**Progetto: Rubrica Dipartimentale**

**Versione: 1.0.0 - ottobre 2024**

**Documento: Documentazione tecnica**

**Descrizione Generale**

Il presente documento descrive la tecnologia applicata per la realizzazione dellaRubrica dipartimentale sviluppata con tecnologie attuali.

**Overview della applicazione**

L’applicazione della Rubrica Dipartimentale gestisce l’archivio dei contatti delle varie sezioni del Dipartimento dei Vigili del Fuoco e dei decreti ad esse legati.

**Tecnologia**

Il progetto è scritto in **Angular/C#.**

**Angular** è un framework front-end open-source sviluppato da Google per lo sviluppo di applicazioni web e mobile, pensato per la creazione di applicazioni single-page (SPA).

La versione utilizzata è la **18**.

Angular utilizza un approccio basato su componenti, dove le applicazioni sono costruite combinando componenti riutilizzabili. Questo promuove la modularità del codice e favorisce la manutenibilità e la scalabilità delle applicazioni.

E’ stato implementato uno state manager con pattern Reduxutilizzando la libreriaNgRx

Per le viste sono utilizzate le librerie bootstrap e angularmaterial.

**C#** è un linguaggio di programmazione sviluppato da Microsoft. È un linguaggio orientato agli oggetti e fortemente tipizzato, che fa parte della famiglia dei linguaggi di programmazione C-like, come C, C++ e Java. C# è ampiamente utilizzato per lo sviluppo di applicazioni desktop, web e mobile su piattaforme Windows.

**Modulo**

Il progetto gestisce il rifacimento della rubrica dipartimentale attualmente in uso ai VVF.

Le informazioni trattate nel software attuale verranno gestite con modalità equivalenti al fine di ricreare lo stesso contesto, avvalendosi però di una tecnologia più avanzata e performante.

**Approccio tecnico**

Angular e C#

Di seguito le sezioni e l’approccio che seguiremo:

**Database**:

Il database di tipo relazionale utilizzato per la persistenza dei dati è MySql.

Lo schema riassuntivo del database è stato sviluppato da un’analisi degli oggetti d’ambito e tenedo conto delle tabelle comuni ([schema delDB](#OLE_LINK2)).

**Backend:**

L'ambiente di Sviluppo:

1. Node.js;
2. Visual studio per lo sviluppo con C#.

Progettazione del Backend:

1. Sono state Identificate le API necessarie per supportare le funzionalità dell'applicazione frontend ([elenco delle API](#OLE_LINK3));
2. Il formato dei dati scambiati tra il frontend e il backendè il JSON.

Implementazione delle API:

1. Creazione delle API necessarie per soddisfare le richieste del frontend;

2. Implementazione dell'autenticazione e dell'autorizzazione, solo per accedere alla modifica e inserimento;

3. Gestione delle operazioni CRUD (Create, Read, Update, Delete) sui dati.

Deployment del Backend:

1. Deployare il backend su un server locale;

**Frontend (Angular):**

Ambiente di Sviluppo:

1. Installazione Node.js e Angular CLI
2. Visual studio code

Progettazione del Frontend:

1. L’utente accede senza autenticazione alle pagine.

2. Per l’interfaccio utente dell'applicazione, sono stati identificati i componenti e i layout necessari ([schema di architettura](#OLE_LINK1));

3. La navigazione e l'organizzazione prevedono un routing con menù laterale.

Sviluppo dei Componenti Angular:

1. Angular CLI sono stati utilizzati per generare i componenti, i servizi e i moduli necessari.

2. Collegamento dei componenti alle API del backend per recuperare e visualizzare i dati, in fase di demo è stata utilizzata una struttura mock con dati staticiutilizzando una libreria javascriptjson-server.

Routing:

1. E’ stata configurazione la tabella dirouting nell'applicazione Angular per consentire la navigazione tra le diverse viste.

Gestione dello Stato:

1. E’ stata implementata la libreria NgRx per gestire lo stato dell'applicazione.

Deployment del Frontend:

1. Compilare l'applicazione Angular per la produzione utilizzando il comando ng build –prod;

2. Deploy dei file generati su un server web

Prossimo step: Integrazione Backend e Frontend:

1. Collegamento dell'applicazione frontend al backend utilizzando le API definite in precedenza.

2. Gestione delle richieste HTTP e le risposte dal backend nei servizi Angular.

3. In fase di testing è stato usato un approccio mock

Monitoraggio e Manutenzione:

1. Monitoring delle prestazioni dell'applicazione in produzione utilizzando strumenti come Google Analytics.

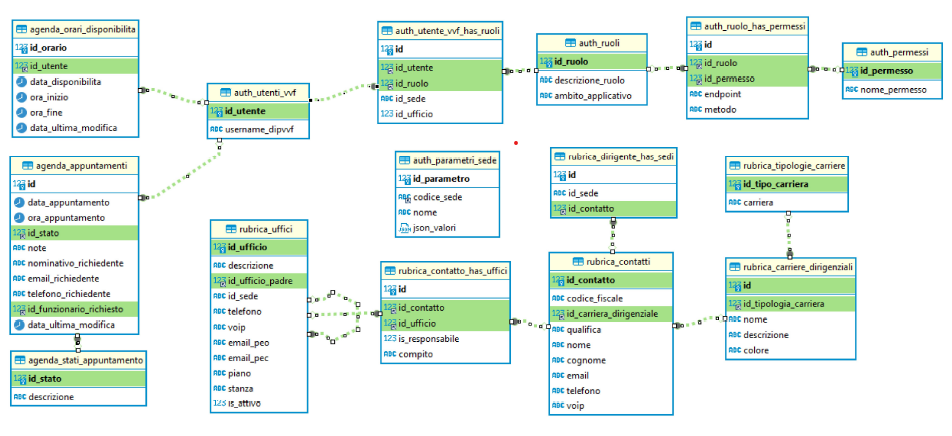
2. Manutenzione dell'applicazione, aggiornando le dipendenze e risolvendo eventuali bug o problemi di sicurezza.

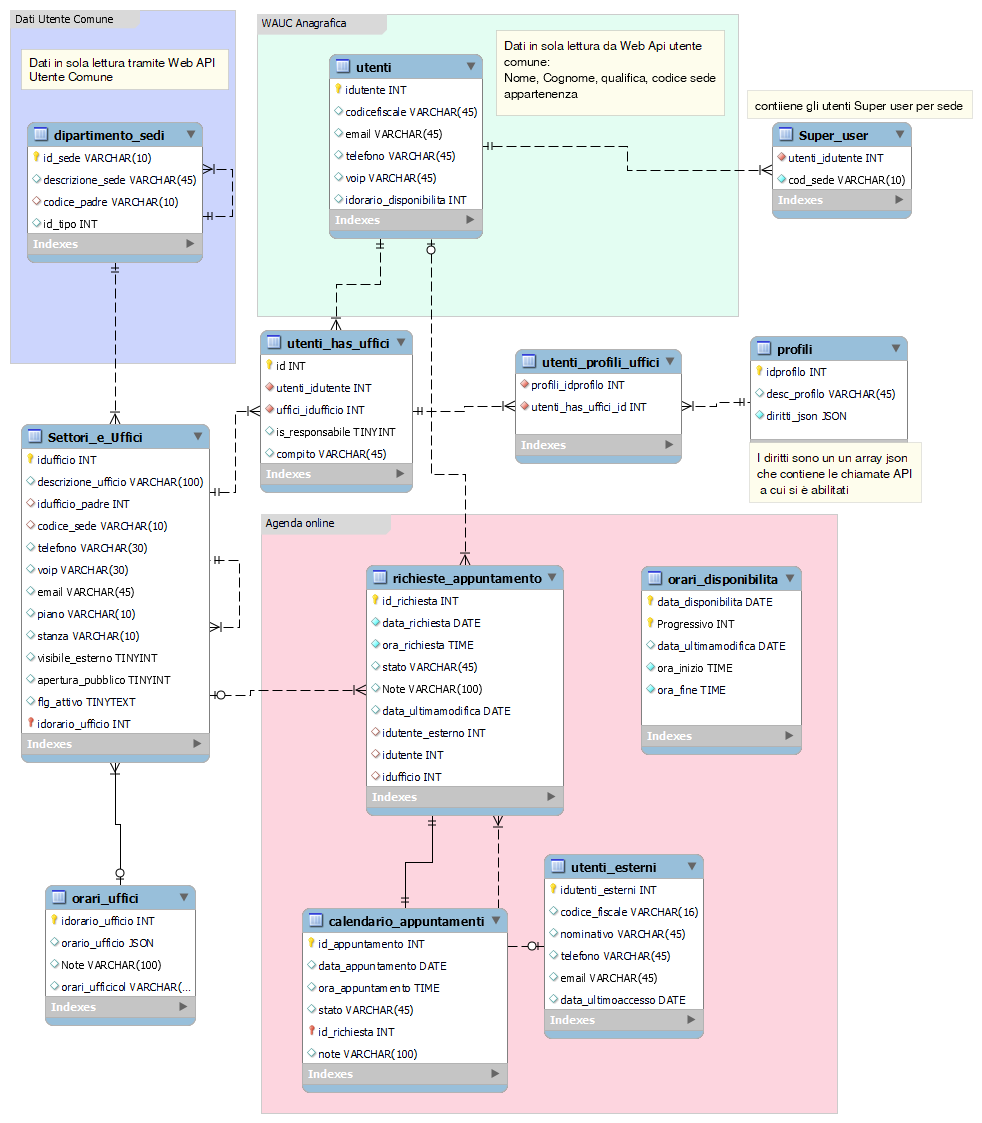
**Documentazione:**

Di seguito verranno analizzate le componenti architetturali dell'applicazione per semplificare la manutenibilità e la collaborazione.

**DATABASE**

**Schema delle tabelle relazionali**

****



**BACKEND**

**API Progetto Rubrica Dipartimentale**

**Premessa**

API REST supporta i seguenti metodi HTTP per manipolare le risorse:

* POST (CREATE) crea una risorsa.
* GET (lettura) restituisce una rappresentazione di una risorsa.
* PUT (UPDATE) aggiorna una risorsa esistente.
* DELETE elimina la risorsa.

**Chiamata API**

Tutte le chiamate inviano i dati richiesti in formato JSON e necessitano di autenticazione che viene effettuata mediante l’utilizzo dell’ http header “Authorization” contente il token ottenuto dall’autenticazione utente

**Risposta API**

La risposta di una API sarà in formato JSON con i seguenti campi

{  
"codiceEsito": "string",  
"descrizioneEsito": "string",  
"errore": boolean  
}

codiceEsito = Codici di stato HTTP

* 200 Successo
* 401 Sistema non autorizzato
* 420 Dati di input mancanti
* 500 Errore interno all'applicazione

descrizioneEsito

* Con errore=false
  + In caso di GET conterrà i dati richiesti
  + In caso di POST, PUT e DELETE conterrà l’esito descrittivo dell’operazione
* Con errore=true
  + conterrà i dettagli dell’errore

**API in Ambito Autenticazione**

• GET /api/GenerazioneToken

{  
"username": "string",  
"password": "string"  
}

**API in Ambito Utenze**

* GET /api/utenze/GetUtente
* POST /api/utenze/CreaUtenteEsterno
* POST /api/utenze/DesignaUtenteGestore
* POST /api/utenze/DesignaUtentePrevenzione
* DELETE /api/utenze/RimuoviRuoloUtente

**API in Ambito Organigramma**

* (AOO Area Organizzativa Omogenea):
* GET /api/aoo/GetOrganigramma
* GET /api/aoo/GetUtentiUfficio
* GET /api/aoo/CercaUfficio
* POST /api/aoo/CreaUfficio
* PUT /api/aoo/ModificaUfficio
* DELETE /api/aoo/CancellaUfficio
* POST /api/aoo/AssegnaUtenteAUfficio
* DELETE /api/aoo/RimuoviUtenteDaUfficio

**API in Ambito Rubrica**

* GET /api/rubrica/CercaContatto
* POST /api/rubrica/AggiungiContatto
* PUT /api/rubrica/AggiornaContatto
* DELETE /api/rubrica/RimuoviContatto

**API in Ambito Agenda**

* GET /api/agenda/ConsultaCalendario
* POST /api/rubrica/RichiediAppuntamento
* POST /api/rubrica/FissaAppuntamento
* PUT /api/rubrica/ModificaAppuntamento
* DELETE /api/rubrica/AnnullaAppuntamento

**FRONTEND**

**Architettura dell’applicazione**

Personale Component

Uffici Component

Sottoufffici Component

Login Component

Home Component

Sedi Component

App Component

Index.html

Si descrive il layout con esempio della Home Uffici Centrali con Uffici dipendenti e con componenti.

