Università Politecnica delle Marche

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Sviluppo di un applicativo web per la gestione di dati farmaceutici utilizzati all'interno di strutture ospedaliere

Development of a web application for managing pharmaceutical drugs used within hospital structures

Tesi di Laurea di:
Danilo Gervasio

Relatore:
Prof. Adriano Mancini

Correlatore:
Ing. Luca Vescovi

Anno Accademico 2019-2020

Alla mia famiglia e a Sara.

Indice

1. Introduzione

Il progetto discusso all'interno di questa tesi riguarda la creazione di un applicativo web per la gestione di dati farmaceutici utilizzati all'interno di strutture ospedaliere. Questi dati sono gestiti dai sistemi automatizzati **Apoteca**: strutture robotiche in grado di preparare autonomamente soluzioni medicali; tali strutture sono state progettate e realizzate dal **Gruppo Loccioni**.
L'obiettivo primario di **Apoteca** è quello di preparare farmaci oncologici con una altissima precisione nel dosaggio dei componenti necessari alla preparazione di ogni singolo farmaco. Il raggiungimento di tale obiettivo è stato reso possibile "informando" la macchina di tutte le caratteristiche di ogni componente, farmaco e principio attivo maneggiato dalla macchina stessa; tali informazioni sono modellate in dati.
L'applicativo web oggetto di tale tesi permette al **Gruppo Loccioni** di gestire i dati utilizzati dalle macchine **Apoteca** installate in tutte le strutture ospedaliere del globo. L'applicazione è stato il risultato di un lungo processo di sviluppo.

1.1 L'impresa

Loccioni è un'impresa familiare, fondata nel 1968 da Enrico Loccioni. L'impresa si occupa di progettare e realizzare sistemi personalizzati sulle esigenze del cliente, integrando la competenza della misura con altre competenze complementari come la robotica, la sensoristica e la scienza dei dati.

Oltre alla realizzazione di sistemi automatici ideati per specifiche commesse, Loccioni ha realizzato dei progetti di mercato operanti in differenti settori tecnologici industriali. Attualmente l'impresa opera su otto progetti differenti nei seguenti mercati: automotive, energia, ambiente, industria, medicale, *train & transport*, *avio*, *electronics*.

1.1.1 Loccioni humancare

Loccioni humancare è l'unità di business del Gruppo Loccioni dedicata allo sviluppo di soluzioni per la cura, la salute, la nutrizione e il benessere; il gruppo opera nel suddetto settore mediante la linea di prodotti Apoteca. **Apoteca Chemo** è un sistema automatizzato per la preparazione di farmaci chemioterapici intravenosi.

Quando si parla di farmaci antitumorali vengono presi in considerazione preparati con un indice terapeutico molto basso, vale a dire farmaci in cui la dose terapeutica del preparato è molto vicina a quella tossica: il più piccolo errore nel dosaggio può avere un effetto tossico nel paziente.
Apoteca attraverso tre fasi di lavoro quali prescrizione, allestimento e somministrazione, garantisce la creazione di un prodotto efficace, sicuro e di elevata qualità. Inoltre i sistemi **Apoteca** (figura 1.1) sono generalmente installati, all'interno della struttura ospedaliera, in un apposito reparto denominato *farmacia*; in questo modo il sistema riesce a mettere in connessione il reparto ospedaliero che ha effettuato la richiesta del farmaco con la farmacia dell'ente ospedaliero.

Figura 1.1: Sistema Apoteca Chemo.

1.2 SWGA

I sistemi Apoteca sono diffusi in numerose strutture ospedaliere in tutto il globo. Ogni singola struttura effettua la preparazione dei composti terapeutici attendendosi alle norme legislative vigenti nel proprio stato. Questo comporta una diversità dei dati farmaceutici gestiti dai sistemi di ciascun ente ospedaliero.
I dati farmaceutici utilizzati dalla macchina hanno un'importanza rilevante nell'iter di creazione del farmaco: questi definiscono le modalità con il quale Apoteca deve creare uno specifico composto, specificando le quantità necessarie nella preparazione, e il modo in cui questo deve essere "maneggiato" dalla macchina.

L'inserimento dei dati farmaceutici all'interno di ciascun sistema Apoteca non è affidato direttamente al personale dell'ente ospedaliero ma è preso in carico dai tecnici del Gruppo Loccioni, al fine di garantire un massimo controllo sui dati inseriti all'interno di ogni specifica macchina. In questo modo la banca dati dei componenti trattati dai sistemi Apoteca è centralizzata e gestita dall'impresa.

I sistemi Apoteca di ciascuna struttura non interagiscono direttamente con la banca dati centralizzata, ma bensì con una base dati locale alla struttura, infatti in ogni ente ospedaliero è installato un database locale che contiene esclusivamente i dati utili ai propri sistemi Apoteca; in questo modo si evita un'architettura *single point of failure* in quanto i dati sono ridondanti nei database installati nelle varie strutture ospedaliere.

Per ogni principio attivo/flacone/farmaco/contenitore per il quale si necessità l'inserimento all'interno della macchina viene redatto un file ***.dat***(figura 1.3) contenente tutte le informazioni necessarie al sistema per la gestione del dato raffigurante l'elemento inserito. I file ***.dat*** sono creati direttamente dai tecnici Loccioni mediante l'utilizzo di una soluzione software desktop: **SWGA** (*SoftWare Gestione Anagrafica*, figura 1.2).

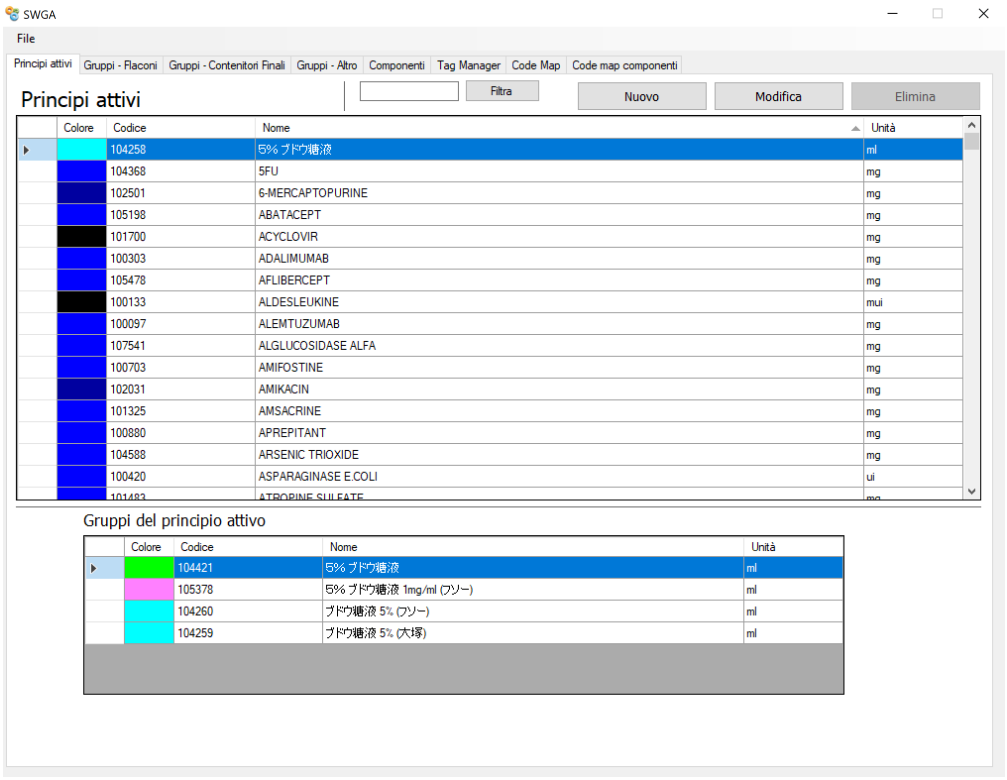


Figura 1.2: Tab dei principi attivi all'interno del programma desktop SWGA

La scelta di centralizzare la base dati ha anche una motivazione ingegneristica: se il Gruppo Loccioni avesse affidato a ciascuna struttura ospedaliera l'inserimento dei dati all'interno della propria macchina, informazioni riguardanti un medesimo componente trattato (ad esempio un farmaco $"x"$ o un principio attivo $"y"$) potrebbero essere inserite in maniera differente, con parametri differenti, da diverse strutture sanitarie.

Questo complicherebbe il controllo del corretto funzionamento dei sistemi installati: il farmaco $"x"$ avrebbe una risposta differente nei sistemi nel quale è utilizzato; il comportamento dipenderebbe dal modo in cui i dati del farmaco $"x"$ sono stati inseriti nel sistema.
Inoltre la centralizzazione della banca dati permette all'impresa di effettuare un'approfondita analisi dei dati.

```
[HEADER]
FILE_NAME=35205.01.01.dat
DECIMAL_SEPARATOR=.,
DATE_FORMAT=YYYYMMDD
COMP_ID=35205
COMP_UID=352050101
COMP_TYPE=DRG
COMP_NAME=SUREFUSER G 2ml/h NaCl 50ml
GRP_ID=100501
```

Figura 1.3: Intestazione del file .dat del componente con ID 352050101.

1.3 Perché SWGA Web

SWGA è installato all'interno di un server accessibile dagli operatori Loccioni mediante l'utilizzo di una VPN[1] e una connessione remota al server stesso. La natura desktop dell'applicativo ne rappresenta una forte limitazione nel suo uso: vi è una restrizione sul numero di utilizzatori contemporanei. Inoltre non è detto che un operatore abbia costantemente la possibilità di collegarsi tramite una connessione remota per utilizzare l'applicativo (ad esempio situazioni in cui il tecnico si trova fuori dalla sede lavorativa ma necessita di utilizzare il software).

Lo sviluppo della versione web del medesimo applicativo ha l'obiettivo di eliminare le sopra citate limitazioni e di concedere un utilizzo *multi-user* tramite la gestione della concorrenza degli utenti nelle operazioni **CRUD**[2] effettuabili sui dati gestiti dal programma.

1.4 Struttura della tesi

Il presente lavoro di tesi vuole fornire le conoscenze per comprendere la logica e le modalità con il quale è stata sviluppata l'applicazione. Non verranno esaminate tutte le componenti dell'applicativo ma solamente le principali, fondamentali al suo corretto funzionamento. Di conseguenza la tesi è organizzata nel seguente modo:

[1] Virtual Private Network: rete di telecomunicazione privata instaurata come connessione tra soggetti che utilizzano, come tecnologia di trasmissione, un protocollo di trasmissione pubblico e condiviso.

[2] Operazioni su basi di dati di creazione, lettura, aggiornamento ed eliminazione (create, read, update, delete)

- nella prima parte si descrive l'applicativo con le rispettive funzionalità e gli strumenti utilizzati per lo sviluppo.
- nella seconda parte si descrive l'architettura dell'applicazione e la suddivisione tra *front-end* e *back-end*.
- nella terza parte si descrivono le attività svolte dal back-end: la gestione delle chiamate API provenienti dai *clients* connessi, la gestione della comunicazione con il database, la gestione dell'autenticazione con il dominio Loccioni e la gestione dell'esportazione dei file *.dat*.
- nella quarta parte si spiega come è stato progettato il front-end e la sua infrastruttura interna.
- nella quinta parte si illustra come installare l'applicativo e renderlo funzionante.
- nella sesta parte sono presenti degli *screenshots* dell'applicativo in esecuzione.
- nell'appendice sono riportati tutti i cenni teorici al quale si fa riferimento nel corso della trattazione.

2. Il software

L'applicativo è composto da una parte *back-end*, sviluppata mediante il linguaggio di programmazione ***C#***[1], ed una parte *front-end*, sviluppata mediante il framework ***Angular 7***[2]. Il *back-end* interagisce con un database ***Oracle SQL***[3] localizzato su un server Loccioni.
La versione Web di SWGA, a differenza di quella desktop, ha un sistema di autenticazione per tenere traccia dei soggetti interagenti con i dati trattati; tale sistema si integra con il dominio aziendale Loccioni al fine di poter consentire l'accesso con le medesime credenziali utilizzate nell'ambiente lavorativo (figura 2.1).

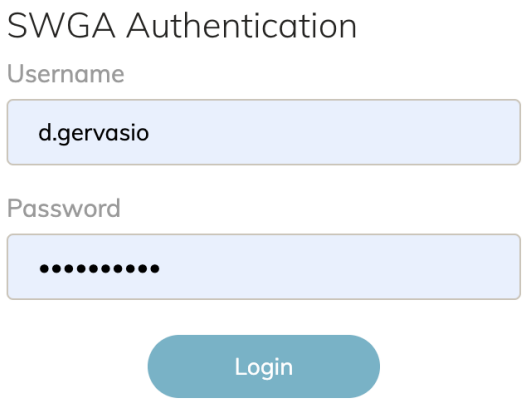


Figura 2.1: Schermata di autenticazione

[1] dettagli in appendice A
[2] dettagli in appendice B
[3] SQL: Structured Query Language, linguaggio utilizzato per l'interazione con basi di dati relazionali, dettagli in appendice C

2.1 I dati

Il trattamento dei dati di SWGA segue la medesimo logica della corrispettiva versione desktop. L'utente può interagire con i dati riguardanti i principi attivi contenuti all'interno di un farmaco, il contenitore finale dove è confezionato il farmaco, il farmaco stesso e altre tipologie di informazioni, ad esempio riguardanti i vari modelli di ago o di valvole per siringhe. Nel corso della trattazione si fa riferimento esclusivamente alle entità principio attivo(*Active Ingredient*), farmaco (*Flacon - Group*) e contenitore finale (*Final Container*), tralasciando le altre informazioni riguardanti un dato farmaceutico, al fine di semplificare e riassumere le funzionalità dell'applicativo con lo scopo di facilitarne la comprensione.

2.1.1 Componenti e gruppi componente

All'interno di SWGA si utilizza il termine **Componente** (*Component*) per indicare un oggetto associato ad altri oggetti, che possono essere a loro volta dei Xomponenti. Tale associazione implica una dipendenza diretta dell'oggetto di riferimento agli oggetti associati.

Esempio:

- *Un farmaco è associato ad un principio attivo e ad un solvente nel caso in cui si tratti di una soluzione in polvere (farmaco liofilizzato). Un farmaco è un Componente, un principio attivo non è un Componente in quanto non dipende da nessun altro oggetto.*
- *Un contenitore finale, come ad esempio una sacca o una siringa, è associato ad un solvente in esso contenuto. Un contenitore finale è un Componente.*

Farmaci e contenitori finali sono considerati Componenti, un principio attivo non è considerato come tale.

È stata fatta un ulteriore distinzione tra **Gruppo Componente** (*Group Component*) e **Componente** (*Component*). Il Gruppo Componente rappresenta un dato generico, il Componente rappresenta una forma specifica del dato. Differenti Componenti possono essere associati al medesimo Gruppo Componente se rappresentano forme differenti dello stesso dato, ad esempio forme differenti di commercializzazione di uno stesso farmaco.

Esempio Gruppo Componente farmaco e Componente farmaco:

Il Gruppo Componente flacone CISPLATINO 1mg/ml[4] *è associato a tutte le versione nel quale questo può essere acquistato, quindi ai seguenti Componenti flacone:*

- *CISPLATINO 100mg/ml*
- *CISPLATINO 50mg/ml*
- *CISPLATINO 10mg/ml*

Esempio Gruppo Componente contenitore finale e Componente contenitore finale:

Il Gruppo Componente contenitore finale SACCA è associato a tutte le versione nel quale questo oggetto è commercializzato, quindi i seguenti Componenti contenitore finale:

- *SACCA 100ml*
- *SACCA 50ml*
- *SACCA 25ml*

Dalla precedente spiegazione si può dedurre che il sistema Apoteca, in fase di preparazione di una soluzione, maneggia i Componenti e non i Gruppi Componente.

2.1.2 Tag

I ***Tag*** sono dei dati di configurazione utili al sistema per capire come utilizzare un generico Componente durante la preparazione di un farmaco. Ogni Componente all'interno di Apoteca ha una propria configurazione di ***Tag***.

I Tag possono essere di tipo *numerico, stringa, booleano, datatime*; ogni Tag ha un valore di default, inoltre Tag di tipo *numerico* e *datatime* hanno un massimo e minimo valore assumibile.

[4]Farmaco cardine del trattamento dei tumori polmonari.

2.2 Requisiti

I requisiti di un software definiscono cosa il sistema deve fare, quali vincoli deve rispettare e quali sono le proprietà essenziali che deve possedere. Tali requisiti si suddividono in:

- Requisiti funzionali: definiscono cosa il sistema deve e non deve fare e come deve reagire agli stimoli esterni.
- Requisiti non funzionali: definiscono delle proprietà del sistema che devono essere soddisfatte.

2.2.1 Requisiti funzionali

La versione web di SWGA svolge le medesime funzioni effettuate dalla corrispettiva versione desktop. I requisiti funzionali sono i seguenti:

- Operazioni CRUD per ***Active Ingredients*** con possibilità di filtraggio per *Code* o *Name* in fase di visualizzazione. Il click su uno specifico *Active Ingredient* permette la visualizzazione di tutti i gruppi componente che sono associati al suddetto elemento.
- Operazioni CRUD per gruppi componenti ***Flacons***, ***Final Containers***, ***Others*** con possibilità di filtraggio per *Code*, *Name* o *tipo gruppo componente* in fase di visualizzazione. Il click su uno specifico gruppo componente permette la visualizzazione di tutti i componenti che sono associati al suddetto elemento.
- Operazioni CRUD per componenti all'interno della tab ***Components***, con possibilità di filtraggio per *Code*, *Name* o *tipo componente* in fase di visualizzazione. Da tale schermata è possibile esportare, mediante il pulsante ***Export all***, tutti i file *.dat* di una determinata tipologia di componente e il file *.dat* del singolo componente selezionato, mediante il pulsante ***Export dat***. L'inserimento e la modifica di un componente ne permettono la configurazione dei tag associati.
- Operazioni CRUD per tag all'interno della tab ***Tag Manager***.
- Operazioni CRUD per code map e code map componenti rispettivamente all'interno dei tab ***Code Map*** e ***Code Map Components***.

2.2.2 Requisiti non funzionali

- Bloccare per tutti gli utenti operazioni di modifica e cancellazione su un gruppo componente in stato di modifica da parte di un utente.
- Bloccare per tutti gli utenti operazioni di modifica e cancellazione sui gruppi componente e principi attivi associati ad un gruppo componente in stato di modifica da parte di un utente.
- Bloccare per tutti gli utenti operazioni di modifica e cancellazione sui componenti associati ad un gruppo componente in stato di modifica da parte di un utente.
- Bloccare per tutti gli utenti operazioni di modifica e cancellazione sui gruppi componenti contenenti un principio attivo in stato di modifica da parte di un utente.
- Bloccare per tutti gli utenti operazioni di modifica e cancellazione sui componenti associati a gruppi componenti contenenti un principio attivo in stato di modifica da parte di un utente.
- Bloccare per tutti gli utenti operazioni di modifica e cancellazione su un componente in stato di modifica da parte di un utente.
- Effettuare il download dei file *.dat* esportati dall'utente tramite una cartella compressa contenente i file scaricati.

2.3 Strumenti utilizzati

La realizzazione del progetto ha richiesto l'utilizzo dei seguenti strumenti software di sviluppo:

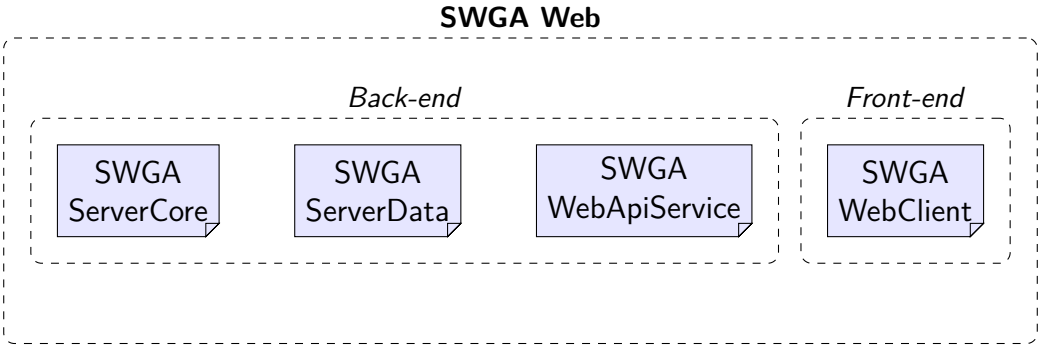
- **Microsoft Visual Studio 2015**: IDE [5] utilizzato per la scrittura del codice dell'applicativo, sia per la parte *front-end* che *back-end*.
- **Visual Studio Code**: editor di codice sorgente sviluppato da Microsoft. È stato utilizzato per sviluppare il *front-end* dell'applicativo.
- **Oracle SQL Developer**: IDE utile a interagire con un *Oracle database* mediante l'utilizzo del linguaggio SQL. Tale software è stato utilizzato per la creazione e il controllo della correttezza delle query utilizzate nell'interazione con la base dati.

[5] *Integrated Development Environment*

- **FortiClient**: software utilizzato per la connessione alla VPN Loccioni, al fine di poter usufruire del dominio di autenticazione Loccioni e della connessione al database.
- **Postman**: software che consente di creare, testare e documentare API. Tale applicazione è stata utilizzata per effettuare il *testing* delle chiamate API al server Web API.

3. Architettura dell'applicativo

La soluzione di **Visual Studio** dell'applicazione è costituita da quattro progetti differenti.



- *SWGA.ServerCore*: contiene le entità delle classi utilizzate all'interno del progetto e le interfacce dei servizi e delle repositories.
- *SWGA.ServerData*: contiene l'implementazione delle interfacce dei servizi e delle repositories definite all'interno di *SWGA.ServerCore*. Tale soluzione contiene una classe per l'esecuzione di **query SQL**[1] parametriche.
- *SWGA.WebApiService*: contiene tutte le informazioni necessarie al funzionamento di un **Web API service** in modalità **self hosted**[2]. Tale soluzione contiene i **Controllers Web API**[3] in cui sono definite le **chiamate Web API**[4] invocabili dai *clients* per la richiesta e il trattamento dei dati.
- *SWGA.WebClient*: contiene la logica e l'interfaccia grafica della parte *front-end* dell'applicativo.

[1]Richiesta di interazione con il database

[2]Servizio eseguito all'interno del proprio ambiente informativo

[3]Classi che ereditano dalla classe ***ApiController*** e definiscono dei metodi pubblici per gestire delle richieste HTTP.

[4]Richieste HTTP che un client può effettuare al *Web API Server*.

I primi tre progetti fanno parte della parte *back-end* dell'applicativo e ne costituiscono il ***Web Server***, l'ultimo progetto costituisce il ***Web Client***, ovvero la parte *front-end* dell'applicativo.

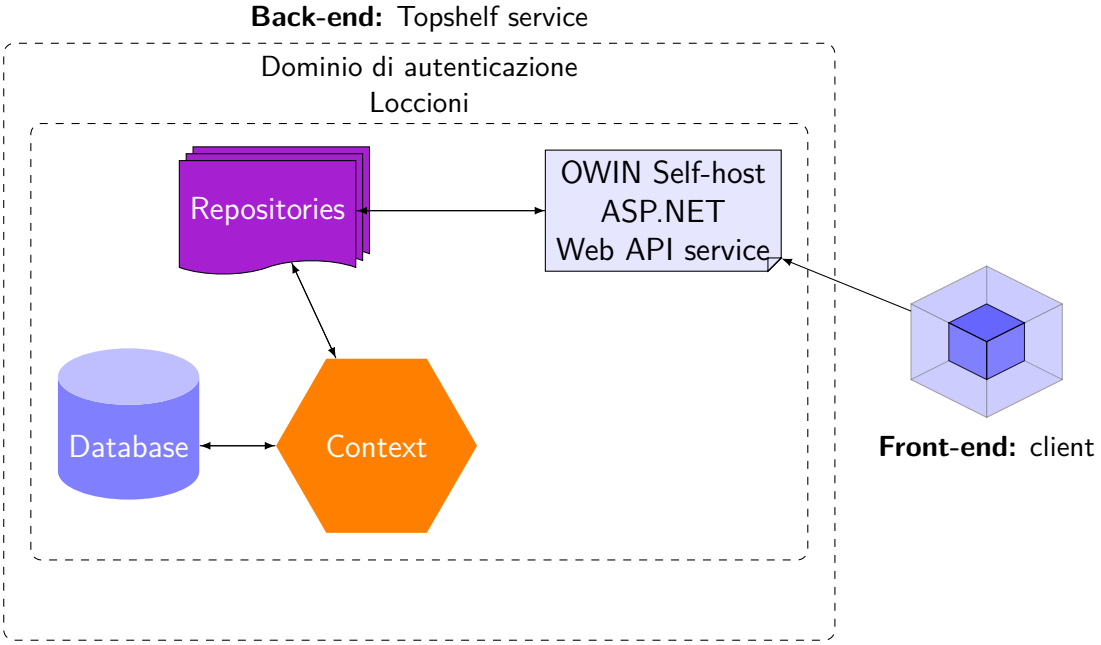
3.1 Web Server

Mediante l'utilizzo di Topshelf[5] l'applicazione può essere installata ed avviata come un *Windows Service*; in questo modo il *Web Server* può essere eseguito in background su una qualsiasi macchina.

L'avvio dell'applicativo consente all'utente di accedere al *Web Client* digitando all'interno del browser l'indirizzo *https://localhost:5001/* oppure *https://*INDIRIZZO IP SERVER*:5001/*.

Il **Web API Service**[6] contenuto all'interno del *Web Server* si predispone in ascolto di richieste Web API da parte di un *Web Client*.
All'arrivo di una richiesta, mediante un'opportuna **Repository**, il *Web API Service* utilizza l'istanza del contesto della base di dati per soddisfare la chiamata. I dati coinvolti nell'operazione sono modellati da una rispettiva **Repository**, richiamata in base al tipo di informazione coinvolta.

[5] Tecnologia discussa nel paragrafo successivo
[6] Dettagli in appendice D.2



Il *Web API Service* implementato all'interno del seguente progetto utilizza il framework **ASP.NET Web API 4.5.2**, che consente di creare servizi HTTP in grado di raggiungere un ampio numero di *clients*; questa è la piattaforma ideale per compilare applicazioni **RESTful**[7] in ambiente **.NET framework**[8].

3.1.1 Self hosted Web API server

La creazione di un *Web Server* indipendente eseguito come un *Windows Service* è stata resa possibile mediante l'utilizzo di **OWIN** e **TopShelf**. Tali librerie sono state aggiunte al progetto *SWGA.WebApiService* mediante pacchetti NuGet[5][9]

[7]Con tale termine si indica tutto ciò che viene programmato seguendo i principi dell'architettura REST, un sistema di trasmissione dei dati *stateless* utilizzante l'HTTP.

[8]Ambiente di esecuzione runtime delle applicazioni sviluppate per la piattaforma **Microsoft .NET**

[9]Meccanismo supportato da Microsoft per la condivisione del codice. Un pacchetto NuGet è un singolo file ZIP con l'estensione .nupkg che contiene codice compilato, altri file correlati a tale codice e un manifesto descrittivo.

OWIN

Generalmente la creazione di un *Web API Server*, mediante l'utilizzo di Visual Studio, viene effettuata attraverso un progetto **ASP.NET MVC**[10], il quale utilizza **IIS Express**[11] come server Web di sviluppo per il testing della applicazione web e delle chiamati Web API.

Durante la progettazione dell'applicativo si è scelto di eliminare la dipendenza del software da IIS attraverso una soluzione ***self-hosting***. Un progetto ASP.NET Web API con tale caratteristica ha la possibilità di essere eseguito in uno specifico *Web Server* differente da **IIS Express**. Per raggiungere l'indipendenza da IIS si è deciso di utilizzare **OWIN**: una specifica che descrive un'interfaccia standard per rendere un'applicazione web .NET disaccoppiata dal *Web Server* che la ospita, ovvero IIS. Un'applicazione web basata su OWIN non è più legata ad IIS ma può funzionare su un qualsiasi *Web Server* che sia conforme alla specifica.

TopShelf

TopShelf è un *Windows Service framework open source* per la piattaforma .NET che permette di creare e testare facilmente *Windows Service*.

3.1.2 Implementazione

Il progetto *SWGA.WebApiService* è stato creato come una **Console Application** attraverso Visual Studio (figura 3.1).

[10]Tale framework è un tipo di *Model-View-Controller* sviluppato dalla Microsoft come alternativa ad **ASP.NET Web Forms** per la creazione di applicazioni web.

[11]Versione di IIS, *Internet Information Services*, utilizzata da Visual Studio. IIS è un complesso di servizi server internet per sistemi operativi Microsoft Windows.

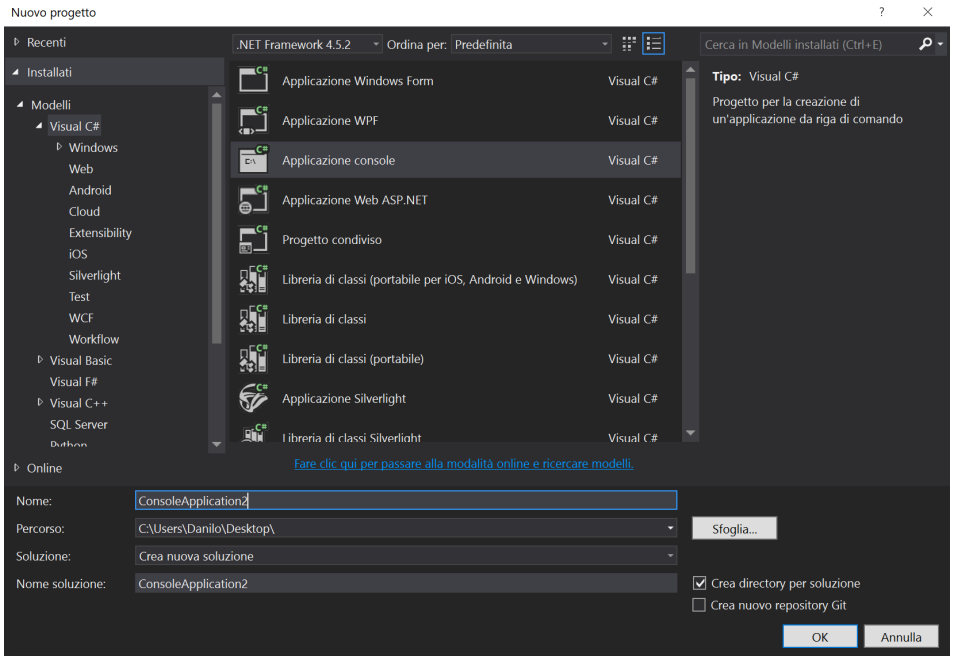


Figura 3.1: Creazione di una **Console Application** utilizzando **Visual Studio 2015**.

All'interno del metodo **Main** nel file **Program.cs** è stato configurato **TopShelf** per l'avvio del *Web Server* come *Windows Service*:

Listing 3.1: Configurazione di TopShelf all'interno del metodo Main nel file Program.cs

```
HostFactory.Run(r =>
                    {
                        r.Service<MainService>
                          (service =>
                          {
                              service.ConstructUsing(instance => new
                                  MainService());
                                service.WhenStarted(instance =>
                                    instance.Start());
                                service.WhenStopped(instance =>
                                    instance.Stop());

                          });
                        r.UseNLog();
                        r.RunAsLocalSystem();
                        r.SetDisplayName("SWGA Server");
                        r.SetServiceName("SWGA Server Service"); } );
```

Mediante la seguente istruzione si definisce **MainService** come classe rappresentante il servizio da eseguire all'avvio dell'applicativo.

```
r.Service<MainService>
```

Tale classe è definita all'interno del file **MainService.cs** ed implementa l'interfaccia **IService** definita all'interno del progetto *SWGA.ServerCore*.

Listing 3.2: Classe MainService

```
public class MainService : IService
    {
        private static Logger logger =
            LogManager.GetCurrentClassLogger();

        private WebApiService webApiSevice;
        public MainService()
        {
            this.webApiSevice = new WebApiService();
        }

        public bool Start()
        {
            logger.Info("Start Services");
            this.webApiSevice.Start();
            return true;
        }

        public bool Stop()
        {
            logger.Info("Stop Services");
            return true; } }
```

La suddetta classe rappresenta il servizio principale della parte *back-end* dell'applicativo; il suo avvio avviene eseguendo il metodo *Start()*. All'interno della suddetta funzione sono avviati tutti i sotto servizi utilizzati all'interno del *back-end*; in questo caso l'unico sotto servizio avviato è il **WebApiService** ma tale gestione predispone il *back-end* dell'applicativo a poter avviare più servizi simultaneamente nel caso fosse necessario.

La classe **WebApiService** avvia il *Web API Server* dell'applicativo. Tale classe è situata nel file **WebApiService.cs** e presenta un metodo *Start()* e *Stop()* rispettivamente invocati per l'avvio e la terminazione del servizio, ed un metodo *Configuration* utile a configurare il *Web API Server*. All'interno di tale classe si utilizza OWIN per disaccoppiare il *back-end* da IIS.

Listing 3.3: Metodi Start e Configuration della classe WebApiService

```
public bool Start()
        {
            StartOptions options = new StartOptions();
            options.Urls.Add("http://*:5001");
            WebApp.Start(options, (appBuilder) =>
            {
               this.Configuration(appBuilder);
               });
            return true;
        }

        public void Configuration(IAppBuilder app)
        {
            HttpConfiguration config = new HttpConfiguration();
            ConfigureOAuth(app);
            WebAPIConfig.Register(config);
            app.UseWebApi(config);
            string filedir = Path.Combine(Path.GetDirectoryName
            (Assembly.GetExecutingAssembly().Location),
            "../../../SWGA.WebClient/dist");
            app.UseFileServer(new FileServerOptions
            {
                FileSystem = new PhysicalFileSystem(filedir),
                EnableDefaultFiles = true,
                EnableDirectoryBrowsing = true,
                StaticFileOptions = { ContentTypeProvider = new
                    CustomContentTypeProvider() }
            });
        }
```

L'invocazione del metodo

```
options.Urls.Add("http://*:5001");
```

permette di registrare gli URLs con il quale i *clients* potranno accedere all'applicativo. All'interno del metodo *Configuration* si accede al percorso in cui è presente il codice compilato della parte *front-end* in modo da rendere l'applicativo accessibile ai *clients* che ne richiedono l'utilizzo attraverso il collegamento a gli URLs precedentemente registrati.

L'istruzione

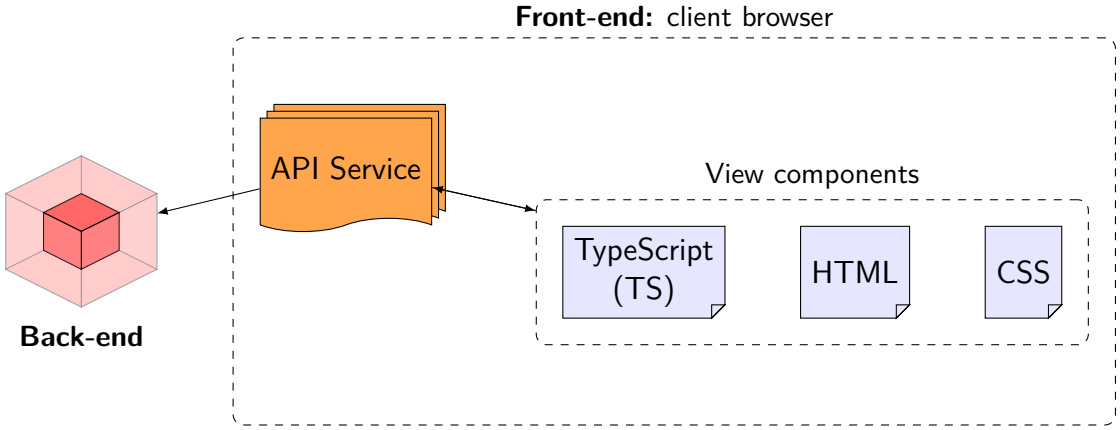
```
ConfigureOAuth(app);
```

all'interno del metodo *Configurarion* permette di configurare il dominio di autenticazione del servizio con il dominio Loccioni.

3.2 Web Client

Il *Web Client* costituisce la parte *front-end* dell'applicativo. Questa è accessibile digitando in un *web browser*[12] l'URL con il quale è stato registrato il *Web Server*: in questo modo la prima pagina da visualizzare del *front-end* è scaricata all'interno del *browser*, il quale esegue le funzioni presenti nella pagina stessa per garantirne una corretta esposizione all'utente.

Il *front-end* è costituito da delle *View Components* che permettono la visualizzazione delle pagine dell'applicativo (ad esempio la pagina del *Login*, la pagina di visualizzazione degli *Active Ingredients* ecc.). Ogni componente invoca le **chiamate Web API** del *Web Server* mediante l'utilizzo dei metodi pubblici esposti dalla classe **APIService** del *Web Client*.



3.2.1 Implementazione

Il progetto *SWGA.WebClient* è stato creato attraverso la creazione di un progetto **Angular 2**[12] (a seguito degli aggiornamenti fatti all'applicativo, l'attuale versione del *front-end* utilizza **Angular 7**) mediante l'utilizzo di un *tool* a riga di comando chiamato **Angular CLI**; questo consente di inizializzare un'applicazione Angular, creare e configurare moduli, servizi, direttive e *pipe*.

Angular CLI è stato installato mediante l'utilizzo del *package manager*[13] NPM

[12] Applicazione utilizzata per l'acquisizione, la presentazione e la navigazione di risorse sul web.

[13] Sistema di gestione dei pacchetti, ovvero una collezione di strumenti software, presenti in un sistema operativo, che automatizzano il processo di installazione, configurazione, aggiornamento e rimozione dei pacchetti software presenti in un computer.

(*Node Package Manager*)[14] attraverso il seguente comando:

```
npm install -g @angular/cli
```

Successivamente è stato creato il *front-end* mediante l'utilizzo del comando:

```
ng new SWGA.WebClient
```

Tale istruzione crea una nuova area di lavoro che prende il nome di **Angular Workspace** all'interno del quale vengono generati vari file di configurazione e le cartelle contenenti il codice dell'applicativo (figura 3.2).

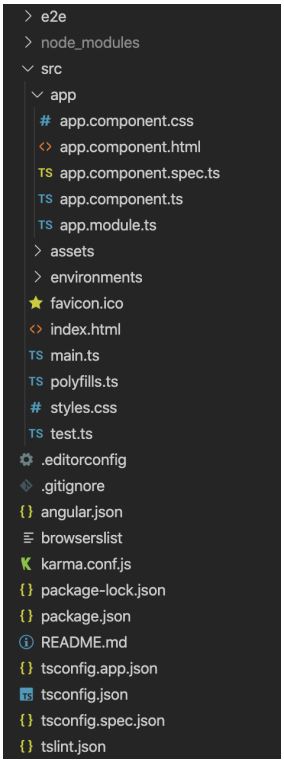


Figura 3.2: File generati inseguito alla creazione di un progetto Angular mediante l'utilizzo di Angular CLI

[14]Principale software utilizzato per maneggiare moduli Node.js, dettagli in appendice B

4. Back-end

In tale capitolo si analizzano le principali funzionalità svolte dal *back-end* dell'applicativo:

- Gestione della comunicazione con il database **Oracle SQL**.
- Gestione delle **chiamate Web API** effettuate dai *clients*.
- Gestione dell'**autenticazione** con il dominio Loccioni.
- Gestione dell'esportazione dei file ***.dat***.

4.1 Comunicazione con il database

La comunicazione con il database è effettuata dalla classe **CytoContext**, la quale espone dei metodi pubblici per l'esecuzione di query parametriche. Tale classe è utilizzata all'interno della classe **UnitOfWork**: quest'ultima mette in connessione un'istanza della classe CytoContext con le varie **Repositories**.

4.1.1 CytoContext

Tale classe è definita all'interno del file **CytoContext.cs** nel progetto *SWGA.ServerData*.

Listing 4.1: Attributi e costruttore della classe CytoContext

```
1 private OracleConnection Connection;
2 private OracleCommand Command;
3 private OracleDataAdapter DataAdapter;
4 private DataSet DataSet;
5 private string ConnectionString;
6
7         public CytoContext()
8         {   this.ConnectionString =
              System.Configuration.ConfigurationManager.
```

```
            ConnectionStrings["CytoTest"].ConnectionString;
            this.Connection = new OracleConnection(ConnectionString);
            this.Command = Connection.CreateCommand();
            this.DataAdapter = new OracleDataAdapter(Command);
            this.DataSet = new DataSet(); }
```

La stringa di connessione è definita nel file XML[1] **App.config** all'interno del progetto *SWGA.WebApiService*.

Listing 4.2: Stringa di connessione al database

```
<connectionStrings>
  <add name="CytoTest" connectionString="STRINGA DI CONNESSIONE
      DATABASE LOCCIONI"
      providerName="Oracle.ManagedDataAccess.Client"  />
</connectionStrings>
```

La classe CytoContext presenta dei metodi per aprire e chiudere la connessione con il database:

Listing 4.3: Metodi per l'apertura e la chiusura della connessione con il database

```
public void open()
      {
          if (Connection.State.Equals(ConnectionState.Closed))
          {
              try
              {
                  Connection.Open();
                  Console.WriteLine("Connection to Cyto db opened
                      succesfully");
              }
              catch (Exception)
              {
                  if (Connection != null)
                  {
                      Connection.Dispose();
                      Connection = null;
                  }

                  throw;
              }
          }
      }

      public void close()
      {
          if (Connection != null)
```

[1]Metalinguaggio per la definizione di linguaggi di *markup*

```
26              {
27                  if (Connection.State.Equals(ConnectionState.Open))
28                  {
29                      try
30                      {
31                          Connection.Close();
32                          Console.WriteLine("Connection to Cyto db
                                closed succesfully");
33                      }
34                      catch (Exception)
35                      {
36                          if (Connection != null)
37                          {
38                              Connection.Dispose();
39                              Connection = null;
40                          }
41                      }
42                  }
43
44              }
45          }
```

Inoltre sono presenti dei metodi per effettuare le operazioni CRUD (*Create, Read, Update, Delete*) con i dati. Tali funzioni richiedono generalmente dei parametri di ingresso: il nome della tabella sul quale si deve effettuare la *query* (*tables*), il set dei campi da visualizzare nel caso in cui si effettui una SELECT (*param*), i valori degli attributi dei dati da aggiungere o aggiornare in una interazione nel caso in cui si effettui una INSERT o una UPDATE (*values*), le condizioni da soddisfare per l'esecuzione della *query* (*conditions*), il nome dell'utente che ha richiesto di effettuare l'interazione (*source*) e una collezione di valori utilizzati all'interno della *query* (*pmCollection*).

Listing 4.4: Metodo per ottenere una lista di dati mediante l'esecuzione del comando SQL *SELECT*.

```
1   public DataTable GetAll(string param, string table, string
        Conditions, OracleParameter[] pmCollection)
2           {
3               if (DataSet.Tables.Count != 0)
4                   DataSet.Tables["cytoFill"].Clear();
5
6               if (Conditions == null)
7                   Command.CommandText = "SELECT " + param + " FROM " +
                        table;
8               else
9                   Command.CommandText = "SELECT " + param + " FROM " +
                        table + " WHERE " + Conditions;
10
```

```
            if (pmCollection != null)
            {
                Command.Parameters.AddRange(pmCollection);
                Command.BindByName = true;
            }

            DataAdapter.Fill(DataSet, "cytoFill");
            DataTable dt = DataSet.Tables["cytoFill"].Copy();

            return dt;

            }
```

Listing 4.5: Metodo per aggiornare dei dati mediante l'esecuzione del comando SQL *UPDATE*

```
public void Update(string table, string values, string conditions,
    string source, OracleParameter[] pmCollection)
        {
            Command.CommandText = "UPDATE " + table + " SET " +
                values + " WHERE " + conditions;
            if (pmCollection != null)
            {
                Command.Parameters.AddRange(pmCollection);
                Command.BindByName = true;
            }
            try
            {
                Command.ExecuteNonQuery();
            }
            catch (Exception e) { }
            Console.WriteLine("Data updated into " + table + " from
                " + source);
            }
```

Listing 4.6: Metodo per aggiungere dei dati mediante l'esecuzione del comando SQL *INSERT*

```
    public void Add(string table, string values, string source,
        OracleParameter[] pmCollection)
        {
            Command.CommandText = "INSERT INTO " + table + " " +
                values;
            if (pmCollection != null)
            {
                Command.BindByName = true;
                Command.Parameters.AddRange(pmCollection);
            }
            try
```

```
10              {
11                  Command.ExecuteNonQuery();
12              }
13              catch (Exception ex) { }
14              Console.WriteLine("Data added into " + table + " from "
                    + source);
15              }
```

Listing 4.7: Metodo per eliminare dei dati mediante l'esecuzione del comando SQL *DELETE*

```
1           public void Delete(string table, string conditions, string
                source, OracleParameter[] pmCollection)
2           {
3               Command.CommandText = "DELETE FROM " + table + " WHERE "
                    + conditions;
4               if (pmCollection != null)
5               {
6                   Command.Parameters.AddRange(pmCollection);
7                   Command.BindByName = true;
8               }
9               try
10              {
11                  Command.ExecuteNonQuery();
12              }
13              catch (Exception e) { }
14              Console.WriteLine("Data deleted into " + table + " from
                    " + source);
15          }
```

4.1.2 UnitOfWork

Tale classe è definita all'interno del file **UnitOfWork.cs** nel progetto *SWGA.ServerData* ed implementa l'interfaccia **IUnitOfWork** definita nel progetto *SWGA.ServerCore*. La creazione di un oggetto di tipo UnitOfWOrk comporta la creazione di un'istanza per ogni **Repository** definita nel progetto e richiede un'istanza della classe **CytoContext** per poter istanziare le varie *Repositories*. La seguente struttura della classe UnitOfWork utilizza le **DI** (***Dependency Injection***[7])[2] per istanziare le varie *Repositories*; tale classe è un **IoC Container**[3] (***InversionOfControler***[4] ***Container***).

[2]Design pattern della Programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni.

[3]Componente software responsabile della gestione delle dipendenze

[4]Principio architetturale basato sul concetto di invertire il controllo del flusso di sistema (*Control Flow*) rispetto alla programmazione tradizionale.

Ogni **Controller Web API** crea, nel proprio costruttore, un oggetto di tipo UnitOfWork, in modo da poter interagire liberamente con ogni tipologia di dato memorizzato all'interno del database, utilizzando un'opportuna *Repository* istanziata nell'oggetto di tipo UnitOfWork.

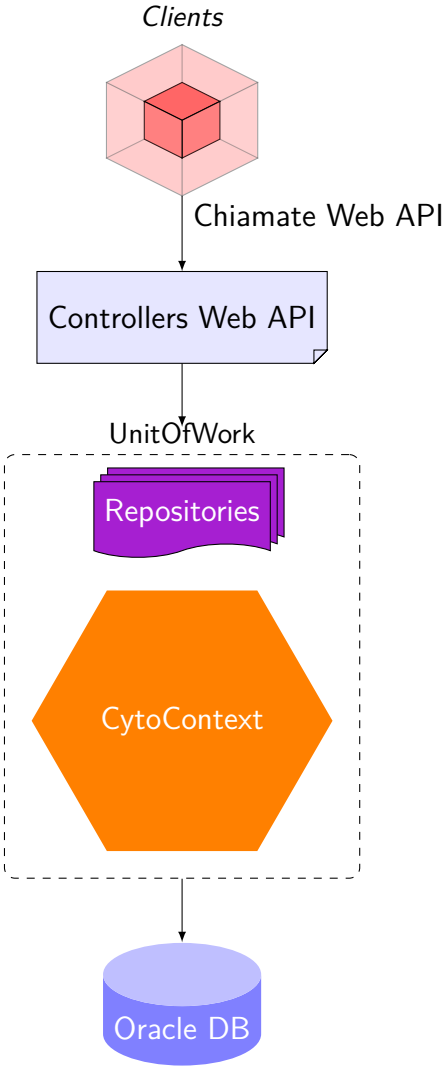
Listing 4.8: Classe UnitOfWork

```
public class UnitOfWork: IUnitOfWork
    {
        private readonly CytoContext cytoContext;
        public IActiveIngredientsRepository
            ActiveIngredientsRepository { get; private set; }
        public IAccesUserRepository AccesUserRepository { get;
            private set; }
        public IUnitRepository UnitRepository { get; private set; }
        public IFlaconsRepository FlaconsRepository { get; private
            set; }
        public IGroupsComponentsRepository
            GroupsComponentsRepository { get; private set; }
        public IFinalContainersRepository FinalContainersRepository
            { get; private set; }
        public IOthersRepository OthersRepository { get; private
            set; }
        public IComponentsRepository ComponentsRepository { get;
            private set; }
        public IColorCodeRepository ColorCodeRepository { get;
            private set;}
        public ITagAnagrafeRepository TagAnagrafeRepository { get;
            private set; }
        public ILanguageRepository LanguageRepository { get; private
            set; }
        public ICodeMapRepository CodeMapRepository {get; private
            set;}
        public IExternalSystemsRepository ExternalSystemsRepository
            { get; private set; }
        public ICodeMapCompRepository CodeMapCompRepository { get;
            private set; }
        public IDeviceDataRepository DeviceDataRepository { get;
            private set; }

        public UnitOfWork(CytoContext cytoContext)
        {
            this.cytoContext = cytoContext;
            this.ActiveIngredientsRepository = new
                ActiveIngredientsRepository(cytoContext);
            this.AccesUserRepository = new
                AccesUserRepository(cytoContext);
            this.UnitRepository = new UnitRepository(cytoContext);
```

```
26              this.FlaconsRepository = new
                    FlaconsRepository(cytoContext);
27              this.GroupsComponentsRepository = new
                    GroupsComponentsRepository(cytoContext);
28              this.FinalContainersRepository = new
                    FinalContainersRepository(cytoContext);
29              this.OthersRepository = new
                    OthersRepository(cytoContext);
30              this.ComponentsRepository = new
                    ComponentsRepository(cytoContext);
31              this.ColorCodeRepository = new
                    ColorCodeRepository(cytoContext);
32              this.TagAnagrafeRepository = new
                    TagAnagrafeRepository(cytoContext);
33              this.LanguageRepository = new
                    LanguageRepostiroy(cytoContext);
34              this.CodeMapRepository=new
                    CodeMapRepository(cytoContext);
35              this.CodeMapCompRepository = new
                    CodeMapCompRepository(cytoContext);
36              this.ExternalSystemsRepository = new
                    ExternalSystemsRepository(cytoContext);
37              this.DeviceDataRepository = new
                    DeviceDataRepository(cytoContext);
38          }
39
40          public void Dispose()
41          {
42              throw new NotImplementedException();
43          }
44      }
```

È possibile schematizzare l'utilizzo della **UnitOfWork** nel seguente modo:



4.1.3 Repositories

Le **Repositories** definite nell'applicativo sono memorizzate all'interno della cartella *Repository* del progetto *SWGA.ServerData*. Per ogni entità utilizzata all'interno dell'applicativo è stata definita una specifica *Repository* che gestisce lo scambio di quella determinata tipologia di dati con il database.

È stato definito un **Repository Pattern** per organizzare al meglio la definizione e l'utilizzo delle *Repositories* coinvolte nel progetto.

All'interno della cartella *Repository* del progetto *SWGA.ServerCore* è presente

l'interfaccia **IRepository** richiedente un generico parametro di tipo **TEntity**. Tale interfaccia è implementata dalla classe generica **Repository** presente nella cartella *Repository* del progetto *SWGA.ServerData*; questa classe presenta un attributo di tipo **CytoContext** inizializzato nel suo costruttore.

Listing 4.9: Interfaccia IRepository

```
public interface IRepository<TEntity> where TEntity: class
    {

        IEnumerable<TEntity> SelectAll ();
        void Add(TEntity entity);
        void Delete(TEntity entity);
        void Update(TEntity entity);
        TEntity FindById(int Id);
    }
```

Listing 4.10: Attributi e costruttore della classe Repository

```
protected readonly CytoContext Context;

        public Repository(CytoContext context)
        {
            Context = context;

        }
```

Per ogni specifica *Repository* è stata definita un'apposita interfaccia che implementa l'interfaccia IRepository con un parametro del medesimo tipo di oggetto gestito dalla *Repository* implementante.

Ogni specifica *Repository* può definire delle proprie funzioni differenti da quelle definite nelle altre *Repositories* (elenco delle *Repositories* in figura 4.1).

Listing 4.11: Interfaccia IActiveIngredientsRepository per la manipolazione di dati di tipo Active Ingredients

```
public interface IActiveIngredientsRepository :
    IRepository<ActiveIngredients>
    {
        IEnumerable<ActiveIngredients> SelectAll(string filter);
        IEnumerable<ActiveIngredients> GetFromID(int id_flacon);

    }
```

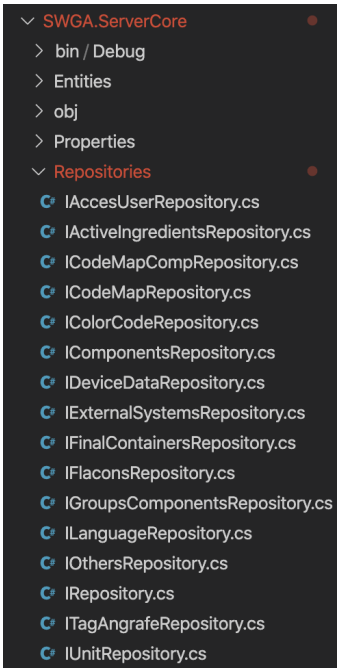


Figura 4.1: Interfacce di tutte le *Repositories* definite

Ciascuna delle specifiche interfacce è implementata dalla corrispettiva *Repository* definita nel progetto *SWGA.ServerData* ed eredita dalla classe generica **Repository<TEntity>**. In questo modo ogni specifica *Repository* ha accesso all'attributo, della classe padre, di tipo **CytoContext** ed è in grado di effettuare autonomamente delle interazioni con il database.

Listing 4.12: Parte di codice dell' ActiveIngredientsRepository. Si mostra il metodo *SelectAll* utilizzato per ottenere la lista degli ActiveIngredients dal database. Parte del codice è stato modificato per oscurare le informazioni riguardanti la struttura del database dell'impresa.

```
1  public class ActiveIngredientsRepository :
       Repository<ActiveIngredients>, IActiveIngredientsRepository
2      {
3          private string table = "ANAGRAFE_GRUPPI_COMPONENTI";
4
5          public ActiveIngredientsRepository(CytoContext context)
6              : base(context)
7          {
8          }
9
```

```
10          public IEnumerable<ActiveIngredients> SelectAll(string
                filter)
11          {
12              List<ActiveIngredients> list = new
                    List<ActiveIngredients>();
13              Context.open();
14              DataTable dt;
15
16              if (filter==null)
17             dt=Context.GetAll("VALORI DA VISUALIZZARE",
18                          "TABLE JOIN",null);
19              else
20                 dt=Context.GetAll("VALORI DA VISUALIZZARE",
21                          "TABLE JOIN CON FILTRO",null);
22
23              for (int i = 0; i < dt.Rows.Count; i++)
24              {
25                  ActiveIngredients ai = new ActiveIngredients(Convert.
26                  ToDecimal(dt.Rows[i]["ID"]));
27                  ai.Name = dt.Rows[i]["NOME"].ToString();
28                  ai.Unit = new
                        Unit(Convert.ToDecimal(dt.Rows[i]["ID_UNITA"]));
29                  ai.Unit.Name = dt.Rows[i]["NOME_UNITA"].ToString();
30                  ai.Color = ServerDataUtilities.
31                ColorFromWin32ToHtml(dt.Rows[i]["COLORE"].ToString());
32                       ai.Flag_Cytostatic =
                             dt.Rows[i]["FLAG_CYTOSTATIC"].ToString();
33                  list.Add(ai);
34              }
35
36              Context.close();
37              return list;
38          }
39          ...
40
41      }
```

4.2 Chiamate Web API

Le **Chiamate Web API** sono gestite dai **Controllers Web API**[5] definiti all'interno della cartella *Controllers* del progetto *SWGA.WebApiService*. Come per le *Repositories*, per ogni entità gestita all'interno dell'applicativo è stato definito

[5]Dettagli in appendice D.2

un apposito Controller Web API che effettua le interazione con i dati mediante la specifica *Repository* associata (elenco dei Controllers Web API in figura 4.2).

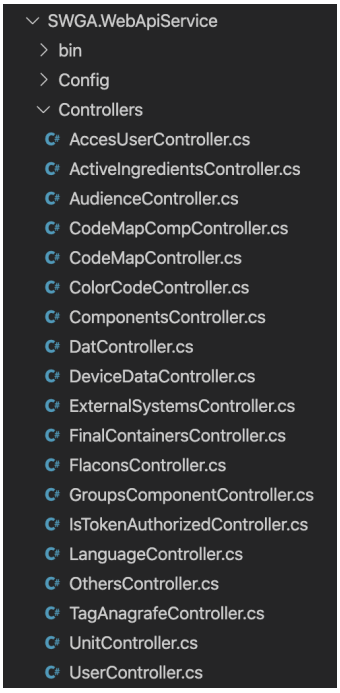


Figura 4.2: Controllers Web API

Tutti i *Controllers* ereditano dalla classe **ApiController** e definiscono un **RoutePrefix** grazie al quale possono essere accessibili dai *clients*.

L'ActiveIngredientsController presenta il seguente RoutePrefix:

```
1 [RoutePrefix("api/ActiveIngredients")]
```

In questo modo un *client* può accedere ai metodi esposti dal *Controller* digitando nel *Web Browser* l'URL con il quale si è registrato il *Web Server* seguito da "*/api/ActiveIngredients*".

Tutti i *Controllers* definiti istanziano una propria UnitOfWork con un nuovo oggetto CytoContext; l'istanza della classe UnitOfWork è utilizzata per accedere alle *Repositories* ed effettuare le interazioni con i dati. Inoltre tutti i *Controllers* definiscono dei metodi GET, PUT, POST, DELETE utilizzati rispettivamente per la visualizzazione, la modifica, l'inserimento e la cancellazione dei dati; ogni funzione richiama il corrispondente metodo della *Repository* per svolgere la rispettiva azio-

ne. Tali metodi possono richiedere dei parametri che sono prelevati dalla richiesta HTTP ricevuta.

Listing 4.13: ActiveIngredientsController

```
[RoutePrefix("api/ActiveIngredients")]
public class ActiveIngredientsController : ApiController
    {
         private readonly IUnitOfWork unitOfWork;

        public ActiveIngredientsController()
        {
            this.unitOfWork = new UnitOfWork(new CytoContext());
        }

        public IEnumerable<ActiveIngredients> Get()
        {

            return
               unitOfWork.ActiveIngredientsRepository.SelectAll(null);

        }

        [HttpGet, Route("{id_flacon}/GetActiveIngredients")]
        public IEnumerable<ActiveIngredients> Get(int id_flacon)
        {
            return unitOfWork.ActiveIngredientsRepository.
            GetFromID(id_flacon);
        }

        [HttpGet, Route("{filter}/ActiveIngredients")]
        public IEnumerable<ActiveIngredients> OrderBy(string filter)
        {
            return unitOfWork.ActiveIngredientsRepository.
            SelectAll(filter);
        }

        [HttpPost]
        public IHttpActionResult Add(string name,string color,string
           id_unit, string flag_cytostatic)
        {
            ActiveIngredients act = new ActiveIngredients();
            act.Name = name;
            act.Unit = new Unit(Convert.ToDecimal(id_unit));
            act.Color = Utilities.colorImplementation(color);
            act.Flag_Cytostatic = flag_cytostatic;
            act.Creator = new User(RequestContext.Principal.Identity.
            Name.ToString());

```

```
43              unitOfWork.ActiveIngredientsRepository.Add(act);
44              return Ok("true");
45          }
46
47
48
49          [HttpPut]
50          public IHttpActionResult Update(string id, string name,
                string color, string id_unit, string flag_cytostatic)
51          {
52              if (id != null && id!= "")
53              {
54                  ActiveIngredients act = new
                        ActiveIngredients(Convert.ToDecimal(id));
55                  act.Name = name;
56                  act.Unit = new Unit(Convert.ToDecimal(id_unit));
57                  act.Color = Utilities.colorImplementation(color);
58                  act.Flag_Cytostatic = flag_cytostatic;
59                  act.Creator = new
                        User(RequestContext.Principal.Identit.
60                      Name.ToString());
61
62                  unitOfWork.ActiveIngredientsRepository.Update(act);
63
64                  return Ok("true");
65              }
66              else
67                  return Ok("false");
68          }
69
70          [HttpDelete, Route("{paramid}")]
71          public IHttpActionResult Delete(string paramid)
72          {
73              ActiveIngredients act = new
                    ActiveIngredients(Convert.ToDecimal(paramid));
74              unitOfWork.ActiveIngredientsRepository.Delete(act);
75              return Ok("true");
76          }
77
78      }
```

4.3 Autenticazione

La gestione dell'autenticazione è una caratteristica fondamentale della versione Web di SWGA che la differisce dalla sua versione Desktop. L'applicativo utiliz-

za il protocollo **OAuth2**[6] (*Open Authentication*) mediante una **JWT**[9] **Bearer Authentication**: l'autenticazione degli utenti è effettuata mediante un **JSON Web Token**[7] univoco.

Quando l'utente accede all'applicazione si controlla se è in possesso di un token valido; in caso affermativo l'utente ha la libertà di navigare liberamente all'interno dell'applicativo, allegando il proprio token di accesso in ogni richiesta HTTP effettuata al *back-end*. Se l'utente non possiede un token valido deve effettuare una nuova richiesta di autenticazione attraverso le credenziali di accesso aziendali (dominio Loccioni). La limitazione della navigazione nell'applicativo ai soli utenti autenticati è stata resa possibile mediante l'aggiunta del seguente attributo ad ogni Controller che restituisce dati fondamentali per il funzionamento dell'app:

```
1 [Authorize(Roles = "user")]
```

Nel caso in cui l'utente sia autenticato, il suo token viene aggiunto alla categoria **user**; solamente gli utenti che possiedono un token di accesso di tipo **user** possono effettuare **Chiamate Web API** e quindi utilizzare l'applicazione.

L'*OWIN Self-host Web API Server* è stato configurato per svolgere il ruolo di **OAuth Server**[3][8]. Il successivo schema in figura 4.3 descrive l'iter di comunicazione tra *client* e *server*.

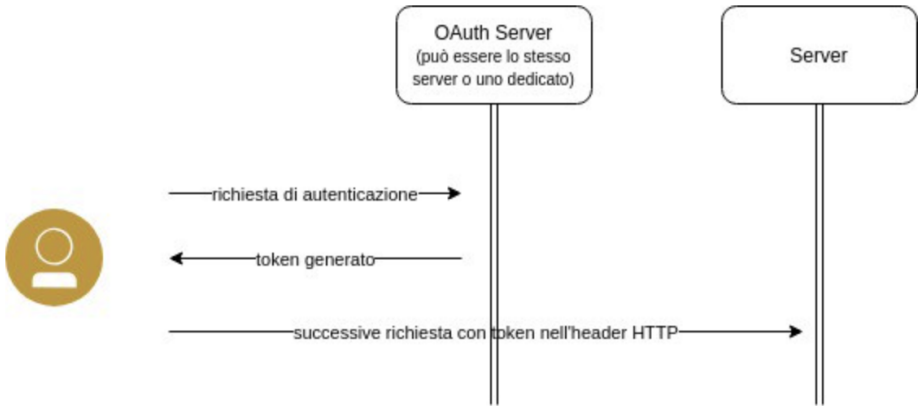


Figura 4.3: Schema di comunicazione client-server in presenza dell'OAuth Server

[6]Protocollo di rete standard che consente un'autorizzazione API sicura.

[7]Il JSON Web Token (JWT) è uno standard open (RFC 7519) che definisce uno schema in formato JSON per lo scambio di informazioni tra vari servizi.

[8]Server che gestisce l'autenticazione degli utenti mediante l'utilizzo di un token.

4.3.1 Implementazione

Inizialmente è stato necessario configurare il protocollo **OAuth** durante l'avvio del **WebApiService**. All'interno del metodo *Configuration* del WebApiService è stato invocato il metodo per la configurazione del protocollo di autenticazione attraverso la seguente istruzione:

```
ConfigureOAuth(app);
```

Listing 4.14: Metodo per la configurazione dell'OAuth

```
public void ConfigureOAuth(IAppBuilder app)
      {
          OAuthAuthorizationServerOptions OAuthServerOptions = new
              OAuthAuthorizationServerOptions()
          {
              AllowInsecureHttp = true,
              TokenEndpointPath = new PathString("/token"),
              AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
              Provider = new TokenAuthProvider(),
              AccessTokenFormat = new CustomJwtFormat("localhost")
          };

          app.UseOAuthAuthorizationServer(OAuthServerOptions);
          var issuer = "localhost";
          var audience =
              ConfigurationManager.AppSettings["clientID"];
          var secret =
              TextEncodings.Base64Url.Decode(ConfigurationManager.
          AppSettings["base64Secret"]);

          app.UseJwtBearerAuthentication(
              new JwtBearerAuthenticationOptions
              {
                  AuthenticationMode = AuthenticationMode.Active,
                  AllowedAudiences = new[] { audience },

                  IssuerSecurityTokenProviders = new
                      IIssuerSecurityTokenProvider[]
                  {
                      new SymmetricKeyIssuerSecurityTokenProvider
                      (issuer, secret)
                  }
              });
      }
```

TokenAuthProvider e **CustomJwtFormat** sono delle classi definite per l'implementazione dell'autenticazione (sono analizzate successivamente nella trattazio-

ne).

In seguito si è dovuto configurare l'*Authorization Server* per abilitare la registrazione dei Server sorgenti delle richieste di autenticazione (chiamati *Audience*). Questa fase è molto importante in quanto si necessita di una modalità per individuare chi sta richiedendo il **JWT Token** in modo che l'*Authentication Server* abbia la possibilità di generare il **JWT Token** esclusivamente per l'Audience richiedente. È stata aggiunta la classe **Audience** e **AudienceModel** rispettivamente all'interno della cartella *Entities* del progetto *SWGA.ServerCore* e nella cartella *Models* del progetto *SWGA.WebApiService*

Listing 4.15: Classe Audience

```
public class Audience
    {
        public string ClientId { get; set; }
        public string Base64Secret { get; set; }
        public string Name { get; set; }
    }
```

Listing 4.16: Classe AudienceModel

```
public class AudienceModel
    {
        public string Name { get; set; }
    }
```

Successivamente si è definita all'interno della cartella *Provider* del progetto *SWGA.WebApiService* la classe **AudiencesStore** che ha il compito di:

- Aggiungere una nuova Audience che ha effettuato la richiesta di autenticazione.
- Individuare la presenza di un Audience che ha già richiesto il token di accessso, attraverso il **Client ID**.

Listing 4.17: Classe AudiencesStore

```
public static class AudiencesStore
    {
        public static ConcurrentDictionary<string, Audience>
            AudiencesList = new ConcurrentDictionary<string,
            Audience>();

        static AudiencesStore()
        {
            AudiencesList.TryAdd(ConfigurationManager.
            AppSettings["clientID"],
```

```
                                new Audience
                                {
                                    ClientId = ConfigurationManager.
                                    AppSettings["clientID"],
                                    Base64Secret =
                                        ConfigurationManager.
                                    AppSettings["base64Secret"],
                                    Name = ConfigurationManager.
                                    AppSettings["name"]
                                });
        }

        public static Audience AddAudience(string name)
        {
            var clientId = Guid.NewGuid().ToString("N");
            var key = new byte[32];
            RNGCryptoServiceProvider.Create().GetBytes(key);
            var base64Secret = TextEncodings.Base64Url.Encode(key);

            Audience newAudience = new Audience { ClientId =
                clientId, Base64Secret = base64Secret, Name = name };
            AudiencesList.TryAdd(clientId, newAudience);
            return newAudience;
        }

        public static Audience FindAudience(string clientId)
        {
            Audience audience = null;
            if (AudiencesList.TryGetValue(clientId, out audience))
            {
                return audience;
            }
            return null;
        }
    }
```

Si è aggiunto un **AudienceController** che svolge il ruolo di *end point* per la registrazione di una nuova Audience.

Listing 4.18: AudienceController

```
[RoutePrefix("api/audience")]
    public class AudienceController : ApiController
    {
        [Route("")]
        public IHttpActionResult Post(AudienceModel audienceModel)
        {
            if (!ModelState.IsValid)
            {
```

```
                return BadRequest(ModelState);
            }
            Audience newAudience =
                AudiencesStore.AddAudience(audienceModel.Name);

            return Ok<Audience>(newAudience);
        }
    }
```

Quando un *client* effettua una chiamata POST al Controller sopra definito, l' *Authorization Server* genera un **Client ID** e una chiave simmetrica per la decodifica.

A questo punto è stato necessario implementare la classe **TokenAuthProvider** all'interno della cartella *Provider* del progetto *SWGA.WebApiService*, responsabile della generazione di un **JSON Web Token** quando il richiedente effettua una chiamata POST all'URL con il quale si è registrato il *Web Server + "/token"*.

Listing 4.19: Classe TokenAuthProvider

```
public class TokenAuthProvider : OAuthAuthorizationServerProvider
    {
        public static User ContextUser { get; set; }
        private static Logger logger =
            LogManager.GetCurrentClassLogger();
        public override Task ValidateClientAuthentication
        (OAuthValidateClientAuthenticationContext context)
        {
            string clientId = string.Empty;
            string clientSecret = string.Empty;
            string symmetricKeyAsBase64 = string.Empty;

            if (!context.TryGetBasicCredentials(out clientId, out
                clientSecret))
            {
                context.TryGetFormCredentials(out clientId, out
                    clientSecret);
            }

            if (context.ClientId == null)
            {
                context.SetError("invalid_clientId", "client_Id is
                    not set");
                return Task.FromResult<object>(null);
            }
            var audience =
                AudiencesStore.FindAudience(context.ClientId);

            if (audience == null)
```

```
25              {
26                  context.SetError("invalid_clientId",
                        string.Format("Invalid client_id '{0}'",
                        context.ClientId));
27                  return Task.FromResult<object>(null);
28              }
29              context.Validated();
30              return Task.FromResult<object>(null);
31          }
32
33          public override Task GrantResourceOwnerCredentials
34          (OAuthGrantResourceOwnerCredentialsContext context)
35          {
36              context.OwinContext.Response.Headers.Add
37              ("Access-Control-Allow-Origin", new[] { "*" });
38              try
39              {
40                  PrincipalContext pc = new
                        PrincipalContext(ContextType.Domain,
                        "loccioni.com");
41                  using (pc)
42                  {
43                      bool isValid =
                            pc.ValidateCredentials(context.UserName,
                            context.Password);
44                      if (!isValid)
45                      {
46                        context.SetError("invalid_grant", "The user
                              name or password is incorrect.");
47                          return Task.FromResult<object>(null);
48                      }
49                  }
50              }
51              catch (Exception ex)
52              {
53                  logger.Error(ex);
54                  return Task.FromResult<object>(null);
55              }
56              var identity = new
                    ClaimsIdentity(context.Options.AuthenticationType);
57              identity.AddClaim(new Claim(ClaimTypes.Name,
                    context.UserName));
58              identity.AddClaim(new Claim(ClaimTypes.Role, "user"));
59              identity.AddClaim(new Claim("username",
                    context.UserName));
60              var props = new AuthenticationProperties(new
                    Dictionary<string, string>
61                  {
62                      {
```

```
                        "audience", (context.ClientId == null) ?
                            string.Empty : context.ClientId
                    }
                });
            var ticket = new AuthenticationTicket(identity, props);
            context.Validated(ticket);
            return Task.FromResult<object>(null);
        }
    }
```

Tale classe implementa i seguenti metodi:

- **ValidateClientAuthentication**: metodo responsabile del controllo della già avvenuta registrazione dell'Audience mittente. Tale controllo è effettuato mediante il **Client ID**.
- **GrantResourceOwnerCredential**: metodo responsabili della validazione delle credenziali di accesso utilizzate nell'autenticazione. Tali credenziali sono inviate all'interno della richiesta HTTP e devono essere valide all'interno del dominio **loccioni.com** per garantire l'accesso dell'utente richiedente.

Infine si è definita la classe **CustomJwtFormat** all'interno della cartella *Provider* del progetto *SWGA.WebApiService*, responsabile della generazione del **JWT Token**.

Listing 4.20: Classe CustomJwtFormat

```
public class CustomJwtFormat :
    ISecureDataFormat<AuthenticationTicket>
    {
        private const string AudiencePropertyKey = "audience";
        private readonly string _issuer = string.Empty;
        public CustomJwtFormat(string issuer)
        {
            _issuer = issuer;
        }
        public string Protect(AuthenticationTicket data)
        {
            if (data == null)
            {
                throw new ArgumentNullException("data");
            }
            string audienceId =
                data.Properties.Dictionary.ContainsKey
            (AudiencePropertyKey) ?
                data.Properties.Dictionary[AudiencePropertyKey] :
                null;
```

```
17              if (string.IsNullOrWhiteSpace(audienceId)) throw new
                    InvalidOperationException
18              ("AuthenticationTicket.Properties does not include
                    audience");
19              Audience audience =
                    AudiencesStore.FindAudience(audienceId);
20              string symmetricKeyAsBase64 = audience.Base64Secret;
21              var keyByteArray =
                    TextEncodings.Base64Url.Decode(symmetricKeyAsBase64);
22              var signingKey = new
                    HmacSigningCredentials(keyByteArray);
23              var issued = data.Properties.IssuedUtc;
24              var expires = data.Properties.ExpiresUtc;
25              var token = new JwtSecurityToken(_issuer, audienceId,
                    data.Identity.Claims, issued.Value.UtcDateTime,
                    expires.Value.UtcDateTime, signingKey);
26              var handler = new JwtSecurityTokenHandler();
27              var jwt = handler.WriteToken(token);
28
29              return jwt;
30          }
31          public AuthenticationTicket Unprotect(string protectedText)
32          {
33              throw new NotImplementedException();
34          }
35      }
```

4.3.2 Testing

Si è testato l'*OAuth Server* mediante l'utilizzo di **PostMan**: effettuando una richiesta HTTP GET all'URL "*http://localhost:5001/token*" ed inserendo le credenziali di accesso dell'utente all'interno della suddetta richiesta (configurazione della richiesta in figura 4.4) si ottiene, da parte del *Server*, la seguente risposta JSON contenente l'**Access Token** necessario ad effettuare l'autenticazione.

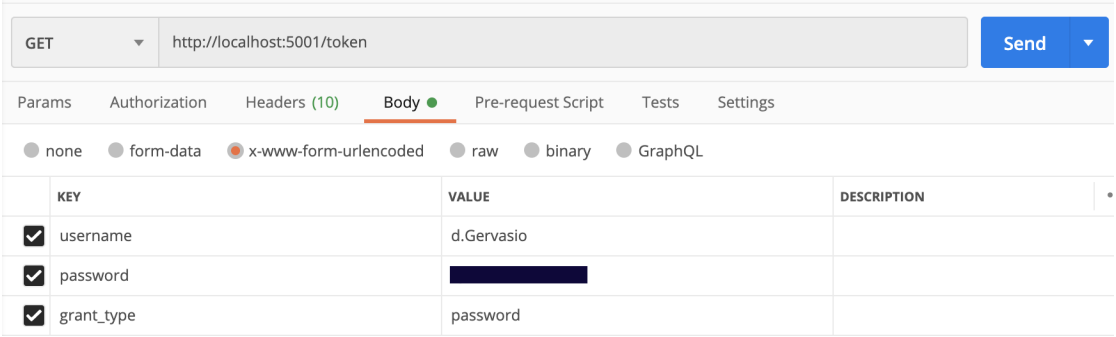


Figura 4.4: Configurazione delle credenziali di accesso nella richiesta HTTP per il testing dell'autenticazione mediante PostMan

Listing 4.21: Risposta JSON ottenuta dopo aver effettuato una richiesta di autenticazione

```
{
    "access_token":
       "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfbmFtZSI6ImQu
    R2VydmFzaW8iLCJyb2xlIjoidXNlciIsInVzZXJuYW1lIjoiZC5HZXJ2YXNpb
    yIsImlzcyI6ImxvY2FsaG9zdCIsImF1ZCI6IjA5OTE1M2MyNjI1MTQ5YmM4Z
    WNiM2U4NWUwM2YwMDIyIiwiZXhwIjoxNTkxODkyMTAzLCJuYmYiOjE1OTE4MDU3
    MDN9.Wxc-nf0M8SoVFZ69wkZQozh3R6AxLtt4np4aLH7sQLQ",
    "token_type": "bearer",
    "expires_in": 86399
}
```

4.4 Esportazione file .dat

SWGA permette di esportare esclusivamente i file **.dat** dei **Componenti** (tale funzione non è abilitata per i **Gruppi Componente**). Sono disponibili due modalità di esportazione differenti:

- L'utente può richiedere l'esportazione di un singolo Componente selezionato all'interno della lista dei Componenti, oppure può richiedere l'esportazione di un singolo Componente quando l'utente è in fase di modifica di quest'ultimo.
- L'utente può richiedere l'esportazione di un'intera categoria di Componenti (ad esempio tutti i Componenti di tipo **Drugs**)

Il click sul pulsante "*Export all*" o "*Export dat*" comporta il download, all'interno del *Web Browser*, delle informazioni richieste tramite un file compresso .zip[9].

4.4.1 Implementazione

Sono state definite le classi **DatDownloader** e **DatExportStreamWriter** all'interno della cartella *Services* del progetto *SWGA.WebApiService*. Le istanze delle suddette classi sono utilizzate per effettuare l'esportazione del file .dat di un singolo Componente.

- **DatDownloader**: la sua istanza permette di prelevare dal database le informazioni relative a tutti gli altri Componenti, Gruppi Componente e Principi Attivi associati al Componente per il quale è stata richiesta l'esportazione del file .dat.
- **DatExportStreamWriter**: la sua istanza è utilizzata all'interno della classe DatDownloader per scrivere all'interno del file .dat tutte le informazioni del Componente.

Infine è stato definito un apposito Controller per permettere all'utente di effettuare la richiesta di esportazione. È stato creato il **DatController** il quale possiede un metodo pubblico *Get* per la richiesta di esportazione; tale metodo richiede il parametro *id_component* ed ha un comportamento differente in base al valore assunto dal suddetto parametro:

- Se *id_component* contiene l'ID di un Componente per il quale è richiesta l'esportazione, allora si soddisfa la richiesta eseguendo il seguente codice:

Listing 4.22: Parte di codice del metodo Get del DatController utile all'esportazione del file .dat di un singolo Componente

```
1  Components toConvert =
       this.unitOfWork.ComponentsRepository.SelectAll
2  ("ANAGRAFE_COMPONENTI.ID=" +
       id_component).ToList<Components>().ElementAt(0);
3              DatDownloader downloader = new DatDownloader();
4              downloader.Download(toConvert);
5              string startPath = downloader.tempDirectory;
6              var path =
                   Directory.GetParent(startPath).FullName +
                   "\\dat"+toConvert.ID+".zip";
7              if (File.Exists(path))
```

[9]Formato libero di compressione dei dati lossless.

```
8                   File.Delete(path);
9                 ZipFile.CreateFromDirectory(startPath, path);
10
11                result = new
                      HttpResponseMessage(HttpStatusCode.OK);
12                var stream = new FileStream(path,
                      FileMode.Open);
13                result.Content = new StreamContent(stream);
14                result.Content.Headers.ContentType =
15                new MediaTypeHeaderValue
16                ("application/octet-stream");
17                result.Content.Headers.ContentDisposition = new
                      ContentDispositionHeaderValue("attachment")
18                {
19                    FileName = "dat_" + id_component + ".zip"
20                };
21                return result;
```

- Se è stata richiesta l'esportazione di un'intera categoria di Componenti, il valore di *id_component* è "*FLAG_*"+"*NOME_CATEGORIA*" (ad esempio "*FLAG_SOLVENTE*" per l'esportazione di tutti i Solventi). L'esportazione viene soddisfatta eseguendo il seguente codice:

  Listing 4.23: Parte di codice del metodo Get del DatController utile all'esportazione dei file .dat di una specifica categoria di Componenti

```
1     List < Components > toConvertList=
          this.unitOfWork.ComponentsRepository.
2     SelectAll(id_component + "='Y'").ToList<Components>();
3               DatDownloader downloader = new DatDownloader();
4               string startPath = downloader.tempDirectory;
5               var path =
                    Directory.GetParent(startPath).FullName +
                    "\\dat.zip";
6               foreach (Components c in toConvertList)
7               {
8                   downloader.Download(c);
9
10              }
11              if (File.Exists(path))
12                  File.Delete(path);
13              ZipFile.CreateFromDirectory(startPath, path);
14              result = new
                    HttpResponseMessage(HttpStatusCode.OK);
15              var stream = new FileStream(path,
                    FileMode.Open);
16              result.Content = new StreamContent(stream);
17              result.Content.Headers.ContentType =
18                  new MediaTypeHeaderValue
```

```
19                      ("application/octet-stream");
20                  result.Content.Headers.ContentDisposition = new
                        ContentDispositionHeaderValue("attachment")
21                  {
22                      FileName = "dat.zip"
23                  };
24                  return result;
```

Il codice completo del **DatController** è il seguente:

Listing 4.24: Codice completo del DatController

```
1   [RoutePrefix("api/Dat")]
2     [Authorize(Roles = "user")]
3     public class DatController : ApiController
4     {
5         private readonly IUnitOfWork unitOfWork;
6         public DatController()
7         {
8             this.unitOfWork = new UnitOfWork(new CytoContext());
9         }
10        [HttpGet]
11        public HttpResponseMessage Get(string id_component)
12        {
13            var result = new HttpResponseMessage();
14            if (id_component != "FLAG_FARMACO" &&
                  id_component!="FLAG_SOLVENTE" &&
                  id_component!="FLAG_ELASTOMERO" &&
                  id_component!="FLAG_FLACONE" &&
                  id_component!="FLAG_SPIKE" &&
                  id_component!="FLAG_TRANSFER_SET" &&
                  id_component!="FLAG_SIRINGA")
15            {
16                Components toConvert =
                      this.unitOfWork.ComponentsRepository.SelectAll
17                ("ANAGRAFE_COMPONENTI.ID=" +
                      id_component).ToList<Components>().ElementAt(0);
18                DatDownloader downloader = new DatDownloader();
19                downloader.Download(toConvert);
20                string startPath = downloader.tempDirectory;
21                var path = Directory.GetParent(startPath).FullName +
                      "\\dat"+toConvert.ID+".zip";
22                if (File.Exists(path))
23                    File.Delete(path);
24                ZipFile.CreateFromDirectory(startPath, path);
25
26                result = new HttpResponseMessage(HttpStatusCode.OK);
27                var stream = new FileStream(path, FileMode.Open);
28                result.Content = new StreamContent(stream);
```

```
29                  result.Content.Headers.ContentType =
30                  new MediaTypeHeaderValue("application/octet-stream");
31                  result.Content.Headers.ContentDisposition = new
                        ContentDispositionHeaderValue("attachment")
32                  {
33                      FileName = "dat_" + id_component + ".zip"
34                  };
35                  return result;
36              }else
37              {
38                  List < Components > toConvertList=
                        this.unitOfWork.ComponentsRepository.
39                  SelectAll(id_component +
                        "='Y'").ToList<Components>();
40                  DatDownloader downloader = new DatDownloader();
41                  string startPath = downloader.tempDirectory;
42                  var path = Directory.GetParent(startPath).FullName +
                        "\\dat.zip";
43                  foreach (Components c in toConvertList)
44                  {
45                      downloader.Download(c);
46
47                  }
48                  if (File.Exists(path))
49                      File.Delete(path);
50                  ZipFile.CreateFromDirectory(startPath, path);
51                  result = new HttpResponseMessage(HttpStatusCode.OK);
52                  var stream = new FileStream(path, FileMode.Open);
53                  result.Content = new StreamContent(stream);
54                  result.Content.Headers.ContentType =
55                      new MediaTypeHeaderValue
56                      ("application/octet-stream");
57                  result.Content.Headers.ContentDisposition = new
                        ContentDispositionHeaderValue("attachment")
58                  {
59                      FileName = "dat.zip"
60                  };
61                  return result;
62              }
63          }
64      }
```

Il metodo *Get* del **DatController** è anche responsabile della compressione della cartella temporanea nella quale le classi **DatDownloader** e **DatExportStream-Writer** hanno memorizzato i file richiesti. Si mostra un esempio di esportazione in figura 4.5.

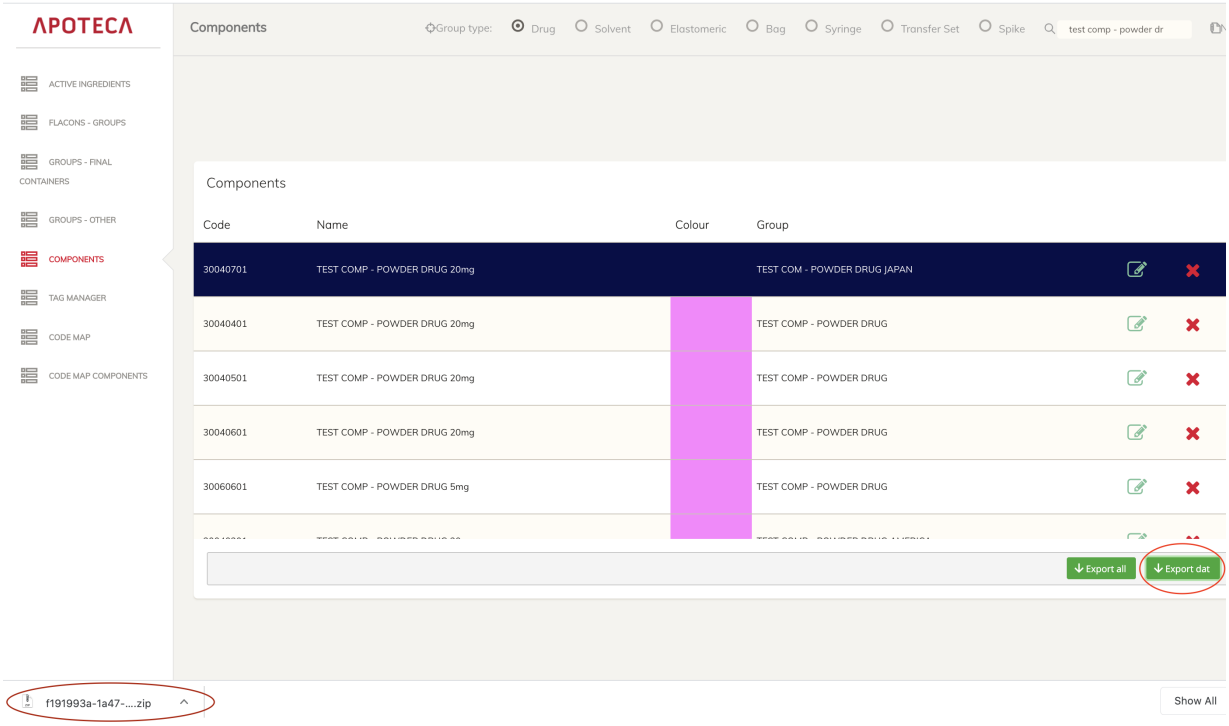


Figura 4.5: Esportazione del file .dat di un Componente di tipo Drugs

5. Front-end

In tale capitolo si analizzano le principali caratteristiche del *front-end* contenuto all'interno del progetto *SWGA.WebClient*. Le caratteristiche prese in considerazione sono:

- Organizzazione interna dei **file** contenente il codice dell'applicativo.
- Gestione dell'**autenticazione**.
- Gestione delle **Chiamate Web API**.
- **Template**[1] grafico utilizzato per la creazione della UI (*User Interface*[2]).

5.1 Organizzazione interna dei file

I *file* del *front-end* sono stati memorizzati all'interno del progetto *SWGA.WebClient* in *folders* differenti in base alla funzionalità e allo scopo delle istruzioni di codice contenute nei vari *file*. All'interno del percorso *src/app* del progetto *SWGA.WebClient* sono presenti le seguenti *folders*:

- **Entities**: contiene il codice di tutte le entità utilizzate all'interno del *front-end*. La maggior parte di queste classi è stata anche definita all'interno del *back-end* al fine di facilitare lo scambio delle informazioni mediante le chiamate Web API. Si visualizza il contenuto della cartella *Entities* in figura 5.1.

[1] Un template è un punto di partenza per creare un particolare tipo di documento informatico, ovvero un file utilizzato come modello per creare ciò di cui si ha bisogno.

[2] L'interfaccia utente, anche conosciuta come UI, è un'interfaccia uomo-macchina, ovvero ciò che si frappone tra una macchina e un utente, consentendone l'interazione reciproca.

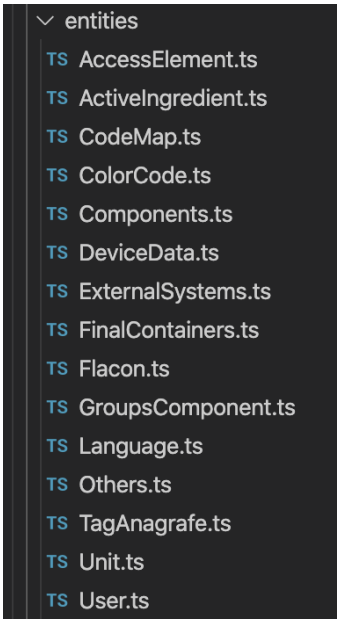


Figura 5.1: File contenuti all'interno della cartella entities, nel percorso src/app del progetto SWGA.WebClient

- **Guards**: al suo interno è definita una **RouteGuard**, ovvero un'interfaccia che permette di intercettare la navigazione dell'utente su un determinato percorso URL del *front-end*. La **RouteGuard** ha il compito di consentire o meno l'accesso a tale percorso in base se l'utente verifica una determinata condizione.[3] Si visualizza il contenuto della cartella *Guards* in figura 5.2.

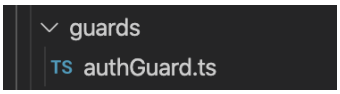


Figura 5.2: File contenuti all'interno della cartella guard, nel percorso src/app del progetto SWGA.WebClient

[3]La classe AuthGuard è analizzata dettagliatamente più avanti nella trattazione.

- **Services**: contiene tutti i servizi utilizzati all'interno del *front-end*. Si visualizza il contenuto della cartella *Services* in figura 5.3.

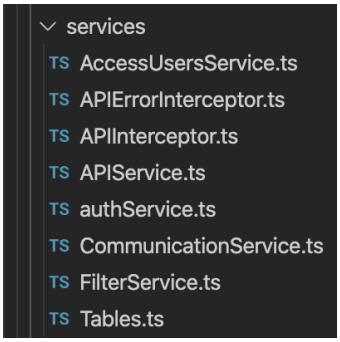


Figura 5.3: File contenuti all'interno della cartella services, nel percorso src/app del progetto SWGA.WebClient

- **Views**: contiene le pagine dell'applicativo (*View Components*) che visualizza l'utente durante la navigazione. All'interno della suddetta cartella sono presenti varie cartelle, una per ogni voce del menù laterale visualizzato nella UI (ad esempio *activeIngredients*, *flaconsGroups* ecc.). Si visualizza il contenuto della cartella *Views* in figura 5.4.

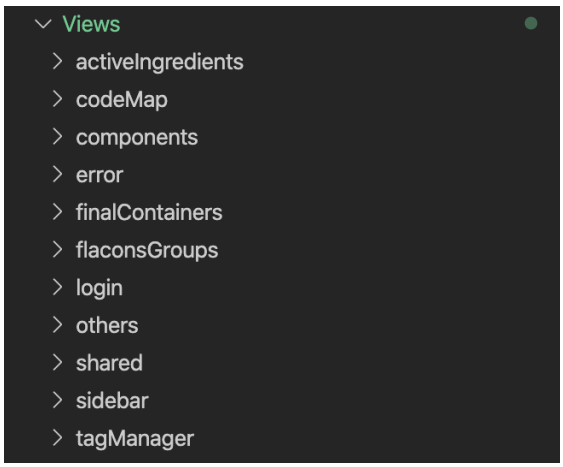


Figura 5.4: Cartelle contenute all'interno della cartella views, nel percorso src/app del progetto SWGA.WebClient

In ogni cartella sono presenti i *file* per la definizione della *View Component* principale da visualizzare (ad esempio i file per la visualizzazione della pagina

Active Ingredients) e due sotto cartelle contenenti i file per la definizione delle pagine di creazione e modifica dell'entità in questione (ad esempio i file per la visualizzazione delle pagine di creazione e modifica di un *Active Ingredient*, figura 5.5).

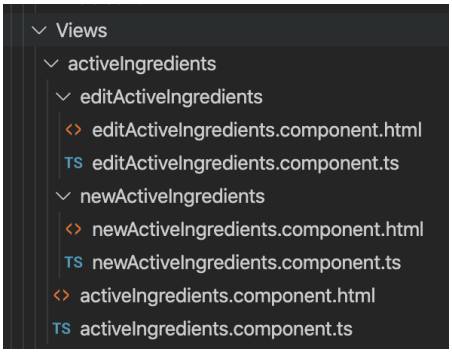


Figura 5.5: File e cartelle contenute all'interno della cartella views/activeIngredients, nel percorso src/app del progetto SWGA.WebClient

La definizione di una *View Component* richiede due file differenti:

- Il *file* **.html** contiene la UI (*User Interface*) del componente da visualizzare.
- Il *file* **.ts**[4] contiene la logica implementata sulla UI del componente. All'interno di questo *file* può essere richiamato l'utilizzo di servizi e possono essere invocate delle chiamate Web API.

All'interno della cartella **Views** sono presenti ulteriori *folders* contenenti *file* che hanno ruoli differenti:

- **error**: contiene la definizione della *View Component* visualizzata quando si verifica un errore durante l'autenticazione dell'utente.
- **login**: contiene la definizione della *View Component* utilizzata per la visualizzazione della pagina di autenticazione dell'utente.
- **shared**: contiene la definizione del *footer*[5] e della *navigation bar*[6] visualizzata nella parte superiore di ciascuna pagina.

[4]File di tipo TypeScript; è un linguaggio di programmazione open source sviluppato da Microsoft.

[5]Con tale termine si indica la parte inferiore di un sito, che normalmente ha lo stesso aspetto per tutte le pagine del dominio.

[6]Sezione di un'interfaccia utente grafica destinata ad aiutare i visitatori ad accedere alle informazioni.

– **sidebar**: contiene la definizione della *sidebar*[7] utilizzata come menù principale dell'applicativo.

5.2 Autenticazione

Nel *front-end* l'autenticazione dell'utente è stata gestita principalmente mediante l'utilizzo della *Route Guard* **AuthGuard** (definita all'interno della cartella *guards*) e del servizio **AuthService** (definito all'interno della cartella *services*).

5.2.1 AuthGuard

Tale classe è definita all'interno del *file authGuard.ts* nella cartella *guards* ed è una **Route Guard**: permette di intercettare la navigazione dell'utente su un determinato URL e cosentirne o meno l'accesso in base a delle condizioni. Implementa l'interfaccia **CanActive**[8] ed effettua l'*override*[9] del metodo **canActive**; in base al valore booleano restituito dal suddetto metodo (*true* o *false*) viene consentita la navigazione su un determinato percorso dell'applicativo.

Listing 5.1: Classe AuthGuard

```
1
2     export class AuthGuard implements CanActivate {
3     constructor(private router: Router, private authService:
          AuthService) { }
4     canActivate(route: ActivatedRouteSnapshot, state:
          RouterStateSnapshot): Promise<boolean> {
5          return new Promise((resolve, reject) => {
6              if (this.authService.hasToken()) {
7                  this.authService.isLogged().then(body => {
8                          resolve(true);
9                  }).catch(error => {
10                     console.log(error);
11                     localStorage.removeItem('currentUser');
12                     console.log("User is not logged in");
13                     this.router.navigate(['login']);
```

[7]La sidebar, letteralmente "barra laterale", è un elemento di design dei siti web e contraddistingue una o più aree all'interno di un *layout* a colonne. Nella configurazione tipica la sidebar è una sola e si posiziona sulla destra o sulla sinistra dell'area principale della pagina web.

[8]Interfaccia della libreria *@angular/router* implementabile da una classe per essere utilizzata come Guard.

[9]Nella programmazione ad oggetti l'override è l'operazione di riscrittura di un metodo ereditato.

```
                    resolve(false);
                })
            } else
            {
                console.log("User does not have the token");
                this.router.navigate(['login']);
                resolve(false);
            }
        })
    }
}
```

Nel metodo **canActive** si utilizza il servizio **AuthService** per verificare se l'utente possiede un *token* di autenticazione. Tale *token* è memorizzato dall'**AuthService** all'interno dello *storage* locale del *Web Browser*; all'interno dello *storage* il *token* è identificato dal nome **"currentUser"**.

Se il token non è presente all'interno dello *storage* (il metodo *hasToken()* dell'AuthService restituisce *false*) si visualizza la *View Component* utilizzata per effettuare l'autenticazione dell'utente ed il metodo **canActive** restituisce *false* in modo da non permettere la navigazione su un URL differente da quello utilizzato per visualizzare la schermata di autenticazione.

Se il token è presente, si controlla la sua validità mediante il metodo *isLogged()* dell'AuthService. Se il *token* è valido l'utente può navigare liberamente sulla pagina richiesta, quindi il metodo **canActive** restituisce *true*; viceversa il *token* scaduto viene eliminato dallo *storage* locale e si reindirizza l'utente sulla schermata di autenticazione.

Per invocare l'**AuthGuard** durante la navigazione dell'utente, tutte i percorsi utilizzati all'interno del *front-end* sono stati definiti in questo modo all'interno del *file app.routing.ts*:

Listing 5.2: Definizione del path utilizzato per accedere alla View Component utile alla gestione degli Active Ingredients.

```
path: 'ActiveIngredients',
component: ActiveIngredientsComponent,
canActivate: [AuthGuard]
```

Questa gestione impone all'utente di effettuare obbligatoriamente l'autenticazione per poter accedere a qualsiasi *path*[10] dell'applicativo.

[10]Un path è un percorso che indica la posizione specifica di un elemento all'interno di un sistema.

5.2.2 AuthService

Tale servizio è definito all'interno del *file authService.ts* presente nella cartella *services*; permette la gestione dell'autenticazione dell'utente e la memorizzazione del *token* fornito dall'*Auth Server* del *back-end*.

All'interno dell'**AuthService** sono stati definiti ed implementati i seguenti metodi:

- **hasToken()**: funzione che controlla la presenza del *token* all'interno dello *storage* locale del *Web Browser* con il quale si utilizza l'applicativo. Il *token* è identificato da un elemento chiamato **"currentUser"**; il metodo restituisce un valore booleano in base alla presenza o meno dell'elemento.

Listing 5.3: Funzione *hasToken* della classe AuthService

```
1     hasToken():boolean {
2         if (JSON.parse(localStorage.getItem('currentUser')) !=
              null)
3         {
4             this.token =
                  JSON.parse(localStorage.getItem('currentUser'))
5             .access_token;
6             this.tokenType =
                  JSON.parse(localStorage.getItem('currentUser'))
7             .token_type;
8             return true;
9         }
10        else
11            return false;
12    }
```

- **getAuthToken()**: funzione che restituisce il *token* memorizzato nello *storage*.

Listing 5.4: Funzione *getAuthToken* della classe AuthService

```
1       getAuthToken()
2    {
3        return JSON.parse(localStorage.getItem('currentUser'))
4        .access_token;
5    }
```

- **isLogged()**: tale funzione effettua una chiamata Web API di tipo GET all'URL con il quale si è registrato il *Web Server + /IsTokenAutherized*. La risposta della suddetta chiamata permette di identificare se il *token* presente nello *storage* è valido o meno. Tale metodo è invocato solamente se all'interno dello *storage* locale è presente un *token*. Il *token* viene allegato alla chiamata

Web API mediante un *Interceptor Service* analizzato successivamente nella trattazione.

Listing 5.5: Funzione *isLogged* della classe AuthService

```
1     isLogged() {
2         return this.http.get('/IsTokenAuthorized').toPromise()
3     }
```

- **login(username,password)**: funzione che effettua la richiesta di un *token* utilizzando le credenziali passate come parametro al metodo (contenute nelle variabili *username* e *password*). Tale metodo effettua una chiamata di tipo POST all'URL con il quale si è registrato il *Web Server + /token*. Se la chiamata restituisce un risultato senza alcun errore, si memorizza il *token* ottenuto. Nel caso in cui si verifica un errore, si invoca la *View Component* definita nella cartella *Views/error*. La suddetta funzione è richiamata per effettuare l'autenticazione dell'utente.

Listing 5.6: Funzione *login* della classe AuthService

```
1     login(username, password){
2         this.myLoaderValue = true;
3         let urlSearchParams = new URLSearchParams();
4         urlSearchParams.append('username', username);
5         urlSearchParams.append('password', password);
6         urlSearchParams.append('grant_type', 'password');
7         urlSearchParams.append('client_id',
              '099153c2625149bc8ecb3e85e03f0022');
8         let body = urlSearchParams.toString()
9         this.http.post('/token', body ).pipe()
10            .subscribe(
11            data => {
12                localStorage.setItem('currentUser',
                      JSON.stringify(data));
13                console.log(data);
14                this.token = JSON.parse(localStorage.
15                getItem('currentUser'))
16                .access_token;
17                this.tokenType = JSON.parse(localStorage.
18                getItem('currentUser'))
19                .token_type;
20                this.router.navigate(['']);
21            },
22            error => {
23                this.myErrorText = error.error_description;
24                console.log(this.myErrorText);
25                this.myLoaderValue = false;
26                this.myErrorValue = true;
27            }
```

```
28
29          );
30      }
```

5.3 Chiamate Web API

Il *front-end* invoca le **Chiamate Web API** definite dal *Web API Server* del *back-end* mediante l'utilizzo dell'**APIService** definito all'interno del *file APIService.ts* nella cartella *services*. In tale servizio sono definiti i metodi per effettuare le operazioni CRUD su qualsiasi entità gestita dall'applicativo (ad esempio visualizzazione, modifica, aggiunta e eleminazione di un *Active Ingredient*). In base al tipo di operazione da effettuare si effettua una chiamata Web API di tipo GET (per la visualizzazione), POST (per la creazione), PUT (per la modifica) e DELETE (per la eliminazione). In **Angular** è possibile effettuare una chiamata Web API utilizzando un oggetto di tipo **HttpClient** (definito all'interno della libreria *@angular/commn/http*).

I sotto paragrafi successivi illustrano come sono stati definiti i vari metodi dell' **APIService** per effettuare le chiamate Web API utili a svolgere le operazioni CRUD su elementi di tipo *ActiveIngredient*; tali metodi sono stati definiti analogamente per effettuare le operazioni CRUD su elementi di altra tipologia

5.3.1 Creazione di un elemento

Di seguito si visualizza il metodo dell'**APIService** per la creazione di un nuovo *Active Ingredient*.

Listing 5.7: Metodo del APIService per effettuare una chiamata Web API utile a creare un nuovo Active Ingredient

```
1   AddActiveIngredients(param: ActiveIngredient)
2   {
3       return this.http.post('api/ActiveIngredients?name=' +
            param.Name + '&color='+color   + '&id_unit='+
            param.ID_Unit + '&flag_cytostatic=' +
            param.Flag_Cytostatic, { observe: 'response' })
4           .toPromise()
5           .then(data => {
6               if (data == "true") {
7                   this.alertNot('This active ingredient was
                        successfully added.', 'ti-check', 2);
```

```
                }
            });
    }
```

Tale metodo richiede come parametro un oggetto di tipo *ActiveIngredient* ed effettua la chiamata Web API inserendo come parametri le informazioni relative al nuovo elemento da creare. Se la chiamata viene processata con successo dal *Server*, si visualizza un *alert*[11] per notificare l'utente del completamento dell'operazione richiesta.

5.3.2 Visualizzazione di una categoria di elementi

Di seguito si visualizza il metodo dell'**APIService** per la visualizzazione di tutti gli *Active Ingredients*.

Listing 5.8: Metodo del APIService per effettuare una chiamata Web API utile a visualizzare tutti gli Active Ingredients

```
    ListActiveIngredients(): any
    {
        return
            this.http.get<ActiveIngredient>('/api/ActiveIngredients',
            { observe: 'response' });
    }
```

La visualizzazione di una differente categoria di elementi è similarmente effettuata mediante una chiamata GET all'URL al quale è stato registrato il *Web Server* + */api/NOME_CATEGORIA*.

5.3.3 Modifica di un elemento

Di seguito si visualizza il metodo dell'**APIService** per la modifica di un *Active Ingredient*.

Listing 5.9: Metodo del APIService per effettuare una chiamata Web API utile a modificare uno specifico Active Ingredient

```
   UpdateActiveIngredients(param: ActiveIngredient): Promise<any>
    {
        return this.http.put('/api/ActiveIngredients?id=' + param.ID
            + "&name=" + param.Name + "&color=" + color + "&id_unit="
            + param.ID_Unit + "&flag_cytostatic=" +
            param.Flag_Cytostatic, { observe: 'response' })
```

[11] Messaggio di avviso visualizzato mediante una finestra di PopUp.

```
            .toPromise()
            .then(data => {
                if (data == "true") {
                    this.alertNot('This active ingredient was
                        successfully edited.', 'ti-check', 2);
                }
            });
    }
```

Tale metodo richiede come parametro un oggetto di tipo *ActiveIngredient* ed effettua la chiamata Web API inserendo come parametri le informazioni relative all'elemento da modificare. Se la chiamata viene processata con successo dal *Server*, si visualizza un *alert* per notificare l'utente del completamento dell'operazione richiesta.

5.3.4 Eliminazione di un elemento

Di seguito si visualizza il metodo dell'**APIService** per l'eliminazione di un *Active Ingredient*.

Listing 5.10: Metodo del APIService per effettuare una chiamata Web API utile ad eliminare uno specifico Active Ingredient

```
    DeleteActiveIngredients (param:ActiveIngredient): Promise<any>
    {
        return this.http.delete('api/ActiveIngredients/' + param.ID,
            { observe: 'response' })
            .toPromise()
            .then(data => {
                if (data.statusText == "OK")
                    this.alertNot('The active ingredients was
                        successfully deleted.', 'ti-trash', 2);
            });
    }
```

Tale metodo richiede come parametro un oggetto di tipo *ActiveIngredient* ed effettua la chiamata Web API inserendo come parametro l'identificatore univoco (ID) dell'elemento da eliminare. Se la chiamata viene processata con successo dal *Server*, si visualizza un *alert* per notificare l'utente del completamento dell'operazione richiesta.

5.3.5 APIInterceptor

L'**APIInterceptor** è un servizio definito all'interno del *file APIInterceptor.ts* memorizzato nella cartella *services*. Tale classe ha lo scopo di intercettare tutte le chiamate Web API effettuate dal *front-end* ed inserirne il corretto *header*[12] in modo da poter effettuare correttamente la richiesta al Server.

Tale servizio implementa l'interfaccia **HttpInterceptor** che ha lo scopo di intercettare e gestire richieste e risposte HTTP. Per implementare l'interfaccia è stato necessario importarla dal *package @angular/common/http*.

L'**APIInterceptor** effettua l'*override* del metodo *intercept*. All'interno della suddetta funzione si imposta l'*header HTTP* in base a se è presente un *token* di autenticazione o meno:

- Se il *token* autenticativo è presente nello *storage*, si costruisce l'*header* inserendo la tipologia del *token* ed il *token* stesso.
- Se il *token* autenticativo non è presente nello *storage*, si costruisce un *header* generico non contenente informazioni relative all'autenticazione dell'utente.

Listing 5.11: Definizione del servizio APIInterceptor

```
export class APIInterceptor implements HttpInterceptor {

   token = null;
   tokenType = null;
   intercept(req: HttpRequest<any>, next: HttpHandler):
      Observable<HttpEvent<any>> {
       if (localStorage.getItem('currentUser') == null) {
           const clone = req.clone({
               setHeaders: {
                   'Content-type':
                      'application/x-www-form-urlencoded'
               }
           });
           return next.handle(clone);
       }
       else {
           this.token =
              JSON.parse(localStorage.getItem('currentUser'))
           .access_token;
```

[12]Un header HTTP è un'intestazione trassmessa attraverso il protocollo HTTP contenente delle informazioni prodotte dall'interazione tra il browser del client mittente delle richieste e il server che le raccoglie e invia delle risorse in risposta alle richieste ricevute.

```
17              this.tokenType =
                    JSON.parse(localStorage.getItem('currentUser'))
18              .token_type;
19              const clone = req.clone({
20                  setHeaders: {
21                      'Authorization': this.tokenType + ' ' +
                            this.token, 'Content-type': 'application/json'
22                  }
23              });
24              return next.handle(clone);
25          }
26      }
27  }
```

L'**APIInterceptor** ha permesso di non vincolare l'**APISerivce** dal compito di costruzione dell'*header HTTP* prima dell'invio di ogni richiesta.

5.4 Template utilizzato

Per rendere piacevole l'esperienza utente durante l'utilizzo dell'applicativo si è deciso di utilizzare un *template* grafico esterno in modo da migliorare la GUI (*Graphic User Interface*) dell'applicazione.

Si è scelto di utilizzare il *template* gratuito **Paper Dashboard** ideato dall'azienda **Creative-Tim**[10]. Questo è un template basato su **BootStrap**: una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web, in particolare è una raccolta di strumenti grafici, stilistici e di impaginazione che permettono di avere a disposizione una gran quantità di funzionalità e di stili modificabili e adattabili a seconda dell'esigenza. La principale funzione di BootStrap è il *responsive web design*, ovvero i suoi elementi sono in grado di adattarsi dinamicamente a seconda della grandezza e delle caratteristiche del dispositivo utilizzato per visualizzare l'applicazione.

In figura 5.6 e 5.7 si mostrano delle anteprime del *template* in funzione.

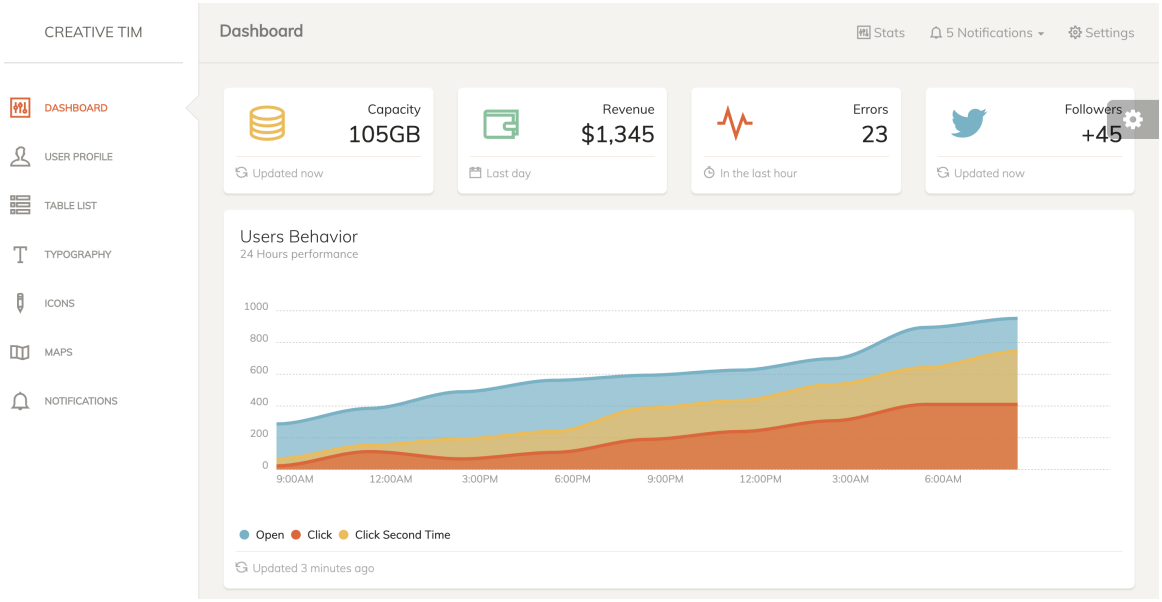


Figura 5.6: Anteprima della dashboard del template gratuito Paper Dashboard

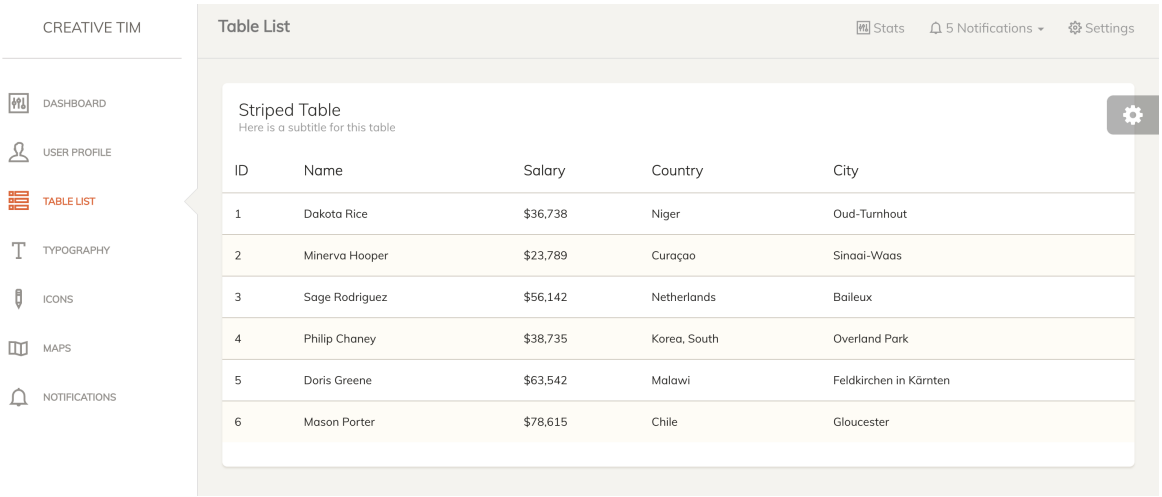


Figura 5.7: Anteprima delle tabelle del template gratuito Paper Dashboard

6. Installazione

In questo capitolo si illustra la procedura seguita per installare l'intera infrastruttura al fine di rendere funzionante l'applicativo. L'installazione del sistema è stata effettuata su un dispositivo Loccioni sul quale è presente il sistema operativo **Windows Server 2012**. Per rendere l'applicativo utilizzabile digitando l'URL *http://IP_SERVER:5001* all'interno del *Web Browser*, si è installato il *back-end* digitando i comandi di **TopShelf** all'interno del *prompt dei comandi*[1] di Windows.

6.1 Comandi TopShelf

TopShelf mette a disposizione una documentazione in cui è illustrata la *command-line*[1] utilizzabile per installare, da terminale, un *Windows Service* che utilizza TopShelf. I comandi principali sono i seguenti:

- ***service.exe install***: installa il servizio *service.exe*
- ***service.exe uninstall***: disinstalla il servizio *service.exe*, se questo è in esecuzione viene precedentemente interrotto.
- ***service.exe start***: avvia il servizio *service.exe*
- ***service.exe stop***: interrompe il servizio *service.exe*

È possibile eseguire ciascun comando aggiungendo dei valori parametrici opzionali in successione al comando principale da eseguire:

- ***-username:NOME_UTENTE***: permette di specificare l'*username* dell'utente che può avviare il servizio.
- ***-password:PASSWORD***: permette di specificare la *password* per l'*username* inserito.

[1]Strumento a riga di comando, disponibile nella maggior parte dei sistemi operativi Windows, utilizzato per eseguire dei comandi immessi dall'utente.

- ***-servicename:NOME_SERVIZIO***: permette di specificare il nome che il servizio deve utilizzare quando è installato.
- ***-description:DESCRIZIONE***: permette di specificare la descrizione che il servizio deve avere quando è installato.

Esempio:

> *MyService.exe install -username:DOMAIN/ServiceAccount -password:user_password*

6.2 Esecuzione dei comandi

È necessario avviare il *prompt dei comandi* di Windows in modalità amministratore ed accedere al percorso: */SWGA.WebApiService/bin/release*. In seguito è possibile eseguire il comando per l'installazione del servizio (figura 6.1):

SWGA.WebApiService.exe install

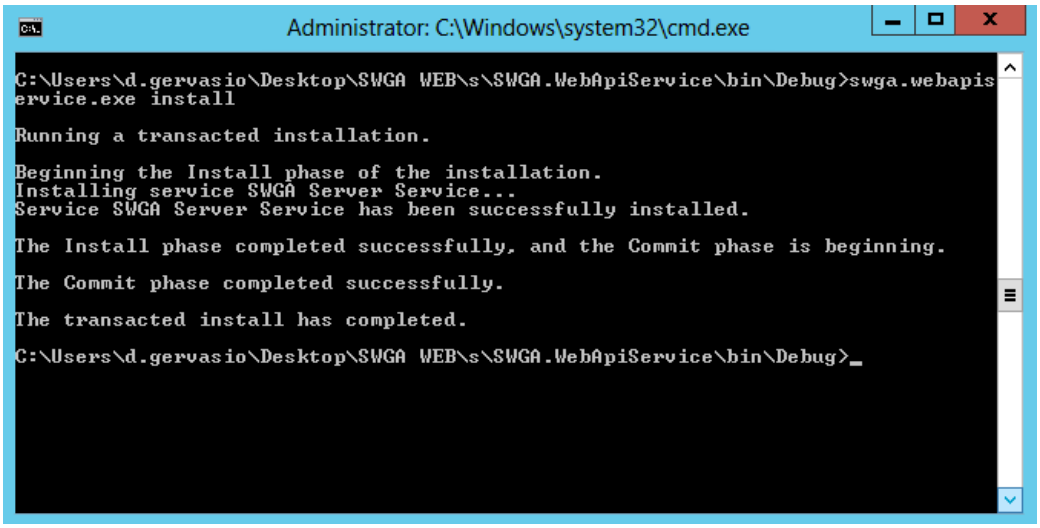


Figura 6.1: Installazione del servizio SWGA.WebApiService.exe da terminale

Successivamente è necessario avviare il servizio tramite il comando (figura 6.2):

SWGA.WebApiService.exe start

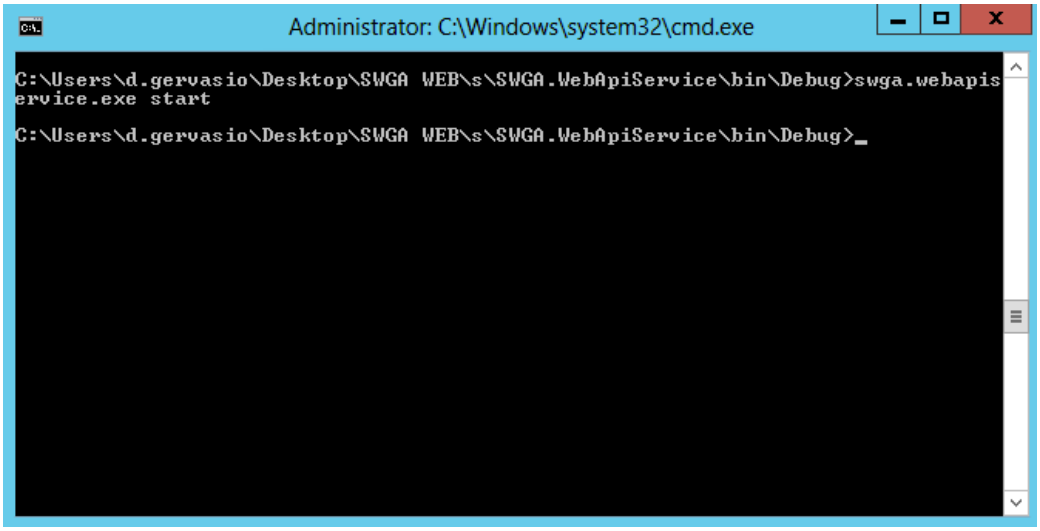


Figura 6.2: Esecuzione del servizio SWGA.WebApiService.exe da terminale

È possibile disinstallare il servizio mediante il seguente comando (figura 6.3):

SWGA.WebApiService.exe uninstall

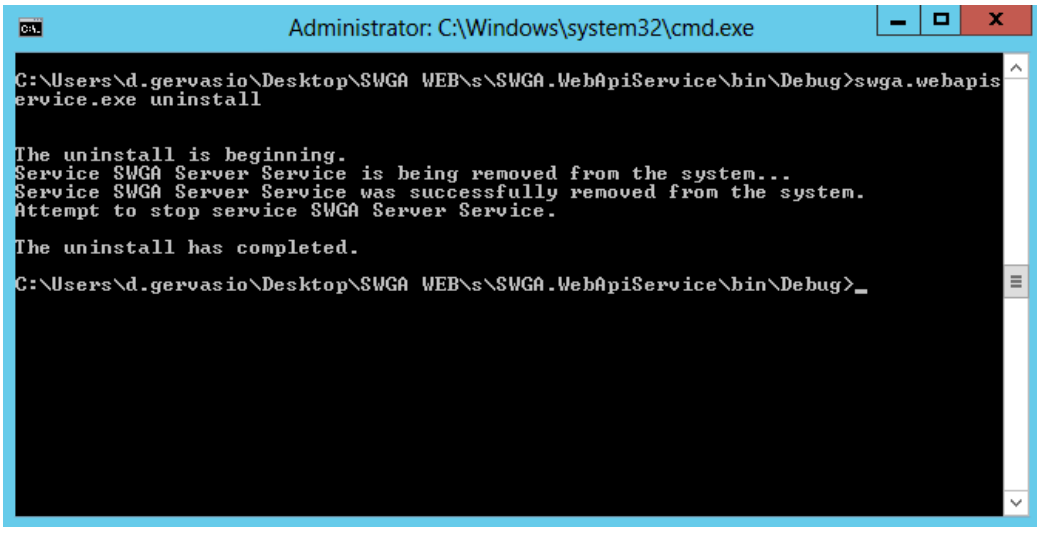


Figura 6.3: Disinstallazione del servizio SWGA.WebApiService.exe da terminale

Mediante il *tool* **Server Manager** messo a disposizione da **Windows Server** è possibile verificare che il servizio sia stato installato correttamente e sia in esecuzione (figura 6.4).

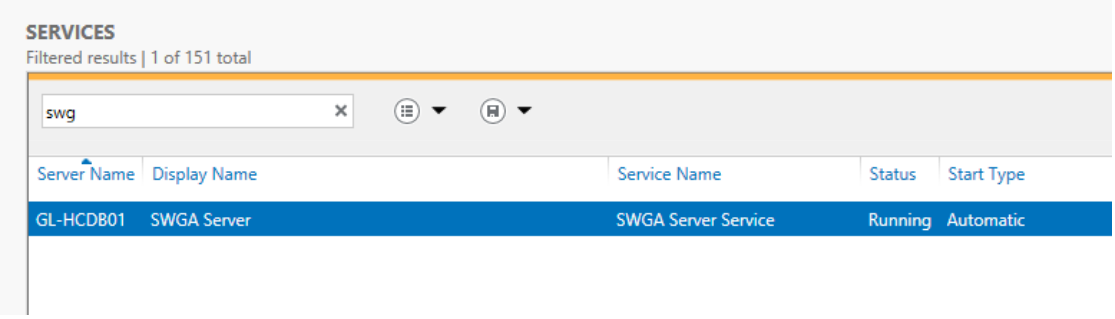


Figura 6.4: Verifica che SWGA.WebApiService sia stato installato correttamente

È possibile effettuare un'ulteriore verifica della corretta installazione del servizio accedendo all'applicativo tramite il *Web Browser*. Si è utilizzato il *Web Browser* **Google Chrome** digitando l'URL *https://localhost:5001* (figura 6.5).

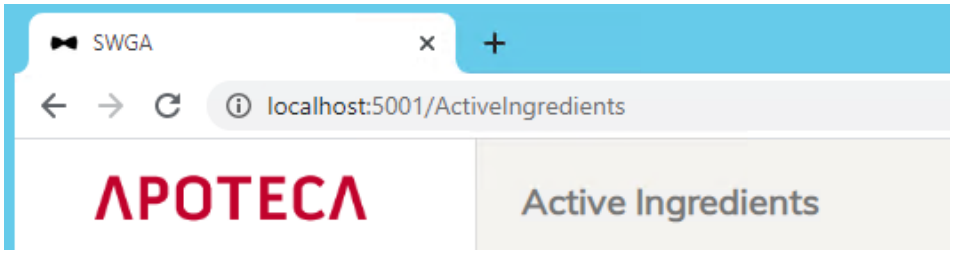


Figura 6.5: Accesso all'applicativo tramite l'utilizzo del Web Browser

7. L'applicativo in esecuzione

Nel seguente capitolo sono mostrate delle immagini dell'applicativo in esecuzione durante il suo normale utilizzo.

In figura 7.1 è mostrata la visualizzazione di tutti gli *Active Ingredients* che rispettino un filtraggio sul nome (il nome deve iniziare con i caratteri 'so').

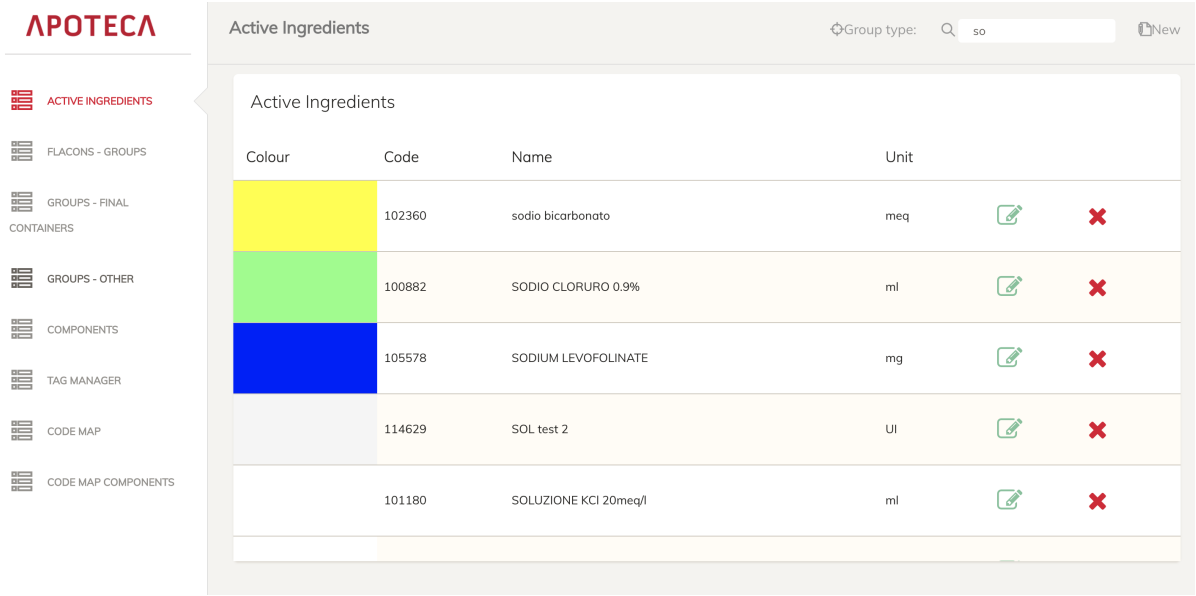


Figura 7.1: Visualizzazione degli *Active Ingredients* con filtraggio sul nome.

In figura 7.2 è mostrata la visualizzazione dei Gruppi Componenti associati all'*Active Ingredient* selezionato ("*SOLVENTE Glucosio*"). Questi sono mostrati in una tabella sottostante, chiudibile dall'utente.

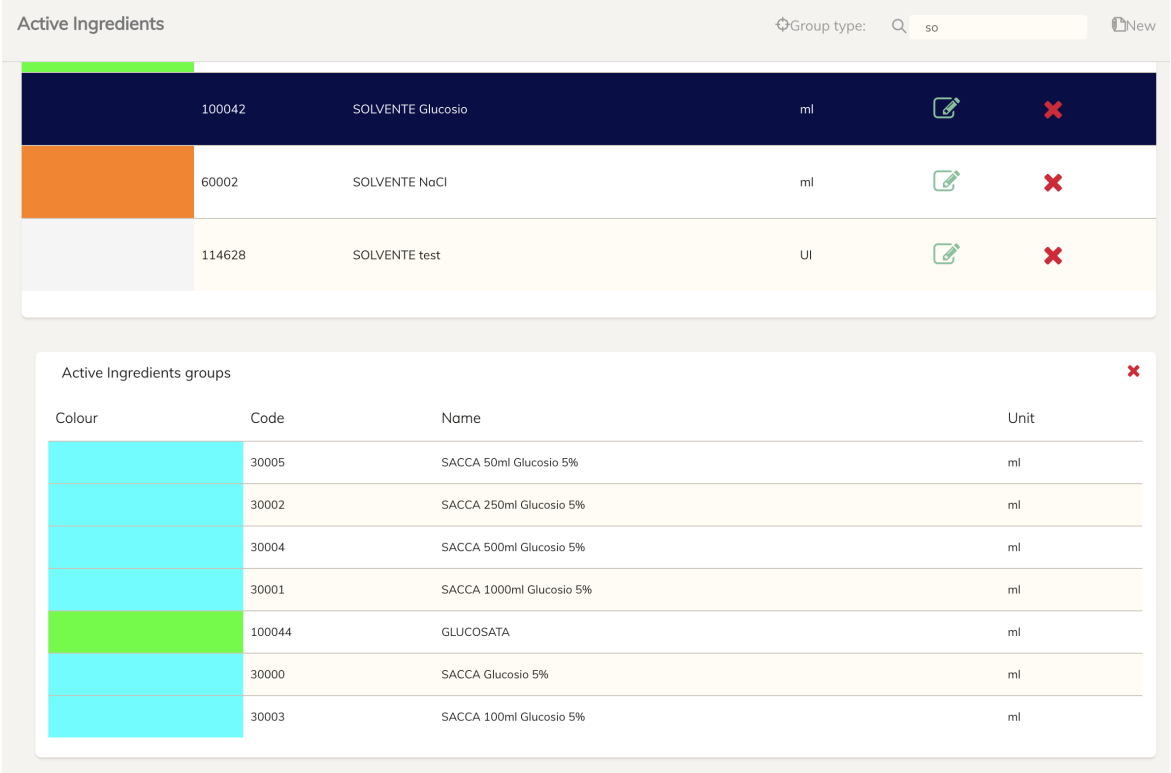


Figura 7.2: Selezione di un *Active Ingredient* e visualizzazione dei gruppi ad esso associati.

In figurar 7.3 è mostrata la schermata di modifica di uno specifico *Active Ingredient*.

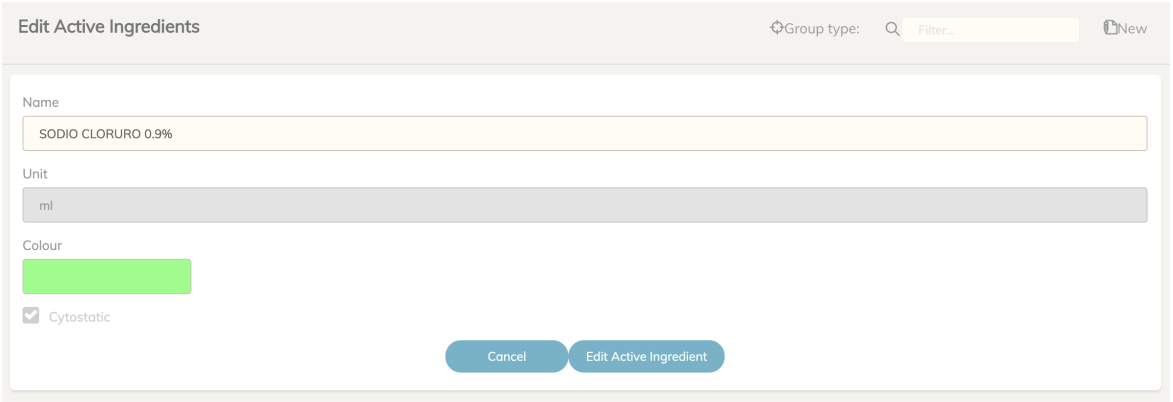


Figura 7.3: Modifica di un *Active Ingredient*.

In figura 7.4 è mostrata la visualizzazione di tutti i *Flacons - Groups* di tipo *Drugs* che rispettino un filtraggio sul nome (il nome deve iniziare con i caratteri 'c').

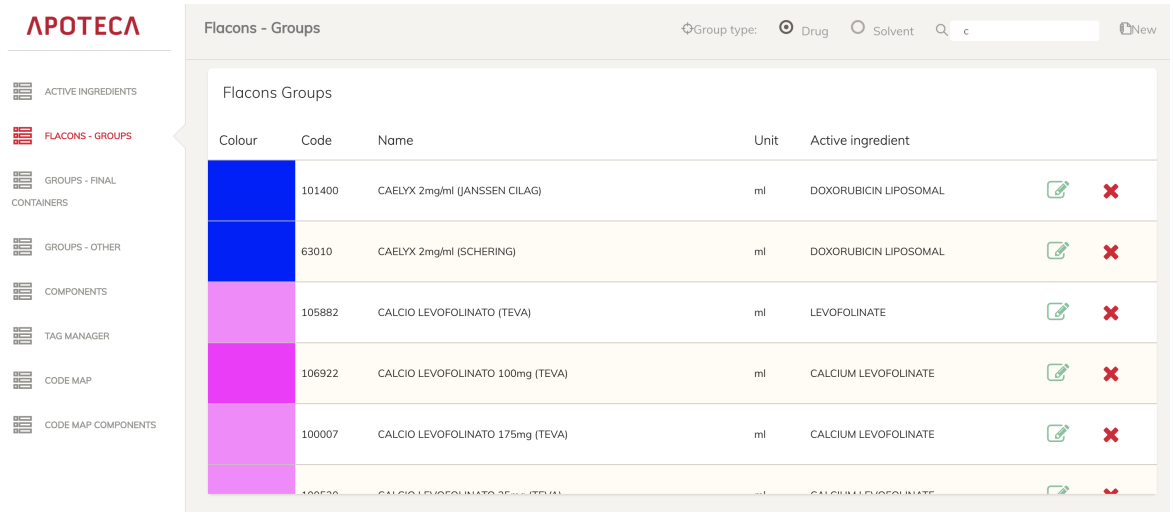


Figura 7.4: Visualizzazione dei *Flacons - Groups* con filtraggio sul nome.

In figura 7.5 è mostrata la visualizzazione dei Componenti associati al *Flacon - Group* selezionato ("*CYTARABINE 100mg/ml (FIZER)*"). Questi sono mostrati in una tabella sottostante, chiudibile dall'utente.

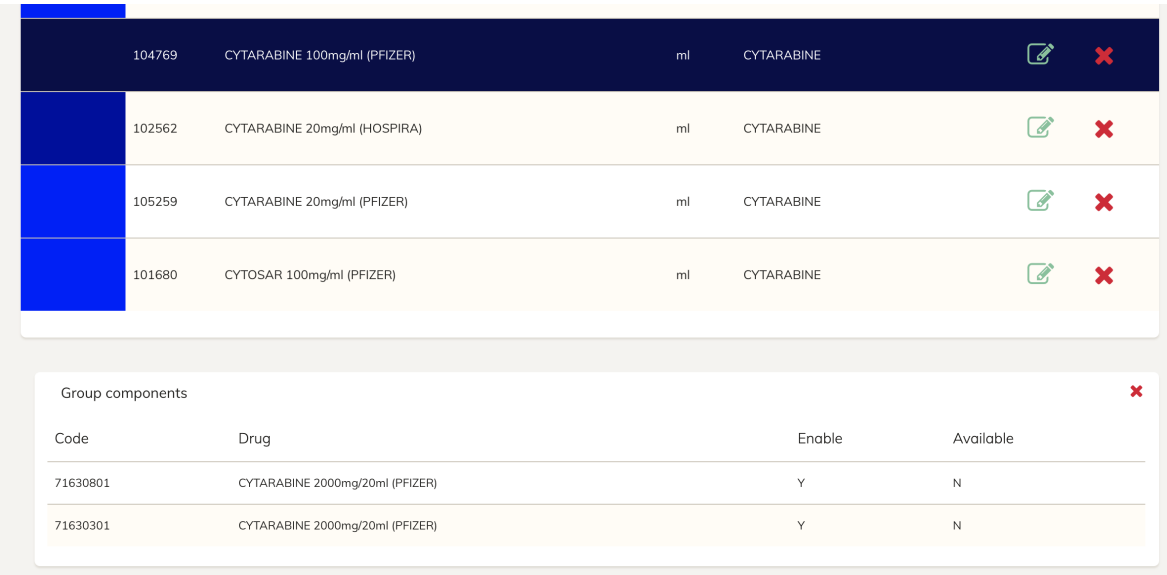


Figura 7.5: Selezione di un *Flacon - Group* e visualizzazione dei componenti ad esso associati.

In figura 7.6 è mostrata la schermata di creazione di un *Flacon - Groups*. Da tale pagina è possibile impostare la tipologia del gruppo dell'elemento creato: *Drugs* o *Solvent*.

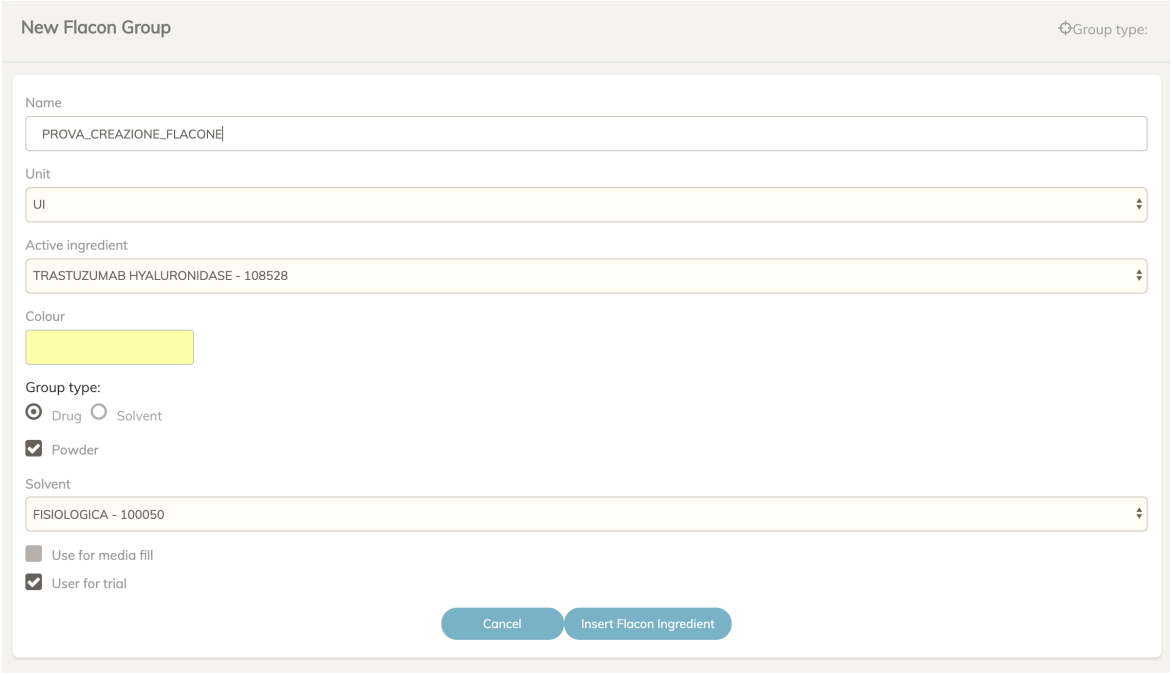


Figura 7.6: Creazione di un *Flacon - Group*.

In figura 7.7 è mostrata la visualizzazione di tutti i *Components* di tipo *Bag* che rispettino un filtraggio sul nome (il nome deve iniziare con i caratteri 'sa').

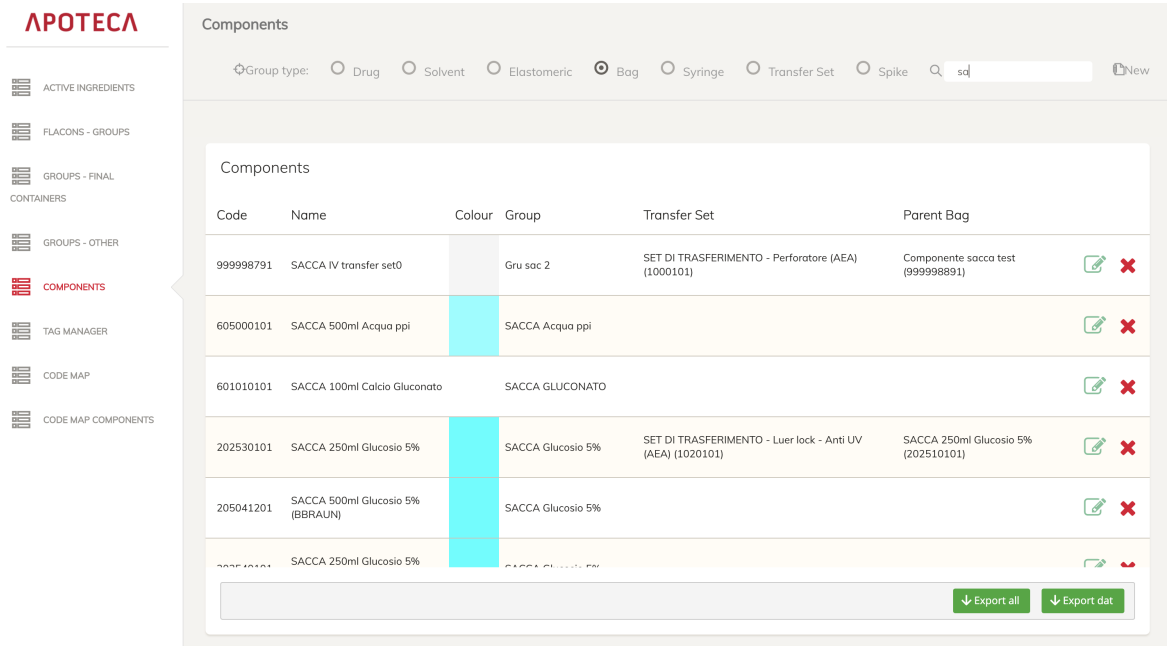


Figura 7.7: Visualizzazione dei *Components* di tipologia *Bag* con filtraggio sul nome.

In figura 7.8 è mostrata la schermata di modifica di uno specifico *Component* di tipologia *Drug*.

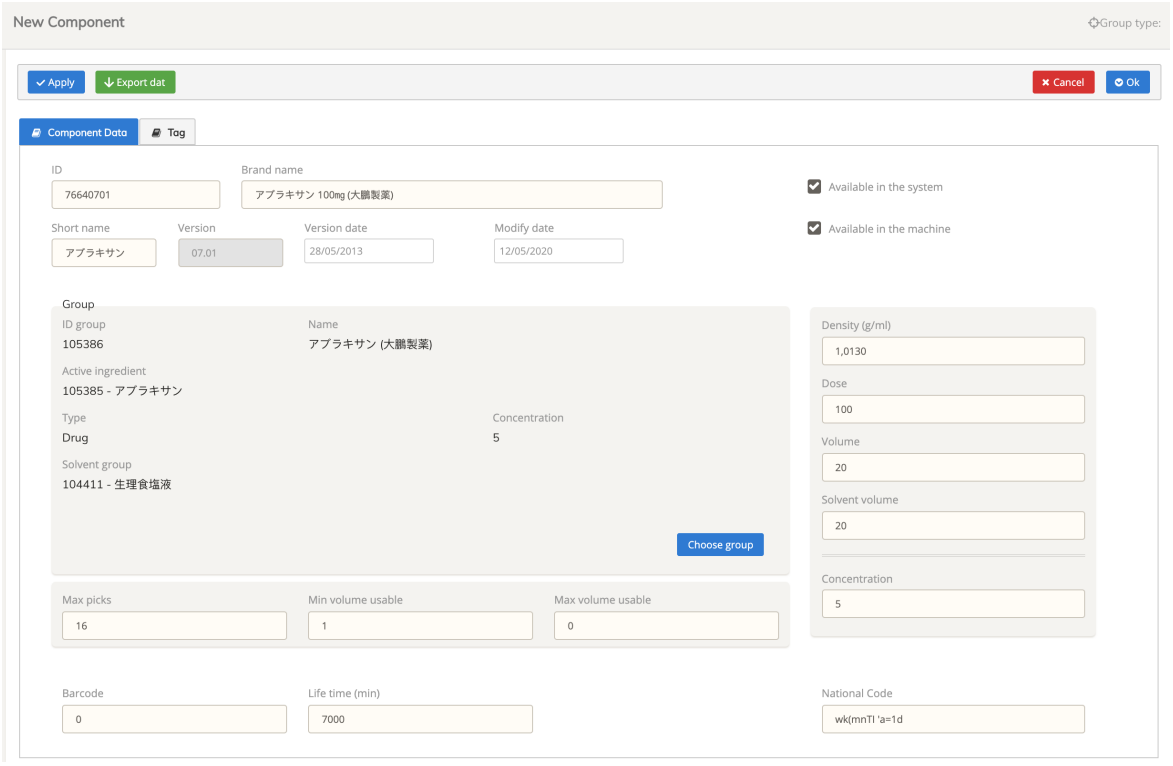


Figura 7.8: Modifica di un *Component* di tipologia *Drug*.

In figura 7.9 è mostrata la schermata di modifica dei *Tag* associati ad uno specifico *Component*.

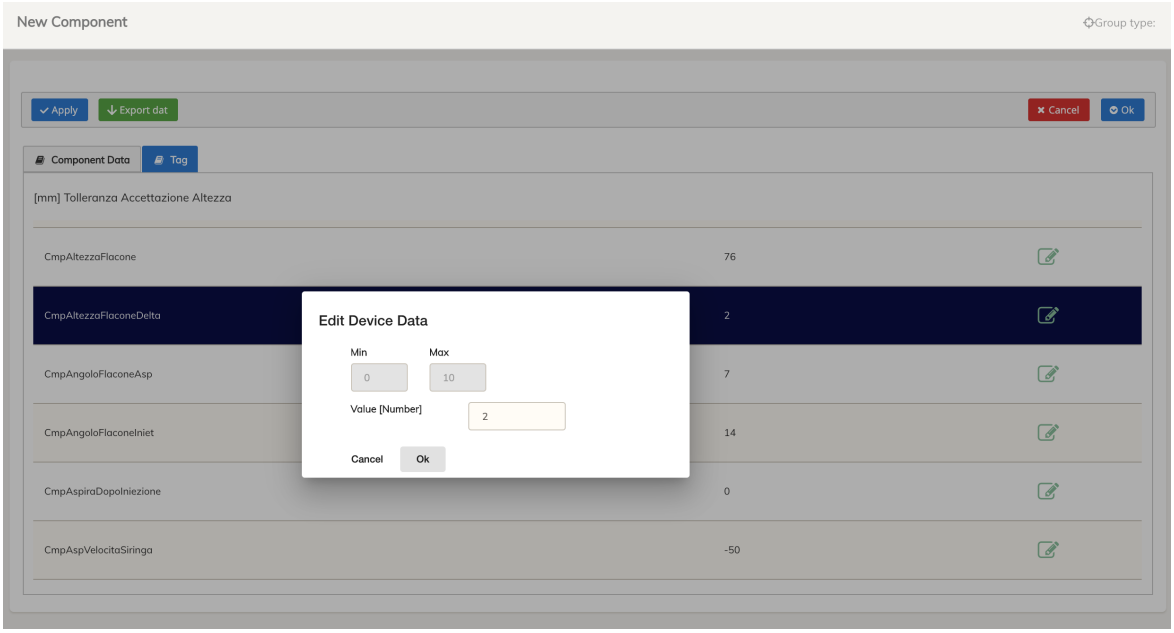


Figura 7.9: Modifica dei Tag associati ad un componente.

In figura 7.10 è mostrata la schermata di modifica delle configurazioni relative ad uno specifico *Tag* ("*CmpAbilAllineamento*").

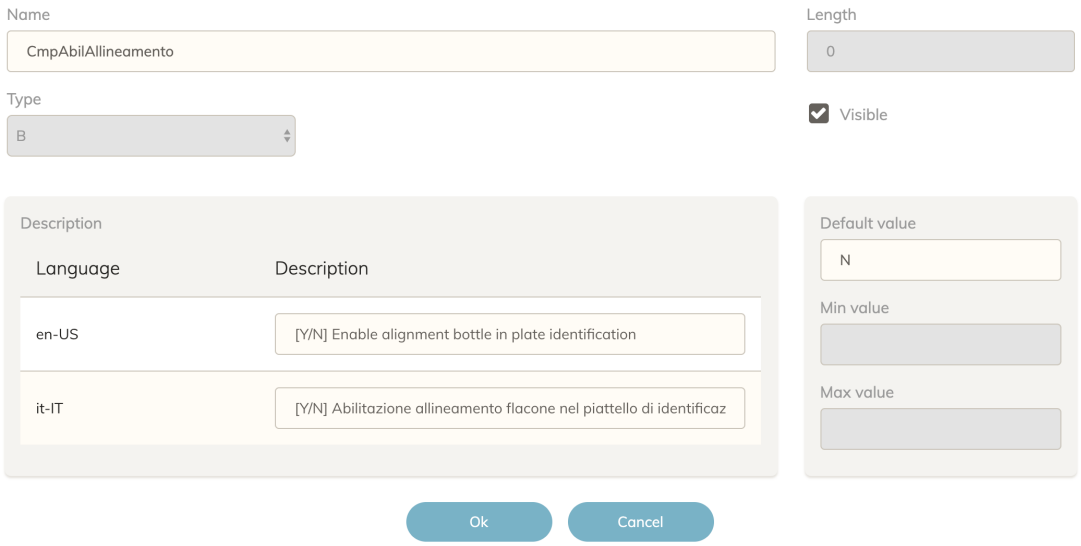


Figura 7.10: Modifica del tag relativo all'abilitazione dell'allineamento del farmaco nel piattello di identificazione della macchina.

8. Conclusioni

L'applicativo è stato correttamente sviluppato raggiungendo pienamente gli obiettivi inizialmente fissati. In seguito alla fase di sviluppo verrà eseguita una fase di *debugging*[1], la quale sarà effettuata da un apposito *team* di collaboratori Loccioni con l'obiettivo di sottoporre l'applicativo a degli *stress test*.

Nonostante la corretta implementazione di tutte le *features* dell'applicativo, si sono definiti dei possibili sviluppi futuri che hanno l'obiettivo di ottimizzare alcune funzioni del software ed introdurre nuove funzionalità aggiuntive.

8.1 Sviluppi futuri

8.1.1 Implementazione SignalR

La visualizzazione delle liste di elementi (*Active Ingredients*, *Flacon Groups* ecc.) è effettuata mediante una richiesta dei dati da parte del *Web Client* al *Server Web*. Per garantire la rappresentazione di dati aggiornati, questa richiesta è effettuata ciclicamente ogni 20 secondi. L'implementazione attraverso **SignalR**[4] permetterebbe al *server* di inviare i dati a tutti i *clients* connessi, senza che questi effettuino una richiesta; in questo modo, nel momento in cui un dato è modificato, il *server* può inviare istantaneamente il dato aggiornato a tutti coloro che ne hanno precedentemente richiesto la visualizzazione. **SignalR** è una libreria per applicazioni **Microsoft ASP.NET** che permette al *server* di inviare notifiche asincrone[2] ai *Web Clients* connessi.

[1] Termine utilizzato in informatica per indicare la fase di ricerca e correzione degli errori di funzionamento di un sistema o di un programma.

[2] Si definisce asincrona una chiamata ad una risorsa esterna che non interferisce con l'esecuzione della risorsa chiamante.

Mediante la suddetta implementazione si limiterebbe notevolmente il traffico dati tra *server* e *clients* in quanto la transizione delle informazioni sarebbe effettuata solamente in caso di necessità, ogni qual volta che un dato visualizzato è aggiornato.

8.1.2 Template grafico

Il Gruppo Loccioni ha recentemente sviluppato un *template* grafico per le varie applicazioni ideate dal Gruppo. Durante lo sviluppo dell'applicativo si è scelto di utilizzare il *template* **Paper Dashboard** in quanto il *template* Loccioni non era ancora implementabili all'interno di una applicazione Web. Un obiettivo futuro è quello di implementare il suddetto *template* per la versione Web di SWGA, in modo da rispettare la linea grafica dei prodotti Loccioni.

8.1.3 Storico utente

Un ulteriore sviluppo futuro riguarda la possibilità di permettere a ciascun utente autenticato di visualizzare lo storico delle operazioni effettuate da un determinato utente in un certo arco temporale. In questo modo ciascun utilizzatore avrà la possibilità di sapere quali sono i dati modificati, aggiunti ed eliminati da uno specifico utente. Inoltre, tale implementazione permetterebbe all'applicativo di mostrare il nome dell'utente che ha effettuato l'ultima modifica su ciascun dato, così da garantire un maggior controllo sulle informazioni trattate.

A. C#

C#[8] è un linguaggio di programmazione orientato agli oggetti (*Object Oriented*) sviluppato da **Microsoft** specificatamente per la programmazione nel *framework* .NET . La sintassi del suddetto linguaggio prende spunto da altri linguaggi di programmazione, quali Delphi, C++, Java e Visual Basic.

A.1 Caratteristiche

- I nomi delle variabili, funzioni e classi sono sensibili rispetto a caratteri minuscoli e maiuscoli: C# è un linguaggio ***case-sensitive***.
- Ogni istruzione è seguita dal carattere ";".
- Le parentesi graffe ({}) sono utilizzate per raggruppare una parte di codice denominata **specifica**.
- Secondo le consuetudini dei linguaggi orientati agli oggetti, le specifiche sono di regola raggruppate in metodi (ovvero funzioni), i metodi sono raggruppati in classi, e le classi sono raggruppate nei *namespace*.

A.2 Differenze con C e C++

C# presenta varie modifiche rispetto a C e C++, volte ad evitare errori ed ambiguità tipici della programmazioni in linguaggio C.

- I puntatori possono essere utilizzati solo in particolari blocchi di codice marcati come ***unsafe***.
- Non si necessita la deallocazione esplicita degli oggetti dinamici utilizzati. C# offre un sistema di ***garbace-collector*** che si occupa di deallocare i vari oggetti che non presentano più dei riferimenti.

- Una classe può ereditare esclusivamente da una sola classe padre ma può implementare un numero indefinito di interfacce.

A.3 Impiego

C# è impiegato in numerose categorie di sviluppo:

- Applicazioni *Mobile*.
- Applicazioni *Desktop*.
- Applicazioni Web.
- Servizi Web.
- Siti Web.
- Videogiochi.
- Applicazioni per database.

B. Angular

Angular[13] è uno dei *framework* **Javascript** *open-source* più popolari, utilizzato da numerosi sviluppatori in tutto il mondo per creare applicazioni web dinamiche grazie a una serie di funzionalità e strumenti che semplificano lo sviluppo delle applicazioni stesse garantendo contemporaneamente ottimi risultati in termini di prestazioni.

B.1 Storia

La prima versione di Angular, a cui spesso si fa riferimento con il nome **AngularJS** (o Angular.js), fu inizialmente sviluppata nel 2009 da Miško Hevery e Adam Abrons con lo scopo di semplificare il processo di sviluppo di applicazioni web. L'intenzione era quella di poter facilitare la realizzazione delle applicazioni attraverso l'uso di un'estensione del linguaggio HTML. Il nome Angular deriva semplicemente dal fatto che i *tag* HTML sono racchiusi da parentesi angolari.

AngularJS fu rilasciato come progetto *open-source* e la versione 1.0 venne pubblicamente annunciata a maggio del 2011. In poco tempo AngularJS divenne un *framework* estremamente popolare e numerosi sviluppatori in tutto il mondo iniziarono a usarlo per creare svariate applicazioni. Quando a ottobre 2014 è stata mostrata un'anteprima di quello che poi sarebbe diventato **Angular 2**, le reazioni da parte della comunità degli sviluppatori di Angular sono state contrastanti. Il motivo per cui in molti erano scettici va ricondotto al fatto che Angular 2 è stato completamente riscritto e non è compatibile con la prima versione di Angular. Il motivo di tale scelta è legato alla necessità di creare un *framework* che tenesse conto delle novità introdotte negli ultimi anni sia lato *client* che *server*. L' attuale versione di Angular è la 10, quasi totalmente compatibile con la iniziale versione 2.

B.2 AngularJS e Angular 2

A differenza di AngularJS, Angular 2, e le successive versioni, è stato pensato per adattarsi perfettamente ai dispositivi mobili come *smartphone* e *tablet*; tutte le applicazioni progettate e create con Angular 2 risultano infatti ***responsive***: tale termine indica la capacità dell'applicazione di adattarsi alla dimensione dello schermo sul quale viene utilizzata.

Tra AngulaJS e Angular 2 vi è una sostanziale differenza nel linguaggio di programmazione utilizzato per lo sviluppo di un'applicazione:

- AngularJS utilizza **JavaScript**.
- Angular 2 utilizza **TypeScript**, un'estensione, creata da Microsoft, di JavaScript che si basa sulle specifiche della versione 6 di *JavaScript - ECMAScript 6* rilasciate nel maggio 2015.

Inoltre anche le *directives* subiscono un notevole miglioramento in Angular 2:

- I tag HTML utilizzati nelle precedenti versioni di Angular, risultano in Angular 2 notevolmente ampliati, con lo scopo di aggiungere maggiori funzioni all'applicazione in corso di programmazione. È inoltre possibile creare nuove *directives* in maniera semplice e veloce per incrementare le potenzialità dell'app, grazie all'utilizzo di semplici comandi come "***@Directive***".

C. Oracle Database

Oracle Systems Corporation e' l'azienda che produce il *software* leader mondiale nel campo dei database. Il **database Oracle**[2] è certamente uno dei migliori prodotti per la gestione di grosse quantità di dati, ma è adatto anche a database di piccole dimensioni. Oracle è un database relazionale ad oggetti, un'evoluzione del database relazionale che offre funzionalità orientate agli oggetti. È un sistema notevolmente complesso, formato da un nucleo che si occupa di gestire il database e un insieme di *tools*.

Il compito del nucleo è quello di organizzare la gestione dei dati, controllarne l'accesso, interpretare i comandi SQL e PL/SQL e gestire il backup e il recupero dei dati. I *tools* principali sono:

- **SQL*Loader**: permette di caricare dati memorizzati su file di testo.
- **SQL*Plus**: funge da interfaccia con il database.
- **Export**: utilizzato per scaricare su un file la struttura e i dati di un database.
- **Import**: utilizzato per caricare nel database i file generati dall'Export.
- **SQL*Net** che permette la comunicazione con i *client* via rete.

Punti di forza di un **database Oracle:**

- **Routine di backup e recovery**: funzionalità che permette di creare copie dei dati per prevenire la perdita degli stessi ripristinando i dati.
- **Gestione di grosse quantità di dati**: grazie a tecnologie come il partizionamento dei dati, Oracle è particolarmente indicato nella gestione di grossi volumi di dati. In questo modo il db è suddiviso in parti più facilmente gestibili, il tutto in maniera trasparente all'utilizzatore.
- **Sicurezza**: l'accesso ai dati è controllato da meccanismi di sicurezza sofisticati che permettono di decidere per ogni utente ciò che può modificare, ciò che può solo leggere o impedirgli completamente l'accesso.

- **Gestione dello spazio su disco**: tutte le informazioni memorizzate nel db sono su file. Oracle permette una gestione molto flessibile dello spazio, consentendo di suddividere i file su più dischi (velocizzando l'accesso ai dati, che può avvenire in parallelo), consente inoltre di memorizzare tabelle differenti su file diversi (in modo da ridurre al minimo i problemi di frammentazione) e di decidere come questi file cresceranno.

D. RESTful Web API Server

D.1 REST

L'acronimo **REST**[11] sta per *REpresentational State Transfer*, cioè trasferimento della rappresentazione dello stato. Il concetto di REST deriva da una dissertazione del 2000 di Roy Fielding, uno dei più importanti sviluppatori di protocolli web. Fielding descrive in forma astratta l'architettura alla base del **World Wide Web**. Fondamentalmente l'architettura REST non dipende da protocolli specifici. La sua caratteristica principale risiede in risorse che devono essere soddisfatte secondo i seguenti requisiti:

- **Indirizzabilità**: ogni risorsa deve poter essere identificata tramite un *Unique Resource Identifier* (**URI**).
- **Interfaccia unica**: deve essere possibile accedere a ogni risorsa in modo semplice e unitario con l'aiuto di metodi standard. Per esempio metodi HTTP come GET, POST o PUT.
- **Struttura client-server**: in generale vale il principio *client-server*, secondo il quale un *server* mette a disposizione un servizio, che se necessario può essere richiesto da un *client*.
- **Assenza di stato**: la comunicazione tra *server* e *client* non presenta alcuno stato. Questo significa che tutti i messaggi scambiati contengono tutte le informazioni necessarie per poterli leggere. Il *server* non memorizza informazioni aggiuntive tra i due messaggi come per esempio sotto forma di sessioni. L'assenza di uno stato rende i servizi REST altamente scalabili, in quanto le richieste in entrata possono essere distribuite facilmente ai diversi *server* tramite bilanciamento del carico.
- **Diverse rappresentazioni delle risorse**: ogni risorsa può avere diverse forme di visualizzazione. In base a cosa il *client* richieda, deve poter essere consegnato in diversi linguaggi o formati come HTML, JSON o XML.

- **Hypermedia**: la messa a disposizione delle risorse avviene tramite *hypermedia*, per esempio sotto forma di attributi "href"- e "src" nei documenti HTML o tramite elementi JSON e XML, definiti per le rispettive interfacce. Di conseguenza il *client* usa un'API REST esclusivamente tramite URL, secondo il principio *Hypermedia as the Engine of Application State (**HATEOAS**)*, che il *server* mette a disposizione, senza aver bisogno di alcuna nozione aggiuntiva sull'interfaccia.

D.2 Web API

Una Web API[6] è un'*Application Programming Interface* utilizzata da *Web Server* e *Web Browser*. A livello *server-side* una Web API consiste in una serie di *endopoints* esposti pubblicamente, ognuno utilizzato per ottenere uno specifico scambio di informazioni attraverso il Web e l'utilizzo dei metodi HTTP. Una Web API che rispetta i principi dell'architettura REST è definita **RESTful**.

ASP NET Web API è un *framework* che consente di creare facilmente servizi HTTP in grado di raggiungere un ampio numero di *client* ed è la piattaforma ideale per creare applicazioni RESTful nel *framework* .NET .

Bibliografia

[1] *http://docs.topshelf-project.com/en/latest/overview/commandline.html.*

[2] *http://www.di-srv.unisa.it/ ads/corso-security/www/CORSO-9900/oracle/oracle1.htm.*

[3] *https://bitoftech.net/2014/10/27/json-web-token-asp-net-web-api-2-jwt-owin-authorization-server/.*

[4] *https://docs.microsoft.com/it-it/aspnet/signalr/overview/getting-started/introduction-to-signalr.*

[5] *https://docs.microsoft.com/it-it/nuget/install-nuget-client-tools.*

[6] *https://en.wikipedia.org/wiki/Web_API.*

[7] *https://it.wikipedia.org/wiki/Dependency_injection.*

[8] *https://it.wikiversity.org/wiki/C_sharp.*

[9] *https://medium.com/@fabiogolfarelli/jwt-incrementiamo-la-sicurezza-con-i-json-web-tokens-cd0f2f9880da.*

[10] *https://www.creative-tim.com/.*

[11] *https://www.ionos.it/digitalguide/server/know-how/rest-la-soluzione-http-per-servizi-web/.*

[12] *https://www.mrwebmaster.it/javascript/introduzione-angular-cli_12717.html.*

[13] *https://www.video-corsi.com/creareapp/angular_2_e_web_app.htm.*

Ringraziamenti

Voglio ringraziare tutte le persone che mi sono state vicine durante questo percorso.

Ringrazio la mia famiglia: mio padre, mia madre, mia sorella, per avermi sempre sostenuto, per aver creduto in me in ogni istante, per aver sacrificato loro stessi per il mio bene, per aver sostenuto le spese economiche del percorso. Grazie, siete la mia vita.

Ringrazio la mia ragazza, Sara, diventata un punto di riferimento all'interno della mia vita. Grazie per la tua contagiosa intraprendenza, grazie per la grinta che mi dai ogni giorni semplicemente guardandoti, grazie per tutte le volte che abbiamo studiato assieme, grazie per aver fatto sempre il tifo per me e avermi consolato nei momenti difficili. Grazie amore mio.

Ringrazio la famiglia della mia ragazza: Stefano, Cristina, Stefania, Andrea, Mattia, Marta, nonna Mariola, nonna Graziella, Sirio. Grazie per avermi sostenuto in questo percorso e aver sempre creduto in me, grazie per avermi trattato come un figlio/fratello, curandovi in ogni momento del mio bene e della mia felicità. Grazie.

Ringrazio il Gruppo Loccioni, in particolar modo Luca Vescovi e tutto il team Ricerca e Sviluppo *Humancare*, Francesco De Stefano, Claudio Loccioni. Grazie per avermi dato l'opportunità di effettuare questo progetto, grazie per aver creduto in me sin dal nostro primo incontro. Grazie per quello che fate ogni giorno. Grazie.