

SUPSI

Gestione e Riservazione delle Piazze di Tiro

Studente/i

Matteo Arena

Relatore

Giuseppe Landolfi

Correlatore

-

Committente

**Base Logistica dell'Esercito
Monteceneri**

Corso di laurea

Ingegneria informatica TP

Codice progetto

C10799

Anno

2023/2024

Indice

INTRODUZIONE	12
MOTIVAZIONE E CONTESTO	13
PROBLEMA ATTUALE	14
3.1 Contesto e Necessità	14
3.2 Limitazioni nel Sistema Attuale	16
3.2.1 Mancanza di Visibilità in Tempo Reale	16
3.2.2 Assenza di Notifiche e Promemoria Automatizzati	16
3.2.3 Limitata Scalabilità	16
3.3 Impatti delle Limitazioni sul Personale e sulle Operazioni	16
STATO DELL'ARTE	17
4.1 Analisi delle tecnologie esistenti	17
4.1.1 Backend Frameworks	17
4.1.2 Frontend Frameworks	18
4.1.3 Database Systems	18
4.1.4 ORMs (Object-Relational-Mapping)	19
PROGETTAZIONE	20
5.1 Scelte fatte	20
5.2 Architettura del Sistema	21
5.2.1 Panoramica dell'architettura	21
5.3 Progettazione del Frontend	22
5.3.1 Struttura dei componenti	22
5.3.2 Interfaccia Utente	23
5.3.3 Comunicazione con il Backend	26
5.4 Progettazione del Backend	27
5.4.1 Struttura del Server	27
5.4.2 Progettazione API	28
5.5 Progettazione del Database	30

IMPLEMENTAZIONE	32
6.1 Gestione delle Lingue	32
6.1.1 File JSON	32
6.1.2 Configurazione del Sistema di Internazionalizzazione	33
6.1.3 Utilizzo Pratico	34
6.2 Processo di Autenticazione	37
6.2.1 Richiesta API	37
6.2.2 Lato Frontend	38
6.2.3 Lato Backend	41
6.3 Gestione dei File	42
6.3.1 FileContainer	42
6.3.2 Salvataggio File	43
6.3.3 Gestione Nome dei File	43
6.4 GeneralList e GeneralItem	44
6.4.1 GeneralItem	44
6.4.2 GeneralList	44
6.4.3 Utilizzo	45
6.5 Gestione Stati dell'utente	46
6.5.1 Approvazione Utenti	46
6.5.2 Modifica Ruolo	46
RISULTATI	47
7.1 Pagina di Login	47
7.2 Pagina di News	49
7.2.1 Vista Amministratore	49
7.2.2 Vista Normale	51
7.3 Barra di Navigazione	52
7.3.1 Utente mai approvato	52
7.3.2 Utente approvato	52
7.3.3 Amministratore	53
7.4 Calendario	54
7.5 Infrastrutture	55
7.5.1 Pagina Base	55
7.5.2 Creazione Pagine	56
7.5.3 Modifica Infrastrutture	56
7.5.4 Creazione Riservazione	57
7.6 Pagina Profilo	58
7.6.1 Informazioni utente	58
7.6.2 Approvazione Civile	58
7.7 Approvazione Utenti	59
7.8 Impostazioni	60
7.8.1 Pagina principale	60
7.8.2 Creazione Bersagli/Sedi	60

7.9 Pagina delle Riservazioni	61
7.9.1 Visualizzazione amministratore	61
7.9.2 Visualizzazione normale	62
 TEST	 63
8.1 Unit Testing	63
8.2 Test End-To-End	64
 CONCLUSIONI	 65
9.1 Risultati	65
9.2 Difficoltà Incontrate	65
9.3 Sviluppi Futuri	65

Indice delle figure

Figura 1 -- Processo di Riservazione Attuale.....	14
Figura 2 -- Diagramma Architettura.....	21
Figura 3 -- Mockup News.....	23
Figura 4 -- Mockup Login	24
Figura 5 -- Mockup Riservazione.....	24
Figura 6 -- Mockup Dashboard Amministratore	25
Figura 7 -- Diagramma E/R.....	30
Figura 8 -- Formattazione JSON Lingue	33
Figura 9 -- Import i18n	33
Figura 10 -- Configurazione i18n	34
Figura 11 -- Import i18n per le traduzioni	34
Figura 12 -- Dichiarazione t	34
Figura 13 -- Utilizzo traduzioni.....	35
Figura 14 -- LanguageSwitcher	35
Figura 15 -- Cambio lingua all'accesso	36
Figura 16 -- Richiesta API per Autenticazione	37
Figura 17 -- Risposta Richiesta	37
Figura 18 -- Richiesta API per il login	38
Figura 19 -- AuthContext.....	39
Figura 20 -- Richiesta API Generale.....	40
Figura 21 -- Richiesta API da Service	40
Figura 22 -- Controllo Token.....	41
Figura 23 -- Salvataggio file – <i>uploadDir</i>	43
Figura 24 -- GeneralItem	44
Figura 25 -- GeneralList.....	45
Figura 26 -- Risultato <i>Pagina di Login</i>	47
Figura 27 -- Risultato <i>Pagina Registrazione</i>	48
Figura 28 -- Risultato <i>Pagina News</i>	49
Figura 29 -- Risultato <i>Creazione News</i>	49
Figura 30 -- Risultato <i>Vista Utente News</i>	51
Figura 31 -- Risultato <i>contenuto News</i>	51
Figura 32 -- Risultato <i>NavBar utente non approvato</i>	52
Figura 33 -- Risultato <i>NavBar utente approvato</i>	52
Figura 34 -- Risultato <i>NavBar Amministratore</i>	53
Figura 35 -- Risultato <i>Calendario</i>	54
Figura 36 -- Risultato <i>Infrastrutture</i>	55
Figura 37 -- Risultato <i>Creazione Infrastrutture</i>	56
Figura 38 -- Risultato <i>Modifica Infrastruttura</i>	56
Figura 39 -- Risultato <i>Creazione riservazione</i>	57
Figura 40 -- Risultato <i>Pagina Profilo</i>	58
Figura 41 -- Risultato <i>Documenti non caricati</i>	58
Figura 42 -- Risultato <i>Approvazione Utenti</i>	59
Figura 43 -- Risultato <i>Impostazioni</i>	60

Figura 44 -- Risultato <i>Creazione Bersaglio</i>	60
Figura 45 -- Risultato <i>Riservazioni amministratore</i>	61
Figura 46 -- Risultato <i>Riservazioni utente</i>	62
Figura 47 -- Risultato Unit Test.....	64
Figura 48 -- Test Richiesta API corretta.....	64

Indice delle tabelle

Tabella 1: Chiamate GET	28
Tabella 2: Chiamate POST	29
Tabella 3: Chiamate GET	29
Tabella 4: Spiegazione stati utenti	30
Tabella 5: Tabella delle Lingue.....	32
Tabella 6: Stati FileContainer.....	42
Tabella 7: Campi FileContainer	42
Tabella 8: Input GeneralItem	44

Abstract

- TODO

Progetto assegnato

In questo capitolo vengono descritti i dettagli del progetto come sono stati commissionati all'interno della piattaforma fornita da SUPSI¹.

Descrizione

La gestione dell'infrastruttura militare viene fatta dalla Base Logistica dell'Esercito (BLEs). Per la riservazione di poligoni di tiro si procede inizialmente con una riservazione sommaria, tramite colloquio telefonico con il responsabile. In seguito, si passa alla riservazione di dettaglio tramite un formulario cartaceo consegnato dal cliente (militari o civili) al gestore una settimana prima dell'attività, contenente il dettaglio di cosa intende utilizzare. Il responsabile (BLEs) prepara l'infrastruttura, la consegna e si occupa della riconsegna tramite un altro protocollo, sempre cartaceo.

Nel caso di utilizzo da parte civile (a pagamento), la parte di controllo ore e dettagli di utilizzo viene anch'essa gestita tramite formulari cartacei utilizzati in seguito per il processo di fatturazione.

L'obiettivo di questo progetto è quello di creare un'applicazione, tipo "Booking" sull'esempio di quanto attualmente ampiamente usato dagli Hotel per la riservazione online.

Compiti

1. Creare un Database, (Cartina digitale?) delle varie infrastrutture che possono essere riservate (poligoni, sale riunioni, magazzini, palestre o campi da calcio per esempio).
2. Portale di riservazione (pagina WEB o applicazione Smartphone) in cui possono essere visualizzate le varie infrastrutture con le loro peculiarità, possibilità di utilizzo, oltre alla possibilità di richiedere delle modifiche (aggiunta bersagli, materiale sport, ecc).
3. CodiciQR: Generare codici QR per un luogo o di un impianto. La scansione di questo codice (ad esempio, con un telefono cellulare) indirizzerà l'utente all'infrastruttura richiesta nell'applicazione.
4. Creazione account. Creare account utente controllati (vidimazione di utilizzo) di modo che unicamente le persone autorizzate possano fare le richieste. Si possono inoltre caricare i file per la richiesta dell'autorizzazione.
5. Notifiche. I responsabili della gestione e le persone selezionate ricevono una notifica via e-mail e push sulla loro app ogni volta che viene inviata una nuova segnalazione, consentendo loro di rispondere immediatamente tramite il proprio smartphone.

¹ Vedi: <https://progettistudio.supsi.ch/dettaglio.php?p=C10799> (Consultato il 10.06.2024)

6. Protocollo consegna/ riconsegna. Anche la consegna ed il ritiro dell'infrastruttura dovrebbero avvenire con sistema informatizzato, senza utilizzo del cartaceo.
7. Fatturazione. Il controllo di cosa è stato utilizzato, quando e per quanto tempo viene tenuto elettronicamente. Grazie a questi dati e in base ad una lista prezzi preimpostata si possono preparare dei consuntivi da inoltrare alla fatturazione.
8. Informazioni. La pagina web oppure l'app servirà per vedere tutte le news riguardante i regolamenti, le prescrizioni di sicurezza oppure le caratteristiche che una società deve avere per richiedere l'autorizzazione all'utilizzo. Inoltre, al suo interno si potranno trovare le versioni aggiornate di tutti i documenti da compilare con le rispettive scadenze da rispettare.

Capitolo 1

Introduzione

Il progetto nasce dal bisogno di dover implementare un applicativo WEB per la gestione delle piazze tiro per l'esercito svizzero. L'obiettivo principale è far risparmiare tempo a chi è incaricato al momento della gestione delle riserve e rendere tutto il processo più facile lato utilizzatore.

Per risolvere questo problema si è pensato di utilizzare React combinato con Express.js e PostgreSQL per creare un sito web semplice ed efficace per la riservazione delle principali infrastrutture dell'esercito svizzero.

Il **capitolo 2** del documento offre una panoramica delle ragioni e delle circostanze che hanno influenzato la creazione del progetto. Viene inoltre fornita una prima introduzione agli obiettivi principali che rappresentano il cuore di questo lavoro.

Continuando al **capitolo 3** viene descritto i problemi presenti nel sistema attuale, con tutte le problematiche di tempo e di procedure che sono molto lunghe e poco intuitive. Grazie a questo capitolo vengono definite le parti più essenziali che l'applicativo dovrà risolvere.

Il **capitolo 4** mostra lo stato dell'arte, con le principali tecnologie disponibili al giorno d'oggi, elencandone i pregi ed i difetti.

Proseguendo il **capitolo 5** mostra le scelte progettuali del progetto, come le tecnologie che si è deciso di utilizzare. Inoltre, è presente anche la descrizione di tutta l'infrastruttura utilizzata, partendo dal database, andando al Backend e finendo con il Frontend.

Il **capitolo 6** mette in atto il capitolo 5 spiegando le parti più interessanti di codice. Questo mostrando come vengono fatte alcune operazioni tra le più essenziali, passando per ogni passaggio e mostrando esplicitamente come le richieste vengono gestite da ogni componente

Nel **capitolo 7** il progetto è già completo e mostra tutte le pagine disponibili all'interno dell'applicativo, indicando anche tutte le azioni che ogni tipo di utente può effettuare (dall'utente normale sino all'amministratore).

I test sono racchiusi nel **capitolo 8**, dove vengono mostrati i principali test effettuati sull'applicativo, dividendoli in due macrocategorie.

Infine, nel **capitolo 9** vengono tratte le conclusioni sul progetto, descrivendo brevemente il risultato finale, le difficoltà incontrate durante lo sviluppo e quanto riguarda lo sviluppo futuro dell'applicativo.

Capitolo 2

Motivazione e Contesto

In questo capitolo vengono delineate le motivazioni che mi hanno portato a sviluppare un gestionale per l'esercito svizzero e il contesto in cui è stato realizzato questo progetto.

Durante il Bachelor in Ingegneria Informatica presso la Scuola Universitaria Professionale della Svizzera Italiana (SUPSI), ogni studente è chiamato a completare un lavoro di diploma. Questo progetto rappresenta una sintesi delle competenze acquisite durante gli studi e offre l'opportunità di applicarle in un contesto pratico. Gli studenti hanno la possibilità di scegliere tra una vasta gamma di progetti proposti dai relatori, spaziando tra vari ambiti tecnologici e settori di applicazione.

La scelta di questo progetto, in particolare, è stata guidata dal mio desiderio di sviluppare un applicativo che potesse essere utilizzato nel tempo, contribuendo concretamente a migliorare un processo esistente. Inoltre, volevo affrontare personalmente lo sviluppo completo di un applicativo web, sfruttando framework moderni come React. Questa sfida mi ha permesso di approfondire le mie competenze nello sviluppo front-end e di gestire tutte le fasi di creazione di un'applicazione web, dall'ideazione fino alla realizzazione finale.

Il progetto è stato seguito da Giuseppe Landolfi, relatore presso SUPSI, e dall'esercito svizzero, che ha collaborato attivamente attraverso riunioni ricorrenti, fornendo feedback e indicazioni cruciali per la definizione dei requisiti e l'implementazione del sistema.

Capitolo 3

Problema attuale

Questo capitolo analizza in dettaglio il procedimento attuale della riservazione delle infrastrutture, riportando tutte le problematiche e le scomodità da esse causate.

3.1 Contesto e Necessità

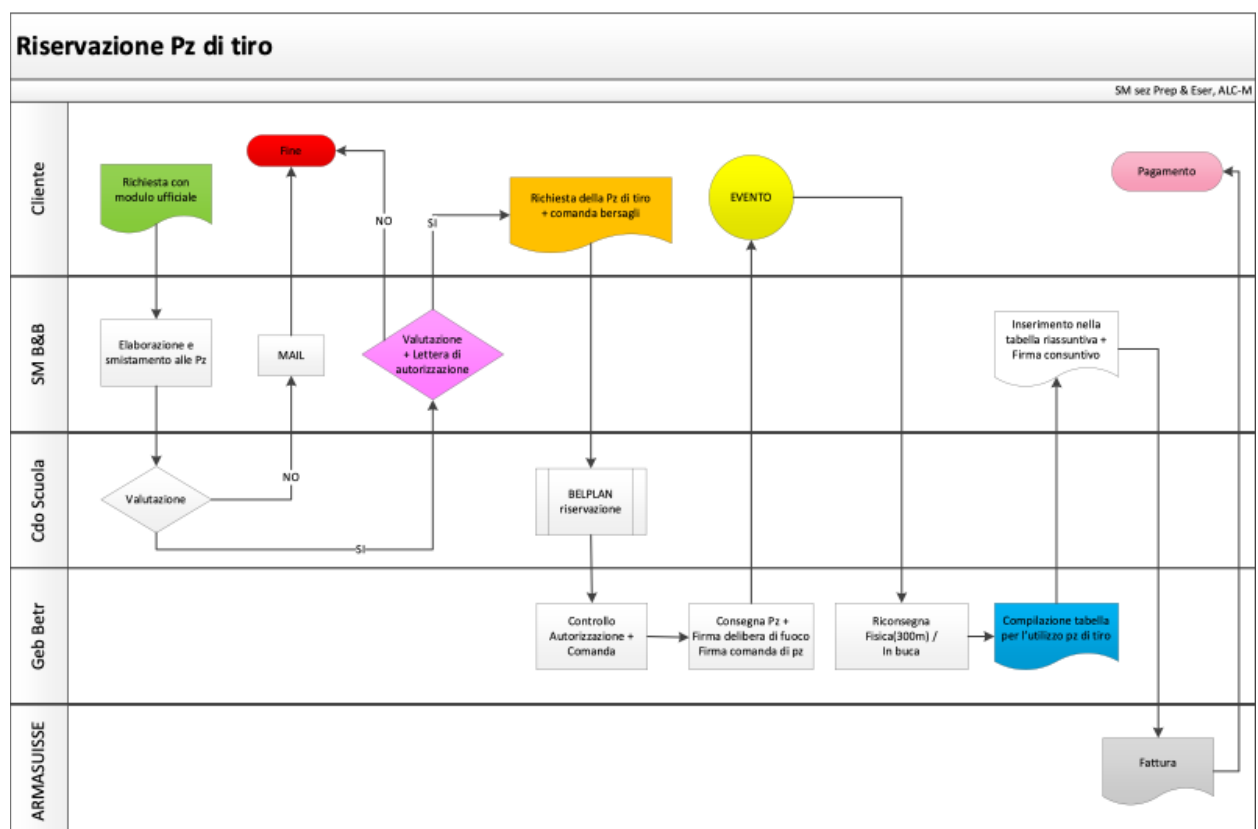


Figura 1 -- Processo di Riservazione Attuale

La gestione delle prenotazioni, illustrata nell'immagine soprastante, evidenzia la complessità e la lunghezza dei passaggi necessari per la prenotazione di una semplice infrastruttura. Il flusso descritto si riferisce in particolare al processo che un civile deve seguire, che risulta essere il più lungo e complesso.

Il primo passo richiede l'approvazione tramite un apposito formulario, senza il quale risulta impossibile riservare le varie infrastrutture.

Una volta compilato il formulario e ottenuta l'approvazione, è possibile inoltrare una richiesta per una piazza di tiro tramite la "Comanda Bersagli", un modulo in cui il richiedente deve specificare i bersagli desiderati durante l'evento. La richiesta, inoltrata tramite e-mail, viene poi confrontata con le richieste già presentate dall'esercito, che hanno sempre la precedenza assoluta.

I civili possono quindi utilizzare le infrastrutture unicamente quando queste sono disponibili, tenendo presente che le richieste dell'esercito possono annullare le riserve civili già confermate, anche con un solo giorno di preavviso.

Se la richiesta civile non viene annullata, si procede all'utilizzo dell'infrastruttura nel giorno stabilito. Al termine dell'evento, viene compilata una tabella con le munizioni utilizzate.

L'ultimo passaggio riguarda la fatturazione, attualmente effettuata tramite un file PDF generato automaticamente da un file Excel e inviato agli utenti su base semestrale, salvo casi eccezionali che richiedono un preventivo per ogni utilizzo.

3.2 Limitazioni nel Sistema Attuale

3.2.1 Mancanza di Visibilità in Tempo Reale

Non avendo la possibilità di sapere quando le infrastrutture sono libere/occupate, gli utenti sono obbligati ad inoltrare richieste per date desiderate che vengono poi, nella maggiorparte dei casi, modificate in altre date. Questo viene fatto al momento principalmente tramite colloquio telefonico, dove la persona incaricata della gestione delle riservezioni arriva a trovare un giorno libero che vada bene anche per l'utilizzatore.

3.2.2 Assenza di Notifiche e Promemoria Automatizzati

L'assenza di notifiche per l'utente porta alla remota possibilità che un utilizzatore si possa addirittura dimenticare di una riservezione effettuata. Inoltre, in caso di cancellazione, la notifica viene attualmente fatta manualmente tramite e-mail/colloquio telefonico, portando poi alla "contrattazione" di una nuova data con l'utente.

3.2.3 Limitata Scalabilità

La gestione manuale di tutte le riservezioni, porta al fatto che se un domani gli utenti dovessero aumentare a dismisura il lavoro per la gestione delle riservezioni diventerebbe eccessive, richiedendo magari addirittura una persona dedicata unicamente a questo

3.3 Impatti delle Limitazioni sul Personale e sulle Operazioni

Tutte queste inefficienze portano il personale incaricato della gestione delle riservezioni ad avere un carico di lavoro significativo maggiore, molto del quale ripetitivo.

Capitolo 4

Stato dell'arte

Questo capitolo porta un'analisi di tutte le tecnologie esistenti rilevanti per il progetto, esaminando i principali framework per backend e frontend, i sistemi di database e gli ORM. Infine, viene detto quali tecnologie sono state scelte e perché.

4.1 Analisi delle tecnologie esistenti

4.1.1 Backend Frameworks

Express.js (Node.js)

Progettato per costruire applicazioni web e API. È noto per la sua semplicità e per l'ampia disponibilità di middleware che consentono di estendere le funzionalità in modo modulare.

Django (Python)

Incoraggia lo sviluppo rapido e il design pulito e pragmatico. Viene fornito con molte funzionalità pronte all'uso, come il sistema di autenticazione, l'amministrazione integrata e l'ORM.

Spring Boot (Java)

Semplifica la creazione di applicazioni stand-alone, pronte per la produzione, basate su Spring. Fornisce configurazioni automatiche e predefinite per velocizzare lo sviluppo.

4.1.2 Frontend Frameworks

React

Sviluppata da Facebook per la costruzione di interfacce utente. Basato sui componenti, React permette di costruire UI modulari e reattive, con un'efficace gestione dello stato.

Angular

Sviluppato da Google, basato su TypeScript. Offre una soluzione integrata per la costruzione di applicazioni web complesse, con strumenti per il data binding, la gestione dello stato e la comunicazione con il backend.

4.1.3 Database Systems

PostgreSQL

Noto per la sua conformità agli standard SQL e per la sua robustezza. Supporta una vasta gamma di funzionalità avanzate, come la gestione delle transazioni ACID, le materialized view, i trigger, gli indici complessi e i tipi di dati personalizzati.

MySQL

Noto per la sua velocità e affidabilità. È ampiamente utilizzato in applicazioni web, in particolare in combinazione con PHP e LAMP stack (Linux, Apache, MySQL, PHP).

SQLite

Embedded, leggero e self-contained. Non richiede un server separato e archivia i dati in un singolo file. È comunemente utilizzato per applicazioni mobili, piccole applicazioni desktop e test.

MongoDB

Database NoSQL orientato ai documenti, che archivia i dati in formato BSON (una forma binaria di JSON). È progettato per gestire grandi quantità di dati non strutturati e semi-strutturati.

4.1.4 ORMs (Object-Relational-Mapping)

Sequelize (Node.js)

Supporta vari database SQL come PostgreSQL, MySQL, SQLite, e MariaDB. Fornisce un'interfaccia di alto livello per interagire con il database utilizzando oggetti JavaScript invece di scrivere direttamente query SQL.

TypeOrm (JavaScript/TypeScript)

Supporta i database relazionali come MySQL, PostgreSQL, SQLite e SQL Server. È stato progettato per essere utilizzato con TypeScript e offre un supporto nativo per decoratori e tipi di dati di TypeScript.

Entity Framework (C#/.NET)

ORM sviluppato da Microsoft per il framework .NET. Consente agli sviluppatori di lavorare con un database utilizzando oggetti .NET, eliminando la necessità di gran parte del codice SQL scritturale.

Hibernate (Java)

Utilizzato per mappare oggetti Java a tabelle di database relazionali. Fornisce un framework per l'implementazione della persistenza e la gestione delle relazioni complesse tra entità.

Django ORM (Python)

Consente di interagire con il database utilizzando modelli Python, mappando automaticamente le classi Python alle tabelle del database.

Capitolo 5

Progettazione

Descrizione di tutte le scelte progettuali prese, divise per Backend, Frontend e Database.

5.1 Scelte fatte

Ho scelto di utilizzare Sequelize con PostgreSQL, React, ed Express.js per il mio progetto per creare un'architettura full-stack robusta, scalabile e moderna. PostgreSQL, grazie alla sua conformità agli standard SQL e alle sue avanzate capacità di gestione dei dati, rappresenta una scelta eccellente per la base dati, offrendo sia affidabilità che flessibilità. Sequelize, come ORM, facilita l'interazione con il database, permettendo di scrivere codice più leggibile e manutenibile, e gestendo in modo efficiente le migrazioni e le relazioni tra i dati. Per il backend, Express.js offre un framework minimalista ma potente, che consente di costruire API performanti con grande flessibilità. Infine, React è stato scelto per il frontend grazie alla sua capacità di creare interfacce utente reattive e modulari, garantendo una user experience dinamica e moderna. L'integrazione di queste tecnologie permette di sviluppare un'applicazione web che è allo stesso tempo performante, facile da mantenere, e pronta a scalare con le esigenze future.

5.2 Architettura del Sistema

5.2.1 Panoramica dell'architettura

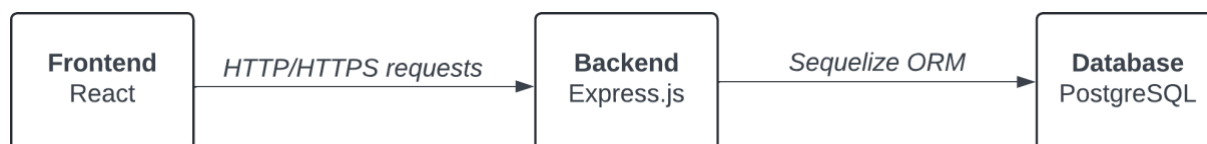


Figura 2 -- Diagramma Architettura

L'architettura del sistema è stata progettata per suddividersi in tre livelli principali, **frontend**, **backend** e **database**. Questo tipo di architettura permette una chiara divisione dei compiti che ogni livello deve fare, portando il codice ad essere più pulito e facile da scalarare.

Frontend

Il frontend è sviluppato interamente utilizzando React, libreria JavaScript che permette di costruire interfacce WEB dinamiche e modulabili. Questo livello si occupa principalmente della comunicazione diretta con l'utente, gestendo i vari click e fornendo una risposta ad ogni interazione effettuata. Inoltre questo livello comunica con il Backend unicamente tramite chiamate API.

Backend

Implementando utilizzando Express.js, il suo ruolo principale è quello di gestire la logica dell'applicazione e di fungere da intermediario tra il database ed il frontend. Questo viene fatto tramite le API esposte che permettono al frontend di interagire con i dati, che vengono preparati per essere presentati all'utente.

Database

Il database utilizzato è PostgreSQL, un sistema di gestione di database relazionali che offre affidabilità e conformità agli standard SQL. I dati vengono gestiti attraverso Sequelize, un ORM per Node.js che facilita l'interazione con il database, mappando le tabelle del database a modelli JavaScript.

Lo scopo di questa architettura è avere un'applicazione che sia scalabile facilmente per la possibilità di aggiungere nuove funzionalità in futuro.

5.3 Progettazione del Frontend

5.3.1 Struttura dei componenti

Dal lato del Frontend i vari componenti sono stati suddivisi in sette macrocategorie con compiti completamente diversi. Le categorie sono le seguenti:

Assets

Questa categoria contiene tutti i file multimediali necessari per il frontend, come icone e file di traduzione. I file sensibili legati agli utenti non sono salvati qui per garantire la sicurezza, ma sono gestiti dal backend.

Components

Include i componenti generici e riutilizzabili, che possono essere utilizzati in altri contesti applicativi. Questi componenti sono personalizzati e dichiarati in modo da poter essere riutilizzati, riducendo così la duplicazione del codice.

Context

Utilizzato principalmente per l'autenticazione, questo modulo fornisce accesso alle informazioni dell'utente e al token di autenticazione a qualsiasi componente dell'applicazione, facilitando la gestione dello stato globale.

Hooks

Contiene funzioni che gestiscono il comportamento delle pagine o dei modelli del database. Questi hook permettono di estrarre e visualizzare i dati direttamente nelle pagine, basandosi sui servizi per recuperare i dati dal backend.

Pages

Questa categoria rappresenta le pagine principali dell'applicazione, dove ogni pagina contiene la propria logica specifica. La pagina principale, BasePage, gestisce la visualizzazione delle pagine desiderate e, se necessario, carica i componenti correlati.

Services

Comprende i moduli che si occupano della comunicazione con il backend. Il loro compito principale è effettuare chiamate API e restituire i risultati all'interfaccia utente.

Utils

Qui si trovano tutte le funzioni utili per l'applicazione, che vengono riutilizzate in più parti del codice, come la traduzione del ruolo dell'utente da ID a stringa.

5.3.2 Interfaccia Utente

Per il design dei Mockup è stato scelto Figma, software per la progettazione di interfacce utente, tra le quali applicazioni e siti web.

Pagina di News

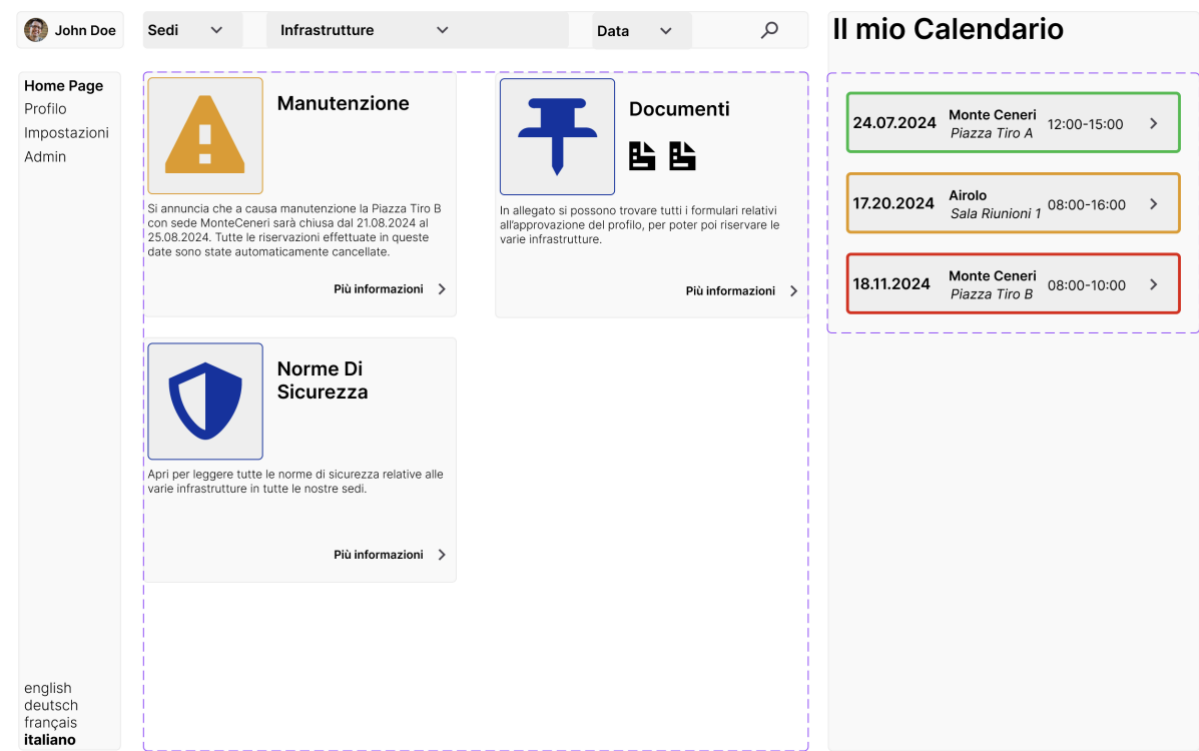


Figura 3 -- Mockup News

La pagina delle News è quella dedicata a tutti gli aggiornamenti che gli amministratori vogliono dare agli utenti. Come si può notare in questa pagina è possibile caricare un numero indefinito di notizie, le quali possono contenere immagini e/o file, permettendo di descrivere ad esempio il motivo per il quale una infrastruttura non è disponibile in uno specifico periodo.

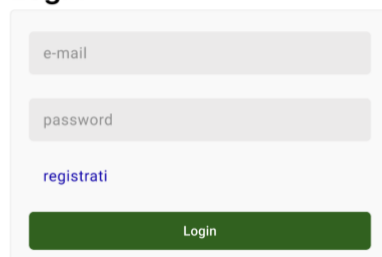
Nella parte destra della pagina è possibile vedere, riassunti, i propri impegni, mostrando la data, il luogo, l'infrastruttura, l'orario e lo stato (nel caso del mockup si è pensato verde in caso di conferma, arancione per le richieste in attesa di conferma e rosso per quelle cancellate).

Oltre alle news in alto è possibile, tramite l'apposita barra di ricerca, trovare l'infrastruttura desiderata nel periodo scelto, vedendo se essa è disponibile o meno.

Infine nella parte sinistra della pagina è possibile navigare tra le varie pagine e, soprattutto, cambiare la lingua del sistema.

Pagina di Login

Login

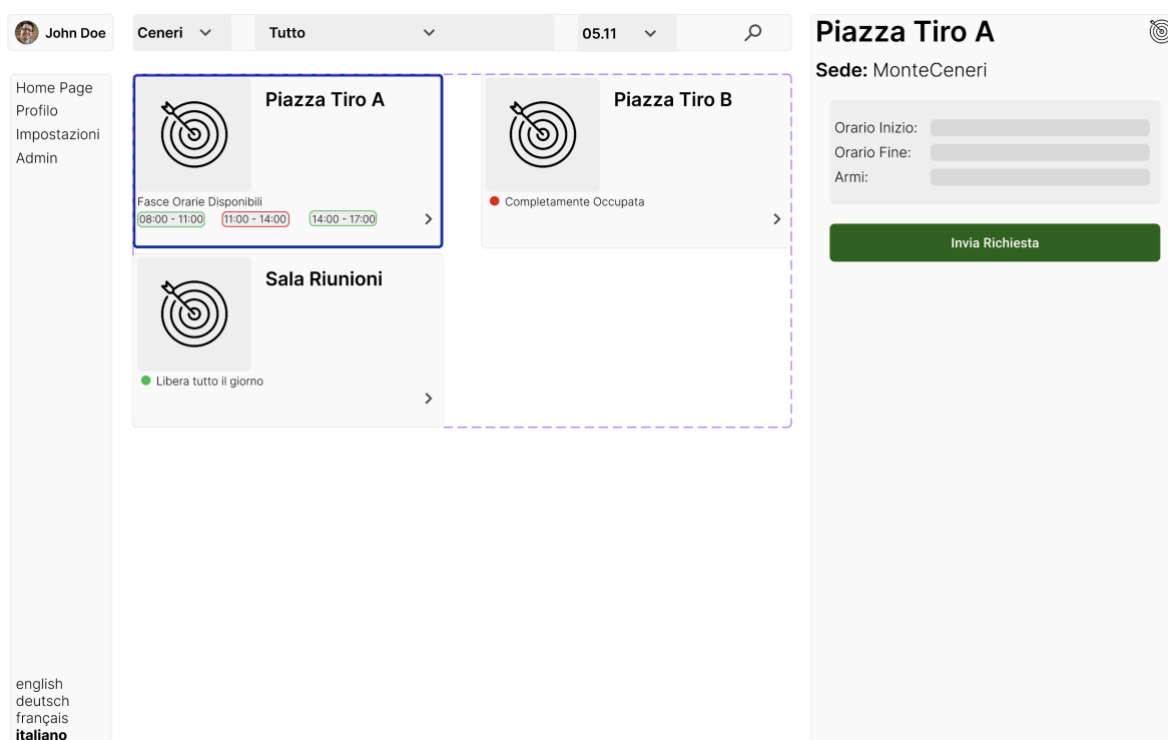


The login form consists of two input fields: 'e-mail' and 'password'. Below the 'password' field is a link labeled 'registrati' in blue. At the bottom is a green button labeled 'Login'.

Figura 4 -- Mockup Login

La pagina di Login, molto semplice ed essenziale, permette unicamente agli utenti già registrati di accedere e a quelli che non lo sono di fare la registrazione per poter poi accedere al portale

Pagina di Riservazione



The reservation page has a complex layout. At the top, there is a header bar with a user profile 'John Doe', location 'Ceneri', a dropdown menu 'Tutto', a time '05.11', and a search icon. On the left is a sidebar with navigation links: 'Home Page', 'Profilo', 'Impostazioni', 'Admin', and a language selector (english, deutsch, français, italiano). The main content area is divided into two columns. The left column contains three reservation cards: 'Piazza Tiro A' with available time slots (08:00-11:00, 11:00-14:00, 14:00-17:00), 'Piazza Tiro B' which is 'Completamente Occupata', and 'Sala Riunioni' which is 'Libera tutto il giorno'. The right column is dedicated to 'Piazza Tiro A', showing the location 'Sede: MonteCeneri' and a form to request a reservation with fields for 'Orario Inizio', 'Orario Fine', and 'Armi', followed by a green 'Invia Richiesta' button.

Figura 5 -- Mockup Riservazione

La pagina di riservazione permette all'utente di vedere le infrastrutture disponibili e di riservarle, specificando tutte le informazioni richieste (tra le quali orario di inizio e fine per il calcolo del preventivo).

Come si può notare la parte di navigazione, a sinistra, rimane fissa per ogni pagina e quella di destra varia in base alla selezione e alla pagina nella quale ci si trova.

Dashboard Amministratore

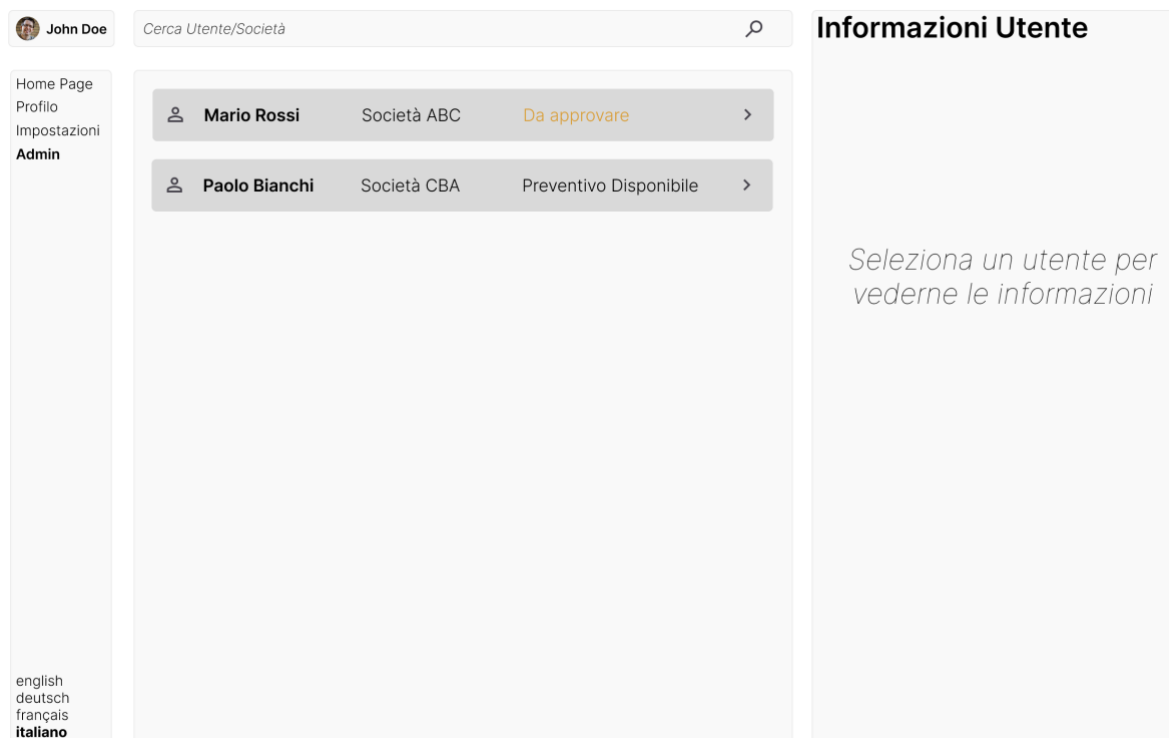


Figura 6 -- Mockup Dashboard Amministratore

La dashboard amministratore, dedicata unicamente agli amministratori, consente a questi ultimi di vedere tutte le informazioni di un utente, tra i quali anche lo stato corrente. Questa viene usata anche per l'approvazione iniziale degli utenti e per la generazione dei preventivi.

Questa è pensata per essere la pagina principale con la quale l'amministratore interagisce con gli utilizzatori dell'applicativo.

In questa pagina la barra di ricerca viene riutilizzata, permettendo all'amministratore di cercare tra i vari utenti.

5.3.3 Comunicazione con il Backend

Per la comunicazione con il Backend è stato deciso di utilizzare esclusivamente chiamate API RESTful, garantendo così una separazione chiara e ben definita tra i due livelli dell'applicazione. Questa scelta permette di mantenere una struttura modulare e scalabile, facilitando la manutenzione e l'evoluzione del sistema.

Le API implementeranno come base i metodi CRUD (Create, Read, Update, Delete), coprendo le operazioni fondamentali su ciascun modello. Tuttavia, per garantire un'elevata efficienza e mantenere la logica più complessa all'interno del backend, alcuni modelli includeranno endpoint API personalizzati. Questa strategia evita di sovraccaricare il Frontend con logiche complesse, migliorando la coesione e riducendo il rischio di duplicazione del codice.

Per quanto riguarda la gestione degli errori, ogni chiamata API restituirà non solo il codice di stato HTTP appropriato, ma anche un messaggio descrittivo che indicherà chiaramente la causa dell'errore. Questo approccio facilita il debug e la risoluzione dei problemi, migliorando l'esperienza di sviluppo e la robustezza complessiva del sistema.

5.4 Progettazione del Backend

5.4.1 Struttura del Server

Come per il Frontend, pure qua i vari file sono stati suddivisi in base loro comportamento in delle cartelle.

config/: Contiene i file di configurazione, come la configurazione del database e altre impostazioni globali.

controllers/: In questa cartella sono definiti i controller, che gestiscono la logica di business e interagiscono con i modelli per rispondere alle richieste del client.

models/: Contiene i modelli ORM (ad esempio, Sequelize), che rappresentano le entità del database.

routes/: Definisce le rotte dell'applicazione, mappando le richieste HTTP ai controller appropriati.

middleware/: Contiene i middleware personalizzati che gestiscono attività comuni come l'autenticazione.

app.js: File principale che inizializza l'applicazione Express definisce i middleware e le rotte principali.

server.js: Punto di ingresso dell'applicazione, dove viene avviato il server.

5.4.2 Progettazione API

Di seguito vengono illustrate le chiamate API che sono supportate dall'applicativo per ogni model. Le chiamate base CRUD non sono riportate, ma per ognuno dei modelli queste chiamate sono supportate (*/<model>/<method>*).

Gli EndPoint base sono i seguenti:

- /auth
- /users
- /infrastructures
- /estimates
- /pages
- /bookings
- /pageTypes
- /documents
- /headQuarters
- /infrastructureTypes
- /user_roles
- /targets

Tutti gli Endpoint aggiuntivi sono riportati nelle seguenti tabelle, in base al metodo supportato:

GET

Endpoint	Descrizione
/users/role/<roleId>	Ritorna tutti gli utenti con il ruolo specificato.
/users/<email>	Data l'email dell'utente, ne ritorna tutte le informazioni.
/infrastructures/type/<id>	Ritorna tutte le infrastrutture di un tipo specifico.
/pages/type/<id>	Ritorna tutte le pagine di un tipo.
/bookings/infrastructure/<infrastructureId>	Dato l'id di un'infrastruttura, ritorna tutte le sue prenotazioni.
/bookings/user/<id>	Dato un utente, ritorna le sue prenotazioni.
/documents/user/<id>	Ritorna tutti i documenti di un utente
/documents/title/<title>	Dato il nome di un documento lo ritorna

Tabella 1: Chiamate GET

POST

Endpoint	Descrizione
/auth/register	Crea l'utente all'interno del Database.
/auth/login	Controlla se l'utente è presente all'interno del Database, in caso positivo ritorna un Token.
/pages/create	Permette di creare una pagina, passando un file tra i dati.
/documents/create	Sovrascrizione del metodo base, permette di caricare anche un file
/infrastructureTypes/targets/id	Aggiunge la possibilità di riservare un target ad un infrastruttura

Tabella 2: Chiamate POST

PUT

Endpoint	Descrizione
/users/nextStatus/<email>	Dato un utente, lo fa proseguire nella procedura di approvazione.
/users/changeRole/<email>	Modifica il ruolo di un utente a quello stabilito.
/users/changeLanguage/<email>	Modifica la lingua di default di utente in quella passata.
/users/approve/<email>	Approva un utente, permettendogli di fare prenotazioni.
/users/removeApproval/<email>	Toglie l'approvazione ad un utente, togliendo pure la possibilità di fare prenotazioni.
/pages/update/<id>	Permette la modifica di una pagina, passando pure un file.
/bookings/cancelBooking/<id>	Mette lo stato di una prenotazione su "Cancellato", all'interno del database però rimarrà
/documents/update/<id>	Permette di modificare un documento, caricando anche un file

Tabella 3: Chiamate GET

DELETE

Tutte le chiamate DELETE si basano sui metodi CRUD base.

5.5 Progettazione del Database

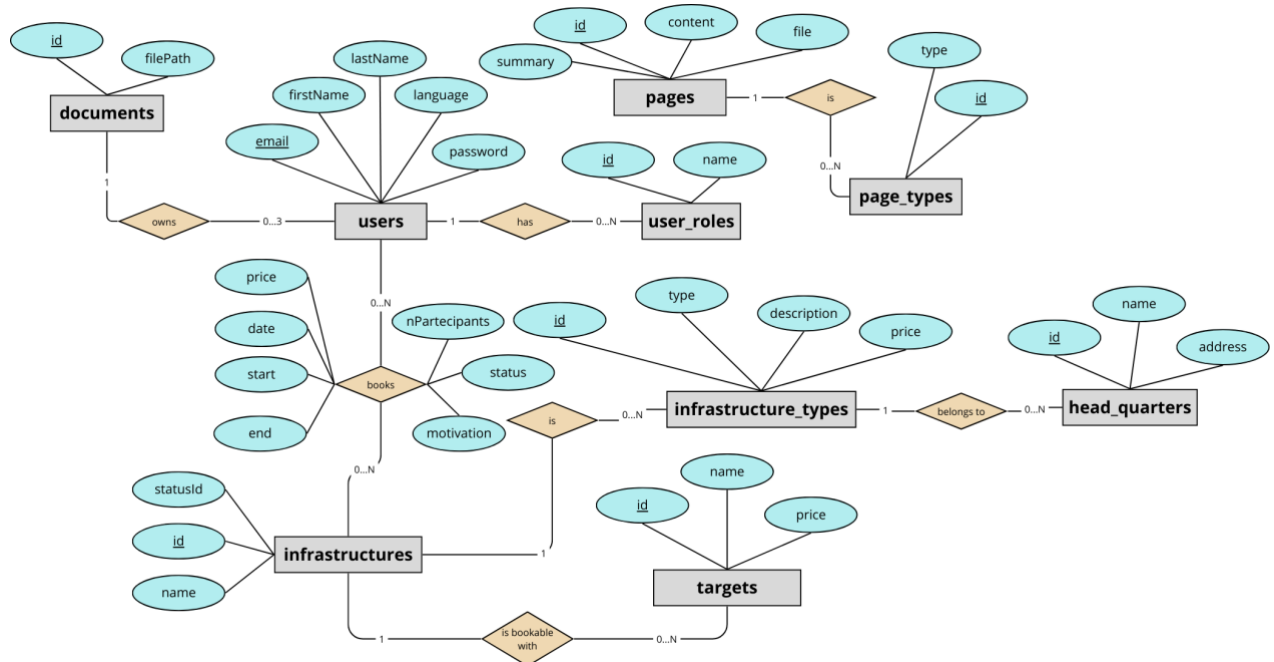


Figura 7 -- Diagramma E/R

Nell'immagine sovrastante è raffigurato il diagramma Entità/Relazione rappresentante l'intero sistema. Di seguito sono riportate le Entità:

documents: File che l'utente può caricare, all'interno di questa tabella sono salvati i documenti riguardanti l'approvazione dell'utente. Ogni documento contiene un identificativo, l'utente proprietario del file ed infine il percorso del file all'interno del Backend.

user_roles: Tabella rappresentante i vari tipi di ruoli che un utente può avere, ad ogni id è associato il nome del ruolo.

users: Utenti del sistema, del quale viene salvato nome, cognome, e-mail, stato, lingua e password (criptata). Per quanto riguarda lo stato esso funziona secondo la seguente tabella:

Stato	Significato
0	Utente appena registrato, non è mai stato autenticato.
1	Utente che ha inviato i documenti per la prima approvazione.
2	Utente già approvato in passato, per il quale però l'approvazione è scaduta.
3	Utente già approvato in passato, al quale è scaduta l'approvazione ma ha inviato i documenti per essere approvato di nuovo.
4	Utente approvato, che può effettuare prenotazioni.

Tabella 4: Spiegazione stati utenti

page_types: Categorie di pagine che si possono creare.

pages: Pagine di news, all'interno è presente anche un campo dedicato ai file, sempre il path di essi, per permettere di associare dei file ad una news

targets: Bersagli disponibili in tutte le infrastrutture, ad essi sono associati nome e prezzo.

infrastructure_types: Tipologie di infrastrutture che si possono creare, queste sono legate ad una sede (head_quarter).

Infrastructures: Infrastrutture che si possono riservare, queste hanno un nome che viene assegnato in automatico (dal Frontend)

head_quarters: Sedi dove è possibile riservare le varie infrastrutture, ognuna ha un nome ed un indirizzo

Capitolo 6

Implementazione

In questo capitolo vengono descritte le parti di codici più interessanti all'interno del progetto che necessitano una spiegazione più approfondita.

6.1 Gestione delle Lingue

6.1.1 File JSON

Per ogni lingua è stato creato un file JSON contenente tutte le stringhe necessarie all'applicativo. Questi documenti si possono trovare all'interno della cartella *frontend/src/assets/locales/<codice_lingua>*. Le lingue implementate sono riportate nella tabella sottostante.

Codice	Lingua
it	Italiano
en	Inglese (lingua di default nell'applicativo)
de	Tedesco
fr	Francese

Tabella 5: Tabella delle Lingue

All'interno ogni file contiene una sezione dedicata alla pagina nella quale viene utilizzata (ad esempio *News Page* o *Profile*) e all'interno di essa tutte le stringhe necessarie, per ogni lingua. Nell'esempio riportato si può vedere come è stato formattato il file per gestire le traduzioni della navigazione.


```

{
  "navigation": {
    "news_page": "News Page",
    "infrastructure": "Infrastructures",
    "profile": "Profile",
    "settings": "Settings",
    "user_approval": "User Approval",
    "reservations": "Reservations",
    "logout": "Logout"
  }
}

```

Figura 8 -- Formattazione JSON Lingue

Questo formato è stato utilizzato per ogni pagina dell'applicativo, portando ad una scalabilità elevata, siccome per implementare una lingua è necessario unicamente aggiungere un nuovo file di traduzioni.

6.1.2 Configurazione del Sistema di Internazionalizzazione

Import

Come prima cosa vengono importate le librerie ed i file essenziali per il funzionamento delle traduzioni. Per gestire ciò è stato scelto di basarsi sulla libreria *i18next* che viene utilizzata tramite il pacchetto *react-i18next*. Oltre a questo, vengono, ovviamente, importati i file contenenti le traduzioni, nel caso del progetto italiano, francese, tedesco ed inglese.

```

import i18n from "i18next";
import { initReactI18next } from "react-i18next";

// Importa i file di traduzione
import translationEN from "../assets/locales/en/translation.json";
import translationIT from "../assets/locales/it/translation.json";
import translationFR from "../assets/locales/fr/translation.json";
import translationDE from "../assets/locales/de/translation.json";

```

Figura 9 -- Import i18n

Configurazione i18n

```
i18n.use(initReactI18next).init({
  resources,
  lng: "en",
  fallbackLng: "en",
  interpolation: {
    escapeValue: false,
  },
});
```

Figura 10 – Configurazione i18n

La configurazione del componente di internazionalizzazione su React è molto semplice ed intuitiva, come prima cosa vengono impostate le lingue, con il rispettivo codice attraverso l'array *resources*, dopodiché viene indicata (tramite *lng*) la lingua di default e quella di fallback. Quest'ultima viene usata in caso che, nel processo di impostazione di una lingua, venga impostata una lingua che non esista all'interno dell'array definito, questo permette dunque di visualizzare comunque dei dati comprensibili.

Come ultima operazione viene impostato *escapeValue* a false, siccome React gestisce già l'escaping in modo sicuro.

6.1.3 Utilizzo Pratico

Come esempio pratico viene utilizzata la *NavBar*, componente dedicato alla gestione degli spostamenti dell'utente tra le varie pagine.

La prima operazione che viene fatta è semplicemente importare il metodo necessario alle traduzioni, che verrà usato successivamente per reperire le stringhe nella corretta lingua.

```
import { useTranslation } from "react-i18next";
```

Figura 11 -- Import i18n per le traduzioni

Avendo importato il metodo necessario, il passo successivo è dichiarare la variabile che si occupa delle traduzioni, chiamata *t*. Questo viene fatto dentro alla dichiarazione del componente, come primissima cosa.

```
const { t } = useTranslation();
```

Figura 12 -- Dichiarazione t

Da questo momento il componente ha tutte i requisiti per utilizzare le stringhe messe a disposizione dai file di lingua. Nel caso sottostante viene mostrato l'utilizzo generale e quello reale, per la gestione della scritta della pagina di Notizie.

```
//Utilizzo Generale
<p>{t('<page_name>.<translation_id>')}}</p>
//Utilizzo Reale
<li
  className={selectedPage === 'news' ? 'selected' : ''}
  onClick={() => navigateTo('news')}
>
  {t('navigation.news_page')}
</li>
```

Figura 13 -- Utilizzo traduzioni

Cambio di Lingua

Dato che l'applicativo deve gestire il supporto a più lingue, deve anche supportare la possibilità del cambio di esse e questo viene fatto sempre tramite l'utilizzo di `i18n`, cambiando lingua globalmente in modo che cambiandola tutte le pagine cambino istantaneamente la lingua di riferimento e salvandola all'interno nel Database.

Il cambio viene eseguito tramite il componente *LanguageSwitcher*, che si appoggia sul hook *useUser* per cambiare la lingua anche all'interno del database e rendere l'operazione persistente nel tempo.

```
function LanguageSwitcher(email) {
  const { i18n } = useTranslation();

  const { changeLanguage } = useUser(email.email);

  const handleChangeLanguage = (lng) => {
    changeLanguage(lng);
    i18n.changeLanguage(lng);
  };

  return (
    <div className="language-switcher">
      <button
        className={`language-button ${i18n.language === "en" ? "active" : ""}`}
        onClick={() => handleChangeLanguage("en")}
      >
        English
      </button>
    </div>
  );
}

export default LanguageSwitcher;
```

Figura 14 -- LanguageSwitcher

Nella figura vengono mostrata l'implementazione delle lingue, in questo caso con un singolo bottone per l'inglese (che viene ripetuto anche per le altre lingue). Quando il bottone viene cliccato, tramite il metodo *onClick*, viene chiamato il metodo *handleChangeLanguage* che si occupa di cambiare la lingua di tutte le pagine e di

salvare, tramite *useUser*, la nuova lingua nelle preferenze dell'utente all'interno del Database.

Ovviamente al primo accesso viene caricata la lingua dell'utente e impostata per tutto l'applicativo tramite il seguente comando.

```
useEffect(() => {  
  if (user && user.language) {  
    i18n.changeLanguage(user.language);  
  }  
}, [user, i18n]);
```

Figura 15 -- Cambio lingua all'accesso

6.2 Processo di Autenticazione

6.2.1 Richiesta API

Per garantire l'autenticazione, l'applicativo deve inviare una richiesta API per contattare il Backend e controllare se le credenziali dell'utente sono corrette o meno.

```
{
  • "email": • "john_doe@example.com",
  • "password": • "securepassword"
}
```

Figura 16 -- Richiesta API per Autenticazione

Tramite l'utilizzo di Postman, piattaforma per creare e testare le API, viene mostrata la richiesta da inoltrare al Backend per confermare l'autenticazione.

La richiesta viene inviata con allegato un JSON contenente e-mail e password dell'utente tramite richiesta POST, altrimenti le credenziali sarebbero in chiaro con l'utilizzo di una richiesta GET.

```
{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJ1c2VySWQiOiJhZGZlbnkBIiwiaWF0IjoxNzI0MDYyMjUzLCJleHAiOjE3MjQwNjU4NTd9.  
    H_ru9encGyZDoaHBB0qZVDr2h0JMMXJFzwFPCmvx5ao"  
}
```

Figura 17 -- Risposta Richiesta

Come risposta viene dato al richiedente un token che viene associato all'utente, in questo modo l'applicativo può fare qualunque richiesta desiderata allegando il token che ne garantirà l'utilizzo.

Invece se le credenziali non risultassero corrette, la risposta sarebbe un codice 400 *Bad Status* con allegato il messaggio “*Bad Credentials*”.

6.2.2 Lato Frontend

Prima Richiesta

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError("");

  try {
    const response = await axios.post(`${process.env.REACT_APP_API_URL}/auth/login`, {
      email,
      password,
    });
    login(response.data.token, email);
    navigate("/homepage");
  } catch (err) {
    setError("Credenziali non valide");
  }
};
```

Figura 18 -- Richiesta API per il login

La prima operazione eseguita, a bottone *Login* premuto, è chiamare l'API. Questo viene fatto tramite l'utilizzo della libreria *axios*, passando e-mail e password.

In caso risposta positiva viene chiamato il metodo *login* all'interno di *AuthContext*. Come si può notare il processo di autenticazione segue in tutto e per tutto la logica illustrata nel capitolo precedente.

Gestione Token

```
export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [auth, setAuth] = useState({
    token: getAuthToken() || null,
    email: getUserEmail() || null,
  });

  const login = (token, email) => {
    loginService(token, email);
    setAuth({ token, email });
  };

  const logout = () => {
    logoutService();
    setAuth({ token: null, email: null });
  };

  return (
    <AuthContext.Provider value={{ auth, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Figura 19 -- AuthContext

In figura viene mostrata le classi *AuthContext* e *AuthProvider*. In questo componente vengono messi a disposizioni due metodi ed una variabile.

I metodi sono *login* e *logout* che, come si può intuire dal nome, si occupano di gestire tutta la procedura di autenticazione e uscita del profilo utente, impostando o eliminando il token di autenticazione.

Essendo un *context*, tutti i componenti che vengono caricati all'interno dell'applicativo sono poi contenuti, come figli, all'interno di questo. In questo modo chiunque può chiamare i metodi *login*, *logout* e, in caso delle chiamate API, richiedere il token da allegare alle richieste. Facendo così è possibile avere tutte le informazioni riguardo l'autenticazione senza il bisogno di dover passare continuamente il token tra i vari componenti.

Utilizzo API con Token

Essendo che il Token è all'interno del contesto dell'applicativo, non ha senso far sì che ogni volta che si vuole fare una richiesta API bisogna passarlo tramite parametro.

Per evitare questo è stata implementata una classe *api* che si occupa di gestire token e URL per l'API.

```
api.interceptors.request.use(  
  (config) => {  
    const token = getAuthToken();  
    if (token) {  
      config.headers.Authorization = `Bearer ${token}`;  
    }  
    return config;  
  },  
  (error) => Promise.reject(error),  
);
```

Figura 20 -- Richiesta API Generale

Il metodo raffigurato rappresenta quello utilizzato da tutti i services per fare richieste API. Quando viene chiamato esso si occupa di ottenere il token ed aggiungerlo alla richiesta, in questo modo l'implementazione dei services per le richieste API è molto più semplice ed intuitivo (siccome non va aggiunta la gestione del token).

```
export const fetchAllUsers = () => api.get("/users");
```

Figura 21 -- Richiesta API da Service

Come si può notare quando si vuole fare una richiesta API, al posto di farla direttamente, il service (in questo caso *userService*) inoltra la richiesta ad *api* che si occupa di aggiungere il Token e gestire la risposta.

6.2.3 Lato Backend

La gestione dell'autenticazione lato backend viene fatta tramite un middleware di appoggio, ovvero *authMiddleware*. Il compito principale di questo middleware è di processare la richiesta eseguita dal frontend, decodificarla con il token e infine far processare la vera richiesta al controller che se ne occupa.

```
const authenticate = (req, res, next) => {
  const token = req.headers.authorization && req.headers.authorization.split(" ")[1];
  if (!token) {
    return res.status(401).json({ message: "Access denied. No token provided." });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(400).json({ message: "Invalid token." });
  }
};
```

Figura 22 -- Controllo Token

Come si può notare, se il token non è riconosciuto o proprio non viene passato, il middleware si occupa di ritornare un errore, non permettendo di effettuare la richiesta e mantenendo la privacy dei dati intatta.

6.3 Gestione dei File

Per la descrizione della gestione dei file è stato deciso di prendere in considerazione la logica di *UserApproval*, ovvero la parte dove l'utente deve caricare i documenti per l'approvazione da parte dell'amministratore.

6.3.1 FileContainer

FileContainer è un componente del Frontend dedicato unicamente alla gestione del caricamento dei file. Questo componente può essere impostato su diversi stati, come mostrato nella tabella seguente:

Stato	Descrizione
toLoad	Caso iniziale, in questo momento il FileContainer permette di caricare file.
preview	Quando si trova in preview significa che il file è appena stato caricato, ma non ancora inviato. Dunque rimane la possibilità di scaricarlo, oppure di eliminarlo.
waiting	Il file è stato caricato ed inviato, dunque rimane la possibilità unicamente di scaricarlo.
accepted	Stato finale, il documento è stato accettato e dunque rimane solo la possibilità di scaricarlo (da waiting cambia il colore del contorno, che passa da arancione a verde).

Tabella 6: Stati FileContainer

Questo componente, utilizzato in più situazioni, richiede i seguenti campi per funzionare correttamente

Stato	Descrizione
File	Il file che si vuole caricare, in caso non ci sia (e che dunque non è mai stato caricato) si può lasciare <i>null</i> .
fileName	Nome del file, questo sarà quello con il quale verrà poi salvato.
initialState	Stato iniziale del componente, illustrati nella tabella precedente.
fileType	Tipo di file, nel caso di pdf è: 'application/pdf'.
onUpload	Evento di caricamento di un file.
onDelete	Evento di eliminazione di un file.
onDownload	Evento di richiesta di download di un file.

Tabella 7: Campi FileContainer

6.3.2 Salvataggio File

Il file viene inoltrato al backend, che gestisce i file tramite un *middleware* apposito, che a sua volta si appoggia sulla libreria *multer*, specificamente progettata per la gestione dei file in arrivo tramite richieste http.

Verifica della directory di Upload

```
const uploadDir = path.join(__dirname, "../uploads");
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir, { recursive: true });
}
```

Figura 23 -- Salvataggio file – *uploadDir*

All'inizio, il middleware si assicura che la directory dove verranno salvati i file, chiamata *uploads*, esista. Se questa directory non esiste, viene creata automaticamente utilizzando *fs.mkdirSync* con l'opzione *recursive* per garantire che tutte le sottodirectory necessarie siano create.

Configurazione di Multer

Multer è configurato per salvare i file direttamente sul disco (all'interno del backend). La destinazione dei file è impostata sulla directory *uploads*, e i file vengono salvati con il loro nome originale, che viene estratto dalla proprietà *originalname* dell'oggetto *file*, sta dunque al frontend la gestione corretta dei nomi dei file per evitare ridondanze. Per motivi di sicurezza è stato deciso di impostare un limite ai file di 10MB.

Funzione *uploadFile*

Questa funzione gestisce il processo effettivo di caricamento. Se c'è un errore durante il caricamento del file, viene restituito un messaggio di errore al client. Altrimenti, il percorso del file caricato viene aggiunto alla richiesta (*req.filePath*) e si procede al prossimo middleware con *next()*.

Salvataggio nel database

All'interno del Database non viene salvato il percorso assoluto del file, ma quello relativo partendo dalla cartella *uploads*. In questo modo non viene specificato esplicitamente la struttura di cartelle del database.

6.3.3 Gestione Nome dei File

Per evitare una sostituzione involontaria dei file, essi vengono salvati con un nome che riesce a soddisfare l'unicità e anche la comprensibilità.

Il formato varia per le news e per i file caricati dall'utente, per quanto riguarda il primo il formato è *<nome_file>-<numero_generato_casualmente>.<ext>*, invece per quanto riguarda i file che carica l'utente per l'approvazione è: *<tipo_documento>-<email_utente>.<ext>*.

Con l'utilizzo di questi formati (gestiti unicamente dal frontend) si riesce ad avere dei file con dei nomi esplicativi e allo stesso momento univoci.

6.4 GeneralList e GeneralItem

Lo scopo di questi due componenti è quello di semplificare la gestione delle pagine dove è presente principalmente una lista di elementi, che devono fare una determinata azione quando cliccati e devono mostrare determinate informazioni.

6.4.1 GeneralItem

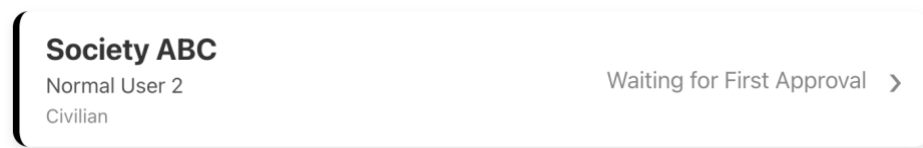


Figura 24 -- GeneralItem

Mostrato nella figura, *GeneralItem* permette di mostrare quattro informazioni di un oggetto a propria scelta, questo deve essere passato con i seguenti campi (tutti facoltativi):

Campo	Descrizione
id	Identificativo dell'oggetto, quando verrà cliccato questo sarà l'unica informazione che verrà ritornato tramite l'apposito metodo.
title	Titolo del Item, nel caso d'esempio <i>SocietyABC</i> .
subtitle	Sottotitolo, nel caso d'esempio <i>Normal User 2</i> .
description	Descrizione, nel caso d'esempio <i>Civilian</i> .
more	Ultima informazione da mostrare a destra.
isRed	Flag, che se impostata a true permette di colorare di rosso la parte sinistra dell'elemento (nell'immagine è nera).
onClick	Metodo chiamato quando un elemento viene cliccato.

Tabella 8: Input GeneralItem

Grazie a questi campi *GeneralItem* permette una gestione chiara e semplice di ogni elemento, questo componente viene però chiamato unicamente da *GeneralList*, togliendo la possibilità di essere chiamato al di fuori di esso.

6.4.2 GeneralList

Elemento con il quale si crea una lista generale, come input accetta unicamente una lista di elementi (che deve avere come campi quelli illustrati nella tabella di *GeneralItem*) e il metodo che verrà chiamato quando un elemento è stato cliccato, *onElementClicked*.

6.4.3 Utilizzo

Nel caso di *InfrastructurePage*, viene creata una lista chiamata *elements* all'interno del hook, che viene poi esportata ed utilizzata direttamente dal componente che chiama *GeneralList* e gliela passa, permettendo la visualizzazione di tutte le infrastrutture. Questo componente è anche utilizzato in altre pagine, come *SettingsPage* e *ReservationPage*.

Questo permette il seguente risultato (il caso preso in considerazione è *InfrastructurePage*).



Figura 25 -- GeneralList

6.5 Gestione Stati dell'utente

Come descritto nella parte di progettazione gli stati dell'utente rappresentano una parte molto importante per quanto riguarda i civili, questo perché in base ad essi si deve sapere quando gli utenti hanno i permessi per fare riserve o meno.

6.5.1 Approvazione Utenti

Il primo momento nel quale lo stato di un utente può essere modificato è quando esso si trova nel processo di approvazione. Questo procedimento segue esattamente quello descritto nella sezione di progettazione

6.5.2 Modifica Ruolo

La modifica dei ruoli si divide in tre categorie principali:

Da civile a qualsiasi altro ruolo

Siccome quando ci si registra ogni utente risulta essere un civile, facendo questa operazione l'amministratore generale decide di promuovere l'utente a "interno" dell'esercito, e dunque di dargli l'approvazione diretta. Questo porta lo stato dell'utente direttamente a 4, ovvero un utente che ha caricato tutti i documenti ed è stato approvato ad effettuare le riserve

Da qualsiasi altro ruolo a civile

Questa operazione è stata considerata come risultato di un errore da parte dell'amministratore, per evitare dunque qualsiasi problema è stato deciso di togliere tutti i permessi all'utente, facendolo tornare allo stato 0, come se non avesse mai caricato dei documenti.

Documenti scaduti

In caso che i documenti siano scaduti l'utente (solo il civile) viene automaticamente portato allo stato 2, ovvero un utente che è stato approvato ma al quale è scaduta l'autorizzazione.

Questo viene controllato appena l'utente effettua l'accesso, evitando dunque che l'utente possa rimanere attivo quando i documenti sono scaduti.

Capitolo 7

Risultati

In questo capitolo vengono mostrate le pagine dell'applicativo, con una spiegazione sul come sono strutturate e come funzionano.

7.1 Pagina di Login

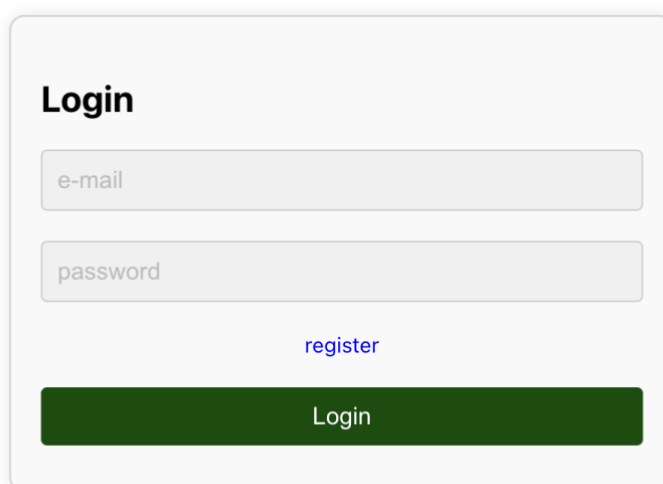
The image shows a login form within a light gray rounded rectangle. At the top left of the form is the title "Login" in bold black text. Below the title are two input fields: the first is labeled "e-mail" and the second is labeled "password", both in a light gray font. Below the password field is a blue text link labeled "register". At the bottom of the form is a dark green rectangular button with the word "Login" in white text.

Figura 26 -- Risultato *Pagina di Login*

La pagina di login, molto semplice ed essenziale, permette all'utente di effettuare l'accesso tramite le proprie credenziali. In caso che l'utente non abbia un account attivo e voglia registrarsi è possibile premere sul bottone *register* per registrarsi.

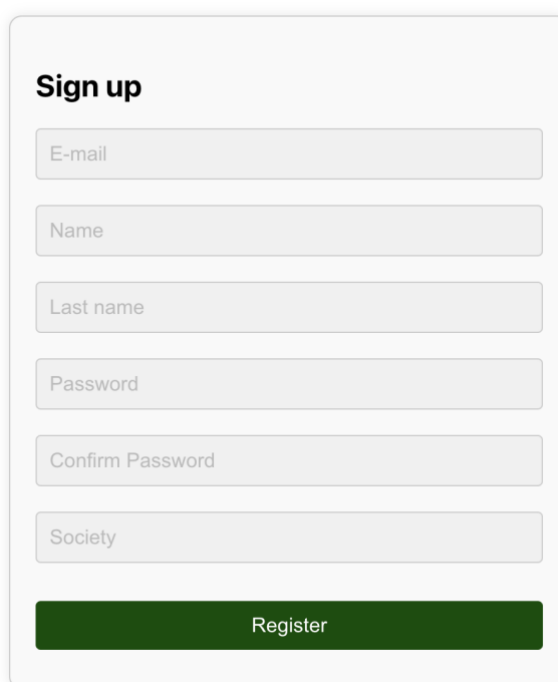
A sign-up form titled "Sign up" with a light gray background. It contains six input fields: "E-mail", "Name", "Last name", "Password", "Confirm Password", and "Society". Each field is a light gray rectangle with its label inside. Below the fields is a dark green "Register" button with white text.

Figura 27 -- Risultato *Pagina Registrazione*

La pagina di registrazione permette a chiunque di creare un utente con tutte le informazioni necessarie, tra queste si trovano *e-mail*, *nome*, *cognome*, *password* e *società*.

In caso che l'utente che si sta registrando faccia parte dell'esercito verrà richiesto all'utente di inserire al posto di *Society* il nome/identificativo della truppa.

Da notare che queste pagine, siccome non ci si può basare sulle informazioni dell'utente, è stata fatta unicamente in inglese, non necessitando perciò delle traduzioni all'interno dell'applicativo.

7.2 Pagina di News

7.2.1 Vista Amministratore

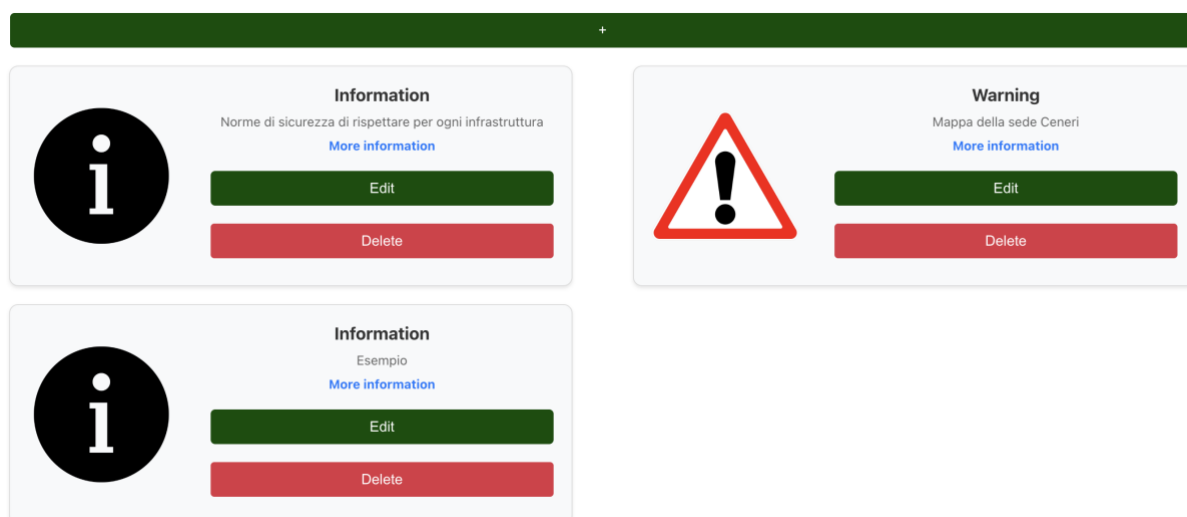


Figura 28 -- Risultato *Pagina News*

La pagina di News per l'amministratore include, come si può notare, molte funzionalità. Le news sono mostrate in ordine cronologico di creazione in base al tipo del quale fanno parte ed ogni tipo mostra un'icona diversa. In questo modo si ha una vista chiara delle notizie che al momento sono disponibili.

The screenshot shows the 'Create News' form. At the top, there is a dark green header bar with a white plus sign. Below the header, there is a form with a light gray background. The form has a title 'Create News' and a red 'X' icon. The form contains the following fields:

- Brief summary:** A text input field.
- Description:** A rich text editor with a toolbar containing icons for H1, H2, Sans Serif, Normal, Bold, Italic, Underline, Strikethrough, Quote, List, Link, and Image.
- Type:** A dropdown menu with the text 'Choose the type'.
- File:** A file upload field with a button 'Scegli file' and the text 'Nessun file selezionato'.

 At the bottom of the form, there is a dark green button labeled 'Create News'.

Figura 29 -- Risultato *Creazione News*

La prima funzionalità a disposizione dell'amministratore è la creazione di News, questo viene fatto premendo sul "+" nella pagina dedicata alle notizie.

La creazione viene effettuata tramite l'utilizzo di un Modal, che richiede all'amministratore le informazioni essenziali sulle notizie.

Brief Summary

Un breve riassunto del contenuto della notizia, questo è quello che viene mostrato nella pagina di notizie sotto alla tipologia di notizia.

Description

Descrizione vera e propria della notizia, questa deve contenere tutte le informazioni ritenute essenziali per l'utente finale. Notare che questa sezione supporta la formattazione, in questo caso Markdown, per far sì di utilizzare titoli, elenchi puntati, etc. Grazie a questa possibilità le notizie sono molto più semplici da scrivere, grazie all'editor, e più veloci da leggere per gli utenti finali.

Type

Il tipo di notizia ne decide l'icona con la quale verrà mostrata all'utente e il titolo di essa.

File

Facoltativo, rappresenta un file che si vuole includere all'interno della notizia. Nel caso che si tratti di un'immagine verrà poi mostrata anche un'anteprima di quest'ultima.

Questo Modal viene aperto anche in caso che si voglia modificare una news già esistente, la differenza è che in questo caso viene aperto con i campi già compilati in base alla notizia che si vuole cambiare. Questa operazione può essere fatta utilizzando il tasto *Edit*.

Infine è possibile, per l'amministratore, eliminare una news facendo sì che essa non sia più visualizzabile per nessun utente.

7.2.2 Vista Normale

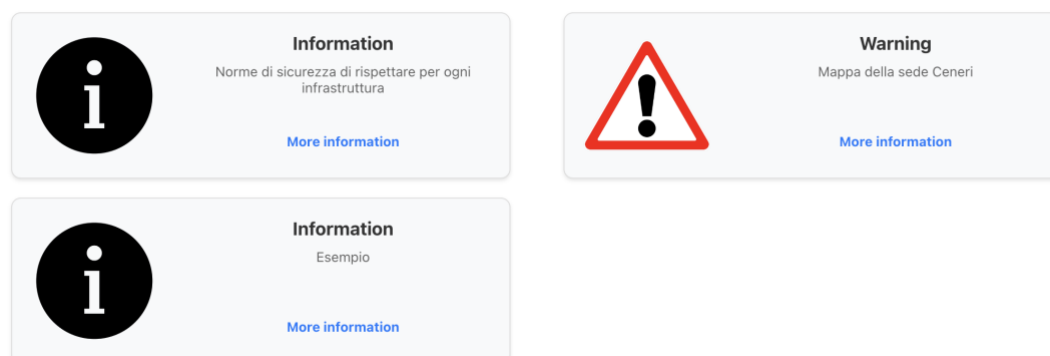


Figura 30 -- Risultato *Vista Utente News*

Nel caso di un utilizzatore, come ad esempio un civile, è possibile unicamente visualizzare il riassunto della notizia e, in caso che *More information* venga cliccato, vengono mostrate tutte le informazioni, tra le quali se presente la possibilità di scaricare il file o di mostrare l'immagine.



Figura 31 -- Risultato *contenuto News*

7.3 Barra di Navigazione

7.3.1 Utente mai approvato

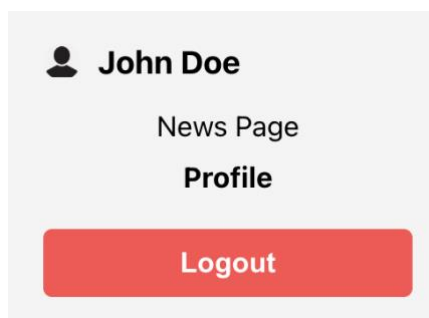


Figura 32 -- Risultato *NavBar* utente non approvato

Nel caso di un utente non registrato vengono mostrate il minimo delle informazioni, ovvero la pagina con le news e le informazioni del profilo. Questa sezione della *NavBar* varia in base allo stato ed al ruolo dell'utente. Ovviamente in ogni tipo di utente rimane sempre la possibilità di fare il logout e il proprio nome e cognome.

7.3.2 Utente approvato

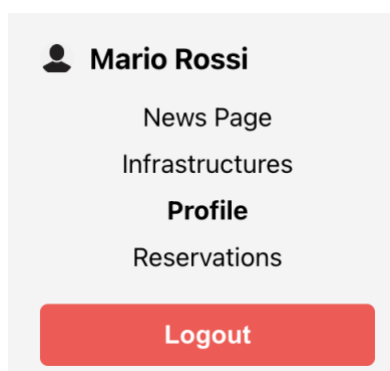


Figura 33 -- Risultato *NavBar* utente approvato

Quando l'utente viene approvato alla *NavBar* si aggiungono due voci, *infrastructures* e *reservations*. Da notare che in caso che l'approvazione dell'utente scada la sua *NavBar* non si modifica, lasciando la possibilità di visualizzare infrastrutture e riserve.

7.3.3 Amministratore

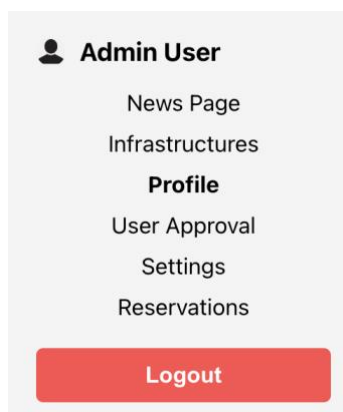


Figura 34 -- Risultato *NavBar Amministratore*

Unicamente nel caso dell'amministratore la NavBar diventa completa, aggiungendo le pagine relative all'approvazione degli utenti e delle impostazioni.

7.4 Calendario



Figura 35 -- Risultato *Calendario*

Unicamente per i non amministratori, è presente una vista calendario con tutte le riserve future. Questo mostra la data nella quale è stata fatta la prenotazione, la infrastruttura riservata e la sede nella quale si svolgerà l'evento.

Questo componente si trova specularmente alla *NavBar* unicamente nelle pagine di News e delle infrastrutture.

7.5 Infrastrutture

7.5.1 Pagina Base



Figura 36 -- Risultato *Infrastrutture*

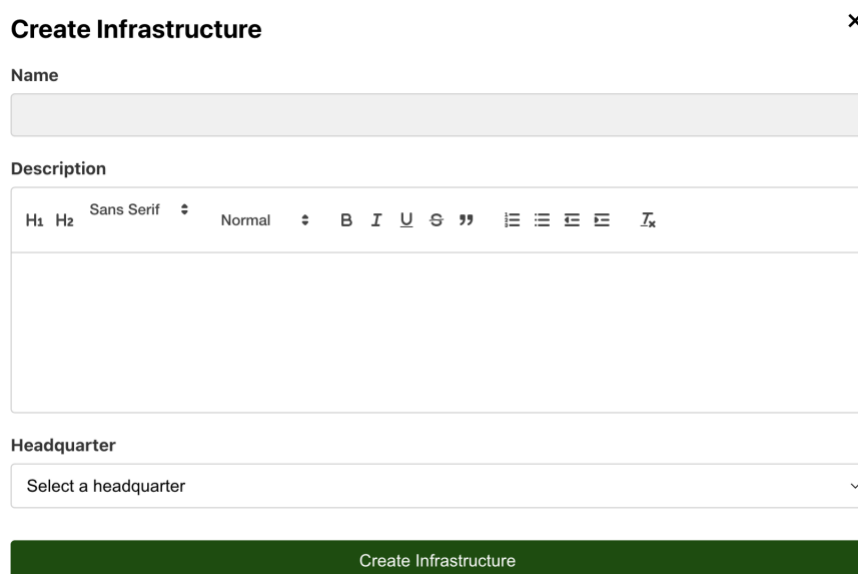
La pagina illustrata mostra tutte le infrastrutture nella vista dell'amministratore. L'unica differenza con gli utenti normali è il checkbox *Edit* e il bottone *Create Infrastructure* che per gli utenti normali non esiste.

In alto a sinistra è possibile impostare un filtro per sede, in modo da mostrare unicamente le infrastrutture relative ad una sede in particolare.

Il comportamento del click sulle infrastrutture varia (unicamente per l'amministratore) in base se *Edit* è spuntato o meno. In caso che è spuntato si apre il modal dedicato alla modifica dell'infrastruttura, altrimenti (come per qualunque altro utente) la possibilità di riservarne una.

Infine è possibile creare un'infrastruttura cliccando sul bottone apposito, che apre il modal dedicato a questa operazione.

7.5.2 Creazione Pagine



Create Infrastructure ×

Name

Description

H₁ H₂ Sans Serif Normal B I U S ” ☰ ☷ ☹ ☺ ↵

Headquarter

Select a headquarter

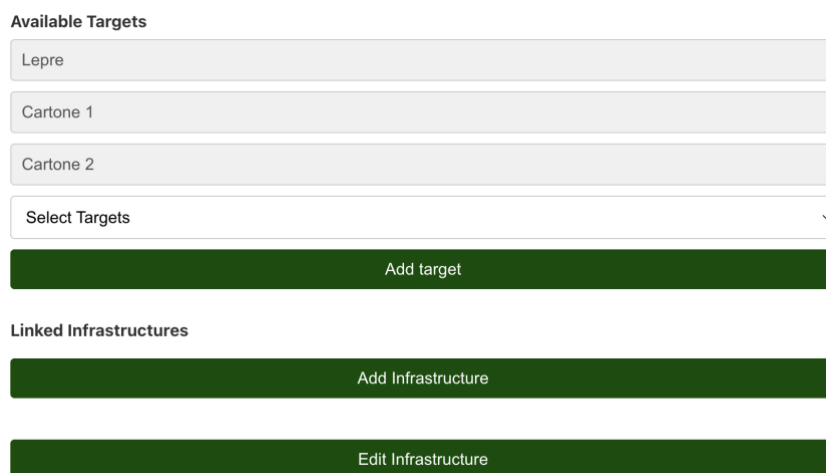
Create Infrastructure

Figura 37 -- Risultato *Creazione Infrastrutture*

Per la creazione delle infrastrutture vengono richieste unicamente le informazioni necessarie per questa operazione, lasciando le informazioni aggiuntive disponibili unicamente nella modifica delle infrastrutture.

Anche in questo caso la descrizione dell'infrastruttura può essere formattata utilizzando Markdown, permettendo all'utente finale di avere un'idea più chiara sull'infrastruttura che sta andando a riservare.

7.5.3 Modifica Infrastrutture



Available Targets

Lepre

Cartone 1

Cartone 2

Select Targets

Add target

Linked Infrastructures

Add Infrastructure

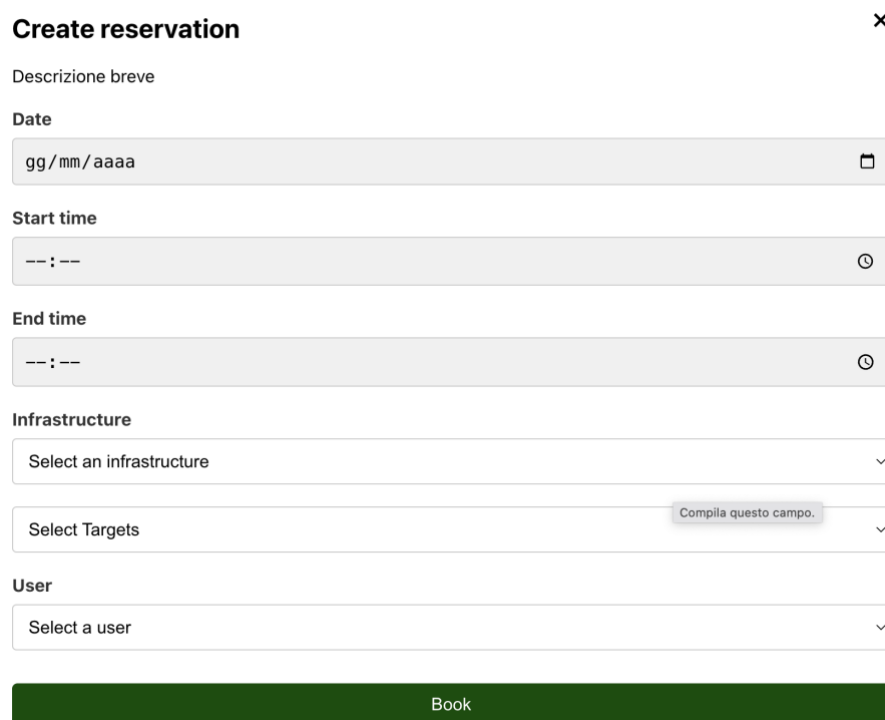
Edit Infrastructure

Figura 38 -- Risultato *Modifica Infrastruttura*

La modifica dell'infrastruttura permette di aggiungere nuove informazioni, tra di esse i bersagli disponibili e delle infrastrutture. Queste vengono create perché sono quelle che verranno successivamente riservate, rendendo l'infrastruttura generale solo per la descrizione.

7.5.4 Creazione Riservazione

La creazione della riservazione si divide in tre schermate diverse in base al tipo di utente che effettua l'operazione.



The screenshot shows a web form titled "Create reservation" with a close button (X) in the top right corner. The form contains several input fields and a final button:

- Descrizione breve**: A text input field.
- Date**: A date input field with the placeholder "gg/mm/aaaa" and a calendar icon.
- Start time**: A time input field with the placeholder "--:--" and a clock icon.
- End time**: A time input field with the placeholder "--:--" and a clock icon.
- Infrastructure**: A dropdown menu with the placeholder "Select an infrastructure".
- Select Targets**: A dropdown menu with the placeholder "Select Targets" and a tooltip that says "Compila questo campo." (Fill in this field).
- User**: A dropdown menu with the placeholder "Select a user".
- Book**: A large green button at the bottom of the form.

Figura 39 -- Risultato *Creazione riservazione*

Nel caso dell'amministratore, mostrato nell'immagine sovrastante, vengono richieste data, orario di inizio e di fine, infrastruttura specifica, target richiesto e infine l'utente che decide di fare questa riservazione.

Se ad effettuare la riservazione è un componente dell'esercito l'ultimo campo non sarà presente, prendendo l'utente in automatico in base a chi è autenticato nell'applicativo.

Infine, nel caso di tutti gli altri viene richiesto pure il numero di partecipanti, togliendo pure in questo caso la selezione dell'utente.

7.6 Pagina Profilo

7.6.1 Informazioni utente

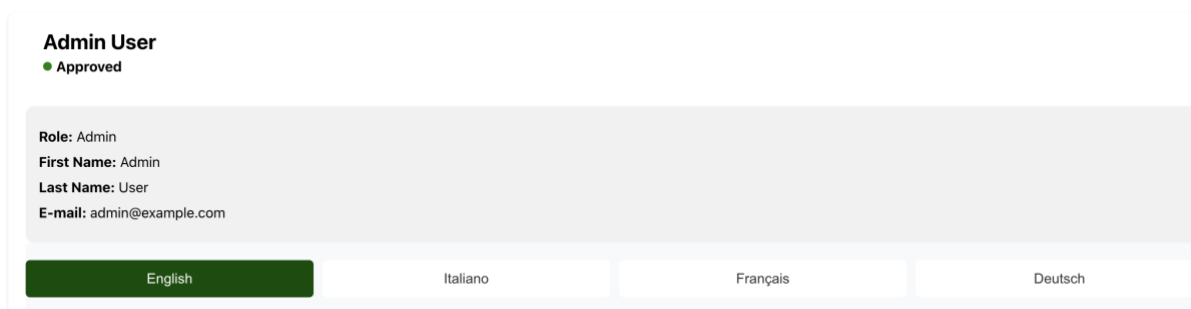


Figura 40 -- Risultato *Pagina Profilo*

Ogni utente ha la possibilità di vedere in ogni momento le proprie informazioni, tra queste ci sono il ruolo, nome, cognome ed e-mail. Inoltre è presente anche uno stato rappresentante a che stato ci si trova con l'approvazione, in questo modo l'utente saprà sempre cosa deve fare per far sì che sia possibile per lui creare una riservazione.

In fondo alle proprie informazioni c'è la possibilità, sempre per ogni utente, di cambiare la lingua del proprio applicativo, rendendo la scelta consistente nel tempo; dunque anche dopo un logout e login tutta l'applicazione rimarrà nella lingua selezionata.

7.6.2 Approvazione Civile

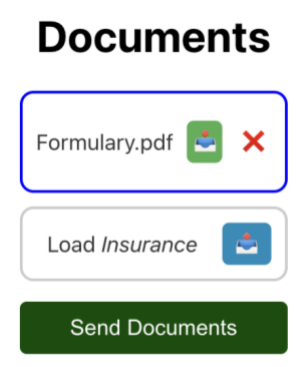


Figura 41 -- Risultato *Documenti non caricati*

Solo per i civili c'è la sezione dedicata all'approvazione dell'utente, in questa sezione (sulla destra della pagina del profilo) è possibile caricare i documenti per l'approvazione, che verranno poi visualizzati dall'amministratore nella pagina dedicata. Infine, ad utente approvato, questa sezione si mostrerà con un documento aggiuntivo, raffigurante la decisione dell'amministratore.

7.7 Approvazione Utenti

The interface is divided into two main sections. The left section displays a list of users with their approval status. The right section provides a detailed view of the selected user, including their role and associated documents.

User	Role	Status
Society ABC Normal User 2 Civilian		Waiting for First Approval
sad John Doe Civilian		Never Approved
Test soc. Mario Rossi Civilian		Approved
Test User Civilian		Never Approved
Truppa 1 Army User Army		Approved

Society ABC

Waiting for First Approval

Normal User 2

normaluser2@example.com

Current Role

Civilian

Documents

- Formulary.pdf
- Insurance.pdf
- Load Decision

Send Documents

Figura 42 -- Risultato *Approvazione Utenti*

L'approvazione degli utenti, presente unicamente per gli amministratori, acconsente agli utenti di proseguire con la loro domanda di approvazione.

Nella lista sono presenti tutti gli utenti all'interno del database, con aggiunte le informazioni sulla società/truppa, il nome e cognome, il ruolo attuale e lo stato attuale della domanda di approvazione

Nella parte destra, che si apre unicamente ad utente selezionato, è possibile rivedere le informazioni sull'utente e di effettuare alcune azioni. La prima, presente per tutti i tipi di utenti che si selezionano è la possibilità di cambiare ruolo all'utente. La seconda, di vedere l'andamento della domanda di autorizzazione, questa è visualizzabile unicamente per i civili, mentre non lo è per i componenti dell'esercito.

7.8 Impostazioni

7.8.1 Pagina principale

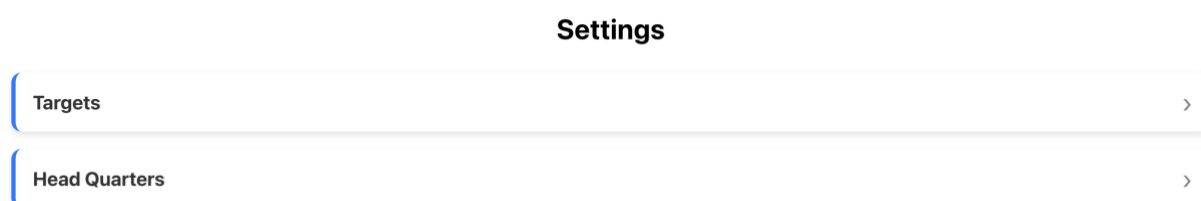


Figura 43 -- Risultato *Impostazioni*

Questa pagina, dedicata unicamente agli amministratori, permette la creazione di Bersagli e di Sedi, che verranno poi visualizzati durante la riservazione di un'infrastruttura.

7.8.2 Creazione Bersagli/Sedi

The screenshot shows a 'Targets' form with two input fields: 'Name' and 'Price'. Below the fields is a green 'Submit' button. To the right of the form is a red 'X' icon. Below the form is a list of three items, each with a left-pointing chevron icon, a name, and a price with a right-pointing chevron icon:

Item Name	Price
CadeBox Con Bersagli	60 chf
CadeBox Senza Bersagli	50 chf
Lepre	100 chf

Figura 44 -- Risultato *Creazione Bersaglio*

Nell'immagine viene mostrata la creazione dei bersagli ma la creazione delle sedi è identica, se non il nome dei campi che varia. In questo Modal che si apre alla selezione di una voce nelle impostazioni, è possibile mettere le informazioni per un nuovo Bersaglio

7.9 Pagina delle Riservazioni

7.9.1 Visualizzazione amministratore

The interface displays a list of reservations on the left and a detailed view of a selected reservation on the right.

List of Reservations:

- Truppa 1**
Army User
Ceneri
15/09/2024 >
- Test soc.**
Mario Rossi
Ceneri
28/12/2024 >

Details for 'Test soc.':

- Contact:** normaluser1@example.com
- Infrastructure:** CadeBox 25 m 2
- Date:** 28/12/2024
- Time Slots:** 12:30, 14:00
- Update** (button)
- Number of Participants:** 11
- Target:** CadeBox Con Bersagli
- Price:** 90 chf
- Cancel Reservation:** Reason (text area)
- Delete** (button)

Figura 45 -- Risultato *Riservazioni amministratore*

In questa vista, della pagina delle prenotazioni di un amministratore, si vedono tutte le prenotazioni future. Nel caso dell'amministratore nella lista vengono visualizzate la società/truppa, chi ha effettuato la prenotazione, il luogo ed infine la data.

Quando si seleziona una prenotazione nel menu di destra sono presenti più informazioni ed è possibile anche cancellare la prenotazione (dando una motivazione scritta) e aggiornare la data e l'orario. Quest'ultima, per motivi di coerenza dei dati, può essere fatta unicamente dall'amministratore.

7.9.2 Visualizzazione normale

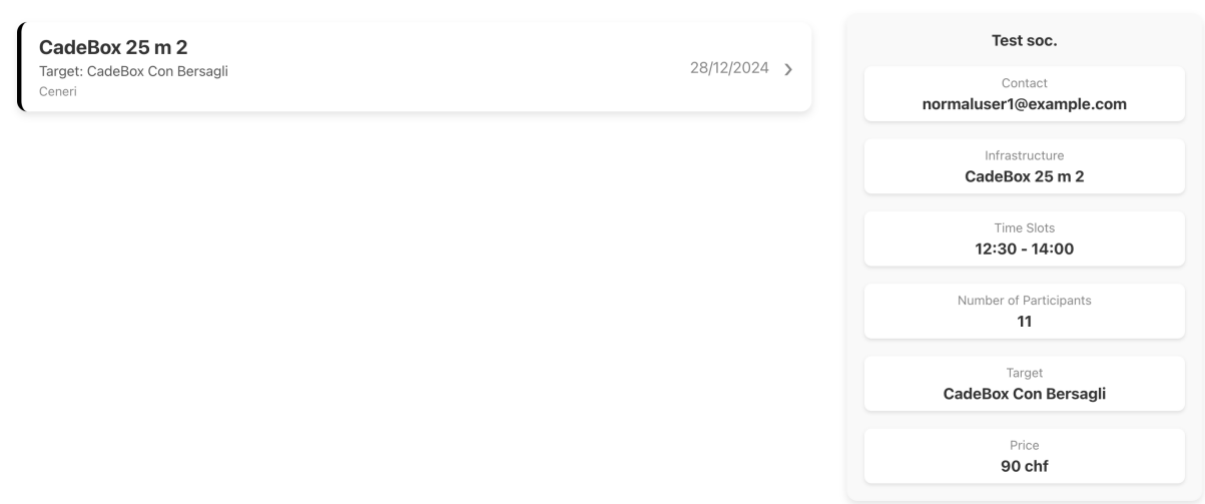


Figura 46 -- Risultato *Riservazioni utente*

Da lato utente non amministratore la lista mostra unicamente le riserve dedicate all'utente, dando solamente la possibilità di visualizzare la prenotazione e non di modificarla e/o eliminarla

Capitolo 8

Test

In questo capitolo vengono visti brevemente i tipi di test che si possono fare in un'applicazione di questo genere. Purtroppo non si ha avuto abbastanza tempo per effettuare molti test.

8.1 Unit Testing

Lo scopo di un unit test è verificare che singole unità di codice, come funzioni o metodi, funzionino correttamente. In questo caso, si sono utilizzate le librerie Mocha, Chai e Sinon. Mocha è il framework che esegue i test, Chai è la libreria che permette di fare le asserzioni (cioè confrontare i risultati ottenuti con quelli attesi), mentre Sinon viene usato per creare mock e stub, cioè simulazioni di funzioni o moduli che permettono di testare il codice in isolamento.

Il test scritto verifica il comportamento di un controller Express, in particolare il metodo `findOne`, che dovrebbe restituire i dettagli di un utente, compreso il suo ruolo, quando viene chiamato con un ID utente valido. Per fare questo, si crea un'istanza simulata dell'utente (`mockUser`) con le proprietà attese, come l'email e il ruolo, e si utilizza Sinon per sostituire la funzione `findOne` di `User` con una versione fittizia che restituisce l'utente simulato.

Il test controlla che:

1. La funzione `findOne` di `User` venga chiamata con i parametri corretti.
2. Il controller restituisca lo status code 200 e i dettagli dell'utente nel formato corretto.
3. Se si verifica un errore durante la ricerca dell'utente, il controller restituisca un errore 500 con un messaggio appropriato.

L'uso di mock e stub permette di testare il comportamento del metodo senza dover dipendere da un database reale, rendendo i test più veloci e affidabili.

```
UserController
  findOne
    ✓ should return user details including role
    ✓ should handle errors

2 passing (4ms)
```

Figura 47 -- Risultato Unit Test

8.2 Test End-To-End

I test End-To-End sono stati fatti utilizzando Postman, un software molto utilizzato per il testing delle chiamate API.

La chiamata che ho voluto testare è una delle più importanti in termini di sicurezza, ovvero il Login. Questo per assicurarsi che il funzionamento del token di autenticazione sia corretto.

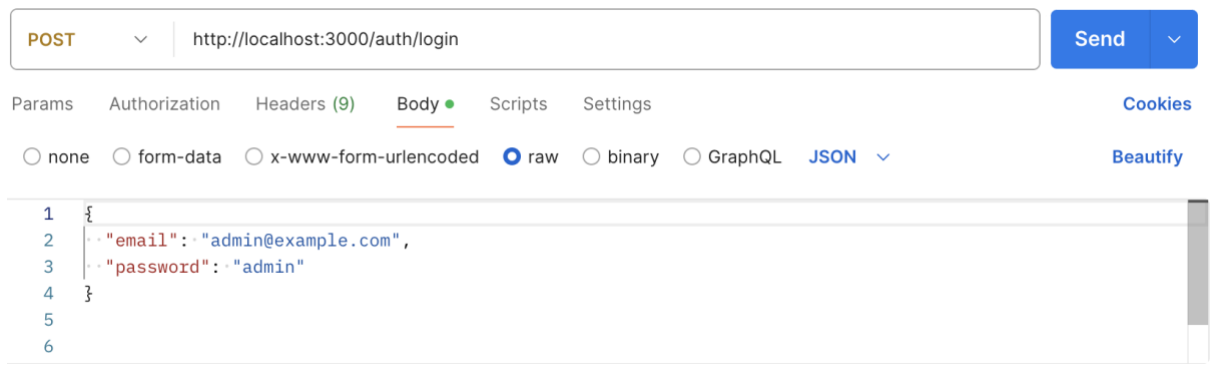


Figura 48 -- Test Richiesta API corretta

Il primo test è stato un semplice login a buon fine, ovvero con le credenziali corrette (usate unicamente nel progetto in locale). Il server in questo caso risponde, correttamente, con codice *200* contenente un JSON con il token, che può essere poi usato per fare richieste all'interno di tutto l'applicativo.

Successivamente è stato provato a fare il test con credenziali sbagliate (ogni possibile combinazione) ed in ogni caso il server ha risposto con un codice *400* Bad Request.

Si può dunque dire che il processo di Login è solido.

Capitolo 9

Conclusioni

<TODO>

9.1 Risultati

9.2 Difficoltà Incontrate

9.3 Sviluppi Futuri

Bibliografia

Indicazioni di riferimenti Internet:

[n] indirizzo Internet, event. pagine specifiche da selezionare