

UNIVERSITÀ DEGLI STUDI DI CAMERINO

SCUOLA DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA (L-31)



# Progettazione e sviluppo di un applicazione desktop per l'analisi di bilancio di un'azienda

Laureando

**Danilo Paoletti**

Matricola 085353

Relatrice

**Dr.ssa Barbara Re**

Anno accademico 2017-2018



# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Bilancio di un'azienda</b>	<b>3</b>
1.1 Stato Patrimoniale . . . . .	4
1.1.1 Riclassificazione Funzionale (Operativo) . . . . .	5
1.1.2 Riclassificazione Finanziario . . . . .	5
1.2 Conto Economico . . . . .	7
1.2.1 Riclassificazione al Valore Aggiunto . . . . .	8
1.3 Forecast . . . . .	8
<b>2 Analisi e progettazione</b>	<b>10</b>
2.1 Descrizione del problema . . . . .	10
2.2 Analisi dei requisiti . . . . .	11
2.2.1 Requisiti funzionali . . . . .	12
2.2.2 Requisiti non funzionali . . . . .	13
2.3 Use case diagram . . . . .	14
2.4 Activity diagram . . . . .	15
2.5 Sequence diagram . . . . .	17
<b>3 Sviluppo</b>	<b>20</b>
3.1 Sviluppo del Back-End . . . . .	21
3.1.1 Utilizzo delle API Electron . . . . .	22
3.1.2 Utilizzo API Node.js . . . . .	24
3.2 Sviluppo del Front-End . . . . .	28
3.2.1 Tecnologie usate nello sviluppo del front-end . . . . .	30
3.3 Packaging dell'applicazione . . . . .	34

<b>4</b>	<b>Possibili sviluppi futuri</b>	<b>37</b>
	<b>Conclusione</b>	<b>39</b>



# Introduzione

L'analisi di bilancio è una delle procedure attraverso cui vengono determinati i fattori che determinano i guadagni, le risorse attraverso cui l'azienda ripaga i debiti e il grado di indebitamento. In questa tesi verrà mostrato il software da me realizzato per il processo di analisi di bilancio attraverso la riclassificazione dello Stato Patrimoniale, del Conto Economico e del forecast per indici di previsioni.

Inizialmente verrà descritto il contesto economico dell'analisi di bilancio, successivamente verranno trattate la fase di analisi e progettazione che ha preceduto la realizzazione del software, descrivendo attraverso diagramma Unified Modeling Language (UML) [1] i vari requisiti, vincoli e funzionalità del prodotto.

Successivamente saranno descritte le tecnologie utilizzate e si entrerà nel merito della realizzazione di tutte le funzionalità richieste.

In conclusione si farà riferimento ad eventuali sviluppi futuri che potrebbero essere implementati nel software realizzato.



# Capitolo 1

## Bilancio di un'azienda

L'analisi di bilancio, è uno strumento tecnico, che permette di studiare i risultati d'esercizio derivanti dalle attività aziendali, focalizzando l'attenzione su:

- situazione economica: condizioni che determinano la produttività dell'azienda e costituisce la capacità di determinare utili d'esercizio;
- situazione finanziaria: condizioni che determinano la capacità di far fronte ai debiti a breve scadenza e all'accesso a prestiti di finanziamento;
- situazione patrimoniale: condizioni che determinano il rapporto tra le immobilizzazioni e i debiti verso terzi.

In altre parole, con lo strumento dell'analisi di bilancio, si cerca di individuare nelle diverse aree (economica, finanziaria e patrimoniale) i primi spunti di riflessione per l'analisi della gestione.

Il bilancio di un'azienda, nella sua forma più semplice, è formato da due documenti: lo **Stato Patrimoniale**, che rappresenta in modo sintetico la composizione qualitativa e quantitativa del patrimonio della società al giorno della chiusura dell'esercizio, e il **Conto Economico**, che espone il risultato economico dell'esercizio attraverso la rappresentazione dei costi e degli oneri sostenuti, nonché dei ricavi e degli altri proventi conseguiti nell'esercizio;.



## 1.1 Stato Patrimoniale

Lo Stato Patrimoniale, definito dall'art. 2424 del Codice Civile, rappresenta la situazione patrimoniale ad una certa data di un'impresa ed è suddiviso in due sezioni contrapposte chiamate Attività e Passività.

In particolare, le attività sono distinte secondo criteri di realizzabilità ed esigibilità, in :

- crediti verso soci per versamenti ancora dovuti, con separata indicazione della parte già richiamata;
- immobilizzazioni: rappresentano gli investimenti destinati per un periodo superiore a 12 mesi, a trasformarsi in denaro. Sono iscritte al netto dei fondi di ammortamento;
- attivo circolante: rappresentano gli investimenti destinati per un periodo inferiore a 12 mesi, a trasformarsi in denaro. Iscritto al netto di eventuali fondi di svalutazione;
- ratei e riscontri;

Le passività, nell'analisi di bilancio, vengono distinte secondo le scadenze di pagamento in:

- patrimonio netto;
- fondi per rischi e oneri;
- trattamento di fine rapporto di lavoro subordinato;
- debiti;
- ratei e riscontri;

Si possono individuare due criteri di riclassificazione dello stato patrimoniale per acquisire migliori informazioni sulle dinamiche aziendali: il criterio funzionale e quello finanziario.

### 1.1.1 Riclassificazione Funzionale (Operativo)

Secondo il criterio funzionale invece le attività (impieghi) e le passività (fonti) sono riclassificate in base all'area gestionale di appartenenza: area caratteristica/operativa (nella quale ricomprendere se marginale anche quella accessoria), comprendente tutti i valori attinenti il core business; area finanziaria, comprendente tutti i valori relativi alla negoziazione di liquidità.

Gli impieghi sono pertanto suddivisi in:

- attività operative: assets materiali e immateriali, crediti operativi, rimanenze, ratei e risconti;
- attività finanziarie: investimenti finanziari (a breve e a medio-lungo), crediti finanziari e disponibilità liquide.

Le fonti sono invece suddivise in:

- patrimonio netto: grandezza non riconducibile né all'area operativa né a quella finanziaria;
- passività operative: fondi rischi ed oneri, debiti operativi e ratei e risconti;
- passività finanziarie: ovvero i debiti finanziari a prescindere dalla scadenza.

Lo stato patrimoniale classificato secondo la logica funzionale mira a verificare l'equilibrio fra investimenti e fonti di finanziamento, e quindi di ausilio a sviluppare l'analisi della solidità

### 1.1.2 Riclassificazione Finanziario

Con il criterio finanziario le attività (impieghi) sono classificate e raggruppate secondo il loro grado di liquidabilità, ovvero in funzione della loro capacità di trasformarsi in liquidità in tempi più o meno rapidi, mentre le passività (fonti) in base alla loro durata temporale, ovvero in base alla loro velocità di estinzione.

L'arco temporale preso a riferimento con termine congruo per circoscrivere il breve dal medio-lungo termine corrisponde a 12 mesi.

Gli impieghi sono pertanto suddivisi, in funzione alla loro effettiva possibilità di trasformarsi in liquidità, in:

- attività correnti, atte ad essere liquidate in un arco temporale inferiore a 12 mesi, ovvero assets destinati alla vendita entro 12 mesi, attività finanziarie detenute a scopo di negoziazione, crediti in scadenza entro 12 mesi, rimanenze (per la parte che presenta un tasso di rotazione inferiore a 12 mesi), liquidità, ratei e risconti;
- attività non correnti, destinate a rimanere vincolate nel medio-lungo periodo, ovvero assets materiali, immateriali e finanziarie (eccetto quelle destinate alla vendita nel breve termine), crediti con scadenza oltre il 12 mesi, rimanenze (per la parte che presenta un tasso di rotazione inferiore a 12 mesi).

Le fonti sono invece suddivise in:

- patrimonio netto, grandezza vincolata e quindi fonte di lungo periodo;
- passività correnti, destinate al rimborso entro i 12 mesi, ossia: debiti a breve (comprese le rate a breve di finanziamenti a medio-lungo termine), ratei e risconti passivi, fondi rischi ed oneri (per la parte che avrà manifestazione finanziaria nel breve periodo);
- passività non correnti, con scadenza superiore a 12 mesi, ossia: debiti a medio-lungo, risconti passivi pluriennali, fondi rischi ed oneri (per la parte che avrà manifestazione finanziaria oltre 12 mesi).

Lo stato patrimoniale classificato secondo la logica finanziaria permette di verificare la capacità dell'azienda di far fronte ai propri impegni di breve periodo con impieghi di egual durata (capitale circolante), ed è pertanto propedeutico all'analisi della liquidità;

## 1.2 Conto Economico

Il Conto Economico, definito dall'art. 2425 del Codice Civile, è l'elenco, ordinato per categorie, dei costi e dei ricavi di competenza dell'esercizio, ossia di competenza di quel lasso di tempo intercorrente tra la data di riferimento del bilancio attuale e quella del bilancio precedente.

Si intendono ricavi le vendite dei propri beni, gli interessi attivi o i fitti attivi. Sono, invece, esempi di costi gli acquisti, le utenze, le spese del personale, i fitti passivi le imposte e le tasse.

In tale contesto è bene far presente che l'Iva, ancorché un'imposta, non rappresenta un costo per la società.

Si tratta, infatti, di un debito o un credito che l'azienda ha nei confronti dell'Erario nel momento in cui compie una vendita o un acquisto e, per questo motivo, trova allocazione nel passivo o nell'attivo dello Stato Patrimoniale.

Esso ha una struttura a forma scalare e una classificazione dei costi per natura (invece che per destinazione). È formato da quattro sezioni (individuate con le prime lettere dell'alfabeto), più alcune voci che illustrano il risultato d'esercizio, ante e dopo le imposte.

Sezioni che compongono il conto economico:

- valore della produzione;
- costi della produzione;
- proventi ed oneri finanziari;
- rettifiche di valore di attività e passività finanziarie.

Il Conto Economico può assumere diverse forme e configurazioni. Le riclassificazioni più utilizzate, per l'analisi di bilancio sono:

- il conto economico a margine di contribuzione, che si basa sulla suddivisione dei costi operativi tra costi fissi e costi variabili;
- il conto economico a costo del venduto, che si basa sulla suddivisione dei costi operativi tra costi diretti e costi indiretti;

- il conto economico a valore aggiunto, che si basa sulla suddivisione dei costi operativi tra costi relativi alle risorse esterne e costi relativi alle risorse interne.

Nel software è stata presa in considerazione solo la riclassificazione a valore aggiunto.

### **1.2.1 Riclassificazione al Valore Aggiunto**

Il valore aggiunto, quale differenza tra ricavi operativi e costi operativi sostenuti per l'acquisto di risorse esterne, esprime la capacità dell'azienda di creare ricchezza per remunerare i fattori produttivi e i diversi portatori di interesse.

In particolare tale margine deve essere in grado di remunerare:

- il personale - costo del personale;
- gli investimenti - ammortamenti e svalutazioni;
- i finanziatori esterni - componenti finanziarie;
- gli eventi straordinari - componenti straordinarie;
- l'Amministrazione finanziaria - imposte.

Deve infine garantire un'adeguata remunerazione, tramite la distribuzione del risultato d'esercizio, ai soci e permettere con l'utile residuo non distribuito un adeguato autofinanziamento.

## **1.3 Forecast**

Il Forecast è un prospetto simile a un normale bilancio. A differenza di quest'ultimo, che registra valori a consuntivo, il Forecast è una tabella che contiene valori di natura previsionale. Nello specifico, dà una previsione che guarda avanti di qualche mese.

L'impostazione e lo scopo del Forecast sono simili a quelli del budget. Cambia però il periodo di analisi. Infatti, mentre il budget è una previsione fatta sull'anno successivo, il Forecast è una previsione sull'anno in corso.



# Capitolo 2

## Analisi e progettazione

### 2.1 Descrizione del problema

Si vuole realizzare un'applicazione desktop per l'analisi di bilancio di un'azienda per l'ordine dei commercialisti della provincia di Ancona. L'applicazione deve funzionare in tutti i vari tipi di sistema operativo, perciò il software deve essere cross-platform. Per poter realizzare un prodotto che sia cross-platform, ossia implementato per più piattaforme di elaborazione, la scelta delle tecnologie da utilizzare è stata fondamentale. Quando si vuole sviluppare un'applicazione desktop la prima cosa a cui si pensa è per quale sistema operativo svilupparla, quale linguaggio usare ed opzionalmente a quale base dati si ha bisogno di connettersi. Il mondo dell'informatica intanto si sta evolvendo verso l'utilizzo di applicazioni web per la loro semplicità di distribuzione e aggiornamento, la compatibilità con ogni sistema operativo e la superfluità di un'installazione, allora perchè non sfruttare questa tecnologia? Per questo la scelta è ricaduta su un compromesso valido, Electron. Definita dallo slogan "Build cross platform desktop apps with Javascript, HTML, and CSS", Electron è una libreria open source sviluppata dal team di GitHub che permette di sviluppare un'applicazione desktop utilizzando le stesse tecnologie di un sito web, unendo la potenza del browser Chromium, la flessibilità di Node.js e il più grande ecosistema di librerie open source al mondo: npm. In aggiunta a tutto questo la possibilità di pacchettizzare l'applicazione per Mac, Windows, e Linux con lo stesso risultato in termini di grafica e funzionalità.

L'applicazione realizzata, nello specifico, dovrà permettere all'utente che la

utilizza l'inserimento in input dei dati relativi all'Anagrafica, all'Analisi Qualitativa, dello Stato Patrimoniale e del Conto Economico di un'azienda. Nella fase di inserimento dei valori dello Stato Patrimoniale e del Conto Economico, il sistema calcola le varie riclassificazioni dello Stato Patrimoniale (Funzionale, Finanziario o entrambi, a seconda della scelta da parte dell'utente) e del Conto Economico (al Valore Aggiunto). L'utente deve avere la possibilità di salvare in corso d'opera il progetto e poterlo riaprire in un secondo momento. Inoltre, va implementata una funzione che permette la stampa in un file pdf del progetto.

## 2.2 Analisi dei requisiti

Al fine di esporre i requisiti in maniera rigorosa, verrà adottata la notazione MoSCoW, la quale fa uso delle seguenti etichette:

- Must Have: sottolinea un requisito di fondamentale importanza per il funzionamento del sistema;
- Should Have: indica un requisito importante ma non essenziale per il funzionamento del sistema;
- Could Have (o Nice to Have): indica un requisito non importante come i precedenti e volto principalmente a migliorare la soddisfazione del cliente.
- Won't Have: requisiti che non devono essere implementati (negazione di una funzione).



### 2.2.1 Requisiti funzionali

I requisiti funzionali descrivono le funzionalità del sistema software, in termini di servizi che il sistema software deve fornire, di come il sistema software reagisce a specifici tipi di input e di come si comporta in situazioni particolari.

ID	Descrizione	MoSCoW
1	Il programma deve permettere l'inserimento e la visualizzazione dei dati dell'azienda di cui si vuole analizzare il bilancio	Must have
2	Il programma deve permettere all'utente il salvataggio del progetto	Must have
3	Il programma deve permettere all'utente l'apertura di un progetto salvato in precedenza	Must have
4	Il programma deve permettere all'utente di scegliere quale tipo di riclassificazione per lo stato patrimoniale utilizzare	Must have
5	Il programma deve calcolare le riclassificazioni sia per lo stato patrimoniale che per il conto economico	Must have
6	Il programma deve permettere all'utente di poter inserire l'annualità di riferimento dell'analisi	Should have
7	Il programma deve permettere all'utente di poter stampare in un formato pdf del progetto	Should have
8	Il programma deve calcolare in automatico il forecast per le annualità previste dall'utente	Could have

Tabella 2.1: Requisiti funzionali

### 2.2.2 Requisiti non funzionali

Descrivono le proprietà del sistema software in relazione a determinati servizi o funzioni e possono anche essere relativi al processo:

- Caratteristiche di efficienza, affidabilità, safety, ecc.
- Caratteristiche del processo di sviluppo (standard di processo, linguaggi di programmazione, metodi di sviluppo, ecc.)
- Caratteristiche esterne (interoperabilità con sistemi di altre organizzazioni, vincoli legislativi, ecc.)

ID	Descrizione	MoSCoW
1	Il programma deve essere sviluppato per tutte le piattaforme di elaborazione	Must have
2	Il programma dovrebbe controllare i formati di immissione dei dati nella maschera di input	Should have
3	L'avvio e in particolar modo la ricezione dell'input da parte dell'utente, dovrebbe essere veloce	Should have
4	Il programma deve calcolare in maniera veloce le riclassificazioni	Should have
5	Il programma può essere adattabile alle differenti risoluzioni dello schermo (responsive)	Could have

Tabella 2.2: Requisiti non funzionali

## 2.3 Use case diagram

Gli Use Case Diagrams descrivono il comportamento funzionale del sistema, come visto dall'utente. Sono diagrammi di facile interpretazione dove abbiamo uno o più attori che possono attivare diverse funzioni/servizi all'interno di uno o più sistemi. Servono a riepilogare i dettagli degli utenti del sistema (noti anche come attori) e le loro interazioni con il sistema.

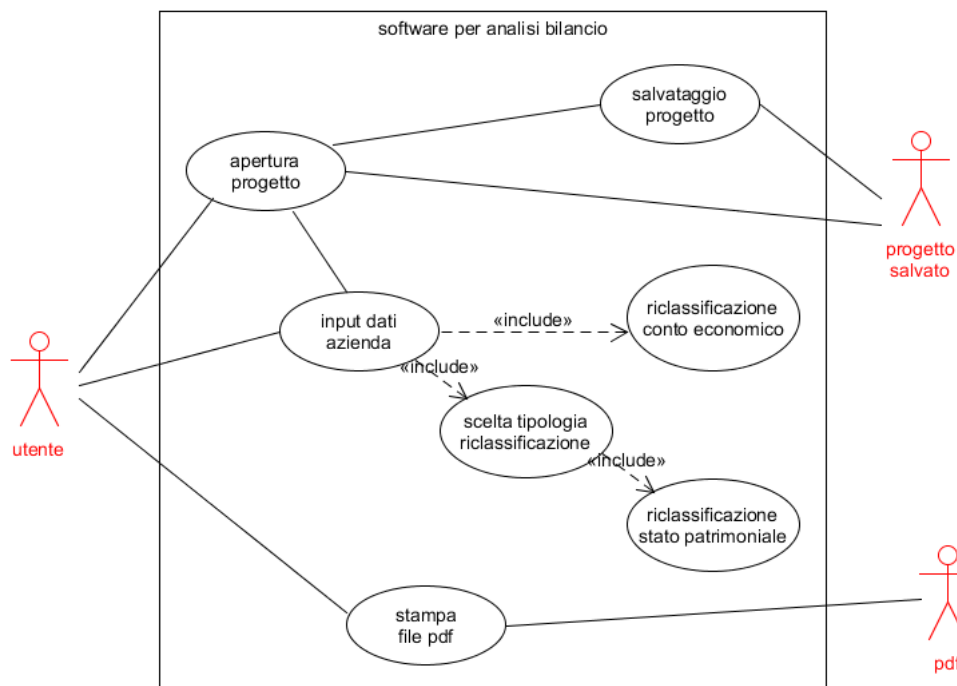


Figura 2.1: UML Use Case Diagram

Nel diagramma in figura 2.1 abbiamo tre attori, l'utente che inserisce i dati dell'azienda, il file contenente il progetto salvato e il file pdf generato dal software. I casi d'uso talvolta includono più passaggi al loro interno, in quei casi è indicata una freccia "include", come nel caso del "input dati azienda" che include "scelta tipologia riclassificazione" che a sua volta include "riclassificazione stato patrimoniale".

## 2.4 Activity diagram

L'Activity Diagram è un diagramma che definisce le attività da svolgere per realizzare una data funzionalità ed è spesso utilizzato come modello complementare dello Use Case Diagram.

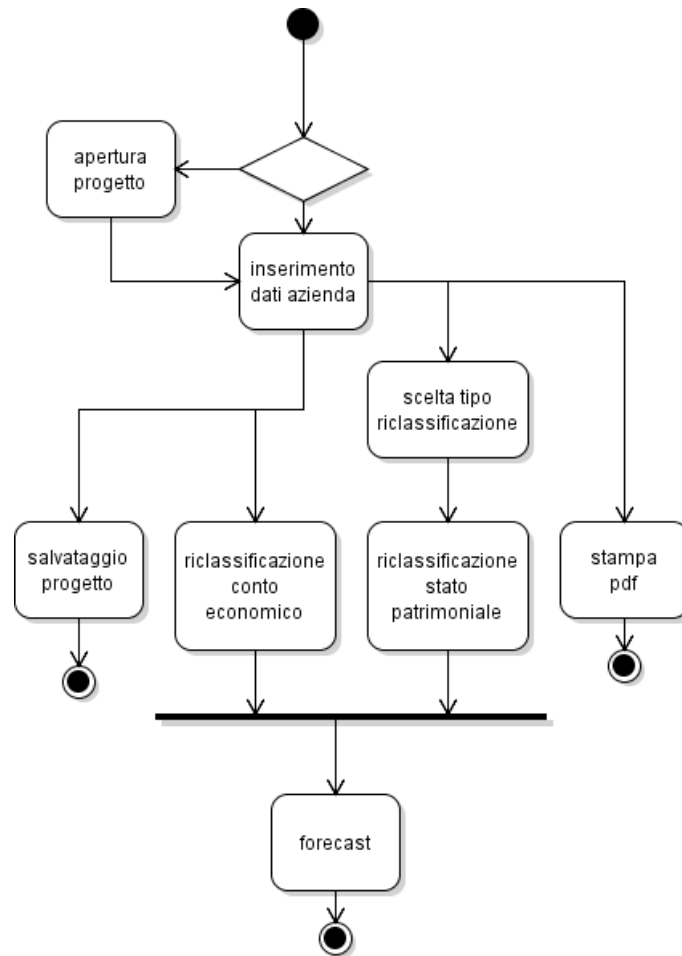


Figura 2.2: UML Activity Diagram

Nella figura 2.2 vediamo descritto il software come nello Use Case precedente, ma dal punto di vista delle attività da svolgere in sequenza per raggiungere un certo risultato. Il diagramma si legge a partire dal pallino in alto, detto nodo iniziale, seguono le azioni indicate dai rettangoli. I rombi indicano le decisioni, cioè le diverse direzioni che può prendere il flusso a seguito di certe condizioni o scelte, ad esempio all'avvio del software se si vuole aprire un progetto salvato precedentemente o inserire i dati per un nuovo progetto. Un altro

elemento dell'activity diagram è la "join", che indica che per passare all'azione successiva devono essere completate tutte le azioni in ingresso, come ad esempio per il calcolo del forecast, che avviene una volta inseriti i dati e quindi calcolate le riclassificazioni dello stato patrimoniale e del conto economico.

## 2.5 Sequence diagram

Il sequence diagram è uno strumento di modellazione visuale che mette in evidenza come i vari componenti di un sistema interagiscono tra di loro in relazione al tempo.

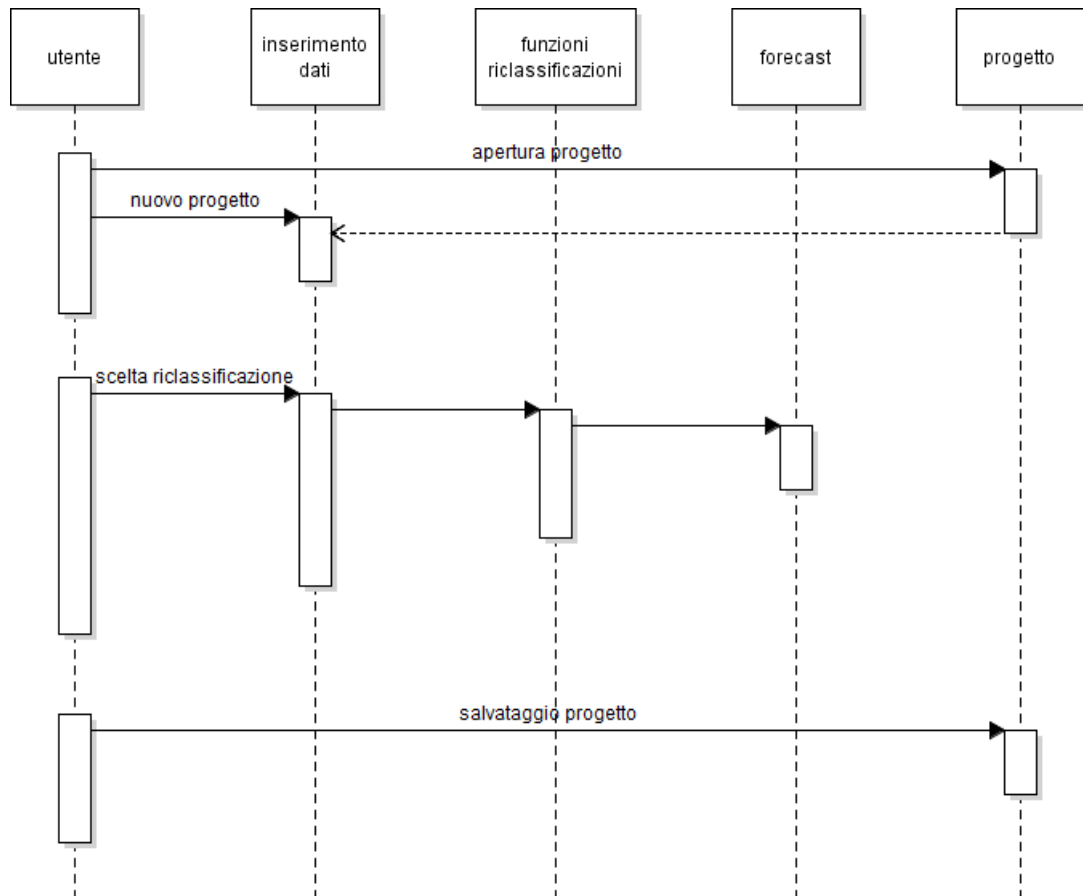


Figura 2.3: UML Sequence Diagram

Nella figura 2.3 vediamo descritto il flusso iniziale del software che inizia con la scelta da parte dell'utente di creazione di un nuovo progetto o di aprire un progetto salvato precedentemente e termina con il salvataggio dello stesso dopo aver calcolato tutte le riclassificazioni. I componenti del diagramma sono l'utente, la form che permette l'input dei dati, le funzioni per le riclassificazioni, la funzione del forecast e il progetto salvato esternamente. Nella fase di inserimento dei dati da parte dell'utente, il sistema calcola le varie riclassificazioni, tenendo conto della scelta dell'utente della tipologia scelta per lo Stato

Patrimoniale, e mette tutto a disposizione per il salvataggio dei dati e per la stampa del pdf.





# Capitolo 3

## Sviluppo

Quando si vuole sviluppare un'applicazione desktop la prima cosa a cui si pensa è per quale sistema operativo svilupparla, quale linguaggio usare ed opzionalmente a quale base dati si ha bisogno di connettersi. In questo caso la scelta è caduta su Electron, un framework Open Source sviluppato dal team di GitHub che permette lo sviluppo di app cross platform utilizzando il medesimo approccio di Apache Cordova, ovvero attraverso JavaScript, HTML, e CSS. Si tratta di un modulo Npm, quindi eredita tutte le API e le funzionalità di Node.js.

Electron, è ormai considerato un package stabile, arrivato alla versione 1.3.9, dispone di molti plugin già pronti all'uso e realizzati specificamente per la piattaforma. Tuttavia, il vero potenziale di Electron è dato dal fatto che si può costruire una desktop app utilizzando qualsiasi libreria scritta in Node.js, rendendolo di fatto uno strumento molto potente anche per interagire con il sistema operativo.

Nello specifico dello sviluppo del software, si possono distinguere tre specifiche aree:

- Back-End, comprende tutto ciò che riguarda lo sviluppo della logica del software, ovvero cosa fare con i dati, come trattare quelli ricevuti, cosa restituire all'utente.
- Front-End, comprende tutto ciò che riguarda l'interfaccia utente, quindi come presentare i dati passati dal Back-End, la grafica, il template, il typewriting, l'accessibilità e l'usabilità.

- Packaging dell'App, la creazione del prodotto finale, ovvero il software in versione eseguibile per tutte le tipologie di piattaforme di elaborazione.

### 3.1 Sviluppo del Back-End

Nella struttura di un app basata su Electron, bisogna discutere di due tipi di processo esistenti:

- il processo chiamato **main** che esegue lo script indicato nel file `package.json` è il processo principale. Lo script che viene eseguito nel processo principale può visualizzare una Graphical User Interface (GUI) tramite la creazione di pagine web. Una applicazione Electron ha sempre un processo principale, ma mai più di uno. Il processo principale crea pagine web mediante la creazione di istanze di `BrowserWindow`, dove viene eseguita la pagina web nel proprio processo di rendering. Quando viene eliminata un'istanza di `BrowserWindow`, il processo di rendering corrispondente viene anch'esso terminato. Il processo principale gestisce tutte le pagine web e il corrispondente processo di rendering.
- il processo di **rendering**, invece è il processo che gestisce ogni pagina web, visualizzata tramite Chromium con la sua architettura multi-processo, tale processo è isolato e può occuparsi solo delle pagine web in esecuzione in esso. Pertanto nelle pagine web chiamare le API dell'interfaccia grafica nativa non è consentito perché la gestione delle risorse di sistema nelle pagine web è molto pericolosa ed è facile perdere risorse. Se si desidera eseguire operazioni di GUI in una pagina web, il processo di rendering della pagina web deve comunicare con il processo principale per richiedere che il processo principale esegua tali operazioni.

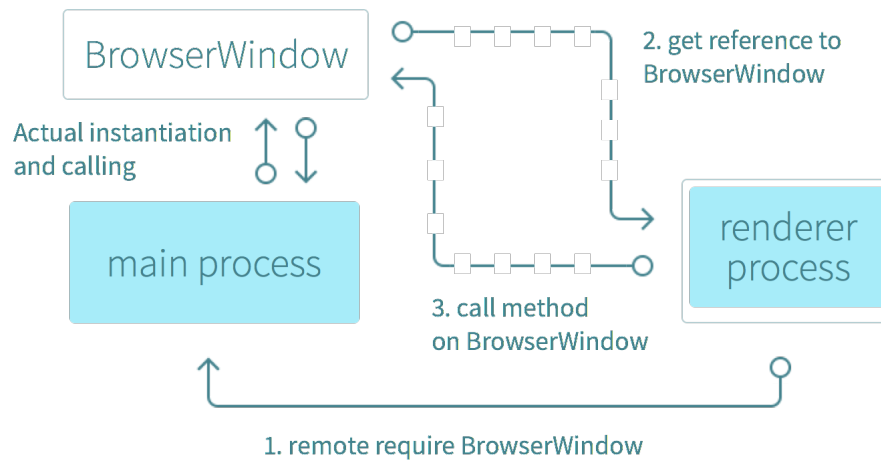


Figura 3.1: Electron - Architettura di un app

### 3.1.1 Utilizzo delle API Electron

Il .NET Framework (si pronuncia dot net) è un framework sviluppato da Microsoft eseguibile principalmente su Microsoft Windows. Include una grande libreria di classi, interfacce e tipi di dato chiamata e garantisce interoperabilità tra i linguaggi di programmazione (ogni linguaggio può usare codice scritto in un altro linguaggio). I programmi scritti in .NET sono eseguiti in un ambiente software (invece che in un ambiente hardware), conosciuto come , una virtual machine che fornisce servizi di sicurezza, gestione della memoria e gestione delle eccezioni. insieme compongono il .NET Framework.

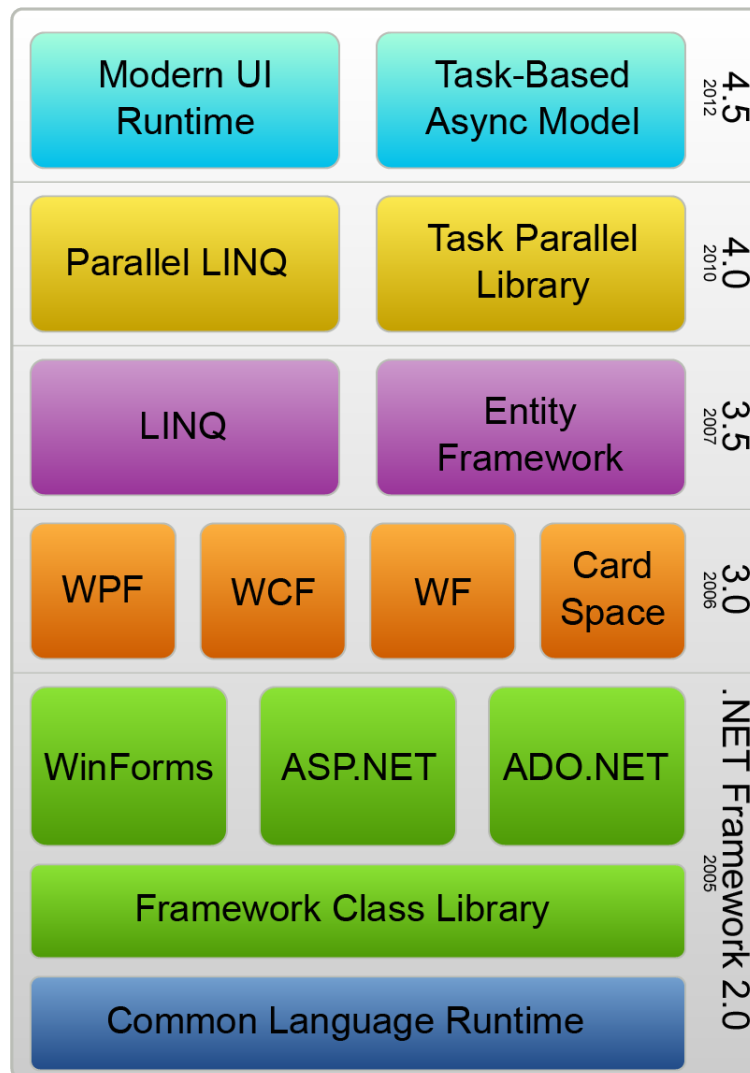


Figura 3.2: .NET Framework - Pila dei componenti

Oltre al .NET possiede un insieme di tecnologie per lo sviluppo su diverse piattaforme, alcune di queste tecnologie sono state utilizzate nel nostro progetto, tra le quali:

- ASP.NET
- Entity Framework
- Web API

**Creazione Pdf**

### 3.1.2 Utilizzo API Node.js

ASP.NET è un insieme di tecnologie software per lo sviluppo di web application e web services, si appoggia sul .NET Framework e offre tre frameworks per la creazione di web application:

- ASP.NET Web Pages, pagine dinamiche dove il markup HTML e il codice sono insieme nello stesso file
- Web Forms, che offre una libreria di controlli che incapsulano markup HTML, script javascript e logiche lato server per garantire una gestione ad eventi simile a quella delle applicazioni desktop
- ASP.NET MVC, alta separazione delle logiche di controllo, da quelle di presentazione e dai dati grazie al paradigm MVC, tipico anche di altri framework moderni

**ASP.NET MVC** fa uso dell'architettura multi-tier chiamata che prevede la separazione del codice in 3 livelli:

- Model (Modello), gestisce l'accesso ai dati dell'applicazione
- View (Vista), gestisce la visualizzazione dei dati all'utente
- Controller (Controllore), gestisce le interazioni dell'utente

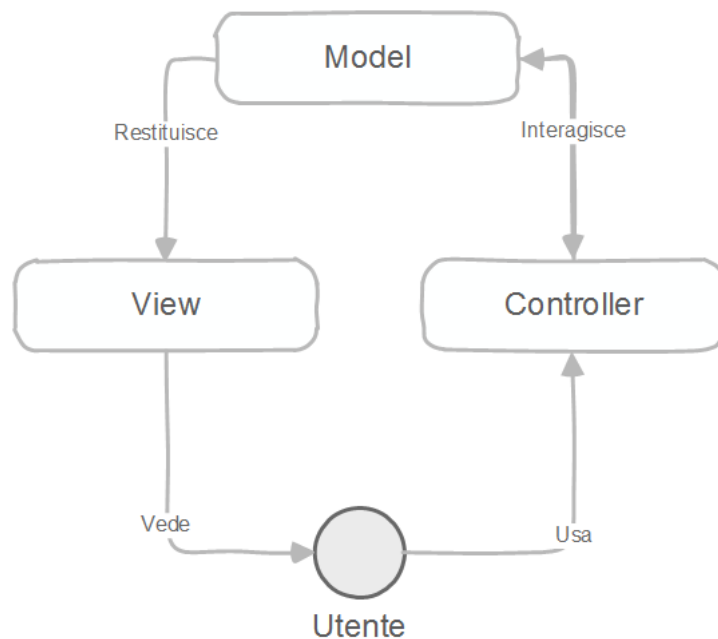


Figura 3.3: Architettura MVC

Il modo in cui questi 3 livelli interagiscono può cambiare da linguaggio a linguaggio, ma nel caso specifico di ASP.NET MVC funziona così:

1. l'utente invia una richiesta al browser (può essere una GET o una POST);
2. la richiesta viene intercettata dal web server ( nel nostro caso);
3. il web server fa gestire la richiesta al controller appropriato;
4. il controller la analizza e decide a quale specifico metodo farla gestire (nel nostro caso i metodi del controller si chiamano Action);
5. il controller può scegliere se lavorare o meno con il Model, in ogni caso alla fine del metodo ritornerà all'utente una View, cioè una pagina web di risposta.

Nel caso in cui il controller dovesse interfacciarsi con il database chiamerebbe il Model, il quale ha accesso diretto al database e colloquia con esso attraverso query . Di seguito in Figura 3.4 vediamo un esempio di Action per la visualizzazione dei Dettagli: tra i parametri del metodo c'è l'ID, passato al metodo

tramite URL, e come valore di ritorno c'è la View specifica per Dettagli, a cui viene passato l'oggetto vmFattura.

```
1  using Fatto.Filters;
2  using Fatto.Helpers;
3  using Fatto.Models;
4  using Fatto.ViewModels;
5  using log4net;
6  using PagedList;
7  using System;
8  using System.Collections.Generic;
9  using System.Linq;
10 using System.Transactions;
11 using System.Web.Mvc;
12
13 namespace Fatto.Controllers
14 {
15     [MyAuthorize(Roles = "Rup, Uo, Lettore, Amministratore")]
16     [MyProfileFilter]
17     public class FatturaController : Controller
18     {
19         private FatturaElettronicaEntities db = new FatturaElettronicaEntities();
20
21         // GET: /Fattura/Dettagli/5
22         public ActionResult Dettagli(int id = 0)
23         {
24             // Cerco la Fattura
25             var fattura = db.ViewDettagliFattura.Find(id);
26
27             if (fattura == null)
28             {
29                 // Non trovata -> Ritorno 404
30                 return HttpNotFound();
31             }
32             else
33             {
34                 // Trovata -> Continuo
35
36                 // Carico il model nel viewmodel
37                 VmFattura vmFattura = new VmFattura(fattura);
38
39                 return View(vmFattura);
40             }
41         }
42     }
```

Figura 3.4: Fattura Controller - Action

A volte può essere necessario aggiungere alcuni controlli ricorrenti all'inizio di alcune Actions o sul Controller stesso (in modo che valgano per tutte le Action del Controller), come ad esempio un metodo che controlli se l'utente è autenticato, in quel caso è possibile creare dei Filters o utilizzare quelli di base forniti dal framework (come Authorize). Nel nostro caso è stato necessario creare un nuovo Filter che ereditasse Authorize, ma che gestisse l'autenticazione Cohesion, richiamando quindi il in caso di mancata autenticazione, come in Figura 3.5.

```

1  using Fatto.Helpers;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Web;
6  using System.Web.Mvc;
7  using System.Web.Routing;
8
9  namespace Fatto.Filters
10 {
11     public class MyAuthorize : AuthorizeAttribute
12     {
13         protected override void HandleUnauthorizedRequest(AuthorizationContext filterContext)
14         {
15             var session = System.Web.HttpContext.Current.Session;
16
17             if (session["ReturnUrl"] == null)
18             {
19                 session["ReturnUrl"] = filterContext.HttpContext.Request.Url.AbsoluteUri;
20             }
21
22             // Ridireziono alla pagina di login con Cohesion
23             filterContext.Result =
24                 new RedirectToRouteResult(
25                     new RouteValueDictionary
26                     {
27                         { "action", "Login" },
28                         { "controller", "Account" }
29                     });
30         }
31     }
32 }

```

Figura 3.5: MyAuthorize Filter

## Salvataggio Json file



## 3.2 Sviluppo del Front-End

Per lo sviluppo di un interfaccia utente efficace si deve tener conto di due aspetti oltre a quello tecnico, che sono:

- Accessibilità
- Usabilità

L'**usabilità** è un approccio alla progettazione volto a rendere l'interazione tra il prodotto e l'utente migliore sotto i seguenti aspetti:

- Efficacia, cioè permettere agli utenti di raggiungere i loro obiettivi in maniera precisa e completa;
- Efficienza, cioè l'ottimizzazione delle risorse impiegate;
- Soddisfazione, come libertà dal disagio e attitudine positiva con cui gli utenti raggiungono specifici obiettivi attraverso l'uso del prodotto.

Nel web, l'usabilità si pone i seguenti obiettivi:

- Presentare l'informazione all'utente in modo chiaro e conciso, evitando termini tecnici o specialistici;
- Semplificare la struttura del compito;
- Offrire all'utente le scelte corrette, in una maniera che risulti ovvia;
- Organizzare ogni pagina in modo che l'utente riconosca la posizione e le azioni da compiere;
- Eliminare ogni ambiguità relativa alle conseguenze di un'azione (es. fare clic su cancella/rimuovi/compra);
- Mettere la cosa più importante nella posizione giusta della pagina web o dell'applicazione web;
- Fare in modo che l'utente abbia un rapido feedback ad ogni azione compiuta, ad esempio la comparsa di un messaggio di successo o errore all'invio di dati con una form;

- Rendere la grafica accattivante ed interessante dal punto di vista visivo attraverso l'uso di diagrammi, tabelle, sezioni informative e coordinando bene i colori;
- Ridurre gli sforzi cognitivi dell'utente.

In generale cercare di rendere il sistema il più semplice possibile da usare, in modo da ridurre al minimo gli sforzi sull'utilizzo del mezzo.

L'**accessibilità** è un aspetto di fondamentale importanza, specialmente per un gestionale web di una ; il suo scopo è garantire l'accesso alle risorse (prodotti e servizi) da parte di chiunque, siano essi soggetti con disabilità, con scarse competenze informatiche o con dispositivi diversi. I criteri dell'accessibilità sono quelli delle linee guida del che attraverso una sua sezione denominata ha definito i linguaggi e le procedure standard per rendere il Web uno strumento realmente democratico e universale. Queste direttive sono state percepite in Italia con la Legge "Stanca" (Legge 4/2004) che sancisce obblighi e sanzioni per la e le aziende appaltate. In generale i requisiti di accessibilità contenuti nella Legge Stanca e successive modificazioni, contenuti anche in un documento online sul sito del governo , sono i seguenti:

- Alternative testuali, obbligatorie per tutte i contenuti non testuali (come immagini, filmati e audio);
- Adattabilità, cioè prevedere che i contenuti si adattino a diversi layout in base alla grandezza e al formato dello schermo utente (responsive web design);
- Distinguibile, rendere più semplice agli utenti la visione e l'ascolto dei contenuti, separando i contenuti in primo piano dallo sfondo;
- Accessibile da tastiera, cioè garantire una buona navigabilità prevedendo un percorso di TAB;
- Colori, il contrasto tra le scritte e lo sfondo deve essere chiaro, ma non si devono usare colori discordanti tra loro o lampeggianti che possano causare crisi epilettiche;
- Navigabile, fornire una struttura del sito chiara, dove l'utente possa orientarsi e raggiungere qualsiasi sezione attraverso links;

- Assistenza per l’inserimento, fornire gli aiuti e le spiegazioni necessarie affinché l’utente sia in grado di compilare correttamente le form del sito;
- Compatibilità, il sito deve essere accessibile da tutte le piattaforme e deve utilizzare tecnologie standard.

### 3.2.1 Tecnologie usate nello sviluppo del front-end

Mentre per lo sviluppo del back-end abbiamo a disposizione una gran quantità di tecnologie concorrenti per svolgere lo stesso compito, per il front-end le tecnologie di base sono sempre le stesse: Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript.

**HTML** è un linguaggio di markup per le pagine web i cui standard sono definiti dal e serve per descrivere il contenuto di una pagina web, sia testuale che multimediale, attraverso dei tag. Sebbene il linguaggio sia in grado di definire anche regole di formattazione come la centratura del testo o la dimensione dei caratteri, il suo scopo primario è quello di descrivere i contenuti e il loro significato semantico, attraverso l’uso dei tag appropriati (ad esempio `<p>` per i paragrafi, `<h1>` per un titolo di primo livello, `<address>` per informazioni relative ad un indirizzo, etc.). Attualmente la versione più recente delle specifiche è chiamata HTML 5 e comporta rispetto alle precedenti versioni l’aggiunta di alcuni tag per gli elementi multimediali, il miglioramento i controlli di input per le form, il controllo della geolocalizzazione e l’introduzione dello Web Storage come alternativa ai cookies.

**CSS** è il linguaggio usato per definire il layout (impaginazione), la formattazione del testo e l’aspetto grafico in generale; può essere definito sia all’interno della pagina HTML che all’esterno come file a sé stante, che è generalmente la scelta migliore. Nel nostro caso sono state usate alcune delle proprietà più recenti del linguaggio come la box-shadow o la border-radius appartenenti alle specifiche CSS 3. Per mantenere la compatibilità tra i differenti browser (e nei limiti del possibile anche con le versioni più vecchie degli stessi) sono state applicate ad ogni proprietà CSS 3 delle tecniche di cross-browsing, cioè la riscrittura della stessa proprietà in modi differenti, vedi figure 3.6 e 3.7.

```

1  div.shadow
2  {
3      /* CSS 3 */
4      box-shadow: 3px 3px 4px #444444;
5
6      /* Mozilla Firefox */
7      -moz-box-shadow: 3px 3px 4px #444444;
8
9      /* Apple Safari, Google Chrome */
10     -webkit-box-shadow: 3px 3px 4px #444444;
11
12     /* Microsoft Internet Explorer */
13     -ms-filter:
14     "progid:DXImageTransform.Microsoft.Shadow(Strength=4, Direction=135, Color='#444444')";
15     filter:
16     progid:DXImageTransform.Microsoft.Shadow(Strength=4, Direction=135, Color='#444444');
17 }

```

Figura 3.6: CSS 3 text-shadow cross-browser

```

1  .round
2  {
3      /* Safari 3-4, iOS 1-3.2, Android 1.6- */
4      -webkit-border-radius: 11px;
5
6      /* Firefox 1-3.6 */
7      -moz-border-radius: 11px;
8
9      /* Opera 10.5, IE 9, Safari 5, Chrome, Firefox 4, iOS 4, Android 2.1+ */
10     border-radius: 11px;
11 }

```

Figura 3.7: CSS 3 border-radius cross-browser

**JavaScript** è un linguaggio di scripting adatto per creare effetti dinamici interattivi basati su eventi, come il click di un bottone, la selezione di una voce da una tendina, la pressione di un tasto o il caricamento di una pagina. Anche se con il javascript è possibile creare applicazioni vere e proprie si deve sempre tener conto che è un linguaggio accessorio, ovvero c'è la possibilità per l'utente di disabilitarlo, specialmente nel caso si abbia delle disabilità o dei problemi di visualizzazione. Pertanto nessuna funzionalità deve contare unicamente su javascript per funzionare, ma il suo uso è comunque consigliato per aumentare l'esperienza utente (quegli obiettivi dell'usabilità esposti sopra) ad esempio prevedendo dei messaggi di errore nel caso si inseriscano dei valori errati nei campi, o un calendario all'immissione di un campo data, o anche per sostituire delle proprietà del CSS 3 nel caso il browser dell'utente non la supporti (i cosiddetti polyfill).

**jQuery** è una libreria Javascript molto diffusa il cui scopo è semplificare la selezione, la gestione e la manipolazione degli eventi, mantenendo però anche la possibilità di utilizzare il linguaggio nella vecchia maniera. L'uso di jQuery non aggiunge quindi nessuna funzionalità, rende solamente più semplice l'utilizzo

del linguaggio e quindi più elevata la produttività del programmatore, non a caso il suo motto è "Write less, do more", anch'essa è stata inclusa nella nostra applicazione web e gli script personalizzati fanno tutti riferimento alla sua sintassi.

è un'altra tecnologia molto importante per migliorare l'esperienza utente, perché permette di aggiornare il contenuto di una pagina dinamicamente senza effettuare il "refresh" e quindi caricando solo le informazioni necessarie, anziché tutta la pagina. Si tratta, più che di un nuovo linguaggio, di un insieme di tecniche che fanno uso delle tecnologie sopra descritte, quindi JavaScript per inviare le richieste asincrone e XML o JSON per ricevere le risposte dal server sottoforma di oggetti. Questa tecnica è stata utilizzata nella nostra applicazione web nella fase di autenticazione post-Cohesion, dove viene richiesto all'utente di scegliere un ruolo da una tendina e automaticamente sulla base di questa scelta viene popolata la tendina delle riferite a quel ruolo. Un altro utilizzo è nella pagina di inserimento di un nuovo utente, dove a seguito della digitazione di almeno 3 caratteri in una casella di testo, compare un menù di autocomplete che suggerisce i possibili nomi degli utenti cercati.

**Razor** infine, è un linguaggio di programmazione lato server, specifico del framework ASP.NET MVC, ma è stato incluso in questa sezione perché serve a descrivere, insieme all'HTML, il contenuto di una View. Una View che contiene codice Razor sarà un file di tipo .cshtml (nel caso il linguaggio lato server utilizzato sia il C#). Questo linguaggio può utilizzare molte delle classi del framework, semplicemente includendo il namespace nella pagina, ma il suo scopo principale è quello di posizionare gli elementi del Model nella View, passati dal Controller, eventualmente anche iterando una Collection affinché venga stampata a schermo una serie di elementi per ogni elemento della Collection, come in Figura 3.8.

```

64
65 <tbody>
66     @foreach (var item in Model)
67     {
68         <tr>
69             <td data-title="Prestatore">
70                 <div>@Html.DisplayFor(modelItem => item.PrestatoreIva)</div>
71                 <div>@Html.DisplayFor(modelItem => item.PrestatoreNome)</div>
72             </td>
73             <td data-title="Dati Generali - Data">@Html.DisplayFor(modelItem => item.DatiGeneraliDocumentoData.ToString("dd/MM/yyyy"))</td>
74             <td data-title="Dati Generali - Numero">@Html.DisplayFor(modelItem => item.DatiGeneraliDocumentoNumero)</td>
75             <td data-title="Causale" class="description">@Html.DisplayFor(modelItem => item.Causale)</td>
76             <td data-title="Stato Attuale">@Html.DisplayFor(modelItem => item.StatoAttuale)</td>
77             <td data-title="RP" class="description">@Html.DisplayFor(modelItem => item.Rup)</td>
78             <td data-title="Scadenze" class="scadenze">
79                 @if (!item.Rifiutata == true || item.Pagata == true || item.Sospesa == true)
80                 {
81                     if (item.Accettata == true)
82                     {
83                         TimeSpan t = TimeSpan.FromSeconds((int)item.ScadenzaPagamento);
84                         <div>
85                             <span>@t.Days</span> giorni per il Pagamento
86                         </div>
87                     }
88                     else
89                     {
90                         TimeSpan t = TimeSpan.FromSeconds((int)item.ScadenzaRifiuto);
91                         <div>
92                             <span>@t.Days</span> giorni per il Rifiuto
93                         </div>
94                     }
95                 }
96             </td>
97             <td>
98                 @Html.ActionLink("Documento", "Documento", new { id = item.Id }) |
99                 @Html.ActionLink("Dettagli", "Dettagli", new { id = item.Id })
100             </td>
101         </tr>
102     }
103 </tbody>

```

Figura 3.8: Esempio di codice Razor per la stampa degli elementi di una collection come righe di una tabella HTML

### 3.3 Packaging dell'applicazione

Per lo sviluppo di un interfaccia utente efficace si deve tener conto di due aspetti oltre a quello tecnico, che sono:

- Accessibilità
- Usabilità

L'**usabilità** è un approccio alla progettazione volto a rendere l'interazione tra il prodotto e l'utente migliore sotto i seguenti aspetti:

- Efficacia, cioè permettere agli utenti di raggiungere i loro obiettivi in maniera precisa e completa;
- Efficienza, cioè l'ottimizzazione delle risorse impiegate;
- Soddisfazione, come libertà dal disagio e attitudine positiva con cui gli utenti raggiungono specifici obiettivi attraverso l'uso del prodotto.

Nel web, l'usabilità si pone i seguenti obiettivi:

- Presentare l'informazione all'utente in modo chiaro e conciso, evitando termini tecnici o specialistici;
- Semplificare la struttura del compito;
- Offrire all'utente le scelte corrette, in una maniera che risulti ovvia;
- Organizzare ogni pagina in modo che l'utente riconosca la posizione e le azioni da compiere;
- Eliminare ogni ambiguità relativa alle conseguenze di un'azione (es. fare clic su cancella/rimuovi/compra);
- Mettere la cosa più importante nella posizione giusta della pagina web o dell'applicazione web;
- Fare in modo che l'utente abbia un rapido feedback ad ogni azione compiuta, ad esempio la comparsa di un messaggio di successo o errore all'invio di dati con una form;

- Rendere la grafica accattivante ed interessante dal punto di vista visivo attraverso l'uso di diagrammi, tabelle, sezioni informative e coordinando bene i colori;
- Ridurre gli sforzi cognitivi dell'utente.

In generale cercare di rendere il sistema il più semplice possibile da usare, in modo da ridurre al minimo gli sforzi sull'utilizzo del mezzo.

L'**accessibilità** è un aspetto di fondamentale importanza, specialmente per un gestionale web di una ; il suo scopo è garantire l'accesso alle risorse (prodotti e servizi) da parte di chiunque, siano essi soggetti con disabilità, con scarse competenze informatiche o con dispositivi diversi. I criteri dell'accessibilità sono quelli delle linee guida che attraverso una sua sezione denominata ha definito i linguaggi e le procedure standard per rendere il Web uno strumento realmente democratico e universale. Queste direttive sono state percepite in Italia con la Legge "Stanca" (Legge 4/2004) che sancisce obblighi e sanzioni per le aziende appaltate. In generale i requisiti di accessibilità contenuti nella Legge Stanca e successive modificazioni, contenuti anche in un documento online sul sito del governo sono i seguenti:

- Alternative testuali, obbligatorie per tutti i contenuti non testuali (come immagini, filmati e audio);
- Adattabilità, cioè prevedere che i contenuti si adattino a diversi layout in base alla grandezza e al formato dello schermo utente (responsive web design);
- Distinguibile, rendere più semplice agli utenti la visione e l'ascolto dei contenuti, separando i contenuti in primo piano dallo sfondo;
- Accessibile da tastiera, cioè garantire una buona navigabilità prevedendo un percorso di TAB;
- Colori, il contrasto tra le scritte e lo sfondo deve essere chiaro, ma non si devono usare colori discordanti tra loro o lampeggianti che possano causare crisi epilettiche;
- Navigabile, fornire una struttura del sito chiara, dove l'utente possa orientarsi e raggiungere qualsiasi sezione attraverso links;



- Assistenza per l'inserimento, fornire gli aiuti e le spiegazioni necessarie affinché l'utente sia in grado di compilare correttamente le form del sito;
- Compatibilità, il sito deve essere accessibile da tutte le piattaforme e deve utilizzare tecnologie standard.

## Capitolo 4

### Possibili sviluppi futuri

Ogni regione è stata chiamata per legge a sviluppare il proprio sistema di fatturazione e, eventualmente, a porsi come nodo locale di interscambio per le fatture degli enti locali. In Regione Lazio il sistema **HUB** svolge appunto questo ruolo, esso prevede un sito per l'adesione al sistema da parte delle e dei privati a tre tipi di servizio: fattura attiva, fattura passiva, conservazione. La Regione Emilia-Romagna predispone anch'essa un sistema di interscambio regionale chiamato **NoTI-ER**, il quale una volta ricevuta la fattura dalloprima la converte nel formato europeo PEPPOL, poi la invia al software **SICIPA-ER** (l'equivalente del nostro Fatto) e al sistema di conservazione Par-ER. Troviamo un sistema di interscambio anche per la Regione Toscana, chiamato **fERT**, che oltre alla comunicazione tr e Ente, si occupa anche della comunicazione con la In Regione Marche il ruolo di intermediario svolto da è quello di mettere in comunicazione lo con Fatto e Fatto con il sistema di protocollo, quindi presiede a tutto il processo di fatturazione fino al momento dell'accettazione/rifiuto. Successivamente le comunicazioni con la passano per il software della contabilità Siagi, il quale prima di inviare i dati relativi ai pagamenti, preleva le informazioni necessarie (associazioni tra fatture e decreti, tempi di accettazione e contabilizzazione, etc) da Fatto.



# Conclusione

La mia collaborazione alla realizzazione di questo progetto mi ha permesso di valutare in un contesto concreto l'efficacia sia delle metodologie di analisi e progettazione, sia delle tecnologie di sviluppo utilizzate. Essendo stato partecipe inoltre ad alcuni processi decisionali nel ruolo di referente tecnico, ho potuto confermare l'importanza di una buona scrittura dei requisiti funzionali e non, accompagnata da una gestione delle attività e delle scadenze.

Dal punto di vista del personale amministrativo non ci sono state difficoltà nell'apprendere l'utilizzo del software, anche grazie agli accorgimenti presi per il front-end e alla continua disponibilità, mia e dei miei referenti. In ogni caso alcune problematiche di dominio sono venute fuori solo successivamente al rilascio della prima versione del software, ciò ha comportato un continuo lavoro di rianalisi e riadattamento che ha interessato tutte le parti responsabili dei software coinvolti nel processo di fatturazione.

Parlando di vantaggi al cittadino, da un lato si sono ottenuti dei tempi certi per quanto riguarda l'accettazione e il rifiuto (garantiti dalla spietatezza dei contatori informatici) e si è indirettamente dato uno standard ai vari software proprietari e non per la gestione delle fatture. Da un altro lato però sono aumentati gli oneri per i fornitori, perché anche ammettendo che si riesca a realizzare autonomamente l'xml della fattura da inviare o che il vecchio software di fatturazione in proprio possesso sia stato aggiornato senza ulteriori spese, si deve far fronte alle spese per la conservazione digitale, ovvero alle spese per un software di conservazione aderente agli standard e alle marche temporali digitali.

Tutti questi problemi spero si risolvano nel futuro, scegliendo magari di gestire la conservazione direttamente lato, con un maggiore sforzo implementativo da parte del Ministero.

Complessivamente mi ritengo soddisfatto di quanto fatto finora, ma sono consapevole che la realizzazione del progetto non è che un passo verso una complessiva digitalizzazione e sostanziale modifica del funzionamento della.

I futuri sviluppi che interesseranno il sistema di fatturazione saranno legati al prossimo rilascio da parte della dei web service per lo scambio delle informazioni relative ai pagamenti effettuati. Non è esclusa tuttavia anche una possibile adozione degli standard PEPPOL, relativi al processo di fatturazione o anche una parziale ridefinizione di alcuni processi amministrativi digitali.

# Dediche

# Bibliografia

## Libri

- [1] S. Gaburri (Traduttore) Martin Fowler (Autore) L. Baresi (a cura di).  
*UML distilled - Guida al linguaggio di modellazione standard, Quarta Edizione*. Pearson, 1981. ISBN: 978-8871925981.