

Corso di Laurea in Informatica

Sviluppo di un applicativo web per il tracciamento dei container in ambito marittimo

Tutore accademico:

Giovanna Rosone

Candidato:

Mirko Michele D'Angelo

Tutore aziendale:

Stefano Garzelli

ANNO ACCADEMICO 2022/2023

Indice

Introduzione	1
1 Introduzione	1
1.1 L'azienda	1
1.2 L'esperienza	2
1.3 Il problema	2
1.4 Struttura della relazione	3
2 Strumenti utilizzati	4
2.1 IIS server	4
2.2 NuGet	4
2.3 Git	4
2.4 Visual Studio	5
2.5 Open project	6
2.6 GitLab	6
2.7 SQL server	7
2.7.1 I DBMS relazionali	7
2.8 SQL Server Management Studio	8
2.9 MSBuild	8
3 Tecnologie utilizzate	9
3.1 Lo Structured Query Language	9
3.2 Le tecnologie del server	10
3.2.1 C#	10
3.2.2 Json.NET	10
3.2.3 EntityFramework	10
3.2.4 DataManagerCore	11
3.2.5 .NET	13

3.2.6	ASP.NET	14
3.2.7	ASP.NET MVC framework	14
3.2.8	EasyPortal	23
3.2.9	NPOI e NPOI.mapper	23
3.3	Le tecnologie del client	24
3.3.1	Bootstrap	24
3.3.2	Javascript e Typescript	24
3.3.3	JQuery	25
3.3.4	Tabulator	26
3.3.5	Tom Select	26
3.3.6	Sweet alert 2	27
4	Il progetto realizzato	28
4.1	Il contesto dell'applicazione	28
4.1.1	La ISO 6346	29
4.2	Descrizione del progetto	33
4.3	Struttura dell'applicazione	34
4.3.1	Funzionamento di una pagina	37
4.4	La base di dati	39
4.5	Funzionamento dell'applicazione	42
4.6	Anagrafiche	43
4.6.1	Anagrafica movimenti contenitori	44
4.6.2	Anagrafica sequenza movimenti	45
4.6.3	Anagrafica contenitori	46
4.7	Sezione movimenti contenitori	48
4.7.1	Generazione automatizzata di report	50
4.8	Organizzazione del progetto	54
4.9	CI/CD	56
5	Conclusioni	57

1. Introduzione

La presente relazione ha lo scopo di descrivere l'esperienza, le competenze e le nozioni apprese dal candidato durante il suo tirocinio curricolare presso l'azienda Blue Team Computers s.r.l.u. [2].

L'obiettivo del tirocinio è far cimentare il candidato col ciclo di sviluppo delle funzionalità di un'applicazione web in ambito aziendale.

1.1 L'azienda

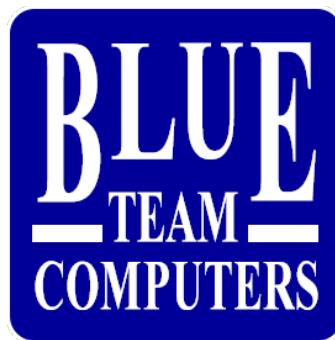


Figura 1.1: Logo dell'azienda.

L'azienda, denominata Blue Team Computers con logo rappresentato in figura 1.1, è stata fondata nel 1983 ed è situata vicino all'area portuale di Livorno. Si specializza nello sviluppo di software e consulenza tecnica nel settore marittimo e commerciale per conto di altre aziende.

I servizi offerti comprendono lo sviluppo di software, applicazioni web e mobili personalizzati insieme al supporto tecnico, consulenza, installazione e manutenzione di reti aziendali.

1.2 L'esperienza

L'esperienza è durata circa 2 mesi, dal 17 luglio fino all'8 settembre 2023, si è svolta all'interno dell'azienda e occasionalmente da remoto sotto la supervisione del proprietario e tutore aziendale Stefano Garzelli e del dipendente Michele Bruzzone.

L'esperienza è suddivisa in due fasi:

1. formazione: durante questa fase il tutore aziendale ha formato il candidato sulle tecnologie e gli strumenti utilizzati nel progetto.
2. sviluppo del progetto: in questa fase il candidato ha impiegato le conoscenze apprese per realizzare le sezioni dell'applicazione.

Il candidato ha iniziato lo sviluppo di ogni sezione tramite un documento fornito dall'azienda, esso contiene la descrizione tutte le funzionalità richieste e le informazioni necessarie per realizzare la sezione in questione.

In ogni momento il candidato ha sempre avuto a disposizione una figura di riferimento all'interno dell'azienda per chiedere aiuto in caso di difficoltà.

1.3 Il problema

Nel trasporto marittimo viene fatto largo uso dei contenitori per poter trasportare merci di vario tipo, con necessità e caratteristiche molto diverse, pertanto risulta vitale tracciare e monitorare il loro stato e i loro spostamenti.

Il software, chiamato "Agency Kit", impiegato fino ad adesso dall'azienda committente è molto datato e risulta poco pratico da utilizzare per assolvere al compito.

L'obiettivo del progetto è l'implementazione, da parte del candidato, delle funzionalità di tracciamento dei contenitori e i loro movimenti all'interno della nuova versione del programma, chiamata "Agency Kit Web".

1.4 Struttura della relazione

La relazione è strutturata in 4 capitoli ognuno dedicato a un aspetto del tirocinio svolto dal candidato:

- il capitolo 2 illustra gli strumenti principali utilizzati con le loro caratteristiche e le funzionalità utili ai fini del progetto.
- il capitolo 3 tratta le tecnologie impiegate nel progetto spiegando il loro funzionamento e ruolo all'interno del progetto.
- il capitolo 4 documenta il progetto realizzato dal candidato.
- l'ultimo capitolo contiene le osservazioni finali sul progetto e sulle capacità sviluppate dal candidato.

2. Strumenti utilizzati

Questo capitolo ha lo scopo di illustrare i principali strumenti utilizzati all'interno del progetto e se necessari i concetti utili alla comprensione del loro funzionamento.

2.1 IIS server

IIS server [10] è il server utilizzato per lo sviluppo dell'applicazione, dispone di una grande integrazione con Visual Studio e ASP.NET oltre ad avere una struttura modulare e altamente configurabile.

Per lo sviluppo e il debug in locale è stata utilizzata la variante IIS Express che è più leggera ed è ottimizzata per gli sviluppatori. Per l'ambiente di test aziendale è stata impiegata la versione normale.

2.2 NuGet

NuGet [21] è un software per la gestione di pacchetti utilizzato per la distribuzione di software sviluppato con il framework .NET.

Viene impiegato dall'azienda per la pacchettizzazione e distribuzione del suo software proprietario.

È stato utilizzato all'interno del progetto per la gestione delle librerie utilizzate.

2.3 Git

Git [8] è un software di versionamento del codice, consente il salvataggio e tracciamento dei progressi fatti durante lo sviluppo insieme alla possibilità di sviluppare più versioni in parallelo del codice e fonderle successivamente.

Le versioni prendono il nome di **branch** e sono tutte raggruppate all'interno di un'unica **repository**.

Le repository possono essere poi salvate tramite software appositi per l'hosting di repository come GitLab [9] su server privati o servizi cloud.

2.4 Visual Studio

Visual studio [31] è un ambiente di sviluppo integrato ed è stato lo strumento di sviluppo principale durante tutto lo svolgimento del tirocinio.

La sua scelta è stata motivata dai seguenti fattori:

- interazione con i database SQL integrata, in quanto consente la visualizzazione della struttura del database all'interno del programma. Se si fa uso del framework EntityFramework consente la generazione automatica delle classi C# per interagire col database tramite il codice.
- molte funzionalità di supporto alla scrittura del codice come template già pronti per velocizzare la creazione di pezzi di codice o l'autocompletamento del codice durante la scrittura.
- template già pronti per i progetti rendendo più veloce la scrittura di un'applicazione e l'avvio di progetti.
- integrazione con NuGet per l'importazione, l'utilizzo e l'organizzazione di librerie all'interno del progetto.
- sistema di versionamento git integrato. Le sue funzionalità sono state incorporate all'interno del programma rendendo utilizzabili in pochi click tutte le operazioni necessarie sulla repository.
- sistema di organizzazione dei file dell'applicazione, infatti i file sono raggruppati logicamente in un progetto.

In particolare, il progetto è composto da un file, detto **file di progetto** con estensione .csproj, contenente informazioni sul progetto e da altri file contenenti il codice del progetto stesso, oppure documenti multimediali usati dall'applicazione, ad esempio un'immagine mostrata a schermo.

A loro volta i progetti sono contenuti all'interno di una **soluzione**, con un corrispondente **file di soluzione**, con estensione .sln, che indica i progetti che ne fanno parte. Il vantaggio di questo sistema è la facilità di condivisione e riutilizzo dei progetti all'interno di più soluzioni, ciò consente la modularizzazione di interi parti di un applicativo e il loro riuso per necessità future.

2.5 Open project

Open Project [23] è un software di project management per la collaborazione in gruppo. Consente la creazione e l'assegnamento di attività da svolgere ai membri del gruppo e il tracciamento del loro andamento e stato.

È disponibile tramite interfaccia web senza necessità di installazione, in alternativa è possibile installare una versione self-hosted su server privati.

È stato impiegato per tenere traccia delle attività svolte e delle funzionalità sviluppate.

2.6 GitLab

GitLab [9] è un software che consente la gestione di repository git. E' stato impiegato dall'azienda come principale software per la gestione delle repository git dei loro progetti. Il software offre anche supporto a meccanismi di Continuous Integration/Continuous Deployment (CI/CD) [5].

L'approccio CI/CD

La CI/CD [5], è un insieme di metodologie e pratiche combinate al fine di automatizzare le operazioni complementari allo sviluppo del software come compilazione, deploy, test dei componenti sviluppati, etc.

All'interno di GitLab sono stati integrati meccanismi per consentire l'approccio CI/CD chiamati **pipelines**.

Una pipeline consente l'esecuzione automatica di comandi secondo un ordine specifico definito tramite il file *.gitlab-ci.yml* presente nella repository del progetto.

Nel file vengono definiti gli **stage** che indicano una fase dell'esecuzione. I comandi da eseguire vengono specificati all'interno dei **job**, che specificano gli stage e le condizioni per eseguire le azioni definite.

La pipeline viene eseguita automaticamente ogni volta che vengono fatti cambiamenti nella repository.

Tramite l'interfaccia grafica di GitLab è possibile visionare l'andamento, i log e il risultato dell'esecuzione.

2.7 SQL server

SQL server [24] è un **DBMS relazionale**, che utilizza la variante del linguaggio SQL detta TRANSACT-SQL.

È stato impiegato come base di dati per il progetto per via della sua grande integrazione con Visual Studio e le sue performance.

2.7.1 I DBMS relazionali

Un Database Management System (DBMS) è un sistema progettato per consentire la creazione, interrogazione e manipolazione di una base di dati.

Un DBMS relazionale fa uso del “modello relazionale” [17] per rappresentare i dati.

Il modello prevede la strutturazione dei dati attraverso **relazioni** che possiedono **attributi**.

Le relazioni modellano, tramite i propri attributi, il contesto rappresentato dalla base di dati insieme a legami e proprietà delle entità al suo interno.

Le relazioni sono rappresentate come tabelle dove gli attributi sono le colonne, ogni riga della tabella indica un'entità di quel tipo. All'interno di ogni tabella i valori di una o più colonne identificano in modo univoco i dati appartenenti alla loro riga e insieme formano la **chiave primaria**.

Alcune colonne, dette **chiavi esterne**, servono a modellare legami con un'altra entità. I loro valori sono quelli delle chiavi primarie della riga interessata, che può essere della stessa tabella o di una differente.

Ogni tabella ha una sola chiave primaria e può avere, o meno, più chiavi esterne.

Le colonne utilizzate per formare le due tipologie di chiavi sono determinate durante la fase di progettazione della base di dati.

L'insieme delle tabelle di un base di dati compone lo **schema della base di dati** della quale è riportato un esempio di istanza in figura 2.1.

The diagram illustrates a relational database instance with three tables:

- Table 1:** Activity
|Activity Code|Activity Name|
|23|Patching|
|24|Overlay|
|25|Crack Sealing|
- Table 2:** Activity Log
|Activity Code|Date|Route No.|
|24|01/12/01|I-95|
|24|02/08/01|I-66|
- Table 3:** Route Log
|Date|Activity Code|Route No.|
|01/12/01|24|I-95|
|01/15/01|23|I-495|
|02/08/01|24|I-66|

A vertical arrow points from the 'Activity Code' column of Table 1 to the 'Activity Code' column of Table 2, with the label 'Key = 24' indicating the primary key mapping. Another vertical arrow points from the 'Activity Code' column of Table 1 to the 'Activity Code' column of Table 3.

Figura 2.1: Esempio di istanza per una base di dati relazionale.

2.8 SQL Server Management Studio

SQL Server Management Studio [25] è un software per la gestione di basi di dati create con SQL server. Offre un' interfaccia grafica per la visualizzazione e modifica dello schema di una base di dati e delle sue tabelle, e la possibilità di effettuare query SQL sulla base di dati visualizzandone il risultato e/o l'effetto sulla base di dati.

2.9 MSBuild

MSBuild [18] è la piattaforma di compilazione delle applicazioni impiegata nel progetto. È stata utilizzata per automatizzare la compilazione e il deploy dell'applicazione tramite la pipeline CI/CD offerta da GitLab.

3. Tecnologie utilizzate

Questo capitolo è dedicato all'illustrazione delle tecnologie utilizzate nel progetto e dei concetti necessari alla loro comprensione.

Le varie tecnologie sono state suddivise in base alle componenti dell'applicazione dove vengono impiegate.

3.1 Lo Structured Query Language

La base di dati fa uso dello Structured Query Language [26] (SQL), che è un linguaggio dichiarativo standardizzato per DBMS relazionali con funzionalità per:

- la creazione e modifica dello schema della base di dati;
- la modifica e lettura dei dati delle tabelle;
- il controllo degli accessi ai dati.

Fornisce una sintassi per accedere alle funzionalità sopra elencate. È stato usato dal candidato durante lo sviluppo del progetto per interagire con la base di dati tramite il programma SQL server illustrato nella sezione 2.8.

In figura 3.1 viene mostrato un esempio di una query SQL che legge i dati all'interno della tabella **NOME_TABELLA**.

Da **NOME_TABELLA** recupera tutte le righe dove l'attributo **CAMPO3** è maggiore di 0, le ordina secondo l'attributo **CAMPO1**, e restituisce le colonne **CAMPO1** e **CAMPO2**.

```
SELECT CAMPO1, CAMPO2  
FROM NOME_TABELLA  
WHERE CAMPO3 > 0  
ORDER BY CAMPO1
```

Figura 3.1: Esempio di query SQL.

3.2 Le tecnologie del server

3.2.1 C#

C# [4] è il linguaggio di programmazione Object Oriented utilizzato per la realizzazione del backend, offre una sintassi semplice da capire.

Fornisce buona parte delle librerie e funzionalità utilizzate all'interno del progetto, dispone inoltre della tecnologia LINQ [15] che consente l'uso di una sintassi simile a quella di SQL.

3.2.2 Json.NET

Json.NET [14] è un framework per serializzazione e deserializzazione di oggetti C# da o verso il formato JSON. La libreria offre alte prestazioni e l'accesso alle sue funzionalità con i metodi **SerializeObject** e **DeserializeObject**, che dato un oggetto passato come parametro effettuano, rispettivamente, la serializzazione di una struttura dati nel formato JSON e l'operazione inversa.

È stata impiegata nel progetto per la realizzazione di alcune parti delle API del backend.

3.2.3 EntityFramework

EntityFramework [7] è un Object Relational Mapper (ORM), che consente l'interazione con i database SQL tramite i costrutti offerti da C# oppure tramite la sintassi LINQ.

Oltre alle funzionalità di ORM il framework consente l'astrazione delle complessità legate alla gestione della connessione alla base di dati, lasciando allo sviluppatore solo il compito di archiviare, recuperare, eliminare e modificare i dati. In figura 3.2 viene illustrato un esempio del suo utilizzo.

```
database.NOME_TABELLA.Where(p => p.CAMPO3 > 0)
    .Select(p=>new {p.CAMPO1,p.CAMPO2})
    .OrderBy(p=> new{p.CAMPO4} )
```

```
from tab in database.NOME_TABELLA
where tab.CAMPO3 > 0
orderby new {tab.CAMPO4}
select new {tab.CAMPO1,CAMPO2}
```

Figura 3.2: Esempio di utilizzo di EntityFramework per effettuare la stessa query della figura 3.1, in cui sono mostrati una query realizzata con i metodi degli oggetti e un'altra che fa uso della sintassi LINQ.

L'accesso alle funzionalità EntityFramework avviene tramite i metodi di classi che possono essere generate in due modi:

1. **code first**: si genera lo schema della base di dati a partire dalla struttura delle classi.
2. **database first**: si generano delle classi a partire dallo schema della base di dati.

Per il progetto è stata usata le seconda modalità.

3.2.4 DataManagerCore

È una libreria proprietaria dell'azienda, fornisce funzionalità non direttamente presenti su EntityFramework rimanendo comunque compatibile con la libreria, come mostrato nelle figure 3.3 e 3.4.

```
dataManager.Select(database.NOME_TABELLA,
                   p=>new {p.CAMP01,p.CAMP02})
    .Where(p => p.CAMP03 > 0)
    .OrderBy(p=> new{p.CAMP04} )
```

Codice 3.3: Codice che utilizza il framework DatamanagerCore per realizzare la query.

```
dataManager.Any(database.NOME_TABELLA,
                 p=>new {p.CAMP01,p.CAMP02})
    .Where(p => p.CAMP03 > 0)
    .OrderBy(p=> new{p.CAMP04} )
```

Codice 3.4: Esempio dell'uso dell'operatore ANY in DataManagerCore.

Inoltre offre anche la possibilità di effettuare log delle operazioni fatte sul proprio server.

È stata impiegata all'interno del progetto per l'interrogazione della base di dati da parte del server.

L'Object Relational Mapping

Il problema che si pone con l'uso di un DBMS relazionale è quello di conciliare la rappresentazione dei dati all'interno della base di dati con le strutture dati fornite dal linguaggio usato.

Per questo motivo è stato ideato l'Object Relational Mapping (ORM) [22], che è una tecnica per l'interazione con una base di dati relazionale, direttamente all'interno del linguaggio utilizzato.

Un ORM fornisce i comandi per interagire con base di dati nascondendo tutti i dettagli implementativi, rende il codice indipendente dal DBMS utilizzato e riduce notevolmente la quantità di manutenzione, testing e codice necessari la manipolazione dei dati.

L'utilizzo di ORM comporta anche dei lati negativi, come difficoltà nella scrittura di query complesse con la sintassi del linguaggio o funzionalità mancanti dall'ORM ma presenti nel DBMS utilizzato.

Nel progetto le problematiche poste sono state superate utilizzando il supporto alla sintassi LINQ di EntityFramework, che è più semplice da utilizzare per query complesse, e tramite l'uso del framework aziendale DataManagerCore per avere accesso alle funziona-

lità mancanti da EntityFramework, come ad esempio l'operatore ANY.

Il candidato ha impiegato entrambi i framework nel progetto, alternando il loro utilizzo come ritenuto più opportuno in base alle circostanze.

3.2.5 .NET

.NET è la piattaforma di sviluppo creata da Microsoft per lo sviluppo general purpose di applicativi, è pensata per supportare l'uso di più modelli e linguaggi di programmazione contemporaneamente in un progetto.

La piattaforma funziona grazie all'ambiente di esecuzione **Common Language Runtime** (CLR) [6]. Il CLR esegue un linguaggio intermedio chiamato **Common Intermediate Language** (CIL).

Durante l'esecuzione il codice del linguaggio di programmazione utilizzato viene tradotto dinamicamente nel CIL e viene eseguito dal CLR, come mostrato in figura 3.5.

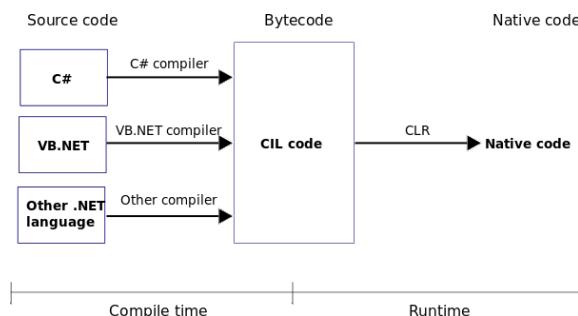


Figura 3.5: Schema rappresentativo del funzionamento dell'ambiente .NET [6].

Questo sistema fa sì che sia possibile utilizzare più tecnologie e linguaggi contemporaneamente in un progetto, utilizzando un solo ambiente di esecuzione.

Oltre al CLR la piattaforma offre molte librerie per lo sviluppo di applicazioni di vario tipo, ad esempio la libreria ASP.NET per lo sviluppo di applicazioni web.

3.2.6 ASP.NET

ASP.NET è una delle librerie offerte di .NET, fornisce funzionalità per la creazione di applicazioni web usando Visual Basic o C#. Nel caso progetto è stato utilizzato quest'ultimo.

Il paradigma utilizzato in questo framework è quello della programmazione a eventi dove vengono invocati blocchi di codice in risposta a eventi fuori dal controllo del programmatore.

Il framework, inoltre, offre vari moduli che consentono approcci diversi alla realizzazione di applicazioni web, fra questi il candidato ha utilizzato il modulo MVC per realizzare le funzionalità principali del progetto.

3.2.7 ASP.NET MVC framework

ASP.NET MVC consente la creazione di applicazioni web basate sull'implementazione di un'architettura Model View Controller [16].

Il modulo fornisce i **Controller** per gestire il routing delle richieste http e le **View** per impostare la presentazione dei contenuti, questi due componenti vengono combinati per elaborare la pagina all'interno server e inviare il risultato al client.

Le View MVC

Le **View**, o **Viste**, MVC sono file con estensione .cshtml, che consentono l'uso della sintassi **Razor** per la creazione dinamica di pagine web.

Fornisce la possibilità di usare la sintassi html insieme a quella di C# per la presentazione dei contenuti, oltre a provvedere un meccanismo per il passaggio dei dati dal Controller alla vista come illustrato nella sezione 3.2.7.

L’elaborazione di una vista è composta da due elementi:

- un file .cshtml chiamato **layout**, che definisce la struttura base della pagina con alcune parti che verranno prese dalla vista ed è utilizzabile con più viste diverse.
- la vista vera e propria, anche questo è un file .cshtml che definisce gli elementi che verranno inseriti all’interno del layout durante la creazione della pagina.

Esistono inoltre le viste parziali, che sono utilizzate per parti di interfacce utilizzate più volte in un’applicazione.

La sintassi Razor

Il carattere @ consente l’utilizzo di codice C# e di direttive utili per la creazione della pagina, fra le varie direttive **@section** è stata particolarmente utile al progetto: essa consente la definizione di sezioni che verranno poi inserite all’interno del layout.

Nel progetto ogni vista è stata divisa in tre parti:

- una sezione **Styles**, contiene codice CSS per lo stile della pagina;
- una sezione con sintassi Razor per la struttura dei contenuti, che non ha bisogno di una sezione apposita;
- una sezione **Scripts**, contiene codice Javascript per le funzioni dinamiche della pagina.

Quando il server deve restituire una pagina al client elabora la sintassi Razor della vista corrispondente, il risultato sarà una pagina html che verrà inviata come risposta insieme a tutti gli altri file necessari.

I metodi helper

Oltre alla sintassi di C# vengono forniti i metodi “helper”, che consentono la creazione di elementi html e altre funzionalità utili tramite i metodi forniti dalle classi **Html** e **Url**.

Durante lo sviluppo del progetto sono stati particolarmente utili due metodi helper:

- **Url.Action** che genera un url completo partendo da uno parziale;
- **Html.RenderPartial** che consente l'utilizzo delle viste parziali, inserendo il loro contenuto nel punto in cui viene collocato.

In figura 3.6 viene mostrata una vista impiegata nell'applicazione, oltre a essere impostata con la struttura appena descritta, mostra anche un esempio di impiego dei metodi helper e della sintassi Razor:

- A riga 66 viene fatto uso della sintassi C# per la generazione dinamica della pagina;
- nella riga 107 viene impiegato Html.RenderPartial per riutilizzare una vista già presente;
- nella riga 111 viene mostrato un esempio di utilizzo di Url.Action per la generazione degli url.

```
1 @section Styles{
2     <link href="/lib/Tabulator/css/tabulator.min.css" rel="stylesheet">
3     <link href="/Styles/Quotations/QuotationStyle.css" rel="stylesheet">
4     <link rel="stylesheet" href="/lib/TabulatorPersonalize/css/tabulatorPersonalization.css" />
5     <link href="/Styles/Registry/RegistryStyle.css" rel="stylesheet">
6     <style>
7         .translate {
8             color: #757575;
9         }
10        [id$="-ts-control"],
11        .plugin-dropdown_input.focus.dropdown-active .ts-control {
12            background-position: center right;
13            background-repeat: no-repeat;
14            background-size: 0.5rem;
15            background-image: url("https://www.reshot.com/download/icons/8EV2HR4UBG/reshot-icon-down-arrow-
triangle-8EV2HR4UBG-faa80.svg");
16            border-bottom-color: #ced4da;
17            border-left-color: transparent;
18            border-right-color: transparent;
19            border-top-color: transparent;
20            border-bottom: 1px solid #ced4da;
21            box-shadow: none;
22            background-color: transparent !important;
23            height:42px;
24        }
25    </style>
26}
27<div class="card p-3 mt-2 backgroundColor quotationSearchCard">
28    <div>
```

```

31     <div class="float-right">
32         <button data-toggle="collapse" href="#collapseSearch" role="button" aria-expanded="true" aria-
33             controls="collapseSearch">
34             <i id="closeSearchView" class="far fa-eye-slash fa-2x"></i>
35             <i id="openSearchView" class="far fa-eye fa-2x" style="display: none;"></i>
36         </button>
37     </div>
38     <h4> Search</h4>
39 </div>
40 <div class="collapse show" id="collapseSearch">
41     <div class="input-group">
42         <div class="col-2">
43             <div class="md-form">
44                 <label class="active mb-3" style="font-weight: bold;" for="codeSearch">
45                     <span class="translate">Code:</span>
46                 </label>
47                 <input id="codeSearch" maxlength="3" type="text" class="form-control"/>
48             </div>
49         <div class="col-3">
50             <div class="md-form">
51                 <label class="active mb-3" style="font-weight: bold;" for="descriptionSearch">
52                     <span class="translate">Description:</span>
53                 </label>
54                 <input id="descriptionSearch" maxlength="40" type="text" class="form-control"/>
55             </div>
56         </div>
57         <div class="col-2">
58             <div class="md-form">
59                 <div class="mb-3">
60                     <label class="active " style="font-weight: bold;" for="descriptionSearch">
61                         <span class="translate">Automatic state:</span>
62                     </label>
63                 </div>
64                 <select id="statusSearch" name="AUTO_STATE">
65                     <option value="0">&nbsp</option>
66                     @foreach (var item in ViewBag.Data.StatusData)
67                     {
68                         <option value="@item.IdRecord"> @item.Description</option>
69                     }
70                 </select>
71             </div>
72         </div>
73         <div class="col-5">
74             </div>
75         </div>
76     </div>
77 </div>
78
79 <div class="container-fluid">
80     <div class="row justify-content-end">
81         <button id="searchButton"
82             style="background-color:dodgerblue !important"
83             class="btn btn-primary bmd-btn-fab waves-effect waves-light"
84             alt="Apply search filters"
85             title="Apply search filters">
86             <i class="fas fa-search"></i>
87         </button>
88         <button id="addButton" style="background-color:dodgerblue !important"
89             class="btn btn-primary bmd-btn-fab waves-effect waves-light"
90             data-toggle="modal"
91             data-target="#modalMovement">

```

```

92         alt="Create new movement"
93         title="Create new movement">
94         <i class="fas fa-plus"></i>
95     </button>
96   </div>
97 </div>
98 <div id="movements-table" class="tabulator m-4">
99 </div>
100
101 @section Scripts{
102     <script type="text/javascript" src="https://unpkg.com/tabulator-tables@4.7.2/dist/js/tabulator.min.js"></
103     script>
104     <script type="text/javascript" src="/Scripts/Akit/Registry/MovementsFunctions.js"></script>
105     <script type="text/javascript" src="/Scripts/Akit/Utils/ModalParserFunctions.js"></script>
106     <script type="text/javascript" src="/Scripts/Akit/Utils/DisplaySnackBarFunctions.js"></script>
107     <script type="text/javascript" src="/lib/TabulatorPersonalize/js/TabulatorPersonalizationTS.js"></script>
108     @{
109         Html.RenderPartial("Modal/_ContainerMovementsEditModal");
110     }
111     <script>
112         _saveContainerMovementURL = "@Url.Action("SaveOrUpdateMovementData", "Registry")";
113         _getMovementURL = "@Url.Action("GetMovementData", "Registry")";
114         _tableDataURL = "@Url.Action("GetMovementTableData", "Registry")";
115         _getStatusDataURL = "@Url.Action("GetStatusData", "Registry")";
116         $(document).ready(() => {
117             setupTableData();
118         })
119         $("#addButton").on('click', () => {
120             wipeModalData("#modalMovement")
121             automaticMovement.setValue("0")
122             defaultStatus.setValue("0")
123             automaticStatus.setValue("0")
124         })
125         $("#saveButton").on('click', () => {
126             addOrUpdateContainerMovement()
127         })
128         $("#cancelButton").on('click', (e) => {
129             wipeModalData("#modalMovement")
130         })
131         $(window).on('beforeunload', ()=>{
132             wipeModalData("#modalMovement")
133         });
134         $('#searchButton').on('click', (e) => {
135             e.preventDefault();
136             handleSearchFilters()
137         })
138     </script>
}

```

Codice 3.6: Vista utilizzata per realizzare la sezione vista movimenti all'interno dell'applicazione.

I controller MVC

ASP.NET introduce il concetto di **controller** con **azioni** che restituiscono **risultati**.

Un controller è un oggetto che eredita dalla classe **BaseController**, e serve a raggruppare un insieme di azioni.

Ogni azione è un metodo del controller che viene invocato con una richiesta http all'url relativo /NomeController/NomeMetodo, tramite gli attributi del tipo [**Http(METODO HTTP)**] è possibile restringere l'insieme di metodi http alla quale l'azione può rispondere, e in base al tipo di ritorno del metodo cambia il contenuto della risposta.

Tramite l'oggetto **RouteCollection** è possibile stabilire un'azione di default da invocare in caso l'url della richiesta non combaci con nessuna delle azioni del controller.

Struttura di un'azione

La struttura di un'azione del controller è divisa in tre parti:

1. la firma del metodo: determina i dati che dovranno essere passati al metodo tramite la richiesta http.

MVC si occupa autonomamente dell'elaborazione dei dati fornendo accesso ad essi tramite i parametri del metodo, come mostrato in figura 3.7.

2. il corpo della funzione: in questa parte è possibile accedere a metodi speciali per l'elaborazione dei dati e all'oggetto **ViewBag**, che consente il passaggio dei dati dal metodo alla vista durante l'elaborazione della richiesta.

3. il tipo di ritorno: determina il tipo di contenuto restituito al client, ASP.NET mette a disposizione il tipo **ActionResult** che indica un tipo qualsiasi e lascia al programmatore la scelta, da questo tipo derivano poi tutti gli altri tipi più specifici.

Il candidato ha fatto uso dei seguenti tipi:

- **ViewResult**: restituisce una vista che si trova all'interno della cartella **View** con lo stesso nome del metodo.

- **JsonResult**: l'oggetto restituito è in formato JSON.
- **FileResult**: viene restituito un file di qualsiasi tipo che verrà scaricato sul browser.

In figura 3.7 viene mostrato un esempio di utilizzo combinato fra una vista e un controller MVC, l'azione utilizzata risponde solo a richieste di tipo Get e all'interno del suo metodo viene utilizzata la ViewBag passare dati alla vista.

Nella vista viene utilizzato il metodo helper **DropDownList** per creare un elemento SELECT tramite i dati recuperati dalla ViewBag.

```
public class Controller {
    ...
    [HttpGet]
    public ViewResult GetOrdini(string userId){
        List<Order> orders=retrieveUserOrders(userId);
        //aggiunta dati alla ViewBag
        ViewBag.UserOrders=orders .Select(o=>new SelectListItem{
            Value=o.Id ,
            Text=o.Name
        })
        .ToList();
        return View();
    }
    ...
}
```

```
<ul>
@if(user .IsLoggedIn){
    <div>
        <p>ciao @user.Name benvenuto </p>
        <p>questi sono i tuoi ordini </p>
        @Html.DropDownList("dati_utente",ViewBag .UserOrders )
    </div>
}
</ul>
```

Figura 3.7: Esempio di utilizzo tipo di una vista ASP.NET e di passaggio dei dati dal controller tramite ViewBag.

L'elaborazione delle richieste di ASP.NET cambia in base al tipo di richiesta http, se sono di tipo GET allora verranno passati come parametri le coppie chiave-valore della query string dell'url. Nel caso di una richiesta di tipo POST i parametri devono essere

inseriti sotto forma di oggetto JSON nel corpo della richiesta.

In entrambi i casi il framework convertirà i parametri della richiesta nella struttura dati del parametro formale, nel metodo invocato, come mostrato in figura 3.8. Il codice viene invocato solo per richieste http di tipo POST, prima di rispondere effettua un controllo sulla validità dei dati e inserisce o modifica i dati del movimento nella base di dati. Se uno dei controlli fallisce invierà al client una risposta in formato JSON contenente i parametri errati oppure un messaggio che notifica il successo dell'operazione.

```
1 [HttpPost]
2 public JsonResult SaveOrUpdateMovement(MOVIMENTI movement, bool forceSequence, string sequenceText, List<string>
3     mandatoryFields)
4 {
5     //non possono essere vuoti tutti e due i campi PREV_MOV e NEXT_MOV
6     if ((movement.COD_PREV_MOV == "" || movement.COD_PREV_MOV == null) &&
7         (movement.COD_NEXT_MOV == "" || movement.COD_NEXT_MOV == null))
8     {
9         return Json(new
10            {
11                success = false,
12                messages = new List<string> { "the movement must have a next movement and a previous movement" }
13            });
14     }
15     //controllo se sono presenti tutti i dati necessari per mantenere correttamente la catena
16     if (!_dm.Any(_db.MOVIMENTIS, m => m.ID_RECORD == movement.COD_PREV_MOV) && (movement.COD_PREV_MOV != ""))
17     ||
18     (!_dm.Any(_db.MOVIMENTIS, m => m.ID_RECORD == movement.COD_NEXT_MOV) && (movement.COD_NEXT_MOV != "0"))
19     ||
20     !_dm.Any(_db.CONTAINERS, m => m.ID_RECORD == movement.COD_CONTAINER))
21     {
22         return Json(new
23            {
24                success = false,
25                messages = new List<string> { "the previous movement,next movement and container must be already
26                present" }
27            });
28     }
29
30     //controlli sulla validit dei dati del movimento
31     bool validBooking = (movement.COD_BOOKING != null &&
32         movement.COD_BOOKING != "") ?
33             (_dm.Any(_db.BOOKING_EQUIPMENT, x => x.COD_BOOKING == movement.COD_BOOKING &&
34                 x.SIGLAEQUIPMENT == movement.COD_CONTAINER)) :
35                 true;
36
37     bool validBL = (movement.COD_POLIZZA != null &&
38         movement.COD_POLIZZA != "") ?
39             (_dm.Any(_db.POLIZZE_EQUIPMENT, x => x.COD_POLIZZA == movement.COD_POLIZZA &&
40                 x.SIGLAEQUIPMENT == movement.COD_CONTAINER)) :
41                 true;
42
43     bool mandatoryData = mandatoryFields.Any((field) => typeof(MOVIMENTI).GetProperty(field) != null &&
44         (string)typeof(MOVIMENTI).GetProperty(field)
45             .GetValue(movement) != "");
46
47     bool isPrevValidMovement = true;
48     bool prevValidDate = true;
```

```

45     MOVIMENTI prevMovement = null;
46     //caso speciale modifica primo movimento -> controllo sul movimento precedente saltato
47     if (movement.COD_PREV_MOV != "")
48     {
49         prevMovement = _db.MOVIMENTIs.Single(x => x.ID_RECORD == movement.COD_PREV_MOV);
50
51         isPrevValidMovement = !forceSequence ?
52             _dm.Any(_db.MOVIMENTI_VALIDI, x => (x.COD_MOV_PREC == prevMovement.COD_MOVIMENTO) &&
53                 (x.COD_MOV == movement.COD_MOVIMENTO) &&
54                 (x.COD_STATO == movement.COD_STATUS))
55             : true;
56         prevValidDate = (movement.DATA.Value != null && prevMovement.DATA.Value != null) && (DateTime.
57             Compare(movement.DATA.Value, prevMovement.DATA.Value) >= 0);
58     }
59
60     bool isValidMovement = true;
61     bool validDate = true;
62     //inserimento nel mezzo -> saltato se      un inserimento in testa
63     MOVIMENTI nextMovement = null;
64     if (movement.COD_NEXT_MOV != null && movement.COD_NEXT_MOV != "0")
65     {
66         nextMovement = _db.MOVIMENTIs.Single(x => x.ID_RECORD == movement.COD_NEXT_MOV);
67
68         //controlli sulla validit dei dati
69         isValidMovement = !forceSequence ?
70             _dm.Any(_db.MOVIMENTI_VALIDI, x => (x.COD_MOV_PREC == movement.COD_MOVIMENTO) &&
71                 (x.COD_MOV == nextMovement.COD_MOVIMENTO) &&
72                 (x.COD_STATO == nextMovement.COD_STATUS))
73             : true;
74         validDate = (movement.DATA.Value != null && prevMovement.DATA.Value != null) && (DateTime.Compare(
75             nextMovement.DATA.Value, movement.DATA.Value) >= 0);
76         _db.MOVIMENTIs.AddOrUpdate(nextMovement);
77     }
78     if (!validBooking ||
79         !validBL ||
80         !isPrevValidMovement ||
81         !prevValidDate ||
82         !mandatoryData ||
83         !isValidMovement ||
84         !validDate)
85     {
86         List<string> messages = new List<string>();
87         if (!validBooking)
88             messages.Add("the container must be present in the booking");
89         if (!validBL)
90             messages.Add("the container must be present in the bill of lading");
91         if (!prevValidDate)
92             messages.Add("the date of the new movement must be present and later than it's previous movement");
93         if (!mandatoryData)
94             messages.Add("some of the mandatory fields are not present");
95         if (!isPrevValidMovement)
96             messages.Add("the movement sequence between the previous and current movement must be valid");
97         if (!validDate)
98             messages.Add("the date of the new movement must be present and must be earlier than it's next movement");
99
100        if (!isValidMovement)
101            messages.Add("the movement sequence between the current and next movement must be valid");
102
103        return Json(new
104        {
105            success = false,
106            messages = messages.ToArray()
107        });

```

```

104         });
105     }
106     //se    nuovo devo dargli un ID
107     if (movement.ID_RECORD == null || movement.ID_RECORD == "")
108         movement.ID_RECORD = (DateTime.Now.Ticks).ToString();
109     if (prevMovement != null)
110     {
111         movement.COD_PREV_MOV = prevMovement.ID_RECORD;
112         prevMovement.COD_NEXT_MOV = movement.ID_RECORD;
113         _db.MOVIMENTIS.AddOrUpdate(prevMovement);
114     }
115     if (nextMovement != null)
116     {
117         movement.COD_NEXT_MOV = nextMovement.ID_RECORD;
118         nextMovement.COD_PREV_MOV = movement.ID_RECORD;
119         _db.MOVIMENTIS.AddOrUpdate(nextMovement);
120     }
121     //se l'inserimento    in testa bisogna aggiornare anche il LAST_MOVEMENT del container
122     if (movement.COD_NEXT_MOV == "0")
123     {
124         CONTAINER container = _db.CONTAINERS.Single(c => c.ID_RECORD == movement.COD_CONTAINER);
125         container.LAST_MOVEMENT = movement.ID_RECORD;
126         _db.CONTAINERS.AddOrUpdate(container);
127     }
128     _db.MOVIMENTIS.AddOrUpdate(movement);
129     _db.SaveChanges();
130
131     return Json(new {
132         success = true,
133         messages = new List<string> { "container movement updated with success" }
134     });
135 }

```

Codice 3.8: Azione del controller responsabile dell'aggiornamento dei movimenti di un contenitore.

3.2.8 EasyPortal

EasyPortal è un framework proprietario di Blue Team Computers, che consente la gestione dell'autenticazione degli utenti e dei permessi a essi associati per l'accesso alle funzionalità offerte.

Questo framework viene utilizzato come base da modificare per poter realizzare l'applicazione desiderata.

3.2.9 NPOI e NPOI.mapper

NPOI [19] è una libreria per la generazione, tramite codice, di documenti in vari formati.

Durante il tirocinio il candidato l'ha usata per la generazione di report in formato .xlsx.

Per facilitare la generazione dei dati è stata utilizzata l'estensione NPOI.mapper [20], che mette a disposizione la possibilità di creare una mappatura fra dati in un oggetto e le celle del foglio di calcolo facilitando la creazione dei report.

3.3 Le tecnologie del client

Quest'ultima sezione è dedicata alla descrizione delle tecnologie utilizzate dal client con le loro funzionalità e utilità ai fini del progetto.

3.3.1 Bootstrap

Bootstrap [3] è un framework CSS utilizzato per lo stile delle pagine web, fornisce una serie di classi CSS per consentire la creazione in modo rapido di stili personalizzati per componenti html.

È stato impiegato per la realizzazione delle interfacce utente delle sezioni dell'applicazione.

3.3.2 Javascript e Typescript

Javascript [12] è un linguaggio di programmazione orientato verso la programmazione a eventi, dispone di una sintassi con molte funzionalità, supporta il paradigma a oggetti e fornisce l'accesso alle altre tecnologie utilizzate nel client.

E' *debolmente tipato*, ovvero i dati possono cambiare dinamicamente il loro tipo durante l'esecuzione, questo però durante lo sviluppo porta a molti errori e comportamenti inaspettati del codice.

Per questo motivo viene impiegato anche Typescript [30], che estende javascript aggiungendo i tipi ai dati ed un controllo statico della loro correttezza. Grazie al controllo statico dei tipi riesce a impedire errori comuni durante lo sviluppo e rende più semplice la documentazione e la comprensione del codice.

I due linguaggi sono stati usati per realizzare le funzionalità dinamiche dell'interfaccia grafica e l'interazione del client con il server.

3.3.3 JQuery

Jquery [13] è una libreria di Javascript che fornisce funzionalità per manipolazione, navigazione e gestione degli eventi degli elementi della pagina html insieme al supporto alla tecnologia AJAX.

Il simbolo \$ consente l'accesso alle sue funzionalità, può essere utilizzato per interagire con gli elementi della pagina oppure per eseguire una richiesta http asincrona utilizzando la tecnologia AJAX.

In figura 3.9 viene mostrato un esempio di utilizzo di jQuery, tramite il metodo **val** recupera il valore di un elemento di input, effettua una richiesta di tipo post con dati di tipo json al server, se ha successo utilizza di nuovo il metodo val per impostare il valore di un elemento e mostrare il risultato.

```
let value:string=$("#InputElement").val()
$.ajax(url,
{
    method:"POST",
    contentType:"application/json",
    data:JSON.stringify({dataValue:value}),
    success:(result)=>{
        $("#outputElement").val(result.value)
    },
    error:():=>{
        alert("impossibile contattare il server")
    }
})
```

Figura 3.9: Esempio di codice che Javascript utilizza AJAX tramite la libreria JQuery.

È stato impiegato per la realizzazione le funzionalità dinamiche dell'interfaccia grafica e per realizzare l'interazione fra client e server per il recupero dinamico dei dati.

La tecnologia AJAX

L'acronimo AJAX [1] sta per Asynchronous JavaScript and XML, ed è una tecnologia che consente l'interazione fra client e server in modo dinamico e asincrono, senza la necessita

di bloccare l'esecuzione di script nel client o di ricaricare la pagina.

Le sue funzionalità sono accessibili tramite jQuery come mostrato nella figura 3.9, oppure è possibile creare delle funzioni utilizzate da librerie, come Tabulator e Tom Select, per recuperare autonomamente i dati dal server.

3.3.4 Tabulator

Tabulator [28] è una libreria che consente la visualizzazione in formato tabellare dei dati. Fornisce la funzionalità di ricerca, manipolazione e formattazione dei dati mostrati in figura 3.10. Offre anche supporto alla tecnologia AJAX per facilitare il recupero dinamico dei dati dal server.

È stata utilizzata nella maggior parte delle sezioni presenti nel progetto per la visualizzazione dei dati.

Field:									Type:	=	Value:	value to filter	Clear Filter
Name	Progress	Gender	Rating	Favourite C...	Date Of Birth	Driver							
Oli Bob	[progress bar]	male	★★★★★	red	14/04/1984	✓							
Mary May	[progress bar]	female	★★★★★	blue	14/05/1982	✓							
Christine Lobowski	[progress bar]	female	★★★★★	green	22/05/1982	✓							
Brendon Philips	[progress bar]	male	★★★★★	orange	01/08/1990	✗							
Margret Marmajuke	[progress bar]	female	★★★★★	yellow	31/01/1999	✗							
Frank Harbours	[progress bar]	male	★★★★★	red	12/05/1966	✓							
Jamie Newhart	[progress bar]	male	★★★★★	green	14/05/1985	✓							
Gemma Jane	[progress bar]	female	★★★★★	red	22/05/1982	✓							

Figura 3.10: Esempio che mostra alcune delle funzionalità fornite da Tabulator.

3.3.5 Tom Select

Tom Select [29] è una libreria che consente la creazione di elementi di tipo SELECT con più funzionalità mostrati in figura 3.11.

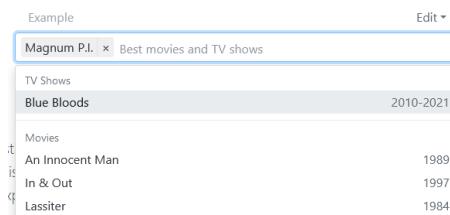


Figura 3.11: Esempio di SELECT creato utilizzando Tom Select.

Fornisce una serie di funzioni per utilizzare la tecnologia AJAX oppure interagire con gli eventi della libreria come ordinamento e ricerca delle opzioni.

Questa libreria è stata impiegata all'interno del progetto per la realizzazione dell'interfaccia utente delle varie sezioni realizzate dal candidato.

3.3.6 Sweet alert 2

Sweet Alert 2 [27] è una libreria che consente la creazione di alert interattivi mostrati in figura 3.12.

Fornisce un API che facilita la creazione di alert più interattivi: è possibile inserire animazioni, pulsanti e testi dinamici e molto altro tramite un singolo oggetto passato al metodo **Swal.fire**.

È stata utilizzata nel progetto per notificare informazioni importanti all'utente.

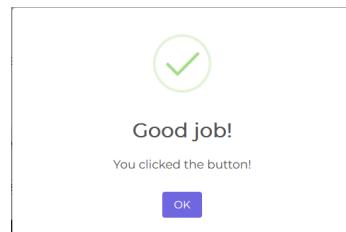


Figura 3.12: Esempio di alert realizzato con sweetalert2.

4. Il progetto realizzato

Questo capitolo ha lo scopo di documentare il progetto illustrando il funzionamento dell'applicazione realizzata insieme alle funzionalità e la logica implementate dal candidato.

4.1 Il contesto dell'applicazione

L'applicazione è stata progettata per un committente che gestisce dei contenitori per il trasporto merci in ambito marittimo, per conto dei suoi clienti.

Di ogni contenitore il committente vuole tenere traccia delle seguenti caratteristiche:

- lo stato attuale, ovvero se è danneggiato, prenotato, etc. ;
- il tipo del container, ad esempio può essere normale, refrigerato, etc. ;
- se è un contenitore refrigerato allora vuole anche sapere i dati dell'attrezzatura per la refrigerazione;
- il materiale di cui è fatto;
- il suo numero identificativo, che è un codice univoco per i contenitori regolato dalla ISO 6346 illustrata nella sezione 4.1.1;
- il proprietario del contenitore;
- i suoi spostamenti;
- eventuali contratti di leasing o sub-leasing.

I contenitori in uso da un cliente sono indicati col termine “parco contenitori”, su di essi vengono fatti dei “movimenti”, che indicano lo spostamento di un contenitore per un motivo specifico, effettuato da un trasportatore, via terra o mare, da o verso un porto, una località o un deposito, in una specifica data.

4.1.1 La ISO 6346

La ISO 6346 [11] si occupa di definire l'identificazione a livello internazionale dei contenitori attraverso un codice univoco.

Il codice è composto da undici caratteri alfanumerici composti da lettere maiuscole e numeri come segue:

- identificativo del proprietario: questo campo occupa 3 lettere ed è unico per ogni proprietario di contenitori.
- categoria: questo campo occupa una lettera e può assumere tre valori, **U** per contenitori merci, **J** per attrezzature rimovibili relative al contenitore e **Z** per rimorchi e telai.
- numero seriale: sono sei numeri unici assegnati dal proprietario, identificano univocamente il contenitore all'interno della sua flotta.
- check digit: è un carattere finale che consente la validazione del codice e della sua trasmissione, deve essere calcolato utilizzando gli altri campi.

In figura 4.1 viene mostrato un esempio di numero valido per un contenitore.

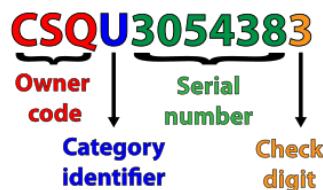


Figura 4.1: immagine d'esempio di un numero valido per un contenitore

Il check digit viene calcolato col seguente algoritmo:

1. assegnare un valore a ogni carattere, per i numeri corrisponde al numero stesso, per le lettere viene assegnato progressivamente partendo dalla lettera A con 10 e omettendo multipli di 11 come mostrato in figura 4.2.

A	B	C	D	E	F	G	H	I	J	K	L	M
10	12	13	14	15	16	17	18	19	20	21	23	24
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
25	26	27	28	29	30	31	32	34	35	36	37	38

Figura 4.2: Esempio di assegnamento corretto del valore alle lettere.

2. calcolare la posizione di ogni carattere andando in modo crescente da sinistra verso destra e partendo da 0 per il primo carattere.

Moltiplicare il valore assegnato al primo punto per $2^{posizione\ carattere}$.

3. sommare tutti i valori ottenuti dal secondo passo e arrotondare la somma per difetto.
4. sottrarre il risultato del terzo punto con quello del primo.

Il risultato sarà il check digit.

Questo algoritmo è stato impiegato sia nel client che nel server per controllare il corretto inserimento dei numeri dei contenitori.

In figura 4.3 è riportata l'implementazione utilizzata nel client, la funzione CheckContainer dopo aver effettuato controlli sulla validità del parametro, esegue l'algoritmo appena descritto per calcolare il check digit. Se il valore calcolato combacia con quello presente nel numero del contenitore allora è conforme e restituisce il valore *true*, altrimenti restituisce il valore *false* e il numero non è conforme.

Questa implementazione fa uso delle seguenti funzioni di supporto:

- CleanConNumberString ripulisce la stringa restituendo un codice di 11 caratteri con solo lettere maiuscole e numeri.
- GetCheckNumber restituisce l'ultimo carattere del numero.
- GetSumm esegue l'algoritmo di calcolo del check digit.

```

var alphabet = {
    //Add Letters
    'A': 10,
    'B': 12,
    'C': 13,
    'D': 14,
    'E': 15,
    'F': 16,
    'G': 17,
    'H': 18,
    'I': 19,
    'J': 20,
    'K': 21,
    'L': 23,
    'M': 24,
    'N': 25,
    'O': 26,
    'P': 27,
    'Q': 28,
    'R': 29,
    'S': 30,
    'T': 31,
    'U': 32,
    'V': 34,
    'W': 35,
    'X': 36,
    'Y': 37,
    'Z': 38,
    //Add Numbers
    '0': 0,
    '1': 1,
    '2': 2,
    '3': 3,
    '4': 4,
    '5': 5,
    '6': 6,
    '7': 7,
    '8': 8,
    '9': 9,
};

function CheckContainer(containerNumberToCheck:string)
{
    //Clean the input string from Chars that are not in the Alphabet
    let containerNumber:string = CleanConNumberString(containerNumberToCheck);
    //Return true if the input string is empty
    //Used mostly for DataGridView to set the False validation only on false Container Numbers
    //and not empty ones
    if (containerNumber === "") return true;
    //Return False if the input string has not enough Characters
    if (containerNumber.length != 11) return false;
    //Get the Sum of the ISO Formula
    let summ:number = GetSumm(containerNumber);
    //Calculate the Check number with the ISO Formula
    let tempCheckNumber:number = summ - (Math.floor(summ / 11) * 11);
    //Set temCheckNumber 0 if it is 10 - In somme cases this is needed
    if (tempCheckNumber == 10) tempCheckNumber = 0;
    //Return true if the calculated check number matches with the input check number
    if (tempCheckNumber == GetCheckNumber(containerNumber))
        return true;
    //If no match return false
    return false;
}

```

```

}

function CleanConNumberString(inputString:string)
{
    //Set all Chars to Upper
    let resultString:string = inputString.toUpperCase();
    //Loop Through all chars
    for (let i: number = 0; i < inputString.length - 1; i++) {
        let c:string = inputString[i];
        if (alphabet[c] == undefined)
            resultString = resultString.replace(c, ""); //Remove chars with the String.Replace Method
    }
    //Return the cleaned String
    return resultString;
}

/// <summary>
/// Provides the Check number from a Container number string
/// </summary>
/// <param name="inputString">String of the Container number</param>
/// <returns>Integer of the Check number</returns>
function GetCheckNumber(inputString:string)
{
    //Loop if string is longer than 1
    if (inputString.length > 1) {
        //Get the last char of the string
        let checkChar:string = inputString[inputString.length - 1];
        //Initialise a integer
        let CheckNumber:number = 0;
        //Parse the last char to a integer
        if (!isNaN(parseInt(checkChar))) {
            CheckNumber = parseInt(checkChar);
            return CheckNumber;
        }
    }
    //If parsing can't be done and the string has just 1 char or is empty
    //Return 11 (A number that can't be a check number!!!)
    return 11;
}

/// <summary>
/// Calculate the sum by the ISO Formula
/// </summary>
/// <param name="inputString">String of the Container number</param>
/// <returns></returns>
function GetSumm(inputString:string)
{
    //Set summ to 0
    let summ:number = 0;
    //Calculate only if the container string is not empty
    if (inputString.length > 1) {
        //Loop through all chars in the container string
        //EXCEPT the last char!!!
        for (let i:number = 0; i < inputString.length - 1; i++)
        {
            //Get the current char
            let temChar:string = inputString[i];
            //Initialise a integer to represent the char number in the ISO Alphabet
            //Set it to 0
            let charNumber:number = 0;
            //If Char exists in the Table get its number
            if (alphabet[temChar] != undefined)
                charNumber = alphabet[temChar];
            //Add the char number to the sum using the ISO Formula
            summ += charNumber * (Math.pow(2, i));
        }
    }
}

```

```

        }
    }

    //Return the calculated summ
    return summ;
}

```

Codice 4.3: Codice Typescript utilizzato nel client che controlla la conformità alla iso 6346.

4.2 Descrizione del progetto

Il progetto consiste nello sviluppo da parte del candidato della funzionalità di tracciamento dei contenitori e dei loro movimenti nell'applicazione “Agency Kit Web”.

L'applicativo è una web app con lo scopo di rimpiazzare il vecchio programma, mostrato in figura 4.4, offrendo un'interfaccia web moderna, accessibile da una vasta gamma di dispositivi e con funzionalità migliorate.

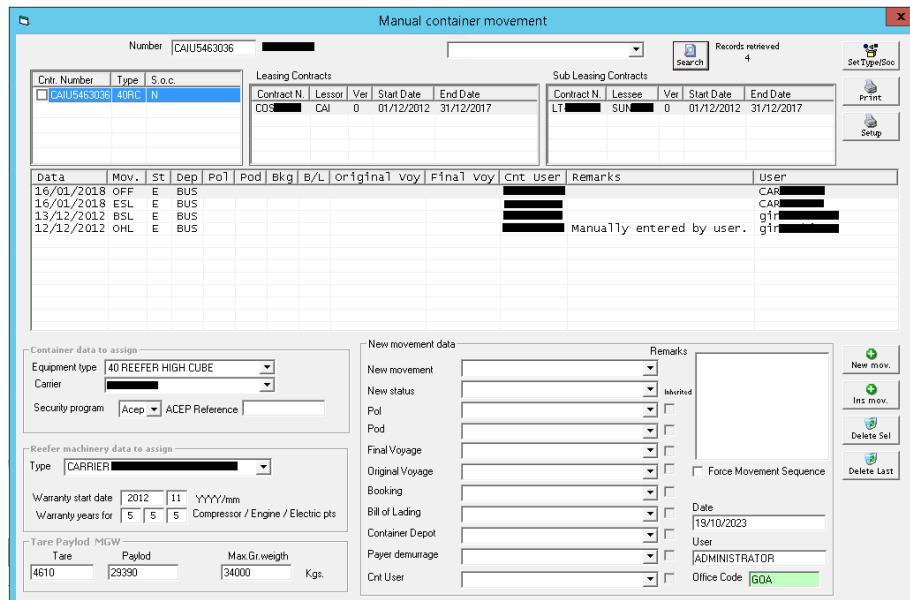


Figura 4.4: Menù principale dell'applicazione “Agency Kit”.

L'applicazione non è una semplice trasposizione da programma per desktop a web app, ma è una re-implementazione delle funzionalità offerte dal vecchio programma applicando tecnologie e metodologie di sviluppo più moderne, offrendo un valore aggiunto maggiore rispetto alla sua precedente versione.

4.3 Struttura dell'applicazione

L'applicazione è stata realizzata seguendo un'implementazione moderna dello stile architettonico Model View Controller [16], che prevede la divisione dell'applicazione in tre componenti, illustrati nella figura 4.5, che interagiscono fra di loro, ognuno con un compito:

- Model: fornisce i meccanismi di accesso ai dati presenti nella base di dati. Questa parte è stata realizzata tramite i meccanismi di accesso forniti da EntityFramework, DataManagerCore e le classi generate.
- View: ha il compito di fornire l'interfaccia grafica usata dall'utente per interagire con l'applicazione, comunicare al Controller le azioni dell'utente e aggiornare l'interfaccia per riflettere le azioni dell'utente.

Questa parte è implementata utilizzando server e client:

- il server utilizza le viste ASP.NET e il passaggio dei dati dal Controller per la creazione dinamica dell'interfaccia.
 - il client, tramite le tecnologie della sezione 3.3, utilizza il protocollo http per comunicare col Controller, offre l'interazione con l'interfaccia grafica all'utente e aggiorna dinamicamente la pagina del browser.
- Controller: Ha la responsabilità di gestire le richieste http fatte dal client, interagisce con il Model per leggere o modificare i dati interessati e notifica al client il risultato dell'azione invocata.

L'implementazione viene fatta tramite i Controller e le azioni offerte da ASP.NET.

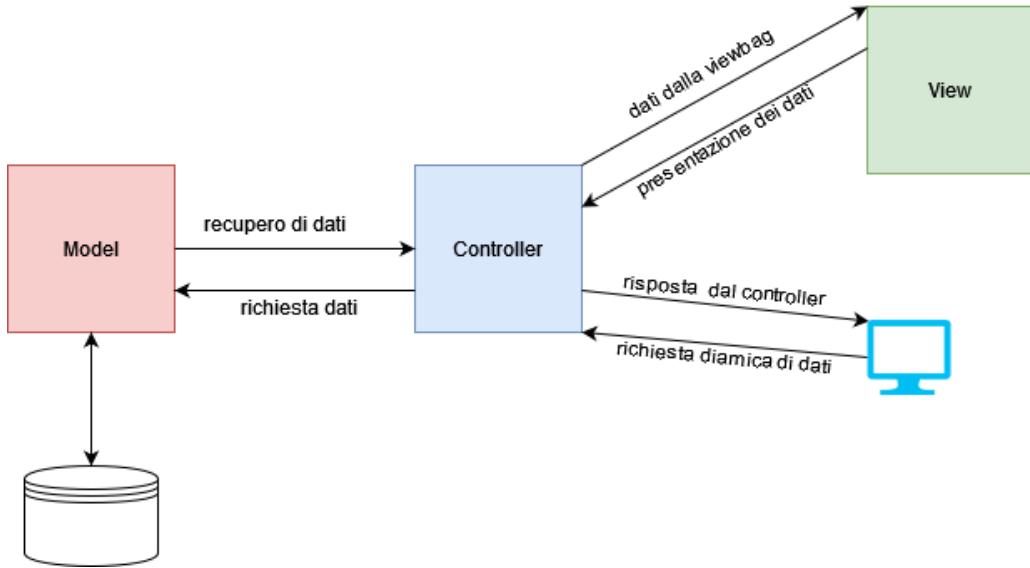


Figura 4.5: Implementazione dello stile MVC di ASP.NET.

Questo stile offre un buon grado di separazione fra i dati e la loro rappresentazione, in questo modo ogni componente è responsabile di un solo aspetto dell'applicazione, ciò conferisce una struttura altamente modulare, riutilizzabile e facile da mantenere al codice. La comunicazione dinamica fra Controller e client usa una rappresentazione intermedia per i dati, Json per il progetto, che fornisce una serie di vantaggi:

- non si pone il problema della conversione fra formati diversi usati da client e server.
- consente rappresentazioni diverse dei dati di uno stesso controller.
- rende possibile il riutilizzo delle viste all'interno di altre.
- è possibile introdurre nuove tecnologie all'interno dell'applicazione senza problemi di integrazione.

Tutte le caratteristiche appena elencate rendono lo stile Model View Controller particolarmente adatto all'utilizzo per lo sviluppo di applicazioni web.

L'implementazione di ASP.NET

Nell'implementazione originale di MVC è prevista una struttura come quella della figura 4.6, ogni volta che vengono modificati i dati mostrati all'utente, per via delle sue azioni o di cambiamenti esterni, il Model deve aggiornare la View inviando al client la pagina aggiornata.

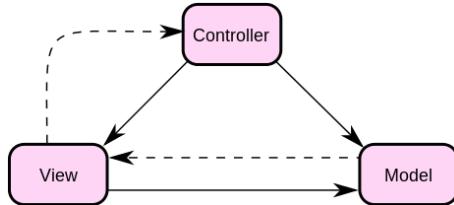


Figura 4.6: Schema dello stile architettonico MVC classico.

All'interno dell'applicazione la comunicazione fra View e il Controller avviene sulla rete, con l'implementazione originale ogni modifica dei dati comporta l'attesa della risposta del server e il ricaricamento di tutta la pagina, peggiorando l'esperienza di utilizzo e comportando un carico di lavoro più pesante al server.

Nel progetto invece l'aggiornamento dei dati viene fatto tramite i metodi JsonResult, jQuery e la programmazione asincrona, come illustrato nella sezione 4.3.1, ciò rende possibile aggiornare la pagina senza necessità di ricaricamento o di bloccare l'esecuzione degli script, risolvendo i problemi appena esposti.

4.3.1 Funzionamento di una pagina

Nella figura 4.7 viene mostrata la struttura della applicazione con le relazioni fra le tecnologie e le parti che la compongono.

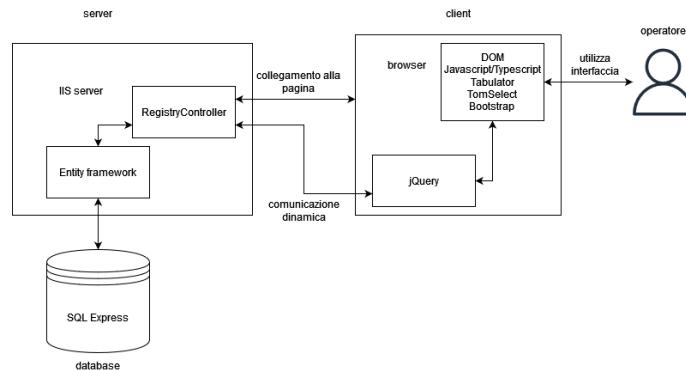


Figura 4.7: Schema della struttura di una pagina, ogni arco rappresenta la comunicazione fra le componenti alle sue estremità.

La figura illustra nel dettaglio il funzionamento dell'interazione fra il client e il server, per la gestione delle richieste è stato impiegato un solo Controller chiamato **RegistryController**, al suo interno sono presenti tutte le azioni che elaborano le richieste del client, e sono organizzate in base alla pagina per la quale vengono impiegate:

1. un'azione ActionResult, che viene invocata dal client quando si connette all'indirizzo della pagina corrispondente. Si occupa del recupero, l'elaborazione ed il passaggio dei dati, tramite la ViewBag, alla vista associata, che verrà poi tradotta in una pagina html ed inviata al client.
2. un insieme di azioni JsonResult, che vengono invocate dal client, in base alle azioni dell'utente sull'interfaccia, per comunicare dinamicamente con il server. Ciascuna delle azioni ha il compito di inserire, leggere, modificare e/o eliminare dei dati dalla base di dati ed inviare al client l'esito dell'operazione.

Quando l'utente si connette all'indirizzo della pagina interessata viene invocata l'azione che restituisce la vista corrispondente, il client mostra la pagina ricevuta utilizzando le

tecnologie illustrate sulla destra nella figura 4.7.

Se l'utente, interagendo con l'interfaccia grafica, fa una qualche azione che richiede la lettura e/o l'aggiornamento dei dati nella base di dati, il client comunica dinamicamente con il server attraverso le azioni di tipo JsonResult, utilizzando jQuery e AJAX.

In ogni caso quando il server elabora una richiesta, se necessario interagirà con la base di dati SQL attraverso le classi e i metodi forniti da EntityFramework e DataManagerInterface per modificare, inserire, eliminare o leggere i suoi dati.

Per la comunicazione dinamica è stato utilizzato il formato Json, per via la sua rappresentazione dei dati leggera, efficiente e con una grande varietà di librerie o integrazioni dirette disponibili per le tecnologie utilizzate nel progetto.

4.4 La base di dati

In questa sezione viene illustrato lo schema della base di dati illustrato in figura 4.8 e la logica che modella.

Tutte le tabelle descritte fanno parte della base di dati realizzata dall'azienda e sono state impiegate dal candidato nella realizzazione del progetto.

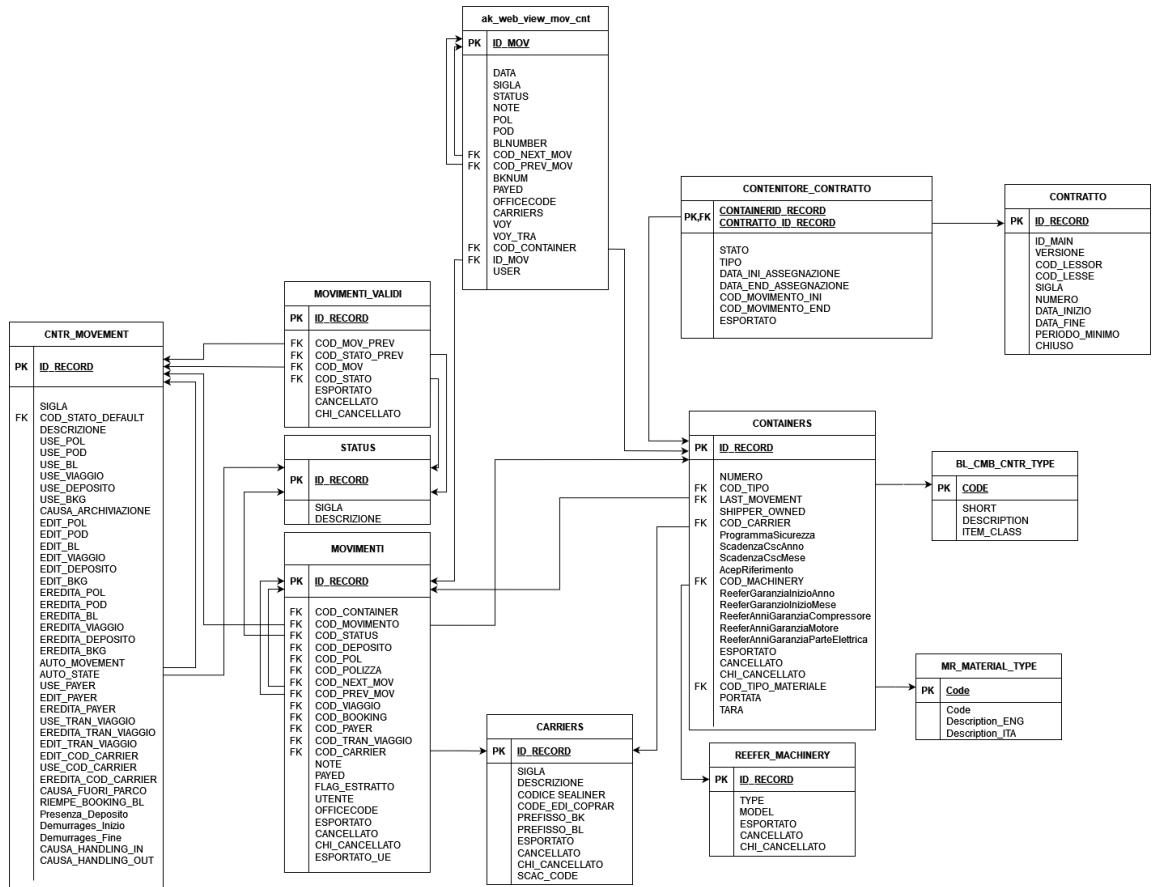


Figura 4.8: Schema delle tabelle utilizzate e delle relazioni fra di esse, **PK** indica la chiave primaria, **FK** indica la chiave esterna e la freccia indica il valore riferito dalla chiave esterna

I dati sui contenitori sono rappresentati dalla tabella **CONTAINERS**, i suoi campi possiedono le informazioni illustrate nella sezione 4.1: il numero identificativo, il tipo, il materiale, il proprietario e altri dati di tipo tecnico riguardanti il contenitore.

Le possibili caratteristiche dei contenitori sono contenute in tabelle diverse:

- **STATUS** contiene tutti gli stati possibili per i contenitori, ad esempio possono essere carichi, vuoti, danneggiati etc.
- **BL_CMB_CNTR_TYPE** indica tutte le tipologie possibili per un contenitore.
- **MR_MATERIAL_TYPE** indica tutti i materiali di cui possono essere fatti.
- **REEFER_MACHINERY** contiene dati legati ai contenitori refrigerati. Le informazioni al suo interno riguardano i macchinari utilizzati per consentire la refrigerazione.
- **CARRIERS** che rappresenta tutte le informazioni sulle aziende, che possono essere i proprietari dei contenitori, oppure i trasportatori che effettuano i movimenti.

I movimenti dei contenitori sono modellati dalle seguenti tabelle:

- La tabella **MOVIMENTI** rappresenta un movimento fatto. I campi indicano il contenitore sulla quale viene effettuato, la data in cui viene fatto, lo stato del contenitore durante il movimento, la tipologia di movimento, il trasportatore che lo effettua e altri dati di tipo tecnico che riguardano il movimento.

Ogni movimento indica un predecessore e un successore, questo fa sì che si formi una catena di movimenti che esprime lo storico dei movimenti fatti, l'ultimo movimento fatto e lo stato attuale del contenitore riferito.

Nella tabella questa dinamica viene modellata dalle chiavi esterne COD_PREV_MOV, COD_MOV, COD_STATUS e COD_MOVIMENTO, le prime due indicano il movimento precedente e quello successivo riferendo ad altre due righe della tabella MOVIMENTI, la terza indica lo stato del contenitore durante il movimento e riferisce la tabella STATUS, l'ultima indica il tipo di movimento e riferisce la tabella CNTR_MOVEMENT.

- La tabella **MOVIMENTI_VALIDI** serve a definire le sequenze dei tipi dei movimenti e stati validi all'interno della catena di movimenti.

Definisce una serie di coppie composte da un tipo di movimento e stato precedenti,

ed un tipo di movimento e stato successivi.

Quando vengono modificati, aggiunti o eliminati movimenti alla catena viene prima consultata questa tabella, se lo stato ed il tipo di un movimento e del suo successore non compaiono all'interno di questa tabella l'operazione non è possibile.

I campi della tabella riflettono questa logica riferendo la tabella CNTR_MOVEMENT con i campi COD_MOV_PREV e COD_MOV, che indicano rispettivamente il tipo di movimento successivo e quello attuale, mentre i campi COD_STATO_PREC e COD_STATO riferiscono la tabella STATUS e corrispondono lo stato successivo e quello attuale.

- **CNTR_MOVEMENT** contiene le informazioni sul tipo di movimento con una descrizione del movimento, una sigla identificativa insieme a uno stato e un movimento che vengono proposti automaticamente all'utente durante l'inserimento dei dati di un movimento.

Per alcuni campi presenti nella tabella MOVIMENTI, ad esempio il campo DEPOSITO, ne esistono altri tre in questa che indicano alcuni dettagli sul campo corrispondente:

- USE_*: il campo deve essere compilato dall'utente;
- EREDITA_*: il valore del campo viene ereditato dal movimento precedente;
- EDIT_*: l'utente può modificare il campo.

Le informazioni contenute in questi campi servono per l'inserimento o modifica dei dati di un movimento nella sezione 4.7, consentono il popolamento automatico dei campi nella sezione *movement data* dell'interfaccia e determinano se i campi possono o devono essere compilati dall'utente e/o avranno dei valori iniziali derivati dal movimento precedente.

Ai contenitori sono legati anche dei contratti di leasing e sub-leasing, i loro dati sono rappresentati con le tabelle **CONTRATTO** e **CONTENITORE_CONTRATTO**, la prima possiede i dati sui contratti, la seconda lega i contenitori con i contratti che li riguardano.

Infine i dati sui movimenti legati a un contenitore sono reperibili dalla vista **ak_web_mov_view_cnt**,

che consente il recupero dei dati, mostrati nella sezione 4.7, con una query molto più semplice da utilizzare.

4.5 Funzionamento dell'applicazione

L'applicazione è utilizzabile tramite un qualsiasi browser, e consente all'operatore di accedere all'applicazione con la schermata di login mostrata in figura 4.9.



Figura 4.9: Schermata di login dell'applicazione

Dopo il login l'applicazione mostrerà all'operatore la dashboard (figura 4.10) che fornisce l'accesso alle funzionalità offerte illustrate nella sezione 4.6.

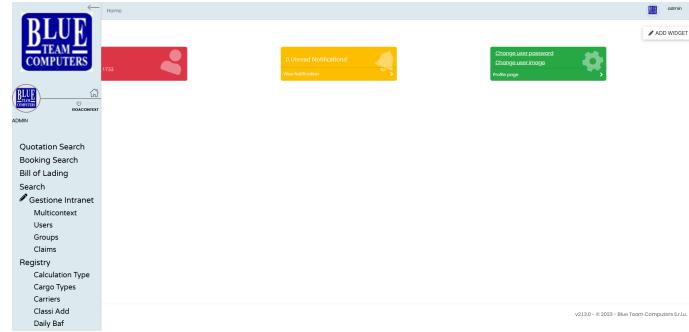


Figura 4.10: Schermata con la dashboard e il menu per accedere alle varie sezioni.

Al momento della stesura di questa relazione le sezioni realizzate dal candidato non sono ancora raggiungibili dai menù dell'applicazione.

4.6 Anagrafiche

L'inserimento, la modifica e l'eliminazione dei dati avviene tramite delle sezioni denominate *anagrafiche* insieme a una sezione che prevede la gestione dei movimenti dei contenitori.

Ogni sezione è strutturata con una pagina principale che verrà mostrata collegandosi all'url relativo, in base ai tasti premuti verranno mostrati i menù modali corrispondenti.

Code	Description	Status
ETS	Empty to Shipper	DAMAGED
RFT	Reposition from Terminal	FULL
INS	Insert in container park	EMPTY
GTO	Transfer Out	EMPTY
OTR	Out to Repair	EMPTY
IFR	In Depot from Repair	EMPTY
CAP	Containerized at Port	FULL
DAP	Decontainerized at Port	EMPTY
DFV	Discharged from Vessel	None
ETD	Empty to Depot	EMPTY

Figura 4.11: Interfaccia grafica realizzata per la sezione movimenti dei contenitori

Le pagine principali delle anagrafiche hanno una struttura simile a quella mostrata nella figura 4.11:

- una tabella che mostra i dati relativi alla sezione, in base alla sezione le righe della tabella possiedono dei button a forma di X, per eliminare, o a forma di penna, per modificare, i dati associati.
- una sezione *Search* che fornisce la possibilità di impostare dei valori per applicare dei filtri durante la ricerca di dati nella tabella.
- un tasto, a forma di lente di ingrandimento, per applicare i filtri sui dati nella tabella, impostati dalla sezione di ricerca del punto precedente.

- un bottone di inserimento per aggiungere nuovi dati riguardanti la sezione, a forma di (+), che quando viene premuto mostra un menù modale per inserire o modificare i dati della tabella.

In base al contenuto dell'anagrafica potranno variare gli elementi dell'interfaccia ma quelli appena illustrati saranno sempre presenti.

4.6.1 Anagrafica movimenti contenitori

L'anagrafica dei movimenti di contenitori riguarda la gestione dei tipi di movimenti possibili su un contenitore e consente la creazione e modifica di essi.

La pagina principale è mostrata nella figura 4.11, che contiene la tabella mostrante codice, descrizione e stato automatico. Per ogni riga è possibile modificare i suoi dati tramite il bottone presente nella prima colonna.

Tramite la sezione *Search* è possibile filtrare i dati nella tabella in base al codice, la descrizione e/o lo stato automatico.

Premendo il tasto inserimento o modifica nella tabella è possibile accedere al menù modale mostrato in figura 4.12.

	Mandatory	Editable	Inherits	
Pol	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Causes the container to be archived
Pod	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Causes the container to leave the park
Bkg	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Implies the presence of the container at the depot
B/L	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Implies an handling-in charge
Original voy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Implies an handling-out charge
Final voy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Fill the container number in the booking
Depot	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Starts the count of demurrage
Payer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> Ends the count of demurrage
Cnt user	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figura 4.12: Interfaccia del menù di inserimento o modifica di un tipo movimento per un contenitore.

Questa parte consente l'inserimento e la modifica dei dati relativi a un movimento nella tabella CNTR_MOVEMENT, inoltre fornisce la possibilità di impostare i valori relativi ad

altre informazioni sul contenitore e i campi USE_*, EDIT_* e EREDITA_* menzionati nella sezione 4.4.

4.6.2 Anagrafica sequenza movimenti

Questa sezione consente la gestione delle sequenze valide per la catena di movimenti di un contenitore.

La sua pagina principale viene mostrata nella figura 4.13, in cui la tabella mostra le combinazioni dei possibili movimenti e stati precedenti e successivi.

Previous Movement	Previous Status	Next Movement	Next Status
Discharged from Vessel	EMPTY	GATE IN	EMPTY
GATE IN	EMPTY	Empty to Shipper	BOOKED
third test	EMPTY	third test	EMPTY
Transport Out	EMPTY	OFFHIRE	EMPTY
idlo	EMPTY	Empty to Shipper	BOOKED
idlo	EMPTY	Empty to Shipper	BOOKED
GATE IN	EMPTY	idlo	EMPTY
On Hire	EMPTY	Sub Lease (OLD)	EMPTY
On Hire	EMPTY	Begin of Sub Lease	EMPTY
Begin of Sub-Lease	EMPTY	End of Sub-Lease	EMPTY

Page 1 of 13 (725 items) Page Size 10

Figura 4.13: Pagina principale per la gestione delle sequenze movimenti.

Tramite la sezione *Search* è possibile filtrare le sequenze in base al movimento e/o lo stato, precedente e successivo.

Utilizzando il tasto di inserimento o quello di modifica relativo alla sequenza interessata nella tabella viene mostrato il menù in figura 4.14, tramite esso l'operatore può modificare sequenze già presenti o inserirne di nuove.

Figura 4.14: interfaccia per l'inserimento o la modifica di una sequenza.

4.6.3 Anagrafica contenitori

Questa sezione consente l'inserimento, e la modifica dei dati dei contenitori.

La pagina principale è illustrata nella figura 4.15, la tabella mostra numero, tipologia, materiale, proprietario del contenitore e se chi spedisce il contenitore ne è anche il proprietario.

The screenshot shows a web-based application for managing containers. At the top, there is a search bar with fields for 'Container number', 'Container type', 'Container owner', 'Material type', and 'Owner shipped'. Below the search bar is a table with columns: Container number, Container type, Container owner, Material type, and Owner shipped. The table contains 10 rows of data, each with a small icon and some text. At the bottom of the table, there is a page navigation bar showing 'Page 1 of 1333 (3325 items)' and 'Page Size 10'.

Container number	Container type	Container owner	Material type	Owner shipped
GFRU6001422	40 BOX	COSARMA		x
GFRU6001438	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001443	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001459	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001464	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001470	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001485	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001490	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001504	40 REEFER HIGH CUBE	COSARMA	None	x
GFRU6001510	40 REEFER HIGH CUBE	COSARMA	None	x

Figura 4.15: Pagina principale dell'anagrafica contenitori.

Tramite l'uso della sezione *Search* è possibile applicare filtri ai dati della tabella in base al numero identificativo, il proprietario, il materiale, il tipo di contenitore e/o se è stato spedito dal proprietario.

Premendo il bottone di inserimento o il tasto di modifica del contenitore interessato, verrà mostrato il menù di inserimento o modifica di un singolo contenitore.

È possibile effettuare inserimenti multipli di contenitori tramite il menù apposito, per accedervi basta utilizzare il bottone accanto a quello per l'inserimento singolo.

Inserimento o modifica di un contenitore singolo

In figura 4.16 è mostrato il menù che l'operatore può usare per inserire o modificare un nuovo contenitore, durante l'inserimento dei dati il sistema controlla che il numero attribuito al contenitore sia conforme allo standard ISO 6346 [11] tramite l'algoritmo mostrato nella sezione 4.1.1.

Figura 4.16: Menù utilizzato per l'inserimento dei dati di un contenitore.

Nel caso in cui il contenitore sia di tipo Reefer verranno mostrati anche i campi **Type**, **Warranty start date** e **Warranty years for** che permettono di inserire ulteriori dati riguardanti l'attrezzatura per la refrigerazione.

Utilizzando il tasto **CREATE/EDIT** sarà disponibile un ulteriore menù modale, illustrato nella figura 4.17, per l'inserimento o la modifica delle attrezzature legate ad un contenitore Reefer.

Type	Model
CARRIER THINLINE	69W10-64-003
CARRIER PRIMELINE	69W10-58B-07
CARRIER THINLINE	69W10-64-001
THERMOKIND	MAGNUM
CARRIER PRIMELINE	69W10-58B-08
CARRIER THINLINE	69W10-64-020
CARRIER THINLINE	69W10-64-003
PROVAMod	PROVAMODSOLMod
PROVAMod	PROVAMODL0102
PROVAMod	PROVAMODL0202
PROVAMod	PROVAMODL0303
PROVAMod	MODELLO4
PROVAMod	MODELLO5
RMTVAR	MODELLO5

Figura 4.17: Menù per l'inserimento e la modifica delle attrezzature per un contenitore di tipo Reefer.

Inserimento multiplo di contenitori

Questo menù, in figura 4.18, consente solamente l'inserimento di più contenitori in una sola immissione. I contenitori inseriti devono avere le stesse caratteristiche, ad eccezione del numero identificativo che per ognuno di essi deve essere univoco e non già presente nella base di dati al momento dell'inserimento.

Figura 4.18: Menù utilizzato per l'inserimento dei dati di più contenitori.

Tramite la sezione *Write containers to check* vengono inseriti i numeri dei contenitori, premendo sul bottone **CHECK ABOVE FOR CONTAINER RECOGNITION** compariranno nella tabella sulla destra i numeri dei contenitori, con uno stato che indica se sono scritti correttamente, se sono già presenti o se il numero identificativo corrispondente è mal formato.

Le informazioni sui contenitori possono essere inserite tramite il form nella parte sinistra, i campi sono uguali a quelli dell'inserimento singolo e i loro valori verranno assegnati a tutti i contenitori coinvolti.

4.7 Sezione movimenti contenitori

Questa sezione è la componente principale della funzionalità per la gestione dei movimenti dei contenitori.

Consente la visualizzazione dei dati del contenitore e la visualizzazione, aggiunta, modifica e rimozione di movimenti della sua catena di movimenti. Inoltre offre anche la possibilità di generazione di report sulle informazioni relative al contenitore, come illustrato nella sezione 4.7.1.

In figura 4.19 viene mostrata la pagina principale di questa sezione in cui, tramite il campo *container number* e la tabella sottostante, è possibile cercare e selezionare un contenitore per visualizzarne i contratti di leasing e sub-leasing, i dati ed i movimenti effettuati su di esso.

The screenshot displays a web-based application interface for managing container movements. At the top, a search bar shows "container number CAIU". Below it, a table lists container details: number, type (40RC), and S.O.C. status (0). To the right, two tables show "Leasing Contracts" and "Sub Leasing Contracts" with their respective details. A central table displays a list of movements with columns for Date, Movement, Status, Deposit, POI, POD, B/L, Original voy, Final voy, Cnt user, Remarks, and user. Below this table is a "Container data to assign" section containing fields for Equipment type (40 REEFER HIGH CUBE), Carrier (COSARMA), and Security program (Acop). Another section shows "Refrigerated machinery Data" with fields for Type (CARRIER PRIMELINE - 09NT4D-56B-01B), Warranty start date (2012-02), and Warranty years for (5 years for Compressor, Engine, and Electric parts). On the right, a "movement data" panel contains dropdown menus for new movement, new status (inherited), and various movement-related fields like Final voyage, Booking, and Container Depot. It also includes a "force movement sequence" checkbox, a date field, a user field, and "SAVE" and "CANCEL" buttons. At the bottom right of the interface are three blue circular icons with symbols: a plus sign, a downward arrow, and a file.

Figura 4.19: interfaccia della sezione realizzata per il movimento dei contenitori

Selezionando un contenitore le tabelle *Leasing Contracts* e *Sub Leasing Contracts* mosteranno i contratti relativi, mentre la tabella centrale visualizzerà i movimenti fatti sul contenitore e consente la loro eliminazione o modifica.

Premendo il tasto con sopra l'icona di un file è possibile generare automaticamente il report dei dati relativi al contenitore selezionato.

L'inserimento di un nuovo movimento può essere fatto in due modi:

1. tramite il tasto + l'inserimento avviene in fondo alla catena di movimenti.
2. utilizzando il tasto alla destra di quello del punto precedente l'inserimento avviene fra due movimenti esistenti. Per utilizzare questa funzionalità è necessario avere prima selezionato un movimento dalla tabella.

L'inserimento dei dati di un nuovo movimento e/o la modifica di uno già esistente avvengono tramite la sezione *Movement data*. In base al tipo di movimento specificato tramite il campo *new status*, che fa riferimento alla tabella CNTR_MOVEMENT, i campi potranno essere modificabili, utilizzabili e/o riempiti automaticamente, come specificato alla sezione 4.4.

4.7.1 Generazione automatizzata di report

Utilizzando le librerie NPOI ed NPOI.mapper sono stati generati automaticamente dei report contenenti i dati del contenitore e i movimenti associati a esso come mostrato dal codice in figura 4.21 assieme a un esempio di report generato in figura 4.22.

I dati del contenitore vengono generati utilizzando NPOI impostando manualmente posizione e valore dei dati all'interno delle celle.

I dati dei movimenti sono stati generati tramite NPOI.mapper e il meccanismo di mappatura dei dati che esso fornisce. La mappatura viene fatta tramite una classe apposita dove impostando gli attributi per i metodi è possibile decidere sia la colonna occupata che altre impostazioni di formattazione come mostrato in figura 4.20.

```

class MovementsExcelModelMapper
{
    [Display(Name = "Date")]
    [Column(0)]
    public string Date { get; set; }

    [Column(1)]
    public string Movement { get; set; }

    [Column(2)]
    public string Status { get; set; }

    [Column(3)]
    public string Deposit { get; set; }

    [Column(4)]
    public string Pol { get; set; }

    [Column(5)]
    public string Pod { get; set; }

    [Column(6)]
    public string Bkg { get; set; }

    [Column(7)]
    public string BL { get; set; }

    [Column(8)]
    public string OriginalVoyage { get; set; }

    [Column(9)]
    public string FinalVoyage { get; set; }

    [Column(10)]
    public string CntUser { get; set; }

    [Column(11)]
    public string Remarks { get; set; }

    [Column(12)]
    public string User { get; set; }
}

```

Codice 4.20: Definizione della classe utilizzata per la mappatura dei dati sul foglio di calcolo.

Tramite il metodo **Put** dell’oggetto **Mapper** è possibile inserire automaticamente i dati di più oggetti, utilizzando una classe mapper ad hoc, senza dover gestire manualmente il loro posizionamento. In figura 4.21 viene mostrata l’azione responsabile della generazione dei report che dato l’identificatore di un contenitore, utilizzando il metodo Put ed il Mapper, crea e restituisce al client il documento corrispondente, con tutti i suoi dati come mostrato in figura 4.22.

```

public FileContentResult GetMovementsTableExportedData(string containerId) {
    CONTAINER container = _db.CONTAINERS.Single(c=> c.ID_RECORD == containerId) ;
    IWorkbook wb = new XSSFWorkbook();
    var sheet=(XSSFSheet)wb.CreateSheet("report dati");

    List<List<string>> excelContainerData = new List<List<string>>
    {
        new List<string>{
            "Estrazione del:",
            DateTime.Now.ToString("dd/MM/yyyy HH:mm")
        },
        new List<string>{
            "Container:",
            container.NOME
        }
    };
}
```

```

        container.NUMERO
    },
    new List<string>{
        "Container type",
        container.COD_TIPO==" " || container.COD_TIPO==null? "":  

        _db.BL_CMB_CNTR_TYPE.Single(c=>c.CODE == container.COD_TIPO)
            .DESCRIPTION
    },
    new List<string>{
        "Material type",
        container.COD_TIPO_MATERIALE == " " || container.COD_TIPO_MATERIALE==null?  

        " ":
        _db.MR_MATERIAL_TYPE.Single(mt=> mt.Code == container.COD_TIPO_MATERIALE)
            .Description_ENG
    },
    new List<string>{
        "Container owner",
        container.COD_CARRIER == " " || container.COD_CARRIER == null ?
            " ":
            _db.CARRIERS.Single(c=> c.ID_RECORD == container.COD_CARRIER )
                .DESCRIZIONE
    },
    new List<string>{
        "Tare",
        container.TARA.ToString(),
        "Max.Gr.Weight.",
        container.PORTATA.ToString(),
        "Kgs"
    }
};

if (container.SHIPPER OWNED != 0) {
    excelContainerData.Add(new List<string> {
        "Shipper owned"
    });
}
if (container.ProgrammaSicurezza == 1) {
    excelContainerData.Add(new List<string>
    {
        "ACEP reference",
        container.AcepRiferimento
    });
} else if (container.ProgrammaSicurezza == 2) {
    excelContainerData.Add(new List<string>
    {
        "CSC expiring date",
        container.ScadenzaCscMese+"/"+container.ScadenzaCscAnno
    });
}
int row = 0;

foreach (var rowData in excelContainerData)
{
    int col = 0;
    IRow docRow= sheet.CreateRow(row);
    row++;
    foreach (var cellData in rowData)
    {
        ICell cell=docRow.CreateCell(col);
        cell.SetCellValue(cellData);
        col++;
    }
}
}

```

```

MemoryOutputStream ms = new MemoryOutputStream();

Mapper mapper=new Mapper(wb);
mapper.FirstRowIndex = 7;
mapper.Put(_db.ak_web_view_mov_cnt.Where(m => m.COD_CONTAINER == container.ID_RECORD)
    .Select( movement => new MovementsExcelModelMapper
    {
        Date = movement.DATA.ToString(),
        Movement = movement.SIGLA,
        Status = movement.STATUS,
        Deposit = movement.DEPOT,
        Pol = movement.POL,
        Pod = movement.POD,
        Bkg = movement.BKNUM,
        BL = movement.BLNUMBER,
        OriginalVoyage = movement.VOY,
        FinalVoyage = movement.VOY_TRA,
        CntUser = movement.CARRIERS,
        Remarks = movement.NOTE,
        User = movement.USER,
    })
    .ToList(),
    "report dati"
);
for (int i = 0; i < 13; i++)
{
    sheet.AutoSizeColumn(i);
}
mapper.Save(ms);
wb.Close();
return File(ms.ToArray(), "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet","report-
contenitore-"+container.NUMERO);
}

```

Codice 4.21: codice utilizzato per generare report sui contenitori e i loro movimenti.

A	B	C	D	E	F	G	H	I	J	K	L	M
1 Estrazione del:	04/09/2023 15:42											
2 Container:	TCLU1214237											
3 Container type	40 REEFER HIGH CUBE											
4 Material type												
5 Container owner												
6 Tare	4640	Max.Gr.Weight.	35000		Kgs							
7 ACEP reference												
8 Date	Movement	Status	Deposit	Pol	Pod	Bkg	BL	OriginalVoyage	FinalVoyage	Remarks		
9 apr 26 2012 12:00AM	BSL	E	BUSAN DEPOT									
10 apr 26 2012 12:00AM	OHL	E	BUSAN DEPOT									
11 mag 2 2019 12:00AM	OFF	E	DAVAO DEPOT									
12 mag 2 2019 12:00AM	ESL	E	DAVAO DEPOT									

Figura 4.22: Esempio di report generato dalla sezione movimenti contenitori.

4.8 Organizzazione del progetto

Il progetto è stato organizzato grazie ai meccanismi forniti da Visual Studio.

È composto da due soluzioni: la prima comprende il progetto con le classi generate dallo schema del DB, la seconda comprende i progetti con tutti i file dell'applicazione.

È stato utilizzato il controller MVC **RegistryController** per realizzare il Controller, per ogni pagina realizzata è stata seguita la seguente struttura che è anche riportata in figura 4.23:

- un file typescript per l'implementazione della componente **View** della pagina che si trova nella cartella AkitWeb/Areas/Scripts/Akit/Registry/<nomefile>.ts.
- vari file typescript che contengono le definizioni per le classi impiegate negli script, si trovano al percorso AkitWeb/Areas/Scripts/Akit/Registry/Model/<nomefile>.ts.
- un file .cshtml con percorso AkitWeb/Areas/Intranet/Views/Registry/<nomefile>.cshtml utilizzato per la creazione della pagina.
- se la pagina fa uso di modali allora si hanno altri file .cshtml con percorso AkitWeb/Areas/Intranet/Views/Registry/Modal/<nomefile>.cshtml.
- varie classi utilizzate per rappresentare dati non presenti nella base di dati.

Oltre ai file sopra citati si aggiunge la classe **ContainerUtils** che fornisce metodi di supporto qualora ci fosse bisogno, ad esempio funzioni da utilizzare in più metodi diversi o elaborazione di dati in vari formati, che si trova al percorso AkitWebLib/Utils/ContainersUtils.cs.

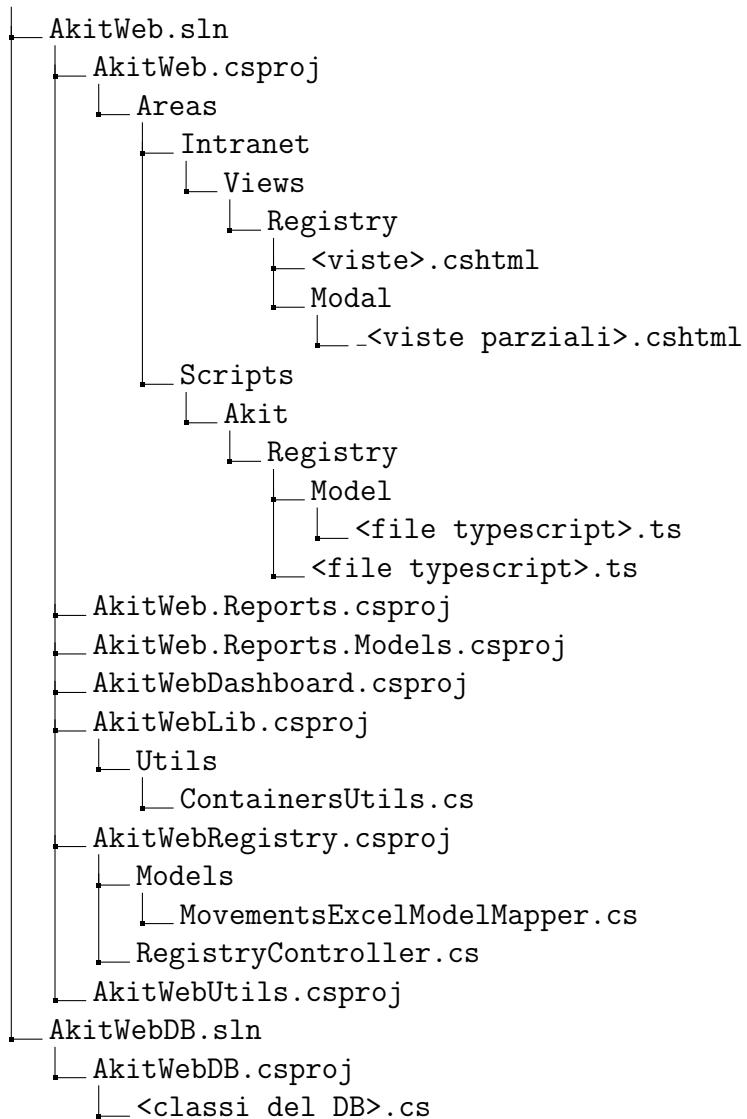


Figura 4.23: Struttura del progetto all'interno di Visual Studio. Sono riportati solo i file e le cartelle rilevanti per l'implementazione delle funzionalità prevista dal progetto.

4.9 CI/CD

Come già menzionato nella sezione 2.6 GitLab mette a disposizione funzionalità per la CI/CD, nel caso dell'applicazione sono stati definiti due stage:

- build: compila l'applicazione.
- deploy: crea un pacchetto che può essere successivamente utilizzato su un server privato o su servizi cloud appositi per eseguire e rendere disponibile l'applicazione.

```
stages:  
  - build  
  - deploy  
  
variables:  
  NPM_ROOT: AkitWeb\Scripts  
  
before_script:  
  - cd $NPM_ROOT  
  - npm ci  
  - cd ..\..  
  
build_job:  
  stage: build  
  script:  
    - 'msbuild -p:RestorePackagesConfig=true -t:restore AkitWeb.sln'  
    - 'msbuild AkitWeb.sln /t:Build /p:Configuration=Debug'  
  
deploy_job:  
  stage: deploy  
  needs:  
    - build_job  
  script:  
    - msbuild -p:RestorePackagesConfig=true -t:restore AkitWeb.sln  
    - msbuild /p:DeployOnBuild=true /p:PublishProfile=FolderProfile  
  artifacts:  
    paths:  
      - AkitWeb\bin\Release\Published
```

Figura 4.24: File di configurazione utilizzato per pipeline del progetto.

In figura 4.24 viene mostrato il file di configurazione utilizzato, al suo interno sono definiti due job corrispondenti agli stage che utilizzano msbuild per compilare e pacchettizzare l'applicazione. Il job che si occupa del deploy ha specificato la necessità che la build sia terminata imponendo una sequenza nell'esecuzione dei job, nel caso la build fallisca il deploy non può avvenire.

5. Conclusioni

Durante l’esperienza durata circa 300 ore il candidato si è misurato con il ciclo di sviluppo di funzionalità per un’applicazione di tipo gestionale in ambito marittimo all’interno di un contesto aziendale.

Il candidato ha acquisito competenze sulle tecnologie sugli strumenti utilizzati dall’azienda, oltre a conoscenze teoriche di base riguardanti la gestione delle spedizioni in ambito marittimo.

Il progetto si è concretizzato con lo sviluppo, da parte del candidato, delle sezioni dell’applicazione “Agency Kit Web” responsabili del tracciamento dello stato dei contenitori e dei loro movimenti.

Per ogni sezione, il candidato ha progettato le funzionalità all’interno di client e server, occupandosi della realizzazione delle seguenti parti:

- interazione fra server e base di dati;
- progettazione e realizzazione interfaccia grafica;
- interazione fra client e server;

Durante l’esperienza il candidato ha affinato le capacità di lavoro di squadra coordinandosi con gli altri dipendenti dell’azienda.

Nel corso del tirocinio il candidato ha messo in pratica le nozioni apprese in diversi corsi presenti nel corso di laurea, come Reti e Laboratorio 3, Basi di dati, Paradigmi di Programmazione, Ingegneria del Software e Laboratorio 1.

Grazie alle nozioni apprese dai corsi il candidato è stato in grado di apprendere senza difficoltà nozioni di sviluppo frontend, backend per un’applicazione web, insieme alla generazione automatizzata di report e l’utilizzo di ORM per interagire con le basi di dati.

Bibliografia

- [1] Ajax. <https://it.wikipedia.org/wiki/AJAX>.
- [2] Blue team computers. <https://www.blueteamcomputers.it/>.
- [3] Boostrap 4.0. <https://getbootstrap.com/>.
- [4] C. <https://learn.microsoft.com/it-it/dotnet/csharp/>.
- [5] Ci/cd. <https://en.wikipedia.org/wiki/CI/CD>.
- [6] Common language runtime. https://it.wikipedia.org/wiki/Common_Language_Runtime.
- [7] Entity framework. <https://learn.microsoft.com/it-it/ef/>.
- [8] Git. [https://it.wikipedia.org/wiki/Git_\(software\)](https://it.wikipedia.org/wiki/Git_(software)).
- [9] Gitlab. <https://about.gitlab.com/>.
- [10] Iisserver. <https://learn.microsoft.com/it-it/iis/get-started/introduction-to-iis/introduction-to-iis-architecture>.
- [11] Iso 6346. https://en.wikipedia.org/wiki/ISO_6346.
- [12] javascript. <https://it.wikipedia.org/wiki/JavaScript>.
- [13] jquery. <https://jquery.com/>.
- [14] Json.net. <https://www.newtonsoft.com/json>.
- [15] Linq. <https://learn.microsoft.com/it-it/dotnet/csharp/linq/>.
- [16] Model view controller. <https://it.wikipedia.org/wiki/Model-view-controller#Componenti>.

- [17] modello relazionale. https://it.wikipedia.org/wiki/Modello_relazionale.
- [18] Msbuild. <https://learn.microsoft.com/it-it/visualstudio/msbuild/msbuild?view=vs-2022>.
- [19] Npoi. <https://github.com/dotnetcore/NPOI>.
- [20] Npoi.mapper. <https://github.com/donnytian/Npoi.Mapper>.
- [21] Nuget. <https://en.wikipedia.org/wiki/NuGet>.
- [22] Object relational mapping. https://it.wikipedia.org/wiki/Object-relational_mapping.
- [23] Open project. <https://www.openproject.org/>.
- [24] Sql server 2016. <https://www.microsoft.com/it-it/sql-server/sql-server-2016>.
- [25] Sql server manager. <https://learn.microsoft.com/it-it/sql/sql-server/?view=sql-server-ver16>.
- [26] Structured query language (SQL). https://it.wikipedia.org/wiki/Structured_Query_Language.
- [27] Sweetalert2. <https://sweetalert2.github.io/>.
- [28] tabulator. <https://tabulator.info/>.
- [29] Tomselect. <https://tom-select.js.org/>.
- [30] typescript. <https://www.typescriptlang.org/>.
- [31] Visual studio 2022. <https://visualstudio.microsoft.com/it/>.