 Trello Web Application

## Architettura

## La **View** è suddivisa in questa maniera:

## Quella principale mostrerà una lista delle Card presenti nella board di Trello;

## una View secondaria mostrerà i dettagli della singola card indicando il suo status, (anche se archiviate), le sue checklist, la scadenza, gli allegati con la possibilità di scaricarli e le sue label;

* **Card Controller** che si occuperà di interrogare le API di Trello, esporre l’interfaccia per far sì che la Web App e il Report Controller possano usufruire di queste info;
* **Report Controller** che si occuperà dell’estrazione del contenuto della Board Trello e salvarlo su file di tipo PDF o di tipo Excel in base a ciò che si seleziona.
* Creazione **Models** necessari per la modifica e la ricezione dei dati presenti nel board di Trello.
* Cartella **Utilities** contenente tre classi: *Api.cs* e *PopolateModel.cs* necessarie al corretto funzionamento di *CardController*, *ReportMethod.cs* necessaria per il corretto funzionamento del *ReportController*.

## Suddivisione dei compiti

·        Nunzio: API Controller

·        Myslim: View;

·        Piero: Report Controller

## Tecnologie e librerie utilizzate

* Come IDE abbiamo utilizzato VisualStudio Community 2017, per l’applicazione web abbiamo utilizzato l’architettura MVC (Model View Control).
* Per esportare l’HTML della pagina in un PDF, si è ricorso alla libreria **Rotativa** (<https://www.nuget.org/packages/Rotativa>), con la quale data una View in HTML, e possibile convertirla completamente in un PDF, attraverso l’oggetto *ActionAsPDF()* inserito in un’apposita funzione.
* Per esportare l’HTML della pagina in un foglio di calcolo Excel, si è ricorso alla libreria **EPPlus** (<https://www.nuget.org/packages/EPPlus>), con la quale data una View in HTML, e possibile esportarla completamente in un file Excel, prendendo i dati da un file JSON e inserendoli nelle celle definite nel ReportController
* **TrelloUtilities**: Libreria creata da noi che contiene i metodi e i modelli utilizzati nella web app Generalizzati per farla usare o usarla in terzi progetti

## Architettura della Web APP (Mokup)

# Funzionamento dell’applicazione

## Web Page

Una volta lanciata l’applicazione ci si trova nella schermata della home, cliccando su un link “*Trello Card List*” posto in alto a sinistra, si verrà trasportati in una schermata che mostrerà un elenco delle Card presenti sul board di Trello con la possibilità di vedere i dettagli della singola card premendo sul link “*details*” posto sulla stessa riga della card desiderata. Una volta cliccato sul link *Details* si passerà ad una seconda schermata nella quale si visualizzeranno sotto forma di tabella i dettagli della card selezionata, ossia:

* **Id**, l’identificativo
* **Stato**, cioè la colonna nella quale si trova, anche nel caso essa sia stata archiviata;
* **Lista di Checklist**, le *Checkbox* presenti nella card assieme alle proprie opzioni;
* **Label (o eventualmente una lista),** ossia la priorità data alla Card, indicandola attraverso un Name ed un Color;
* **Allegati,** ossia i vari allegati presenti all’interno della Card (quest’ultimi vengono visualizzati con un link avente il nome dell’allegato, attraverso il quale sarà possibile scaricare l’allegato).
* **Data di Scadenza,** ossia la data entro la quale bisogna portare a termine l’attività espressa all’interno della Card;

Subito dopo la tabella con i dettagli della card ci sono due pulsanti: EXPORT PDF e EXPORT EXCEL che una volta cliccati chiamano rispettivamente le rispettive azioni (ExportPDF/ExportExcel) del ReportController che permettono all’ utente di scaricare un file in formato Pdf o Excel in base a ciò che si seleziona contenente i dati della pagina html.

* Export PDF: Chiama la rispettiva azione (ExportPDF) del ReportController che crea e scarica un file PDF con il contenuto della pagina html.
* Export Excel: Chiama la rispettiva azione (ExportExcel) del ReportController che crea e scarica un file Excel con il contenuto della pagina html.

### Funzionalità inserimento commento alla card:

®Aggiunto bottone giallo “Aggiungi commento” subito dopo la tabella dei dettagli della card che una volta premuto fa comparire tramite **Javascript** un campo di inserimento testo ed un bottone che conferma l’invio del commento alla card di Trello (che prima era “nascosta”), una volta premuto il tasto di conferma viene chiamata l’azione *AddComment* di *CardController* che esegue una chiamata POST alle api di Trello ed inserisce all’interno della card su cui ci si trovava il commento scritto sul campo di inserimento.

## Card Controller

Questa parte si occupa di recuperare i dati di Trello e trasferirli nei nostri modelli attraverso i file **Api** e **PopolateModel**:

®Nel file *Api* vengono gestite le chiamate alle API di Trello, sia in GET che in POST (per i commenti). Per farlo ci siamo avvalsi delle librerie **Net** e **Script.Serialization**, necessaria per deserializzare i file JSON recuperati grazie alla prima, e trasformarla in un oggetto tramite la seconda.

®In *PopolateModel* richiamando Api popolo l’oggetto con alcune modifiche rispetto al JSON originale, come ad esempio le date, cambiandone il formato della stringa, in poche parole riunendo e modificando i vari oggetti ottenute grazie alla API.

## Report Methods

Questa sezione si occupa delle funzioni riportate sopra ed esporta i dati rispettivamente in un file **PDF** ed in un file .**xls**, in questa maniera:

®Per esportare la View desiderata in un file PDF, inizialmente si è optato per utilizzare la libreria **Rotativa** e la classe **ActionAsPDF(),** alla quale veniva data la *View* nella quale recarsi, ne creiamo una di intermezzo nella quale vengono inseriti solamente i dati desiderati, la funzione prende il codice (più un modello per avere i dettagli di una card) ed infine scaricare tale file nel PC, con tutte le caratteristiche visualizzate nel browser;

® Nel caso dell’inserimento in un file Excel, si richiede il modello di della *Card*  o della *ListCard*; una volta creato il modello, la libreria di **NuGet** utilizzata (**EPPlus**), creiamo un file Excel (attraverso *ExcelPackage*() ), all’interno del quale vi sarà un foglio di calcolo(*ExcelWorkSheet*), nel quale abbiamo strutturato una tabella coi vari campi, e nel caso quel campo risulti vuoto, comparirà un messaggio composto da *NO* seguito dal titolo di quel campo.

In aggiunta, utilizzando la libreria **Quartz**, è stato creato un progetto aggiuntivo che si occupa di inviare all’utente, in maniera regolare, un’email contenente un file Excel contenente tutte le Card aggiornate. Questo è possibile grazie a *JobScheduler*, il quale dice ogni quanto deve essere eseguito il Job, ossia la funzione designata, in questo caso la creazione del corpo della mail e l’invio di quest’ultima.

# Punti critici

Le difficoltà principali riscontrate sono state nell’utilizzo dei merge, poiché a volte andava a buon fine, ma le modifiche non avvenivano, inoltre non siamo riusciti ad evitare le chiamate alle API per ogni refresh della pagina, anche quando dal tempo t0 al tempo t1 non è cambiato nulla, poiché per assegnare il *callbackURL* era necessario “ospitare” la pagina. Abbiamo riscontrato difficoltà anche nel CSS poiché certi punti del layout erano in comune e create da visual studio. Quindi non era sempre comprensibile la gerarchia.

# Limitazioni per il cliente

* Scaricamento automatico delle immagini
* Chiamata alle api solo se necessario. Non si è potuto fare poiché non ostando il nostro web site, non possiamo ricevere le risposte dal Webhook Trello.
* In certi punti non totalmente responsive per il punto critico delle gerarchie qui sopra esposto