|  |  |
| --- | --- |
| Gestione e Riservazione delle Piazze di Tiro | |
|  | |
| Studente/i | Relatore |
| Matteo Arena | Giuseppe Landolfi |
| Correlatore |
| - |
| Committente |
| Base Logistica dell’Esercito Monteceneri |
| Corso di laurea | Codice progetto |
| Ingegneria informatica TP | C10799 |
| Anno |  |
| 2023/2024 |  |

|  |  |
| --- | --- |
| 5 settembre 2024 |  |

|  |  |
| --- | --- |
|  |  |

Indice

[Introduzione 10](#_Toc174980361)

[Motivazione e Contesto 11](#_Toc174980362)

[Stato dell’arte 12](#_Toc174980363)

[3.1 Analisi delle tecnologie esistenti 12](#_Toc174980364)

[3.1.1 Backend Frameworks 12](#_Toc174980365)

[3.1.2 Frontend Frameworks 13](#_Toc174980366)

[3.1.3 Database Systems 13](#_Toc174980367)

[3.1.4 ORMs (Object-Relational-Mapping) 14](#_Toc174980368)

[3.2 Scelte fatte 14](#_Toc174980369)

[Problema attuale 15](#_Toc174980370)

[4.1 Contesto e Necessità 15](#_Toc174980371)

[4.2 Limitazioni nel Sitema Attuale 17](#_Toc174980372)

[4.2.1 Mancanza di Visibilità in Tempo Reale 17](#_Toc174980373)

[4.2.2 Assenza di Notifiche e Promemoria Automatizzati 17](#_Toc174980374)

[4.2.3 Limitata Scalabilità 17](#_Toc174980375)

[4.3 Impatti delle Limitazioni sul Personale e sulle Operazioni 17](#_Toc174980376)

[Progettazione 18](#_Toc174980377)

[5.1 Architettura del Sistema 18](#_Toc174980378)

[5.1.1 Panoramica dell’architettura 18](#_Toc174980379)

[5.2 Progettazione del Frontend 19](#_Toc174980380)

[5.2.1 Struttura dei componenti 19](#_Toc174980381)

[5.2.2 Interfaccia Utente 20](#_Toc174980382)

[5.2.3 Comunicazione con il Backend 23](#_Toc174980383)

[5.3 Progettazione del Backend 24](#_Toc174980384)

[5.3.1 Struttura del Server 24](#_Toc174980385)

[5.3.2 Progettazione API 25](#_Toc174980386)

[5.4 Progettazione del Database 27](#_Toc174980387)

[Implementazione 29](#_Toc174980388)

[6.1 Gestione delle Lingue 29](#_Toc174980389)

[6.1.1 File JSON 29](#_Toc174980390)

[6.1.2 Configurazione del Sistema di Internazionalizzazione 30](#_Toc174980391)

[6.1.3 Utilizzo Pratico 31](#_Toc174980392)

[6.2 Processo di Autenticazione 34](#_Toc174980393)

[6.2.1 Richiesta API 34](#_Toc174980394)

[6.2.2 Lato Frontend 35](#_Toc174980395)

[6.2.3 Lato Backend 38](#_Toc174980396)

[6.3 Gestione dei File 39](#_Toc174980397)

[6.3.1 FileContainer 39](#_Toc174980398)

[6.3.2 Salvataggio File 40](#_Toc174980399)

[6.3.3 Gestione Nome dei File 40](#_Toc174980400)

[6.4 GeneralList e GeneralItem 41](#_Toc174980401)

[6.4.1 GeneralItem 41](#_Toc174980402)

[6.4.2 GeneralList 41](#_Toc174980403)

[6.4.3 Utilizzo 42](#_Toc174980404)

[6.5 Gestione Stati dell’utente 43](#_Toc174980405)

[6.5.1 Approvazione Utenti 43](#_Toc174980406)

[6.5.2 Modifica Ruolo 43](#_Toc174980407)

**Indice delle figure**

[Figura 1 -- Processo di Riservazione Attuale 15](#_Toc174980408)

[Figura 2 -- Diagramma Architettura 18](#_Toc174980409)

[Figura 3 -- Mockup News 20](#_Toc174980410)

[Figura 4 -- Mockup Login 21](#_Toc174980411)

[Figura 5 -- Mockup Riservazione 21](#_Toc174980412)

[Figura 6 -- Mockup Dashboard Amministratore 22](#_Toc174980413)

[Figura 7 -- Diagramma E/R 27](#_Toc174980414)

[Figura 8 -- Formattazione JSON Lingue 30](#_Toc174980415)

[Figura 9 -- Import i18n 30](#_Toc174980416)

[Figura 10 – Configurazione i18n 31](#_Toc174980417)

[Figura 11 -- Import i18n per le traduzioni 31](#_Toc174980418)

[Figura 12 -- Dichiarazione t 31](#_Toc174980419)

[Figura 13 -- Utilizzo traduzioni 32](#_Toc174980420)

[Figura 14 -- LanguageSwitcher 32](#_Toc174980421)

[Figura 15 -- Cambio lingua all'accesso 33](#_Toc174980422)

[Figura 16 -- Richiesta API per Autenticazione 34](#_Toc174980423)

[Figura 17 -- Risposta Richiesta 34](#_Toc174980424)

[Figura 18 -- Richiesta API per il login 35](#_Toc174980425)

[Figura 19 -- AuthContext 36](#_Toc174980426)

[Figura 20 -- Richiesta API Generale 37](#_Toc174980427)

[Figura 21 -- Richiesta API da Service 37](#_Toc174980428)

[Figura 22 -- Controllo Token 38](#_Toc174980429)

[Figura 23 -- Salvataggio file – *uploadDir* 40](#_Toc174980430)

[Figura 24 -- GeneralItem 41](#_Toc174980431)

[Figura 25 -- GeneralList 42](#_Toc174980432)

**Indice delle tabelle**

[Tabella 1: Chiamate GET 25](#_Toc174980442)

[Tabella 2: Chiamate POST 26](#_Toc174980443)

[Tabella 3: Chiamate GET 26](#_Toc174980444)

[Tabella 4: Spiegazione stati utenti 27](#_Toc174980445)

[Tabella 5: Tabella delle Lingue 29](#_Toc174980446)

[Tabella 6: Stati FileContainer 39](#_Toc174980447)

[Tabella 7: Campi FileContainer 39](#_Toc174980448)

[Tabella 8: Input GeneralItem 41](#_Toc174980449)

**Abstract**

* TODO

**Progetto assegnato**

In questo capitolo vengono descritti i dettagli del progetto come sono stati commissionati.

**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. Protocollo consegna/ riconsegna. Anche la consegna ed il ritiro dell'infrastruttura dovrebbero avvenire con sistema informatizzato, senza utilizzo del cartaceo.
2. 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.
3. 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****

TODO

Capitolo 2

# ****Motivazione e Contesto****

TODO

Capitolo 3

# ****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.

## 3.1 Analisi delle tecnologie esistenti

### 3.1.1 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.

### 3.1.2 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.

### 3.1.3 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.

### 3.1.4 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.

## 3.2 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.

Capitolo 4

# ****Problema attuale****

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

## 4.1 Contesto e Necessità



Figura 1 -- Processo di Riservazione Attuale

La gestione delle riservazioni, illustrata nell'immagine soprastante, evidenzia la complessità e la lunghezza dei passaggi necessari per la riservazione 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 riservazioni 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.

## 4.2 Limitazioni nel Sitema Attuale

### 4.2.1 Mancanza di Visibilità in Tempo Reale

Non avendo la possibilità di sapere quando le infrastrutture sono libere/occupate, gli utenti sono obbligati ad inotrare 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 riservazioni arriva a trovare un giorno libero che vada bene anche per l’utilizzatore.

### 4.2.2 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 riservazione 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.

### 4.2.3 Limitata Scalabilità

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

## 4.3 Impatti delle Limitazioni sul Personale e sulle Operazioni

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

Capitolo 5

# ****Progettazione****

<Breve Descrizione>

## 5.1 Architettura del Sistema

### 5.1.1 Panoramica dell’architettura

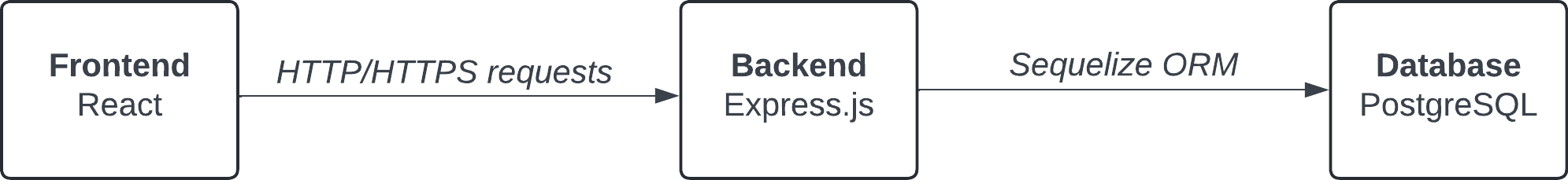


Figura 2 -- Diagramma Architettura

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 effetuata. Inoltre questo livello comunica con il Backend unicamente tramite chiamate API.

**Backend**

Implementando utlizzando 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.

## 5.2 Progettazione del Frontend

### 5.2.1 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.

### 5.2.2 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**

Immagine che contiene testo, schermata, Pagina Web, Sito Web

Descrizione generata automaticamente

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’ifnrastruttura, 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 e 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**

Immagine che contiene testo, schermata, design

Descrizione generata automaticamente

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**

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

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**

Immagine che contiene testo, software, schermata, Pagina Web

Descrizione generata automaticamente

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.

### 5.2.3 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.

## 5.3 Progettazione del Backend

### 5.3.1 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.

### 5.3.2 Progettazione API

Di seguito vengono illustrate le chiamate API che sono supportate dall’applicativo per ogni model. Le chiamate base CRUD non sono riportare, 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 riservazioni. |
| /**bookings**/user/<id> | Dato un utente, ritorna le sue riservazioni. |
|  |  |
| /**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 riservazioni. |
| /**users**/removeApproval/<email> | Toglie l’approvazione ad un utente, togliendo pure la possibilità di fare riservazioni. |
|  |  |
| /**pages**/update/<id> | Permette la modifica di una pagina, passando pure un file. |
|  |  |
| /**bookings**/cancelBooking/<id> | Mette le stato di una riservazione 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.

## 5.4 Progettazione del Database

Immagine che contiene diagramma, origami, linea, modello

Descrizione generata automaticamente

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 riservazioni. |

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****

<Breve descrizione>

## 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.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

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.

*Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente*

Figura 9 -- Import i18n

**Configurazione i18n**

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

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.



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.



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.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

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.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

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.

Immagine che contiene testo, Carattere, bianco, schermata

Descrizione generata automaticamente

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.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

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.

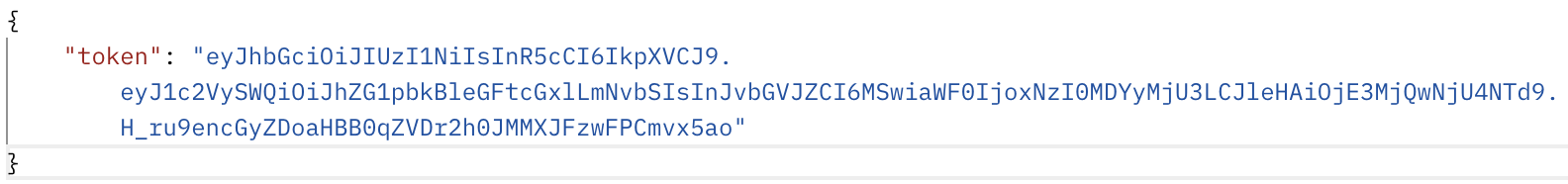


Figura 17 -- Risposta Richiesta

Come risposta viene dato al richiedente un token che viene associato all’utente, in questo modo l’applicativo può fare qualunque richiesta desidera 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**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Figura 18 -- Richiesta API per il login

La prima operazione eseguita, a bottone *Login* premuto, e 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**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

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 si 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.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

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).



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.

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

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, utilizzati 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 *null.* |
| 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. |
| onDowload | 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**

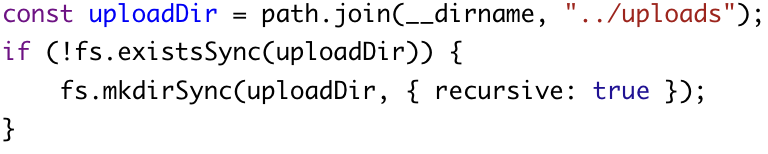
****

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 epslicitamente 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

Immagine che contiene testo, Carattere, bianco, schermata

Descrizione generata automaticamente

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 *SocietyABC.* |
| subtitle | Sottotitolo, nel caso d’esempio *Normal User 2.* |
| description | Descrizione, nel caso d’esempio *Civilian.* |
| 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*).

Immagine che contiene ricevuta, testo, linea, bianco

Descrizione generata automaticamente

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 riservazioni 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 riservazioni

**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.

Bibliografia

Va suddivisa secondo le fonti e con riferimenti numerici tra parentesi quadre [n], che vanno inseriti nel testo subito dopo la parola o frase cui quel riferimento bibliografico si riferisce.

Indicazioni di libri:

[n] autore/autori, titolo del libro, editore, anno, edizione

Indicazioni di articoli:

[n] autore/autori, titolo dell’articolo, titolo della rivista, volume, data, pagine specifiche

Indicazioni di riferimenti Internet:

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

****Appendici****

Sono inserite direttamente nel fascicolo della documentazione.

Vanno identificate singolarmente con A1, A2, A3, ecc. e i relativi titoli, oppure a gruppi con A1, A2, ecc., B1, B2, ecc., C1, C2, ecc., nel caso si vogliano evidenziare dei blocchi di appendici dello stesso tipo, come ad es. schemi, diagrammi, listati, ecc.

Nel caso di appendici su cui sia impossibile o troppo complicato inserire la numerazione delle pagine, come ad esempio listati già formattati, va inserita una prima pagina numerata che riporti il contenuto e il numero di pagine dell’appendice stessa, facendo poi seguire le pagine originali (con numerazione autonoma o senza numerazione).

****Allegati****

Fanno parte della documentazione, ma non del relativo fascicolo, trattandosi di materiale separato, anche se riferito alla documentazione stessa.

Si tratta del CD contenente la documentazione stessa e altro materiale riferito al progetto, ev. fascicoli separati (come ad es. un Manuale d’uso), ev. materiale sperimentale del progetto.

L’identificazione avviene come per le Appendici, usando però lettere iniziali diverse, in modo da non confondere Appendici e Allegati.

Importante: gli Allegati, essendo separati, vanno sempre corredati di scritte o etichette che li identifichino come relativi al progetto e alla documentazione cui si riferiscono (titolo, codice, ecc.).