**Documentazione Applicazione Medico**

**Linguaggio di programmazione:** C#

**Ambiente di sviluppo:** Visual Studio 2015

**Modello Dati:** Sql Server

**DBMS:** Sql Server Management Studio 2014

**SINCRONIZZAZIONE**

Ad ogni avvio, l’applicazione effettua una sincronizzazione (tramite chiamata RESTFUL) con il server remoto *(Vedi Fig. 1)*, collegandosi ai servizi API disponibili, così da poter aggiornare i dati contenuti sul DB locale. Al contempo esegue l’invio dei certificati creati successivamente all’ultima data di sincronizzazione.

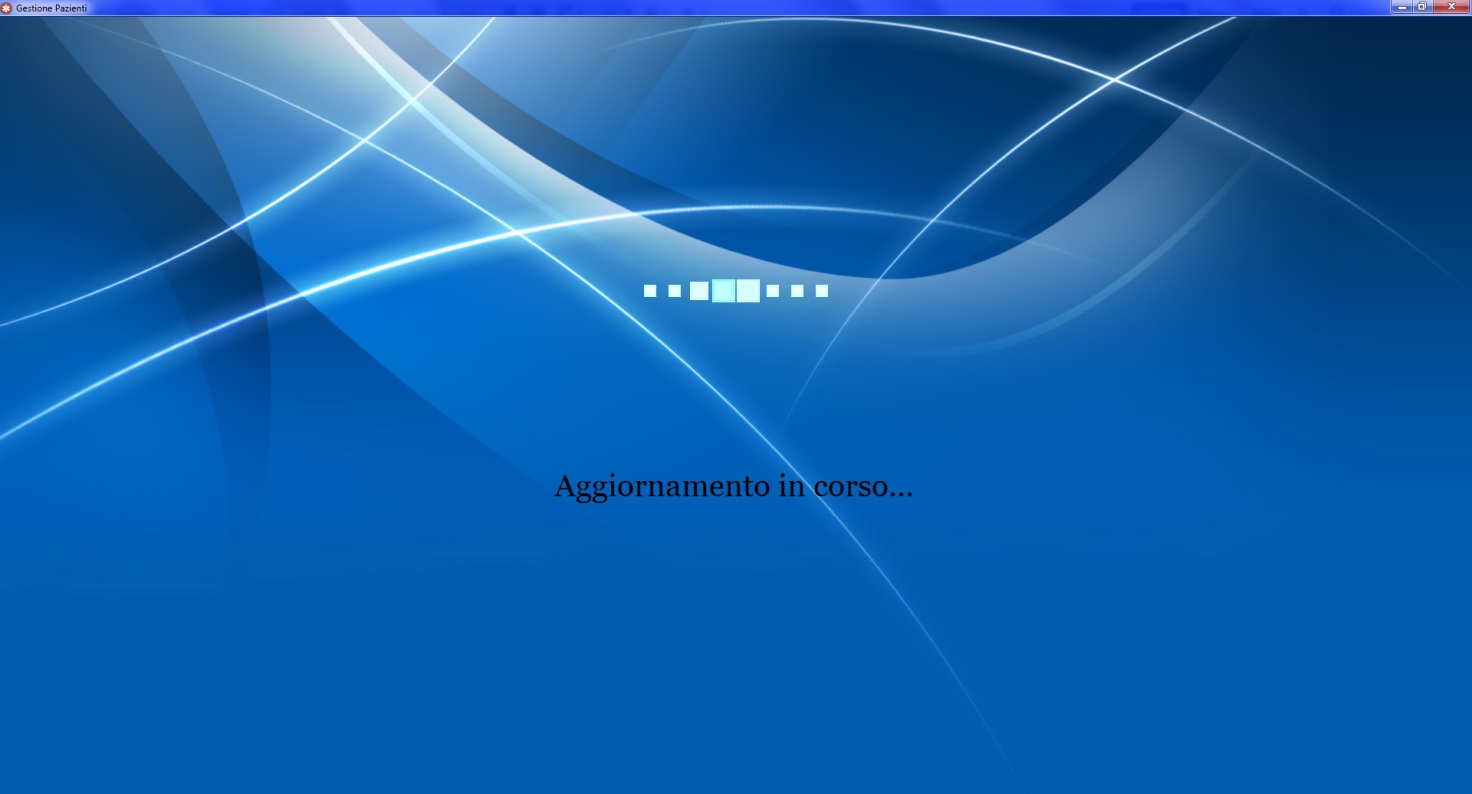
**

Fig. 1

È inoltre possibile effettuare la sincronizzazione in qualsiasi momento scegliendo la voce “Sincronizza” dal menù a discesa della voce File. *(Vedi Fig. 1.1)*

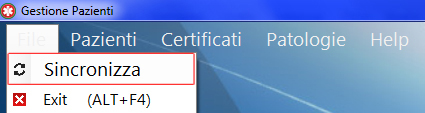


Fig. 2.1

**VISUALIZZAZIONE DATI**

Una volta effettuato l’aggiornamento automatico all’avvio dell’applicazione, l’utilizzatore viene reindirizzato alla sezione di visualizzazione della lista dei pazienti. *(Vedi Fig. 2)*



Fig. 2

Le altre due sezioni (Certificati e Patologie), accessibili dalle voci di menù, hanno un aspetto di base similare alla schermata della lista pazienti mostrata in precedenza*.* *(Vedi Fig. 3, 4)*:



Fig. 3

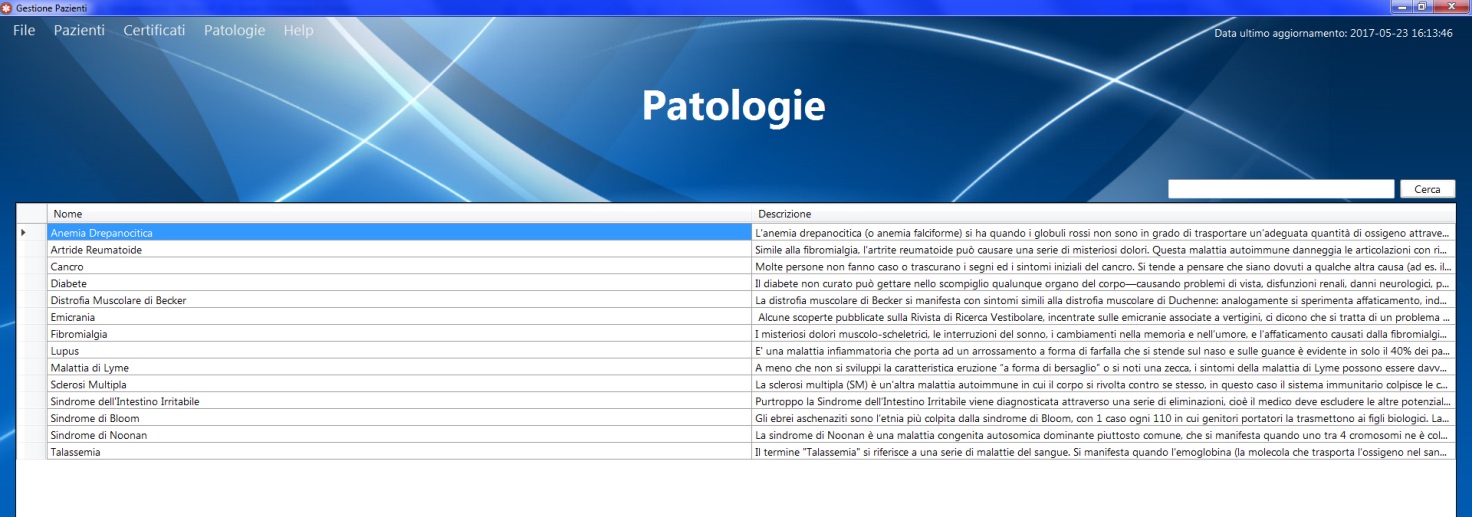


Fig. 4

In ognuna delle tre sezioni citate è possibile filtrare i dati visualizzati attraverso l’apposito campo di ricerca, posizionato in alto a destra della griglia. Tale filtro si applica a tutti i campi della tabella e ha effetto immediato, alla pressione di un qualsiasi carattere della tastiera, o al click del bottone “Cerca”.

Premendo sull’intestazione di una colonna è inoltre possibile ordinare i dati visualizzati (con carattere ascendente o discendente) in base al testo contenuto all’interno della colonna selezionata.

**SCHEDA DI DETTAGLIO DEL SINGOLO PAZIENTE**

C:\Users\admin\Documents\Alessandro\ProjectWork\Icone\1494256731_info.pngLa griglia dei pazienti si differenzia dalle altre, in quanto presenta all’inizio di ogni riga, l’icona   
 (*Vedi Fig. 2*) che consente di poter accedere alla scheda di dettaglio del paziente desiderato. Quest’ultima presenta un’intestazione, che varia dinamicamente a seconda del nominativo del paziente selezionato, e una griglia contenente i certificati ad esso associati. *(Vedi Fig. 5)*

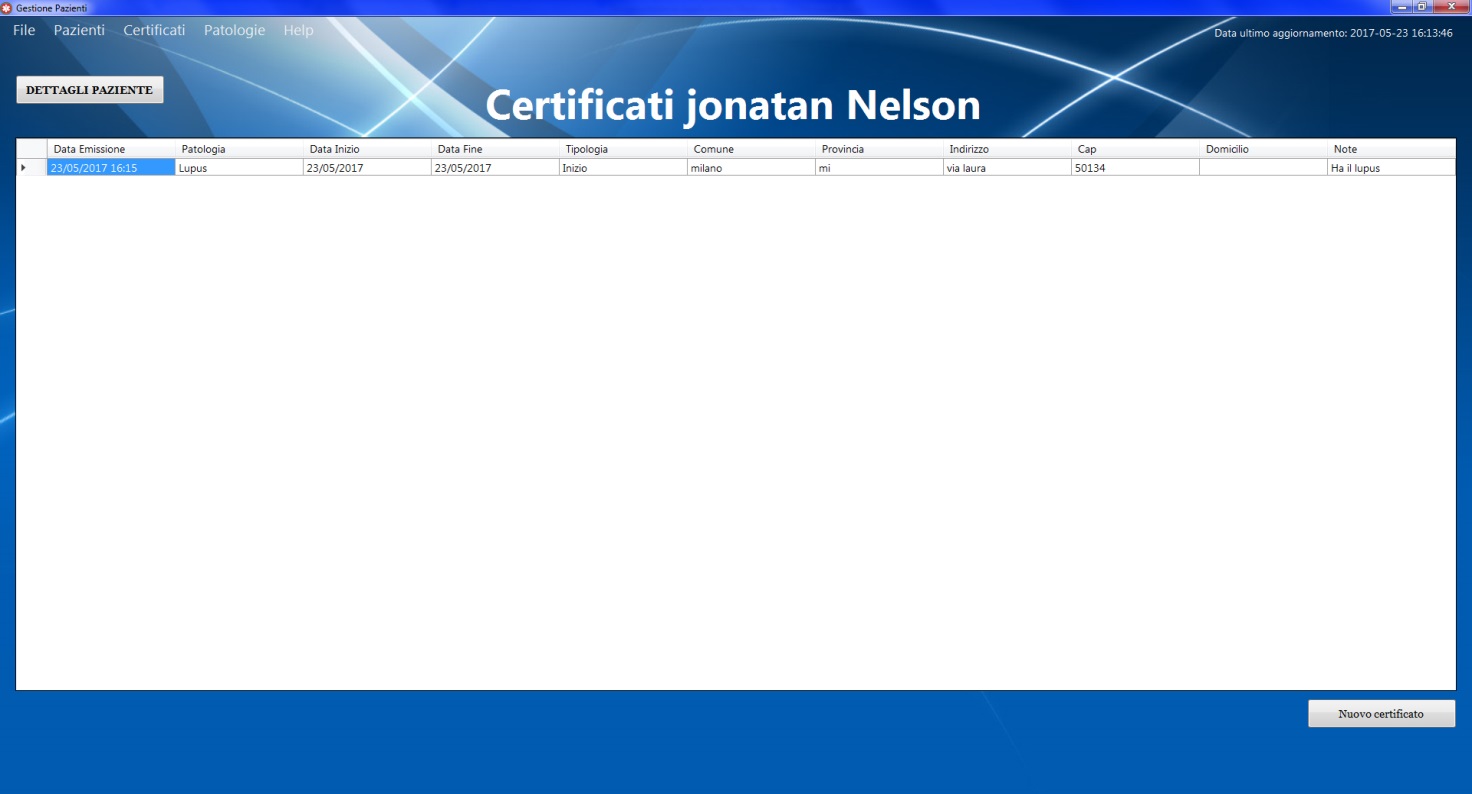


Fig. 5

Inizialmente i dettagli del paziente sono nascosti, ma è sempre possibile visualizzarli facendo click sul bottone “DETTAGLI PAZIENTE”, posto in alto alla sinistra del titolo. *(Vedi Fig. 6)*

**

Fig. 6

Il pulsante **Nuovo certificato**, posto in basso a destra *(Vedi Fig. 5)*, consente di accedere alla schermata di creazione di un nuovo certificato, partendo da un modello precompilato con i dati del paziente scelto. *(Vedi Fig. 7)*

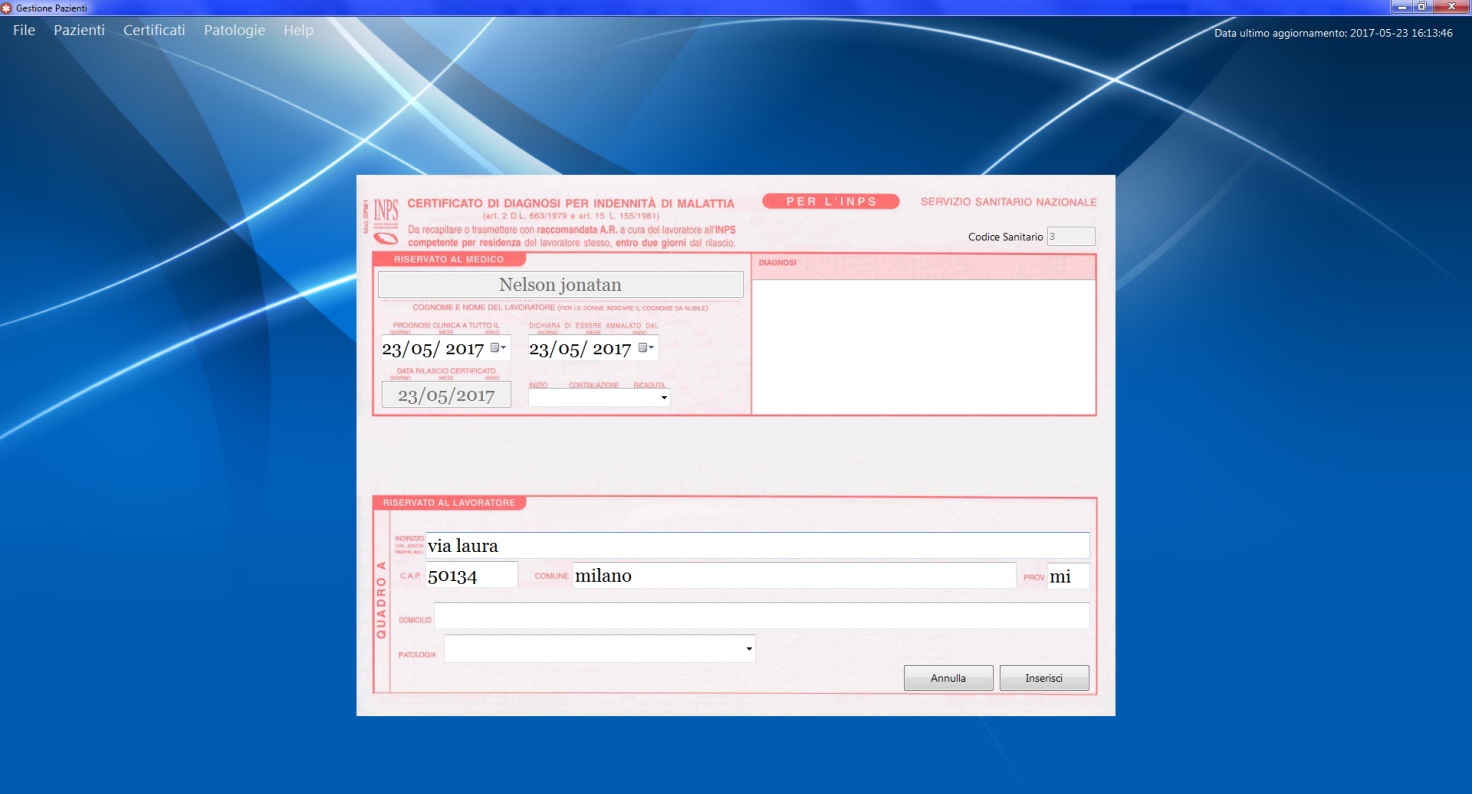
**

Fig. 7

Premendo sul pulsante **Inserisci** il certificato verrà aggiunto sul database come nuovo record e di conseguenza visualizzato nella griglia generale, e anche in quella del paziente a cui è associato.  
L’invio al server remoto del certificato appena inserito avverrà al riavvio dell’applicazione, o al click della voce di menù **Sincronizza**, come illustrato in *Fig. 1.1*Premendo invece sul pulsante **Annulla** l’utilizzatore verrà reindirizzato alla schermata precedente, ovvero alla scheda di dettaglio del paziente.

**ALTRE FUNZIONALITA’**

La voce **Exit**, presente nel menù a discesa **File**,permette di chiudere l’applicazione. (E’ possibile anche farlo con lo shortcut **ALT+F4**)

La voce di menù **Help**, per questioni di tempistiche e semplicità, mostra i nominativi degli sviluppatori del software in questione. In una situazione più “realistica” si potrebbe prevedere un reindirizzamento dell’utilizzatore ad un servizio di help desk, oppure ad una documentazione dettagliata su come poter risolvere eventuali errori.

In ogni momento è possibile visualizzare data e ora dell’**ultima sincronizzazione** col server (se il servizio è disponibile al momento della sincronizzazione), posta nell’angolo in alto a destra della finestra.

**SPECIFICHE NON IMPLEMENTATE**

Principalmente per questioni di tempo, alcune delle funzionalità sono state scartate in fase di progettazione del presente software, ma vengono documentate in questa sezione per tener traccia della loro importanza.

1. È possibile che un medico sia costretto ad utilizzare il presente software in più ambulatori, dunque è necessario prevedere un meccanismo di sincronizzazione più complesso di quello implementato al momento, in quanto ad ogni cambio di ambulatorio sarebbe necessario riscaricare i nuovi dati (pazienti, certificati e patologie), aggiornare quelli presenti e, avendo lato server una traccia dei dati eliminati per vari motivi, effettuare l’eliminazione dei dati oramai non più necessari.  
   Prendendo in esame il caso appena citato, considerando che i servizi di inserimento e aggiornamento dei dati sono presenti, ma sono stati gestiti per un utilizzo semplificato (ovvero ipotizzando che ogni medico ha un solo ambulatorio) ci siamo accorti che il medico in questione si ritroverebbe vari duplicati di alcuni certificati, e in alcuni casi alcune mancanze, nel caso in cui effettuasse realmente la sincronizzazione dell’applicazione su più di un database locale.
2. Non è presente la possibilità di poter eliminare alcun dato direttamente dall’applicazione. Tale specifica è stata decisa in fase di progettazione, anche in questo caso per questioni di tempi di sviluppo.
3. Al momento non è possibile definire quale medico effettua l’accesso all’applicazione, se non andando a cambiare manualmente il **codice medico** scritto all’interno del file di configurazione **config.ini** presente nella cartella dell’applicazione. In una situazione reale l’applicazione avrebbe sicuramente previsto una procedura di login all’avvio, permettendo in base alle credenziali immesse di poter visualizzare e lavorare con i dati associati all’utilizzatore corrente.

**DOCUMENTAZIONE TECNICA**

Il **cuore** dell’applicazione risiede principalmente in 3 funzionalità:

1. Ricezione di dati nuovi e/o aggiornati dal server remoto
2. Invio dei certificati creati dopo l’ultima data di sincronizzazione
3. Serializzazione e de-serializzazione delle informazioni da inviare/ricevere

Come si può vedere in *Fig. 8*, per prima cosa l’applicazione crea una nuova istanza della classe **WebClient**.   
WebClient è una classe presente nella libreria .NET che consente di comunicare con i servizi API esposti da un server, tramite protocollo RESTFUL.  
Nel nostro caso viene utilizzato sia per la ricezione di Pazienti, Certificati e Patologie, sia per l’invio dei nuovi Certificati inseriti.

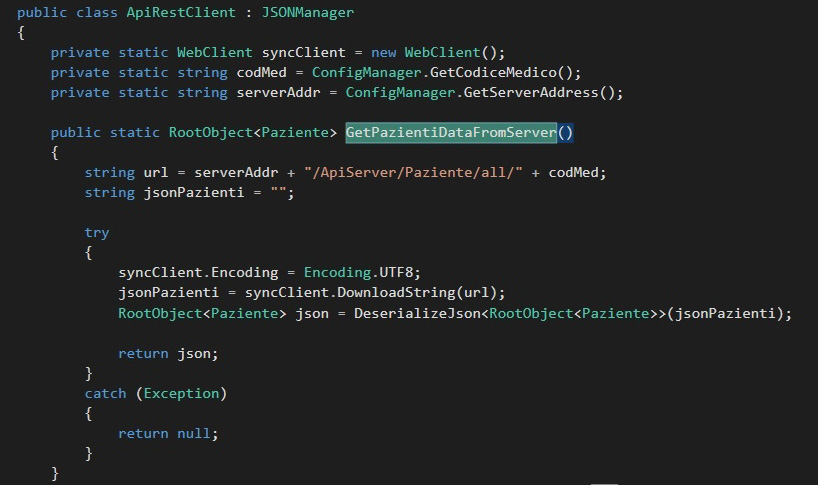


Fig. 8

Subito dopo, tramite due metodi statici presenti all’interno della classe **ConfigManager** (creata dagli sviluppatori), si vanno a leggere da un file di configurazione, inserito per semplicità nella cartella dell’applicazione, l’indirizzo IP del server remoto e il codice del medico che sta accedendo all’applicazione. *(Vedi Fig. 9)*

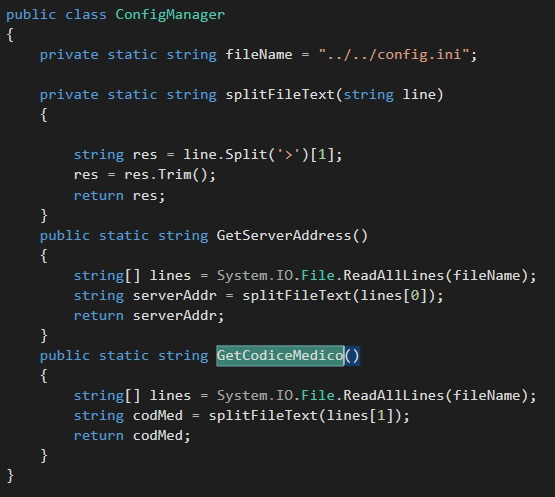


Fig. 9

Tornando alla *Fig. 8* si può vedere che il metodo **GetPazientiDataFromServer()** crea una stringa contenente la concatenazione dell’indirizzo IP del server remoto, del servizio API esposto e del codice medico, così da poterla passare come parametro al metodo **DownloadString** dell’istanza di classe di **WebClient**. Tale metodo restituisce una stringa (in formato JSON) contenente il risultato delle istruzioni associate a tale servizio API, in questo caso tutti i pazienti associati ad un singolo medico. Tali dati per essere processati correttamente dall’applicazione devono essere de-serializzati e ciò avviene tramite il metodo **DeserializeJson** della classe **JsonManager** *(vedi Fig. 10).*

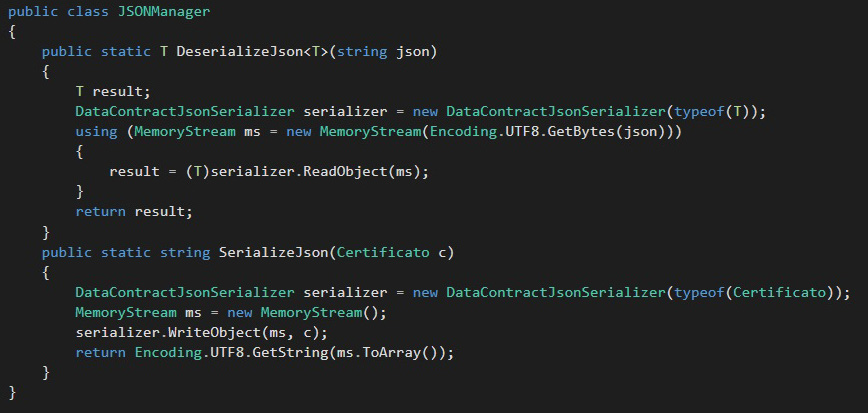
**

Fig. 10

Entrambi i metodi, esposti dalla classe **JsonManager**, utilizzano la classe **DataContractJsonSerializer** e la classe **MemoryStream** (presenti entrambe all’interno del framework .NET) per poter trasformare i dati ricevuti (o quelli da inviare) in un formato comprensibile dall’applicazione (o dal server remoto).  
Nel primo caso il metodo riceve una stringa in ingresso (ovvero il JSON risultante dalla chiamata REST) e restituisce un oggetto generico, così da non dover implementare un metodo per ogni classe di oggetti presente nel programma, che ha necessità di ricevere dati da un servizio API.  
Il secondo metodo invece riceve come parametro di ingresso un oggetto della classe **Certificato**, lo serializza, e restituisce una stringa formattata ad hoc per l’invio al servizio API relativo.  
Questo secondo metodo viene richiamato all’interno di un altro metodo dell’applicazione   
*(Vedi Fig. 11)* il quale, eseguendo un ciclo **foreach** su una lista di certificati, richiama il metodo **UploadString** della classe **WebClient**, passando come parametri in ingresso l’indirizzo del servizio API e la stringa JSON serializzata, così da poter inviare al server remoto ogni certificato non ancora spedito.

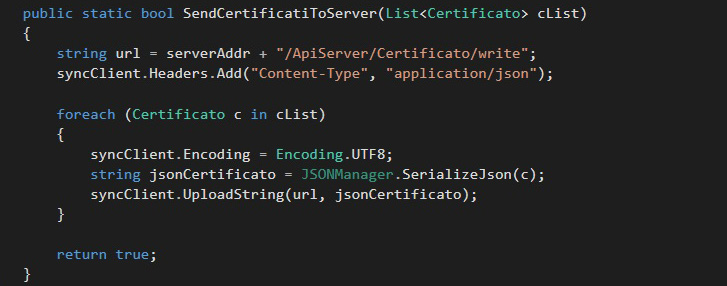


Fig. 11

L’interfaccia utente è stata creata interamente dagli sviluppatori, a partire da oggetti, e da eventi associati ad essi, presenti all’interno del framework. Le parti più significative sono:

* Ogni sezione è un **pannello** (ovvero un contenitore di altri oggetti) che viene mostrato e nascosto all’esigenza per simulare correttamente il flusso di utilizzo del software.
* Il **loader** iniziale, è un’immagine GIF in loop, alla quale sono stati applicati 5 secondi di **stallo** (gestito da un **Timer**) per dare l’impressione di un vero e proprio caricamento, anche se questo dovesse durare troppo poco per essere percepito dall’utente.
* L’immagine del **nuovo certificato**, la quale prende spunto da un vero e proprio certificato medico lavorativo, ma è stata editata per poter “ospitare” le caselle di testo contenenti date e informazioni del paziente.

Inoltre sono stati implementati svariati metodi di “aiuto” per poter gestire ogni componente in base alle idee degli sviluppatori. Tra questi ci sono **due metodi più significativi** che consentono ad ogni oggetto dell’interfaccia utente di avere una posizione e una dimensione definita programmaticamente, così da ottenere un **risultato visivo ottimale per l’utilizzatore.**