# Introduzione

MPIntranet è un’applicazione web che ha lo scopo di catalogare, gestire e rendere fruibili informazioni associate alle produzione dell’aziende Metal plus e Top Finish.

La scelta di non inserire queste informazioni in un sistema gestionale ERP ma di sviluppare una applicazione dedicata è dovuta ai seguenti fattori:

* Difficoltà nel reperire e comprendere quali informazioni sono da salvare
* Difficoltà nel capire a chi queste informazioni devono essere messe a disposizione
* Difficoltà nel capire come rendere disponibili le informazioni raccolte.

La scelta di una applicazione stand alone permette di

* Effettuare modifiche all’architettura indipendentemente dall’architettura scelta
* Costi di integrazione ridotti o assenti
* Nessun rischio di influire sui processi gestiti dal gestionale

Le informazioni che l’applicazione dovrebbe gestire sono:

* Informazioni associate all’impianto galvanico
* Informazioni associate ai processi galvanici
* Scheda articolo: insieme di informazioni associate ad un articolo in produzione, deroghe, blocchi, certificati,…
* Reportistica aziendale su budget e carico di lavoro
* Scadenziario delle manutenzioni
* Preventivatore

Lo scopo che si pone l’applicazione è quello di raccogliere le informazioni non gestite dal sistema gestionale di produzione (ERP), integrarle e creare un sistema coerente e verificato di dati da cui ogni utente può estrarre l’informazione di cui ha bisogno certo che rappresenta l’informazione ufficiale dell’azienda.

Per rendere tali informazioni disponibili su ogni dispositivo presente in azienda si è scelto di pubblicarle usando C# ASP.Net MVC, per alcune applicazioni destinate ad un numero limitato di utenti e che richiedono l’uso di maggiori risorse di memoria o di calcolo si è scelto di andare verso la soluzione delle Windows Form classiche.

# Architettura

Dal punto di vista dell’architettura si è scelto una soluzione ‘ibrida’ in cui le normali classi della modellazione ad oggetti sono state spezzate in due parti: un model che contiene soltanto i dati destinati all’interfaccia web, una classe di metodi e funzioni che operando sui model esegue le operazioni necessarie per la logica di business.

Inoltre ogni singolo blocco funzionale dell’applicazione è stato isolato dagli altri in modo da essere completamente indipendente dagli altri elementi e quindi spostato o modificato senza doversi preoccupare delle conseguenze per gli altri elementi. Questa scelta, se da un lato ha fornito una notevole flessibilità, dall’altro ha imposto la duplicazione (se non la quadruplicazione) di alcuni comandi e logiche comuni a più blocchi funzionali.

Scendendo nel dettaglio andiamo ad analizzare come è stata organizzata la soluzione.

## MPIntranetWeb

E’ il progetto che contiene il sito web principale dell’applicazione. Si tratta di un normale progetto ASP.Net MVC. Il progetto contiene 10 controller che corrispondono a 10 aree funzionali separate:

### ControllerBase

È un finto controller, nel senso che rappresenta la classe base da cui devono derivare tutti i controller del sito. Questo controller di base mette a disposizione un set di comandi e di funzionalità come

* Gestione del token di autonticazione
* La gestione delle abilitazioni al menu del sito
* Il mapping delle cartelle applicative per l’utente ( ad esempio mappa la cartella delle immagini del gestionale RVL)
* L’estrazione del nome dell’utente connesso
* Gestione accentrata delle eccezioni ed invio al controller della pagina di erroe
* Verifica delle abilitazioni dell’utente connesso ai web method del controlle figlio

### AccountController

Gestisce la fase di login e logout dell’utente. Gestisce il menu delle abilitazioni dell’utente e la registrazione di nuovi utenti. Gestisce l’abilitazione dell’utente al menu delle funzionalità.

### AnagraficaController

Gestisce alcune anagrafiche come inserimento modifica e cancellazione dei brand, colore, tipo documento, materiali

### ArticoloController

Gestisce le principali funzionalità sugli articoli come, scheda articolo, associazione processo galvanico articolo, ricerca degli articoli, creazione nuovi articoli, cancella articoli,

### DocumentController

Gestisce il salvataggio, la cancellazione, l’estrazione e visualizzazione dei documenti che possono essere associati ai vari elementi del sito come articoli, processi, impianti.

### ErrorPageController

Gestione delle pagine di errore. E’ possibile creare pagine di errore dedicate a seconda del tipo di errore riscontrato ad oggi sono presnti solo due pagine: errore generico ed utente non autorizzato

### GalvanicaController

Gestisce gli impianti galvanici ed i relativi componenti come le vasche. Atraverso questo controller si definiscono anche i processi standard delle finiture.

### HomeController

Semplice landing page dopo il login. Gestisce il menu a sinistra.

### ManutenzioneController

Gestisce le anagrafiche e lo scadenziario della manutenzione ordinaria e straordinaria.

### ReportController

Produce i vari report aziendali

## MPIntranet.Business

Questo progetto contiene le classi responsabili di sviluppare la logica di business. Per ogni controller è stata creata una classe che ospita la relativa logica di business; in genere il controller e la classe di business hanno lo stesso nome

## MPIntranet.Common

In questo progetto sono salvate le classi base usate dalle applicazioni windows form.

### MPIBaseForm

Classe base per le finestre di tipo MDI. Permette di recuperare il contesto di sicurezza, gestire la chiusura delle finestre figlie e disabilita i menu. Mostra anche la finestra standard delle eccezioni

### ChildBaseForm

Finestra base per le finestre figlie di una MDI che deriva da MPIBaseForm. Permette di recuperare il contesto salvato nella finestra padre.

### ExceptionFrm

Finestra delle eccezioni.

## MPIntranet.DataAccess

Sistema di classi che permette di accedere al database.

Per ogni classe definita nella logica di business viene creata una cartella, che ha lo stesso nome della classe di business a cui si riferice, che a sua volta ospita due classi:

* Una classe il cui nome finisce con business che si occupa di gestire le transazioni
* Una classe che termina con adapter che esegue i comandi verso il database

Queste due classi derivano da classi MPIntranetAdapterBase e MPIntranetBusinessBase che contengono gli strumenti per creare la connessione al database, gestire i comandi e la transazione.

In breve il sistema si basa su l’attributo DataContext associato al metodo esposto dalla classe business. Tramite l’attributo viene attivata una classe che apre lo stack dell’applicazione e va a vedere se esiste una connessione verso il database. Se ne trova una la recupera e la usa per i comandi di interrogazione verso il database, se non c’è ne viene creata una connessione e viene salvata nello stack per condividwerla con le altre classi.

Questo permette di recuperare sempre la stessa connessione e di mettere in transazione facilmente i comandi. Infatti per mettere un metodo in transazione basta mettere nell’attributo DataContext la proprietà true.

La classe Adapter invece gestisce la connessione verso il database e permette di passare da SQL Server a Oracle in modo automatico, se si utilizzano alcune semplici regole di scrittura delle query che permettono di gestire i paramtri delle query.

## MPIntranet.Entities

Contiene semplicemente i vari dataset creati per ospitare i dati ritornati dal database. In genere si crea un dataset per ogni classe di business. Ogni dataset contiene poi le datatable che gli sono necessarie; può accadere che una stessa datatable sia presente in più dataset. Questa duplicazione permette di ‘staccare’ in qualsiasi momento un dataset senza doversi preoccupare di quello che accade agli altri, anche se poi la gestione dei dati viene duplicata.

## MPIntranet.Helpers

Contiene classi di uso generale come helper per leggere e creare file excel, pdf, eseguire JSON, …

## MPIntranet.Models

Contiene la definizine dei modelli che sono consumati dai controller per produrre le pagine html. Per scelta si cerca di non far accedere direttamente il controller al dataset, ma si preferisce tenerli separati. Nella classe di business viene il contenuto di un dataset viene convertito nel relativo model e questo poi viene inviato al controller per essere utilizzato.

## MPIntranet.ScheduledServices

Progetto di un servizio windows schedulato … in corso di realizzazine

## MPIntranet.Security

Questo progetto gestisce la sicurezza dell’intera solution.

Per le applicazioni web si è scelto di far eseguire il login all’utente in modo che chiunque, da qualsiasi postazione in azienda, possa accedere al sistema con il proprio profilo; per il login l’utente deve usare le proprie credenziali di domino. In questo caso l’accesso ai menu viene gestito tramite apposita finestra all’interno dell’applicazione web.

Per le applicazioni windows form si è scelto di usare il profilo di dominio con cui l’utente si è loggato a windows; in questo caso il profilo e le autorizzazioni vengono gestite tramite gruppi sul dominio di login.

## MPIntranet.ServiceManagers

Gestice le classi a cui il servizio windows schedulato demanda l’esecuzione dei comandi.

## MPPreventivatore

Applicazione windows form del preventivatore.

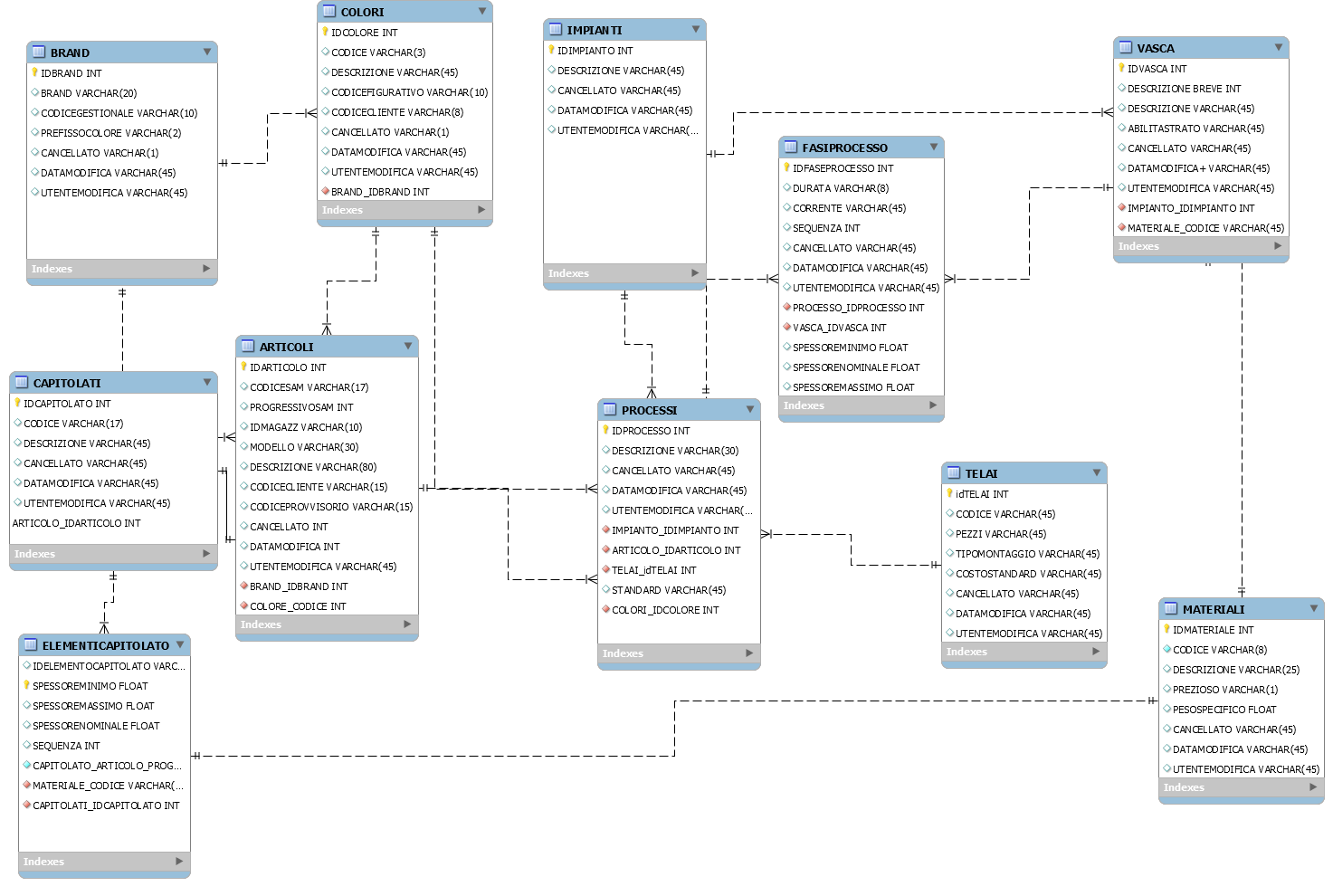
# Regole di sviluppo

Di seguito vengono riportate le regole di sviluppo che sono definite

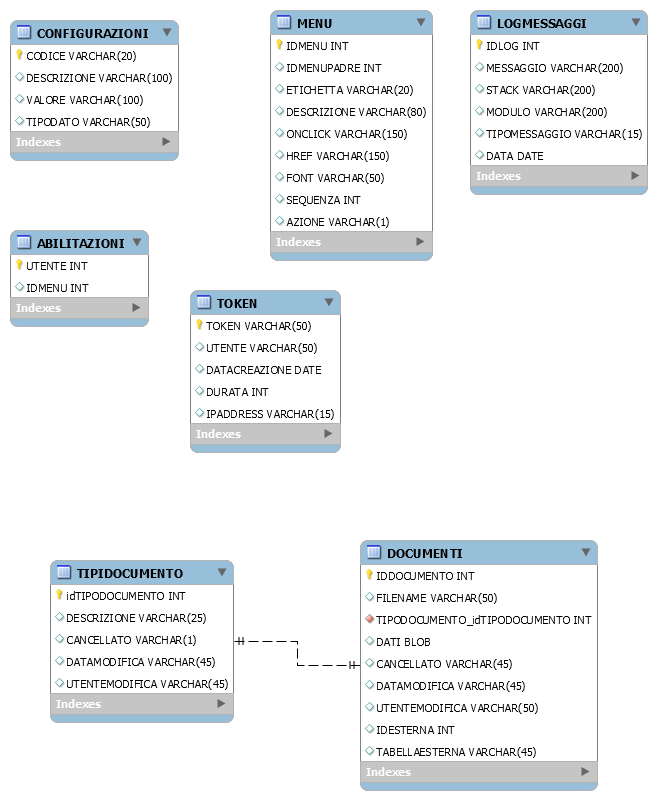
* Ogni voce di menu dell’applicazione web raccoglie un set di funzioni coerenti in relazione ad un preciso problema di business come ad esempio galvanica, manutenzione, anagrafica…
* Si crea un controller per ogni voce di menu dell’applicazione web
* Per ogni voce di menu che identifica una possibile funzione viene creato nel controller un metodo che ritorna la relativa vista. La vista in genere è molto scarna.
* Quando viene caricata una vista di un comando, in genere è presente una funzione javascript $function per eseguire la prima ricerca o il primo caricamento dei dati
* Ogni caricamento di dati avviene andando a richiamare uno specifico web method tramite apposita chiamata ajax. Il web method ritorna una partial view che la chiamata ajax usa per popolare una div
* Ogni operazione di modifica inserimanto o cancellazione nell’applicazione web è associata ad un apposito comando gestito nella finestra dei abilitazione dell’utente rimanda ad uno specifico web method nel relativo controller
* Le righe nelle tabelle non vengono mai cancellate fisicamente ma solo logicamente usando l’apposita colonna CANCELLATA
* Per ogni tabella creata nel database viene creata la relativa tabella di log che viene alimentata da un serie di trigger. Ogni modifica ai dati di una tabella alimenta con una riga nuova la relativa tabella di log
* Ogni tabella ha sempre una primary key per permettere la scrittura dei dati in forma semplificata tramite il ComnadBuilder
* Ogni tabella che ha riferimenti con altre tabelle espone sempre delle foreign key
* Ad ogni controller corrisponde una classe che ospita la logica di business.
* I dati recuperati finiscono sempre in un dataset
* Il dataset non può essere esposto nel controller, per arrivare al controller i dati vengono convertiti nel relativo model

# DB SCHEMA

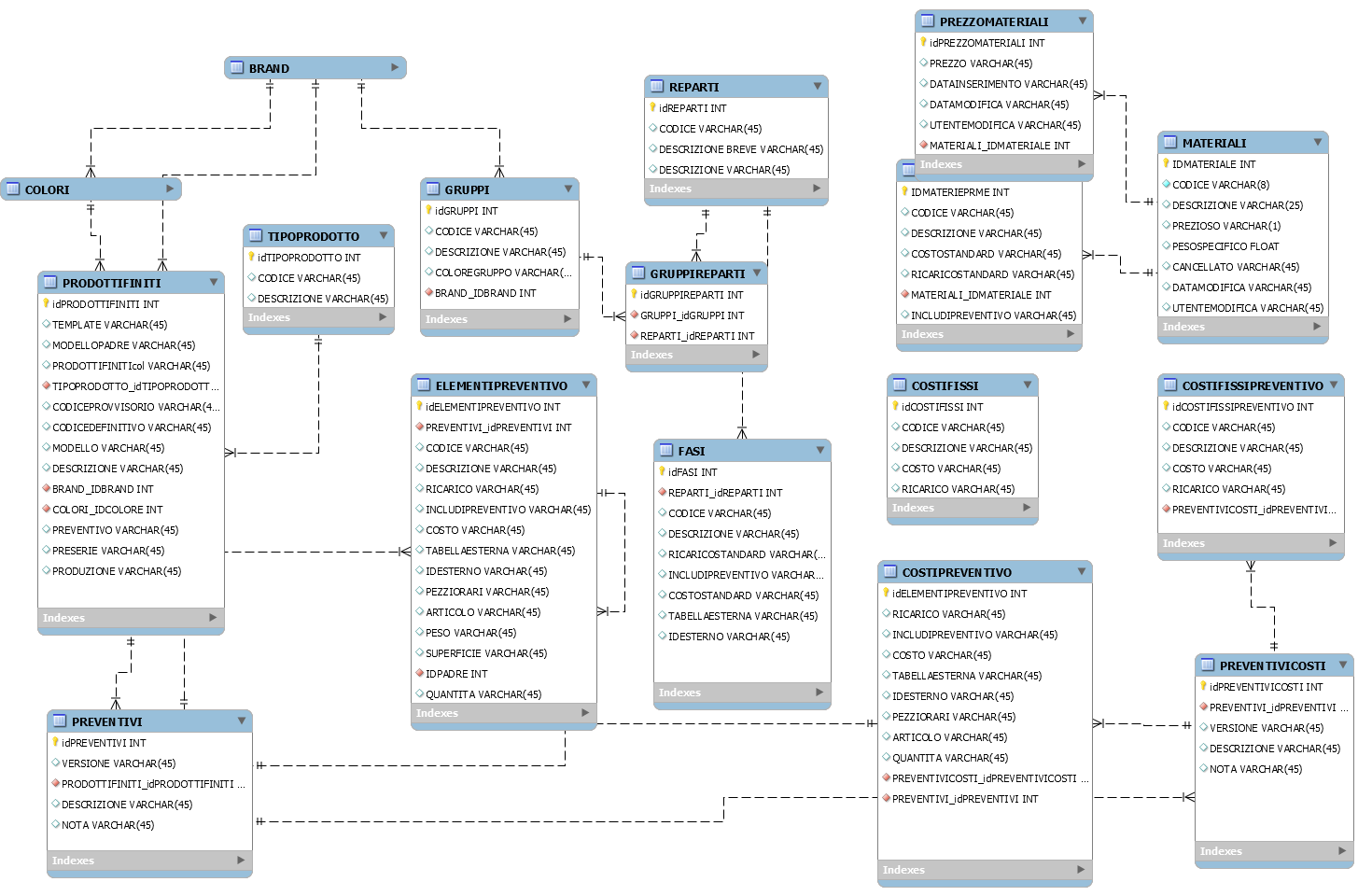
## GALVANICA



## SISTEMA



## PREVENTIVATORE



# PREVENTIVATORE

Ha lo scopo di permettere di stimare il costo di un articolo e successivamente permettere di definirne il prezzo inteso come costo più ricavo.

L’applicazione è una windows form di tipo MDI composta da una MainForm e da una serie di finestre figlie ognuna destinata ad una singola operazione.

## Processo di creazione di un preventivo

Il processo di creazione di un preventivo passa attraverso i seguenti passi:

1. Creazione dell’articolo oggetto del preventivo.
2. Creazione della distinta base del preventivo
3. Creazione della distinta di costo del preventivo
4. Definizione del prezzo

Vediamo nel dettaglio i singoli passi

### Creazione articolo preventivo

Per creare un nuovo articolo occorre selezionare dal menu Preventivi la voce Gestisci prodotto finito…

L’applicazione apre una finestra di ricerca per un articolo esistente in cui è presente il pulsante Nuovo prodotto finito.

Premendo nuovo prodotto finito si apre una maschera attraverso cui vengono raccolte le informazioni minime necessarie per definire l’articolo:

* Codice: codice interno associato all’articolo
* Modello: codice modello usato nel gestione RVL. Serve per avere un collegamento con il gestionale
* Descrizione
* Brand: proprietario del prodotto
* Codice provvisorio: codice provvisorio del brand
* Codice definitivo: codice definitivo usato dal brand
* Colore: codice colore associato alla finitura. Il colore dipende ovviamente dal brand ed è identificato da un codice interno chiamato codice colore figurativo in cui i primi due caratteri identificano il brand e gli altri la finitura all’interno del brand.
* Tipo prodotto: indica la tipologia del prodotto
* Esistono poi tre checkbox che indicano a che livello di avanzamento è arrivato l’articolo. Il primo livello è preventivo, l secondo preserie ed infine produzione.

Una volta creato l’articolo è necessario aprirlo in modifica per assegnare l’immagine.