

NOLEGGI

| | |
|-----------------------------|------------------|
| Titolo del progetto | Noleggi |
| Alunno/a | Alessandro Perri |
| Classe | Informatica 4BB |
| Anno scolastico | 2023/2024 |
| Docente responsabile | Ugo Bernasconi |

1 Indice

| | | |
|-------|--|----|
| 1 | Indice..... | 2 |
| 2 | Introduzione..... | 4 |
| 2.1 | Informazioni sul progetto..... | 4 |
| 2.2 | Abstract..... | 4 |
| 2.3 | Scopo..... | 5 |
| 3 | Analisi..... | 6 |
| 3.1 | Analisi del dominio..... | 6 |
| 3.2 | Analisi e specifica dei requisiti..... | 6 |
| 3.3 | Use case..... | 10 |
| 3.4 | Pianificazione..... | 11 |
| 3.5 | Analisi dei mezzi..... | 12 |
| 3.5.1 | Software..... | 12 |
| 3.5.2 | Hardware..... | 12 |
| 4 | Progettazione..... | 13 |
| 4.1 | Design dei dati..... | 13 |
| 4.2 | Design delle interfacce..... | 13 |
| 4.2.1 | Pagina home..... | 14 |
| 4.2.2 | Pagina clienti..... | 15 |
| 4.2.3 | Pagina risorse..... | 16 |
| 4.2.4 | Pagina pianificazione..... | 17 |
| 4.2.5 | Pagina stampa..... | 18 |
| 4.2.6 | Pagina impostazioni..... | 19 |
| 4.3 | Design procedurale..... | 20 |
| 5 | Implementazione..... | 22 |
| 5.1 | Creazione progetto core..... | 22 |
| 5.2 | Pacchetti NuGet..... | 23 |
| 5.3 | Struttura cartelle core..... | 23 |
| 5.3.1 | Models..... | 24 |
| 5.3.2 | Configurations..... | 30 |
| 5.3.3 | Services..... | 34 |
| 5.4 | Creazione database..... | 41 |
| 5.5 | Creazione progetto blazor..... | 42 |
| 5.6 | Struttura cartelle blazor..... | 43 |
| 5.6.1 | Libreria Radzen..... | 43 |
| 5.6.2 | Classe Program..... | 44 |
| 5.6.3 | appsettings.json..... | 45 |
| 5.6.4 | Components..... | 45 |
| 5.6.5 | Pages..... | 46 |
| 5.6.6 | Shared..... | 65 |
| 5.7 | Design finale delle interfacce..... | 66 |
| 5.7.1 | Pagina home..... | 66 |
| 5.7.2 | Pagina clienti..... | 66 |
| 5.7.3 | Pagina risorse..... | 67 |

| | | |
|-------|--------------------------------------|----|
| 5.7.4 | Pagina noleggi..... | 67 |
| 5.7.5 | Pagina pianificazione | 68 |
| 6 | Test..... | 69 |
| 6.1 | Protocollo di test..... | 69 |
| 6.2 | Risultati test | 74 |
| 6.3 | Risultati UnitTest | 75 |
| 6.4 | Mancanze/limitazioni conosciute..... | 75 |
| 7 | Consuntivo | 76 |
| 8 | Conclusioni..... | 77 |
| 8.1 | Sviluppi futuri..... | 77 |
| 8.2 | Considerazioni personali..... | 77 |
| 9 | Bibliografia..... | 78 |
| 9.1 | Sitografia | 78 |
| 10 | Glossario | 79 |
| 11 | Indice delle figure | 80 |
| 12 | Allegati..... | 81 |

2 Introduzione

2.1 Informazioni sul progetto

- Allievo: Alessandro Perri
 - Supervisore: Ugo Bernasconi
 - Scuola: Arti e Mestieri Trevano
 - Sezione: Informatica
 - Classe: I4BB
-
- Data di inizio: 30.08.2023
 - Data di consegna: 07.12.2023

2.2 Abstract

Situazione iniziale

In questo progetto mi è stato chiesto di realizzare un'applicazione Web che simuli il noleggio di una risorsa qualsiasi. È possibile inserire i clienti, inserire le risorse, creare un noleggio e visualizzare il calendario. L'applicativo deve anche tener traccia delle risorse messe a disposizione del cliente.

Approccio

Per questo progetto, saranno necessarie buone conoscenze nella programmazione web, poiché per la creazione di quest'applicazione Web si utilizzerà il framework Blazer che utilizza C# e HTML. Per migliorare la grafica sarà necessario saper utilizzare la libreria Radzen.

Risultati

Gli obiettivi di questo progetto sono stati raggiunti, l'applicativo è funzionante e rispecchia buona parte dei requisiti. In conclusione, si può utilizzare senza avere problemi di funzionamento.

2.3 Scopo

Scopi

- Didattici:
 - Saper creare e rispettare una progettazione.
 - Saper documentare il lavoro.
 - Saper creare i diari.
 - Saper creare Use-case.
- Operativi:
 - Saper creare una GUI
 - Saper utilizzare il framework Blazor
 - Saper programmare in C# e HTML
 - Saper utilizzare la libreria Radzen

3 Analisi

3.1 Analisi del dominio

Questo progetto consiste nel creare un'applicazione Web che permette ad un utente di noleggiare delle risorse. Quest'app potrà essere utilizzata quando un cliente avrà bisogno di noleggiare per un periodo di tempo, un'oggetto disponibile tra le risorse messe a disposizione.

3.2 Analisi e specifica dei requisiti

| ID: REQ-01 | |
|-----------------|--|
| Nome | Poter accedere alla pagina tramite browser |
| Priorità | 1 |
| Versione | 1.0 |

| ID: REQ-02 | |
|-----------------|-----------------------------------|
| Nome | Visualizzare la pagina principale |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Visualizzare pulsante home |
| 002 | Visualizzare pulsante clienti |
| 003 | Visualizzare pulsante risorse |
| 004 | Visualizzare pulsante noleggio |
| 005 | Visualizzare pulsante calendario |

| ID: REQ-03 | |
|-----------------|--|
| Nome | Poter accedere alla sezione clienti |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Poter inserire i dati anagrafici dei clienti: <ul style="list-style-type: none"> • Nome • Cognome • Data di nascita • Indirizzo • Numero di telefono • Email |
| 002 | Visualizzare i clienti inseriti |

| ID: REQ-04 | |
|-----------------|---|
| Nome | Poter accedere alla sezione risorse |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Poter inserire una risorsa con i suoi dati relativi: <ul style="list-style-type: none"> • Nome • Categoria • Periodo • Prezzo |

| ID: REQ-05 | |
|-----------------|---|
| Nome | Poter accedere alla sezione noleggio |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Poter creare un nuovo noleggio con i suoi dati: <ul style="list-style-type: none"> • Cliente • Risorsa • Data di ritiro • Data di fine noleggio • Data effettiva di fine noleggio (In caso di ritardo nella consegna) • Costo effettivo (Di base è uguale al costo della risorsa, in caso di sconto cambia) • Costo totale (Calcolo automatico del costo in base al periodo della risorsa) |

| ID: REQ-06 | |
|-----------------|--|
| Nome | Poter accedere alla sezione calendario |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Visualizzare il calendario con i vari noleggi al suo interno |
| 002 | Poter creare un nuovo noleggio quando premo una casella vuota |
| 003 | Poter modificare un noleggio quando premo su uno già esistente |

| ID: REQ-07 | |
|-----------------|---|
| Nome | Pulsante stampa |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Visualizzare il pulsante all'interno delle sezioni cliente e risorse |
| 002 | Nella sezione cliente: quando premo deve stampare il contenuto della pagina |
| 003 | Nella sezione risorse: quando premo deve stampare il contenuto della pagina |

| ID: REQ-08 | |
|-----------------|-------------------------------|
| Nome | Macchina virtuale funzionante |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Installazione di IIS |

3.3 Use case

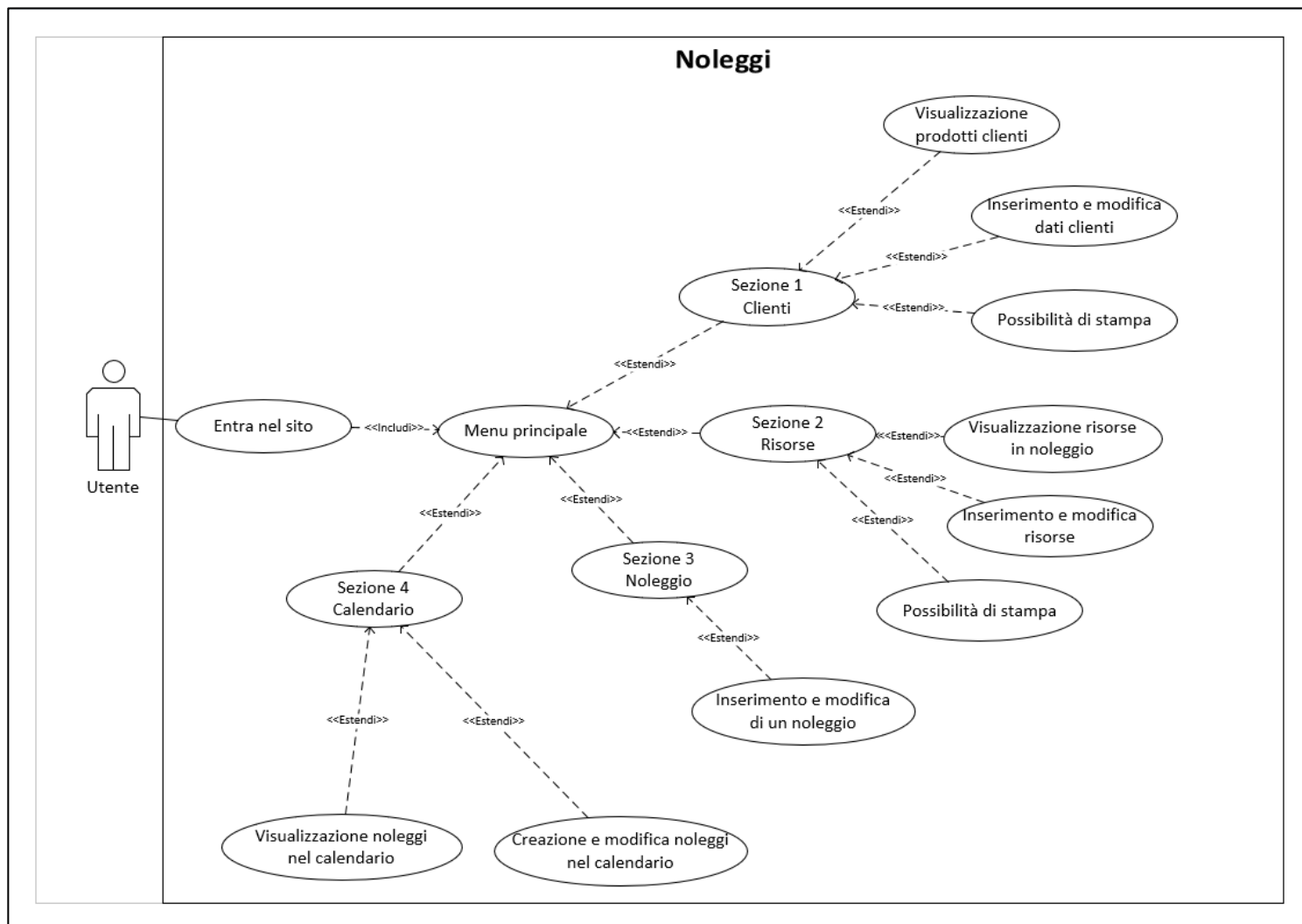


Figura 1: Use-case

3.4 Pianificazione

Di seguito c'è la pianificazione preventiva, nel Gantt preventivo è stimato il tempo necessario per completare le task del progetto.

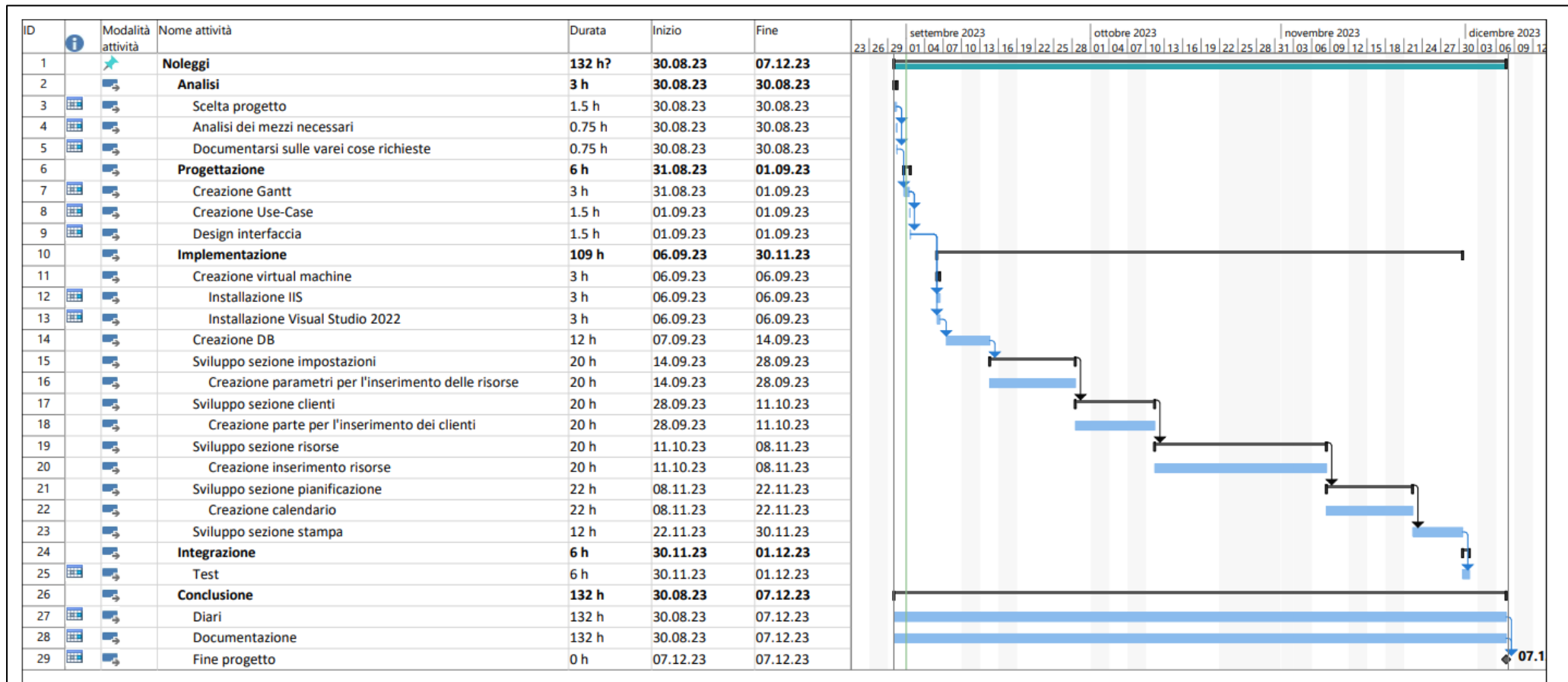


Figura 2: Gantt Preventivo

3.5 Analisi dei mezzi

Per la realizzazione di questo progetto è stato utilizzato un PC scolastico con Visual Studio installato.

3.5.1 Software

Per realizzare questo progetto sono stati utilizzati i seguenti software:

- Microsoft Word
- Microsoft Project
- Microsoft Visio
- Visual Studio 2022 – 17.6.2
- Framework .NET – 6.0
- Libreria Radzen – 4.15.10

3.5.2 Hardware

Un computer con:

- CPU Intel Xeon E3-1240
- RAM 16 GB

4 Progettazione

4.1 Design dei dati

Questa è la struttura dello schema E-R per poi successivamente realizzare il database.

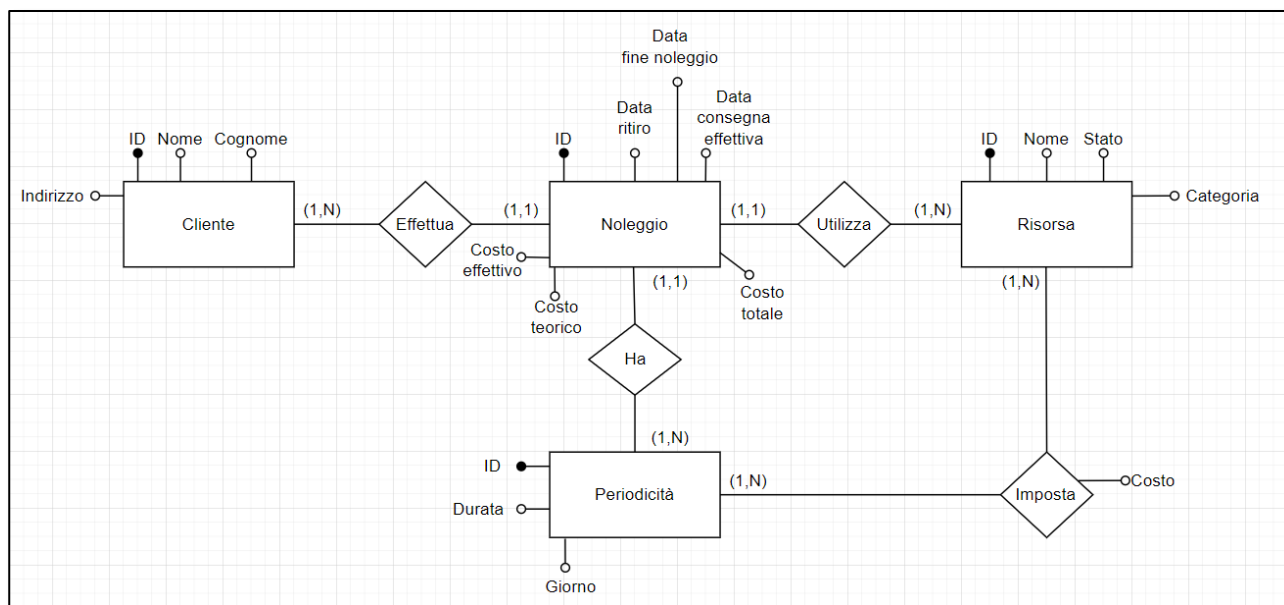


Figura 3: Schema E-R

4.2 Design delle interfacce

La struttura dell'applicativo è divisa nelle seguenti sezioni:

- Pagina home
- Pagina clienti
- Pagina risorse
- Pagina pianificazione
- Pagina stampa
- Pagina impostazioni

4.2.1 Pagina home

La home è la prima pagina che viene visualizzata quando si avvia l'applicativo.

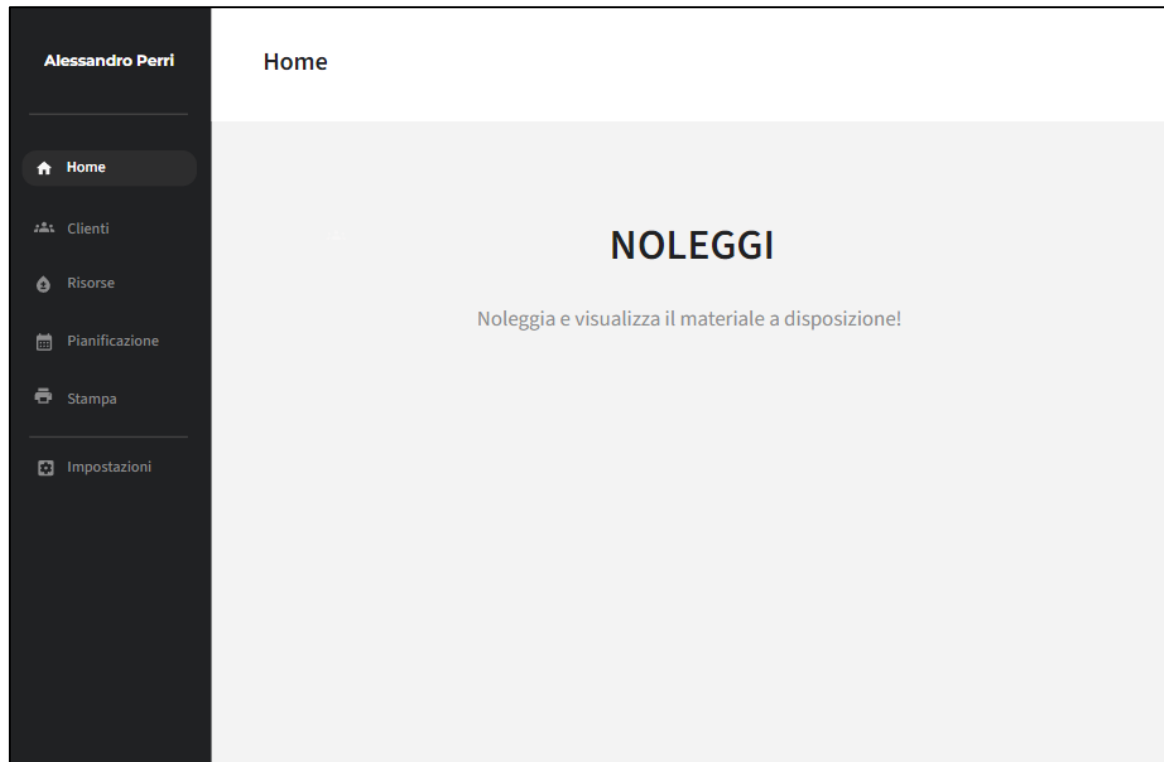


Figura 4: Pagina home

4.2.2 Pagina clienti

In questa pagina sarà possibile aggiungere dei clienti e si potrà filtrare una persona per vedere le risorse che ha noleggiato.

Alessandro Perri

Clienti

+ Aggiungi cliente

| Nome e cognome | Data di nascita | Indirizzo | Luogo | ID |
|------------------|-----------------|----------------|--------|----|
| Alessandro Perri | 01.06.2005 | Via Boschina 2 | Bedano | 1 |

Alessandro Perri ▼

| Nome | Stato | Categoria | Data iniziale | Data finale | ID |
|---------|-------------|-----------|---------------|-------------|----|
| Macbook | In noleggio | Computer | 30.08.2023 | 02.09.2023 | 1 |

Figura 5: Pagina clienti

4.2.3 Pagina risorse

In questa sezione è possibile inserire una nuova risorsa, visualizzare quelle già presenti e filtrare per vedere se la risorsa è in noleggio oppure chi l'ha noleggiata.

Alessandro Perri

- Home
- Clienti
- Risorse**
- Pianificazione
- Stampa
- Impostazioni

Risorse

+ Aggiungi risorsa

| Nome | Stato | Categoria | Data iniziale | Data finale | ID |
|---------|-------------|-----------|---------------|-------------|----|
| Macbook | In noleggio | Computer | 30.08.2023 | 02.09.2023 | 1 |

Macbook ▼

| Nome e cognome | Risorsa | Stato | ID |
|------------------|---------|-------------|----|
| Alessandro Perri | Macbook | In noleggio | 1 |

Figura 6: Pagina risorse

4.2.4 Pagina pianificazione

Nella sezione pianificazione vengono mostrati i noleggi all'interno del calendario.

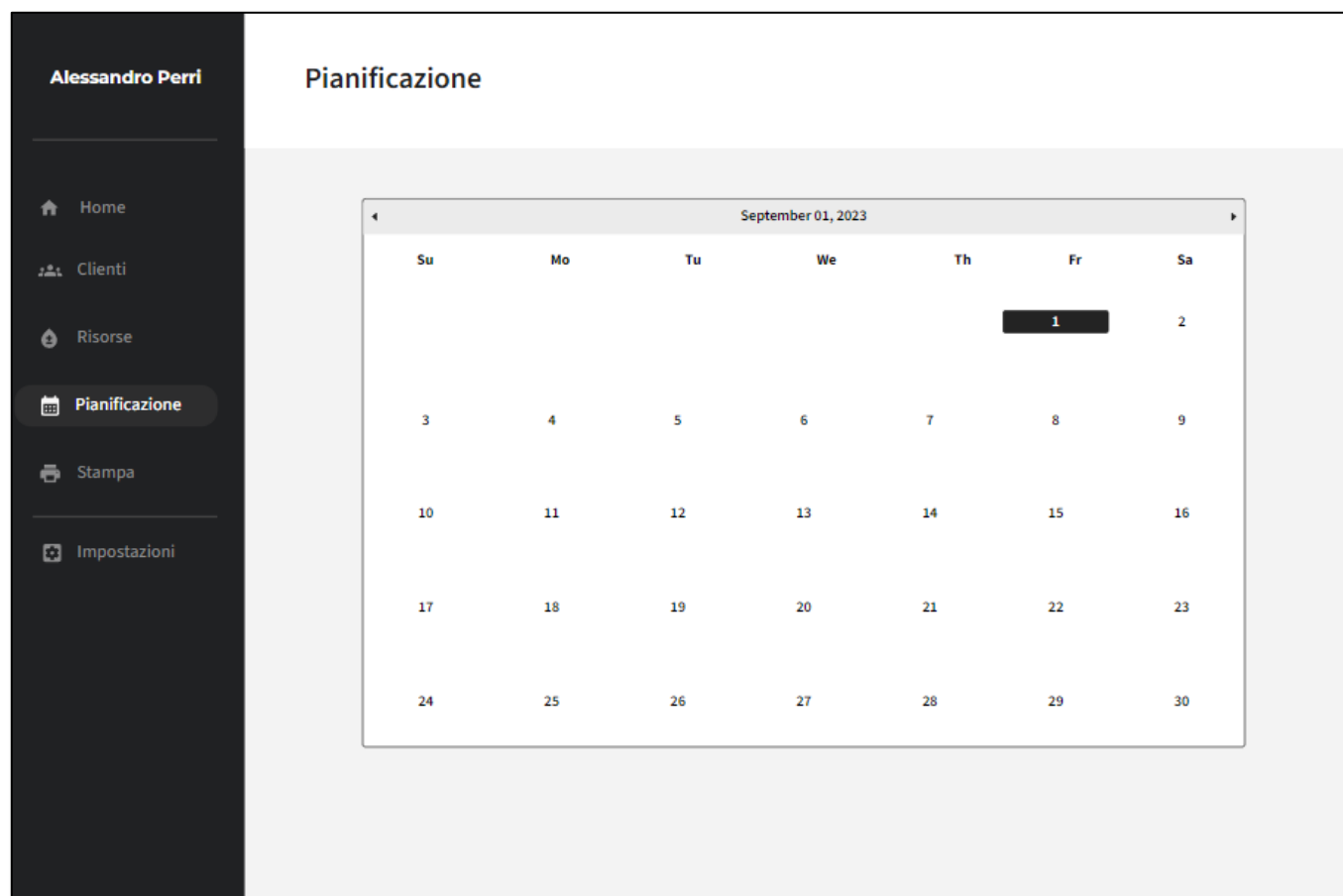


Figura 7: Pagina pianificazione

4.2.5 Pagina stampa

Nella pagina stampa verranno visualizzati i dati dei clienti e delle risorse.

The screenshot shows the 'Stampa' (Print) page in the SAMT system. The page has a dark sidebar on the left with the user's name 'Alessandro Perri' and navigation links: Home, Clienti, Risorse, Pianificazione, **Stampa** (highlighted), and Impostazioni. The main content area is titled 'Stampa' and contains two tables.

The first table is titled 'Alessandro Perri' and displays the following data:

| Nome | Stato | Categoria | Data iniziale | Data finale | ID |
|---------|-------------|-----------|---------------|-------------|----|
| Macbook | In noleggio | Computer | 30.08.2023 | 02.09.2023 | 1 |

The second table is titled 'Macbook' and displays the following data:

| Nome e cognome | Risorsa | Stato | ID |
|------------------|---------|------------|----|
| Alessandro Perri | Macbook | Consegnato | 1 |

Figura 8: Pagina stampa

4.2.6 Pagina impostazioni

Nelle impostazioni si potranno settare i vari parametri che definiscono una risorsa.

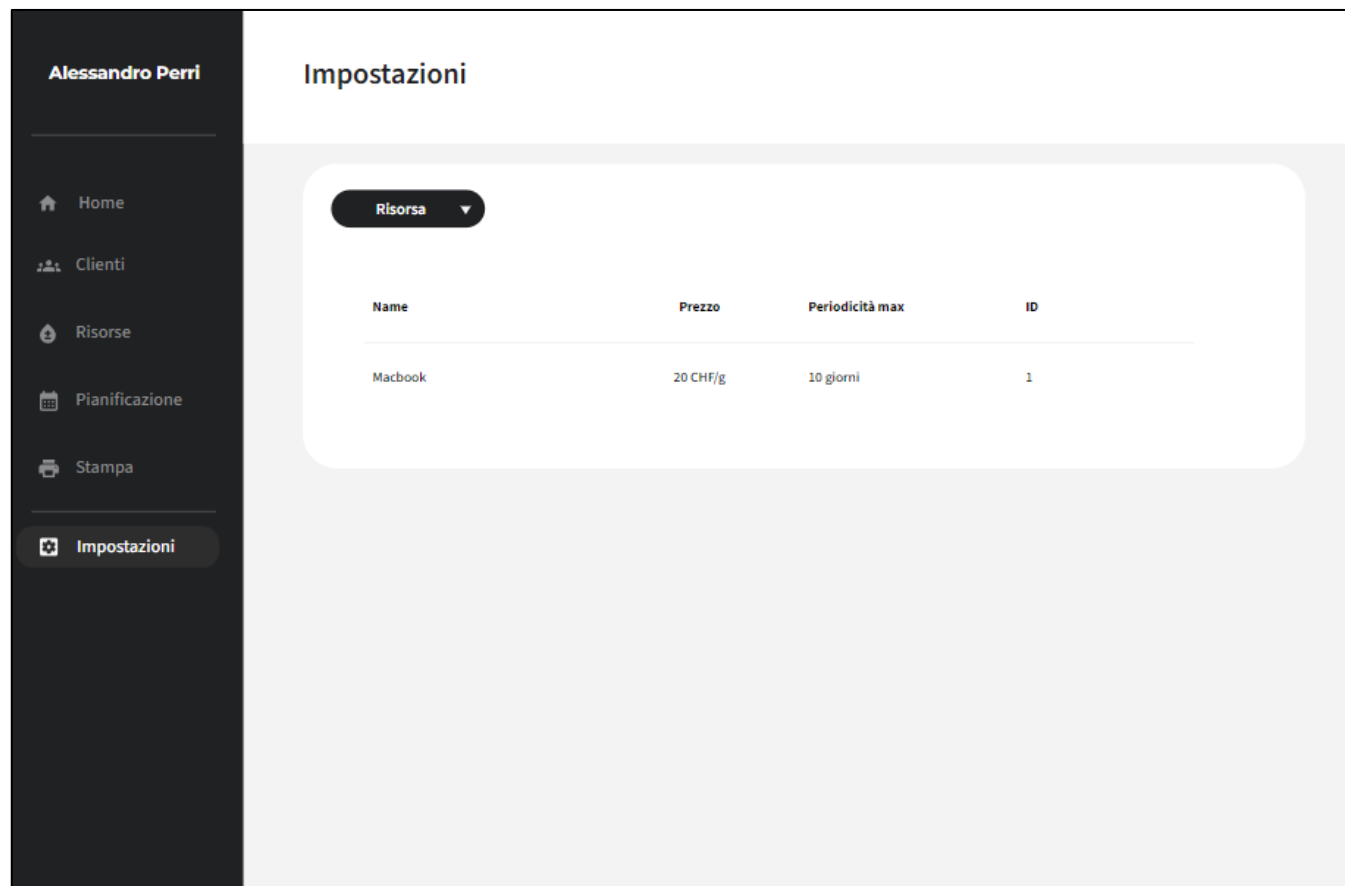


Figura 9: Pagina impostazioni

4.3 Design procedurale

Questo è il diagramma delle classi base create per la realizzazione del database.

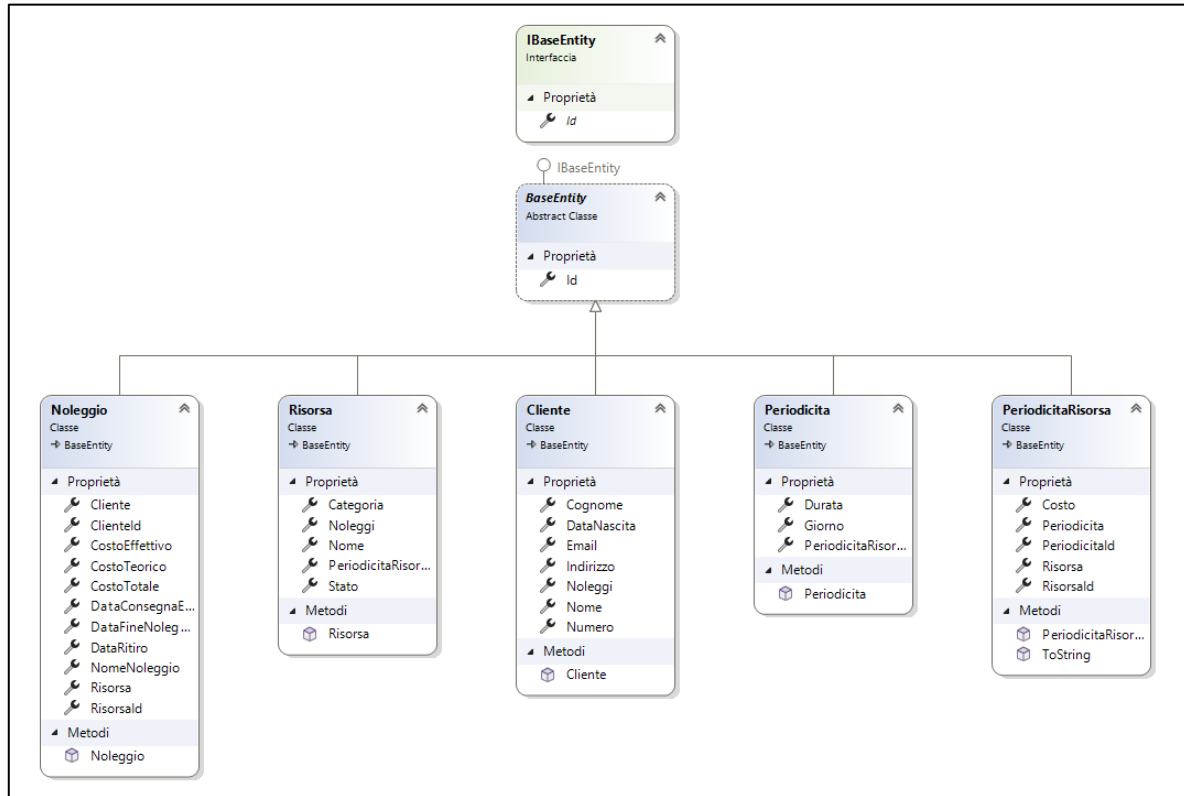


Figura 10: Diagramma delle classi

Questo è il diagramma delle interfacce, per poter mandare i dati al progetto blazor.

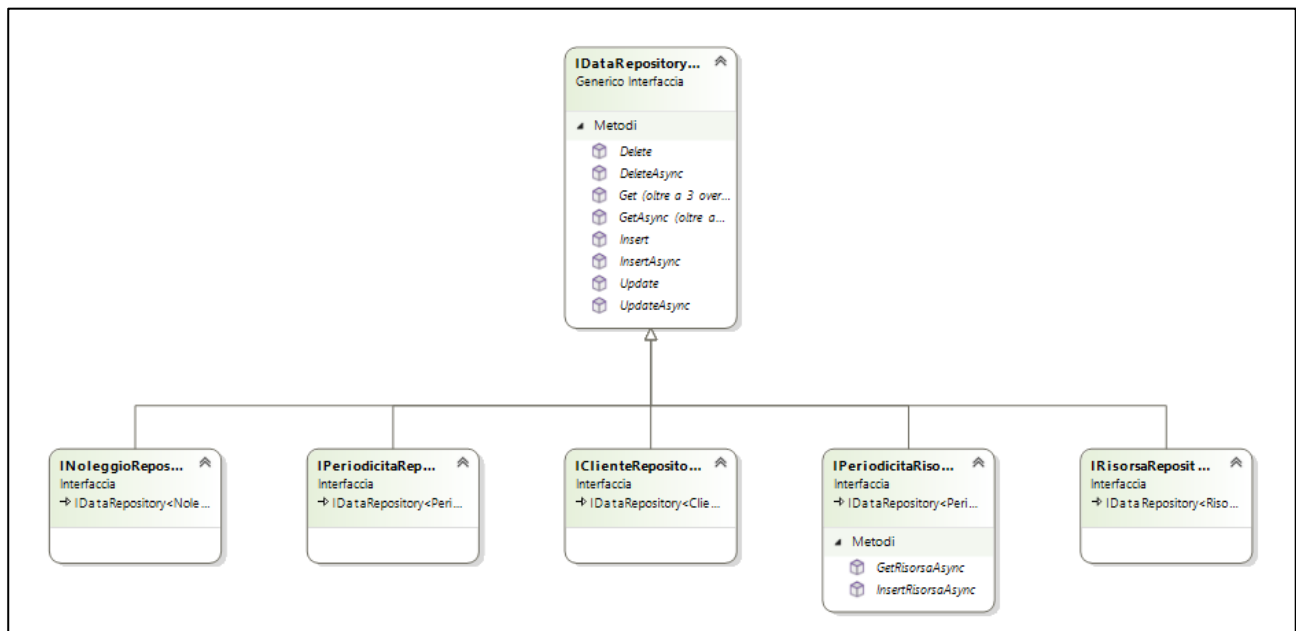


Figura 11: Diagramma delle interfacce

Successivamente troviamo il diagramma delle classi che fanno riferimento alle interfacce.



Figura 12: Diagramma delle classi repository

5 Implementazione

5.1 Creazione progetto core

Per prima cosa bisognerà creare un nuovo progetto “Libreria di classi”. Questo tipo di progetto avrà al suo interno tutti i file base necessari per la rappresentazione e la popolazione del database.

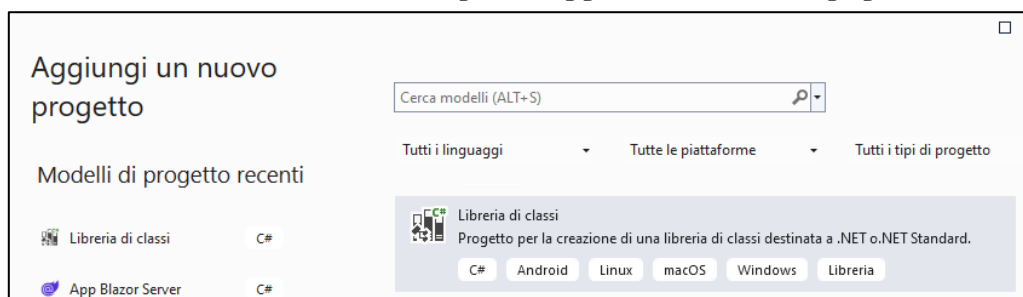


Figura 13: Creazione progetto core

Successivamente, inserire il nome del progetto e il percorso.

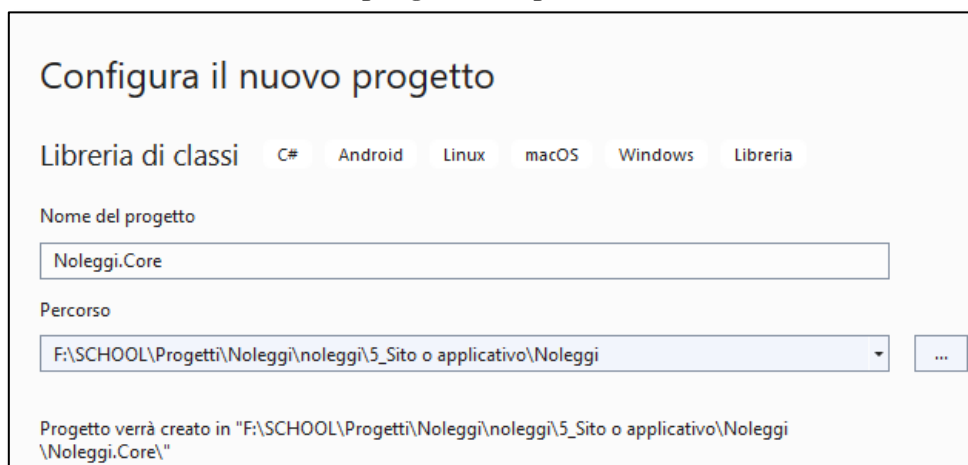


Figura 14: Configurazione progetto core

Infine troveremo il framework utilizzato, in questo caso .NET 6.0



Figura 15: Informazioni progetto core

5.2 Pacchetti NuGet

Una volta creato il progetto, bisognerà installare dei pacchetti NuGet.

Io ho installato:

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.Sqlite
- Microsoft.EntityFrameworkCore.Tools

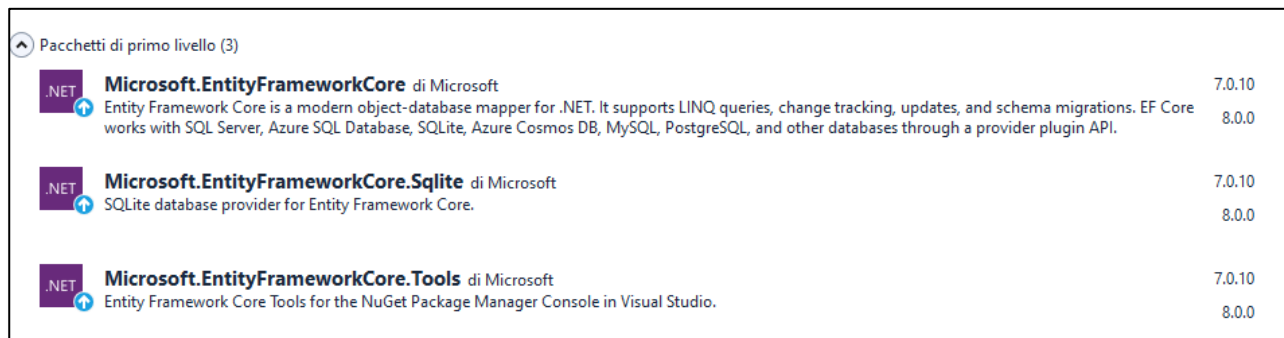


Figura 16: Pacchetti installati

5.3 Struttura cartelle core

Dopo aver installato i pacchetti, prima di iniziare a lavorare sarà necessario creare delle cartelle specifiche che conterranno i vari files.

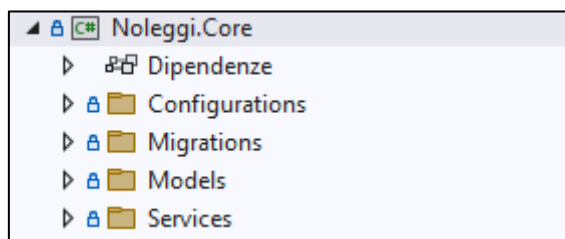


Figura 17: Struttura delle cartelle

5.3.1 Models

All'interno della cartella models troveremo i file che rappresentano le tabelle del database.

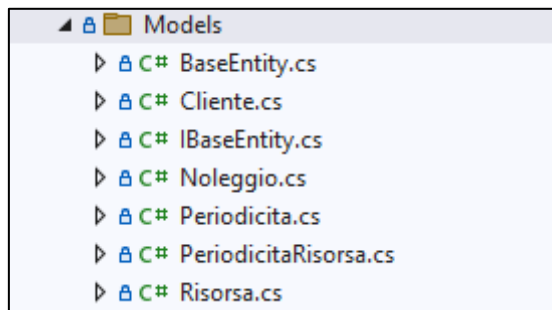


Figura 18: Cartella models

5.3.1.1 Interfaccia IBaseEntity

Creo l'interfaccia IBaseEntity con l'id.

```
public interface IBaseEntity
{
    int Id { get; set; }
}
```

5.3.1.2 Classe BaseEntity

La classe astratta BaseEntity implementa l'interfaccia in modo da obbligare tutte le altre tabelle ad avere l'id.

```
public abstract class BaseEntity : IBaseEntity
{
    public int Id { get; set; }
}
```


5.3.1.3 Classe Cliente

Questa classe conterrà tutte le varie informazioni base che deve avere un cliente.

Al suo interno come mostrato di seguito troveremo diverse proprietà come: Nome, Cognome, Indirizzo, Data di nascita, Email, Numero.

Successivamente troveremo la riga “`public virtual List<Noleggio> Noleggi { get; set; }`” questa proprietà è dichiarata come una lista di oggetti di tipo “Noleggio”, che rappresenta i noleggi effettuati dal cliente.

Infine avremo il costruttore della classe che inizializza le proprietà come stringhe vuote.

```
public class Cliente : BaseEntity
{
    #region =01=== costanti & statici =====
    #endregion

    #region =02=== membri & proprietà =====
    public string Nome { get; set; }
    public string Cognome { get; set; }
    public string Indirizzo { get; set; }
    public DateTime DataNascita { get; set; }
    public string Email { get; set; }
    public string Numero { get; set; }
    public virtual List<Noleggio> Noleggi { get; set; }
    #endregion

    #region =03=== costruttori =====
    public Cliente()
    {
        Nome = string.Empty;
        Cognome = string.Empty;
        Indirizzo = string.Empty;
        Email = string.Empty;
        Numero = string.Empty;
    }
    #endregion
}
```

5.3.1.4 Classe Risorsa

La classe “Risorsa” conterrà le informazioni che dovrà avere una risorsa da noleggiare. Al suo interno troviamo diverse proprietà, tra cui il Nome, la Categoria e lo Stato della risorsa.

Anche in questo caso abbiamo la lista “Noleggi” che rappresenta i noleggi effettuati dal cliente e la lista “PeriodicitaRisorse” che rappresenta le risorse con la loro periodicità.

Infine avremo il costruttore della classe che inizializza le proprietà come stringhe vuote mentre lo stato come “Disponibile”.

```
public class Risorsa : BaseEntity
{
    #region =01=== costanti & statici =====
    #endregion

    #region =02=== membri & proprietà =====
    public string Nome { get; set; }
    public string Categoria { get; set; }
    public string Stato { get; set; }
    public virtual List<Noleggio> Noleggi { get; set; }
    public virtual List<PeriodicitaRisorsa> PeriodicitaRisorse { get; set; }
    #endregion

    #region =03=== costruttori =====
    public Risorsa()
    {
        Nome = string.Empty;
        Categoria = string.Empty;
        Stato = "Disponibile";
    }
}
```

5.3.1.5 Classe Periodicità

In questa classe sono contenute le informazioni che riguardano la durata con la quale si noleggia una risorsa.

Abbiamo le proprietà “Durata” che corrisponde a: Giornaliera, Settimanale, Mensile, Annuale. E la proprietà “Giorno” che in base alla durata avrà un numero di giorni prefissato.

Successivamente troviamo la lista “PeriodicitaRisorse” e la lista “Noleggi”.

```
public class Periodicita : BaseEntity
{
    #region =01=== costanti & statici =====
    #endregion

    #region =02=== membri & proprietà =====
    public string Durata { get; set; }
    public int Giorno { get; set; }

    public virtual List<PeriodicitaRisorsa> PeriodicitaRisorse { get; set; }
    public virtual List<Noleggio> Noleggi { get; set; }
    #endregion

    #region =03=== costruttori =====
    public Periodicita()
    {
        Durata = string.Empty;
        Giorno = 0;
    }
    #endregion
}
```

5.3.1.6 Classe PeriodicitaRisorsa

PeriodicitaRisorsa corrisponde alla tabella ponte tra le classi Risorsa e Periodicita. Al suo interno avremo le foreign key rappresentate con il virtual.

Ogni virtual è accompagnato dal proprio id con sopra la voce [required], dove indica che la proprietà è richiesta e deve essere specificata.

```
public class PeriodicitaRisorsa : BaseEntity
{
    #region =01=== costanti & statici =====
    #endregion

    #region =02=== membri & proprietà =====
    public virtual Periodicita Periodicita { get; set; }
    [Required]
    public int PeriodicitaId { get; set; }

    public virtual Risorsa Risorsa { get; set; }
    [Required]
    public int RisorsaId { get; set; }

    public double Costo { get; set; }
    #endregion

    #region =03=== costruttori =====
    public PeriodicitaRisorsa()
    {
        Costo = 0;
    }
    #endregion
}
```

5.3.1.7 Classe Noleggio

Questa classe permette di creare un noleggio con le varie informazioni.

Abbiamo le proprietà Cliente e Risorsa che fanno riferimento alle classi spiegate precedentemente.

Successivamente le proprietà che riguardano il costo:

- CostoTeorico, il costo base
- CostoEffettivo, il costo effettivo che faccio al cliente
- CostoTotale, il costo finale

E infine quelle che riguardano le date:

- DataRitiro, quando il cliente prende la risorsa
- DataFineNoleggio, quando il cliente dovrebbe riportare la risorsa
- DataConsegnaEffettiva, quando il cliente riporta effettivamente la risorsa noleggiata

```
public class Noleggio : BaseEntity
{
    public virtual Cliente Cliente { get; set; }
    [Required]
    public int ClienteId { get; set; }

    public virtual Risorsa Risorsa { get; set; }
    [Required]
    public int RisorsaId { get; set; }

    public virtual Periodicita Periodicita { get; set; }
    [Required]
    public int PeriodicitaId { get; set; }

    public double CostoTeorico { get; set; }
    public double CostoEffettivo { get; set; }
    public double CostoTotale { get; set; }

    public DateTime DataRitiro { get; set; }
    public DateTime DataFineNoleggio { get; set; }
    public DateTime DataConsegnaEffettiva { get; set; }

    public string NomeNoleggio { get; set; }

    public Noleggio()
    {
        DataRitiro = DateTime.Now;
        NomeNoleggio = String.Empty;
        CostoEffettivo = CostoTeorico;
        DataConsegnaEffettiva = DataFineNoleggio;
    }
}
```

5.3.2 Configurations

All'interno della cartella “Configurations”, troveremo i files che servono per andare a riempire le tabelle con dei valori, alla creazione del database.

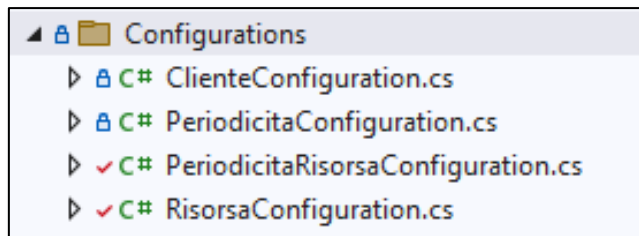


Figura 19: Cartella configurations

5.3.2.1 Classe ClienteConfiguration

Questa classe ci permette di andare a creare dei clienti automaticamente alla creazione del database.

Abbiamo le righe di codice come “`builder.Property(m => m.Nome).IsRequired();`”, esse indicano che la proprietà non può essere null ma deve avere un valore.

Successivamente invece abbiamo il “`builder.HasData(new Cliente...`” che ci va a creare un nuovo cliente con i valori che inseriremo noi.

```
public class ClienteConfiguration : IEntityTypeConfiguration<Cliente>
{
    public void Configure(EntityTypeBuilder<Cliente> builder)
    {
        builder.Property(m => m.Nome).IsRequired();
        builder.Property(m => m.Cognome).IsRequired();
        builder.Property(m => m.Indirizzo).IsRequired();
        builder.Property(m => m.Email).IsRequired();
        builder.Property(m => m.Numero).IsRequired();

        builder.HasData(
            new Cliente
            {
                Id = 1,
                Nome = "Alessandro",
                Cognome = "Perri",
                Indirizzo = "Bedano",
                DataNascita = DateTime.ParseExact
("2005-06-01", "yyyy-MM-dd", System.Globalization.CultureInfo.InvariantCulture),
                Email = "alessandro.perri@samtrevano.ch",
                Numero = "+41765545772",
            },
        );
    }
}
```

Classe RisorsaConfiguration

```
public class RisorsaConfiguration : IEntityTypeConfiguration<Risorsa>
{
    public void Configure(EntityTypeBuilder<Risorsa> builder)
    {
        builder.Property(m => m.Nome).IsRequired();
        builder.Property(m => m.Categoria).IsRequired();
        builder.Property(m => m.Stato).IsRequired();

        builder.HasData(
            new Risorsa
            {
                Id = 1,
                Nome = "MacBook",
                Categoria = "Informatica"
            },
            new Risorsa
            {
                Id = 2,
                Nome = "Iphone 15",
                Categoria = "Informatica"
            }
        );
    }
}
```

5.3.2.2 Classe PeriodicitaConfiguration

```
public class PeriodicitaConfiguration : IEntityTypeConfiguration<Periodicita>
{
    public void Configure(EntityTypeBuilder<Periodicita> builder)
    {
        builder.Property(m => m.Durata).IsRequired();
        builder.HasData(
            new Periodicita
            {
                Id = 1,
                Durata = "Giornaliera",
                Giorno = 1
            },
            new Periodicita
            {
                Id = 2,
                Durata = "Settimanale",
                Giorno = 7
            },
            new Periodicita
            {
                Id = 3,
                Durata = "Mensile",
                Giorno = 30
            },
            new Periodicita
            {
                Id = 4,
                Durata = "Annuale",
                Giorno = 360
            }
        );
    }
}
```


5.3.2.3 Classe PeriodicitaRisorsaConfiguration

```
public class PeriodicitaRisorsaConfiguration :
IEntityTypeConfiguration<PeriodicitaRisorsa>
{
    public void Configure(EntityTypeBuilder<PeriodicitaRisorsa> builder)
    {
        builder.HasKey(rcc => new { rcc.RisorsaId, rcc.PeriodicitaId });
        builder.HasData(
            new PeriodicitaRisorsa
            {
                Id = 1,
                RisorsaId = 1,
                PeriodicitaId = 1,
                Costo = 1
            },
            new PeriodicitaRisorsa
            {
                Id = 2,
                RisorsaId = 1,
                PeriodicitaId = 2,
                Costo = 10
            },
            new PeriodicitaRisorsa
            {
                Id = 3,
                RisorsaId = 1,
                PeriodicitaId = 3,
                Costo = 100
            },
            new PeriodicitaRisorsa
            {
                Id = 4,
                RisorsaId = 1,
                PeriodicitaId = 4,
                Costo = 1000
            },
            new PeriodicitaRisorsa
            {
                Id = 5,
                RisorsaId = 2,
                PeriodicitaId = 1,
                Costo = 2
            },
            new PeriodicitaRisorsa
            {
                Id = 6,
                RisorsaId = 2,
                PeriodicitaId = 2,
                Costo = 20
            }
        );
    }
}
```

5.3.3 Services

Le classi all'interno di questa cartella permettono di interagire con il database, quindi grazie ad esse nel progetto blazor avremo modo di recuperare i dati del db.

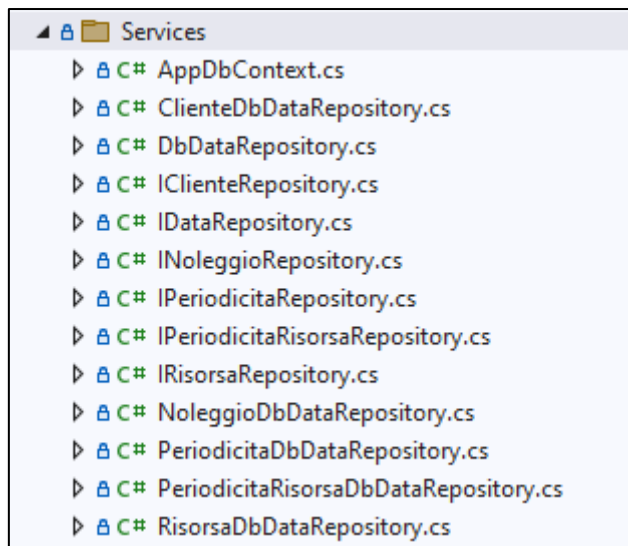


Figura 20: Cartella services

5.3.3.1 Classe AppDbContext

In questa classe troveremo i metodi che permettono di istanziare le tabelle del database e di andare a popolarle. Inizialmente troveremo che la classe estende “DbContext”.

Nelle prime righe di codice come “`public virtual DbSet<Cliente> Clienti { get; set; }`”, andiamo ad istanziare le varie tabelle, con le classi create nella cartella “Models”.

Nel metodo “OnConfiguring” andiamo ad indicare il percorso dove deve essere creato il database.

Successivamente nel metodo “OnModelCreating”, andremo a popolare queste tabelle con le classi nella cartella “Configurations” documentate al punto [5.3.2](#).

Infine avremo il metodo “SaveChanges” async e non per andare a salvare.

```
public class AppDbContext : DbContext
{
    public virtual DbSet<Cliente> Clienti { get; set; }
    public virtual DbSet<Risorsa> Risorse { get; set; }
    public virtual DbSet<Noleggio> Noleggi { get; set; }
    public virtual DbSet<Periodicita> Periodi { get; set; }
    public virtual DbSet<PeriodicitaRisorsa> PeriodicitaRisorse { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            string dbName = "DBNoleggio";
            string path = "F:\\\\SCHOOL\\\\Progetti\\\\Noleggi\\\\noleggi\\\\6_Database\\";
            string connection = "Data Source=" + path + dbName + ".db";
            optionsBuilder.UseSqlite(connection);
        }
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new ClienteConfiguration());
        modelBuilder.ApplyConfiguration(new RisorsaConfiguration());
        modelBuilder.ApplyConfiguration(new PeriodicitaConfiguration());
        modelBuilder.ApplyConfiguration(new PeriodicitaRisorsaConfiguration());

        base.OnModelCreating(modelBuilder);
    }
    public override int SaveChanges(bool acceptAllChangesOnSuccess)
    {
        return base.SaveChanges(acceptAllChangesOnSuccess);
    }
    public override Task<int> SaveChangesAsync(bool acceptAllChangesOnSuccess,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        return base.SaveChangesAsync(acceptAllChangesOnSuccess, cancellationToken);
    }
}
```

5.3.3.2 Interfaccia IRepository

In questa interfaccia vengono dichiarati i metodi async e non delle seguenti operazioni:

- Insert.
- Get.
- Update.
- Delete.

```
public interface IRepository<T> where T : BaseEntity
{
    T Get(int id);
    T Get(Expression<Func<T, bool>> expression, Func<IQueryable<T>,
IIncludableQueryable<T, object>> includes = null);
    IEnumerable<T> Get();
    IEnumerable<T> Get(
        Expression<Func<T, bool>> expression = null,
        Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
        Func<IQueryable<T>, IIncludableQueryable<T, object>> includes = null);
    T Insert(T entity);
    void Update(T entity);
    void Delete(T entity);

    Task<T> GetAsync(int id);
    Task<T> GetAsync(Expression<Func<T, bool>> expression, Func<IQueryable<T>,
IIncludableQueryable<T, object>> includes = null);
    Task<IEnumerable<T>> GetAsync();
    Task<IEnumerable<T>> GetAsync(
        Expression<Func<T, bool>> expression = null,
        Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
        Func<IQueryable<T>, IIncludableQueryable<T, object>> includes = null);
    Task<T> InsertAsync(T entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(T entity);
}
```

5.3.3.3 Classe DbRepository

Questa classe estende l'interfaccia "IRepository", al suo verranno implementati i metodi async e non che vediamo sopra.

All'estensione di questa classe vengono richiesti 2 parametri:

- Il primo di tipo C che deve estendere il "DbContext"
- Il secondo di tipo T che deve estendere la classe "BaseEntity".

```
public abstract class DbRepository<C, T> : IRepository<T> where T : BaseEntity
    where C : DbContext
```

5.3.3.4 Interfaccia IClienteRepository

Creo le interfacce di ogni classe, perché nel progetto blazor è meglio ed è più sicuro andare a prendere i dati tramite l'interfaccia che tramite la classe che implementa il tutto.

```
public interface IClienteRepository : IRepository<Cliente>
{
}
```

5.3.3.5 Classe ClienteDbDataRepository

Come spiegato nella classe “DbDataRepository”, quando si estende vengono richiesti 2 parametri, in questo caso sono la classe “AppDbContext” che estende “DbContext” e la classe “Cliente”.

Così facendo andando ad utilizzare l'interfaccia “IClienteRepository” andremo ad interagire con il database per il cliente.

Nel metodo “GetAsync()”, vado a ordinare i clienti in base al nome e al cognome.

```
public class ClienteDbDataRepository : DbDataRepository<AppDbContext, Cliente>,
IClienteRepository
{
    public ClienteDbDataRepository(AppDbContext ctx) : base(ctx)
    {
    }
    public async override Task<IEnumerable<Cliente>> GetAsync()
    {
        return (await base.GetAsync()).OrderBy(c => c.Nome + c.Cognome);
    }
}
```

5.3.3.6 Interfaccia IRisorsaRepository

```
public interface IRisorsaRepository : IRepository<Risorsa>
{
}
```

5.3.3.7 Classe RisorsaDbDataRepository

Anche qua ordino le risorse in base alla “categoria”.

```
public class RisorsaDbDataRepository : DbDataRepository<AppDbContext, Risorsa>,
IRisorsaRepository
{
    public RisorsaDbDataRepository(AppDbContext ctx) : base(ctx)
    {
    }
    public async override Task<IEnumerable<Risorsa>> GetAsync()
    {
        return (await base.GetAsync()).OrderBy(c => c.Categoria);
    }
}
```

5.3.3.8 Interfaccia IPeriodicitaRepository

```
public interface IPeriodicitaRepository : IDataRepository<Periodicita>
{
}
```

5.3.3.9 Classe PeriodicitaDbDataRepository

```
public class PeriodicitaDbDataRepository : DbDataRepository<AppDbContext, Periodicita>,
IPeriodicitaRepository
{
    public PeriodicitaDbDataRepository(AppDbContext ctx) : base(ctx)
    {
    }
}
```

5.3.3.10 Interfaccia INoleggioRepository

```
public interface INoleggioRepository : IDataRepository<Noleggio>
{
}
```

5.3.3.11 Classe NoleggioDbDataRepository

```
public class NoleggioDbDataRepository : DbDataRepository<AppDbContext, Noleggio>,
INoleggioRepository
{
    public NoleggioDbDataRepository(AppDbContext ctx) : base(ctx)
    {
    }
}
```

5.3.3.12 Interfaccia IPeriodicitaRisorsaRepository

In questa interfaccia dichiaro 2 metodi che servono per inserire all'interno della classe "risorsa" i valori della tabella ponte.

```
public interface IPeriodicitaRisorsaRepository : IRepository<PeriodicitaRisorsa>
{
    Task<IEnumerable<PeriodicitaRisorsa>> GetRisorsaAsync(int risorsaId);
    Task InsertRisorsaAsync(int risorsaId, IPeriodicitaRepository pRepo);
}
```

Classe PeriodicitaRisorsaDbDataRepository

Nel primo metodo “GetAsync()” vado ad indicare che la classe “PeriodicitaRisorsa” include le classi Periodicita e Risorsa.

Grazie al metodo “GetRisorsaAsync” in base alla risorsa selezionata vengono mostrati i dati della classe “PeriodicitaRisorsa”, quindi la durata e il costo.

Il terzo metodo “InsertRisorsaAsync” quando creo una nuova “Risorsa” va a creare anche una nuova “PeriodicitaRisorsa” con i propri valori a 0, per poi andare a modificarli a piacere.

```
public class PeriodicitaRisorsaDbDataRepository : DbDataRepository<AppDbContext,
PeriodicitaRisorsa>, IPeriodicitaRisorsaRepository
{
    public PeriodicitaRisorsaDbDataRepository(AppDbContext ctx) : base(ctx)
    {
    }
    public override async Task<IEnumerable<PeriodicitaRisorsa>> GetAsync()
    {
        var v = await base.GetAsync(includes: v => v.Include(x => x.Periodicita)
                                                .Include(x => x.Risorsa));
        return v;
    }

    public async Task<IEnumerable<PeriodicitaRisorsa>> GetRisorsaAsync(int risorsaId)
    {
        var x = base.Get().Where(pr => pr.RisorsaId == risorsaId);
        var v = await Task.Run(() => base.Get()
                                .Where(pr => pr.RisorsaId ==risorsaId));

        return v;
    }

    public async Task InsertRisorsaAsync(int risorsaId, IPeriodicitaRepository p)
    {
        foreach (var item in await p.GetAsync())
        {
            var pr = new PeriodicitaRisorsa();
            pr.PeriodicitaId = item.Id;
            pr.RisorsaId = risorsaId;
            await base.InsertAsync(pr);
        }
    }
    public override Task<PeriodicitaRisorsa> InsertAsync(PeriodicitaRisorsa entity)
    {
        throw new NotImplementedException();
    }
}
```


5.4 Creazione database

Dopo aver realizzato il progetto core, andremo a creare il database.

Prima di tutto bisognerà fare click destro sul progetto core e selezionare la voce seguente:

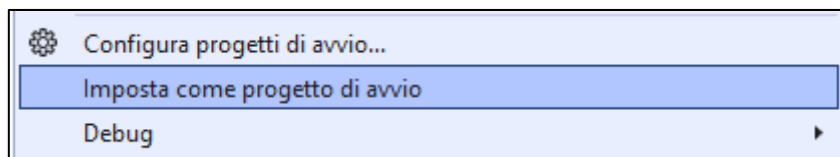


Figura 21: Imposta progetto di avvio

Dopodiché bisognerà andare nella “Console di Gestione Pacchetti” e assicurarsi che nella voce “Progetto predefinito” ci sia il progetto core.

Dopo aver fatto ciò bisognerà fare il comando “Add-Migration Initial”, questo comando andrà a creare lo schema del database.

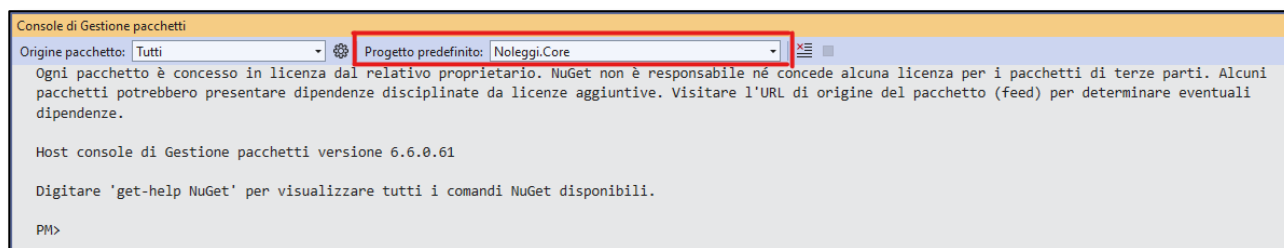


Figura 22: Progetto predefinito

Dopo averlo eseguito nel progetto core dovremo visualizzare la cartella “Migrations”.

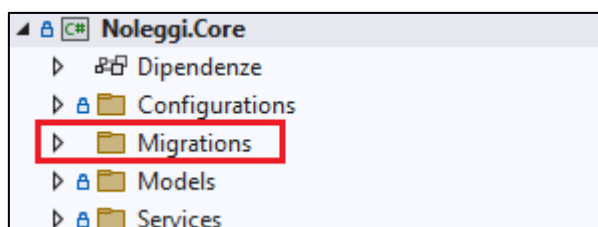


Figura 23: Cartella migrations

Infine bisognerà fare il comando “Update-Database”, questo comando invece andrà ad applicare le migrations del database.

Una volta eseguito questo comando, nel percorso specificato precedentemente al punto [5.3.3.1](#), per la creazione del database dovremmo visualizzare il file .db.

| Nome | Ultima modifica | Tipo | Dimensione |
|---|------------------|----------------|------------|
|  DBNoleggio.db | 30.11.2023 14:23 | Data Base File | 56 KB |

Figura 24: File .db

5.5 Creazione progetto blazor

Creare un nuovo progetto “App Blazor Server”. Questo tipo di progetto serve per andare a sviluppare la parte web dell’applicativo.

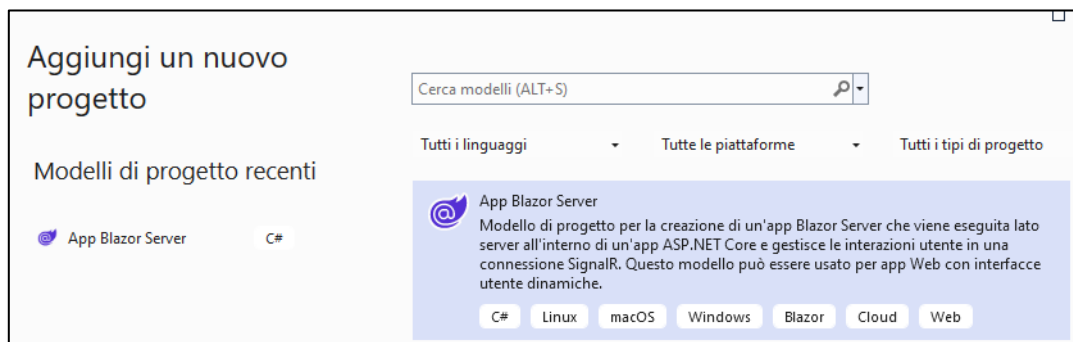


Figura 25: Creazione progetto blazor

Dopodiché si dovrà inserire il nome e il percorso.

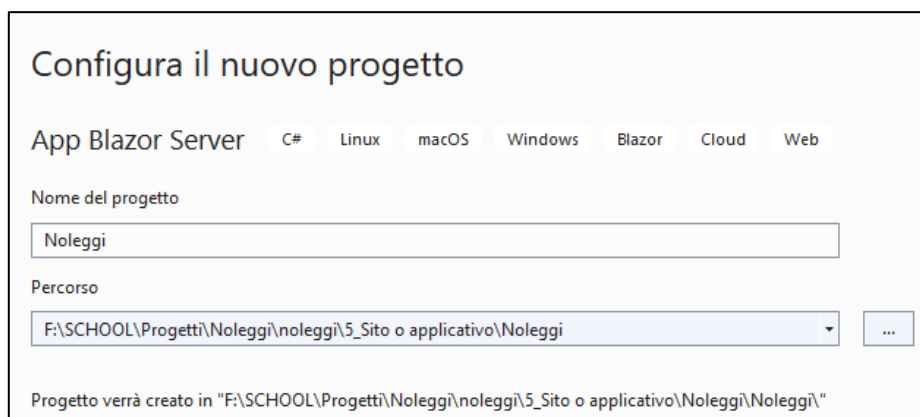


Figura 26: Configurazioni progetto blazor

Infine, troveremo delle informazioni aggiuntive. Continuare e cliccare crea.

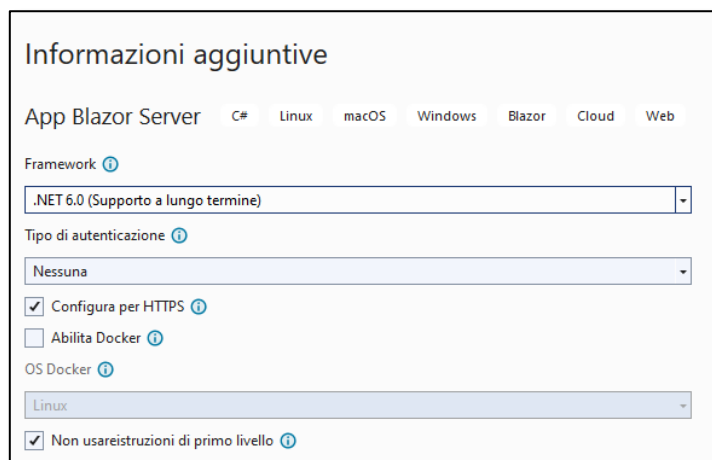


Figura 27: Informazioni progetto core

5.6 Struttura cartelle blazor

All'interno del progetto, troveremo al suo interno delle cartelle e dei files di default.

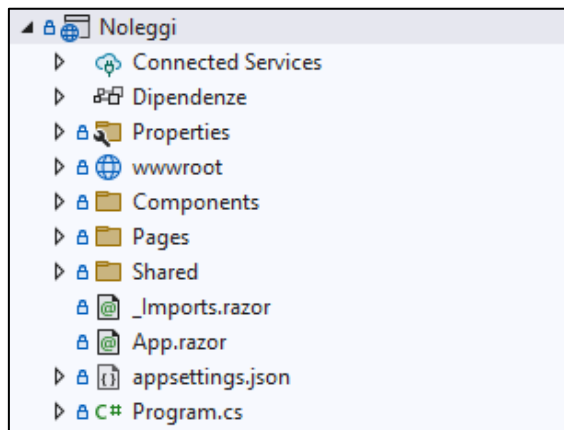


Figura 28: Struttura cartelle blazor

5.6.1 Libreria Radzen

Per questo progetto è stato richiesto di usare la libreria “Radzen”, essa è utile per migliorare in generale la grafica dei vari oggetti. Per prima cosa bisognerà andare ad installare il pacchetto “Radzen.Blazor”.

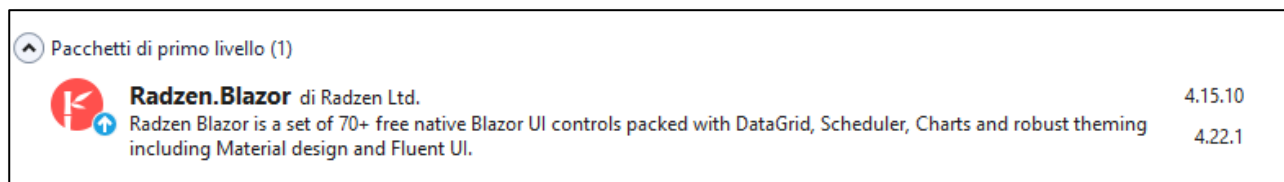


Figura 29: Pacchetto Radzen.Blazor

Successivamente andare nel file “_Imports.razor” dove al suo interno sono contenuti tutti gli using e inserire le seguenti due righe:

- @using Radzen
- @using Radzen.Blazor

Infine dopo aver importato Radzen, si dovrà includere il tema, per farlo bisogna andare nella cartella “pages”, aprire il file “_Layout.cshtml” e al suo interno inserire le seguenti righe:

- <link rel="stylesheet" href="_content/Radzen.Blazor/css/software-base.css">
- <script src="_content/Radzen.Blazor/Radzen.Blazor.js"></script>

5.6.2 Classe Program

All'interno di questa classe, dichiaro una stringa "connection", che in base a quella che gli passo nel metodo "GetConnectionString("NoleggioConnection")" va a cercare e prendere il database. Il percorso di "NoleggioConnection" lo definisco all'interno della classe "appsettings.json".

Successivamente nelle stringhe come

"builder.Services.AddScoped<IClienteRepository, ClienteDbDataRepository>();" assegno per ogni interfaccia il proprio repository, questo mi servirà per le "Dependency Injection" nei vari files per la creazione delle pagine, quando avrò bisogno del repository dove andrò a scrivere @inject <NomeInterfaccia>. Il resto è tutto di default

```
using Microsoft.EntityFrameworkCore;
using Noleggi.Core.Services;
using Radzen;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddRadzenComponents();
string connection = builder.Configuration.GetConnectionString("NoleggioConnection") ?? "";
builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite(connection));

builder.Services.AddScoped<IClienteRepository, ClienteDbDataRepository>();
builder.Services.AddScoped<IRisorsaRepository, RisorsaDbDataRepository>();
builder.Services.AddScoped<INoleggioRepository, NoleggioDbDataRepository>();
builder.Services.AddScoped<IPeriodicitaRepository, PeriodicitaDbDataRepository>();
builder.Services.AddScoped<IPeriodicitaRisorsaRepository, PeriodicitaRisorsaDbDataRepository>();
var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");
app.Run();
```

5.6.3 appsettings.json

Come spiegato precedentemente, assegno il percorso del database a “NoleggioConnection”

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "NoleggioConnection":
      "Datasource=F:\\SCH00L\\Progetti\\Noleggi\\noleggi\\6_Database\\DBNoleggio.db"
  }
}
```

5.6.4 Components

BackToList.razor

Nel caso in cui cambio pagina, grazie a questo file tramite il “RadzenButton” posso tornare indietro.

```
@inject NavigationManager nm

<RadzenButton Class="btn btn-success" ButtonStyle="ButtonStyle.Danger"
Size="ButtonSize.Large" Icon="refresh" Text="BACK" Click="@NavigateToList" />

@code {
    [Parameter] public string Target { get; set; }

    void NavigateToList()
    {
        nm.NavigateTo($"{nm.BaseUri}{Target}");
    }
}
```

5.6.5 Pages

In questa cartella saranno contenuti tutti i files per le varie pagine web. I files fuori dalle cartelle sono di default con la creazione del progetto.

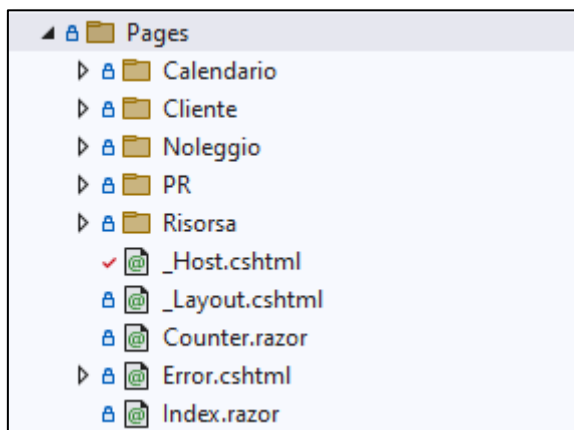


Figura 30: Cartella pages

5.6.5.1 Index.razor

In questo file con il “@page “/”” andiamo ad indicare l’URL in cui sarà mostrata la seguente pagina.

Successivamente troveremo righe come “@inject IRisorsaRepository repoRisorsa“, come spiegato al punto 5.6.2, queste righe di codice serviranno per andare a prendere i dati del database attraverso l’interfaccia che fa riferimento al repository.

Infine avremo dei tag html con il testo da mostrare nella pagina.

```
@page "/"
@inject INoleggioRepository repoNoleggio
@inject IClienteRepository repoCliente
@inject IRisorsaRepository repoRisorsa
@inject IPeriodicitaRisorsaRepository repoRR

<link rel="stylesheet" href="_content/Radzen.Blazor/css/material-base.css">
<PageTitle>Home</PageTitle>

<h1 style="text-align: center;">NOLEGGIO</h1>
<h4 style="text-align: center;">Benvenuto, noleggia e visualizza il materiale a
disposizione!</h4>
@code{
}
```

5.6.5.2 Cliente

Index.razor

Le pagine razor sono divise in 2, una parte dedicata al codice in c# e l’altra parte dedicata alla grafica utilizzando html che fa riferimento al codice c#.

Codice c#

Inizialmente creo una lista di clienti, che andrò a riempire nel metodo “OnInitializedAsync()”, così facendo avrò la lista di tutti i clienti presenti nel database.

Dopodiché creo una Grid di clienti che è richiesta nel componente “RadzenDataGrid”. Infine un valore booleano che servirà per attivare e disattivare il pulsante di creazione.

```
public List<Cliente> Models { get; set; } = new();
public Cliente Model { get; set; }

RadzenDataGrid<Cliente> clientiGrid;

bool addButtonEnable = false;

protected override async Task OnInitializedAsync()
{
    var dati = await repoCliente.GetAsync();
    Models = dati.ToList();
}
```

Il primo metodo “**EditRowAsync(Cliente cliente)**”, partirà al click del pulsante di modifica, esso serve per andare a fare una modifica del cliente nella griglia grazie al metodo “**clientiGrid.EditRow(Model)**” della classe “**RadzenDataGrid**”.

| NOME | COGNOME | DATA DI NASCITA | INDIRIZZO | NUMERO | EMAIL | |
|------------|---------|------------------|-----------|--------------|-----------------------------|--|
| Alessandro | Perri | 01.06.2005 00:00 | Bedano | +41765545772 | alessandro.perri@samtrevano | <input checked="" type="checkbox"/> <input type="checkbox"/> |

Figura 31: Metodo EditRowAsync

Nel secondo metodo “**InsertRowAsync()**”, andremo a creare e ad inserire un nuovo cliente all’interno della griglia, sempre grazie al metodo della classe “**RadzenDataGrid**”. Esso partirà al click del pulsante “Nuovo cliente”.

| NOME | COGNOME | DATA DI NASCITA | INDIRIZZO | NUMERO | EMAIL | |
|------|---------|-----------------|-----------|--------|-------|--|
| | | | | | | <input checked="" type="checkbox"/> <input type="checkbox"/> |

Figura 32: Metodo InsertRowAsync

Infine l’ultimo metodo andrà a salvare il tutto al click del pulsante verde.



```

async Task EditRowAsync(Cliente cliente)
{
    Model = cliente;
    await clientiGrid.EditRow(Model);
    addButtonEnable = true;
}
async Task InsertRowAsync()
{
    Model = new Cliente();
    await clientiGrid.InsertRow(Model);
    addButtonEnable = true;
}

async Task SaveRowAsync(Cliente cliente)
{
    await clientiGrid.UpdateRow(cliente);
}

```


Il primo metodo servirà per annullare la creazione di una nuova riga oppure la modifica.



In “OnUpdateRowAsync()” andremo ad effettuare la modifica anche al database, grazie al repository e al metodo “UpdateAsync(Model)” documentato al punto [5.3.3.3](#).

Mentre nel metodo “OnCreateRowAsync()”, andremo ad aggiungere nel database il cliente creato.

```
void CancelEdit(Cliente cliente)
{
    clientiGrid.CancelEditRow(cliente);
    addButtonEnable = false;
}
private async Task OnUpdateRowAsync()
{
    await repoCliente.UpdateAsync(Model);
    addButtonEnable = false;
}

private async Task OnCreateRowAsync()
{
    if (await VerifyClientExists(Model))
    {
        await repoCliente.InsertAsync(Model);
        addButtonEnable = false;
    }
}
```

Il metodo “DeleteAsync(Cliente cliente)” servirà per andare a cancellare il cliente sia all’interno della griglia che all’interno del database.

La variabile “confirm” grazie al “js.InvokeAsync” andrà a mostrare un alert che mi chiederà se sono sicuro di voler eliminare il cliente.

“VerifyClientExists(Cliente newClient)” va a fare una verifica se il cliente esiste già all’interno del database.

```
private async Task DeleteAsync(Cliente cliente)
{
    if (Models.Contains(cliente))
    {
        var confirm = await js.InvokeAsync<bool>("confirm", "Vuoi cancellare " +
            cliente.Nome + " " + cliente.Cognome + "?");

        if (confirm)
        {
            await repoCliente.DeleteAsync(cliente);
            await OnInitializedAsync();
            await clientiGrid.Reload();
        }
    }
    else
    {
        clientiGrid.CancelEditRow(Model);
        await clientiGrid.Reload();
    }
}

private async Task<bool> VerifyClientExists(Cliente newClient)
{
    IEnumerable<Cliente> allClient = await repoCliente.GetAsync();
    foreach (Cliente client in allClient)
    {
        if (newClient.Email.Equals(client.Email))
        {
            await js.InvokeVoidAsync("alert", "Non puoi creare il seguente cliente, un
            cliente con questa email: " + newClient.Email + " è già esistente!");
            return false;
        }
    }
    return true;
}
```

Parte grafica

Inizialmente indico l'URL della pagina, successivamente il repository ed infine l'interfaccia “**IJSRuntime**” che mi servirà per mostrare gli alert all'eliminazione di un valore.

Successivamente inserisco il titolo e una riga grigia per separare dal resto della pagina.

Infine ci saranno due pulsanti creati con la libreria radzen, di cui il primo farà partire il metodo “**InsertRowAsync**” documentato precedentemente e in base alla variabile “**addButtonEnable**” si abiliterà e si disabiliterà.

Mentre il secondo bottone farà partire il metodo “**PrintMe**” documentato al punto [5.6.5.4](#).

```
@page "/clienti"
@inject IClienteRepository repoCliente
@inject IJSRuntime js

<h1 style="text-align: center;"><span style="font-weight: bold;">CLIENTI</span></h1>
<hr style="height:4px;border-width:0;color:gray;background-color:gray" />
<br />

<RadzenButton Text="NUOVO CLIENTE" ButtonStyle="ButtonStyle.Success" Icon="add_circle_outline"
class="mt-2 mb-4" id="printPage" Click="@InsertRowAsync" Disabled=@(addButtonEnable == true) />

<RadzenButton Text="STAMPA" ButtonStyle="ButtonStyle.Primary" Icon="add_circle_outline"
class="mt-2 mb-4" id="printPage" Click="@PrintMe" />
```

Il componente “**RadzenDataGrid**” è colui che andrà a mostrare la tabella con tutti i clienti, utilizzando tutti i metodi documentati precedentemente. Per andare ad utilizzarlo sono richiesti diversi parametri come “**@ref**” che vuole la griglia, “**Data**” che vuole la lista dei clienti, i metodi “**RowUpdate**” e “**RowCreate**”.

```
<RadzenDataGrid @ref="clientiGrid" AllowFiltering="true" AllowPaging="true" PageSize="5"
AllowSorting="true" Data="@Models" TItem="Cliente" RowUpdate="@OnUpdateRowAsync"
RowCreate="@OnCreateRowAsync" ColumnWidth="150px">
```

All'interno del “RadzenDataGrid” vengono definite le colonne con “RadzenDataGridColumn”, dove nel “Property” viene definito l'attributo e nel “Title” il titolo della colonna.

Successivamente ci sarà “EditTemplate” dove quando andremo a fare una modifica oppure una creazione del cliente entrerà in azione. Al suo interno avremo in questo caso il componente “RadzenTextBox” che richiede tramite il “@bind-Value” il valore da inserire che faccia riferimento all'attributo del database, in questo caso “cliente.Nome”.

Infine avremo il componente “RadzenRequiredValidator” che se non viene inserito farà comparire un popup di richiamo.

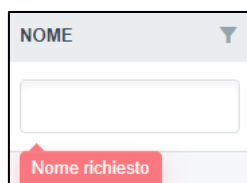


Figura 33: Popup di richiamo

```
<RadzenDataGridColumn TItem="Cliente" Property="Nome" Title="Nome">
  <EditTemplate Context="cliente">
    <RadzenTextBox @bind-Value="cliente.Nome" Name="Nome" Style="width:100%; display: block" />
    <RadzenRequiredValidator Text="Nome richiesto" Component="Nome" Popup="true" />
  </EditTemplate>
</RadzenDataGridColumn>
```

Per finire troveremo tutti i pulsanti che serviranno per far eseguire i vari metodi.

I primi due “RadzenButton” saranno il pulsante di modifica e quello di eliminazione. Gli altri due invece saranno per salvare o cancellare.

```
<RadzenDataGridColumn TItem="Noleggio" Filterable="false" Sortable="false"
  TextAlign="TextAlign.Right" Width="90px">
  <Template Context="noleggio">
    <RadzenButton ButtonStyle="ButtonStyle.Light" Icon="edit" Variant="Variant.Flat"
      Size="ButtonSize.Medium" Click="@((args => EditRowAsync(noleggio)))" />
    <RadzenButton ButtonStyle="ButtonStyle.Danger" Icon="delete" Variant="Variant.Flat"
      Shade="Shade.Lighter" Size="ButtonSize.Medium" class="my-1 ms-1"
      Click="@((args => DeleteAsync(noleggio))) @onclick:stopPropagation="true" />
  </Template>
  <EditTemplate Context="noleggio">
    <RadzenButton Icon="check" ButtonStyle="ButtonStyle.Success" Variant="Variant.Flat"
      Size="ButtonSize.Medium" Click="@((args) => SaveRowAsync(noleggio)))" />
    <RadzenButton Icon="close" ButtonStyle="ButtonStyle.Light" Variant="Variant.Flat"
      Size="ButtonSize.Medium" class="my-1 ms-1"
      Click="@((args) => CancelEdit(noleggio)))" />
  </EditTemplate>
</RadzenDataGridColumn>
```

5.6.5.3 Risorsa

Per riuscire a realizzare la tabella ponte dello schema E-R, la risorsa è strutturata in maniera diversa.

Index.razor

Nell'index al posto di utilizzare il componente “Radzen”, utilizzo una tabella normale, dove tramite un “foreach” vado a prendere tutte le risorse.

Nella riga “” al click del pulsante di modifica, vado ad assegnare l'id della risorsa.

```
@if (Models == null)
{
    <div class="alert alert-info">Caricando...</div>
}
else
{
    <table class="table table-hover">
        <thead>
            <tr>
                <th>Nome</th>
                <th>Categoria</th>
                <th>Stato</th>
                <th>Azioni</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Models)
            {
                <tr>
                    <td>@item.Nome</td>
                    <td>@item.Categoria</td>
                    <td>@item.Stato</td>
                    <td>
                        <a href="/risorse/edit/@item.Id" id="printPage">
                            <RadzenButton Icon="edit" ButtonStyle="ButtonStyle.Light"
                                Variant="Variant.Flat" Size="ButtonSize.Medium" id="printPage" />
                        </a>
                        <RadzenButton ButtonStyle="ButtonStyle.Danger" Icon="delete"
                            Variant="Variant.Flat" Shade="Shade.Lighter" Size="ButtonSize.Medium"
                            class="my-1 ms-1" id="printPage" Click="@((args) => DeleteAsync(item.Id))" />
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
```

Form.razor

In questo file utilizzo il componente “**RadzenTemplateForm**” dove al suo interno definisco le varie colonne.

Grazie alla riga “**Noleggi.Pages.PR.Edit RisorsaId = "@Model.Id"**” all’interno di questa pagina ci sarà anche il riferimento con la tabella ponte documentata successivamente al punto [5.6.5.5](#), dove sarà possibile inserire i costi della risorsa a dipendenza della durata.

```
<RadzenTemplateForm Data="@Model" OnInvalidSubmit="@OnValidSubmit">
  <RadzenRow Gap="2rem" Class="rz-p-0 rz-p-lg-4 JustifyContent="JustifyContent.Center">
    <RadzenColumn Size="12" SizeMD="6">
      <RadzenFieldset Text="Dati Cliente">
        <RadzenStack Gap="1rem">

          <RadzenRow AlignItems="AlignItems.Center">
            <RadzenColumn Size="12" SizeMD="4">
              <RadzenLabel Text="Nome" />
            </RadzenColumn>
            <RadzenColumn Size="12" SizeMD="8">
              <RadzenTextBox Style="width: 100%;" Name="Nome"
                @bind- Value="@Model.Nome" />
              <RadzenRequiredValidator Text="Nome richiesto"
                Component="Nome" Popup="true" />
            </RadzenColumn>
          </RadzenRow>

          <RadzenRow AlignItems="AlignItems.Center">
            <RadzenColumn Size="12" SizeMD="4">
              <RadzenLabel Text="Categoria" />
            </RadzenColumn>
            <RadzenColumn Size="12" SizeMD="8">
              <RadzenTextBox Style="width: 100%;" Name="Categoria"
                @bind-Value="@Model.Categoria" />
              <RadzenRequiredValidator Text="Categoria richiesta"
                Component="Categoria" Popup="true" />
            </RadzenColumn>
          </RadzenRow>

        </RadzenStack>
      </RadzenFieldset>
    </RadzenColumn>
  </RadzenRow>

  <Noleggi.Pages.PR.Edit RisorsaId="@Model.Id"/>
</RadzenTemplateForm>
```

Create.razor

In questa classe richiamo il “**Form**” passando come parametro la nuova risorsa, dopodiché creo un pulsante che andrà ad eseguire il metodo nel “**@code**” e richiamo il pulsante “**BackToList**” documentato al punto [5.6.4](#).

```
@page "/risorse/create"
@inject IRisorsaRepository repoRisorsa
@inject IPeriodicitaRisorsaRepository repoPR
@inject IPeriodicitaRepository repoP
@inject NavigationManager nm

<h1 style="text-align: center;"><span style="font-weight: bold;">CREAZIONE DI UNA NUOVA
RISORSA</span></h1>
<br />
<br />
<br />

<Form Model="@Model" />

<RadzenStack Orientation="Orientation.Horizontal" JustifyContent="JustifyContent.Center"
              Gap="2rem" Class="rz-p-0 rz-p-lg-4">
    <RadzenButton Class="btn btn-success" ButtonStyle="ButtonStyle.Success"
                  Size="ButtonSize.Large" Icon="save"
                  Text="SAVE" Click="@InsertAsync" />

    <BackToList Target="risorse" />
</RadzenStack>

@code {
    public Risorsa Model { get; set; } = new();

    private async Task InsertAsync()
    {
        await repoRisorsa.InsertAsync(Model);
        await repoPR.InsertRisorsaAsync(Model.Id, repoP);

        nm.NavigateTo("/risorse");
    }
}
```

Edit.razor

Inizialmente nell'URL indico che deve essere assegnato un "Id".

Tramite questo "Id" vado a riempire una nuova risorsa con il valore selezionato grazie al metodo "OnParametersSetAsync()".

Infine al click del pulsante "salva" partirà il metodo "UpdateAsync()" che aggiornerà la risorsa.

```
@page "/risorse/edit/{id:int}"
@inject IRisorsaRepository repoRisorsa
@inject NavigationManager nm

<h1 style="text-align: center;"><span style="font-weight: bold;">MODIFICA DI UNA
RISORSA</span></h1>
<br />
<br />
<br />

<Form Model="@Model" />

<RadzenStack Orientation="Orientation.Horizontal"
JustifyContent="JustifyContent.Center" Gap="2rem" Class="rz-p-0 rz-p-lg-4">
    <RadzenButton Class="btn btn-success" ButtonStyle="ButtonStyle.Success"
Size="ButtonSize.Large" Icon="save" Text="SAVE" Click="@UpdateAsync" />
    <BackToList Target="risorse" />
</RadzenStack>

@code {
    public Risorsa Model { get; set; } = new();
    [Parameter] public int Id { get; set; }

    protected override async Task OnParametersSetAsync()
    {
        Model = await repoRisorsa.GetAsync(Id);
    }

    private async Task UpdateAsync()
    {
        await repoRisorsa.UpdateAsync(Model);
        nm.NavigateTo("/risorse");
    }
}
```


5.6.5.4 Stampa

Per soddisfare il requisito della stampa, nelle pagine “Clienti” e “Risorse” ho aggiunto il metodo “PrintMe()” che partirà al click dei pulsanti “stampa” all’interno delle pagine.



```
private async Task PrintMe()
{
    await js.InvokeVoidAsync("window.print");
}
```

Per migliorare la stampa, nel file “site.css” nel percorso: “wwwroot/css” ho aggiunto una nuova istruzione “@media print”, che tramite l’id “#printPage” andrà a nascondere il componente alla stampa

```
@media print {
    #printPage {
        display: none;
    }
}
```

5.6.5.5 PR

Edit.razor

Il contenuto di questo file è uguale a quello nella pagina “Cliente” documentata al punto [5.6.5.2](#). La parte grafica verrà mostrata nella pagina “Risorsa” documentata al punto [5.6.5.3](#).

5.6.5.6 Noleggio

Index.razor

In questo file la parte grafica sarà uguale a quella della pagina “Cliente”, mentre per quanto riguarda il codice ci sono alcune cose aggiuntive.

Creo le liste di tutti gli oggetti che fanno riferimento al noleggio.

```
public List<Risorsa> Risorse { get; set; } = new();
public List<Cliente> Clienti { get; set; } = new();
public List<Periodicita> Periodi { get; set; } = new();
public List<PeriodicitaRisorsa> PR { get; set; } = new();
```

Ci sono due variabili booleane per disabilitare alcuni campi quando si crea oppure si modifica. E c’è anche una variabile double che servirà per fare il calcolo dei giorni in base alle date di prenotazione della risorsa.

```
bool disabilitaCreate;
bool disabilitaModifiy;
double days = 0;
```

All'interno di questo metodo andrò a riempire le liste tramite i repository. Questo mi servirà per mostrare i valori nella grafica tramite il “RadzenDropDown” e per andare ad impostare automaticamente il costo della risorsa.

```
protected override async Task OnInitializedAsync()
{
    var dati = await repoNoleggio.GetAsync();
    Models = dati.ToList();

    Risorse = (await repoRisorsa.GetAsync()).ToList();
    Clienti = (await repoCliente.GetAsync()).ToList();
    Periodi = (await repoPeriodicita.GetAsync()).ToList();
    PR = (await repoPR.GetAsync()).ToList();

    addButtonEnable = false;
}
```

Nella variabile “days” verrà salvato il valore di differenza tra i giorni mentre in “noleggio.CostoTotale” verrà effettuata la moltiplicazione del costo in base ai giorni.

```
async Task SaveRowAsync(Noleggio noleggio)
{
    days = noleggio.DataConsegnaEffettiva.Subtract(noleggio.DataRitiro).Days;
    noleggio.CostoTotale = noleggio.CostoEffettivo * (days + 1);
    await noleggiGrid.UpdateRow(noleggio);
}
```

Alla “DataFineNoleggio” viene aggiunta 1 ora perché essendo di tipo “DateTime” il calcolo dei giorni sarebbe sbagliato.

Mentre tramite un foreach che fa passare tutte le “PeriodicitaRisorse” assegno in base alla risorsa il costo.

```
private async Task OnCreateRowAsync()
{
    if (await VerifyNoleggioExists(Model))
    {
        Model.DataFineNoleggio = Model.DataFineNoleggio.AddHours(1);
        Model.DataConsegnaEffettiva = Model.DataFineNoleggio;
        await repoNoleggio.InsertAsync(Model);

        Model.NomeNoleggio = Model.Cliente.Email + " - " + Model.Risorsa.Nome;

        ModelRisorsa = await repoRisorsa.GetAsync(Model.RisorsaId);
        ModelRisorsa.Stato = "In noleggio";

        foreach (var item in PR)
        {
            if (Model.RisorsaId == item.RisorsaId)
            {
                if (Model.PeriodicitaId == item.PeriodicitaId)
                {
                    Model.CostoTeorico = item.Costo;
                }
            }
        }
        Model.CostoEffettivo = Model.CostoTeorico;
        await repoRisorsa.UpdateAsync(ModelRisorsa);

        disabilitaCreate = false;
        disabilitaModifiy = false;
        addButtonEnable = false;
    }
}
```

5.6.5.7 Calendario

Index.razor

Codice c#

Creo una variabile di tipo “RadzenScheduler” e un “Dictionary” che sono richiesti per lo sviluppo del calendario.

La variabile “events” è contiene coppie di “chiave-valore” dove la chiave è un oggetto di tipo DateTime e il valore è una stringa.

Il metodo “OnSlotRender(SchedulerSlotRenderEventArgs args)” era già implementato, esso andrà ad evidenziare nel calendario il giorno attuale.

```
@code {
    public List<Noleggio> Models { get; set; } = new();
    public List<PeriodicitaRisorsa> PR { get; set; } = new();

    RadzenScheduler<Noleggio> scheduler;
    Dictionary<DateTime, string> events = new Dictionary<DateTime, string>();

    public Noleggio Model { get; set; }
    public Risorsa ModelRisorsa { get; set; } = new();

    double days = 0;

    protected async override Task OnInitializedAsync()
    {
        // Recupero dei dati
        var dati = await repoNoleggio.GetAsync();
        Models = dati.ToList();
        PR = (await repoPR.GetAsync()).ToList();
    }

    void OnSlotRender(SchedulerSlotRenderEventArgs args)
    {
        if (args.View.Text == "Month" && args.Start.Date == DateTime.Today)
        {
            args.Attributes["style"] = "background: rgba(255,220,40,.2)";
        }

        if ((args.View.Text == "Week" || args.View.Text == "Day") && args.Start.Hour >
            8 && args.Start.Hour < 19)
        {
            args.Attributes["style"] = "background: rgba(255,220,40,.2)";
        }
    }
}
```

Questo metodo “CasellaVuotaSelezionata” fa le stesse cose del metodo “OnCreateRowAsync()” nella pagina “Noleggio”, esso partirà quando una casella vuota verrà premuta.

L'unica cosa aggiuntiva è la variabile “data” che tramite il parametro “.OpenAsync<Form>” aprirà la pagina “Form.razor”. Esso apre un dialogo con il titolo “Nuovo Noleggio” e passa due chiavi-valori (“DataInizio” e “DataFine”) all'interno di un dizionario. Questi valori verranno trasmessi per popolare i campi di inizio e di fine del noleggio.

```

async Task CasellaVuotaSelezionata(SchedulerSlotSelectEventArgs args)
{
    if (args.View.Text != "Year")
    {
        Noleggio data = await DialogService.OpenAsync<Form>("Nuovo Noleggio",
            new Dictionary<string, object>
            { { "DataInizio", args.Start }, { "DataFine", args.End } });

        Model = data;
        if (data != null)
        {
            if (await VerifyNoleggioExists(data))
            {
                Models.Add(data);

                Model.DataFineNoleggio = Model.DataFineNoleggio.AddHours(1);
                Model.DataConsegnaEffettiva = Model.DataFineNoleggio;

                await repoNoleggio.InsertAsync(Model);

                Model.NomeNoleggio = Model.Cliente.Email + " - " +
                    Model.Risorsa.Nome;

                ModelRisorsa = await repoRisorsa.GetAsync(Model.RisorsaId);
                ModelRisorsa.Stato = "In noleggio";

                days = Model.DataConsegnaEffettiva.Subtract(Model.DataRitiro).Days;

                foreach (var item in PR)
                {
                    if (Model.RisorsaId == item.RisorsaId)
                    {
                        if (Model.PeriodicitaId == item.PeriodicitaId)
                        {
                            Model.CostoTeorico = item.Costo;
                        }
                    }
                }
                Model.CostoEffettivo = Model.CostoTeorico;
                Model.CostoTotale = Model.CostoEffettivo * (days + 1);

                await repoRisorsa.UpdateAsync(ModelRisorsa);
                await scheduler.Reload();
            }
        }
    }
}

```

Se all'interno del calendario verrà cliccato un noleggio già esistente partirà il seguente metodo.

Nella variabile “copy” faccio una copia del noleggio selezionato.

Dopodiché tramite la variabile “data” verrà aperta la pagina “Edit.razor” e al “**ModelNoleggio**” verrà assegnata la copia.

```

async Task NoleggioSelezionato(SchedulerAppointmentSelectEventArgs<Noleggio> args)
{
    var copy = new Noleggio
    {
        Id = args.Data.Id,
        ClienteId = args.Data.ClienteId,
        RisorsaId = args.Data.RisorsaId,
        PeriodicitaId = args.Data.PeriodicitaId,
        DataRitiro = args.Start,
        DataFineNoleggio = args.End,
        DataConsegnaEffettiva = args.Data.DataConsegnaEffettiva,
        CostoEffettivo = args.Data.CostoEffettivo,
        NomeNoleggio = args.Data.NomeNoleggio
    };

    Noleggio data = await DialogService.OpenAsync<Edit>("Modifica Noleggio",
        new Dictionary<string, object> { { "ModelNoleggio", copy } });
    Model = data;
    if (data != null)
    {
        Noleggio noleggioToModify = await repoNoleggio.GetAsync(data.Id);
        noleggioToModify.ClienteId = data.ClienteId;
        noleggioToModify.RisorsaId = data.RisorsaId;
        noleggioToModify.PeriodicitaId = data.PeriodicitaId;
        noleggioToModify.DataRitiro = data.DataRitiro;
        noleggioToModify.DataFineNoleggio = data.DataFineNoleggio;
        noleggioToModify.DataConsegnaEffettiva = data.DataConsegnaEffettiva;
        noleggioToModify.CostoEffettivo = data.CostoEffettivo;

        foreach (var item in PR)
        {
            if (Model.RisorsaId == item.RisorsaId)
            {
                if (Model.PeriodicitaId == item.PeriodicitaId)
                {
                    Model.CostoTeorico = item.Costo;
                }
            }
        }
        Model.CostoEffettivo = Model.CostoTeorico;
        Model.CostoTotale = Model.CostoEffettivo * (days + 1);

        await repoNoleggio.UpdateAsync(noleggioToModify);
        noleggioToModify.NomeNoleggio = noleggioToModify.Cliente.Email + " - " +
            noleggioToModify.Risorsa.Nome;
    }

    await scheduler.Reload();
}

```

Parte grafica

Grazie alla libreria Radzen ho utilizzato il componente “**RadzenScheduler**” dove nella parte grafica mostrerà il calendario.

```
<RadzenScheduler @ref=@scheduler style="height: 600px;" TItem="Noleggio" Data=@Models
SelectedIndex="2" TextProperty="NomeNoleggio"
    StartProperty="DataRitiro" EndProperty="DataConsegnaEffettiva"
    SlotRender=@OnSlotRender AppointmentSelect=@NoleggioSelezionato
    SlotSelect=@CasellaVuotaSelezionata>
    <RadzenDayView />
    <RadzenWeekView />
    <RadzenMonthView />
    <RadzenYearView />
</RadzenScheduler>
```

Form.razor

Codice c#

Nel codice richiederemo 3 parametri da passare, questi parametri verranno assegnati nella pagina “Index.razor” all’interno del metodo “CasellaVuotaSelezionata” tramite la variabile “data”.

Nel metodo “OnParametersSet” assegno alle due date richieste i valori del noleggio.

```
@code {
    [Parameter] public Noleggio Model { get; set; } = new();

    [Parameter] public DateTime DataInizio { get; set; }
    [Parameter] public DateTime DataFine { get; set; }

    public List<Risorsa> Risorse { get; set; }
    public List<Cliente> Clienti { get; set; }
    public List<Periodicita> Periodi { get; set; } = new();
    public List<PeriodicitaRisorsa> PR { get; set; } = new();

    protected async override Task OnInitializedAsync()
    {
        // Recupero dei dati
        Risorse = (await repoRisorsa.GetAsync()).ToList();
        Clienti = (await repoCliente.GetAsync()).ToList();
        Periodi = (await repoPeriodicita.GetAsync()).ToList();
        PR = (await repoPR.GetAsync()).ToList();
    }
    protected override void OnParametersSet()
    {
        DataInizio = Model.DataRitiro;
        DataFine = Model.DataConsegnaEffettiva;
    }

    public void OnSubmit(Noleggio noleggio)
    {
        DialogService.Close(noleggio);
    }
}
```

Parte grafica

Per la parte grafica viene usato il componente “RadzenTemplateForm” come nella pagina “Risorsa” al punto [5.6.5.3](#).

Edit.razor

Codice c#

Nel codice è richiesto un solo parametro, che verrà assegnato nella pagina “Index.razor”, nel metodo “NoleggioSelezionato” all’interno della variabile “data”.

```
@code {
    [Parameter] public Noleggio ModelNoleggio { get; set; } = new();

    public Noleggio Model { get; set; } = new();

    public List<Risorsa> Risorse { get; set; }
    public List<Cliente> Clienti { get; set; }
    public List<Periodicita> Periodi { get; set; } = new();
    public List<PeriodicitaRisorsa> PR { get; set; } = new();

    protected async override Task OnInitializedAsync()
    {
        // Recupero dei dati
        Risorse = (await repoRisorsa.GetAsync()).ToList();
        Clienti = (await repoCliente.GetAsync()).ToList();
        Periodi = (await repoPeriodicita.GetAsync()).ToList();
        PR = (await repoPR.GetAsync()).ToList();
    }
    protected override void OnParametersSet()
    {
        Model = ModelNoleggio;
    }

    public void OnSubmit(Noleggio noleggio)
    {
        DialogService.Close(noleggio);
    }
}
```

Parte grafica

Anche qua per la parte grafica si utilizzerà un “RadzenTemplateForm” come componente.

5.6.6 Shared

NavMenu.razor

In questo file vado a modificare soltanto il secondo div, aggiungendo un “NavLink” per ogni pagina creata, così da poterla visualizzare e poterci interagire nella navbar.

```
<div class="top-row ps-3 navbar navbar-dark" id="printPage">
  <div class="container-fluid">
    <a class="navbar-brand" href="">Noleggi</a>
    <button title="Navigation menu" class="navbar-toggler"
@onclick="ToggleNavMenu">
      <span class="navbar-toggler-icon"></span>
    </button>
  </div>
</div>

<div class="@NavMenuCssClass" id="printPage" @onclick="ToggleNavMenu">
  <nav class="flex-column">
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
      </NavLink>
    </div>
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="clienti">
        <span class="oi oi-person" aria-hidden="true"></span> Clienti
      </NavLink>
    </div>
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="risorse">
        <span class="oi oi-plus" aria-hidden="true"></span> Risorse
      </NavLink>
    </div>
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="noleggi">
        <span class="oi oi-book" aria-hidden="true"></span> Noleggi
      </NavLink>
    </div>
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="calendar">
        <span class="oi oi-calendar" aria-hidden="true"></span> Calendario
      </NavLink>
    </div>
  </nav>
</div>

@code {
  private bool collapseNavMenu = true;

  private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

  private void ToggleNavMenu()
  {
    collapseNavMenu = !collapseNavMenu;
  }
}
```

5.7 Design finale delle interfacce

In questo capitolo ci saranno le interfacce finali della struttura, esse somigliano molto a quelle progettate al punto [4.2](#).

5.7.1 Pagina home

Questa è la home iniziale di benvenuto.

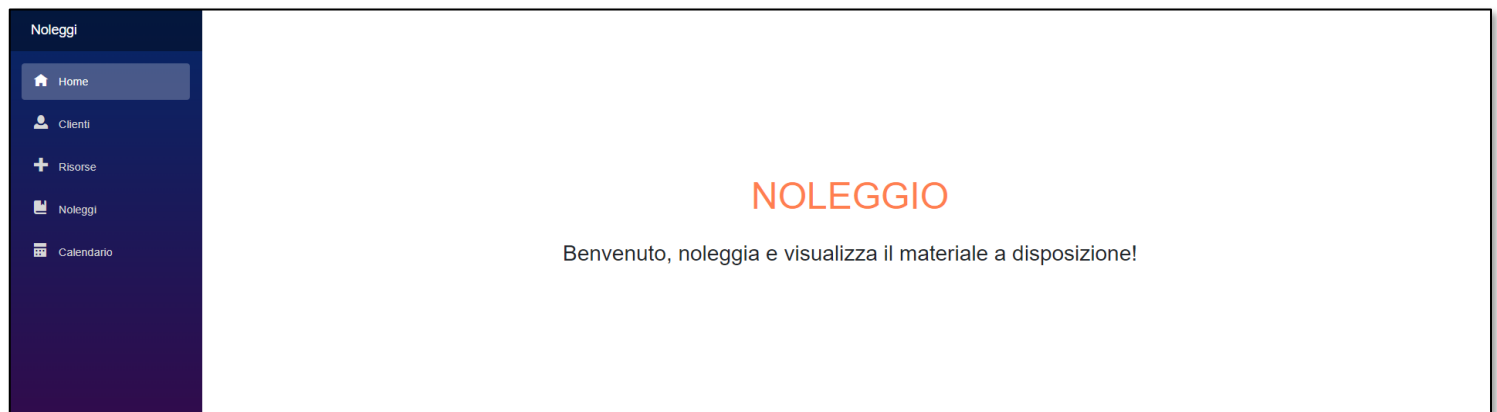


Figura 34: Pagina home finale

5.7.2 Pagina clienti

In questa pagina vengono mostrati tutti i clienti e ci sarà la possibilità di inserirne di nuovi, di modificare quelli già esistenti o di eliminarli. Sarà possibile anche eseguire una stampa.

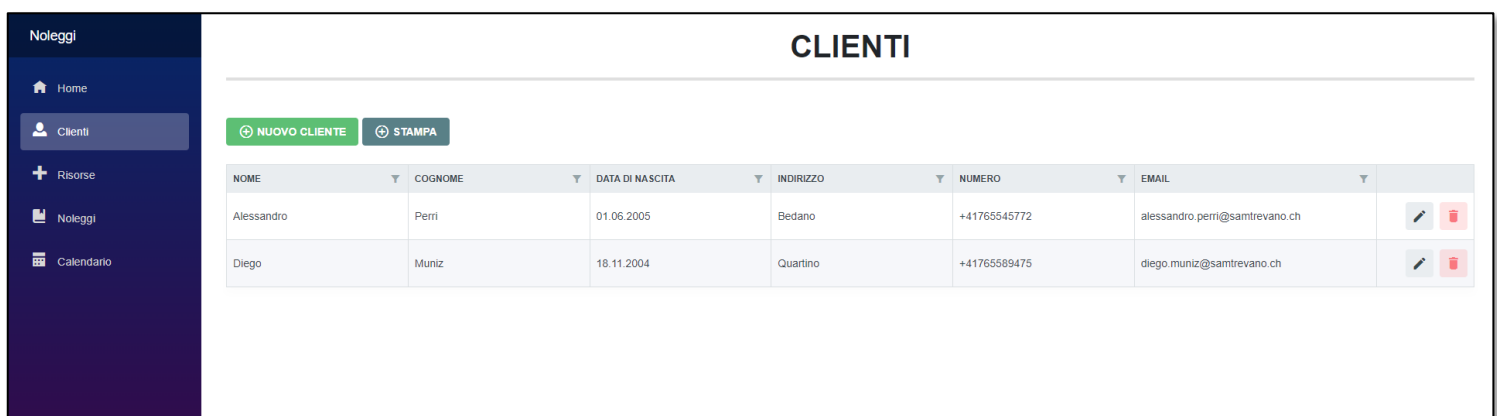


Figura 35: Pagina clienti finale

5.7.3 Pagina risorse

Come nella pagina dei clienti, qua sarà possibili visualizzare, creare, modificare ed eliminare le risorse messe a disposizione dei clienti. Anche qua sarà possibile stampare.

| Noleggi | RISORSE | | | |
|------------|---------|--|--|--|
| | | | | |
| Home | | | | |
| Clients | | | | |
| + Risorse | | | | |
| Noleggi | | | | |
| Calendario | | | | |




| + NUOVA RISORSA | | + STAMPA | |
|-----------------|-------------|-------------|---|
| Nome | Categoria | Stato | Azioni |
| MacBook | Informatica | Disponibile |   |
| Iphone 15 | Informatica | Disponibile |   |

Figura 36: Pagina risorse finale

5.7.4 Pagina noleggi

All'interno di questa pagina, sarà possibile effettuare un noleggio delle risorse messe a disposizione.

| Noleggi | NOLEGGI | | | | | | | |
|------------|---------|--|--|--|--|--|--|--|
| | | | | | | | | |
| Home | | | | | | | | |
| Clients | | | | | | | | |
| + Risorse | | | | | | | | |
| Noleggi | | | | | | | | |
| Calendario | | | | | | | | |

| + NUOVO NOLEGGIO | | | | | | | | |
|--------------------------------|-----------|-------------|-------------|------------|---------------|-----------|--------------|---|
| CLIENTE | RISORSA | DURATA | DATA RITIRO | DATA FINE | DATA CONSEGNA | COSTO | COSTO TOTALE | |
| alessandro.perri@samtrevano.ch | Iphone 14 | Giornaliera | 04.12.2023 | 08.12.2023 | 08.12.2023 | CHF 50.00 | CHF 250.00 |   |

Figura 37: Pagina noleggi finale

5.7.5 Pagina pianificazione

Infine ci sarà il calendario, al suo interno sarà possibile visualizzare, modificare i noleggi effettuati e crearne di nuovi.

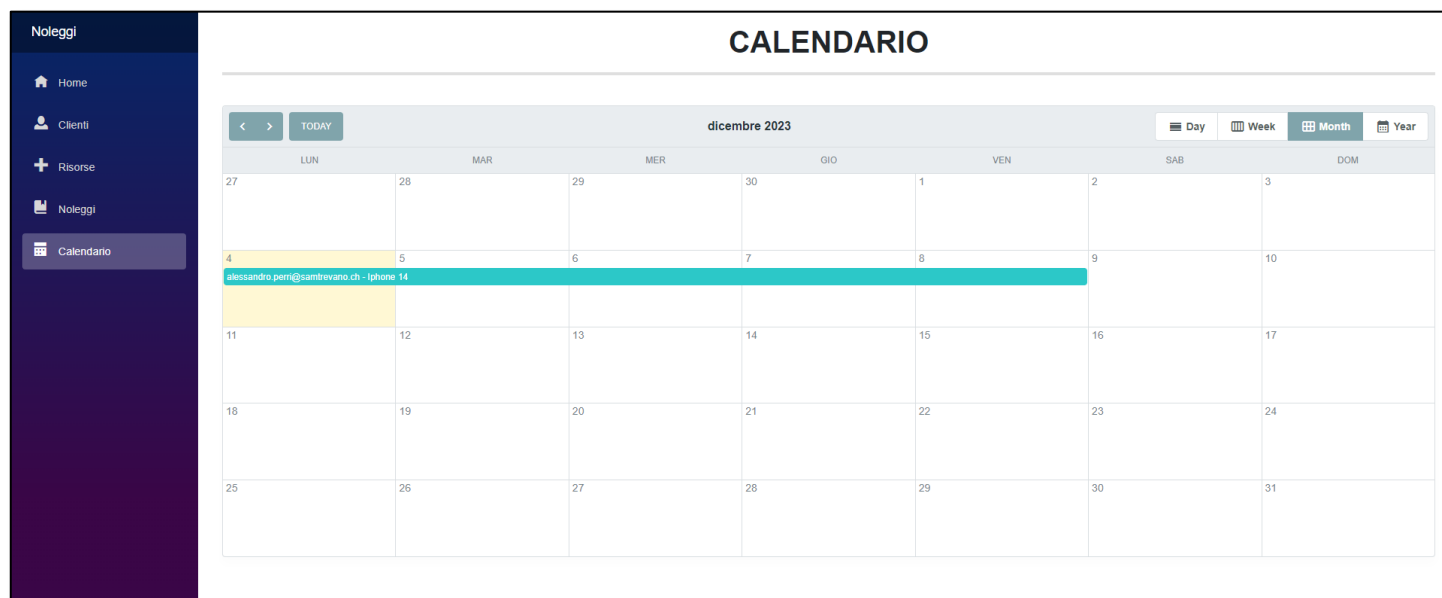


Figura 38: Pagina pianificazione finale

6 Test

6.1 Protocollo di test

| | | | |
|-------------------------|---|-------------|---------------------------------|
| Test Case | TC-001 | Nome | Visualizzare la pagina iniziale |
| Riferimento | REQ-01 | | |
| Descrizione | Visualizzare la home all'avvio dell'applicativo | | |
| Prerequisiti | | | |
| Procedura | 1. Avviare l'applicativo | | |
| Risultati attesi | Visualizzare la homepage e i pulsanti per accedere alle altre sezioni | | |

| | | | |
|-------------------------|---|-------------|---------------------------------|
| Test Case | TC-002 | Nome | Visualizzare la sezione clienti |
| Riferimento | REQ-03 | | |
| Descrizione | Accedere alla sezione clienti e visualizzare i vari pulsanti | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | 1. Premere il pulsante clienti 2. Visualizzare pulsante “nuovo cliente” 3. Visualizzare pulsante “stampa” | | |
| Risultati attesi | Visualizzare la pagina e i vari pulsanti | | |

| | | | |
|------------------------------|--|-------------|------------------------|
| Test Case Riferimento | TC-003 REQ-03 | Nome | Opzioni pagina clienti |
| Descrizione | Poter eseguire le varie opzioni: creare, modificare ed eliminare | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | <ol style="list-style-type: none"> 1. Premere il pulsante “nuovo cliente” 2. Inserire i dati e salvare 3. Premere il pulsante di modifica 4. Modificare e salvare 5. Eliminare il cliente | | |
| Risultati attesi | Posso creare, modificare ed eliminare un cliente | | |

| | | | |
|------------------------------|---|-------------|---------------------------------|
| Test Case Riferimento | TC-004 REQ-04 | Nome | Visualizzare la sezione risorse |
| Descrizione | Accedere alla sezione risorse e visualizzare i vari pulsanti | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | <ol style="list-style-type: none"> 1. Premere il pulsante risorse 2. Visualizzare pulsante “nuova risorsa” 3. Visualizzare pulsante “stampa” | | |
| Risultati attesi | Visualizzare la pagina e i vari pulsanti | | |

| | | | |
|------------------------------|--|-------------|------------------------|
| Test Case Riferimento | TC-005 REQ-04 | Nome | Opzioni pagina risorse |
| Descrizione | Poter eseguire le varie opzioni: creare, modificare ed eliminare | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | <ol style="list-style-type: none"> 1. Premere il pulsante “nuova risorsa” 2. Inserire i dati e salvare 3. Premere il pulsante di modifica 4. Modificare e salvare 5. Eliminare la risorsa | | |
| Risultati attesi | Posso creare, modificare ed eliminare una risorsa | | |

| | | | |
|------------------------------|---|-------------|---------------------------------|
| Test Case Riferimento | TC-006 REQ-05 | Nome | Visualizzare la sezione noleggi |
| Descrizione | Accedere alla sezione noleggi e visualizzare i vari pulsanti | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | <ol style="list-style-type: none"> 1. Premere il pulsante noleggi 2. Visualizzare pulsante “nuovo noleggio” | | |
| Risultati attesi | Visualizzare la pagina e i vari pulsanti | | |

| | | | |
|-------------------------|---|-------------|------------------------|
| Test Case | TC-007 | Nome | Opzioni pagina noleggi |
| Riferimento | REQ-05 | | |
| Descrizione | Poter eseguire le varie opzioni: creare, modificare ed eliminare | | |
| Prerequisiti | Avvio dell'applicativo Aver creato dei clienti Aver creato delle risorse | | |
| Procedura | 1. Premere il pulsante “nuovo noleggio” 2. Inserire i dati e salvare 3. Premere il pulsante di modifica 4. Modificare e salvare 5. Eliminare la risorsa | | |
| Risultati attesi | Posso creare, modificare ed eliminare un noleggio | | |

| | | | |
|-------------------------|--|-------------|------------------------------------|
| Test Case | TC-008 | Nome | Visualizzare la sezione calendario |
| Riferimento | REQ-06 | | |
| Descrizione | Accedere alla sezione calendario e visualizzare un calendario con all'interno i noleggi effettuati | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | 1. Premere il pulsante calendario | | |
| Risultati attesi | Visualizzare il calendario | | |

| | | | |
|-------------------------|---|-------------|---------------------------|
| Test Case | TC-009 | Nome | Opzioni pagina calendario |
| Riferimento | REQ-06 | | |
| Descrizione | Poter eseguire le varie opzioni: creare e modificare | | |
| Prerequisiti | Avvio dell'applicativo Aver creato dei clienti Aver creato delle risorse | | |
| Procedura | 1. Premere una casella vuota 2. Inserire i dati e salvare 3. Premere un noleggio già esistente 4. Modificare e salvare | | |
| Risultati attesi | Posso creare e modificare un noleggio | | |

| | | | |
|-------------------------|---|-------------|--------------------------|
| Test Case | TC-010 | Nome | Utilizzo pulsante stampa |
| Riferimento | REQ-07 | | |
| Descrizione | Al click del pulsante mi esce la pagina per poter stampare il contenuto | | |
| Prerequisiti | Avvio dell'applicativo | | |
| Procedura | 1. Andare nella sezione clienti o risorse 2. Premere il pulsante “stampa” 3. Visualizzare la pagina per stampare il contenuto | | |
| Risultati attesi | Poter stampare il contenuto della pagina | | |

| | | | |
|-------------------------|--|-------------|--|
| Test Case | TC-011 | Nome | Macchina virtuale funzionante e applicativo caricato |
| Riferimento | REQ-08 | | |
| Descrizione | Avviare la macchina e verificarne il funzionamento | | |
| Prerequisiti | | | |
| Procedura | <ol style="list-style-type: none"> 1. Avviare la macchina 2. Digitare l'URL del sito visualizzarlo | | |
| Risultati attesi | Si visualizza l'applicazione web dei noleggi. | | |

6.2 Risultati test

| Test Case | Risultato ottenuto | Stato |
|-----------|---|---------|
| TC-001 | L'applicativo parte senza errori e si visualizza la homepage | Passato |
| TC-002 | La pagina clienti viene visualizzata | Passato |
| TC-003 | Creare, modificare ed eliminare un cliente funziona | Passato |
| TC-004 | La pagina delle risorse viene visualizzata | Passato |
| TC-005 | Creare, modificare ed eliminare una risorsa funziona | Passato |
| TC-006 | La pagina dei noleggi viene visualizzata | Passato |
| TC-007 | Creare, modificare ed eliminare un noleggio funziona | Passato |
| TC-008 | Il calendario viene visualizzato | Passato |
| TC-009 | Nel calendario posso creare o modificare un noleggio | Passato |
| TC-010 | Quando clicco il pulsante stampa nella pagina clienti e risorse viene visualizzata la pagina per stampare | Passato |
| TC-011 | La macchina virtuale e il caricamento dell'applicativo non sono stati realizzati. | Errore |

6.3 Risultati UnitTest

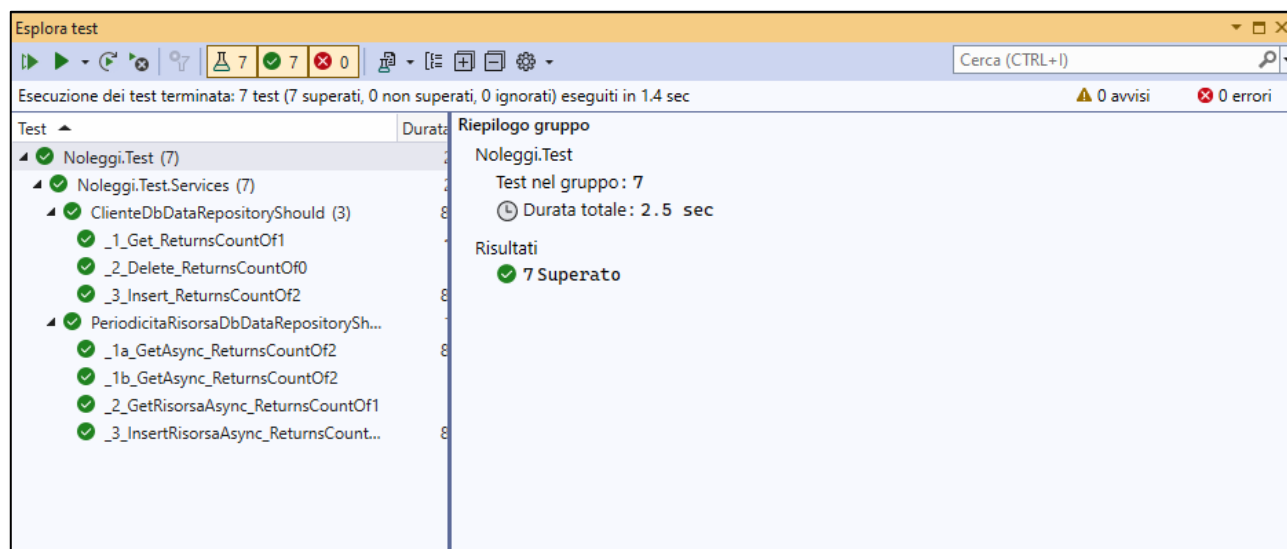


Figura 39: UnitTest

6.4 Mancanze/limitazioni conosciute

Nel QdC era richiesta una macchina con IIS (Internet Information Services), per gestire il mio applicativo. Non sono riuscito a soddisfare questo requisito in quanto mi sono focalizzato molto di più sulla parte del codice e sul funzionamento dell'applicativo.

7 Consuntivo

Rispetto alla pianificazione preventiva, nel Gantt consuntivo sono cambiati diversi tempi di diverse attività, questo è dovuto al fatto che sono stati calcolati male i tempi e l'importanza di alcune attività. Nonostante ciò, sono comunque riuscito a stare nei tempi e a finire il progetto.

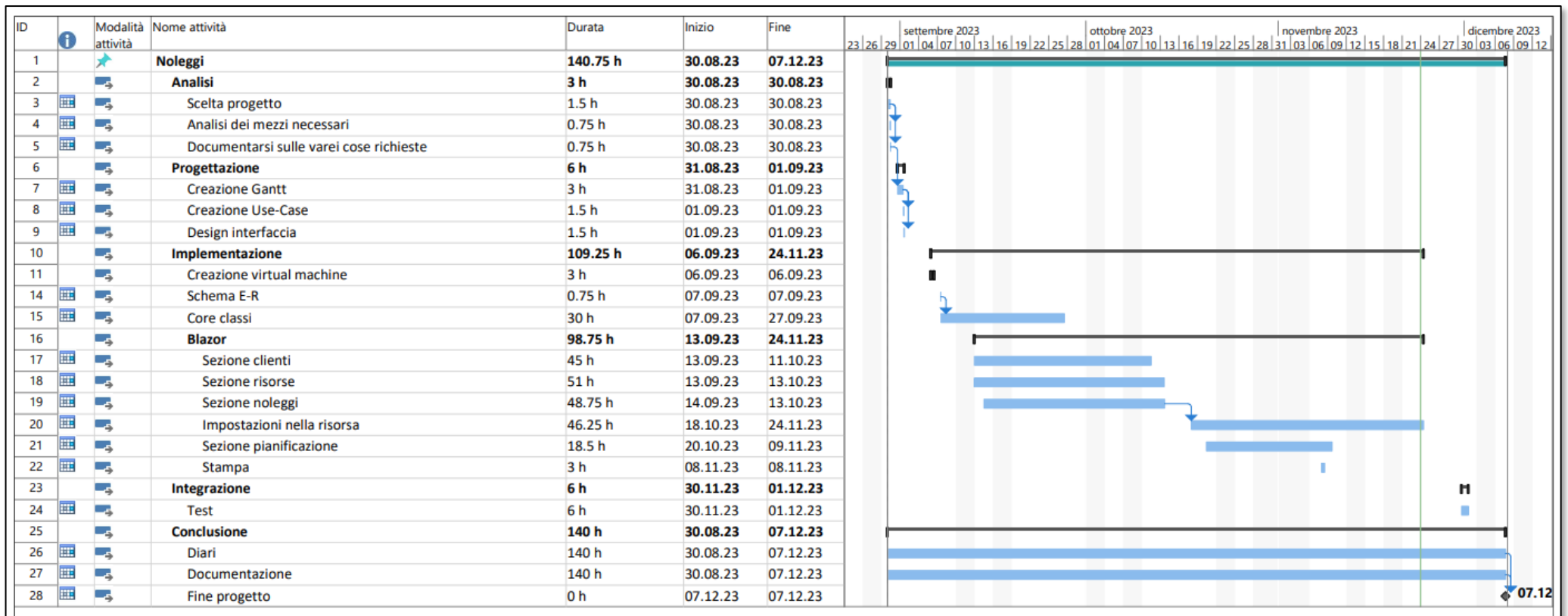


Figura 40: Gantt Consuntivo

8 Conclusioni

8.1 Sviluppi futuri

Sono molto soddisfatto del risultato ottenuto, ma ho pensato ad alcune migliorie:

- Una miglioria della grafica in generale.
- Aggiungere un attributo “quantità” alla tabella “Risorsa” per indicarne la quantità.
- Aggiungere l’eliminazione automatica di un noleggio, una volta che la risorsa è stata riportata.
- Quando la data di consegna si avvicina, si potrebbe automaticamente inviare un email al cliente dando un avviso.

8.2 Considerazioni personali

In conclusione, posso dire che questo progetto mi ha insegnato molto, sia a livello personale che a livello lavorativo. Sono contento di aver scelto questo lavoro e mi è piaciuto molto svolgerlo. A livello di tempistiche posso dire che sono soddisfatto del progetto, siccome sono riuscito a gestire al meglio tutto il tempo a mia disposizione. Infine, posso dire di aver lavorato con molto impegno e di aver ampliato e migliorato le mie conoscenze nell’ambito della programmazione.

9 Bibliografia

9.1 Sitografia

- <https://app.moqups.com/>, GUI, 01.09.2023
- <https://www.youtube.com/watch?v=TcH87xJVylw>, video tutorial, 07.09.2023
- <https://blazor.radzen.com/get-started>, Documentazione Radzen, 07.09.2023
- <https://www.youtube.com/watch?v=ijIlg2XAVK8>, Video per i components, 04.10.2023
- <https://blazor.radzen.com/templateform>, Parte grafica, 04.10.2023
- <https://blazor.radzen.com/dropdown>, Dropdown, 06.10.2023
- <https://blazor.radzen.com/datagrid-inline-edit>, Datagrid, 12.10.2023
- <https://blazor.radzen.com/requiredvalidator>, Controllo dati, 12.10.2023
- <https://blazor.radzen.com/scheduler>, Calendario, 25.10.2023
- <https://icon-sets.iconify.design/oi/page-5.html>, Icone, 27.10.2023

10 Glossario

| Termine | Significato |
|-----------------|---|
| .NET | Microsoft .NET: è una piattaforma di sviluppo general purpose, ideata e sviluppata da Microsoft, che mette a disposizione varie funzionalità come il supporto per più linguaggi di programmazione. |
| .NET Framework | .NET Framework: è l'ambiente di esecuzione runtime della piattaforma tecnologica .NET. |
| Blazor | Blazor: è un framework Web front-end .NET che supporta il rendering lato server e l'interattività client in un singolo modello di programmazione. |
| C# | C Sharp: è un linguaggio di programmazione del 2000 sviluppato da Mads Torgersen (Microsoft). |
| DbContext | DbContext: è una classe fornita da Entity Framework, un ORM (Object-Relational Mapping), che semplifica l'accesso e la gestione dei dati in un database relazionale in un'applicazione .NET. |
| Migrazione | Migration: rappresenta una modifica nel database, come ad esempio un'aggiunta, una modifica o l'eliminazione di una tabella o di una colonna. |
| Pacchetti NuGet | Pacchetti NuGet: sono una forma di distribuzione e gestione delle librerie e dei componenti di software in ambienti basati su .NET, come C#. |
| Radzen | Radzen: è una libreria che fornisce strumenti e componenti visivi per la progettazione, la creazione rapida di interfacce utente e la gestione dei dati, rendendo migliore lo sviluppo di applicazioni web e desktop |

11 Indice delle figure

| | |
|---|----|
| Figura 1: Use-case | 10 |
| Figura 2: Gantt Preventivo | 11 |
| Figura 3: Schema E-R | 13 |
| Figura 4: Pagina home..... | 14 |
| Figura 5: Pagina clienti | 15 |
| Figura 6: Pagina risorse | 16 |
| Figura 7: Pagina pianificazione | 17 |
| Figura 8: Pagina stampa | 18 |
| Figura 9: Pagina impostazioni | 19 |
| Figura 10: Diagramma delle classi | 20 |
| Figura 11: Diagramma delle interfacce | 21 |
| Figura 12: Diagramma delle classi repository..... | 21 |
| Figura 13: Creazione progetto core | 22 |
| Figura 14: Configurazione progetto core | 22 |
| Figura 15: Informazioni progetto core | 22 |
| Figura 16: Pacchetti installati..... | 23 |
| Figura 17: Struttura delle cartelle | 23 |
| Figura 18: Cartella models | 24 |
| Figura 19: Cartella configurations..... | 30 |
| Figura 20: Cartella services | 34 |
| Figura 21: Imposta progetto di avvio..... | 41 |
| Figura 22: Progetto predefinito..... | 41 |
| Figura 23: Cartella migrations | 41 |
| Figura 24: File .db | 41 |
| Figura 25: Creazione progetto blazor..... | 42 |
| Figura 26: Configurazioni progetto blazor | 42 |
| Figura 27: Informazioni progetto core | 42 |
| Figura 28: Struttura cartelle blazor | 43 |
| Figura 29: Pacchetto Radzen.Blazor | 43 |
| Figura 30: Cartella pages | 46 |
| Figura 31: Metodo EditRowAsync..... | 48 |
| Figura 32: Metodo InsertRowAsync | 48 |
| Figura 33: Popup di richiamo..... | 52 |
| Figura 34: Pagina home finale..... | 66 |
| Figura 35: Pagina clienti finale | 66 |
| Figura 36: Pagina risorse finale | 67 |
| Figura 37: Pagina noleggi finale | 67 |
| Figura 38: Pagina pianificazione finale..... | 68 |
| Figura 39: UnitTest..... | 75 |
| Figura 40: Gantt Consuntivo | 76 |

12 Allegati

- Mandato e/o QdC
- Diari di lavoro
- Manuale utente