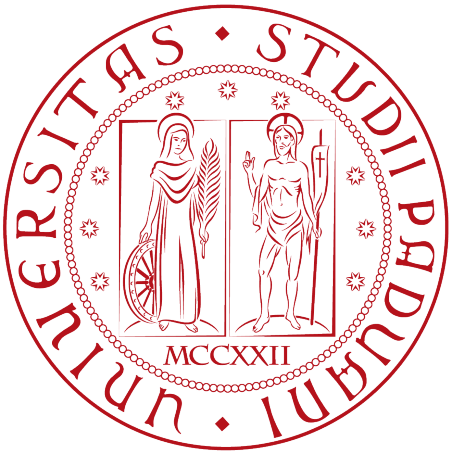
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di una piattaforma web e mobile per la gestione di eventi aziendali

Tesi di laurea

Relatrice
Prof.ssa Ombretta Gaggi

Laureando
Michele Bonavigo
1216752

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa 320 ore, dal laureando Michele Bonavigo presso l'azienda Ergon Informatica Srl dal 19/06/2023 al 25/08/2023.
Lo stage consiste nella progettazione e sviluppo di una piattaforma finalizzata alla gestione di eventi, affrontando tematiche relative allo sviluppo *mobile cross-platform*, *web services* e *web app*.

"Se camminassimo solo nelle giornate di sole non raggiungeremmo mai la nostra destinazione"

— Paulo Coelho

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof.ssa Ombretta Gaggi, relatrice della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori, mio fratello e mia sorella per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Infine, desidero ringraziare i miei nonni per tutto il supporto che mi hanno dato e per aver sempre creduto in me.

Padova, Settembre 2023 Michele Bonavigo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

Ergon Informatica Srl è una società italiana operante nel settore IT dal 1988, con sede a Castelfranco Veneto (TV).
Essa si occupa principalmente di soluzioni gestionali per piccole e medie imprese e dello sviluppo di *software* *ERP*[g] per i settori dell'alimentare e dei trasporti, ma completa l'offerta fornendo ai propri clienti prodotti *hardware*, servizi *web* e *hosting*, nonché progetti di *Server Consolidation*[g] e virtualizzazione dei sistemi.
Il logo dell'azienda è riportato in figura 1.1.

Figura 1.1: Logo Ergon Informatica Srl

1.2 Descrizione dello stage

1.2.1 Introduzione

Lo *stage* proposto consiste nell'implementare una piattaforma di gestione degli eventi a supporto di *Erglink*[g]. *Erglink* è una piattaforma social che un'azienda rende disponibile ai propri clienti per creare un canale di comunicazione diretto tra l'azienda e il cliente, un luogo dove condividere iniziative, promozioni, ecc. Nella nuova piattaforma di gestione eventi l'azienda potrà creare nuovi eventi ed invitare tutti i propri clienti. Gli eventi verranno gestiti attraverso un *backoffice web* dove sarà possibile creare nuovi eventi, gestire quelli esistenti e le prenotazioni effettuate dai clienti. Ogni evento avrà degli attributi, come ad esempio titolo, descrizione, numero massimo di biglietti, numero massimo di biglietti per utente, fasce orarie, ecc. Alla creazione dell'evento, tutti gli invitati riceveranno una notifica nel proprio *smartphone* e potranno iscriversi all'evento direttamente dall'*app*. Successivamente, oltre a vedere la prenotazione nell'*app*, riceveranno via *mail* una conferma di avvenuta registrazione. Il progetto prevede sia lo sviluppo dell'interfaccia utente lato *backoffice web* sia delle funzionalità nell'*app mobile*.

1.2.2 Obiettivi

Nella tabella 1.1 sono riportati gli obiettivi previsti dallo *stage*, dove i codici con prefisso *OB* e *DE* rappresentano rispettivamente gli obiettivi obbligatori e desiderabili.

Tabella 1.1: Tabella degli obiettivi

Codice obiettivo	Descrizione obiettivo
OB1	Sviluppo dei *web services* per la gestione del flusso di dati
OB2	Sviluppo *app* per iscrizione eventi
OB3	Sviluppo interfaccia *web* per creazione degli eventi
OB4	Acquisizione di competenze sullo sviluppo *cross-platform mobile*, interfacce *web* e *web services*

1.3 Organizzazione del testo

Il secondo capitolo approfondisce lo studio di fattibilità effettuato durante scelta della tecnologia da utilizzare per lo sviluppo dell'*app mobile cross-platform*.

Il terzo capitolo descrive l'analisi dei requisiti del progetto, comprensiva di diagrammi dei casi d'uso e tracciamento dei requisiti.

Il quarto capitolo descrive la fase di progettazione del progetto, comprensiva di diagrammi delle classi.

Il quinto capitolo approfondisce la fase di codifica del progetto.

Il sesto capitolo descrive i processi di verifica e validazione del progetto.

Il settimo capitolo presenta le conclusioni tratte dallo *stage*.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Studio di fattibilità

2.1 Introduzione

La prima fase dello stage consisteva in uno studio di fattibilità, con lo scopo di decidere se sviluppare le funzionalità nell'*app* per la gestione degli eventi con la tecnologia *Xamarin* o con la nuova tecnologia *.NET MAUI*, successore di *Xamarin*. Entrambe le tecnologie consentono lo sviluppo di applicazioni in C#[g] e XAML[g] per *iOS*, *Android* e *Windows* con il *framework .NET*, condividendo lo stesso codice, logica di *business* e test tra le varie piattaforme.

2.2 Xamarin

Xamarin è una piattaforma open source[g] nata nel 2011 e successivamente acquistata da *Microsoft* nel 2016 per la creazione di applicazioni per *iOS*, *Android* e *Windows*. *Xamarin* può essere quindi visto come uno strato di astrazione che gestisce la comunicazione del codice condiviso con il codice nativo della piattaforma sottostante. Ciò consente agli sviluppatori di condividere in media il 90% del codice della loro applicazione tra le varie piattaforme.

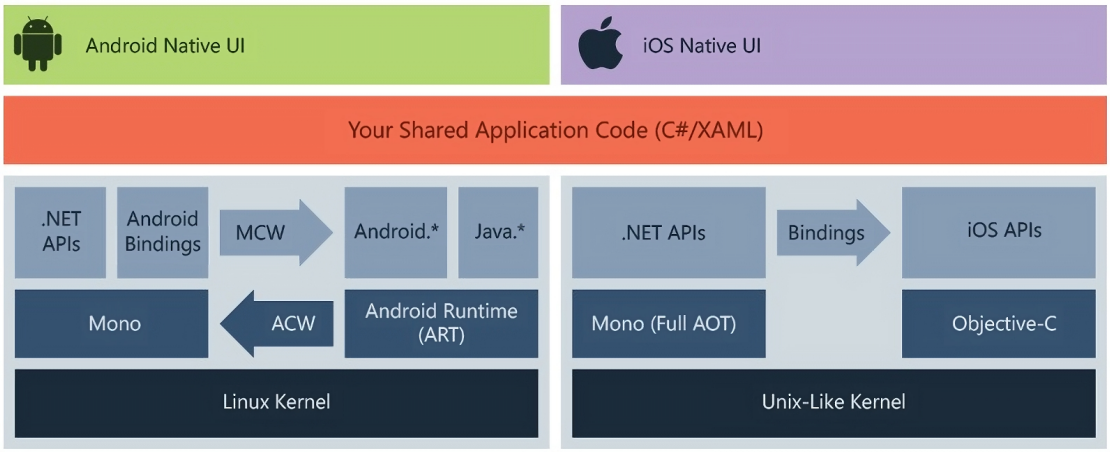


Figura 2.1: Architettura *Xamarin*

Come si può notare in figura 2.1, *Xamarin* richiede la scrittura di un'interfaccia grafica nativa per ogni piattaforma, rendendo quindi lo sviluppo ancora legato alle piattaforme di destinazione dell'applicazione.
Questo è stato risolto con l'introduzione di *Xamarin.Forms*, un *framework* che permette di condividere anche il *layout* e la progettazione dell'interfaccia utente tra le piattaforme.

Infatti, come si può vedere in figura 2.2, in fase di esecuzione *Xamarin.Forms* utilizza i *renderer* delle singole piattaforme per convertire gli elementi dell'interfaccia utente multipiattaforma in controlli nativi in *Xamarin.Android*, *Xamarin.iOS* e *UWP*. Ciò consente agli sviluppatori di ottenere l'aspetto e le prestazioni nativi continuando a usufruire dei vantaggi della condivisione del codice tra le piattaforme.

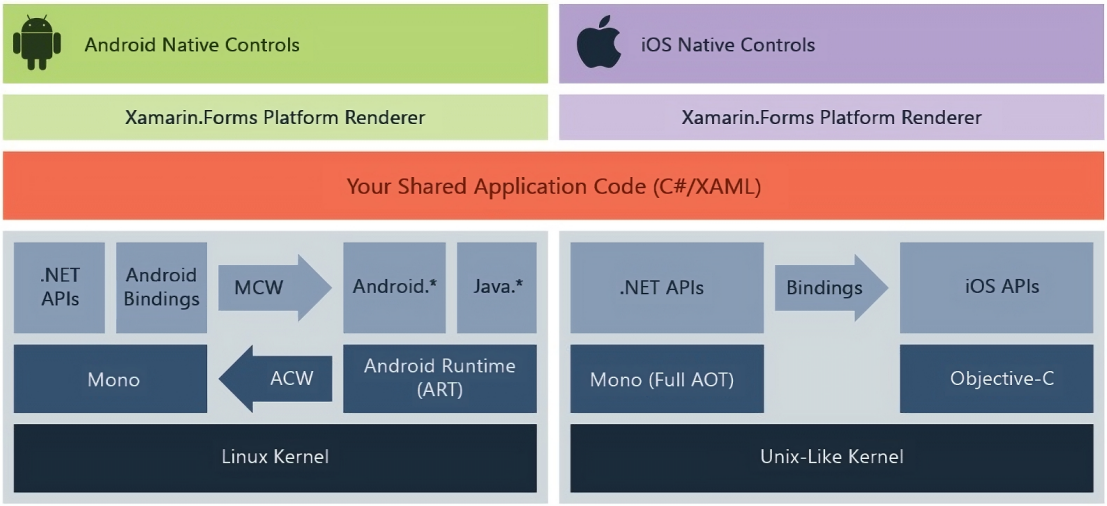


Figura 2.2: Architettura *Xamarin.Forms*

2.2.1 Gestione delle differenze delle funzionalità

La differenza di funzionalità non è solo un problema tra piattaforme, perché anche dispositivi della stessa piattaforma possono avere funzionalità diverse, soprattutto l'ampia gamma di dispositivi *Android* disponibili. Un esempio può essere la grandezza dello schermo, ma possono variare anche altre caratteristiche di un dispositivo e ciò richiede all'applicazione di verificare la presenza o l'assenza di certe funzionalità e comportarsi di conseguenza. In generale, si possono individuare tre categorie di funzionalità dei dispositivi:

Funzionalità fondamentali *cross-platform*: sono quelle caratteristiche che generalmente sono universali per tutti i dispositivi, ad esempio la possibilità di utilizzare:

- visualizzazione a schede o menù;
- visualizzazione di elenchi di dati con scorrimento;
- navigazione all'indietro tra le schermate.

Funzionalità specifiche per piattaforma: sono quelle caratteristiche che possono essere diverse in base alla piattaforma di riferimento, ad esempio:

- tastiera, che può essere diversa nelle varie piattaforme o addirittura fisica;
- notifiche *push*, che possono avere funzionalità e/o implementazioni diverse;
- *touch* e *gestures*, che possono essere gestite diversamente soprattutto nei dispositivi più datati che ne hanno un supporto limitato.

Funzionalità specifiche per dispositivo: sono quelle caratteristiche che variano in base al dispositivo, ad esempio:

- fotocamera, che può avere funzionalità diverse o addirittura può non essere presente;
- accelerometro, giroscopio e bussola, che non sempre sono presenti nel dispositivo;
- geolocalizzazione, che può non essere presente;
- NFC[g], che non è spesso disponibile nei dispositivi.

Per gestire tutte queste differenze, può diventare necessaria la scrittura di codice specifico per la piattaforma e/o dispositivo di riferimento, il che va contro al principio fondamentale di *Xamarin*, ovvero quello di avere un unico codice condiviso per tutte le varie piattaforme target.
Per risolvere ciò è stata introdotta la libreria *Xamarin.Essentials*, che fornisce agli sviluppatori delle API[g] multipiattaforma per semplificare l'accesso alle funzionalità dei diversi dispositivi nativi. Tra le molte funzionalità supportate da *Xamarin.Essentials* ci sono:

- accelerometro;
- appunti;
- barometro;
- batteria, per rilevarne facilmente il livello, la fonte e lo stato;
- flash della fotocamera;
- geolocalizzazione;
- gestione dei permessi;
- informazioni sul dispositivo;
- *screenshot*;
- vibrazione.

2.2.2 Aspetti negativi

Il *framework Xamarin* sicuramente semplifica lo sviluppo di applicazioni multipiattaforma, ma presenta comunque degli aspetti negativi che vanno considerati durante la scelta della tecnologia da utilizzare nel progetto di *stage*:

- *multiple projects targeting multiple platforms*: la soluzione dell'applicazione necessita di un progetto per ogni piattaforma *target*, il che rende la struttura generale abbastanza complessa;
- ogni progetto per piattaforma gestisce le proprie risorse (immagini, icone, ...), quindi nella maggior parte dei casi sono duplicate. Questo però può anche essere utile ad esempio se si necessita di immagini con risoluzioni diverse per piattaforma;
- *Microsoft* terminerà il supporto a *Xamarin* nel 2024, quindi in futuro sarà sicuramente necessaria una migrazione.

2.3 .NET MAUI

.NET MAUI è un *framework* open source multipiattaforma che nasce nel 2022 come evoluzione di *Xamarin.Forms*, esteso dai dispositivi mobili ai *desktop*, con controlli dell'interfaccia utente ricompilati da zero per prestazioni ed estendibilità.

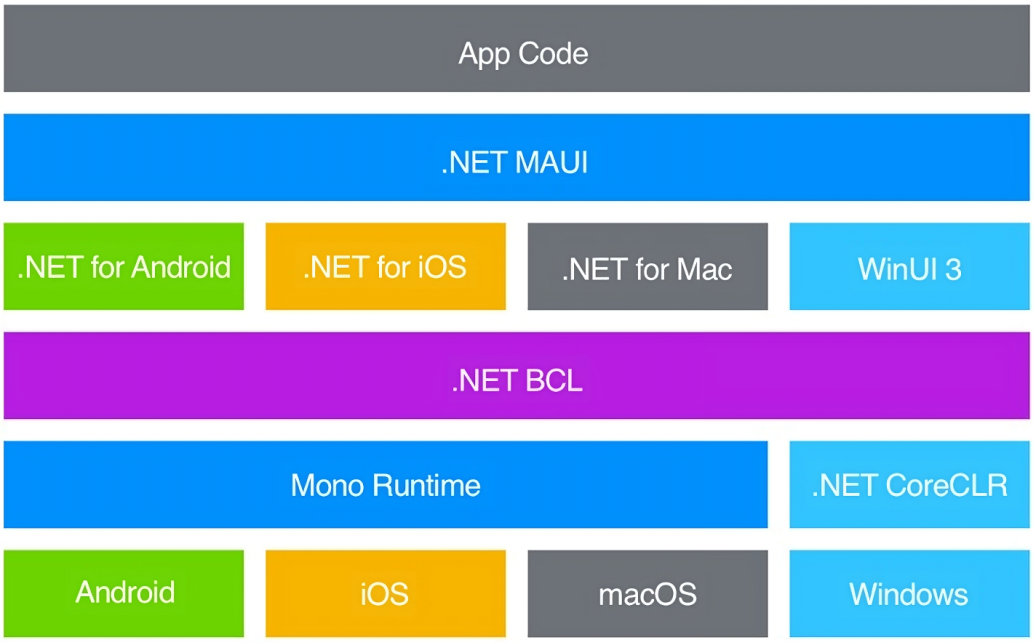


Figura 2.3: Architettura *.NET MAUI*

2.3.1 Miglioramenti rispetto a Xamarin

Essendo *.NET MAUI* la diretta evoluzione di *Xamarin*, mantiene lo stesso obiettivo di poter sviluppare *app* che possono essere eseguite su piattaforme diverse (*Android*, *iOS*, *macOS* e *Windows*) da un unico codice sorgente condiviso, migliorandone però alcuni aspetti:

- come si può notare in figura 2.3, *.NET MAUI* offre un singolo framework per la creazione di interfacce utente per le *app* per dispositivi mobili e desktop. Principalmente, il codice scritto interagisce con l'API *.NET MAUI*, che a sua volta utilizza le API della piattaforma nativa. Inoltre, il codice dell'*app* può anche interagire direttamente con le API della piattaforma nativa;
- *single project targeting multiple platforms*: a differenza di *Xamarin*, *.NET MAUI* utilizza un unico progetto che gestisce tutte le piattaforme *target*, semplificandone così la struttura e la gestione;
- essendoci un unico progetto, viene risolto il problema delle risorse duplicate per ogni piattaforma, ma rimane comunque la possibilità di introdurre delle risorse specifiche per una o più piattaforme;
- unifica diverse librerie, tra cui *Xamarin.Essentials*, in modo da astrarre il più possibile le caratteristiche di piattaforme e/o dispositivi diversi;
- introduce nuovi *pattern* architetturali, tra cui *MVU(Model-View-Update)*, oltre al già presente in *Xamarin MVVM(Model-View-ViewModel)*;

- come si può vedere nelle figure 2.4 e 2.5, *.NET MAUI* introduce un nuovo *layer* astratto di interfacce di controlli, aumentando il disaccoppiamento con l'interfaccia utente nativa. In questo modo sarà molto più semplice per lo sviluppatore modificare un *handler* per includere/escludere nuove piattaforme.

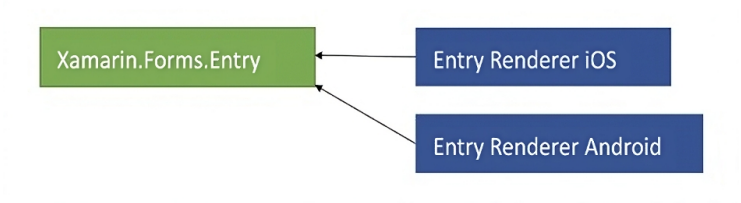


Figura 2.4: *Xamarin.Forms Renderers*

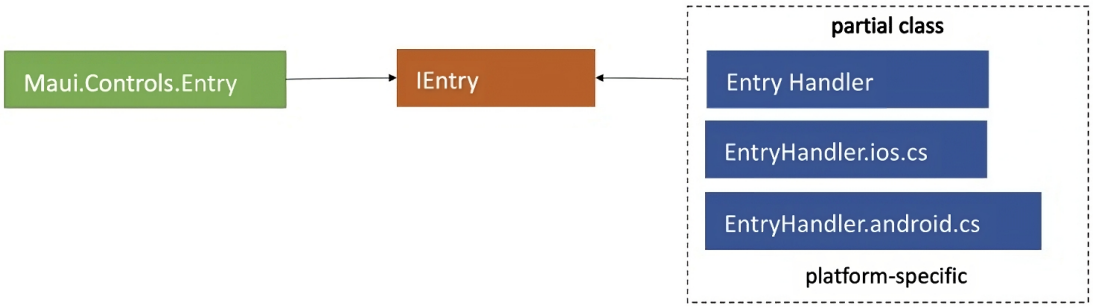


Figura 2.5: *.NET MAUI Handlers*

2.3.2 Aspetti negativi

Purtroppo, essendo stata rilasciata nel 2022, presenta ancora numerosi *bug* che vengono segnalati nella pagina github di *.NET MAUI*.

2.4 Conclusioni

.NET MAUI si presenta sicuramente come un *framework* interessante, perché promette di semplificare la scrittura di applicazioni per piattaforme diverse, evitando quindi il più possibile il codice nativo. Purtroppo però, data la presenza di numerosi *bug*, risulta ancora troppo poco matura per un contesto aziendale dove è necessaria la stabilità che in questo caso offre *Xamarin*. Considerando inoltre che la migrazione da *Xamarin* a *.NET MAUI* non è ancora obbligatoria e/o urgente, è stato ritenuto opportuno continuare lo sviluppo dell'applicazione Erglink utilizzando *Xamarin*, in attesa che *.NET MAUI* diventi abbastanza maturo e stabile da rendere opportuna una migrazione.

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo UML[g] dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.
Per identificare i casi d'uso è stata scelta la seguente codifica:

UC<CodicePadre>.<CodiceFiglio>

È importante ribadire come questo formalismo sia gerarchico, ovvero un codice figlio può essere codice padre di un suo eventuale codice figlio. Possono essere figli le generalizzazioni e i sottocasi d'uso.

3.1.1 Attori

Gli attori rappresentano gli utenti che interagiscono con il sistema. Per questo progetto sono stati individuati tre tipologie di attori:

- **Utente C:** rappresenta l'utente base dell'*app* Erglink, ovvero colui che può iscriversi agli eventi;
- **Utente LDI:** rappresenta l'utente dipendente dell'*app* Erglink, ovvero colui che può validare i biglietti degli utenti **C**;
- **Utente CAM:** rappresenta l'utente del *backoffice*, ovvero colui che può creare e gestire gli eventi.

Dato che l'utente **C** e l'utente **LDI** condividono alcune funzionalità, come la possibilità di visualizzare gli eventi disponibili, è stato introdotto un utente generico, che rappresenta la generalizzazione degli altri due tipi di utenti dell'*app*.

3.1.2 App

Nella figura 3.1 è riportato il diagramma del sistema principale del modulo per la gestione degli eventi dell'*app* Erglink, con tutti i casi d'uso.

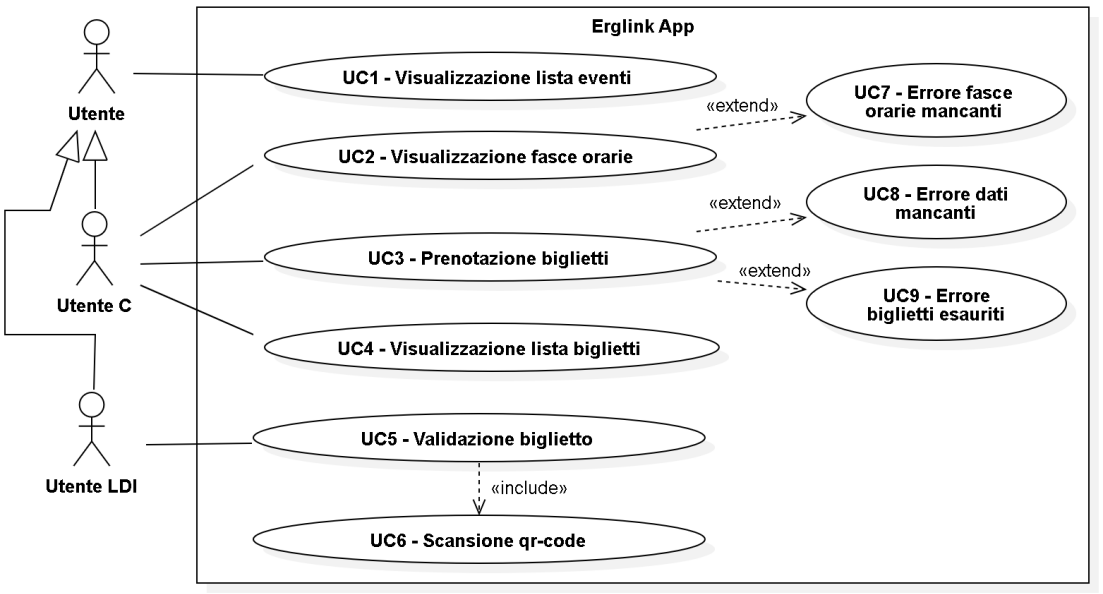


Figura 3.1: Sistema principale *app*

UC1 - Visualizzazione lista eventi

In figura 3.2 è rappresentato il caso d'uso tramite diagramma UML.

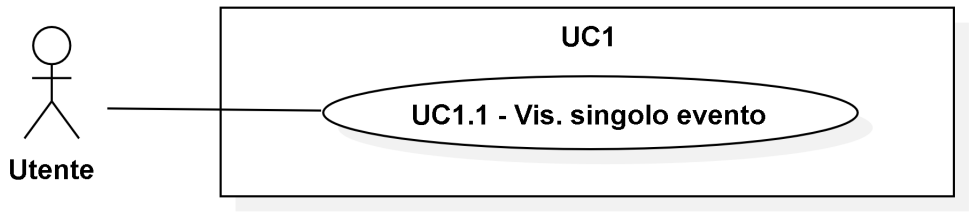


Figura 3.2: UC1 - Visualizzazione lista eventi

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente ha effettuato il *login* e vuole visualizzare la lista degli eventi disponibili.
- **Scenario principale:**
  1. L'utente tocca il pulsante degli eventi sulla barra del menu di navigazione.
- **Postcondizioni:** L'utente visualizza la lista degli eventi disponibili.

UC1.1 - Visualizzazione singolo evento

In figura 3.3 è rappresentato il caso d'uso tramite diagramma UML.

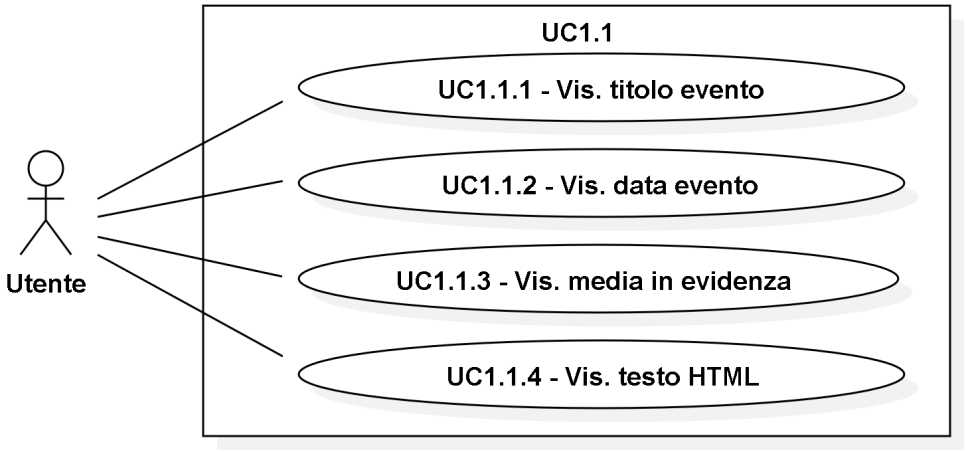


Figura 3.3: UC1.1 - Visualizzazione singolo evento

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente visualizza correttamente la lista degli eventi.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del singolo evento.
- **Postcondizioni:** L'utente visualizza il singolo evento.

UC1.1.1 - Visualizzazione titolo evento

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del titolo dell'evento.
- **Postcondizioni:** L'utente visualizza il titolo dell'evento.

UC1.1.2 - Visualizzazione data evento

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data dal quale è possibile visualizzare l'evento.
- **Postcondizioni:** L'utente visualizza la data dell'evento.

UC1.1.3 - Visualizzazione *media* in evidenza

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del *media* in evidenza dell'evento.
- **Postcondizioni:** L'utente visualizza il *media* in evidenza dell'evento.

UC1.1.4 - Visualizzazione testo *HTML*

- **Attori Primari:** Utente.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del testo *HTML* dell'evento.
- **Postcondizioni:** L'utente visualizza il testo *HTML* dell'evento.

UC2 - Visualizzazione fasce orarie

In figura 3.4 è rappresentato il caso d'uso tramite diagramma UML.

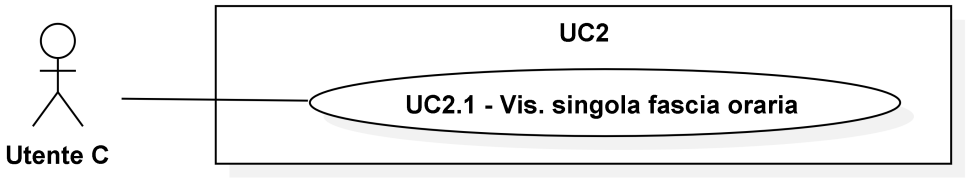


Figura 3.4: UC2 - Visualizzazione fasce orarie

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente vuole iniziare la prenotazione per un evento e vuole visualizzare la lista delle fasce orarie.
- **Scenario principale:**
  1. L'utente preme il pulsante "PRENOTA BIGLIETTI" nel singolo evento.
- **Postcondizioni:** L'utente visualizza la lista delle fasce orarie disponibili.
- **Scenario alternativo:**
  1. Viene segnalato un errore se non sono presenti fasce orarie (UC7).

UC2.1 - Visualizzazione singola fascia oraria

In figura 3.5 è rappresentato il caso d'uso tramite diagramma UML.

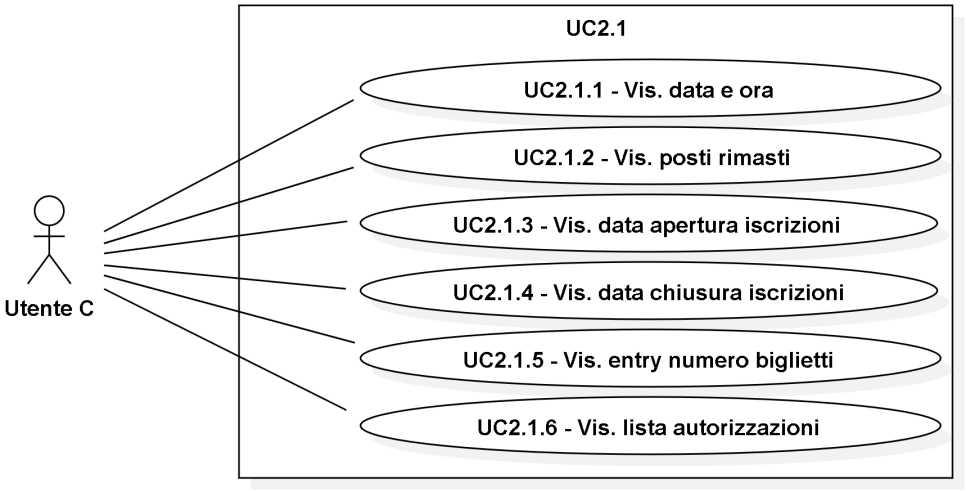


Figura 3.5: UC2.1 - Visualizzazione singola fascia oraria

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la lista delle fasce orarie.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione di una singola fascia oraria.
- **Postcondizioni:** L'utente visualizza la singola fascia oraria.

UC2.1.1 - Visualizzazione data e ora

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data e ora.
- **Postcondizioni:** L'utente visualizza la data e ora.

UC2.1.2 - Visualizzazione posti rimasti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del numero di posti rimasti.
- **Postcondizioni:** L'utente visualizza il numero di posti rimasti.

UC2.1.3 - Visualizzazione data apertura iscrizioni

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data di apertura delle iscrizioni.
- **Postcondizioni:** L'utente visualizza la data di apertura delle iscrizioni.

UC2.1.4 - Visualizzazione data chiusura iscrizioni

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data di chiusura delle iscrizioni.
- **Postcondizioni:** L'utente visualizza la data di chiusura delle iscrizioni.

UC2.1.5 - Visualizzazione *entry* numero biglietti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione dell'*entry* per il numero dei biglietti da prenotare.
- **Postcondizioni:** L'utente visualizza l'*entry* per il numero di biglietti da prenotare.

UC2.1.6 - Visualizzazione lista autorizzazioni

In figura 3.6 è rappresentato il caso d'uso tramite diagramma UML.

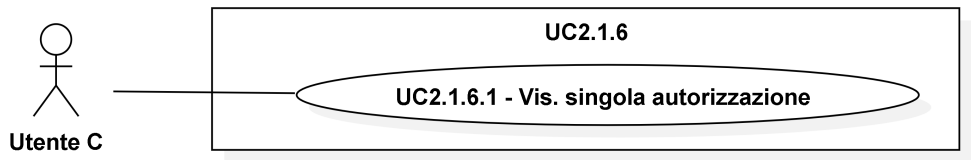


Figura 3.6: UC2.1.6 - Visualizzazione lista autorizzazioni

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della lista delle autorizzazioni da accettare.
- **Postcondizioni:** L'utente visualizza la lista delle autorizzazioni da accettare.

UC2.1.6.1 - Visualizzazione singola autorizzazione

In figura 3.7 è rappresentato il caso d'uso tramite diagramma UML.

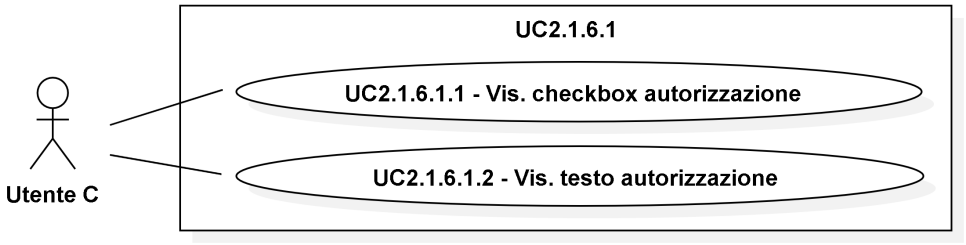


Figura 3.7: UC2.1.6.1 - Visualizzazione singola autorizzazione

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la lista di autorizzazioni.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione di una singola autorizzazione.
- **Postcondizioni:** L'utente visualizza la singola autorizzazione.

UC2.1.6.1.1 - Visualizzazione *checkbox* autorizzazione

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola autorizzazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della *checkbox* dell'autorizzazione.
- **Postcondizioni:** L'utente visualizza la *checkbox* dell'autorizzazione.

UC2.1.6.1.2 - Visualizzazione testo autorizzazione

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la singola autorizzazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del testo dell'autorizzazione.
- **Postcondizioni:** L'utente visualizza il testo dell'autorizzazione.

UC3 - Prenotazione biglietti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente ha selezionato la fascia oraria e vuole prenotare dei biglietti.
- **Scenario principale:**
  1. L'utente inserisce il numero di biglietti da prenotare;
  2. l'utente accetta le autorizzazioni dell'evento;

3. l'utente preme il pulsante "PRENOTA BIGLIETTI" nel singolo evento.

- **Postcondizioni:** L'utente viene avvisato della prenotazione andata a buon fine.
- **Scenario alternativo:**
  1. Viene segnalato un errore se non sono stati inseriti tutti i dati (UC8);
  2. viene segnalato un errore se non ci sono abbastanza biglietti (UC9).

UC4 - Visualizzazione lista biglietti

In figura 3.8 è rappresentato il caso d'uso tramite diagramma UML.

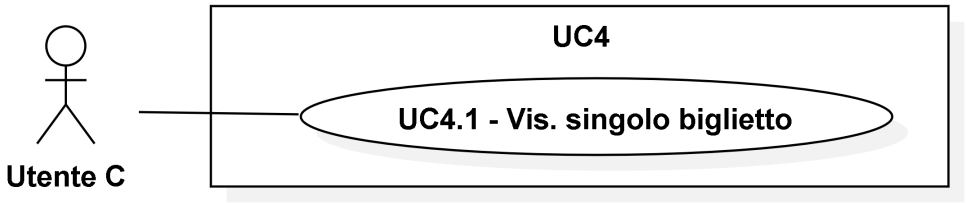


Figura 3.8: UC4 - Visualizzazione lista biglietti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente vuole visualizzare la lista di biglietti prenotati per un evento.
- **Scenario principale:**
  1. L'utente preme il pulsante "VISUALIZZA BIGLIETTI" nel singolo evento.
- **Postcondizioni:** L'utente visualizza la lista dei biglietti prenotati.

UC4.1 - Visualizzazione singolo biglietto

In figura 3.9 è rappresentato il caso d'uso tramite diagramma UML.

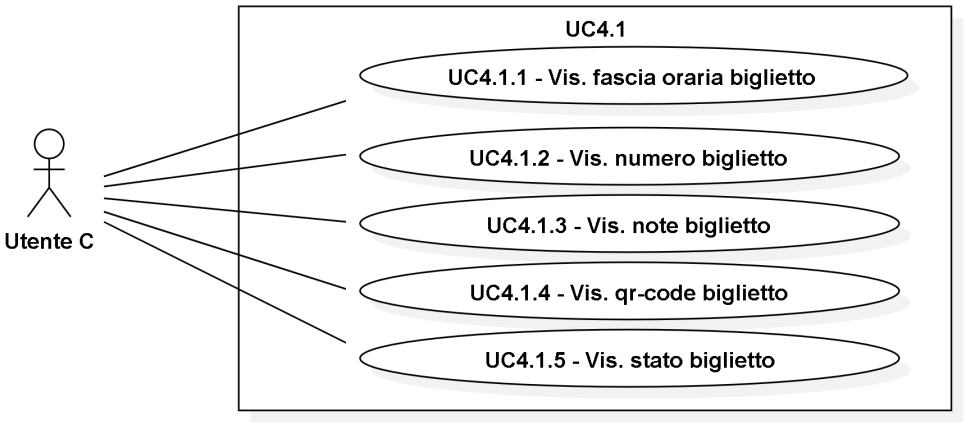


Figura 3.9: UC4.1 - Visualizzazione singolo biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente la lista dei biglietti.
- **Scenario principale:**

1. L'utente richiede la visualizzazione di un singolo biglietto.

- **Postcondizioni:** L'utente visualizza il singolo biglietto.

UC4.1.1 - Visualizzazione fascia oraria biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente il singolo biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data e ora della fascia oraria del biglietto.
- **Postcondizioni:** L'utente visualizza la data e ora della fascia oraria del biglietto.

UC4.1.2 - Visualizzazione numero biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente il singolo biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del numero del biglietto.
- **Postcondizioni:** L'utente visualizza il numero del biglietto.

UC4.1.3 - Visualizzazione note biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente il singolo biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione le note del biglietto.
- **Postcondizioni:** L'utente visualizza le note del biglietto.

UC4.1.4 - Visualizzazione *qr-code* biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente il singolo biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del *qr-code* del biglietto.
- **Postcondizioni:** L'utente visualizza il *qr-code* del biglietto.

UC4.1.5 - Visualizzazione stato biglietto

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente visualizza correttamente il singolo biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione dello stato del biglietto.
- **Postcondizioni:** L'utente visualizza lo stato del biglietto.

UC5 - Validazione biglietto

In figura 3.8 è rappresentato il caso d'uso tramite diagramma UML.

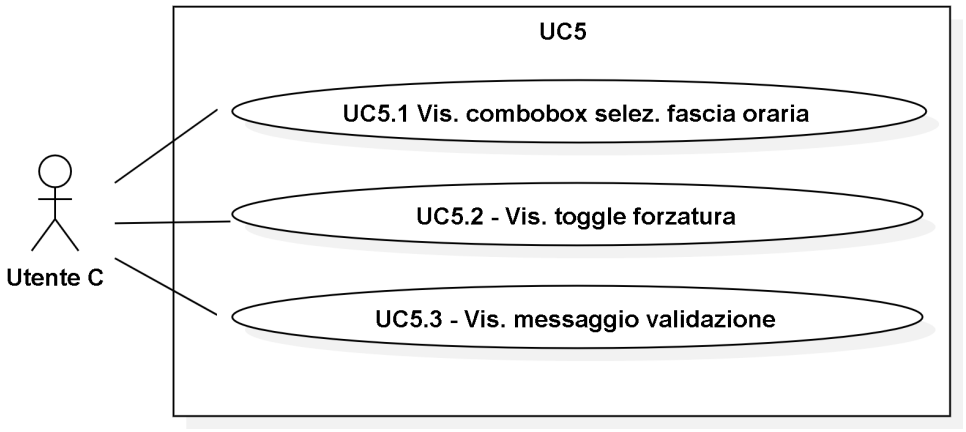


Figura 3.10: UC5 - Validazione biglietto

- **Attori Primari:** Utente LDI.
- **Precondizioni:** L'utente vuole validare un biglietto.
- **Scenario principale:**
  1. L'utente preme il pulsante "VALIDAZIONE BIGLIETTI" nel singolo evento;
  2. l'utente seleziona la fascia oraria di riferimento;
  3. l'utente decide se forzare o meno la validazione;
  4. l'utente scansiona il *qr-code* (UC6).
- **Postcondizioni:** L'utente visualizza il messaggio di eventuale validazione.

UC5.1 - Visualizzazione *combobox* fascia oraria

- **Attori Primari:** Utente LDI.
- **Precondizioni:** L'utente vuole validare un biglietto.
- **Scenario principale:**
  1. L'utente richiede di poter selezionare la fascia oraria di riferimento.
- **Postcondizioni:** L'utente può selezionare la fascia oraria di riferimento.

UC5.2 - Visualizzazione *toggle* forzatura

- **Attori Primari:** Utente LDI.
- **Precondizioni:** L'utente vuole validare un biglietto.
- **Scenario principale:**
  1. L'utente richiede di poter scegliere se forzare o meno la validazione.
- **Postcondizioni:** L'utente può forzare la validazione di un biglietto.

UC5.3 - Visualizzazione messaggio validazione

- **Attori Primari:** Utente LDI.
- **Precondizioni:** L'utente vuole validare un biglietto.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del messaggio di eventuale validazione.
- **Postcondizioni:** L'utente visualizza il messaggio di eventuale validazione.

UC6 - Scansione *qr-code*

- **Attori Primari:** Utente LDI.
- **Precondizioni:** L'utente vuole scansionare un *qr-code* di un biglietto.
- **Scenario principale:**
  1. L'utente preme sul pulsante "SCANSIONA QR-CODE".
- **Postcondizioni:** L'utente scansiona il *qr-code*.

UC7 - Errore fasce orarie mancanti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente vuole iniziare la prenotazione per un evento e vuole visualizzare la lista delle fasce orarie.
- **Scenario principale:**
  1. L'utente preme il pulsante "PRENOTA BIGLIETTI" nel singolo evento.
- **Postcondizioni:** L'utente viene avvisato che non ci sono fasce orarie disponibili.

UC8 - Errore dati mancanti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente vuole prenotare dei biglietti.
- **Scenario principale:**
  1. L'utente preme il pulsante "PRENOTA" nella schermata di prenotazione.

- **Postcondizioni:** L'utente viene avvisato che ci sono alcuni dati mancanti.

UC9 - Errore biglietti esauriti

- **Attori Primari:** Utente C.
- **Precondizioni:** L'utente vuole prenotare dei biglietti.
- **Scenario principale:**
  1. L'utente preme il pulsante "PRENOTA" nella schermata di prenotazione.
- **Postcondizioni:** L'utente viene avvisato che non ci sono abbastanza biglietti per la sua prenotazione.

3.1.3 Backoffice

Nella figura 3.11 è riportato il diagramma del sistema principale del modulo per la gestione degli eventi del *backoffice*, con tutti i casi d'uso.

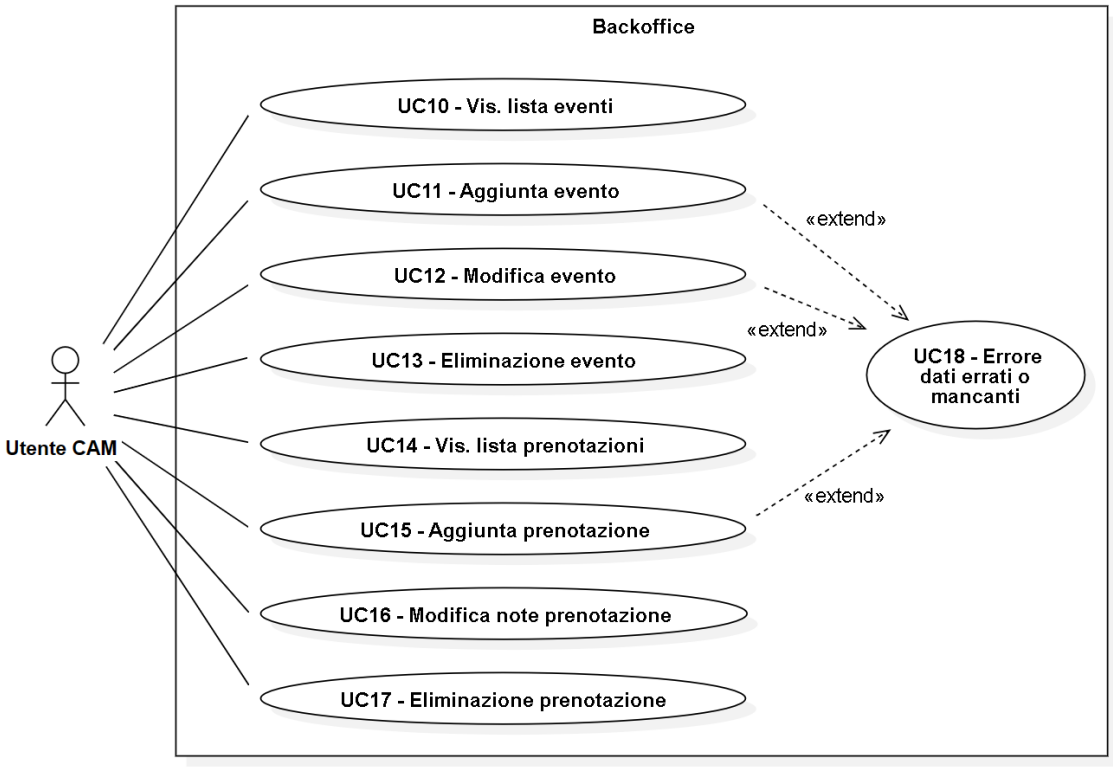


Figura 3.11: Sistema principale *backoffice*

UC10 - Visualizzazione lista eventi

In figura 3.12 è rappresentato il caso d'uso tramite diagramma UML.

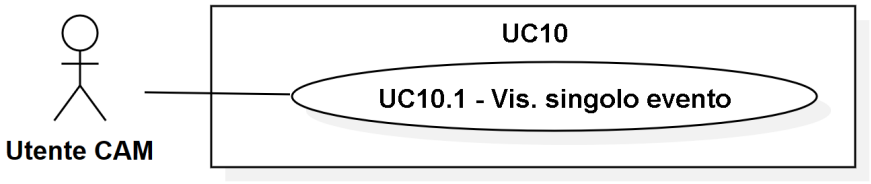


Figura 3.12: UC10 - Visualizzazione lista eventi

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente ha effettuato il login e vuole visualizzare la lista degli eventi disponibili.
- **Scenario principale:**
  1. L'utente clicca sul pulsante degli eventi sulla barra del menu di navigazione.
- **Postcondizioni:** L'utente visualizza la lista degli eventi disponibili.

UC10.1 - Visualizzazione singolo evento

In figura 3.13 è rappresentato il caso d'uso tramite diagramma UML.

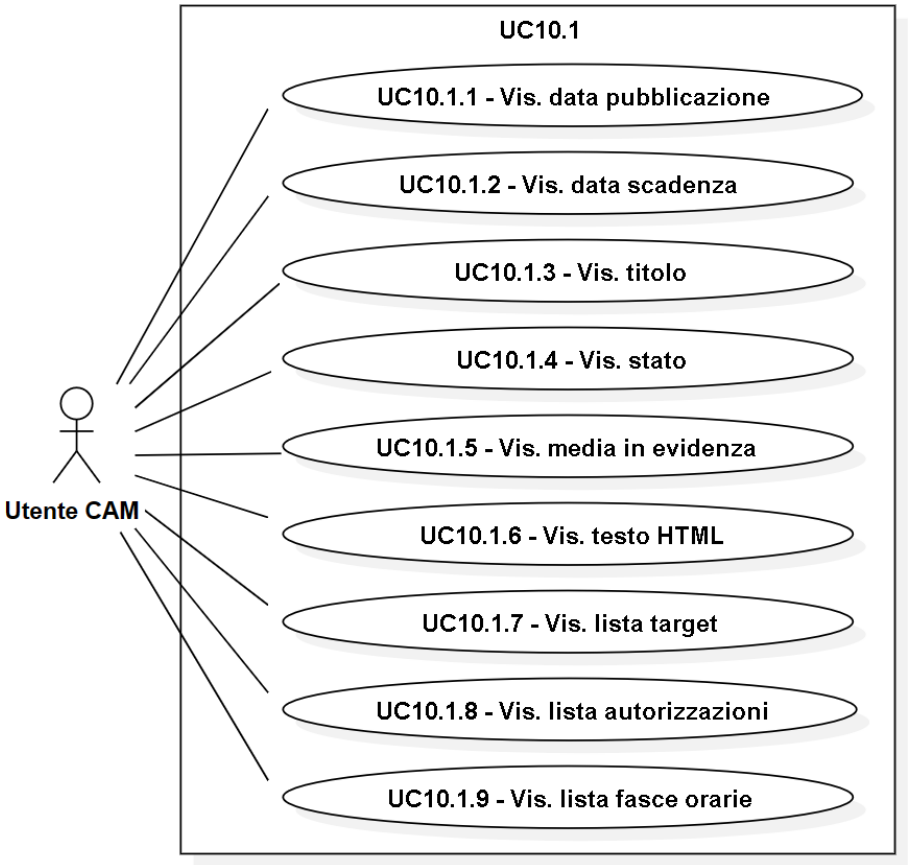


Figura 3.13: UC10.1 - Visualizzazione singolo evento

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la lista degli eventi.
- **Scenario principale:**
    1. L'utente richiede la visualizzazione del singolo evento.
- **Postcondizioni:** L'utente visualizza il singolo evento.

UC10.1.1 - Visualizzazione data pubblicazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
    1. L'utente richiede la visualizzazione della data di pubblicazione dell'evento.
- **Postcondizioni:** L'utente visualizza la data di pubblicazione dell'evento.

UC10.1.2 - Visualizzazione data scadenza

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
    1. L'utente richiede la visualizzazione della data di scadenza dell'evento.
- **Postcondizioni:** L'utente visualizza la data di scadenza dell'evento.

UC10.1.3 - Visualizzazione titolo

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
    1. L'utente richiede la visualizzazione del titolo dell'evento.
- **Postcondizioni:** L'utente visualizza il titolo dell'evento.

UC10.1.4 - Visualizzazione stato

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
    1. L'utente richiede la visualizzazione dello stato dell'evento.
- **Postcondizioni:** L'utente visualizza lo stato dell'evento.

UC10.1.5 - Visualizzazione *media* in evidenza

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del *media* in evidenza dell'evento.
- **Postcondizioni:** L'utente visualizza il *media* in evidenza dell'evento.

UC10.1.6 - Visualizzazione testo *HTML*

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del testo *HTML* dell'evento.
- **Postcondizioni:** L'utente visualizza il testo *HTML* dell'evento.

UC10.1.7 - Visualizzazione lista *target*

In figura 3.14 è rappresentato il caso d'uso tramite diagramma UML.

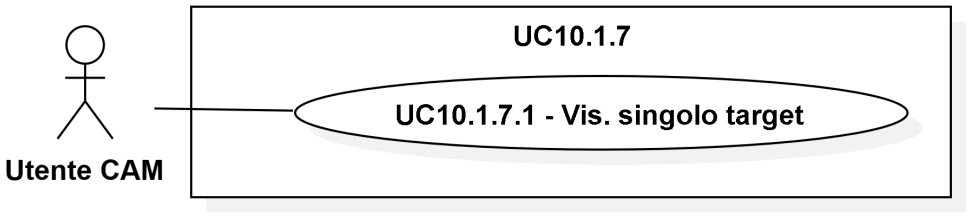


Figura 3.14: UC10.1.7 - Visualizzazione lista *target*

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della lista di *target* dell'evento.
- **Postcondizioni:** L'utente visualizza la lista di *target* dell'evento.

UC10.1.7.1 - Visualizzazione singolo *target*

In figura 3.15 è rappresentato il caso d'uso tramite diagramma UML.

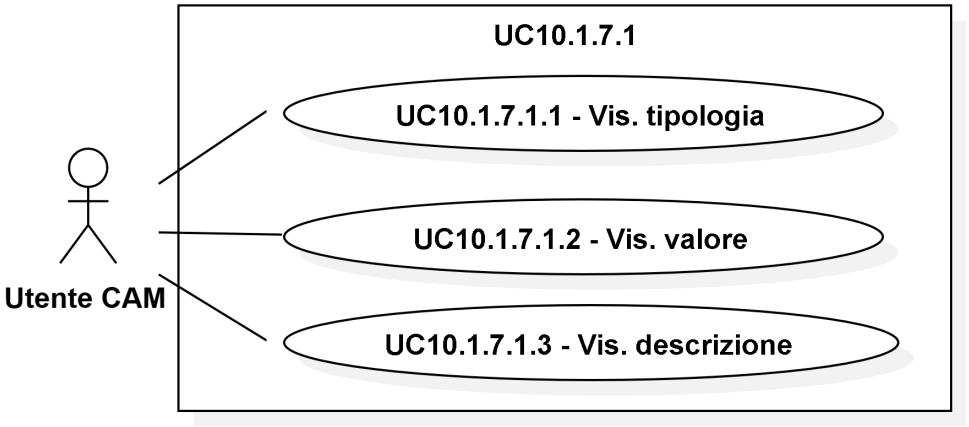


Figura 3.15: UC10.1.7.1 - Visualizzazione singolo *target*

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la lista di *target*.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del singolo *target*.
- **Postcondizioni:** L'utente visualizza il singolo *target*.

UC10.1.7.1.1 - Visualizzazione tipologia

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo *target*.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della tipologia del *target*.
- **Postcondizioni:** L'utente visualizza la tipologia del *target*.

UC10.1.7.1.2 - Visualizzazione valore

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo *target*.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del valore del *target*.
- **Postcondizioni:** L'utente visualizza il valore del *target*.

UC10.1.7.1.3 - Visualizzazione descrizione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo *target*.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della descrizione del *target*.
- **Postcondizioni:** L'utente visualizza la descrizione del *target*.

UC10.1.8 - Visualizzazione lista autorizzazioni

In figura 3.16 è rappresentato il caso d'uso tramite diagramma UML.

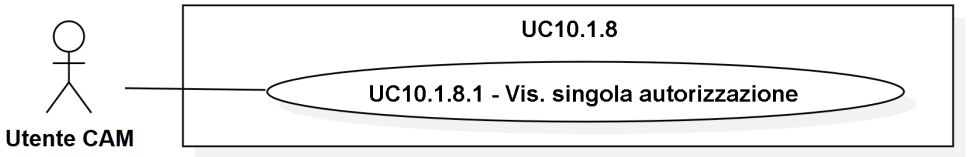


Figura 3.16: UC10.1.8 - Visualizzazione lista autorizzazioni

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della lista di autorizzazioni dell'evento.
- **Postcondizioni:** L'utente visualizza la lista di autorizzazioni dell'evento.

UC10.1.8.1 - Visualizzazione singola autorizzazione

In figura 3.17 è rappresentato il caso d'uso tramite diagramma UML.

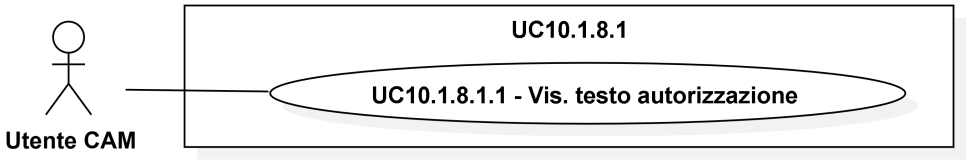


Figura 3.17: UC10.1.8.1 - Visualizzazione singola autorizzazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la lista di autorizzazioni.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della singola autorizzazione.
- **Postcondizioni:** L'utente visualizza la singola autorizzazione.

UC10.1.8.1.1 - Visualizzazione testo autorizzazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola autorizzazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del testo dell'autorizzazione.
- **Postcondizioni:** L'utente visualizza il testo dell'autorizzazione.

UC10.1.9 - Visualizzazione lista fasce orarie

In figura 3.18 è rappresentato il caso d'uso tramite diagramma UML.

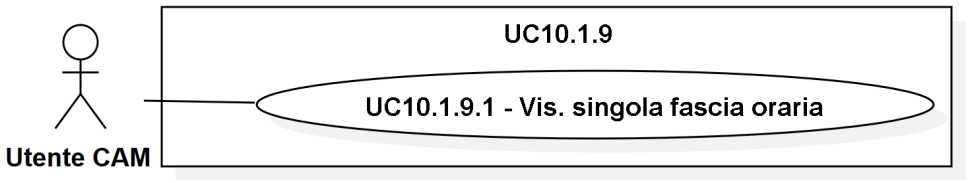


Figura 3.18: UC10.1.8 - Visualizzazione lista fasce orarie

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente il singolo evento.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della lista di fasce orarie dell'evento.
- **Postcondizioni:** L'utente visualizza la lista di fasce orarie dell'evento.

UC10.1.9.1 - Visualizzazione singola fascia oraria

In figura 3.19 è rappresentato il caso d'uso tramite diagramma UML.

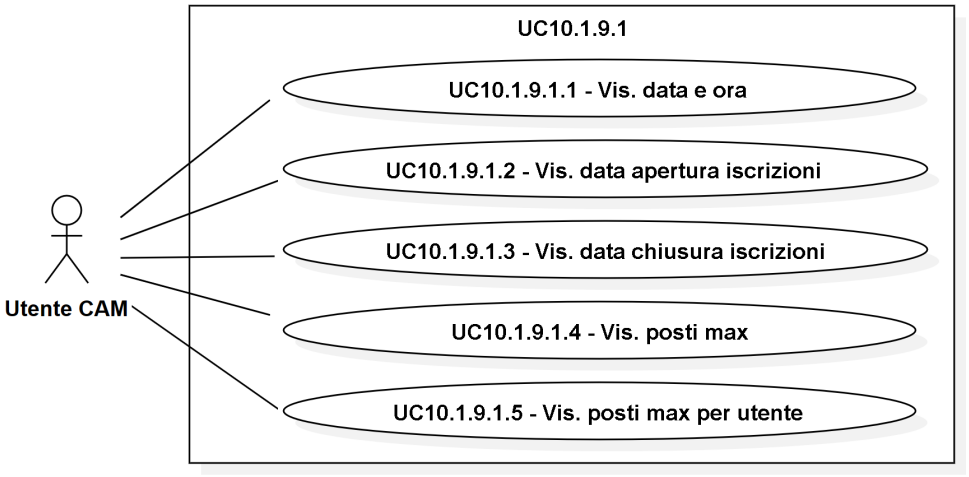


Figura 3.19: UC10.1.9.1 - Visualizzazione singola fascia oraria

- **Attori Primari:** Utente CAM.

- **Precondizioni:** L'utente visualizza correttamente la lista di fasce orarie.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della singola fascia oraria.
- **Postcondizioni:** L'utente visualizza la singola fascia oraria.

UC10.1.9.1.1 - Visualizzazione data e ora

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data e ora della fascia oraria.
- **Postcondizioni:** L'utente visualizza la data e ora della fascia oraria.

UC10.1.9.1.2 - Visualizzazione data apertura iscrizioni

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data di apertura delle iscrizioni della fascia oraria.
- **Postcondizioni:** L'utente visualizza la data di apertura delle iscrizioni della fascia oraria.

UC10.1.9.1.3 - Visualizzazione data chiusura iscrizioni

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data di chiusura delle iscrizioni della fascia oraria.
- **Postcondizioni:** L'utente visualizza la data di chiusura delle iscrizioni della fascia oraria.

UC10.1.9.1.4 - Visualizzazione posti max

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del numero massimo di posti della fascia oraria.
- **Postcondizioni:** L'utente visualizza il numero massimo di posti della fascia oraria.

UC10.1.9.1.5 - Visualizzazione posti max per utente

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola fascia oraria.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del numero massimo di posti per singolo utente della fascia oraria.
- **Postcondizioni:** L'utente visualizza il numero massimo di posti per singolo utente della fascia oraria.

UC11 - Aggiunta evento

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole aggiungere un nuovo evento.
- **Scenario principale:**
  1. L'utente inserisce i dati dell'evento;
  2. l'utente salva l'evento.
- **Postcondizioni:** L'utente ha aggiunto un nuovo evento.
- **Scenario alternativo:**
  1. Viene segnalato un errore se ci sono dati mancanti o errati (UC18).

UC12 - Modifica evento

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole modificare un evento esistente.
- **Scenario principale:**
  1. L'utente modifica i dati dell'evento;
  2. l'utente salva l'evento.
- **Postcondizioni:** L'utente ha modificato un evento esistente.
- **Scenario alternativo:**
  1. Viene segnalato un errore se ci sono dati mancanti o errati (UC18).

UC13 - Eliminazione evento

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole eliminare un evento esistente.
- **Scenario principale:**
  1. L'utente seleziona l'evento desiderato;
  2. l'utente elimina l'evento.

- **Postcondizioni:** L'utente ha eliminato un evento esistente.

UC14 - Visualizzazione lista prenotazioni

In figura 3.20 è rappresentato il caso d'uso tramite diagramma UML.

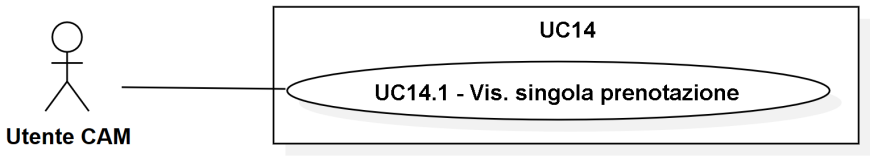


Figura 3.20: UC14 - Visualizzazione lista prenotazioni

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente ha effettuato il login e vuole visualizzare la lista delle prenotazioni di un singolo evento.
- **Scenario principale:**
  1. L'utente clicca sul link "Gestisci prenotazioni" del singolo evento.
- **Postcondizioni:** L'utente visualizza la lista delle prenotazioni di un singolo evento.

UC14.1 - Visualizzazione singola prenotazione

In figura 3.21 è rappresentato il caso d'uso tramite diagramma UML.

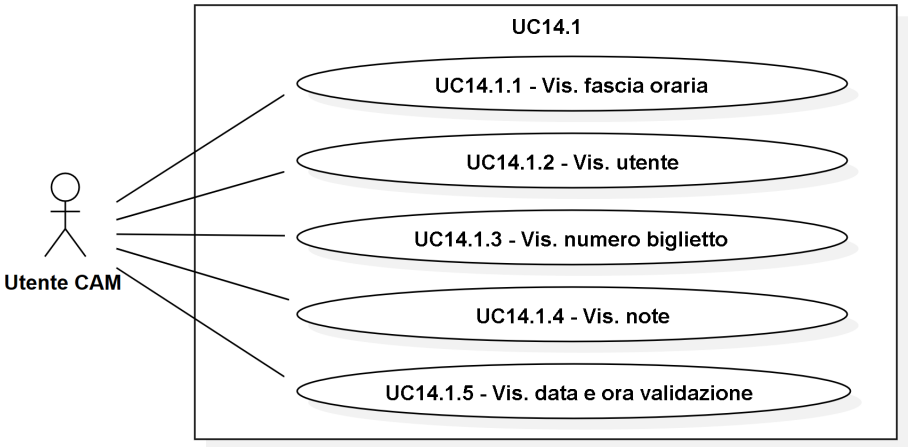


Figura 3.21: UC14.1 - Visualizzazione singola prenotazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la lista delle prenotazioni.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della singola prenotazione.
- **Postcondizioni:** L'utente visualizza la singola prenotazione.

UC14.1.1 - Visualizzazione fascia oraria

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola prenotazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data e ora della fascia oraria della prenotazione.
- **Postcondizioni:** L'utente visualizza la data e ora della fascia oraria della prenotazione.

UC14.1.2 - Visualizzazione utente

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola prenotazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione dell'utente della prenotazione.
- **Postcondizioni:** L'utente visualizza l'utente della prenotazione.

UC14.1.3 - Visualizzazione numero biglietto

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola prenotazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione del numero del biglietto.
- **Postcondizioni:** L'utente visualizza il numero del biglietto.

UC14.1.4 - Visualizzazione note

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola prenotazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione delle note della prenotazione.
- **Postcondizioni:** L'utente visualizza le note della prenotazione.

UC14.1.5 - Visualizzazione data e ora validazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente visualizza correttamente la singola prenotazione.
- **Scenario principale:**
  1. L'utente richiede la visualizzazione della data e ora di validazione della prenotazione.
- **Postcondizioni:** L'utente visualizza la data e ora di validazione della prenotazione.

UC15 - Aggiunta prenotazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole aggiungere manualmente una o più prenotazioni per un singolo evento.
- **Scenario principale:**
  1. L'utente seleziona la fascia oraria;
  2. l'utente seleziona l'utente della prenotazione;
  3. l'utente seleziona il numero di biglietti;
  4. l'utente salva le nuove prenotazioni.
- **Postcondizioni:** L'utente ha aggiunto una o più prenotazioni.
- **Scenario alternativo:**
  1. Viene segnalato un errore se ci sono dati mancanti o errati (UC18).

UC16 - Modifica note prenotazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole modificare le note di una prnotazione esistente.
- **Scenario principale:**
  1. L'utente modifica le note della prenotazione;
  2. l'utente salva la prenotazione.
- **Postcondizioni:** L'utente ha modificato le note di una prenotazione esistente.

UC17 - Eliminazione prenotazione

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente vuole eliminare una prenotazione esistente.
- **Scenario principale:**
  1. L'utente seleziona la prenotazione desiderata;
  2. l'utente elimina la prenotazione.
- **Postcondizioni:** L'utente ha eliminato una prenotazione esistente.

UC18 - Errore dati mancanti o errati

- **Attori Primari:** Utente CAM.
- **Precondizioni:** L'utente ha inserito o modificato dei campi, durante una aggiunta o modifica.
- **Scenario principale:**
  1. L'utente salva le modifiche effettuate.
- **Postcondizioni:** L'utente viene avvisato del fatto che ha inserito dati errati o che alcuni sono mancanti.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli *use case* effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli *use case*.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato:

R<Tipo><Classificazione><Numero>

In particolare, il tipo può assumere i seguenti valori:

- **F:** funzionale
- **Q:** qualitativo
- **V:** di vincolo

Per quanto riguarda la classificazione, essa può assumere i seguenti valori:

- **O:** obbligatorio
- **D:** desiderabile
- **F:** facoltativo

Nelle tabelle 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con gli *use case* delineati in fase di analisi. Nella tabella 3.4 è presente un riepilogo generale dei requisiti.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	**Descrizione**	***Use case***
RFO1	L'utente deve poter vedere gli eventi disponibili nell'applicazione	UC1
RFO2	L'utente deve poter scegliere un evento e visualizzarne i dettagli	UC1.1
RFO3	L'utente deve poter prenotare uno o più biglietti per un evento	UC3
RFO4	L'utente deve poter visualizzare i biglietti prenotati per un evento	UC4
RFO5	L'utente deve poter scegliere un biglietto e visualizzarne i dettagli	UC4.1
RFO6	L'utente deve poter validare un biglietto	UC5
RFO7	L'utente deve poter forzare la validazione di un biglietto	UC5.1
RFO8	L'utente deve ricevere un messaggio significativo che indichi l'eventuale validazione o errore	UC5.3

RFO9	L'utente deve poter scansionare un *qr-code* per validare un biglietto	UC6
RFO10	L'utente deve essere avvisato se non ci sono fasce orarie disponibili per un evento	UC7
RFO11	L'utente deve essere avvisato se mancano dei dati durante la prenotazione di uno o più biglietti	UC8
RFO12	L'utente deve essere avvisato se sta cercando di prenotare troppi biglietti	UC9
RFO13	L'utente deve poter visualizzare la lista di tutti gli eventi nel *backoffice*	UC10
RFO14	L'utente deve poter scegliere un evento e visualizzarne i dettagli	UC10.1
RFO15	L'utente deve poter visualizzare e modificare i target di un evento	UC10.1.7, UC12
RFO16	L'utente deve poter visualizzare e modificare le autorizzazioni di un evento	UC10.1.8, UC12
RFO17	L'utente deve poter visualizzare e modificare le fasce orarie di un evento	UC10.1.9, UC12
RFO18	L'utente deve poter creare un nuovo evento dal *backoffice*	UC11
RFO19	L'utente deve poter modificare un evento esistente dal *backoffice*	UC12
RFO20	L'utente deve poter eliminare un evento esistente	UC13
RFO21	L'utente deve poter visualizzare le prenotazioni effettuate per un particolare evento	UC14
RFO22	L'utente deve poter visualizzare una singola prenotazione	UC14.1
RFO23	L'utente deve poter aggiungere manualmente una o più prenotazioni per un particolare evento	UC15
RFO24	L'utente deve poter scegliere l'utente per cui effettuare la prenotazione	UC15
RFO25	L'utente deve poter modificare le note di una prenotazione dal *backoffice*	UC16

RFO26	L'utente deve poter eliminare una prenotazione dal *backoffice*	UC17
RFO27	L'utente deve essere avvisato se ci sono dati mancanti o errati durante un operazione di inserimento o modifica	UC18

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	**Descrizione**	***Use case***
RQO1	Deve essere redatto un documento che descrive l'architettura dell'applicazione	-
RQO2	Il codice deve essere documentato tramite commenti	-
RQO3	Deve essere redatto manuale utente per l'utilizzo dell'applicazione	-
RQD4	Devono essere eseguiti test in forma documentale	-
RQD5	Devono essere eseguiti test in forma di codice sorgente	-

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	**Descrizione**	***Use case***
RVO1	Il *software* prodotto deve essere scritto con il linguaggio C#	-
RVO2	L'applicazione deve essere sviluppata con il *framework Xamarin*	-
RVO3	La *web app* deve essere sviluppata con i componenti DevExpress[g] e il *framework* ASP .NET Core[g]	-
RVO4	I *web services* devono essere sviluppati con il *framework* ASP .NET Core	-
RVO5	L'applicazione deve funzionare su dispositivi *Android*	-
RVO6	L'applicazione deve funzionare su dispositivi *iOS*	-

Tabella 3.4: Tabella con il riepilogo dei requisiti

Tipo	Obbligatori	Desiderabili	Facoltativi
Funzionali	27	0	0
Qualitativi	3	2	0
Vincolo	6	0	0

3.3 Tecnologie e strumenti

Di seguito vengono elencate le tecnologie utilizzate durante il progetto di *stage*. Ognuna di esse è stata stabilita da *Ergon Informatica*, poiché rappresentano le tecnologie e gli strumenti da loro impiegati per lo sviluppo *software*.

C#

C#, il cui logo è rappresentato in figura 3.22, è un linguaggio di programmazione multi-paradigma orientato agli oggetti sviluppato da *Microsoft*. È caratterizzato da una sintassi chiara e leggibile, ed è noto per la sua capacità di combinare l'efficienza di esecuzione con un alto livello di sicurezza. C# è strettamente legato alla piattaforma *.NET*, che offre un ambiente di esecuzione e una vasta libreria di classi e servizi predefiniti per semplificare lo sviluppo. Grazie alla sua flessibilità, viene utilizzato per una vasta gamma di applicazioni, tra cui applicazioni *desktop*, *web* e *mobile*, nonché per lo sviluppo di giochi e soluzioni aziendali complesse.

Figura 3.22: Logo *C#*

Visual Studio 2022

Visual Studio 2022, il cui logo è rappresentato in figura 3.23, è un'IDE[g] fornito da *Microsoft*, sviluppato per semplificare la creazione, compilazione e *debugging* di *software* su piattaforme diverse. Infatti, esso consente agli sviluppatori di sviluppare applicazioni *desktop*, *web* e *mobile*. In generale, *Visual Studio 2022* supporta oltre 30 linguaggi di programmazione, tra cui C#, *C++*, *Python* e *Javascript*. Per lo sviluppo *web*, *Visual Studio 2022* offre strumenti come ASP .NET Core, mentre per lo sviluppo *mobile* offre strumenti come *Xamarin* e *.NET MAUI*.

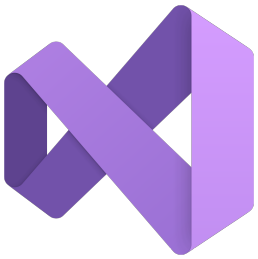


Figura 3.23: Logo *Visual Studio 2022*

DevExpress

DevExpress, il cui logo è rappresentato in figura 3.24, è una piattaforma di componenti e strumenti per lo sviluppo *software*, progettata per semplificare lo sviluppo di interfacce grafiche per applicazioni *desktop*, *web* e *mobile*. La caratteristica principale di DevExpress è la sua flessibilità e adattabilità alle diverse esigenze. Infatti, gli sviluppatori possono personalizzare l'aspetto e il comportamento di ogni componente.

Figura 3.24: Logo *DevExpress*

Xamarin

Xamarin, il cui logo è rappresentato in figura 3.25, è una piattaforma di sviluppo che consente agli sviluppatori di creare applicazioni *mobile* per *Android* e *iOS* condividendo un unico codice base. Una descrizione più approfondita di *Xamarin* è riportata nella sezione 2.2.

Figura 3.25: Logo *Xamarin*

ASP .NET Core

ASP .NET Core, il cui logo è rappresentato in figura 3.26, è un *framework* di sviluppo web open source fornito da *Microsoft*, sviluppato per creare applicazioni *web* moderne, performanti e scalabili. Esso supporta diversi linguaggi di programmazione, con C# come opzione principale. Inoltre, fornisce strumenti per la gestione di richieste *HTTP*, l'elaborazione delle risposte, la sicurezza e molto altro.

Figura 3.26: Logo *ASP .NET Core*

Capitolo 4

Progettazione

4.1 Architettura generale

Di seguito è riportato uno schema ad alto livello delle componenti che formano il sistema e della loro interazione.

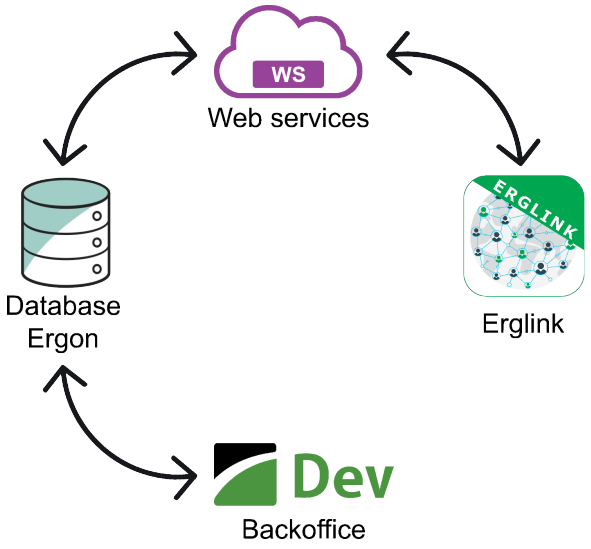


Figura 4.1: Architettura generale del sistema

Come si può notare in figura 4.1, il sistema è formato principalmente da quattro componenti:

- *Database* Ergon;
- *Backoffice*, che interagisce direttamente con il *database* con operazioni CRUD[g];
- *Web services*, che gestiscono il flusso di dati tra il *database* e l'*app*;
- Erglink, che interagisce con i *web services* per effettuare operazioni CRUD sul *database*.

4.2 Web services

Per la progettazione dei *web service* è stato deciso utilizzare il *framework* ASP .NET Core, che permette di sviluppare delle API REST[g] utilizzando Entity Framework[g].

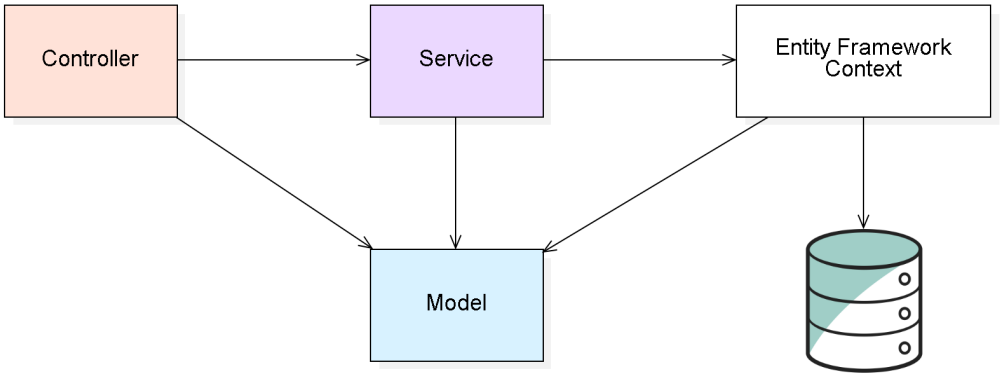


Figura 4.2: Schema generale API - Entity Framework

Come si può notare in figura 4.2, sono presenti quattro componenti: i ***Controller***, i ***Service***, i ***Model*** e il ***Context***. I *model* rappresentano i dati dell'applicazione e rispecchiano le entità presenti nel *database*. Ai controller spetta il compito di gestire le richieste provenienti dall'esterno, tipicamente attraverso richieste *HTTP* che vengono mappate grazie ad un sistema di *routing* ai metodi dei *controller*, in base all'*url* e al metodo *HTTP*. Questi metodi poi andranno ad individuare ed invocare il *service* appropriato per elaborare la richiesta. Infatti, i service sono quelli che implementano la logica di *business*, interagendo con il *database*, elaborando i dati e fornendo una risposta al *controller*. Per interagire con il *database* viene utilizzato un *context*, che ha il compito di astrarre i dettagli di accesso ad esso e di fornire i metodi per permettere l'esecuzione di operazioni CRUD sulle entità.
Per quanto riguarda i *web service* da sviluppare per il modulo per la gestione degli eventi, sono stati progettati i diagrammi delle classi nelle figure 4.3, 4.4, 4.5.

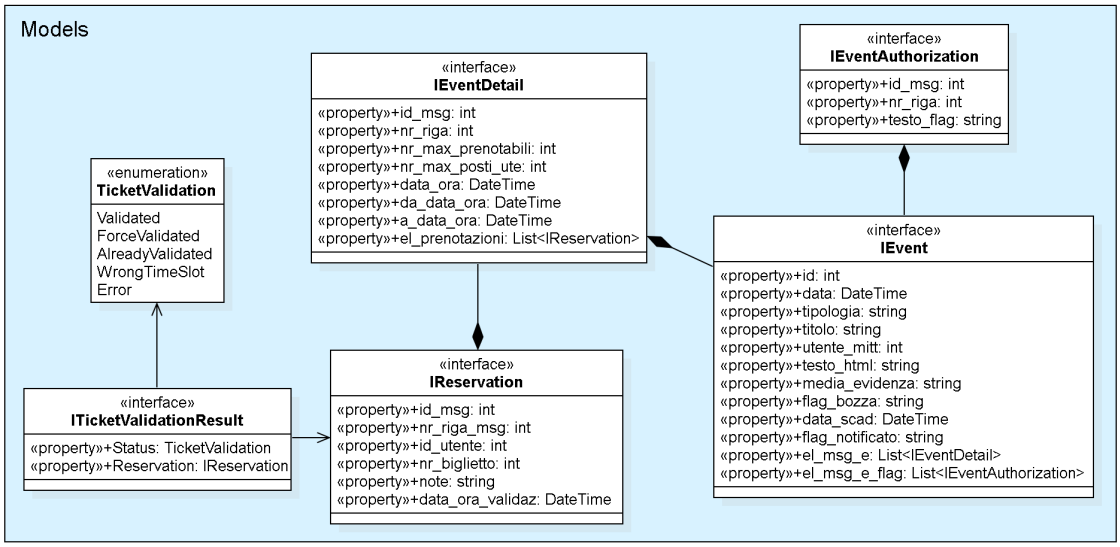


Figura 4.3: *Web services - Models*

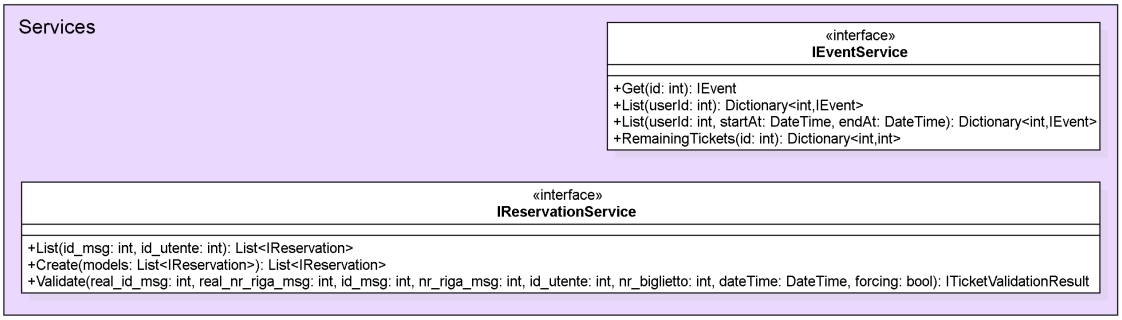


Figura 4.4: *Web services - Services*

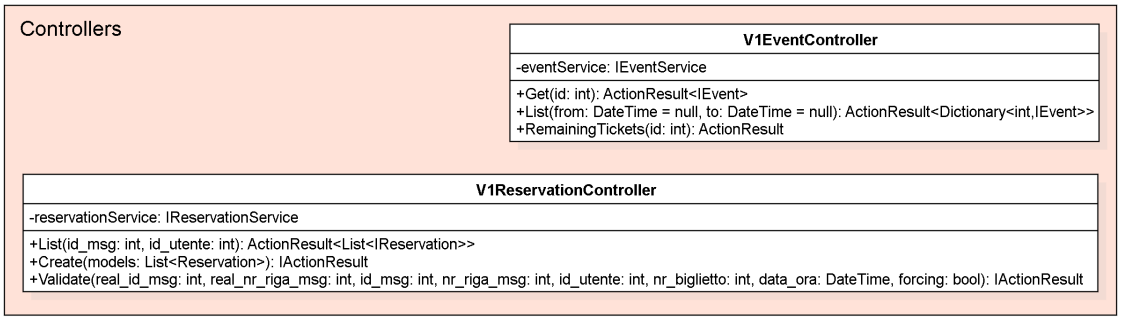


Figura 4.5: *Web services - Controllers*

Come si può notare in figura 4.5, sono presenti due *controller* che espongono le API REST per le due principali aree, ovvero gli eventi e le prenotazioni:

- ***V1EventController*:** fornisce dei metodi per avere informazioni sugli eventi esistenti, le loro fasce orarie ed il numero di posti rimasti;
- ***V1ReservationController*:** fornisce dei metodi per interagire con gli eventi disponibili, quindi effettuare delle prenotazioni o visualizzare quelle già presenti.

Ognuno dei *controller* invoca il corrispettivo *service* presente in figura 4.4, che andrà a fornire i risultati dopo aver interagito con il *database*.
Questi risultati sono i modelli in figura 4.3, che rappresentano le entità necessarie per la gestione degli eventi:

- ***IEvent*:** che rappresenta un singolo l'evento;
- ***IEventDetail*:** che rappresenta una singola fascia oraria relativa ad un evento;
- ***IEventAuthorization*:** che rappresenta una singola autorizzazione da accettare relativa ad un evento;
- ***IReservation*:** che rappresenta una singola prenotazione o biglietto, relativa ad una fascia oraria di un evento;
- ***ITicketValidationResult*:** che rappresenta il risultato di un tentativo di validazione di un biglietto;
- ***TicketValidation*:** che rappresenta lo stato che può assumere la validazione di un biglietto.

4.2.1 Endpoint

Per dare uniformità agli *endpoint* è stato deciso di seguire uno *standard* per le risposte fornite dai *controller*. Infatti, ogni risposta è contrassegnata da un codice di stato *HTTP* e da una stringa in formato JSON[g], ovvero il vero e proprio contenuto della risposta. In particolare, per le richieste di tipo **GET**, i possibili codici di stato *HTTP* per le risposte sono:

- **200:** richiesta andata a buon fine;
- **400:** errore nella richiesta;
- **401:** utente non autenticato;
- **403:** l'utente non può effettuare questa richiesta;
- **404:** il dato richiesto non esiste;
- **500:** errore lato server;

Per quanto riguarda la richieste di tipo **POST**, ognuna di esse ha una lista di possibili codici di stato *HTTP* per la risposta, ad esempio per segnalare un particolare errore. Di seguito sono elencati gli endpoint progettati che sono stati mappati ai metodi dei *controller*, inclusi i codici di risposta *HTTP* per le richieste di tipo **POST**:

V1EventController:

- **GET** /api/v{version}/el_msgs_e/{id}
  Parametri *path*:
  - int id

  Descrizione: Ritorna i dati e i dettagli di uno specifico evento, tranne le prenotazioni.
- **GET** /api/v{version}/el_msgs_e
  Parametri *query*:
  - DateTime from
  - DateTime to

  Descrizione: Ritorna i dati di tutti gli eventi visibili all'utente compresi nel *range* di date.
- **GET** /api/v{version}/el_msgs_e/remaining_tickets/{id}
  Parametri *path*:
  - int id

  Descrizione: Ritorna il numero di posti rimanenti per ogni fascia oraria di un evento.

V1ReservationController:

- **GET** /api/v{version}/el_prenotazioni/prenotazioni_utente_evento
  Parametri *query*:
  - int id_msg
  - int id_utente

  Descrizione: Ritorna i dati di tutte le prenotazioni di un evento di uno specifico utente.

- **POST** /api/v{version}/el_prenotazioni/list
  Parametri *request body*:
  - List<Reservation> models

  Codici risposta:
  - **200:** prenotazioni avvenute correttamente;
  - **400:** errore nella richiesta;
  - **401:** utente non autenticato;
  - **409:** conflitto nel *database*;
  - **410:** biglietti terminati;
  - **413:** l'utente vuole prenotare più biglietti del massimo per utente;
  - **416:** non ci sono abbastanza biglietti per la prenotazione;
  - **500:** errore nel *server*.

  Descrizione: Inserisce nel database una lista di nuove prenotazioni.

- **POST** /api/v{version}/el_prenotazioni/validate
  Parametri *query*:
  - int real_id_msg
  - int real_nr_riga_msg
  - int id_msg
  - int nr_riga_msg
  - int id_utente
  - int nr_biglietto
  - DateTime data_ora
  - bool forcing

  Codici risposta:
  - **200:** biglietto validato correttamente;
  - **205:** biglietto validato forzatamente, ovvero validato anche se la fascia oraria indicata non corrisponde, o validato per una seconda volta;
  - **307:** biglietto non validato: corrisponde ad un'altra fascia oraria, ma può essere forzato;
  - **400:** errore nella richiesta;
  - **401:** utente non autenticato;
  - **406:** biglietto non validato: corrisponde ad un altro evento, non può essere forzato;
  - **409:** conflitto nel *database*;
  - **412:** biglietto già validato: può essere rivalidato forzatamente;
  - **500:** errore nel *server*.

  Descrizione: Valida un biglietto.

4.3 App Erglink

Per la progettazione del modulo da inserire nell'applicazione Erglink è stato deciso di adottare il *pattern MVVM(Model-View-ViewModel)* già presente in *Xamarin*.

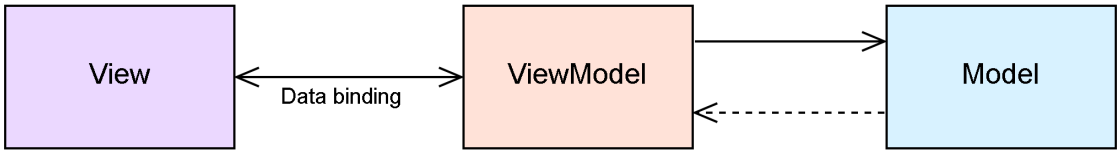


Figura 4.6: *MVVM Pattern*

Come si può notare in figura 4.6, il *pattern MVVM* ha tre componenti principali: i ***Model***, le ***View*** e i ***ViewModel***. Anche in questo caso, il *model* rappresenta l'insieme di dati e logiche dell'applicazione, ovvero le informazioni fondamentali che l'applicazione gestisce e manipola. D'altra parte, la *view* è responsabile dell'aspetto visivo dell'interfaccia utente, il *layout* e la struttura degli elementi visualizzati sullo schermo. Infine, la *view* interagisce con il *model* solo attraverso il *viewModel*, in modo da consentire una chiara separazione delle responsabilità. Questa interazione è resa possibile tramite il *data binding*, che consente l'aggiornamento dinamico dell'interfaccia in base ai cambiamenti dei dati nel *model*. Per fare ciò, il *viewModel* espone proprietà e comandi pubblici, che vengono utilizzati dalla *view* per mostrare i dati e gestire le interazioni con l'utente. Inoltre, il *viewModel* ha la flessibilità di adattare i dati del *model* in un formato più adatto per essere visualizzati nella *view*, ed esegue operazioni come la formattazione o la conversione dei tipi di dati. L'adozione di questo *pattern* consente quindi di separare la logica di presentazione da quella di *business*, portando diversi vantaggi:

- il codice è più manutenibile e facile da comprendere, perché ogni componente ha una responsabilità distinta e chiara;
- permette di creare test di unità per il *model* e *viewModel* senza utilizzare la *view*;
- favorisce il riutilizzo del codice, poiché lo stesso *model* o *viewModel* può essere utilizzato da più *view*;
- consente agli sviluppatori e ai progettisti dell'interfaccia utente di collaborare più facilmente, perché possono lavorare in modo indipendente sui rispettivi componenti.

In particolare, per quanto riguarda il modulo per la gestione degli eventi per l'applicazione *mobile* Erglink, sono stati progettati i diagrammi delle classi in figura 4.7.

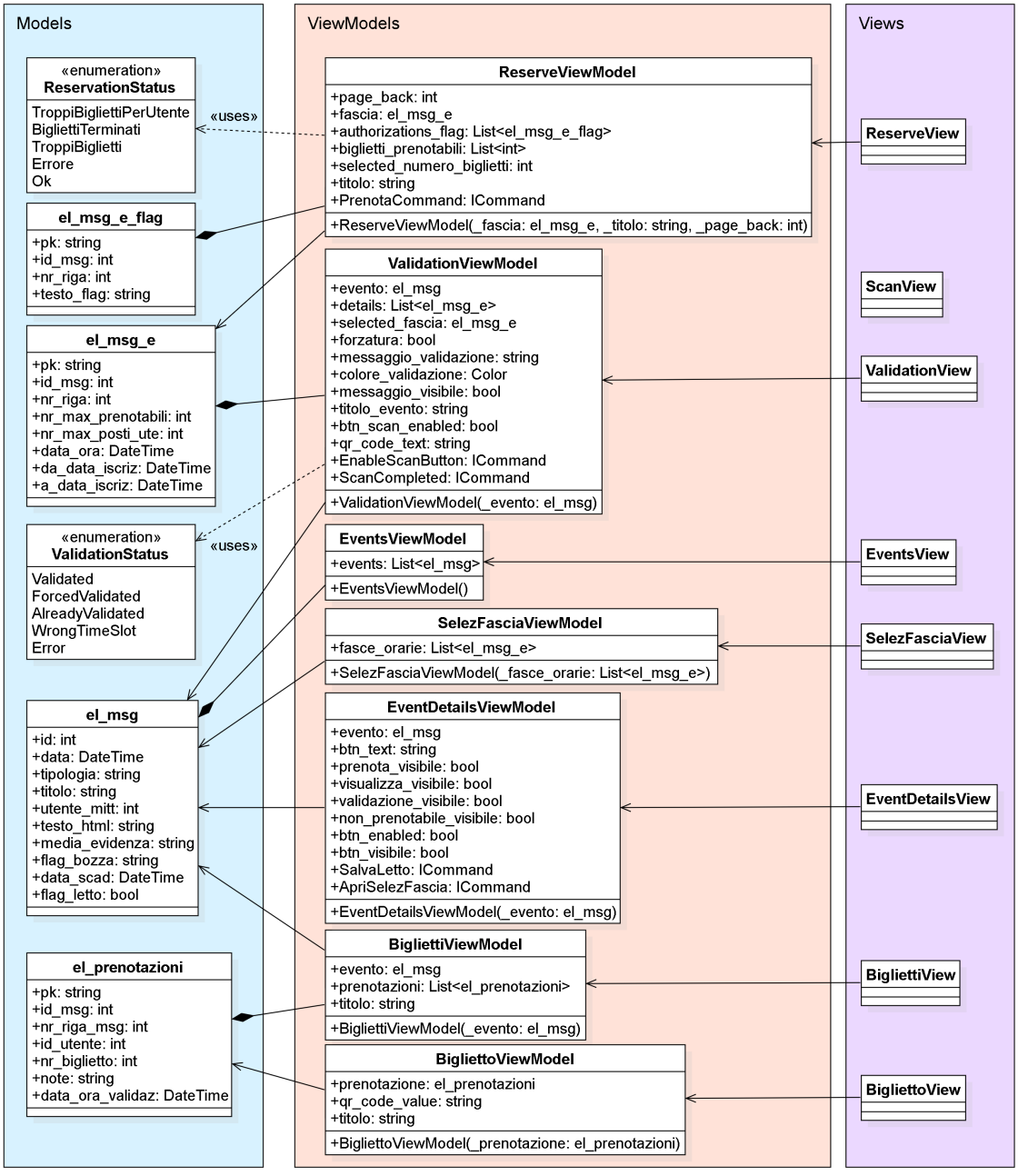


Figura 4.7: *Diagramma classi app*

Di seguito sono elencati i *model* e il loro significato:

- ***el_msg*:** rappresenta l'evento;
- ***el_msg_e*:** rappresenta una fascia oraria di un evento;
- ***el_msg_e_flag*:** rappresenta un'autorizzazione da accettare per partecipare all'evento;
- ***el_prenotazioni*:** rappresenta una prenotazione o biglietto;
- ***ReservationStatus*:** che rappresenta lo stato che può assumere la prenotazione di uno o più biglietti;

- ***ValidationStatus*:** che rappresenta lo stato che può assumere la validazione di un biglietto.

Di seguito sono elencate le *view* e il loro compito:

- ***EventsView*:** visualizza la lista di eventi disponibili;;
- ***EventDetailsView*:** visualizza i dettagli di un singolo evento;
- ***SelezFasciaOraria*:** visualizza la lista di fasce orarie disponibili per un singolo evento, per poi far procedere ad una prenotazione;
- ***ReserveView*:** richiede l'inserimento dei dati per effettuare una prenotazione;
- ***BigliettiView*:** visualizza la lista di biglietti prenotati per un singolo evento;
- ***BigliettoView*:** visualizza i dati di una singola prenotazione;
- ***ValidationView*:** richiede l'inserimento dei dati per validare un biglietto;
- ***ScanView*:** effettua la scansione del *qr-code* di un biglietto e lo passa alla *ValidationView*. Dato che non è necessario il *data binding*, essa non possiede un *viewModel*.

4.4 Backoffice

Per la progettazione del *backoffice* è stato utilizzato il *framework* ASP .NET Core, che permette di sviluppare *web app* utilizzando il *pattern MVC(Model-View-Controller)* con pagine Razor[g].

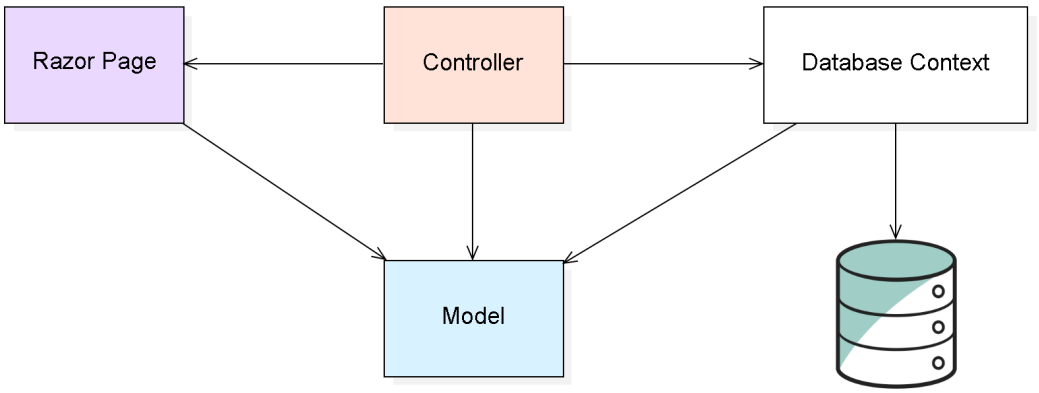


Figura 4.8: *MVC Pattern* e *Razor Pages - ASP.NET Core*

Come si può notare in figura 4.8, il *pattern MVC* con pagine Razor ha le seguenti componenti: le ***Razor Page***, i ***Controller***, i ***Model*** e il ***Context***. Come nei *pattern* visti precedentemente, il *model* costituisce l'insieme dei dati e logiche dell'applicazione, contenendo le informazioni da manipolare e gestire. Le pagine *razor* sono l'equivalente della *view* nella versione classica del *pattern MVC*, quindi rappresentano la struttura, il *layout* e l'aspetto di ciò che l'utente vede sullo schermo. Queste pagine incorporano codice C# nel classico *HTML* e possono usare *layout*, viste parziali, modelli e *helper tag* per condividere il codice e il *markup* tra le pagine. Il *controller* invece, ha il ruolo di gestire le richieste provenienti dal *browser*, recuperando i dati e richiamando le viste per restituire una risposta. Per fornire i dati alle viste, il *controller* interagisce con il *context*, che come visto in precedenza astrae i dettagli di accesso al *database* e fornisce i metodi per eseguire operazioni CRUD sulle entità. Per quanto riguarda il *backoffice* da sviluppare per il modulo per la gestione degli eventi, sono stati progettati i diagrammi delle classi nelle figure 4.9, 4.10, 4.11.

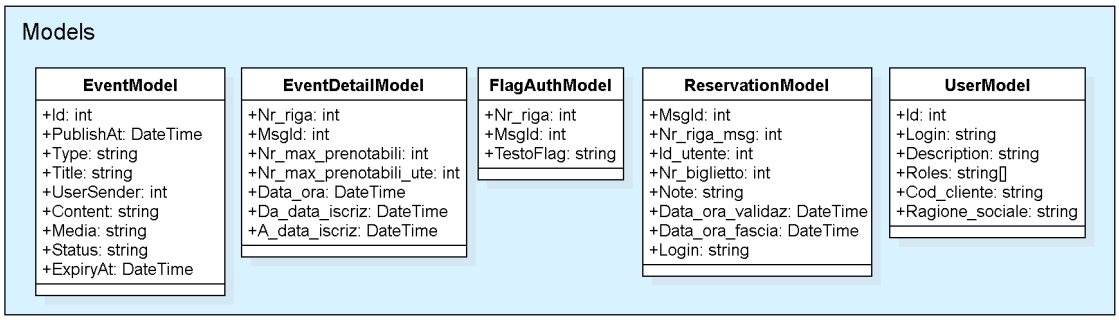


Figura 4.9: *Backoffice - Models*

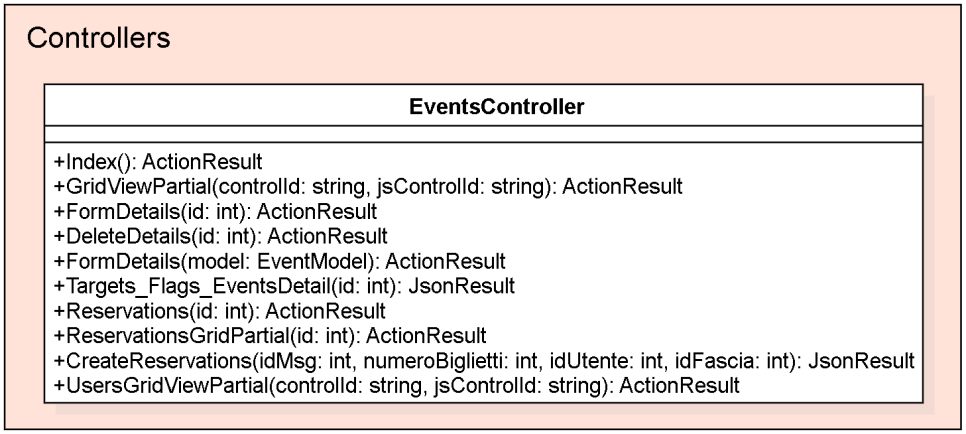


Figura 4.10: *Backoffice - Controllers*

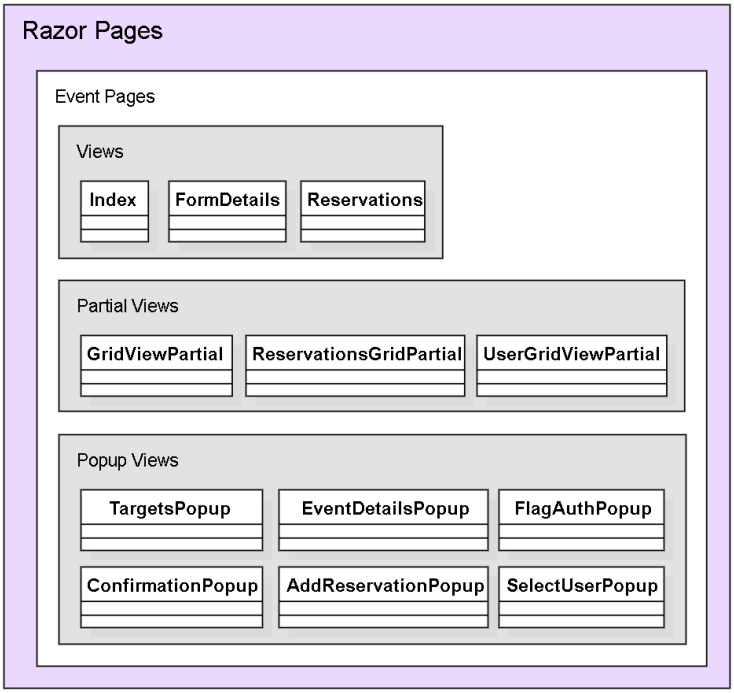


Figura 4.11: *Backoffice - Razor Pages*

Di seguito sono elencati i *model* e il loro significato:

- ***EventModel*:** rappresenta l'evento;
- ***EventDetailModel*:** rappresenta una fascia oraria di un evento;
- ***FlagAuthModel*:** rappresenta un'autorizzazione da accettare per partecipare all'evento;
- ***ReservationModel*:** rappresenta una prenotazione o biglietto;
- ***UserModel*:** rappresenta un utente.

Per quanto riguarda le pagine Razor, si può notare in figura 4.10 che possono essere divise in tre categorie:

- ***Views*:** sono le pagine principali e possono essere composte da una o più *partial views*. Ne fanno parte:
  - ***Index*:** è la pagina principale per gli eventi;
  - ***FormDetails*:** è la pagina principale per le aggiunte e modifiche di singoli eventi;
  - ***Reservations*:** è la pagina principale per le prenotazioni di un singolo evento.
- ***Partial Views*:** sono parti di *views* che possono essere riutilizzate. Ne fanno parte:
  - ***GridViewPartial*:** è la griglia che contiene gli eventi nella *view "Index"*;
  - ***ReservationsGridPartial*:** è la griglia che contiene le prenotazioni di un singolo evento nella *view "Reservations"*;
  - ***UserGridViewPartial*:** è la griglia che contiene gli utenti selezionabili nella *popup view "SelectUserPopup"*.
- ***Popup Views*:** sono le pagine che vengono usate come *popup* per dare avvisi o richiedere l'inserimento di alcuni dati. Ne fanno parte:
  - ***TargetsPopup*:** è il *popup* che chiede l'inserimento dei dati per la creazione o modifica dei *target* di un evento;
  - ***EventDetailsPopup*:** è il *popup* che chiede l'inserimento dei dati per la creazione o modifica delle fasce orarie di un evento;
  - ***FlagAuthPopup*:** è il *popup* che chiede l'inserimento dei dati per la creazione o modifica delle autorizzazioni per un evento;
  - ***ConfirmationPopup*:** è il *popup* che chiede una conferma per la pubblicazione di un evento con *target*, fasce orarie o autorizzazioni mancanti;
  - ***AddReservationPopup*:** è il *popup* che chiede l'inserimento dei dati per la creazione manuale di una o più prenotazioni;
  - ***SelectUserPopup*:** è il *popup* che chiede di selezionare un utente a cui assegnare le prenotazioni create tramite *"AddReservationPopup"*.

Per quanto riguarda l'unico *controller* presente *"EventsController"*, si può notare in figura 4.11 come i suoi metodi si dividano in due categorie: quelli che ritornano una vista e quelli che ritornano un dato, ad esempio un oggetto JSON. In generale, per i metodi che ritornano una vista, essa viene identificata tramite il nome del metodo.

Capitolo 5

Codifica

5.1 Organizzazione dello sviluppo

Lo sviluppo del progetto è stato diviso in tre parti principali, che sono state sviluppate nel seguente ordine:

1. sviluppo *web services*;
2. sviluppo *app* Erglink;
3. sviluppo *backoffice*.

È stato necessario sviluppare per primi i *web services* per permettere successivamente all'applicazione Erglink di interagire con il *database* tramite essi, come riportato in figura 4.1.

5.2 Web services

Per quanto riguarda lo sviluppo dei *web services* è statio adottato un approcio incrementale, in modo tale da avere sempre un prodotto funzionante. Come prima cosa sono stati creati i modelli e sono stati mappati alle entità del *database*. Successivamente, ogni *endpoint* è stato aggiunto individualmente secondo il seguente ordine:

1. codifica dei metodi del *service* per fornire la risposta desiderata;
2. codifica del metodo del *controller* mappato alla richiesta *HTTP* dell'*endpoint*.

5.3 App Erglink

Per quanto riguarda l'applicazione Erglink, è stato deciso di dividere la codifica in quattro parti il più indipendenti possibile tra di loro, ovvero **visualizzazione eventi**, **prenotazione biglietti**, **visualizzazione biglietti** e **validazione biglietti**.

5.3.1 Visualizzazione eventi

La visualizzazione degli eventi comprende lo sviluppo di due pagine: ***EventsView*** e ***EventDetailsView***, con i rispettivi *viewModel* e *model*. Come si può notare in figura 5.1, la prima pagina permette la visualizzazione e selezione degli eventi disponibili all'utente. Dopo la selezione di uno di essi, sarà possibile visualizzarne i dettagli con tre casi per le azioni successive: **prenotazione** dei biglietti, se l'utente è di tipo C e non ci sono biglietti prenotati (vedi figura 5.2); **visualizzazione** dei biglietti, se l'utente è di tipo C e ci sono biglietti prenotati precedentemente (vedi figura 5.3); **validazione** dei biglietti, se l'utente è di tipo LDI (vedi figura 5.4).

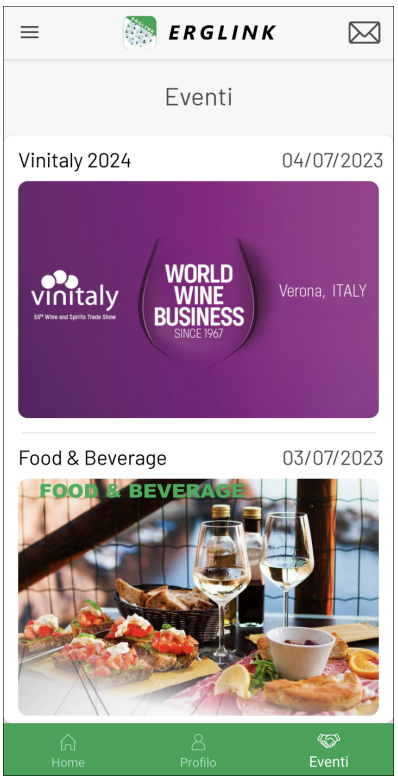


Figura 5.1: *EventsView*

Figura 5.2: *EventDetailsView* - Prenotazione

Figura 5.3: *EventDetailsView* - Visualizzazione

Figura 5.4: *EventDetailsView* - Validazione

5.3.2 Prenotazione biglietti

La prenotazione dei biglietti comprende lo sviluppo due pagine: *SelezFasciaView* e *ReserveView*, con i rispettivi *viewModel* e *model*. Dopo aver premuto il pulsante "Prenota biglietti" presente in figura 5.2, verrà chiesto all'utente di selezionare la fascia oraria per il quale vuole effettuare la prenotazione, come mostrato i figura 5.5. Successivamente verrà chiesto all'utente di inserire gli ulteriori dati necessari per completare la prenotazione (vedi figura 5.6).

Figura 5.5: *SelezFasciaView*

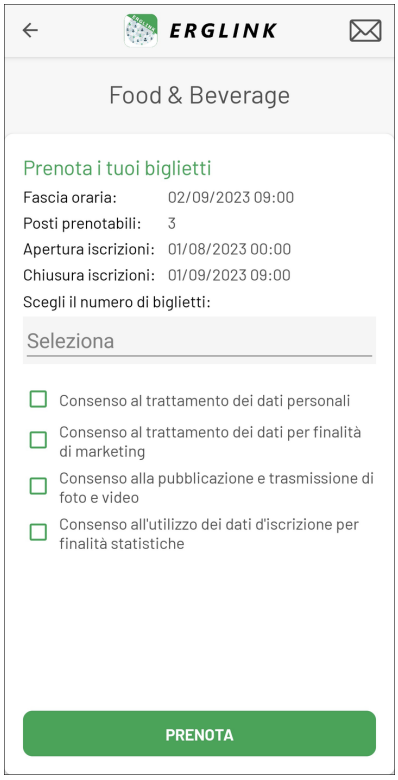


Figura 5.6: *ReserveView*

Dopo aver premuto il pulsante "Prenota", ci potranno essere due scenari: nel primo caso la prenotazione è andata a buon fine, verrà quindi mostrato a schermo un messaggio di conferma (vedi figura 5.7) e successivamente l'utente verrà reindirizzato alla pagina del dettaglio dell'evento, mostrando questa volta il pulsante per la visualizzazione dei biglietti (vedi figura 5.3). Nel secondo caso la prenotazione non è andata a buon fine, e ciò può essere dovuto ad uno dei seguenti motivi:

- dati mancanti nella prenotazione;
- i biglietti sono terminati;
- non ci sono abbastanza biglietti per la prenotazione richiesta.

Per ognuno di questi casi verrà mostrato a schermo un messaggio che informerà l'utente. Un esempio è visibile in figura 5.8.

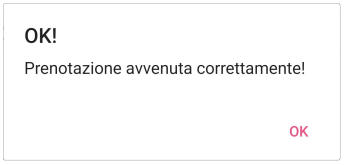


Figura 5.7: *ReserveView* - Prenotazione effettuata

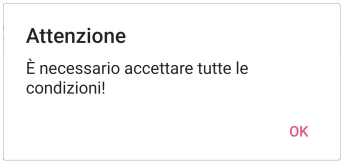


Figura 5.8: *ReserveView* - Errore

5.3.3 Visualizzazione biglietti

La visualizzazione dei biglietti comprende lo sviluppo di due pagine: *BigliettiView* e *BiglietoView*, con i rispettivi *viewModel* e *model*. Dopo aver premuto il pulsante "Visualizza biglietti" presente in figura 5.3, verrà visualizzata la lista dei biglietti per il particolare evento, come mostrato il figura 5.9. Successivamente, l'utente potrà selezionare un biglietto per visualizzarne i dettagli.

Figura 5.9: *BigliettiView*

Figura 5.10: *BiglietoView*

Come si può notare in figura 5.10, la visualizzazione di un singolo biglietto comprende la creazione di un *qr code*, che verrà utilizzato durante la validazione per riconoscere tale biglietto e validarlo.

5.3.4 Validazione biglietti

La validazione dei biglietti comprende lo sviluppo di due pagine: *ValidationView* e *ScanView*. Dopo aver premuto il pulsante "Validazione biglietti" presente in figura 5.4, verrà visualizzata la prima pagina che chiederà all'utente di scegliere la fascia oraria per la validazione dei biglietti e darà la possibilità di forzare la validazione di un biglietto nel caso fosse già stato validato in precedenza o in caso di appartenenza ad un' altra fascia orarie dello stesso evento (vedi figura 5.11). Successivamente, premendo il pulsante "Scansiona Qr Code" sarà possibile scansionare il *qr-code* del biglietto da validare (vedi figura 5.12). Come descritto nella sezione relativa alla progettazione degli *endpoint* (vedi 4.2.1), il tentativo di validazione di un biglietto può generare diversi risultati, ognuno dei quali informa l'utente con un messaggio e colore significativo. Nelle figure 5.13, 5.14 e 5.15 sono riportati alcuni esempi di risposta ad un tentativo di validazione.

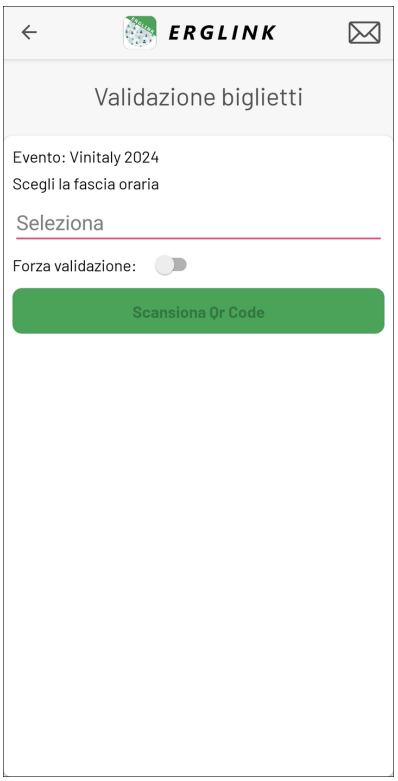


Figura 5.11: *ValidationView*

Figura 5.12: *ScanView*

Figura 5.13: Biglietto validato

Figura 5.14: Fascia oraria errata

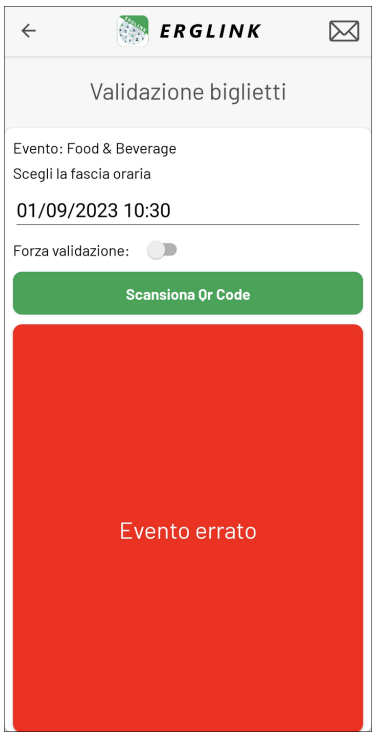


Figura 5.15: Evento errato

5.4 Backoffice

Per quanto riguarda il *backoffice*, come prima cosa sono stati codificati i modelli necessari e mappati alle entità del *database*. Successivamente è stato deciso di dividere la codifica in quattro parti principali, ovvero la **visualizzazione degli eventi**, la **gestione degli eventi**, la **visualizzazione delle prenotazioni** e la **gestione delle prenotazioni**.

5.4.1 Visualizzazione degli eventi

Per visualizzare la lista degli eventi è stato necessario sviluppare due pagine *razor*: *Index* e *GridViewPartial*, con i rispettivi metodi del *controller* per permetterne la visualizzazione. Come descritto nella sezione relativa alla progettazione del *backoffice* (vedi 4.4), "*Index*" è la pagina principale degli eventi ed incorpora la vista parziale "*GridViewPartial*", la quale contiene la griglia con i principali dati degli eventi. In figura 5.16 è presente la renderizzazione della pagina finale.

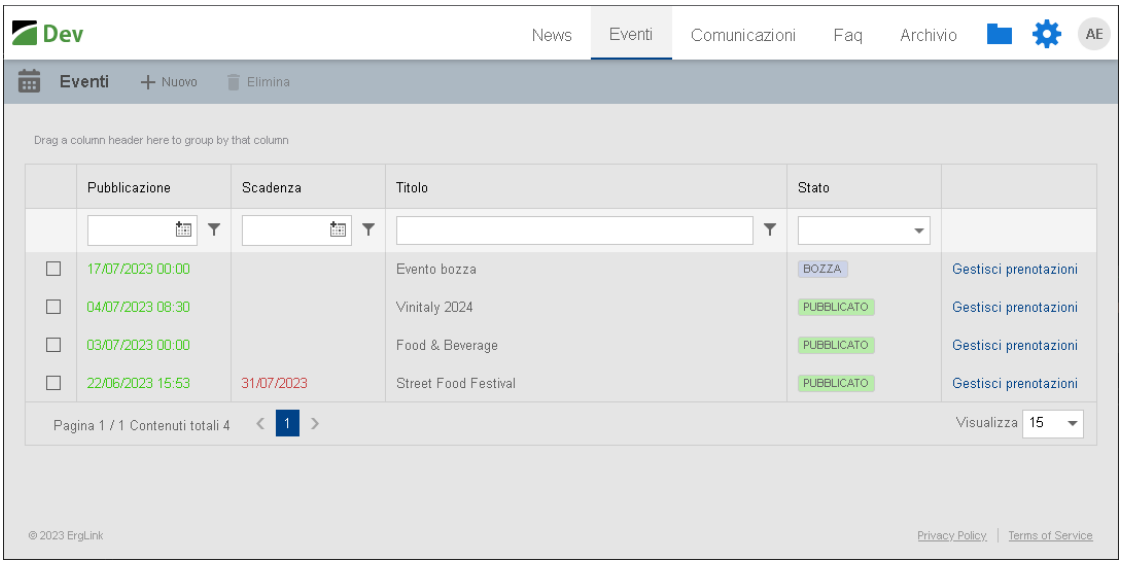


Figura 5.16: *Index* e *GridViewPartial*

5.4.2 Gestione degli eventi

La gestione degli eventi comprende lo sviluppo di cinque pagine *razor*: *FormDetails*, *TargetsPopup*, *EventDetailsPopup*, *FlagAuthPopup* e *ConfirmationPopup*. La prima pagina consente all'utente di aggiungere o modificare un evento. In figura 5.17 è riportato un esempio di modifica di un evento. Infatti, dopo aver selezionato un evento presente in figura 5.16, l'utente visualizzerà tutti i dati di quell'evento e avrà la possibilità di modificarli. Premendo sui pulsanti con le icone per i "Targets", "Autorizzazioni" e "Fasce orarie" verranno visualizzati i tre *popup* per l'inserimento o modifica dei dati visibili nelle figure 5.18, 5.19 e 5.20. Successivamente, se tutti i dati sono stati inseriti, premendo il pulsante "Salva" l'evento verrà salvato. Nal caso in cui non fossero presenti *target*, fasce orarie o autorizzazioni, verrà mostrato all'utente il *popup* *ConfirmationPopup*, che chiederà una conferma sul fatto di salvare l'evento come bozza o come pubblicato (vedi figura 5.21).

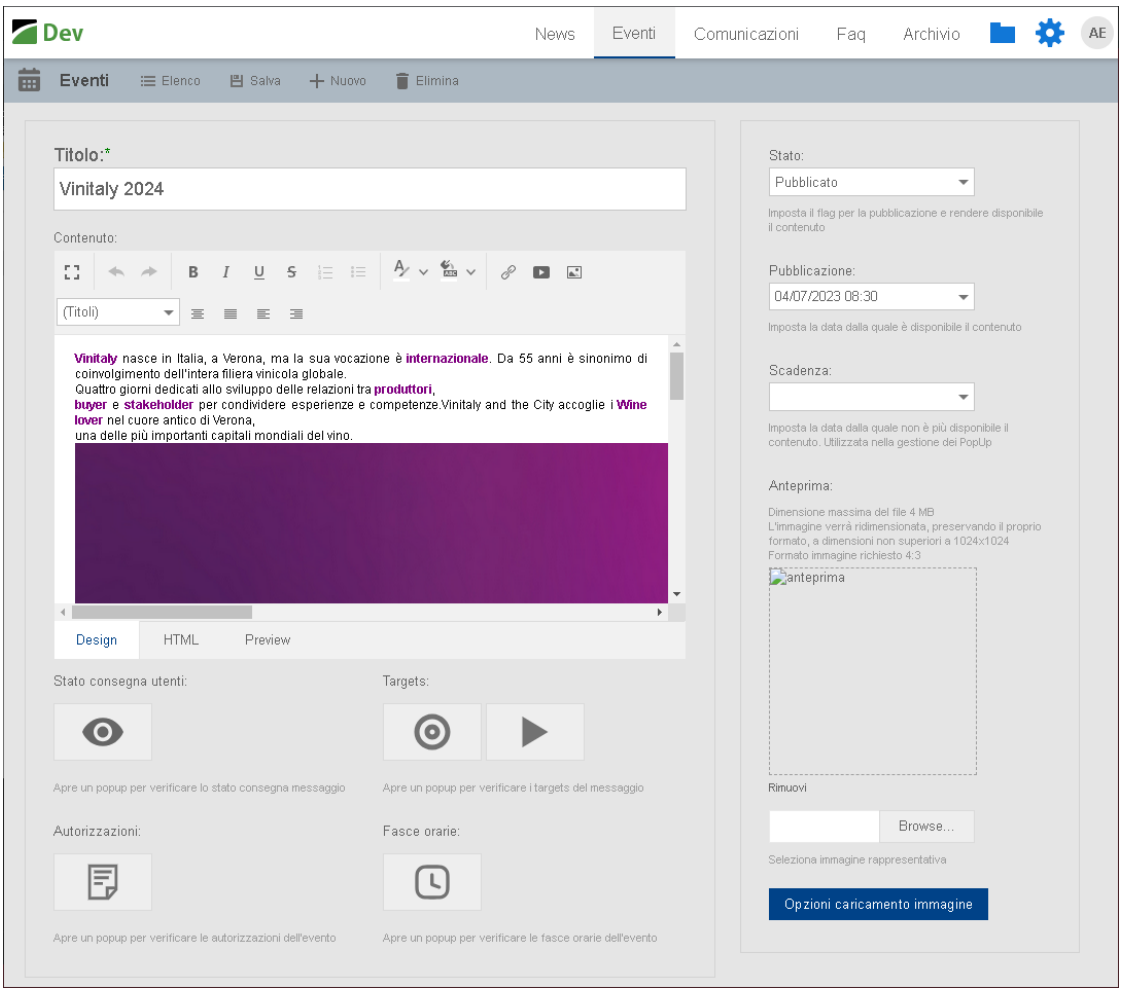


Figura 5.17: *FormDetails*

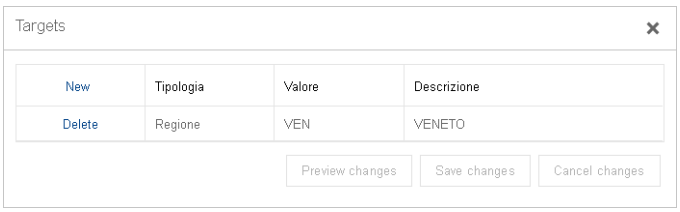


Figura 5.18: *TargetsPopup*

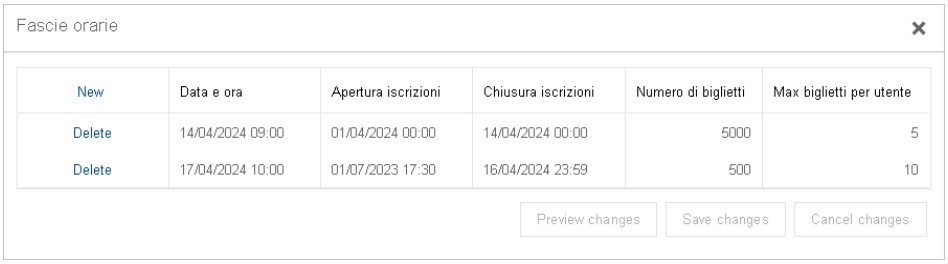


Figura 5.19: *EventDetailsPopup*

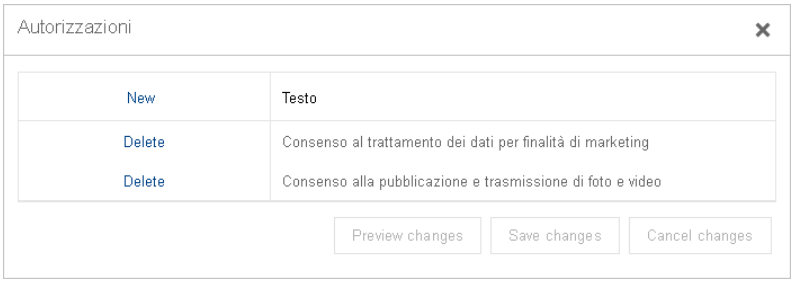


Figura 5.20: *FlagAuthPopup*

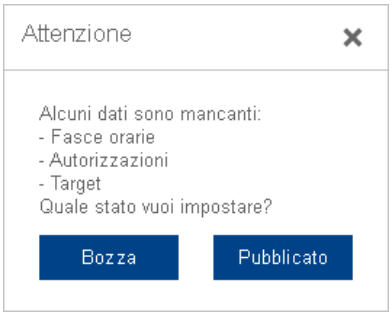


Figura 5.21: *ConfirmationPopup*

5.4.3 Visualizzazione delle prenotazioni

La visualizzazione delle prenotazioni comprende lo sviluppo di due pagine *Razor*: *Reservations* e *ReservationsGridPartial*, con i rispettivi metodi del *controller* per permetterne la visualizzazione. Come descritto nella sezione relativa alla progettazione del *backoffice* (vedi 4.4), "Reservations" è la pagina principale delle prenotazioni, ed incorpora la vista parziale "ReservationsGridPartial", la quale contiene la griglia con i dati delle prenotazioni. In figura 5.22 è presente la renderizzazione della pagina finale, raggiungibile dopo aver selezionato un evento dalla pagina degli eventi in figura 5.16.

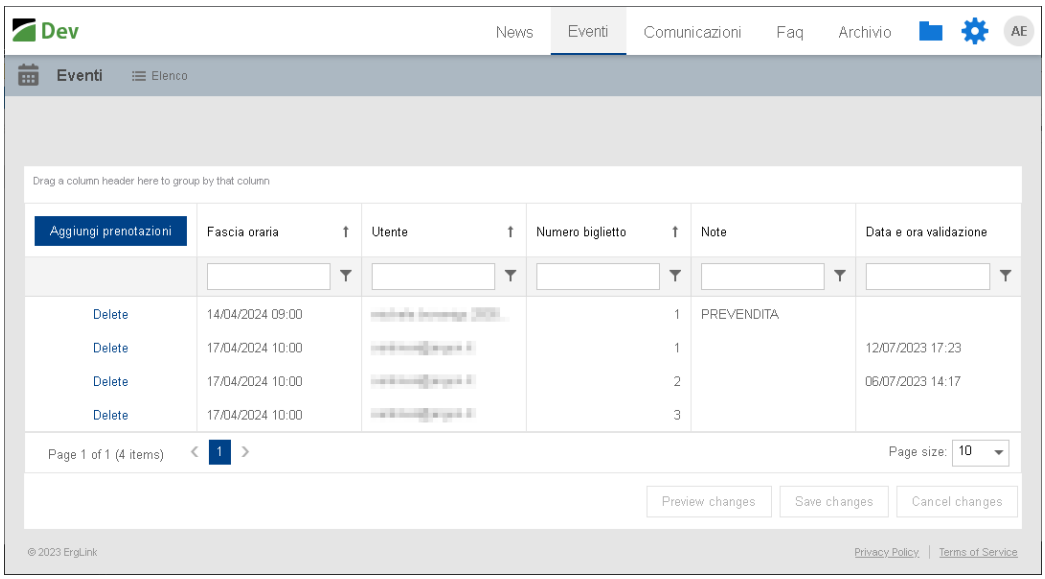


Figura 5.22: *Reservations* e *ReservationsGridPartial*

5.4.4 Gestione delle prenotazioni

La gestione delle prenotazioni comprende lo sviluppo di tre pagine *Razor*: *AddReservationPopup*, *SelectUserPopup* e *UsersGridViewPartial*, con i rispettivi metodi del *controller* per permetterne la visualizzazione. *AddReservationPopup* è la pagina che permette all'utente di aggiungere manualmente una o più prenotazioni, ed è visibile dopo aver premuto il pulsante "Aggiungi prenotazioni" in figura 5.22. All'interno di essa sarà possibile inserire i dati della prenotazione, come visibile in figura 5.23. In particolare, per scegliere l'utente alla quale assegnare la prenotazione, viene utilizzato il *popup* *SelectUserPopup* che incorpora la vista parziale *UsersGridViewPartial*, contenente la griglia con i dati degli utenti selezionabili (vedi figura 5.24).

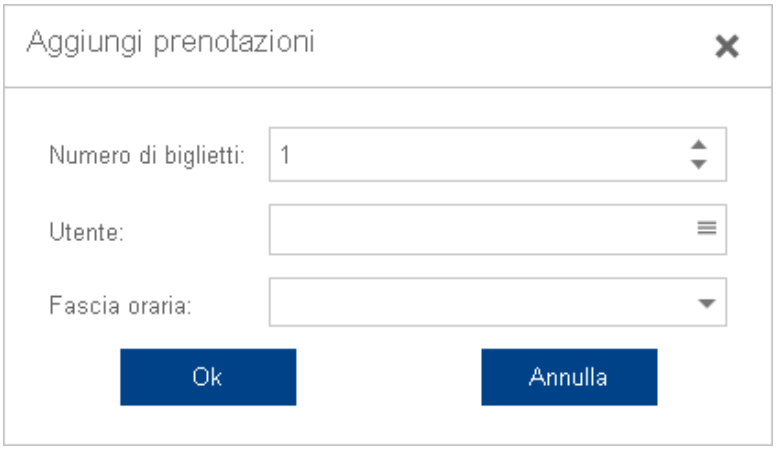


Figura 5.23: *AddReservationsPopup*

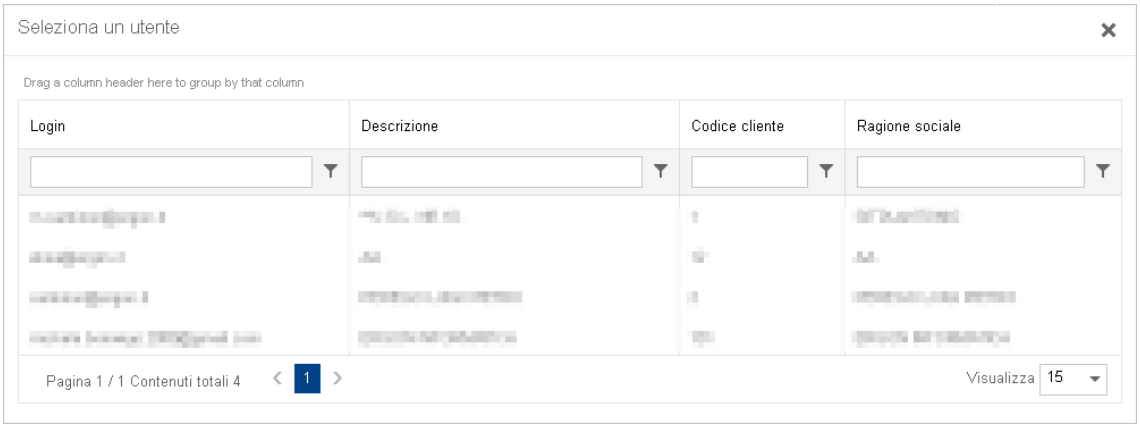


Figura 5.24: *SelectUserPopup* e *UsersGridViewPartial*

Capitolo 6

Verifica e validazione

6.1 Verifica

Il progetto di *stage* prevedeva inizialmente lo sviluppo di *test* sia in codice sorgente che documentale. Successivamente, dato che quasi tutto il *software* prodotto richiede interazione fisica con l'utente, è stato deciso insieme al *tutor* di procedere con dei *test* basati sull'esecuzione delle varie parti del progetto, in modo da verificarne il comportamento.

6.1.1 Web services

Considerando l'indipendenza tra i vari *endpoint*, è risultato conveniente scrivere ed eseguire i *test* per ognuno di essi appena ne era stata conclusa la codifica. In questo modo, è stato possibile avere un prodotto funzionante durante l'intero sviluppo. In particolare, per testare ogni singolo *endpoint* è stato utilizzato Postman[g], che permette facilmente di effettuare richieste *HTTP* specificando il metodo e i parametri. In figura 6.1 è riportato un esempio di *test* di un *endpoint* utilizzando Postman.

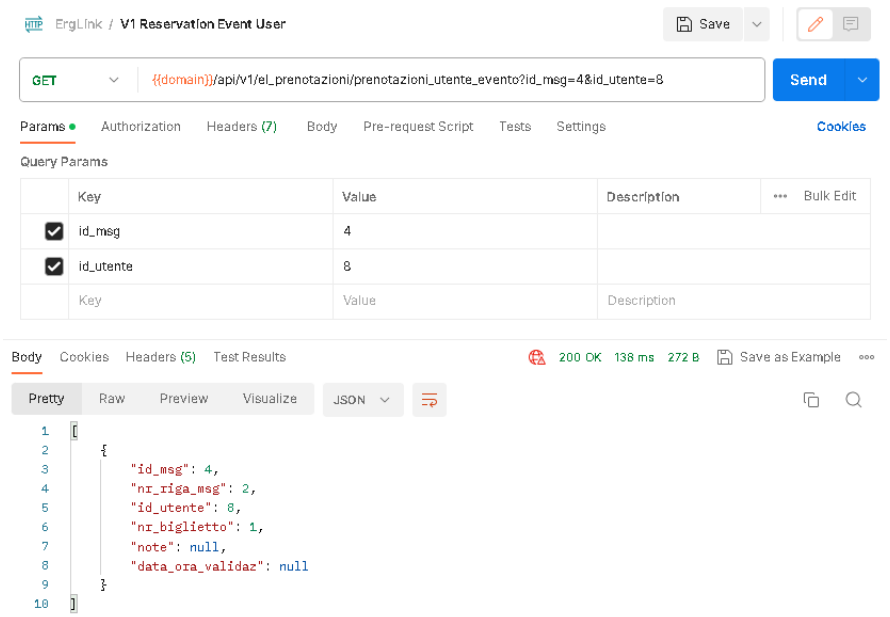


Figura 6.1: Esempio di *test* con *Postman*

6.1.2 App Erglink e Backoffice

Per quanto riguarda l'applicazione Erglink e il *backoffice* è stato deciso di semplificare l'attività di *testing*, eseguendo *test* manuali basati sull'esecuzione delle singole pagine, andando a valutarne il comportamento in una serie di scenari, comprendendo sia quelli considerati "corretti", sia quelli che rappresentano situazioni limite.

6.2 Validazione

6.2.1 Codice

La validazione del codice prodotto è stata effettuata dal *tutor*, che periodicamente prendeva visione del codice ad alto livello. L'obiettivo principale di questa procedura di validazione era quello di verificare che il codice rispettasse le caratteristiche desiderate in termini di correttezza e leggibilità.

6.2.2 Requisiti

Durante l'attività di *testing* è stato validato anche il raggiungimento dei requisiti tracciati durante l'analisi iniziale. Sono stati raggiunti tutti i requisiti obbligatori, mentre per quanto riguarda quelli desiderabili, non sono stati soddisfatti per mancanza di tempo. Nelle tabelle 6.1, 6.2 e 6.3 sono presenti i risultati della validazione dei requisiti.

Tabella 6.1: Tabella della validazione dei requisiti funzionali

Requisito	Risultato
RFO1	Soddisfatto
RFO2	Soddisfatto
RFO3	Soddisfatto
RFO4	Soddisfatto
RFO5	Soddisfatto
RFO6	Soddisfatto
RFO7	Soddisfatto
RFO8	Soddisfatto
RFO9	Soddisfatto
RFO10	Soddisfatto
RFO11	Soddisfatto
RFO12	Soddisfatto
RFO13	Soddisfatto

RFO14	Soddisfatto
RFO15	Soddisfatto
RFO16	Soddisfatto
RFO17	Soddisfatto
RFO18	Soddisfatto
RFO19	Soddisfatto
RFO20	Soddisfatto
RFO21	Soddisfatto
RFO22	Soddisfatto
RFO23	Soddisfatto
RFO24	Soddisfatto
RFO25	Soddisfatto
RFO26	Soddisfatto
RFO27	Soddisfatto

Tabella 6.2: Tabella della validazione dei requisiti qualitativi

Requisito	**Risultato**
RQO1	Soddisfatto
RQO2	Soddisfatto
RQO3	Soddisfatto
RQD4	Non soddisfatto
RQD5	Non soddisfatto

Tabella 6.3: Tabella della validazione dei requisiti di vincolo

Requisito	**Risultato**
RVO1	Soddisfatto
RVO2	Soddisfatto
RVO3	Soddisfatto
RVO4	Soddisfatto
RVO5	Soddisfatto
RVO6	Soddisfatto

Capitolo 7

Conclusioni

Come descritto nei capitoli precedenti, il prodotto finale consiste in una piattaforma di gestione degli eventi a supporto di Erglink, con la possiblità di creare e gestire eventi con le annesse prenotazioni. Nella tabella 7.1 viene illustrato il riepilogo del soddisfacimento degli obiettivi.

Tabella 7.1: Tabella degli obiettivi

Codice obiettivo	Descrizione obiettivo	Risultato
OB1	Sviluppo dei *web services* per la gestione del flusso di dati	Soddisfatto
OB2	Sviluppo *app* per iscrizione eventi	Soddisfatto
OB3	Sviluppo interfaccia *web* per creazione degli eventi	Soddisfatto
OB4	Acquisizione di competenze sullo sviluppo *cross-platform mobile*, interfacce *web* e *web services*	Soddisfatto

7.1 Conoscenze acquisite

Per la realizzazione di questo progetto di *stage* sono risultate fondamentali sia le nozioni apprese durante il corso di studi universitario, sia quelle apprese durante la scuola secondaria di secondo grado, tra le quali è risultata molto utile la buona conoscenza del linguaggio C#. Di seguito sono elencate le principali conoscenze acquisite:

- *framework Xamarin*, per la creazione di applicazioni *mobile* multipiattaforma;
- *framework* ASP .NET Core, per la creazione di interfacce *web* e *web services*.

Oltre alle conoscenze tecniche, durante l'attività di *stage* ho potuto migliorare anche competenze trasversali, come la capacità di analizzare un problema, il *problem solving* e la gestione delle risorse temporali, ad esempio per rispettare le scadenze del progetto. Un altro aspetto importante è sicuramente quello sociale. Infatti, in queste settimane di *stage* ho dovuto frequentare un vero luogo di lavoro, che aiuta a migliorare la capacità di relazionarsi con le persone.

Glossario

API (*Application Programming Interface*) intermediario software che permette a due applicazioni non correlate di comunicare tra loro. 5, 6, 38, 39

ASP .NET Core *framework* multipiattaforma e ad alte prestazioni per la creazione di *app* moderne abilitate per il *cloud* e connesse a *Internet*. 33, 35, 36, 38, 44, 60

C# linguaggio di programmazione multi-paradigma sviluppato da *Microsoft* che supporta tutti i concetti della programmazione orientata agli oggetti. 3, 33, 35, 36, 44, 60, 61

CRUD (*Create-Read-Update-Delete*) sono le quattro operazioni basilari della gestione persistente dei dati. 37, 38, 44

DexExpress *framework* utile per la creazione di interfacce utente. 33, 36

Entity Framework è un ORM(*Object-Relational-Mapper*) che permette di lavorare con dati relazionali utilizzando oggetti specifici del dominio. 38

Erglink piattaforma social sviluppata dall'azienda *Ergon Informatica Srl*. 1, 7–9, 37, 42, 47, 57, 60

ERP (*Enterprise Resource Planning*) software di gestione che integra tutti i processi aziendali e tutte le funzioni aziendali rilevanti, ad esempio vendite, acquisti, gestione magazzino, finanza o contabilità. 1

IDE (*Integrated Development Environment*) *software* che aiuta gli sviluppatori nella fase di programmazione con vari strumenti di analisi del codice e integrazione di altre componenti. 35

JSON (*JavaScript Object Notation*) è un formato per lo scambio dati basato sul linguaggio di programmazione *JavaScript*. 40, 46

NFC (*Near Field Communication*) tecnologia di ricetrasmissione che fornisce connettività senza fili bidirezionale a distanza a corto raggio. 5

Open source indica un software distribuito sotto i termini di una licenza *open source*, che ne concede lo studio, l'utilizzo, la modifica e la redistribuzione. 3, 6, 36

Postman *software* che permette di testare gli *endpoint* sviluppati, effettuando richieste *HTTP*. 56

Razor è una sintassi di *markup* per l'incorporamento di codice basato su *.NET* in pagine *Web*, ed è costituita da *markup Razor*, C# e *HTML*. 44, 46

REST (*Representational state transfer*) rappresenta un sistema di trasmissione di dati su *HTTP*. 38, 39

Server Consolidation processo di unificazione di più server in un unico server con il fine di ridurne il costo e il numero. 1

UML (*Unified Modeling Language*) linguaggio di modellazione che permette tramite l'utilizzo di modelli visuali, di analizzare, descrivere, specificare e documentare un sistema software. 8–15, 17, 20, 22–25, 28

XAML (*eXtensible Application Markup Language*) è un linguaggio di *markup* basato su *XML*, utilizzato per descrivere l'interfaccia grafica delle applicazioni basate sulla libreria *Windows Presentation Foundation*. 3

Capitolo 8

Bibliografia

Siti web consultati

Confronto Xamarin - .NET MAUI. URL: https://www.telerik.com/blogs/part-1-should-we-migrate-xamarin-forms-dotnet-maui-when.

Definizione ASP .NET Core. URL: https://learn.microsoft.com/it-it/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0.

Definizione di Server Consolidation. URL: https://www.geeksforgeeks.org/server-consolidation-in-cloud-computing/.

Definizione di termini. URL: https://www.wikipedia.org/.

Definizione di UML. URL: https://www.html.it/guide/guida-uml/.

Definizione Entity Framework. URL: https://learn.microsoft.com/en-us/aspnet/entity-framework.

Framework Xamarin.Forms. URL: https://learn.microsoft.com/it-it/xamarin/get-started/what-is-xamarin-forms.

Libreria Xamarin.Essentials. URL: https://learn.microsoft.com/en-us/xamarin/essentials/.

Pattern Model View ViewModel. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm.

Piattaforma Xamarin. URL: https://learn.microsoft.com/it-it/xamarin/get-started/what-is-xamarin.

Platform Divergence & Features. URL: https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/platform-divergence-abstraction-divergent-implementation.