Documentazione Progetto Spring Boot con MongoDB

Descrizione generale

Il progetto è un'applicazione **Spring Boot** che utilizza **MongoDB** come database NoSQL per gestire le entità: **cittadini**, **ospedali**, **patologie**, e **ricoveri**. L'applicazione offre servizi di gestione dei dati (CRUD) e di ricerca con filtri avanzati. La parte di interfaccia utente è realizzata usando **Thymeleaf** per generare pagine HTML dinamiche. I file statici come immagini, CSS e JavaScript si trovano nella directory `static`.

Obiettivi principali:

1. Gestione dei cittadini (anagrafica di persone).
2. Gestione degli ospedali.
3. Gestione delle patologie.
4. Gestione dei ricoveri.
5. Ricerca avanzata con filtri su ogni entità.
6. Funzionalità CRUD (Create, Read, Update, Delete) su tutte le entità.

Struttura del progetto

La struttura del progetto si articola nelle seguenti sezioni principali:

- **Configurazione applicativa**: `application.properties`
- **Service Layer**: `MongoService.java` per la gestione della logica aziendale e interazione con MongoDB.
- **Risorse statiche**: file CSS, immagini e JavaScript.
- **Template HTML**: per la presentazione e interfaccia utente tramite Thymeleaf.
- **Configurazione di build**: `build.gradle` che definisce le dipendenze del progetto.

1. `src/main/resources/application.properties`

Il file `application.properties` contiene tutte le configurazioni principali del progetto, come la porta del server, l'URI di connessione a MongoDB e le configurazioni per la gestione dei dati.

Configurazioni principali:

- **`server.port=8080`:** Definisce la porta su cui l'applicazione è eseguita (8080).
- **`spring.application.name=progWeb2`**: Nome dell'applicazione.
- **`spring.jpa.hibernate.ddl-auto=update`** e **`spring.jpa.show-sql=true`**: Queste sono configurazioni di default per JPA (che non è utilizzato in questo progetto basato su MongoDB).
- **`spring.data.mongodb.uri=mongodb+srv://altre-info/credenziali`**: Questa stringa contiene l'URI semi-completo per connettersi al cluster MongoDB, incluse le credenziali e il database MongoDB da usare.
- **`spring.data.mongodb.database=prjweb`**: Definisce il nome del database MongoDB, in questo caso **`prjweb`**.

2. `src/main/java/com/prj2/service/MongoService.java`

La classe **`MongoService`** è la parte centrale della logica di business dell'applicazione. Questa classe gestisce tutte le interazioni con il database MongoDB attraverso il client **`MongoClient`**. Contiene metodi per eseguire operazioni CRUD su cittadini, ospedali, patologie e ricoveri.

2.1. Connessione a MongoDB

La connessione a MongoDB viene configurata nel costruttore della classe **`MongoService`**. Viene utilizzato **`MongoClientSettings`** per gestire le impostazioni della connessione, come l'URI e la versione dell'API.

```
public MongoService() {
    ServerApi serverApi = ServerApi.builder()
            .version(ServerApiVersion.V1)
            .build();

    MongoClientSettings settings = MongoClientSettings.builder()
            .applyConnectionString(new ConnectionString(CONNECTION_STRING))
            .serverApi(serverApi)
            .build();

    this.mongoClient = MongoClients.create(settings);
}
```

- `MongoClientSettings`: Configura il client MongoDB utilizzando l'URI di connessione (`CONNECTION_STRING`).
- `ServerApiVersion.V1`: Imposta la versione dell'API MongoDB da usare.
- `MongoClients.create(settings)`: Crea l'istanza del client MongoDB con le impostazioni configurate.

2.2. Metodi CRUD per cittadini, ospedali, patologie, e ricoveri

Di seguito sono riportati i metodi chiave definiti nel servizio.

a. **Cittadini**
   i. `getCittadini()`: Ritorna una lista di tutti i cittadini presenti nella collezione `cittadini` del database.

```
public List<Document> getCittadini() {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(CITIZEN_COLLECTION_NAME);
    return collection.find().into(new ArrayList<>());
}
```

   ii. `ricercaCittadini()`: Esegue una ricerca filtrata sui cittadini in base a vari parametri: nome, cognome, città, data di nascita, con possibilità di ordinare i risultati per un campo specificato.

```
public List<Document> ricercaCittadini(String nome, String cognome, String citta,
 LocalDate data_inizio, LocalDate data_fine, String order) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(CITIZEN_COLLECTION_NAME);

    Document query = new Document();
    if (nome != null && !nome.isEmpty()) {
        query.append("NOMEcittadino", new Document("$regex", nome).append("$options", "i"));
    }
    // Filtri simili per cognome, citta, data_inizio, data_fine
    return collection.find(query).sort(Sorts.ascending(order)).into(new ArrayList<>());
}
```

b. **Ospedali**
   i. `getOspedali()`: Ritorna una lista di tutti gli ospedali nella collezione `ospedali`.

```
public List<Document> getOspedali() {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(HOSPITAL_COLLECTION_NAME);
    return collection.find().into(new ArrayList<>());
}
```

ii. `ricercaOspedali()`: Filtra gli ospedali in base a nome, codice o città. Utilizza espressioni regolari (`$regex`) per ricerche parziali e case-insensitive.

```
public List<Document> ricercaOspedali(String nome, String codice, String citta) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(HOSPITAL_COLLECTION_NAME);

    Document query = new Document();
    if (nome != null && !nome.isEmpty()) {
        query.append("NOMEospedale", new Document("$regex", nome).append("$options", "i"));
    }
    // Filtri simili per codice e citta
    return collection.find(query).into(new ArrayList<>());
}
```

c. **Patologie**

i. `getPatologie()`: Ritorna una lista di tutte le patologie nel database.

```
public List<Document> getPatologie() {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(PATHOLOGY_COLLECTION_NAME);
    return collection.find().into(new ArrayList<>());
}
```

ii. `ricercaPatologie()`: Permette di cercare patologie per nome, tipo (cronica/mortale) e gravità minima. È possibile non applicare filtri specifici usando la scelta "qualsiasi".

```
public List<Document> ricercaPatologie(String nome, String opt, Integer gravita) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(PATHOLOGY_COLLECTION_NAME);

    Document query = new Document();
    if (nome != null && !nome.isEmpty()) {
        query.append("nome", new Document("$regex", nome).append("$options", "i"));
    }
    // Filtri per cronica, mortale e gravit
    return collection.find(query).into(new ArrayList<>());
}
```

d. **Ricoveri**

i. `getRicoveri()`: Ritorna una lista di tutti i ricoveri nella collezione `ricovero`.

```
public List<Document> getRicoveri() {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(RECOVERY_COLLECTION_NAME);
    return collection.find().into(new ArrayList<>());
}
```

ii. `ricercaRicoveri()`: Filtra i ricoveri per date di inizio/fine, durata e costo, con la possibilità di ordinare i risultati per un campo specificato.

```
public List<Document> ricercaRicoveri(LocalDate dataInizio, LocalDate dataFine,
 Integer ggInizio, Integer ggFine, Double costoInizio, Double costoFine, String order) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(RECOVERY_COLLECTION_NAME);

    Document query = new Document();
    if (dataInizio != null && dataFine != null) {
        query.append("DATAricovero", new Document("$gte", dataInizio).append("$lte",
         dataFine));
    }
    return collection.find(query).sort(Sorts.ascending(order)).into(new ArrayList<>());
}
```

iii. `deleteRicovero()`: Cancella un ricovero in base all'ID specificato.

```
public void deleteRicovero(String id) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(RECOVERY_COLLECTION_NAME);
    collection.deleteOne(new Document("_id", new ObjectId(id)));
}
```

iv. `aggiungiRicovero()`: Aggiunge un nuovo documento `ricovero` nel database

```
public void aggiungiRicovero(Document ricovero) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(RECOVERY_COLLECTION_NAME);
    collection.insertOne(ricovero);
}
```

v. `updateRicovero()`: Aggiorna un ricovero esistente in base all'ID.

```
public void updateRicovero(String id, Document update) {
    MongoDatabase database = mongoClient.getDatabase(DATABASE_NAME);
    MongoCollection<Document> collection = database.getCollection(RECOVERY_COLLECTION_NAME);
    collection.updateOne(new Document("_id", new ObjectId(id)), new Document("$set", update));
}
```

3. Risorse Statiche (src/main/resources/static/)

Questa directory contiene i file statici che includono:

- CSS: Fogli di stile usati per definire l'aspetto grafico dell'applicazione.
- Immagini: Risorse visive utilizzate nelle pagine.
- JavaScript: File JavaScript per la logica client-side (se necessario).

Questi file vengono serviti direttamente dal server e inclusi nelle pagine HTML.t.

4. Template Thymeleaf (src/main/resources/templates/)

Questa directory contiene i file di template HTML gestiti da **Thymeleaf**. Thymeleaf permette di generare pagine HTML dinamiche server-side, inserendo dati e logica direttamente nei file HTML.

Esempio di un template Thymeleaf:

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Lista Cittadini</title>
    <link rel="stylesheet" href="static/styles.css">
</head>
<body>
    <h1>Elenco Cittadini</h1>
    <table>
        <tr>
            <th>Nome</th>
            <th>Cognome</th>
            <th>Data di nascita</th>
        </tr>
        <tr th:each="cittadino : ${cittadini}">
            <td th:text="${cittadino.nome}">Nome</td>
            <td th:text="${cittadino.cognome}">Cognome</td>
            <td th:text="${cittadino.dataNascita}">Data di Nascita</td>
        </tr>
    </table>
</body>
</html>
```

5. Configurazione di Build (`build.gradle`)

Il file `build.gradle` gestisce le dipendenze del progetto e definisce come il progetto viene compilato ed eseguito.

Contenuto principale:

- **Plugins**:
  - `org.springframework.boot`: Plugin per gestire la configurazione e l'esecuzione di applicazioni Spring Boot.
  - `io.spring.dependency-management`: Permette una gestione centralizzata delle versioni delle dipendenze di Spring.

Versione Java:
```
sourceCompatibility = '22'
targetCompatibility = '22'
```

- **Dipendenze**:
  - **`spring-boot-starter-thymeleaf`**: Aggiunge il supporto per i template Thymeleaf.
  - **`spring-boot-starter-web`**: Fornisce tutto il necessario per creare una web application con Spring MVC.
  - **`spring-boot-starter-data-mongodb`**: Aggiunge il supporto per MongoDB.
  - **`mongodb-driver-sync`**: Driver sincrono di MongoDB per gestire le operazioni di lettura/scrittura su MongoDB.
  - **`spring-boot-devtools`**: Plugin per velocizzare lo sviluppo, con il supporto per il reload automatico delle modifiche.

Ecco una parte dell'esempio del file `build.gradle`:

```
plugins {
    id 'org.springframework.boot' version '3.1.2'
    id 'io.spring.dependency-management' version '1.1.0'
    id 'java'
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'org.mongodb:mongodb-driver-sync:4.10.2'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
}
```

Documentazione del File HTML

Descrizione Generale

Il file HTML rappresenta la struttura di una pagina web per un portale sanitario regionale. Utilizza Thymeleaf come motore di template per generare contenuti dinamici basati su dati forniti dal server. La pagina include elementi di navigazione, un modulo di ricerca e una sezione per visualizzare i risultati.

Struttura del Documento

1. Dichiarazione del Documento

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```

- **Dichiarazione DOCTYPE**: Indica che il documento è un HTML5.
- **Namespace Thymeleaf**: Definisce il namespace `th` per utilizzare le funzionalità di Thymeleaf.

2. Sezione `<head>`

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="/image/icon.png"> <!-- Percorso aggiornato per l'icona -->
    <link rel="stylesheet" href="/css/stylesheet.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
    <script src="/js/gestioneCRUD.js"></script>
    <title>Servizio Sanitario</title>
</head>
```

- **Meta Charset**: Imposta la codifica del carattere su UTF-8.
- **Meta Viewport**: Configura il viewport per una corretta visualizzazione su dispositivi mobili.
- **Link per l'icona**: Specifica l'icona da visualizzare nella scheda del browser.
- **Link al Foglio di Stile CSS**: Collega il foglio di stile per la formattazione della pagina.
- **Script jQuery**: Includo jQuery per la manipolazione del DOM e altre funzionalità.
- **Script personalizzato**: Collega il file `gestioneCRUD.js` per le funzionalità di gestione CRUD.
- **Titolo della Pagina**: Imposta il titolo del documento.

3. Sezione `<aside>`

Menu Laterale: Fornisce collegamenti alle diverse sezioni del portale.

```
<aside class="aside-menu">
        <div class="sidebar">
            <p id="navbar-text">Esplora Portale</p>
            <a href="/" class="sidebar-link">HomePage</a>
            <a href="/ospedali" class="sidebar-link">Ospedali</a>
            <a href="/ricovero" class="sidebar-link">Ricoveri</a>
            <a href="/cittadini" class="sidebar-link">Cittadini</a>
            <a href="/patologia" class="sidebar-link">Patologie</a>
        </div>
</aside>
```

4. Sezione di Ricerca

Modulo di Ricerca: Permette agli utenti di filtrare i cittadini per nome, cognome, città e data di nascita.

```
<div class="search-container">
        <form action="/cittadini" method="get">
            <p class="title-search-container">Filtra</p>
            <input type="text" placeholder="Cerca per Nome"
                name="nome" th:value="${param.nome}">
            <input type="text" placeholder="Cerca per Cognome"
                name="cognome" th:value="${param.cognome}">
            <input type="text" placeholder="Cerca per Citta"
                name="citta" th:value="${param.citta}">
            <hr>
            <label class="label-data" for="data_inizio">Data di nascita:</label>
            <input type="date" name="data_inizio"
                th:value="${param.dataInizio != null ? #dates.format(param.dataInizio, 'yyyy-MM-dd') : ''}">
            <input type="date" name="data_fine"
                th:value="${param.dataFine != null ? #dates.format(param.dataFine, 'yyyy-MM-dd') : ''}">
            <input type="submit" value="Cerca">
        </form>
    </div>
```

5. Sezione di Ordinamento

```
<div class="ordering">
        <form method="get" action="#">
            <div class="sort-bar">
                <label for="sort-select">Ordina per:</label>
                <select id="sort-select" name="order">
                    <option value="nome">Nome</option>
                    <option value="cognome">Cognome</option>
                </select>
            </div>
            <button type="submit" id="ordering-submit">Ordina</button>
        </form>
    </div>
```

6. Sezione dei Risultati

```
<div class="displayer-results">
        <div class="titles-results-cittadini">
            <div class="titles-elements-cittadini">CSSN</div>
            <div class="titles-elements-cittadini">Nome</div>
            <div class="titles-elements-cittadini">Cognome</div>
            <div class="titles-elements-cittadini">Indirizzo</div>
            <div class="titles-elements-cittadini">Luogo di Nascita</div>
            <div class="titles-elements-cittadini">Data di Nascita</div>
        </div>
        <div th:if="${#lists.isEmpty(cittadini)}" class="no-results">
            Nessun cittadino trovato con i criteri di ricerca.
        </div>
        <div th:each="cittadino : ${cittadini}" class="records-cittadini">
            <div class="result-element" th:text="${cittadino.CSSNcittadino}"></div>
            <div class="result-element" th:text="${cittadino.NOMEcittadino}"></div>
            <div class="result-element" th:text="${cittadino.COGNOMEcittadino}"></div>
            <div class="result-element" th:text="${cittadino.INDIRIZZOcittadino}"></div>
            <div class="result-element" th:text="${cittadino.LUOGONASCITAcittadino}"></div>
            <div class="result-element" th:text="${cittadino.DATANASCITAcittadino}"></div>
        </div>
    </div>
```

9. Script per Evidenziare la Navigazione Attiva

```
document.addEventListener('load', activeTag());
function activeTag(){
    var url = window.location.href;
    var navTags = document.getElementsByClassName("sidebar-link");
    const path = new URL(url).pathname;
    Array.from(navTags).forEach(a => {
        var aHref = a.getAttribute("href");
        if (path === aHref) {
            a.classList.add("active");
        }
    });
}
```

Questo file HTML è una parte fondamentale del portale sanitario, poiché gestisce l'interazione dell'utente, permette la ricerca e visualizza i dati in modo organizzato. L'uso di Thymeleaf per l'integrazione dei dati facilita la creazione di pagine dinamiche in base alle esigenze degli utenti.