

## Capitolo 1

# Dettagli componenti

## 1.1 Updater - Parte client

### 1.1.1 Interfaccia grafica

#### 1.1.1.1 Componenti

Il programma Updater è costituito dalla finestra principale creata dall'oggetto:

```
MainWindow;
```

La quale è costituita da tre elementi grafici i quali ne racchiudono altri resi disponibili dal framework Windows Presentation Foundation e per questo non considerati. Essi sono:

1. `ProductsListView`;
2. `ProductControlView`;
3. `ProductInfoView`.

Nella figura 1.1 sono evidenziati e indicati attraverso la corrispondenza del numero dell'elenco precedente.

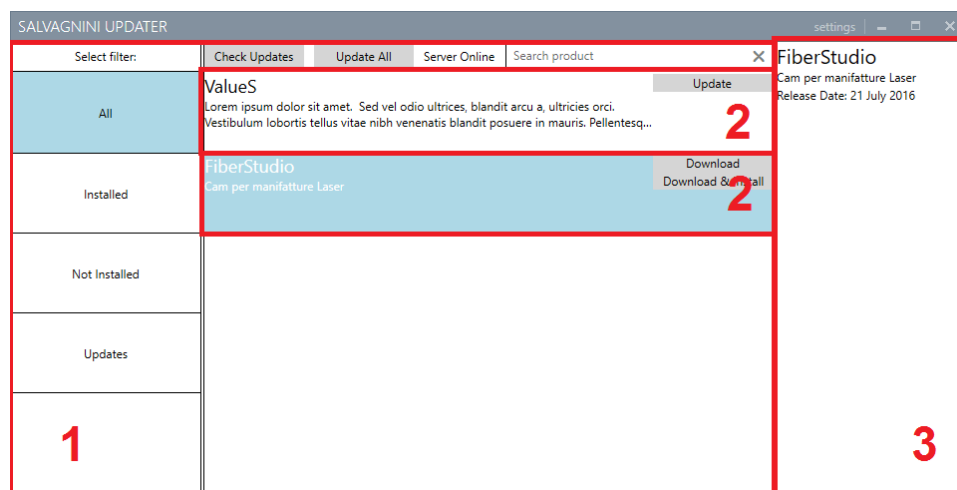


Figura 1.1: Componenti grafiche della finestra principale

È inoltre presente una schermata delle impostazioni del programma che si sovrappone agli elementi grafici ??, essa è definita nel componente `SettingsView`.

### 1.1.1.2 Funzionalità

La schermata principale (figura 1.2) permette subito di avere la lista dei prodotti software interagendo su di essi attraverso alla lista di pulsanti relativa. Questa cambia rendendo disponibili diversi pulsanti in base allo stato del prodotto software. Alla sinistra è possibile filtrare la lista di prodotti software mentre in alto verso sinistra è possibile filtrarla attraverso la ricerca del nome all'interno della casella di ricerca. Sempre in alto è possibile trovare i pulsanti:

**Check Updates** che aggiorna la lista dei prodotti software attraverso una nuova richiesta al server;

**Update All** che aggiorna tutti i prodotti software in uno stato di aggiornamento.

Alla sinistra invece troviamo uno spazio disponibile a mostrare tutti i dati disponibili di un prodotto software.

Tramite il click della parola *Settings* situata in alto a sinistra è possibile aprire una schermata che si sovrappone agli elementi grafici (figura 1.3).

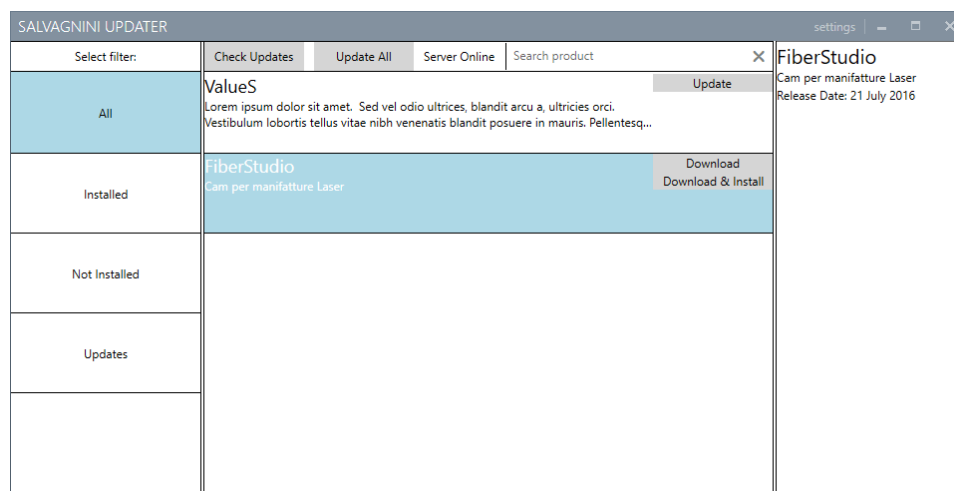


Figura 1.2: Updater - Finestra principale

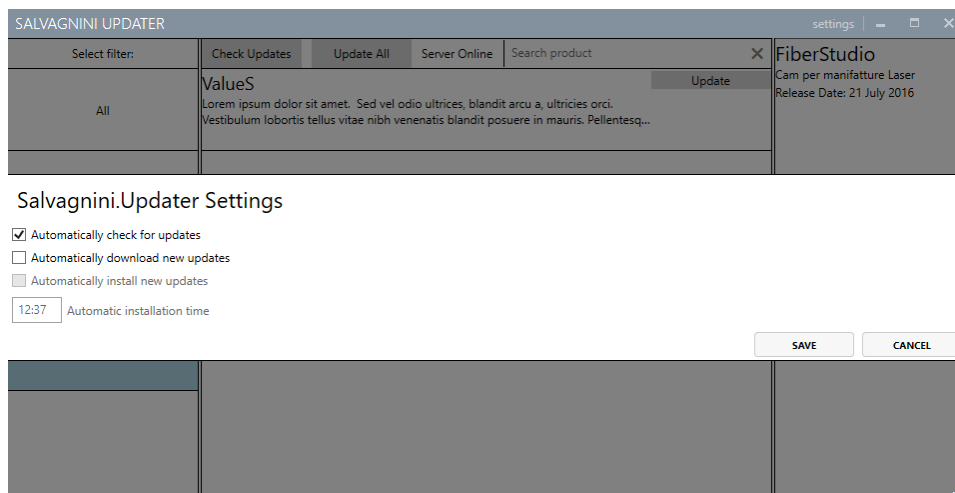


Figura 1.3: Updater - Schermata delle impostazioni

### 1.1.2 Flusso d'esecuzione operazioni principali

Di seguito vengono presentati i diagrammi di attività che mostrano in dettaglio i passaggi che la logica del programma effettua alla richiesta di esecuzione delle seguenti operazioni:

- Download (figura 1.4);
- Installazione (figura 1.5);
- Aggiornamento (figura 1.6);
- Reinstallazione;
- Download e installazione;

In particolare si vuole mostrare nel modo più chiaro possibile come si sia risolto il problema della gestione dei prerequisiti prima dell'installazione di un prodotto software che li richiede. I diagrammi manterranno nella colonna centrale il flusso con i casi peggiori. Per i diagrammi *“Reinstallazione”* e *“Download e installazione”* si veda il diagramma *“Aggiornamento”* (figura 1.6).

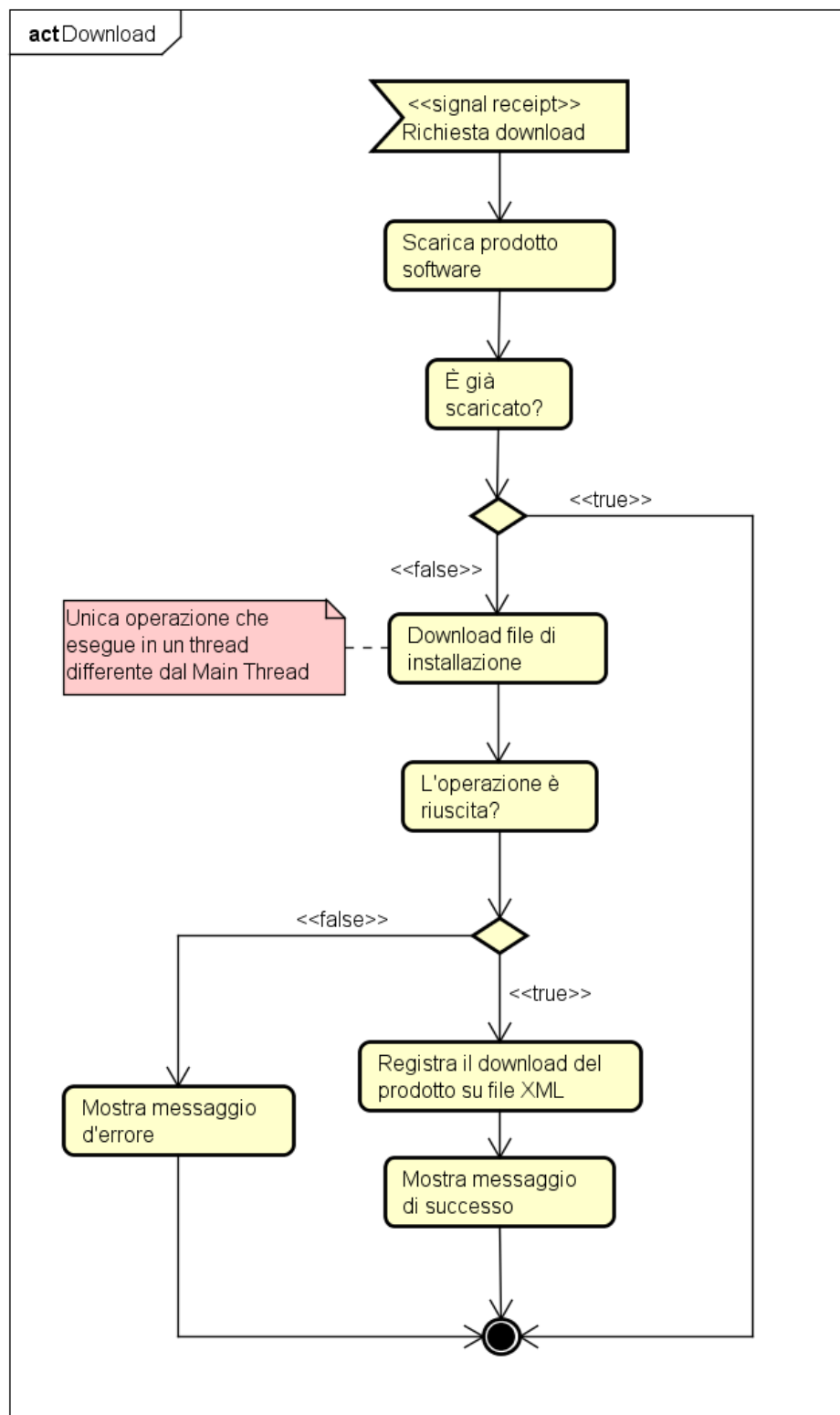


Figura 1.4: Diagramma di attività - Download

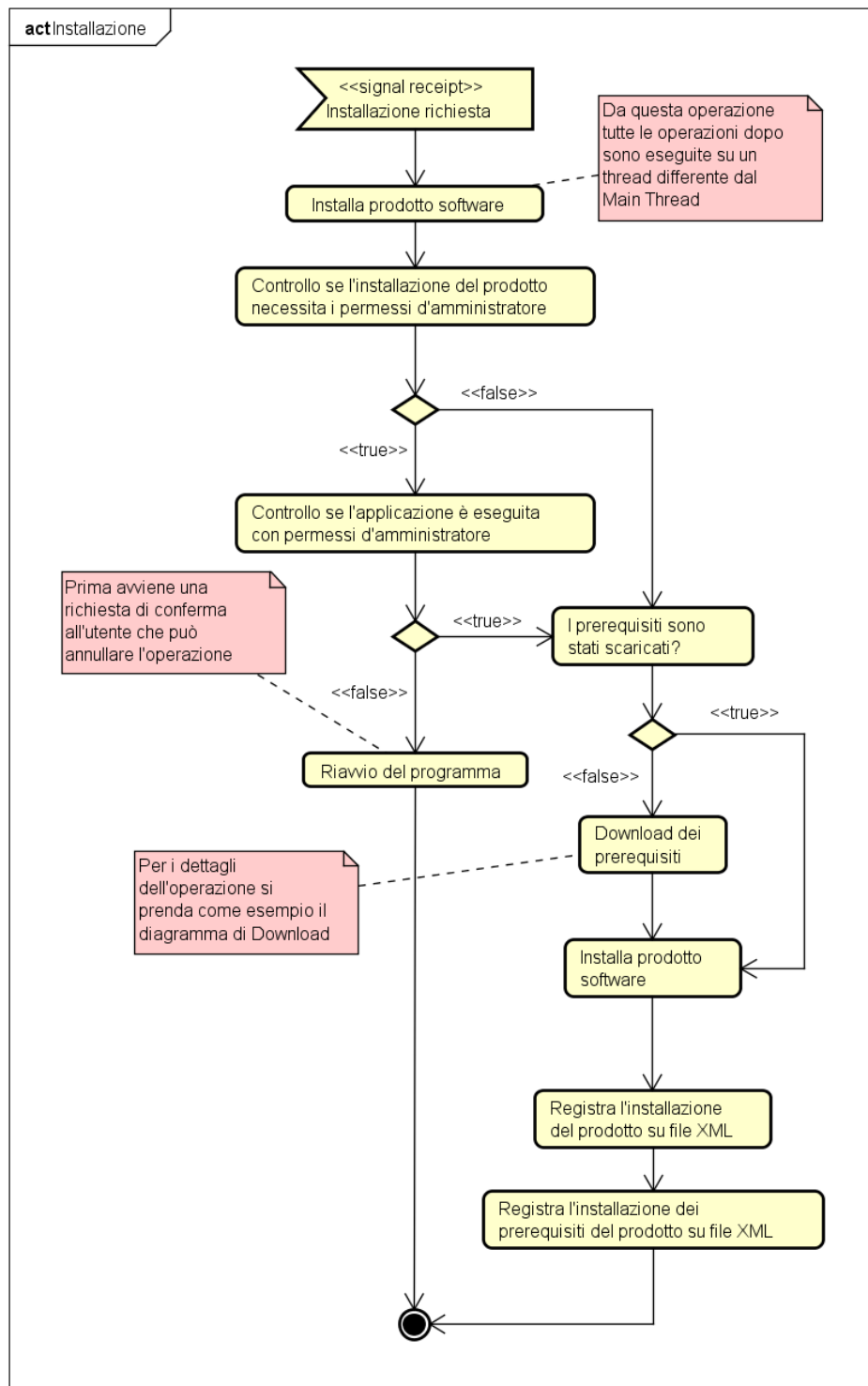


Figura 1.5: Diagramma di attività - Installazione

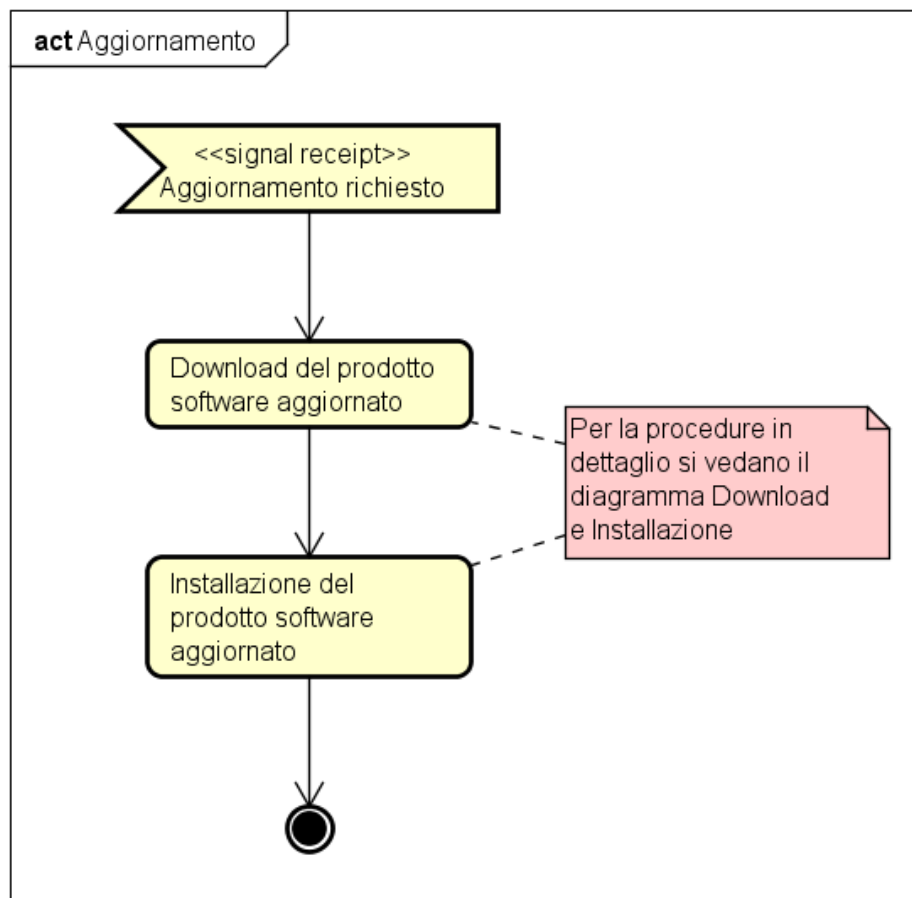


Figura 1.6: Diagramma di attività - Aggiornamento

## 1.2 Updater - Parte server

### 1.2.0.1 Sito web amministrativo

Per semplificare e agevolare le modifiche alla base di dati si è costruito un piccolo sito web costituito da quattro pagine web composte con elementi di Bootstrap e rese dinamiche grazie all'uso di Javascript con la libreria KnockoutJs.

### 1.2.0.2 Pagine web

**Login.html** permette l'autenticazione per accedere alle altre pagine;

**Products.html** permette di aggiungere, eliminare o modificare un prodotto software (figura 1.7).

**Prerequisites.html** permette di aggiungere, eliminare o modificare un prerequisito software (figura 1.8).





**Dependencies.html** permette di associare e dissociare i prodotti software dai prerequisiti software (figura 1.9).



Salvagnini Updater Admin

LogoutProductsProduct dependenciesPrerequisites

Products

Id	Name	Description	Arc	Min		Path		
				OS				
83	ValueS	Lorem ipsum dolor sit amet. Sed vel odio ultrices, blandit arcu a, ultricies orci. Vestibulum lobortis tellus vitae nibh venenatis blandit posuere in mauris. Pellentesque auctor convallis tellus in eleifend. Nulla suscipit tellus nec massa consequat scelerisque.	64	6.1		C:\Salvagnini\syscon\bin\ValueS\ValueS.exe		
87	FiberStudio	Cam per manifatture Laser	64	6.1				

Add Product

Name

Description

Architecture

x64

Minimum OS version

Windows XP

Executable path

C:\Salvagnini\syscon\bin\Example\Example.exe

Upload product setup

C:\your\path\setup.exe

Browse

Add

















Reset

Figura 1.7: Pagina web - Products.html

Salvagnini Updater Admin

[Logout](#) [Products](#) [Product dependencies](#) **Prerequisites**

### Prerequisites

Id	Name	
27	dotnetfx20sp1.exe	 
28	dotnetfx45.exe	 
29	vcredist2005_x64.exe	 
30	vcredist2005_x86.exe	 
31	vcredist2013_x64.exe	 
32	vcredist2013_x86.exe	 
33	vcredist_x86.exe	 
34	vcredist_x64.exe	 

Add Prerequisite

Upload prerequisite setup

C:\your\path\setup.exe

Browse

Add

Reset

Figura 1.8: Pagina web - Prerequisites.html

Salvagnini Updater Admin

LogoutProductsProduct dependenciesPrerequisites

Select product

Id	Name	Description	Arc	Min OS	Path	
83	ValueS	Lorem ipsum dolor sit amet. Sed vel odio ultrices, blandit arcu a, ultricies orci. Vestibulum lobortis tellus vitae nibh venenatis blandit posuere in mauris. Pellentesque auctor convallis tellus in eleifend. Nulla suscipit tellus nec massa consequat scelerisque.	64	6.1	C:\Salvagnini\syscon\bin\ValueS\ValueS.exe	Select
87	FiberStudio	Cam per manifatture Laser	64	6.1		Select

Select dependencies

Product: ValueS - Id product: 83

☒ Id: 27 - dotnetfx20sp1.exe

☒ Id: 28 - dotnetfx45.exe

☒ Id: 29 - vcredist2005\_x64.exe

☒ Id: 30 - vcredist2005\_x86.exe

☒ Id: 31 - vcredist2013\_x64.exe

☒ Id: 32 - vcredist2013\_x86.exe

☐ Id: 33 - vcredist\_x86.exe

☐ Id: 34 - vcredist\_x64.exe

ApplicaReset

Figura 1.9: Pagina web - Dependencies.html

### 1.2.0.3 Script Javascript

Per rendere le pagine web dinamiche si è utilizzato l'attributo **data-binding** di HTML5 e la libreria KnockoutJs che lo gestisce. Per ogni pagina si è codificato il relativo file di script più un file **Common** che contiene funzioni utilizzate in tutte le pagine.

La struttura degli script è la seguente:

1. Si assegna all'elemento HTML l'attributo **data-bind** al cui interno vengono usate keyword che Knockout riconosce e interpreta. Ad esempio:

```
<p data-bind="text : message"></p>
```

2. A questo punto sul file script associato alla pagina si crea un oggetto che rappresenta il ViewModel di quella pagina web:

```
var ViewModel = function () {  
    // ...  
}  
  
ko.applyBindings(new ViewModel);
```

3. All'interno del ViewModel saranno codificate i campi dati e i metodi che agiranno agli input dell'utente nella pagina html. Per rendere l'aggiornamento dei dati (data-binding) automatico è sufficiente definire i campi dati del ViewModel con un costrutto offerto dalla libreria (il nome delle variabili deve corrispondere alla stringa inserita nel data-bind delle pagine web):

```
var message = ko.observable();
```

4. Ora qualsiasi cambiamento sul valore della variabile **message** sarà automaticamente riflesso nella pagina web.

### 1.2.1 Web Api - Documentazione

Di seguito si elencano i **controller** definiti nella costruzione del servizio REST Updater Server. Per ogni controller vengono elencati gli indirizzi uri per poter effettuare una richiesta HTTP. Le tipologie di richieste HTTP supportate sono:

- Get: alla richiesta viene risposto restituendo dati;
- Post: alla richiesta vengono processati i dati contenuti in essa, se conformi vengono salvati nella base di dati;
- Delete: alla richiesta viene effettuata una cancellazione definitiva dei dati indicati;
- Put: alla richiesta vengono processati i dati contenuti in essa, se conformi vengono salvati nella base di dati sovrascrivendo quelli indicati.

La trasmissione dei dati avviene attraverso la serializzazione in formato JSON.

#### 1.2.1.1 ProductsController

Operazioni GET:

- **api/Products**  
Restituisce tutti i prodotti software all'interno della base di dati;
- **api/Products/{id}**  
Restituisce il prodotto software identificato dall'id indicato;
- **api/Products/{productId}/Prerequisites**  
Restituisce la lista di prerequisiti software del prodotto identificato da productId;
- **api/Products/check/{id}/{fileName}**  
Restituisce vero se il prodotto software identificato dall'id indicato ha il fileName indicato altrimenti falso;

Operazioni PUT:

- **api/Products/{id}**  
Sovrascrive con nuovi dati le informazioni associate al prodotto software identificato dall'id indicato;

Operazioni POST:

- **api/Products**  
Aggiunge un prodotto software con i dati contenuti nella richiesta;

- `api/Products/{productId}/Prerequisites`

Associa i prerequisiti contenuti nella richiesta al prodotto software identificato dal `productId` indicato. Dopo l'operazione solo i prerequisiti segnalati saranno associati al prodotto;

Operazioni DELETE:

- `api/Products/{id}`

Elimina il prodotto software identificato dall'`id` indicato.

### 1.2.1.2 PrerequisiteController

Operazioni GET:

- `api/Prerequisites`

Restituisce tutti i prerequisiti software all'interno della base di dati;

- `api/Prerequisites/check/{id}/{fileName}`

Restituisce vero se il prodotto software identificato dall'`id` indicato ha il `fileName` indicato altrimenti falso;

- `api/Prerequisites/check/{id}/hasDependencies`

Restituisce vero se il prerequisito software identificato dall'`id` indicato è associato almeno ad un prodotto software;

Operazioni PUT:

- `api/Prerequisites/{id}`

Sovrascrive con nuovi dati le informazioni associate al prerequisito software identificato dall'`id` indicato;

Operazioni POST:

- `api/Prerequisites`

Aggiunge un prerequisito software con i dati contenuti nella richiesta;

Operazioni DELETE:

- `api/Prerequisites/{id}`

Elimina il prerequisito software identificato dall'`id` indicato.

## 1.3 Concorrenza

La concorrenza ha coinvolto solamente la parte client del sistema sviluppato. La necessità di ricorrere alla concorrenza giunge quando si vuole rendere la propria applicazione responsive, ossia non impegnare il Main Thread, definito anche come UI Thread, in operazioni lunghe mostrando di fatto all'utente un'interfaccia bloccata. Mostrare all'utente l'interfaccia bloccata non è mai cosa buona, l'utente riceve infatti traduce questo feedback in qualcosa del tipo: "l'applicazione si è bloccata" e non "L'applicazione sta elaborando qualcosa per te". Per questo motivo nella parte client del sistema si è deciso di inserire multipli thread, e incapsulare le elaborazioni più complesse in Task.

### 1.3.1 Pattern

In C# esistono vari pattern per l'esecuzione di un ambiente concorrente, l'ultimo di questi, nonché quello consigliato dalla stessa Microsoft <sup>1</sup> e scelto per il progetto, è il *Task-based Asynchronous Pattern* (TAP).

#### 1.3.1.1 Applicazione Task-based Asynchronous Pattern

Esso si basa sui tipi `Task` e `Task<TResult>` definiti all'interno del namespace `System.Threading.Tasks` che sono utilizzati per rappresentare operazioni asincrone. Il pattern permette una notevole semplificazione nella codifica per introdurre codice concorrente, inoltre attraverso l'uso delle keyword `async` e `await`, parte del linguaggio C# è molto più facile capire quali porzioni di codice sono eseguite concorrentemente e quali no. La definizione di un metodo asincrono sarà di questo tipo:

```
public async Task<TResult> AsyncMethod(Type param)
```

e potrà essere chiamato attraverso l'uso di `await`:

```
var returnValue = await AsyncMethod(param);
```

la concorrenza viene creata attraverso l'uso del metodo statico `Task.Run()`:

```
var returnValue = await Task.Run(() => AsyncMethod());
```

Da qui in poi tutti i metodi definiti `async` a partire da `AsyncMethod()` saranno eseguiti in un thread differente dal Main Thread.

### 1.3.2 Componenti coinvolte

Di seguito si elencano le componenti e i relativi metodi che includono la concorrenza:

---

<sup>1</sup>Asynchronous Programming Patterns [https://msdn.microsoft.com/it-it/library/jj152938\(v=vs.110\).aspx](https://msdn.microsoft.com/it-it/library/jj152938(v=vs.110).aspx)

```
ProductWrapper: InitializePrerequisitesWrapper();
```

```
ProductWrapper: Download() (attraverso l'uso della classe WebClient  
di .NET);
```

```
ProductsPanelModel: LoadProduct();
```

```
ProductControlModel: InstallProduct().
```

In generale, nell'implementazione si sono seguite queste linee guida:

- Qualsiasi operazione che richiede l'accesso alla rete è asincrona;
- Le operazioni che non richiedono l'accesso a internet ma comunque necessitano di tempo per i calcoli sono asincrone e sono contenute esclusivamente nel macro componente *Updater UI Controls*;

### 1.3.3 Criticità e miglioramenti

Fortunatamente le operazioni da eseguire in concorrenza non riportano i problemi di accessi condivisi a risorse e questo ha semplificato le cose. Ciononostante la *best practice* di mantenere separato il codice concorrente dal codice non concorrente non è seguita. Seppur le componenti che includono l'avvio di nuovi task sono poche e i casi in cui si verifica sono limitati a quelli elencati precedentemente la linea di codice:

```
returnValue = await Task.Run(() => object.Method()));
```

non risulta ben isolata dal resto del codice sincrono.

Una più corretta soluzione sarebbe quella di incorporare l'avvio di nuovi task all'interno di singoli oggetti ben definiti in cui è chiaramente visibile l'implementazione della concorrenza.

### 1.3.4 Approfondimenti

- Stephen Cleary, *Async & Await* <http://blog.stephencleary.com/2012/02/async-and-await.html>;
- Stephen Cleary, *Asynchronous Programming Best Practices* <https://msdn.microsoft.com/magazine/jj991977.aspx>;
- Stephen Toub, *Task-based Asynchronous Pattern*, disponibile al seguente indirizzo <https://www.microsoft.com/en-us/download/details.aspx?id=19957>;
- Stephen Cleary, *Concurrency in C# Cookbook*.



## 1.4 PowerShell script

Nella presente sezione viene presentato e descritto lo script powershell che si occupa di automatizzare l'aggiornamento dei prodotti software all'interno della base di dati.

### 1.4.1 Ambito applicativo dello script

Il sistema di *Continuous Integration* (CI) aziendale consente di definire dei processi per la compilazione del codice, l'esecuzione dei test e l'esecuzione di script batch. L'aggiornamento di un prodotto software attraversa i seguenti passaggi:

1. **Commit** dello sviluppatore nel repository;
2. **Esecuzione processo** definito nella piattaforma di **CI** (solitamente compilazione ed esecuzione dei test);
3. **Creazione dell'eseguibile binario** di installazione del prodotto se il passaggio precedente non ha riscontrato errori;

A questo punto entra in gioco la nuova parte sviluppata:

4. **Esecuzione** di uno **script batch**, il quale contiene gli input per lo script powershell;
5. **Esecuzione** dello **script powershell**. A questo punto lo script esegue una normale richiesta PUT al servizio REST e aggiorna il prodotto specificato dagli input in ingresso.

### 1.4.2 Implementazione script powershell

#### Input richiesti

- **\$InFile**: il percorso dell'eseguibile per l'installazione;
- **\$uri**: l'indirizzo per effettuare la richiesta PUT al server;
- **\$user**: il nome utente per l'autenticazione della richiesta;
- **\$pass**: la password per l'autenticazione della richiesta.

#### Dipendenze esterne con .NET

- System.Web
- System.Net.Http

### Logica operativa

La logica operativa segue la costruzione di una richiesta HTTP di tipo PUT con autorizzazione base:

1. Defizione dell'autorizzazione base attraverso la coppia username e password per l'invio della richiesta HTTP;
2. Costruzione `ContentDispositionHeaderValue` di tipo *form-data* e inserimento del file e del suo nome;
3. Costruzione dell'header della richiesta HTTP inserendo all'interno l'oggetto precedentemente creato: `ContentDispositionHeaderValue`;
4. Invio della richiesta HTTP.