Progetto 1

Matan Davidi

Versione: 09.11.2018

1 Introduzione

1.1 Informazioni sul progetto

Autore: Matan Davidi
Scuola: Arti e Mestieri Trevano
Classe: I3AA
Anno scolastico: 2018/19
Sezione: Informatica
Materia: Modulo 306
Docenti responsabili: Adriano Barchi, Luca Muggiasca, Francesco Mussi, Elisa Nannini
Data di inizio: 05.09.2018
Data di consegna: 09.11.2018

1.2 Abstract

We see it very frequently: a user needs to sign themselves up to a service and their data has to be saved somewhere so register them successfully. This web application will greet a user, ask them for their personal information, validate it and save it to two CSV files. The information stored in these files can later be used to offer personalized services to each user.

1.3 Scopo

Lo scopo di questo progetto è quello di imparare a progettare e documentare un progetto prima di cominciare con la sua implementazione.

Analisi

1.4 Analisi del dominio

Attualmente esistono moltissime applicazioni che gestiscono la registrazione degli utenti, la maggior parte dei quali su un database con, ad esempio, SQL. In questo caso il lavoro è diverso in quanto gli utenti sono da registrare su due file CSV. Il prodotto deve funzionare sul web e deve essere accessibile ad un gruppo di utenti, magari come requisito per accedere ad un servizio specifico, come un forum oppure una condivisione di file tra dipendenti di un'azienda. In questo caso gli utenti saranno delle aziende che vogliono restringere, controllare e/o offrire esperienze personalizzate alle persone che usufruiscono di uno o più dei loro prodotti, per esempio servizi come Gmail o Reddit. L'utente che utilizza questo prodotto non deve disporre di preconoscenze specifiche, se non una familiarità di base con i computer e Internet.
In generale i campi obbligatori hanno una lunghezza minima, mentre alcuni campi possiedono delle convenzioni. Ad esempio, l'email deve avere una sola '@' e al minimo un punto e il numero di telefono deve avere un certo numero di caratteri.

1.5 Analisi e specifica dei requisiti

Vedi DomandeProgetto1.docx

ID: REQ-001	
Nome	Il prodotto deve scrivere su due file CSV distinti: uno giornaliero ed uno complessivo
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	I file si devono chiamare “Registrazione_tutte.csv” e “Registrazione_aaaa_mm_gg.csv”.
002	Se i file non esistono, devono essere creati
003	I file verranno salvati in una cartella “Registrazioni” nella cartella principale del programma.
004	Il formato di questi file deve essere “data, nome, cognome, data_nascita, via, numero_civico, citta, nap, telefono, email, sesso, hobby, professione”.

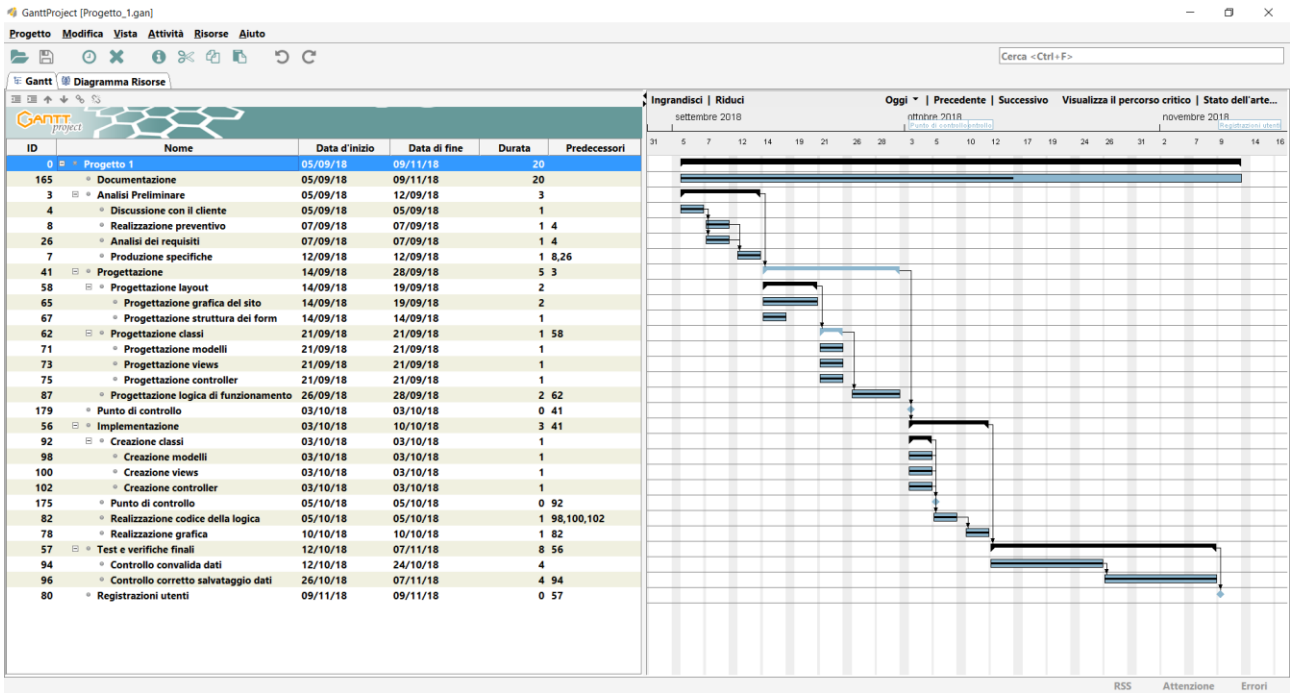
ID: REQ-002	
Nome	Pagina di benvenuto
Priorità	3
Versione	1.0
Note	
Sotto requisiti	
001	La pagina deve contenere un messaggio di benvenuto e un bottone “Registrati” che porta alla pagina di registrazione.

ID: REQ-003	
Nome	Pagina di registrazione dei dati
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	I dati che l’utente deve inserire sono: **nome**, **cognome**, **data di nascita**, **via**, **numero civico**, **città**, **NAP**, **numero di telefono**, **email**, **sesso**, hobby e professione. I dati in grassetto sono obbligatori.
002	Nella pagina della registrazione dei dati devono esserci due pulsanti: “Cancella” che azzera i valori dei campi e “Avanti” che ridireziona alla prossima pagina.
003	I campi obbligatori devono essere contrassegnati con un asterisco (*)
004	Per inserire i dati, i campi devono essere input dei seguenti tipi e con i seguenti criteri di validazione: **Nome:** testo **Cognome:** testo **Data di nascita:** data **Via:** testo **Numero civico:** testo, 4 cifre, solo lettere e numeri **Città:** testo **NAP:** numerico, 5 cifre **Numero di telefono:** testo, solo cifre, spazi e/o trattini **E-mail:** testo, controllo che sia un’email valida **Sesso:** testo, 1 carattere, “F” o “M” Hobby: testo Professione: testo
005	Tutti i campi devono essere validati prima di poter proseguire alla prossima pagina e i campi obbligatori devono essere riempiti.

ID: REQ-004	
Nome	Pagina di verifica della correttezza dei dati
Priorità	2
Versione	1.0
Note	La pagina serve all'utente per controllare che i dati che ha inserito siano corretti.
Sotto requisiti	
001	Nella pagina di verifica deve comparire il pulsante "Correggi" che riporta alla pagina precedente
002	Nella pagina di verifica deve comparire il pulsante "Registra" che salva i dati su file.

ID: REQ-005	
Nome	Pagina di verifica della corretta scrittura dei dati
Priorità	3
Versione	1.0
Note	La pagina serve all'utente a capire che i suoi dati sono stati scritti correttamente sui file CSV.
Sotto requisiti	
001	Nella pagina di verifica deve comparire un pulsante "Home" che riporti alla pagina di benvenuto.

1.6 Pianificazione



ID	Nome	Data d'inizio	Data di fine	Durata	Predecessori
0	Progetto 1	05/09/18	09/11/18	20	
165	Documentazione	05/09/18	09/11/18	20	
3	Analisi Preliminare	05/09/18	12/09/18	3	
4	Discussione con il cliente	05/09/18	05/09/18	1	
8	Realizzazione preventivo	07/09/18	07/09/18	1	4
26	Analisi dei requisiti	07/09/18	07/09/18	1	4
7	Produzione specifiche	12/09/18	12/09/18	1	8,26
41	Progettazione	14/09/18	28/09/18	5	3
58	Progettazione layout	14/09/18	19/09/18	2	
65	Progettazione grafica del sito	14/09/18	19/09/18	2	
67	Progettazione struttura dei form	14/09/18	14/09/18	1	
62	Progettazione classi	21/09/18	21/09/18	1	58
71	Progettazione modelli	21/09/18	21/09/18	1	
73	Progettazione views	21/09/18	21/09/18	1	
75	Progettazione controller	21/09/18	21/09/18	1	
87	Progettazione logica di funzionamento	26/09/18	28/09/18	2	62
179	Punto di controllo	03/10/18	03/10/18	0	41
56	Implementazione	03/10/18	10/10/18	3	41
92	Creazione classi	03/10/18	03/10/18	1	
98	Creazione modelli	03/10/18	03/10/18	1	
100	Creazione views	03/10/18	03/10/18	1	
102	Creazione controller	03/10/18	03/10/18	1	
175	Punto di controllo	05/10/18	05/10/18	0	92
82	Realizzazione codice della logica	05/10/18	05/10/18	1	98,100,102
78	Realizzazione grafica	10/10/18	10/10/18	1	82
57	Test e verifiche finali	12/10/18	07/11/18	8	56
94	Controllo convalida dati	12/10/18	24/10/18	4	
96	Controllo corretto salvataggio dati	26/10/18	07/11/18	4	94
80	Registrazioni utenti	09/11/18	09/11/18	0	57

Figura 1 - pianificazione rappresentata attraverso un diagramma di Gantt

1.7 Analisi dei mezzi

1.7.1 Software

Il progetto è stato sviluppato su un sistema operativo Windows 10 Home a 64 bit e utilizzando il seguente software:

- Microsoft Visual Studio Enterprise 2017 15.5.7
- Microsoft Visual Studio Code 1.27.2
- GanttProject 2.8.9
- Microsoft Word 16.0.10730.20102
- Mozilla Firefox 62.0
- Google Chrome 69.0.3497.100
- GitHub Desktop 1.4.0
- Microsoft Visio 2010 14.0.4756.1000

Le librerie utilizzate comprendono:

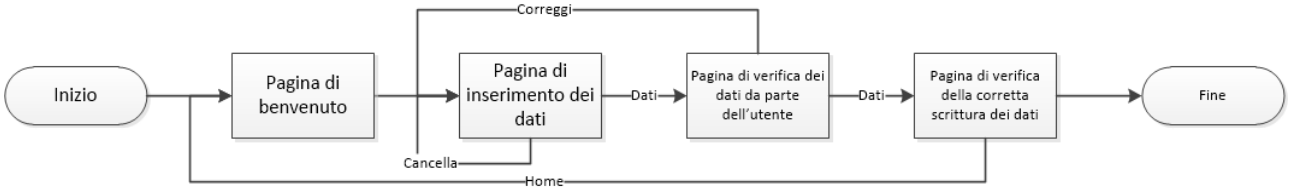
- Bootstrap
- jQuery
- jQuery Validation

1.7.2 Hardware

- HP Pavilion - 15-au147nz: il computer portatile sul quale è stato realizzato il progetto. Le sue specifiche comprendono:
  - Intel® Core™ i7-7500U
  - 16 GB RAM
  - NVIDIA® GeForce® 940MX
  - Intel® HD Graphics 620

2 Progettazione

2.1 Design dell'architettura del sistema



2.2 Design delle interfacce

Figura 2 - Layout della pagina di benvenuto

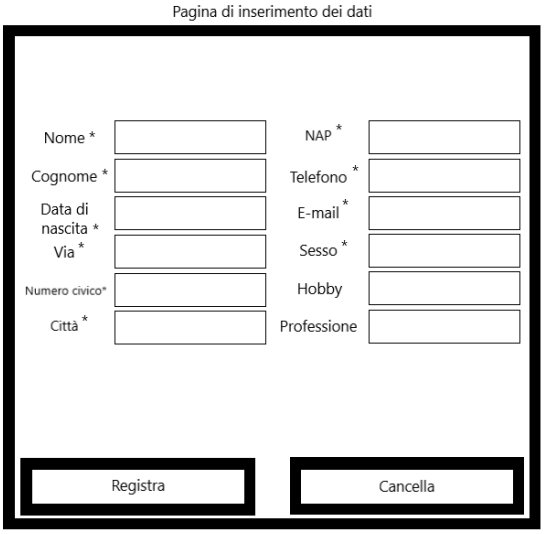


Figura 3 - Layout della pagina di inserimento dei dati

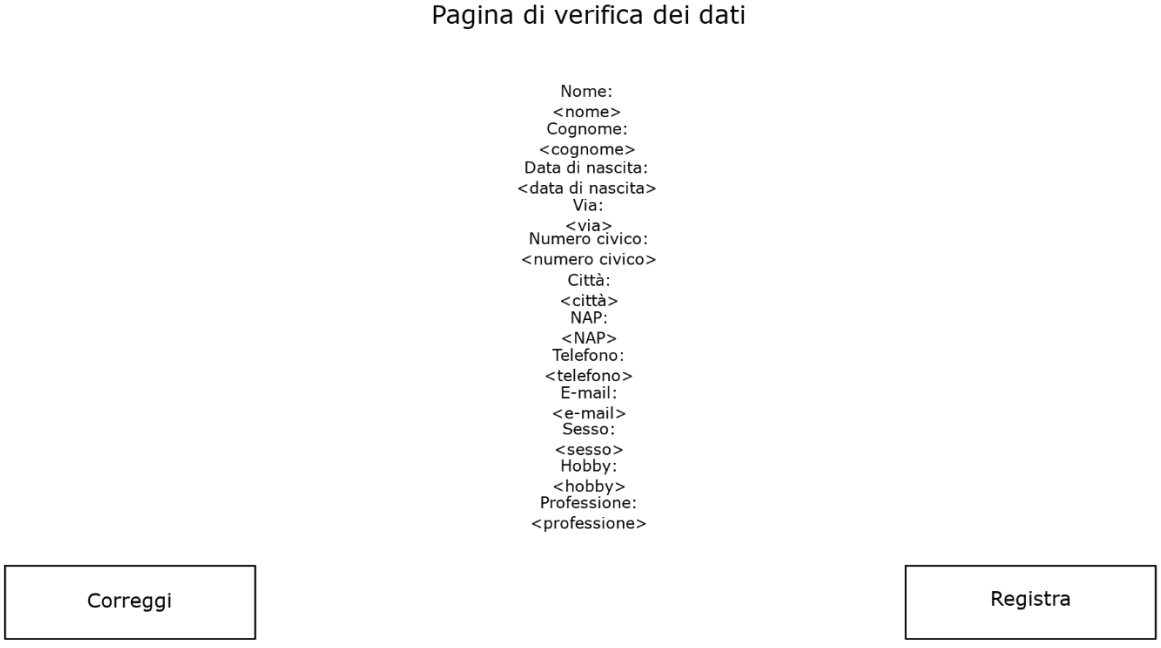


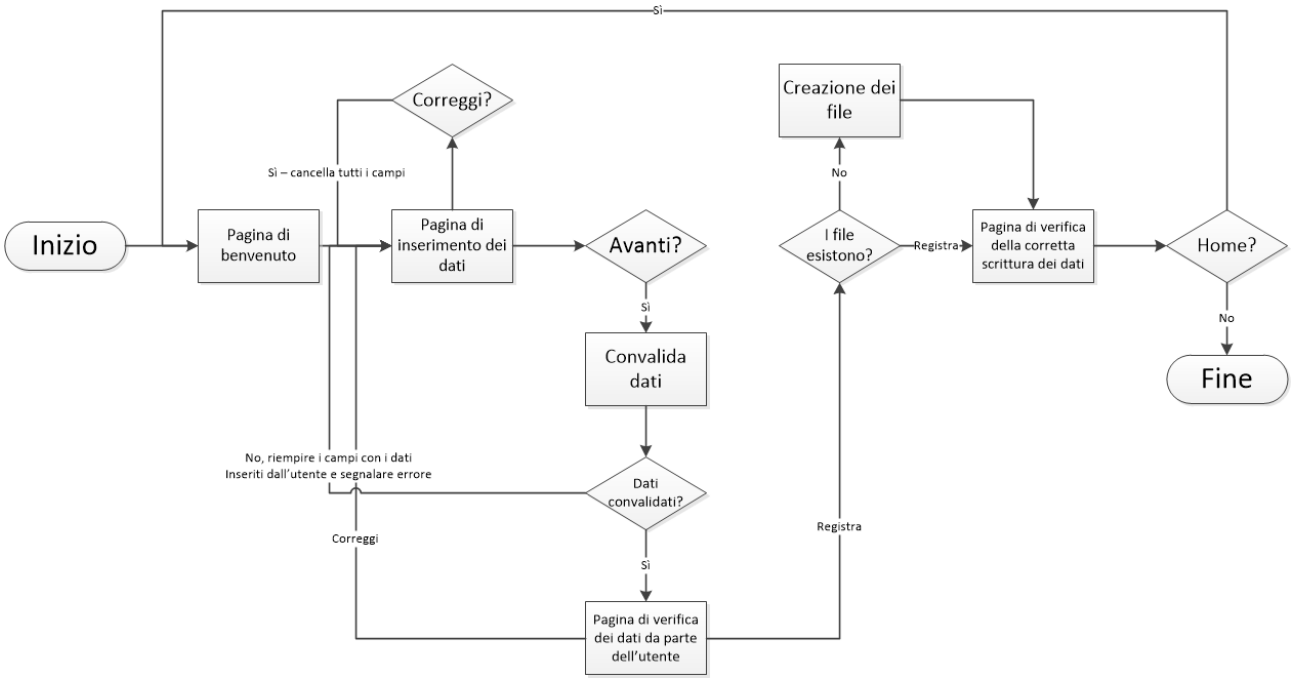
Figura 4 - Layout della pagina di verifica per l'utente

Testo di ringraziamento

Data	Nome	Cognome	Data di nascita	Via	Numero civico	Città	NAP	Telefono	E-mail	Sesso	Hobby	Professione

Figura 5 - Layout della pagina di verifica di scrittura

2.3 Design procedurale



3 Implementazione

Questa applicazione è divisa in quattro pagine: la pagina principale, la pagina di inserimento dei dati, la pagina di verifica di corretto inserimento dei dati e la pagina contenente i dati appena inseriti in uno dei file CSV.

3.1 Classe User

Tutta l'applicazione ruota attorno ad una classe chiamata User che contiene tante proprietà quanti i dati dell'utente da salvare, quindi:

- Nome
- Cognome
- Data di nascita
- Indirizzo
- Numero civico
- Città
- NAP
- Numero di telefono
- Indirizzo e-mail
- Hobby
- Professione

Questa classe è il modello dell'applicazione e viene creata una sua istanza ogni volta che viene registrato un utente. Essa contiene inoltre due costruttori, uno vuoto che non fa niente ed uno che crea una nuova istanza di User utilizzando un array di stringhe con lo stesso ordine dei campi del file CSV, che utilizzo una volta che leggo i dati dal file delle registrazioni giornaliero.

3.2 Pagina principale (Home)

La pagina principale contiene al suo interno anche le pagine di About e di Contatti, oltre alla pagina effettiva di benvenuto. Questa pagina utilizza la classe HomeController, che è stata generata automaticamente da Visual Studio alla creazione del progetto e che non ho modificato.

3.2.1 Pagina di benvenuto

Questa pagina si chiama 'Index.cshtml' e contiene il nome del progetto, l'autore dell'applicazione ed un piccolo messaggio di benvenuto, oltre ad un bottone che permette di cominciare il processo di registrazione portando l'utente alla pagina di inserimento dati. Questo bottone è stato realizzato con il seguente codice:

```
<input class="btn btn-block btn-default" type="submit" name="signup" value="Registrati"
onclick="window.location.href='@Url.Action("", "Signup")';" />
```

3.2.2 Pagina About

Questa pagina si chiama 'About.cshtml' e contiene solo il titolo della pagina, una breve descrizione dell'applicazione, qualche informazione personale e la licenza con la quale viene distribuita, ossia GNU 3.

3.2.3 Pagina di contatti

Questa pagina si chiama 'Contact.cshtml' e contiene il titolo della pagina ed un singolo tag `<address>` che, a sua volta, contiene il mio indirizzo email ed il mio sito web.

3.3 Pagina inserimento dati (Signup)

La pagina di inserimento dei dati contiene al suo interno la view vera e propria in cui inserire i dati, ma anche la view di verifica di corretto inserimento dei dati, in modo da poter passare i dati tra una e l'altra più comodamente. Questa pagina utilizza la classe SignupController.

3.3.1 Pagina di inserimento dei dati

Questa pagina si chiama 'Index.cshtml' e contiene un titolo, le istruzioni su come proseguire, 12 input da riempire con i valori che si vuole registrare e due bottoni: "Azzera" e "Registra". Per la realizzazione di questa pagina ho creato una nuova view con un template di tipo Create e ho modificato quello. Concretamente, all'interno del file si trova questo codice:

```
<div class="row">
    <form asp-action="DataCheck" class="saveForm">
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <div class="col-md-12">
            <div class="col-md-6 form-group">
                <label asp-for="FirstName" class="control-label">Nome<strong class="text-danger">*</strong></label>
                <input asp-for="FirstName" class="form-control field" placeholder="John" />
                <span asp-validation-for="FirstName" class="text-danger"></span>
            </div>
```

Ovviamente c'è un `<div class="col-md-6 form-group">` per ogni proprietà della classe User.

In fondo alla pagina ci sono i due bottoni:

```
<div class="col-md-12">
    <div class="col-md-6 form-group ">
        <input type="button" value="Azzera" class="btn btn-default form-control clear" />
    </div>
    <div class="col-md-6 form-group ">
        <input type="submit" value="Registra" class="btn btn-default form-control save" />
    </div>
</div>
```

Il bottone con `value="Azzera"` è quello che cancella i campi. La cancellazione è gestita con jQuery richiamato quando viene cliccato un bottone con la classe '`clear`'. Infatti, all'interno del file site.js troviamo:

```
$(".clear").click(function () {

    $(".field").val("");
    $(".radio-inline").first().prop("checked", true);
    $(".radio-inline").last().removeAttr("checked");

    bDayInput.addClass("valid");
    bDayInput.removeClass("input-validation-error");
    bDayInput.prop("aria-invalid", false);
    bDayInput.siblings("span").removeClass("field-validation-error");
    bDayInput.siblings("span").addClass("field-validation-valid");
    bDayInput.siblings("span").children("span").text("");

});
```

Questa pagina utilizza due azioni Index:

```
0 references | Matan Davidi, 13 days ago | 2 authors, 5 changes | 0 requests | 0 exceptions
public IActionResult Index()
{
    return View();
}

[HttpPost]
0 references | Matan Davidi, 13 days ago | 1 author, 3 changes | 0 requests | 0 exceptions
public IActionResult Index(IFormCollection collection)
{
    User u = CollectionToUser(collection);
    return View("Index", u);
}
```

La prima delle quali viene richiamata quando viene premuto il pulsante nella pagina precedente, quindi la prima volta che si entra nella pagina; la seconda viene richiamata quando si giunge alla pagina dopo aver premuto il pulsante "Correggi" della pagina successiva. La differenza tra le due è che la seconda riceve un'istanza di `IFormCollection` come parametro, che contiene i valori passati come modello dalla pagina DataCheck. Questi valori vengono usati per ricostruire il modello per la pagina Index.
Per farlo viene utilizzato un metodo `private User CollectionToUser(IFormCollection collection)` che utilizza il metodo `collection.TryGetValue()` per ottenere i valori, con i quali ricostruisce un'istanza di User.

Per la validazione utilizzo delle regole che appartengono alla libreria `System.ComponentModel.DataAnnotations`, che assomigliano a queste:

```
#region ================== costanti ==================
public const string TEXTREGEX = "[a-zA-ZàáâãäåąčćęèéêëėįìíîïłńòóôõöøùúûüųūÿýżźñçčšžÀÁÂÃÄÅĄĆČĖĘÈÉÊËÌÍÎÏĮŁŃÒÓÔÕÖØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽð
#endregion

#region ================== membri & proprietà =========
[Required(ErrorMessage = "Inserire un nome")]
[StringLength(50, ErrorMessage = "La lunghezza del nome deve essere al massimo di 50 caratteri")]
[Display(Name = "Nome")]
[RegularExpression(TEXTREGEX, ErrorMessage = "Il nome contiene caratteri non consentiti")]
7 references | Matan Davidi, 13 days ago | 1 author, 4 changes | 0 exceptions
public string FirstName { get; set; }
```

3.3.2 Pagina di verifica di inserimento

Questa pagina si chiama 'DataCheck.cshtml' e contiene le istruzioni su come proseguire o tornare indietro, i dati inseriti e la loro intestazione e i bottoni per ridirezionare alle altre pagine. Tutto è contenuto in un contenitore di classe '`container`'. I dati sono contenuti in una lista descritta, ossia un tag `<dl>` fatta in questo modo:

```
<dl class="dl-horizontal">
    <dt>
        @Html.DisplayNameFor(model => model.FirstName)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.FirstName)
    </dd>
    <dt>
        @Html.DisplayNameFor(model => model.LastName)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.LastName)
    </dd>
```

Anche in questo caso c'è una coppia `dd-dt` per ogni proprietà della classe User.
In fondo alla pagina ci sono due div di classe `col-md-6` che contengono i bottoni all'interno di uno di due form che hanno come `asp-action` rispettivamente '`Index`' e '`Save`', assieme a tanti input invisibili quante le proprietà della classe User.

```
<div class="col-md-6">
    <form asp-action="Index" method="post">
        @Html.HiddenFor(model => model.FirstName)
        @Html.HiddenFor(model => model.LastName)
        @Html.HiddenFor(model => model.Birthday)
        @Html.HiddenFor(model => model.Address)
        @Html.HiddenFor(model => model.CivicNumber)
        @Html.HiddenFor(model => model.City)
        @Html.HiddenFor(model => model.Nap)
        @Html.HiddenFor(model => model.PhoneNumber)
        @Html.HiddenFor(model => model.Email)
        @Html.HiddenFor(model => model.Gender)
        @Html.HiddenFor(model => model.Hobby)
        @Html.HiddenFor(model => model.Job)
        <input type="submit" value="Correggi" class="btn btn-default form-control" />
    </form>
</div>
```

I dati vengono passati al modello di questa view tramite l'azione
`public IActionResult DataCheck(IFormCollection collection)` che utilizza il metodo
`private User CollectionToUser(IFormCollection collection)` per ricostruire il modello tramite i dati contenuti nella collection e lo ritorna assieme alla View.

Una volta che viene premuto il pulsante "Registra", viene richiamato il metodo
`public IActionResult Save(IFormCollection collection)` che ricostruisce il modello (un'istanza di User) con il metodo `private User CollectionToUser(IFormCollection collection)`, poi crea due istanze della classe CSVHelper, una per il file giornaliero ed una per il file "Registrazioni_tutte.csv".

```
CSVHelper all = new CSVHelper("wwwroot/Registrazioni/Registrazioni_tutte.csv", ";", new
 string[] { "data", "nome", "cognome", "data_nascita", "via", "numero_civico", "citta",
 "nap", "telefono", "email", "sesso", "hobby", "professione" });

CSVHelper day = new CSVHelper(string.Format("wwwroot/Registrazioni/Registrazioni_{0}_{1
}_{2}.csv", dt.Year, dt.Month, dt.Day), ";", new string[] { "data", "nome", "cognome",
"data_nascita", "via", "numero_civico", "citta", "nap", "telefono", "email", "sesso", "
hobby", "professione" });
```

Poi richiama il metodo `public void Write(string[] values)`, passando un array di stringhe che contiene la data di oggi convertita in stringa e i valori del modello, poi redireziona alla pagina 'Index' del ReadCheckController con il codice `return RedirectToAction("", "ReadCheck");`

3.3.2.1 CSVHelper

La classe CSVHelper contiene 4 proprietà:

- `public string Path`, che specifica il percorso in cui salvare / da cui leggere il file
- `public string Separator`, che indica il separatore tra i valori all'interno del file
- `public string[] Columns`, che determina il nome delle colonne all'interno del file
- `public Encoding Encoding`, che precisa la codifica con cui salvare / leggere il file

Essa contiene anche 4 metodi costruttori:

- `CSVHelper(string path, string separator, string[] columns, Encoding encoding)`
- `CSVHelper(string path, string[] columns, Encoding encoding)`
- `CSVHelper(string path, string separator, string[] columns)`
- `CSVHelper(string path, string[] columns)`

Inoltre ci sono 3 metodi generali:

- `public void Write(string[] values)`, che scrive i valori contenuti all'interno di un array di stringhe dentro una nuova riga del file CSV al quale punta il valore della proprietà Path, eventualmente creando il file e la struttura delle cartelle in caso esse non esistano
- `public static void AddToBeginning(string text, string path)`, che scrive del testo, contenuto nella stringa `text` come prima riga di un file CSV al quale punta la stringa `path`
- `public List<string[]> ReadAllLines()`, che legge il contenuto del file al quale punta il valore della proprietà Path e ritorna una lista di array di stringhe contenente i valori delle righe del file, un valore per elemento dell'array

3.3.2.1.1 Metodo Write

Per prima cosa questo metodo, che accetta come parametro un array di stringhe chiamato `values` che contiene i valori da scrivere nel file, controlla se la cartella di destinazione esista, con la seguente condizione:
`if (!Directory.Exists(Path.Remove(Path.LastIndexOf("/"))))`
In caso essa non dovesse esistere, viene creata con il metodo `CreateDirectory`:
`Directory.CreateDirectory(Path.Remove(Path.LastIndexOf("/")));`
Poi viene creata una prima istanza della classe StringBuilder chiamata `valuesToWrite` che contiene i dati da scrivere nel file, e una seconda chiamata `header` che contiene l'intestazione del file, ossia i nomi delle colonne, che viene subito riempito con un loop foreach che percorra i valori all'interno della proprietà `Columns` ed il metodo `header.Append(column + Separator)`. In seguito viene controllato se il file che si trova nel percorso contenuto nella proprietà `Path` contenga come prima riga l'intestazione, in caso negativo viene aggiunta tramite il metodo `AddToBeginning`.

Infine viene controllato se il numero di valori passati come parametro `values` sia uguale al numero di colonne contenute nella proprietà `Columns`, in caso affermativo viene usato un loop foreach che percorre il parametro `values` ed ognuno dei suoi elementi viene aggiunto a `valuesToWrite`, separato dal valore della proprietà Separator, salvando alla fine il valore di `valuesToWrite` nel file.

```
//If there's as many values as the columns of the file
if (values.Length == Columns.Length)
{
    //Create the row
    foreach (string value in values)
    {
        valuesToWrite.Append(value + Separator);
    }

    //Remove the last separator
    valuesToWrite.Remove(valuesToWrite.Length - 1, 1);

    //Add the text to the file and create it
    File.AppendAllText(Path, valuesToWrite.ToString() + "\n", Encoding);
}
else
{
    throw new FormatException("Il numero di colonne e di valori sono differenti.");
}
```

3.3.2.1.2 Metodo AddToBeginning

Questo metodo statico accetta come parametri una stringa `text` che contiene il testo da scrivere ed una stringa `path` che contiene il percorso al file, si occupa di scrivere del testo come prima riga di un file.
Questo viene fatto tramite un procedimento trovato su StackOverflow, dove l'autore afferma che questo sia il metodo che utilizza meno memoria RAM in assoluto per svolgere questo lavoro:
Viene creato un file temporaneo in cui vengono scritti in ordine il testo contenuto `text` all'interno del parametro e poi i contenuti del file che si trova al percorso contenuto nel parametro `path`, poi il contenuto di questo file temporaneo viene rimpiazzato con quello originale a cui punta il percorso contenuto nel parametro.

```
public static void AddToBeginning(string text, string path)
{
    string tempfile = System.IO.Path.GetTempPath() + Guid.NewGuid().ToString() + ".csv";
    using (var writer = new StreamWriter(tempfile))
    using (var reader = new StreamReader(path))
    {
        writer.WriteLine(text);
        while (!reader.EndOfStream)
            writer.WriteLine(reader.ReadLine());
    }
    File.Copy(tempfile, path, true);
}
```

3.3.2.1.3 Metodo ReadAllLines

Questo metodo, che non accetta parametri, si limita a prendere i valori ricevuti dal metodo `File.ReadAllLines(Path, Encoding)` e li trasforma in array di stringhe, separandoli con il metodo `Split(Separator)` e li aggiunge ad una lista che viene restituita. In questo modo il risultato è una lista di

array di stringhe, dove ogni array contiene i valori di una riga del file senza separatore.

```
public List<string[]> ReadAllLines()
{
    List<string[]> re = new List<string[]>();
    string[] lines = File.ReadAllLines(Path, Encoding);
    foreach (string line in lines)
    {
        re.Add(line.Split(Separator));
    }
    return re;
}
#endregion
```

3.4 Pagina verifica scrittura

Questa pagina contiene solo una view, ossia quella di verifica della scrittura dei dati. Usa la classe ReadCheckController.

3.4.1 Pagina di verifica della corretta scrittura dei dati

Questa view contiene un messaggio di ringraziamento per essersi registrati, delle istruzioni per tornare alla pagina principale ed una tabella che viene popolata con tutti i dati letti dal file giornaliero. In fondo alla pagina compare un bottone per venire redirezionati alla pagina Index dell'HomeController.
Per farlo, viene usato come modello un'istanza di `IEnumerable<Progetto_1.Models.User>` invece che un semplice `Progetto_1.Models.User` come nel resto dell'applicazione.
La tabella è contenuta all'interno di un div di classe `table-responsive` e contiene una tabella di classe `table table-hover` al cui interno ci sono due sezioni: `thead`, che contiene le intestazioni della tabella e `tbody` che contiene un loop foreach che percorre tutti gli elementi all'interno di Model che genera tanti tag td quante le proprietà di User. Ogni elemento che si trova all'interno della tabella ha un tooltip che mostra il suo valore per intero, visto che la tabella, essendo responsive, potrebbe rimpicciolirsi e non mostrare i valori al suo interno per intero.

```
<div class="table-responsive">
    <table class="table table-hover">
        <thead>
            <tr>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.FirstName)">
                    @Html.DisplayNameFor(model => model.FirstName)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.LastName)">
                    @Html.DisplayNameFor(model => model.LastName)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.Birthday)">
                    @Html.DisplayNameFor(model => model.Birthday)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.Address)">
                    @Html.DisplayNameFor(model => model.Address)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.CivicNumber)">
                    @Html.DisplayNameFor(model => model.CivicNumber)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.City)">
                    @Html.DisplayNameFor(model => model.City)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.Nap)">
                    @Html.DisplayNameFor(model => model.Nap)
                </th>
                <th data-toggle="tooltip" data-placement="top" title="@Html.DisplayNameFor(model => model.PhoneNumber)">
```

Figura 6: ReadCheck - thead

```
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.FirstName)">
                @Html.DisplayFor(modelItem => item.FirstName)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.LastName)">
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.Birthday)">
                @Html.DisplayFor(modelItem => item.Birthday)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.Address)">
                @Html.DisplayFor(modelItem => item.Address)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.CivicNumber)">
                @Html.DisplayFor(modelItem => item.CivicNumber)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.City)">
                @Html.DisplayFor(modelItem => item.City)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.Nap)">
                @Html.DisplayFor(modelItem => item.Nap)
            </td>
            <td data-toggle="tooltip" data-placement="top" title="@Html.DisplayFor(modelItem => item.PhoneNumber)">
                @Html.DisplayFor(modelItem => item.PhoneNumber)
```

Figura 7: ReadCheck - tbody

Per costruire il modello da passare alla view viene utilizzate nuovamente la classe CSVHelper. Questo viene fatto nel controller, dove viene istanziato un nuovo oggetto di tipo CSVHelper identico a quello usato per salvare i dati nel file giornaliero, solo che viene poi creata una lista di oggetti di tipo User chiamata `users`, che viene riempita leggendo le righe del file, ottenute con il metodo `ReadAllLines()`, istanziando gli user da inserire nella classe con il costruttore che accetta come parametro un array di stringhe, ossia i valori letti dal file. Ogni riga corrisponde quindi ad un utente. Viene infine ritornata la lista creata come Enumerable usando il metodo `users.AsEnumerable()`.

```
public IActionResult Index()
{
    CSVHelper csv = new CSVHelper(
        string.Format("wwwroot/Registrazioni/Registrazioni_{0}_{1}_{2}.csv", DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day),
        ";",
        new string[] {
            "data", "nome", "cognome", "data_nascita", "via", "numero_civico",
            "citta", "nap", "telefono", "email", "sesso", "hobby", "professione" });
    List<User> users = new List<User>();
    try
    {
        List<string[]> readFromFile = csv.ReadAllLines();
        for (int i = 1; i < readFromFile.Count; ++i)
        {
            User u = new User(readFromFile.ElementAt(i));
            users.Add(u);
        }
    }
    catch (FileNotFoundException)
    {

    }

    return View(users.AsEnumerable());
}
```

4 Test

4.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

Test Case: **Riferimento**:	TC-001 REQ-001	**Nome:**	Verificare che, in mancanza dei file CSV, essi vengono creati prima di scriverci dentro
Descrizione:	Verificare che, in mancanza dei file CSV, essi vengono creati prima di scriverci dentro i dati inseriti dall'utente		
Prerequisiti:	Dei dati da salvare nei file CSV La struttura delle cartelle all'interno delle quali salvare i file corretta		
Procedura:	1. Cancellare I file CSV dalla cartella Registrazioni 2. Avviare l'applicazione e proseguire con la procedura di registrazione fino al punto in cui viene premuto "Registrati" all'interno della pagina di verifica		
Risultati attesi:	I file "Registrazione_tutte.csv" e "Registrazione_aaaa_mm_gg.csv" sono stati creati all'interno della cartella Registrazioni e al loro interno si trovano i dati passati		

Test Case: **Riferimento**:	TC-002 REQ-003	**Nome:**	Verificare che premendo il bottone "Azzera" vengano cancellati tutti i campi
Descrizione:	Verificare che quando l'utente preme il pulsante "Azzera" i campi vengano cancellati e rimangano solamente dei campi vuoti.		
Prerequisiti:			
Procedura:	1. Avviare l'applicazione e proseguire con la procedura di registrazione fino alla pagina di inserimento dei dati 2. Inserire dei dati in almeno uno dei campi 3. Premere il pulsante "Azzera"		
Risultati attesi:	Tutti i valori inseriti nei campi sono stati azzerati.		

Test Case: **Riferimento**:	TC-003 REQ-003	**Nome:**	Verificare che sia impossibile lasciare la pagina di inserimento se una delle validazioni non venga passata
Descrizione:	Verificare che non sia possibile lasciare la pagina di inserimento dei dati se una delle validazioni dei campi in cui sono stati inseriti i dati non venga passata		
Prerequisiti:	Dei dati da inserire nei campi della pagina di inserimento dei dati		
Procedura:	4. Avviare l'applicazione e proseguire con la procedura di registrazione fino alla pagina di inserimento dei dati 5. Inserire dei dati non validi in almeno uno dei campi 6. Premere il pulsante "Registra"		
Risultati attesi:	Il bordo dei campi in cui sono stati inseriti dei dati non validi diventa rosso e compare un messaggio di errore sotto di essi. Inoltre viene selezionato il primo dei campi non validi e viene evitato il passaggio alla prossima pagina		

Test Case: **Riferimento**:	TC-004 REQ-003	**Nome:**	Verificare che, premendo il pulsante "Correggi" della pagina di verifica, l'utente venga riportato alla pagina precedente contenente tutti i dati che egli ha inserito.
Descrizione:	Verificare che, premendo il pulsante "Correggi" della pagina di verifica del corretto inserimento dei dati, l'utente venga riportato alla pagina di inserimento dei dati contenente tutti i valori che egli ha inserito in modo da poterli correggere senza doverli riscrivere.		
Prerequisiti:	Dei dati da inserire nei campi della pagina di inserimento dei dati		
Procedura:	1. Avviare l'applicazione e proseguire con la procedura di registrazione fino alla pagina di inserimento dei dati		

	2. Inserire dei dati validi in tutti i campi 3. Premere il pulsante “Registra” 4. Premere il pulsante “Correggi”
Risultati attesi:	Si viene riportati alla pagina di inserimento dei dati contenente tutti i dati inseriti al punto 2.

Test Case: **Riferimento**:	TC-005 REQ-003	**Nome:**	Verificare che sia possibile registrare un utente senza specificare né l’hobby né la professione
Descrizione:	Verificare che si possa proseguire con la registrazione di un utente anche senza inserire un valore per i campi “Hobby” e “Professione”		
Prerequisiti:	Dei dati da inserire nei campi della pagina di inserimento dei dati		
Procedura:	1. Avviare l’applicazione e proseguire con la procedura di registrazione fino alla pagina di inserimento dei dati 2. Inserire dei dati validi in tutti i campi tranne “Hobby” e “Professione” 3. Premere il pulsante “Registra”		
Risultati attesi:	Si viene riportati alla pagina di verifica dei dati normalmente		

4.2 Risultati test

Test case	Risultato
TC-001	I file vengono creati con l’intestazione e al loro interno viene scritta la registrazione corrente
TC-002	Una volta premuto il bottone, i campi si svuotano
TC-003	È impossibile passare alla pagina di verifica se le validazioni non vengono passate
TC-004	L’utente viene riportato alla pagina di inserimento e i campi sono riempiti con i dati inseriti precedentemente
TC-005	È possibile proseguire senza specificare hobby e professione

4.3 Mancanze/limitazioni conosciute

Non ci sono mancanze né limitazioni conosciute.

5 Consuntivo

Anche se mi sono ritrovato a consegnare in orario, ho dovuto lavorare molto al di fuori delle ore di lezione, quindi la mia pianificazione è stata svolta in modo totalmente errato e la causa di ciò deve essere che ho sottovalutato il lavoro da svolgere. La prossima volta dovrò prevedere un lavoro maggiore per ogni attività.

6 Conclusioni

Anche se, come detto anche nel punto 1.4, esistono tantissime applicazioni di questo tipo che per la maggior parte salvano i dati inseriti all’interno di un database invece che all’interno di un file CSV come questa, trovo che realizzare questa applicazione mi abbia aiutato molto da un punto di vista formativo. Pur non cambiando il mondo, questo progetto ha gettato delle basi che si trovano molto spesso in altri lavori, come lo scrivere in un file, CSV o meno.

6.1 Sviluppi futuri

È possibile aggiungere o togliere dei campi, o inserire una pagina di login confrontando i dati inseriti con quelli salvati.

6.2 Considerazioni personali

In questo progetto ho imparato a non fare applicazioni web con Visual Studio perché possono essere caricate unicamente su IIS, quindi su Windows Server, il che è un peccato perché l'applicazione si è rivelata molto facile da realizzare visto che è stata realizzata per la maggior parte in C#.

7 Bibliografia

7.1 Sitografia

https://www.iconfinder.com, *Access, login, register, signup icon,* consultato in data 17 ottobre 2018 (ringrazio Mariano Tardon per l'icona)
http://www.stackoverflow.com, *StackOverflow*, consultato in data 24 ottobre 2018
https://www.w3schools.com, *w3schools.com*, consultato in data 26 ottobre 2018
https://api.jquery.com/click/, *.click() | jQuery API documentation*, 26 ottobre 2018
http://api.jquery.com/prop/, *.prop() | jQuery API documentation*, 26 ottobre 2018

8 Allegati

- Diari di lavoro