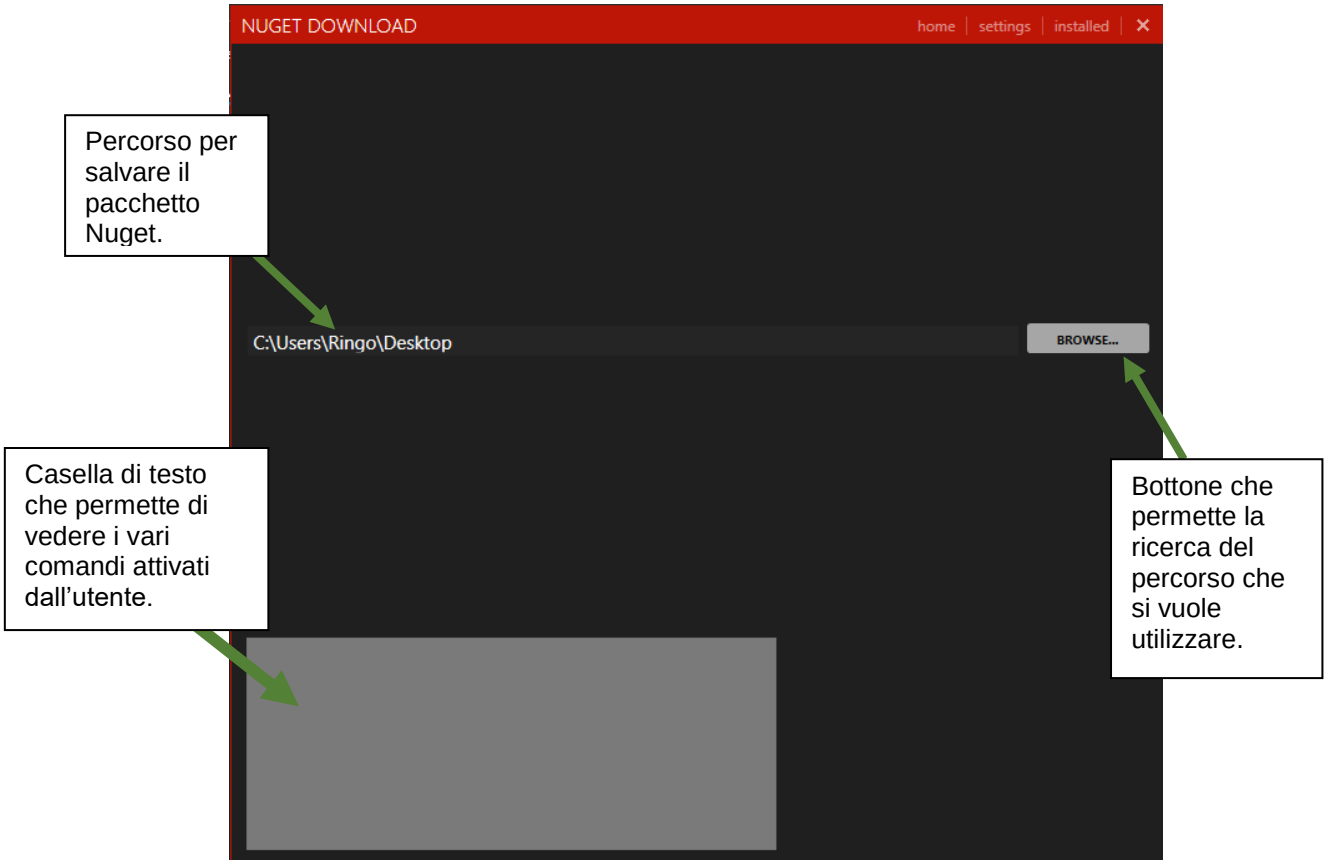
## 3 Progettazione

### 3.1 Design della interfaccia utente

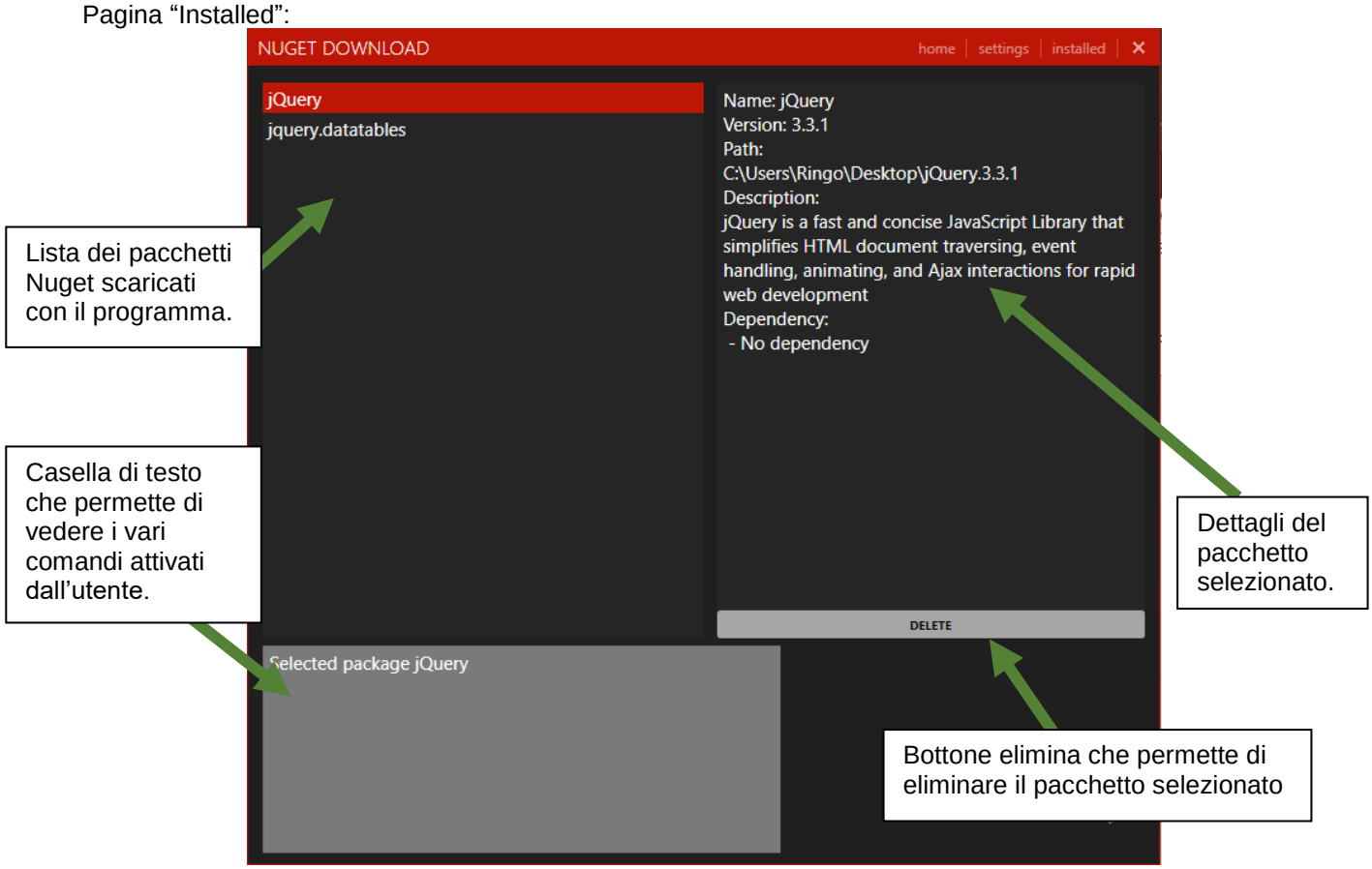
Pagina “Home”:




Pagina “Settings”:

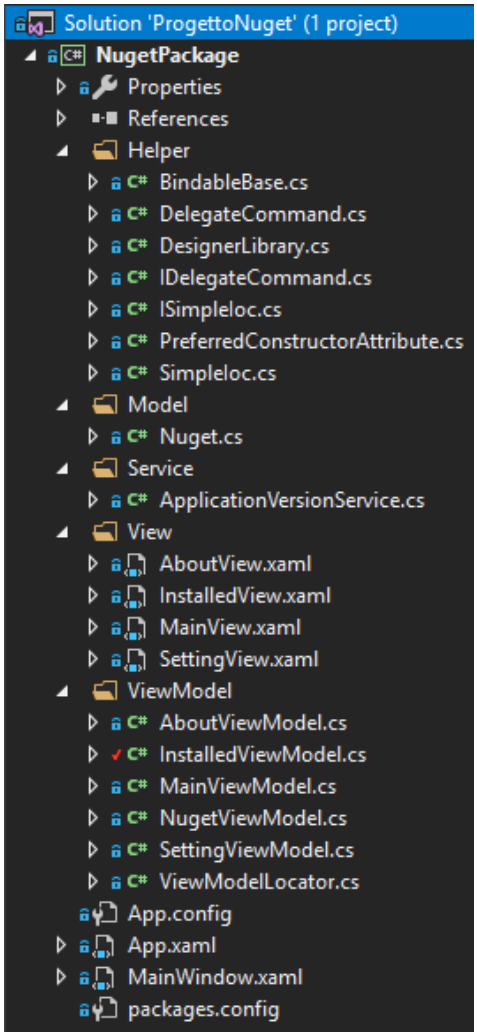



Pagina "Installed":



### 3.2 Design delle cartelle

Il codice che permette il funzionamento del programma si trova nella cartella ViewModel e invece la parte grafica si trova nella cartella View, dentro la cartella Model si trova il file con tutti gli attributi che servono per far comunicare le varie classi tra di loro. Dentro la cartella Helper ci sono tutti I file che permettono di utlizzare più finestre nella stessa finestra.




Per il completo funzionamento sono stati importati dei pacchetti Nuget:

- Nuget.Core 2.14.0 (Microsoft.Web.Xdt 2.1.2):
  Servono a far funzionare il codice che comunica con il sito dei pacchetti Nuget.
- System.Windows.Interactivity.WPF 2.0.202525:
  Serve a far funzionare gli eventi che riguardano i componenti della parte grafica.
- WPFFolderBrowser 1.0.2:
  Viene utilizzato quando si deve aprire una finestra con il percorso da scegliere.
- MahApps.Metro 1.6.5 (ControlzEx 3.0.2.4):
  Viene utilizzato per la finestra principale, questo pacchetto permette di utilizzare una grafica diversa invece del solito standard Windows.
- CommonServiceLocation 1.0.0:
  Questo pacchetto viene utilizzato per I file che si trovano dentro la cartella Helper, permette il funzionamento della communicazione tra I vari programmi.
- PDFsharp 1.32.3057:
  Permette di utilizzare il codice che serve per creare un file PDF insieme a tutta la struttura che si trova all'interno.

## 4 Implementazione

In questo capitolo si mostreranno alcuni pezzi di codice che sono stati importanti per la realizzazione del progetto:
Per lo svolgimento del progetto ho utilizzato 16 attributi che permettono il funzionamento di tutto il codice:

```
public string Directory { get; set; }
```

L'Attributo **Directory** serve per contenere il percorso in cui si salverà il pacchetto Nuget con le relative dipendenze, nel caso in cui l'attributo **Directory** è vuoto verrà utilizzato un percorso di default che punta alla cartella di downloads, ogni volta che si apre il programma verrà mostrato l'ultimo percorso scelto, il tutto verrà salvato dentro un file di log che contiene la variabile per sapere l'ultimo percorso, la variabile si chiama **logFilePath.txt**.

```
public string NamePackage { get; set; }
```

L'attributo **NamePackage** serve per il nome del pacchetto Nuget che è stato scelto e ha anche la funzione di ID del pacchetto che si deve cercare per scaricare il pacchetto Nuget, la variabile cambia ogni volta che viene scelto un nuovo pacchetto Nuget dalla **ListBox**.

```
public string VersionPackage { get; set; }
```

L'attributo **VersionPackage** serve alla versione del pacchetto che si deve scaricare, nel programma verrà sempre presa la ultima versione, nel caso si deve aggiornare il pacchetto verrà confrontata la vecchia versione con la nuova versione che si trova su internet.

```
public string VersionNewsPackage { get; set; }
```

L'attributo **VersionNewsPackge** serve in caso sia stato selezionato un aggiornamento, perché quando si seleziona un elemento da aggiornare viene mostrato quale versione è stata installata e affianco viene mostrata l'ultima versione da installare. Per fare ciò serve questo attributo **VersionNewsPackage** che tiene salvata la versione precedente del pacchetto da aggiornare.

```
public ObservableCollection<string> ResultSearch { get; set; }
```

L'attributo **ResultSearch** permette di raccogliere tutti i nomi dei pacchetti che sono stati trovati utilizzando la barra di ricerca, il numero massimo di pacchetti che si possono vedere sono 14 perché se si caricano tutti i pacchetti si rischierebbe di aspettare troppo per vedere i primi risultati della ricerca. Invece utilizzando un numero massimo di nomi da mostrare si può velocizzare il processo di caricamento.

```
public ObservableCollection<string> ResultSearchNews { get; set; }
```

L'attributo **ResultSearchNews** ha la stessa funzione del **ResultSearch** ma carica nel **ListBox delle News** i pacchetti che hanno bisogno di un aggiornamento.

```
public string StartSearch { get; set; }
```

L'attributo **StartSearch** contiene una stringa che permette di cercare i pacchetti con quella determinata parola, la variabile cambia ogni volta che un tasto della tastiera viene premuto, quando inizia l'evento la variabile **StartSearch** cambia valore e fa una ricerca per vedere quali pacchetti si trovano, appena si trovano dei pacchetti vengono messi nella variabile **ResultSearch**.

```
public string ResultPackage { get; set; }
```

L'attributo **ResultPackage** serve per avere tutte le informazioni riguardanti al pacchetto che è stato scelto, dentro alla variabile vengono messe le informazioni riguardanti il nome del pacchetto, la versione, l la descrizione e anche le varie dipendenze che verranno scaricate insieme al pacchetto con le varie versioni accettabili e infine verrà tutto messo nella casella di testo affianco alla **ListBox**.

```
public string ResultLog { get; set; }
```

L'attributo **ResultLog** serve per informare gli utenti su cosa succede nel programma quando si fa iniziare un evento, per esempio nel caso in cui si sceglie un nuovo percorso per salvare il file, verrà scritto nella casella di testo, che è stato selezionato un nuovo percorso, così facendo l'utente sa cosa stia accadendo e ha la sicurezza di aver fatto quella azione.

```
public string DescriptionPackage { get; set; }
```

L'attributo **DescriptionPackage** viene utilizzato per quando si deve creare il file **PDF** che contiene tutte le informazioni riguardanti il pacchetto scaricato.

```
public string DependencyPackage { get; set; }
```

L'attributo **DependenyPackage** ha la stessa funzione del **DescriptionPackage**, viene utilizzato solo ciò che riguarda il file **PDF**.

```
public static ObservableCollection<string> InstalledPackage { get; set; }
```

In questo attributo vengono messi tutti i pacchetti che sono stati installati con il programma, alla fine verrà utilizzato per far visualizzare dentro la **ListBox** tutti i pacchetti installati per decidere se eliminarli oppure creare un **PDF**.

```
public static string NameInstalledPackage { get; set; }
```

Contiene il nome del pacchetto selezionato tra quelli installati.

```
public static string ResultInstalledPackage { get; set; }
```
Contiene tutte le informazioni riguardanti il pacchetto selezionato nella zona di quelli installati, fa vedere la descrizione del pacchetto con la versione scaricati e non l'ultima disponibile.
```
public static string PathInstalledPackage { get; set; }
```
In questo attributo viene salvato il percorso in qui si è salvato il pacchetto, per fare in modo che quando venga eliminato si possa eliminare il pacchetto nell'esatta posizioni, non gestisce il fatto di un pacchetto che viene spostato.
```
public static bool IsFastDownloader { get; set; }
```
L'attributo **IsFastDownloader** controlla che l'impostazione per scaricare il pacchetto con il doppio click sia atttivo, nel caso in cui è **false** significa che è disattivata, invece con **true** significa che è attiva.
Nel programma ci sono 12 metodi che permettono il funzionamento del programma, tutti i metodi funzionano a coppia perché si usa il file **DelegateCommand.cs**, viene utilizzato perché un metodo permette di controllare se è possibile utilizzare l'altro metodo, di solito per capire chi è il controllore si mette in inglese davanti al metodo la parola "Can" e invece per quello che fa tutto si mette "On", questi comandi si trovano all'interno del file **NugetViewModel.cs**:
```
SaveCommand = new DelegateCommand(OnSave, CanSave);
```
Il metodo **CanSave** controlla se il file **logFilePath.txt** è stato creato, per fare in modo che quando si salva si ha la traccia del pacchetto salvato, perché nel caso in cui c'è un aggiornamento si può controllare nel file quale versione è già stata installata. Il metodo **OnSave** serve a salvare il pacchetto selezionato nel percorso scelto in precedenza e modificare il pacchetto in caso sia avvenuto un aggiornamento oppure aggiungere una riga con la informazione del pacchetto installato.
```
SaveFastCommand = new DelegateCommand(OnSaveFast, CanSaveFast);
```
Il metodo **CanSaveFast** controlla che il file che contiene la informazione riguardante lo stato del bottone, sia esistente, nel caso non esiste verrà creato il file insieme al valore di default che è false.
Invece il metodo **OnSaveFast** contiene il metodo **OnSave** che viene richiamato per salvare il file, **SaveFastCommand** si attiva con il doppio click su un pacchetto nella **ListBox**.
```
ShowCommand = new DelegateCommand(OnShow, CanShow);
```
Il metodo **CanShow** controlla che il pacchetto è stato selezionato, perché nel caso in cui si clicca sulla **ListBox** ma non ci sono pacchetti da selezionare verrà trasmesso l'errore nel casella di log, che l'utente non ha selezionato nessun percorso. Il metodo **OnShow** fa mostrare tutti i dettagli importanti del pacchetto selezionato, verranno mostrati i dati che riguardano il titolo, la versione, la descrizione e le varie dipendenze che occorrono al pacchetto Nuget.
```
SearchCommand = new DelegateCommand(OnSearch, CanSearch);
```
Il metodo **CanSearch** controlla se la esiste una connessione a internet perché se no, non si possono cercare i vari pacchetti. Il metodo **OnSearch** fa una connessione con il server dei pacchetti Nuget per fare un controllo su quali pacchetti hanno l'informazione richiesta dall'utente, quando vengono presi i dati si mostra nella **ListBox** i primi 14 pacchetti perché se no il programma ci mette troppo tempo a stabilire una connessione con tutti i dati del database.
```
SearchNewsCommand = new DelegateCommand(OnSearchNews, CanSearchNews);
```
Il metodo **CanSearchNews** viene eseguito appena si apre il programma, permette di verificare se una connessione a internet è stabilita, nel caso contrario non si eseguirà il metodo **OnSearchNews**, viene anche controllato se il file **logFileNews.txt** è stato creato, se non esiste si crea. Il metodo **OnSearchNews** fa la ricerca dei pacchetti che devono essere aggiornati, nel caso in cui deve essere aggiornato verrà mostrato nella **ListBox** delle **News**. Nel caso in cui non c'è un pacchetto da aggiornare non verrà mostrato niente.
```
CheckDeletedCommand = new DelegateCommand(OnCheckDeleted, CanCheckDeleted);
```
Il metodo **CanCheckDeleted** controlla che il file **logFIleNews.txt** si stato creato e nel caso in cui non esistesse verrebbe creato, il metodo **OnCheckDeleted** controlla la cartella in cui è installato il pacchetto, nel caso in cui il pacchetto fosse stato eliminato o spostato senza utilizzo del programma, appena si apre il programma verranno eliminati dal file i pacchetti che non esisterebbero più, per fare in modo che quando si prova ha eliminare il pacchetto con il programma non riceve nessun errore di ricerca del file. Questi comandi successivi si trovano nel file **SettingViewModel.cs**:
```
BrowseCommand = new DelegateCommand(OnBrowse, CanBrowse);
```
Il metodo **CanBrowse** non fa nessun controllo perché non è possibile avere degli errori durante l'utilizzo del metodo **OnBrowse** che serve ad aprire una finestra di dialogo per scegliere quale percorso mettere nella casella di testo e cambiando la variabile dentro il file **logFilePath.txt**, quando si sceglie un percorso viene aggiunta al percorso una cartella di nome **NugetPackage** in cui verranno salvati tutti i pacchetti.
```
CheckCommand = new DelegateCommand(OnCheck, CanCheck);
```
Il metodo **CanCheck** controlla che il file che contiene le informazioni riguardanti lo stato del **CheckBox**, nel caso in cui non esistesse verrebbe creato e avrebbe il valore di default che è false, invece il metodo **OnCheck** contiene la modifica del file sul cambiamento di stato, se è false diventa **true** e viceversa.

Questi comandi successivi si trovano nel file **InstalledViewModel.cs**:

```
ShowInstalledCommand = new DelegateCommand(OnShowInstalled, CanShowInstalled);
```

Il metodo **CanShowInstalled** controlla che un pacchetto sia stato selezionato dalla **ListBox** nel caso in cui non fosse selezionato nessun pacchetto non si potrà continuare con il codice e verrà mostrato nella casella di testo che nessun pacchetto è stato selezionato, invece il metodo **OnShowInstalled** fa visualizzare tutti i dettagli riguardanti il pacchetto selezionato, ci sono i dettagli riguardanti il nome, la versione, il percorso, la descrizione e le dipendenze.

```
DeleteCommand = new DelegateCommand(OnDelete, CanDelete);
```

Il metodo **CanDelete** non fa nessun controllo, invece il metodo **OnDelete** permette di cancellare in modo sicuro il pacchetto insieme a tutte le dipendenze, nel caso in cui le dipendenze servono ad un altro pacchetto, quella dipendenza non verrà eliminata, nel caso in cui la dipendenza è un pacchetto scaricato precedentemente questa non verrà eliminata. Appena il pacchetto viene eliminato dentro il file **logFileNews.txt** viene eliminato la stringa in cui si trova il pacchetto. Infine la pagina verrà aggiornata con tutti i pacchetti installati senza quello eliminato.

```
SearchInstalledCommand = new DelegateCommand(OnSearchInstalled, CanSearchInstalled);
```

Il metodo **CanSearchInstalled** controlla che il file **logFileNews.txt** esista e controlla anche la connessione a internet per trovare i dettagli per i pacchetti installati, il metodo **OnSearchInstalled** cerca tutti i pacchetti che si trovano dentro il file **logFileNews.txt**, per fare in modo che l'utente può vedere quali pacchetti sono stati installati nel computer, controllando tutti i percorsi in cui si trovano i vari pacchetti.

```
GenerateCommand = new DelegateCommand(OnGenerate, CanGenerate);
```

Il metodo **CanGenerate** non fa nessun controllo perché per generare un file **PDF** non si ha bisogno di nessun controllo in precedenza anche se non ci sono pacchetti installati, in questo verrà creato un file vuoto con solamente in titolo. Per generare il file **PDF** viene fatto nel metodo **OnGenerate**, che permette di prendere tutti i pacchetti e controllare uno alla volta i vari pacchetti e prendere le informazioni da mettere dentro il **PDF**. Nel file **PDF** ci sono queste informazioni, il titolo, la versione e le dipendenze.
Nei prossimi pezzi di codici che verranno mostrati qui sotto ci saranno tutte le informazioni sui codici più utili e complicati utilizzati nel programma:

```
IPackageRepository repo =
PackageRepositoryFactory.Default.CreateRepository("https://packages.nuget.org/api/v2");
```

Questo pezzo di codice serve a connettersi al database di Nuget, inizialmente dava problemi perché il proxy della scuola CPT blocca il passaggio dati dei pacchetti Nuget, questo codice ha tutti i dati che servono ad un pacchetto Nuget.

```
String result = repo.Search(packageID, false).First()
```

Questo metodo insieme al codice di sopra serve a cercare tutti i pacchetti con un collegamento all'Id della variabile **packageID**, invece il **false** serve a specificare se si vogliono prendere anche i pacchetti prerilasciati, non ho messo a **true** perché alcune dipendenze dei vari pacchetti non sono possibili da scaricare, invece il commando **First()** permette di prendere il primo dato che si trova nel metodo prima. Da utilizzare quando l'Id del pacchetto è esattamente quello che si cercava.
Dopo che si è usato il comando **Search()** si possono prendere tutte le informazioni relative a un pacchetto, per esempio si può prendere la versione:

```
result.Version.toString()
```

Invece se si vuole prendere tutte le dipendenze che serve al pacchetto Nuget si deve fare un comando più complesso:

```
FrameworkName frameworkName = new FrameworkName("Anything", new Version("3.5"));
result.GetCompatiblePackageDependencies(frameworkName).Select(x => x));
```

Per trovare tutte le dipendenze che richiede un pacchetto Nuget si deve creare per prima cosa una variabile di tipo **FrameworkName** che serve a scegliere con quale tipo di versione si deve fare la ricerca delle dipendenze, per trovare i pacchetti si deve usare il metodo **GetCompatiblePackageDependencies()** che serve a trovare tutte le dipendenze compatibili per il pacchetto che si sta cercando, infine si deve fare un **Select()** per mettere tutti i valori in una lista.

```
PackageManager packageManager = new PackageManager(repo, Directory);
packageManager.InstallPackage(NamePackage, SemanticVersion.Parse(VersionPackage));
```

Questo codice serve a scaricare e estrarre il pacchetto Nuget appena scaricato, la prima linea di codice serve per creare un collegamento per scaricare il pacchetto Nuget, vengono richieste due valori, il primo serve da collegamento al database invece il secondo serve a scegliere in quale percorso mettere il pacchetto Nuget. Il secondo pezzo di codice serve a installare il pacchetto, il primo campo serve a scegliere quale pacchetto scaricare, invece il secondo pacchetto serve a scegliere quale versione si vuole scaricare per il pacchetto, si deve convertire la variabile in **SemanticVersion** perché è richiesto dal metodo **InstallPackage()**.

Dopo che si è installato il pacchetto si può andare a creare il file **PDF** che contiene tutti i dati relativi ai pacchetti scaricati dal programma nel caso che si vuole un riassunto di tutto ciò che è stato installato:

```
private void onGenerate(object obj)
{
	PdfDocument pdf = new PdfDocument();
	pdf.Info.Title = "Installed package";
	PdfPage pdfPage = pdf.AddPage();
	XGraphics graph = XGraphics.FromPdfPage(pdfPage);
	XFont font = new XFont("Arial", 11, XFontStyle.Regular);
	XFont fontTitle = new XFont("Arial", 12, XFontStyle.Bold);
	XFont fontStartTitle = new XFont("Arial", 18, XFontStyle.Bold);
	int newLine = 40;
	graph.DrawString("Nuget package", fontStartTitle, XBrushes.Black, new
XRect(pdfPage.Width.Point/2 - 80, newLine, pdfPage.Width.Point - 20, pdfPage.Height.Point -
20), XStringFormats.TopLeft);
	newLine += 60;
	foreach (string newsName in fileNewsContent)
	{
		string[] getVersion = newsName.Split(':');
		string[] nameCurrentId = getVersion[1].Split('\\');
		string packageID = nameCurrentId[nameCurrentId.Length - 1];
		IPackageRepository repo =
PackageRepositoryFactory.Default.CreateRepository("https://packages.nuget.org/api/v2");
		IPackage package = repo.Search(packageID, false).First();
		graph.DrawString("Title:", fontTitle, XBrushes.Black, new XRect(40, newLine,
pdfPage.Width.Point - 20, pdfPage.Height.Point - 20), XStringFormats.TopLeft);
		graph.DrawString(packageID, font, XBrushes.Black, new XRect(140, newLine,
pdfPage.Width.Point - 20, pdfPage.Height.Point - 20), XStringFormats.TopLeft);
		[...]
	}
}
```

Questo metodo **createPDF()** viene utilizzato quando si clicca sul bottone che permette di generare il file **PDF**, il metodo permette di creare un file all'interno della cartella che è stata scelta nei settings con dentro le seguenti informazioni: titolo, versione, descrizione e dipendenze per tutti i pacchetti precedentemente installati. Il codice mostrato qui sopra fa vedere cosa serve per creare un file **PDF**, per usarlo si è usata la libreria **PDFSharp**. L'oggetto **PdfDocument()** permette di inizializzare il documento **PDF** vuoto, per aggiungere la prima pagina si deve richiamare il metodo **AddPage()** che si può utilizzare quante volte si vuole, ma si deve utilizzare obbligatoriamente all'inizio per aggiungere la prima pagina. Si deve anche aggiungere il titolo del documento. Per la parte grafica si deve utilizzare la classe **XGraphics** che permette di scrivere all'interno del file oppure disegnarci, per scrivere all'interno del file serve un font per la scrittura oppure un **font** per i sottotitolo oppure un altro font per il titolo principale, per scrivere dentro un file **PDF** si deve richiamare il comando **DrawString()** che richiede vari campi che sono: stringa da scrivere, il font che deve usare, il colore della scritta, grandezza del campo in cui si vuole mettere la scritta e infine quale direzione scrivere se da in alto a sinistra oppure a destra in basso, il problema di questi comandi e che quando si deve mettere una stringa molto lunga, il programma non farà andare a capo automaticamente, quindi per farlo funzionare si deve dividere la stringa e in mezzo si deve creare una nuova stringa che permette di andare a capo. Per fare in modo che vengono stampati tutti i pacchetti installati si deve far passare un ciclo **foreach** che permette di controllare pacchetto per pacchetto le varie informazioni che verranno aggiunte al file, nel caso non ci siano pacchetti verrà creato un file nuovo. Questo è il codice che permette la creazione del file **PDF**:

```
string pdfFilename = "InstalledNuget.pdf";
try {
	pdf.Save(Directory + "\\" + pdfFilename);
	ResultLog += "Generated PDF file in the path " + Directory + "\n";
} catch (IOException) {
	ResultLog += "PDF already in use\n";
}
```

Il codice mostrato qui sopra fa un controllo, nel caso in cui fosse già creato e aperto verrà fuori l'errore che non si può scrivere perché è già in uso, nel caso contrario si utilizza il metodo **Save()** che salva il file nel percorso desiderato.

In questo pezzo di codice serve a aprire una finestra di dialogo dove si può scegliere quale percorso si vuole utilizzare:

```
FolderBrowserDialog folderDialog = new FolderBrowserDialog
{
       SelectedPath = "C:\\"
};
DialogResult result = folderDialog.ShowDialog();
if (result.ToString() == "OK")
{
       Directory = folderDialog.SelectedPath;
       File.WriteAllText("logFilePath.txt", Directory + "\\NugetPackage");
       ResultLog += "Selected path " + Directory + "\n";
       Directory += "\\NugetPackage";
}
```

Le prime tre righe servono a inizializzare un oggetto che contiene la finestra per la scelta del percorso, nella quarta riga serve a far mostrare la finestra di dialogo con il metodo **ShowDialog()**, il metodo ritorna una stringa che permette di vedere se è andato tutto a buon fine, nel **if** si controlla che l'utente abbia premuto il tasto OK che si trova nella finestra di dialogo per confermare la scelta appena effettuata, nell'ultimo pezzo di codice viene mostrata la variabile **Directory** che salva il nuovo valore preso dalla finestra di dialogo precedentemente utilizzata, nel codice che viene dopo serve a cambiare il percorso all'interno del file per tenerlo salvato nei casi in cui si apra il programma in futuro viene tenuto salvo il percorso, dopo viene scritto all'interno della casella di testo per far capire all'utente che tutto è andato a buon fine e l'ultima riga aggiunge al nuovo percorso una cartella che permette di tenere tutti pacchetti Nuget sotto un'unica cartella chiamata **NugetPackage**.
Nel programma viene utilizzato un percorso di default, che si usa nel caso nessun percorso è stato scelto:

```
string defaultPath
 = Path.GetDirectoryName(Environment.GetFolderPath(Environment.SpecialFolder.Personal));
defaultPath = Path.Combine(defaultPath, "Downloads");
```

Questo pezzo di codice serve a creare una stringa con dentro il percorso della cartella di download, per fare ciò bisogna utilizzare la classe **System.IO.Path** che contiene un metodo che si chiama **GetDirectoryName()** che permette di cercare il percorso di qualsiasi utente anche senza l'utilizzo del nome dell'account, dentro il metodo serve una stringa che contiene il percorso da cercare, con la variabile **Environment** si può prendere il percorso con dentro tutte le cartelle personali, per esempio la cartella di downloads, documents oppure desktop, con la variabile **Environment.SpecialFolder.Personal** si può prendere il percorso utente (es.C:\Users\Ringo).
Con il codice nella seconda riga si può utilizzare il metodo **Combine()** che permette di combinare due variabili che fungono da percorso nel codice si può vedere che si fa la combinazione tra il percorso di default insieme alla cartella **Downloads**, in fine dentro la variabile **defaultPath** si troverà il percorso della cartella di **Downloads**.

Per gestire un aggiornamento si devono fare dei controlli nel file di testo **logFileNews.txt**:

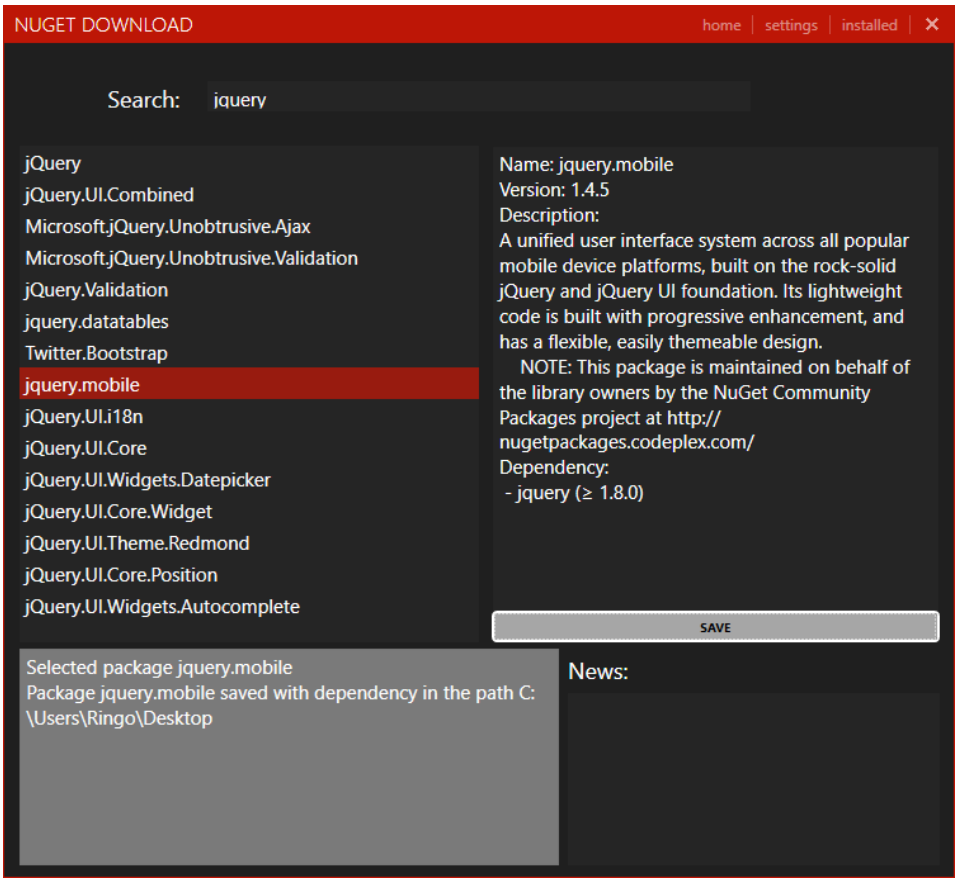
```
string[] fileNewsContent = File.ReadAllLines("logFileNews.txt");
bool change = true;
if (fileNewsContent.Length == 0)
{
    File.AppendAllText("logFileNews.txt", pathVersion + Environment.NewLine);
}
else
{
    int i = 0;
    int d = 0;
    foreach (string newsName in fileNewsContent)
    {
        string[] getVersion = newsName.Split(':');
        string package = getVersion[1].Split('\\')[getVersion[1].Split('\\').Length - 1];
        if (package == NamePackage)
        {
            d = i;
            change = false;
            break;
        }
        else
        {
            i++;
            change = true;
        }
    }
    if (change)
    {
         File.AppendAllText("logFileNews.txt", pathVersion + Environment.NewLine);
    }
    else
    {
         List<string> linesList = File.ReadAllLines("logFileNews.txt").ToList();
         linesList.RemoveAt(d);
         File.WriteAllLines("logFileNews.txt", linesList.ToArray());
         File.AppendAllText("logFileNews.txt", pathVersion + Environment.NewLine);
         OnSearchNews(obj);
         change = true;
    }
}
```

In questo codice viene mostrato il processo di quando si scarica un pacchetto senza aggiornamento oppure con aggiornamento, nel file **logFileNews.txt** ci sono tutti i pacchetti installati con il percorso e la versione per fare in modo che quando esce un aggiornamento il programma controlla la versione installata con quella che si trova sul sito di Nuget. Nella prima riga di codice si prende il file **logFileNews.txt** è si legge per mettere le varie righe in un **array** di **string**, subito dopo si crea una variabile che controlla se le versioni sono cambiate, l'**if** che segue serve a vedere se il file è vuoto, nel caso il file è vuoto si può aggiungere la prima linea senza controlli perché significa che non ci sono altri pacchetti installati. Nel caso in cui ci sono già informazioni dentro il file si inizia a controllare riga per riga le varie versioni che sono già state installate, nel caso in cui si trova una versione che esiste già viene salvato il numero della linea e insieme cambia anche la variabile **change** che fa significare che ci sono stati cambiamenti e si esce dal **foreach**, oppure continua la ricerca di versioni uguali tra di loro. Quando si esce dal **foreach** e si controlla se ci sono stati dei cambiamenti, nel primo caso si vede che non ci sono stati cambiamenti e si aggiunge al file il nuovo pacchetto scaricato, invece se è stato aggiornato il pacchetto si prende la linea da rimuovere e si aggiungi il pacchetto con la nuova versione e si ricarica la lista dei pacchetti che devono essere aggiornati.

Questi pezzi di codice si trovano nel file **NugetViewModel.cs** invece per la parte grafica si trova dentro il file **MainView.xaml** dove viene utilizzato del codice per far gestire gli eventi dentro la casella di testo oppure per la lista, per entrambi viene utilizzato lo stesso sistema che permette di gestire gli eventi anche sono non sono bottoni:

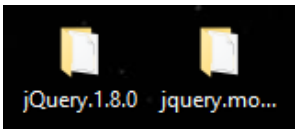
```
<TextBox>
   <i:Interaction.Triggers>
        <i:EventTrigger EventName="KeyUp">
             <i:InvokeCommandAction Command="{Binding Path=SearchCommand}"/>
        </i:EventTrigger>
    </i:Interaction.Triggers>
</TextBox>
```

Per il funzionamento di questo pezzo di codice si deve creare la variabile nel **UserControl** che crea il collegamento che permette il funzionamento degli eventi dei vari oggetti che si possono posizionare sullo schermo **xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"**. Per far funzionare il tutto si deve inserire dentro il tag **TextBox** oppure **ListBox** un altro tag che permette di creare una interazione con l'utente con il tag **i:Interaction.Triggers**, dentro il tag si deve inserire un altro tag che mostri il tipo di evento che si deve gestire, nel esempio mostrato viene creato un evento per quando l'utente scrive nella casella di testo e ogni vola che si alza il dito dalla tastiera viene attivato il metodo che viene richiamato dentro il tag **i:InvokeCommandAction**.

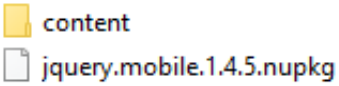



Questa è la finestra in cui ci sono tutti I campi attivi, in questo caso è stato scelto il pacchetto JQuery.mobile che è stato salvato nel percorso C:\Users\Ringo\Desktop, nella finestra di destra ci sono tutti i dettagli del pacchetto selezionato e in baso a sinistra ci sono tutti I passaggi avvenuti durante la selezione e il scaricamento del pacchetto. Quando si scarica il pacchetto sarà messo in una cartella con il nome e la versione del pacchetto insieme a tutte le varie dipendenze del pacchetto, la versione del pacchetto dipendente è la prima versione possibile e non l'ultima che si trova, in questo caso viene scaricato il pacchetto JQuery con la versione 1.8.0, perchè è la prima versione possibile per permettere il funzionamento del pacchetto jquery.mobile.

Questo è ciò che si può visuallizzare all'interno della cartella NugetPackage:




Dentro la cartella si troverà il file .nupkg che può essere installato su VisualStudio insieme a tutti i contenuti inerenti al pacchetto:



# 5 Test

## 5.1 Protocollo di test

| **Test Case:** | Test1 | **Nome:** | Creazione di una interfaccia grafica |
|---|---|---|---|
| **Descrizione:** | Visualizzare la parte grafica | | |
| **Prerequisiti:** | REQ-001 | | |
| **Procedura:** | 1. Cliccare sulla finestra Main<br>2. Cliccare sulla finestra Settings<br>3. Cliccare sulla finestra Installed<br>4. Cliccare sulla finestra About | | |
| **Risultati attesi:** | Tutte le pagine devono mostrare la propria parte grafica | | |

| **Test Case:** | Test2 | **Nome:** | Inserire dentro la barra di ricerca |
|---|---|---|---|
| **Descrizione:** | Nessuna descrizione | | |
| **Prerequisiti:** | REQ-002 | | |
| **Procedura:** | 1. Scrivere dentro la barra di ricerca | | |
| **Risultati attesi:** | Visualizzare dentro la ListBox dei pacchetti Nuget con similitudini con la stringa della barra di ricerca | | |

| **Test Case:** | Test3 | **Nome:** | Salvare il pacchetto |
|---|---|---|---|
| **Descrizione:** | Salvare il pacchetto nel percorso scelto | | |
| **Prerequisiti:** | REQ-003 | | |
| **Procedura:** | 1. Scegliere un pacchetto nella barra di ricerca<br>2. Visualizzare le informazioni del pacchetto<br>3. Cliccare sul bottone Salva | | |
| **Risultati attesi:** | Vedere il pacchetto salvato dentro il percorso che è stato scelto | | |

| **Test Case:** | Test4 | **Nome:** | Cambiare percorso |
|---|---|---|---|
| **Descrizione:** | Provare a cambiare percorso | | |
| **Prerequisiti:** | REQ-004 | | |
| **Procedura:** | 1. Andare nella pagina Settings<br>2. Cliccare il tasto Browse…<br>3. Scegliere il nuovo percorso<br>4. Visualizzare il nuovo percorso scelto | | |
| **Risultati attesi:** | Controllare che quando si salva un pacchetto, venga salvato nel percorso scelto | | |

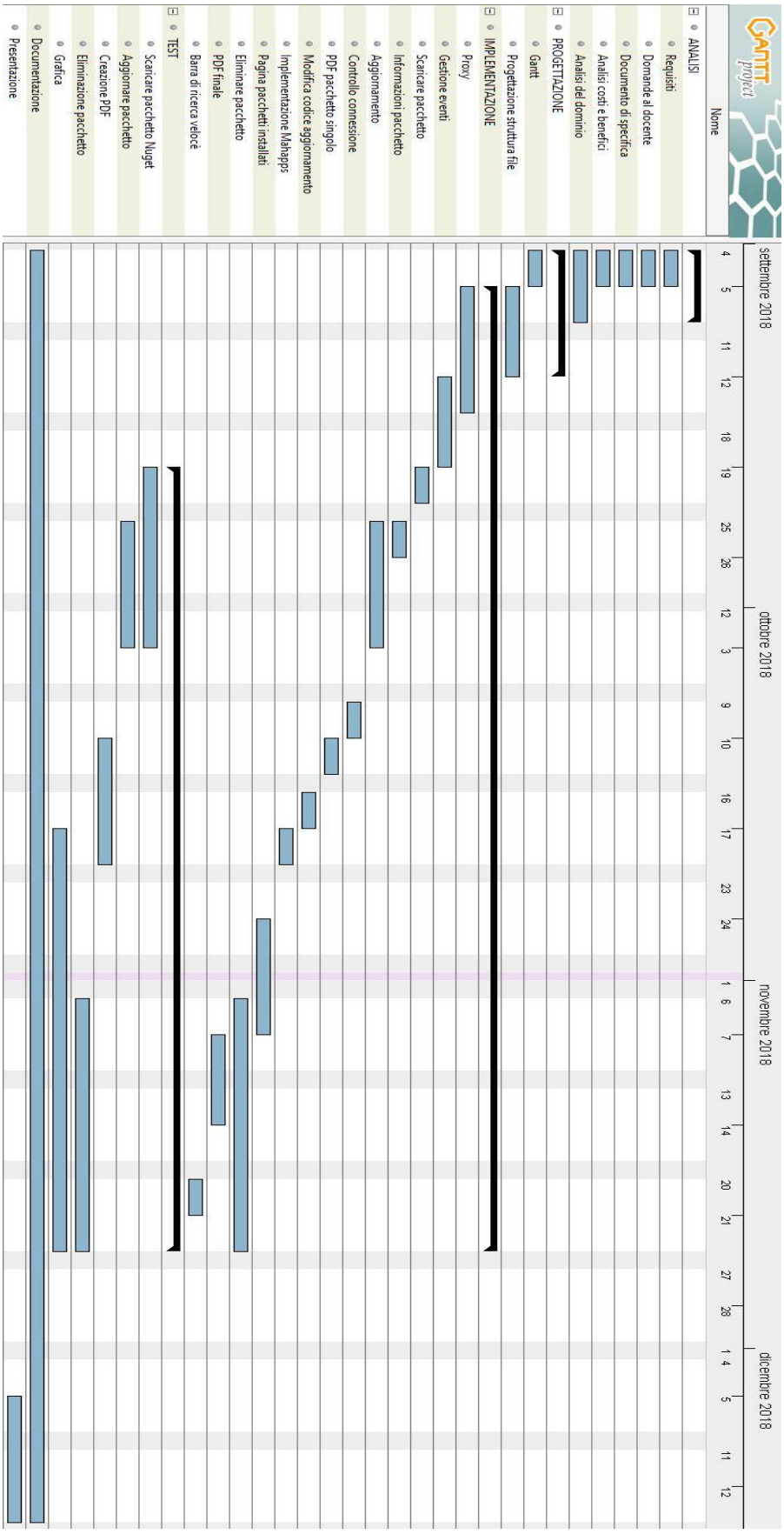
| **Test Case:** | Test5 | **Nome:** | Eliminare pacchetto Nuget |
|---|---|---|---|
| **Descrizione:** | Eliminazione di un pacchetto Nuget | | |
| **Prerequisiti:** | REQ-005 | | |
| **Procedura:** | 1. Andare nella pagina Installed<br>2. Scegliere un pacchetto da eliminare<br>3. Cliccare su Elimina | | |
| **Risultati attesi:** | Visualizzare che il pacchetto sia eliminato insieme alle su dipendenze | | |

| **Test Case:** | Test6 | **Nome:** | Creare file PDF |
|---|---|---|---|
| **Descrizione:** | Nessuna descrizione | | |
| **Prerequisiti:** | REQ-006 | | |
| **Procedura:** | 1. Andare nella pagina Installed<br>2. Cliccare su Generate PDF | | |
| **Risultati attesi:** | Controllare che il file PDF sia stato creato con tutte le informazioni dei pacchetti installati | | |

### 5.2 Risultati test

| Test | Risultati |
|---|---|
| Test1 | Visualizzazione corretta di tutte le pagine, senza problemi di grafica. |
| Test2 | Scrivere all'interno della barra e visualizzare ciò che è stato scritto con la tastiera. |
| Test3 | Visualizzare nella casella di testo grigia che il pacchetto è stato correttamente scaricato insieme alle su dipendenze. |
| Test4 | Vedere il nuovo percorso all'interno della casella di testo al posto del vecchio percorso, vedere anche l'aggiunta della cartella /NugetPackage. |
| Test5 | Visualizzare il pacchetto eliminato all'interno del cestino. |
| Test6 | Controllare che il file PDF sia stato generato senza problemi, con all'interno tutte le informazioni riguardanti i pacchetto installati. |

## 6 Consuntivo



# 7 Conclusioni

Questo progetto inizialmente ero scettico, perché non sapevo quanto poteva essere complicato il funzionamento del programma, ma quando ho iniziato a programmare mi sono sentito felice di cercare e aggiungere funzioni al programma che portassero un miglioramento al funzionamento finale, mi è piaciuto anche il fatto che molte funzioni del programma sono state scelte da me, con la approvazione del mio docente, questo ha reso possibile usare la mia creatività per creare un programma con il mio nome sopra e anche le mie idee all'interno.

## 7.1 Sviluppi futuri

In futuro si può provare a cercare in qualche modo una soluzione che permette di visualizzare tutte le versioni riguardanti un pacchetto Nuget e avere anche la possibilità di scegliere quale versione installare, come altra miglioria si può trovare un modo di velocizzare il processo di visualizzazione dei pacchetti cercati e far visualizzare tutti i pacchetti trovati invece di bloccare il numero di pacchetti da visualizzare.

## 7.2 Considerazioni personali

In questo progetto ho imparato a programmare con il codice di Nuget, e ho anche imparato a scrivere in un file PDF con C#, ho anche scoperto l'utilità di molti pacchetti Nuget installati, per esempio Mahapps che serve a gestire la grafica del programma, grazie a questo progetto ho capito che C# è uno dei miei linguaggi di programmazione preferiti.

# 8 Bibliografia

## 8.1 Sitografia


- https://blog.nuget.org/20130520/Play-with-packages.html
- https://docs.microsoft.com/en-us/nuget/
- https://stackoverflow.com/
- https://mahapps.com/
- https://archive.codeplex.com/?p=pdfsharp


# 9 Allegati

- Diari di lavoro