Spring Boot, Controller e Architettura Web

Spring Boot sta mostrando una Whitelabel Error Page, che è la pagina di errore generica visualizzata quando non trova un mapping appropriato per gestire una richiesta. L'errore principale è NoResourceFoundException con status quattrocentoquattro, che indica che Spring Boot non riesce a trovare una risorsa statica o un endpoint che corrisponda all'URL richiesto.

Il metodo findFastest presente nel repository è un metodo di business logic che non è e non deve essere mappato direttamente. In Spring Boot, solo i controller hanno metodi mappati agli endpoint HTTP. Il flusso corretto segue la catena Repository verso Service, opzionalmente, verso Controller verso Endpoint HTTP.

Mappare significa creare un collegamento diretto tra un indirizzo web, ovvero un URL, e un metodo Java nel controller. Spring Data JPA segue una naming convention specifica per interpretare i nomi dei metodi. La struttura è find più First più By più OrderBy più Queue più Asc, che si traduce in SELECT con LIMIT uno, nessun filtro, ORDER BY con il campo queue in ordine ASC.

Spring analizza il nome findFirstByOrderByQueueAsc parola per parola seguendo regole precise. Il prefisso può essere find, get, read, query, search o stream, tutti equivalenti per SELECT. I limitatori First, Top, Top tre o First cinque servono per limitare il numero di risultati. I filtri vengono aggiunti dopo By con condizioni come ByName, ByQueueLessThan o ByIdAndName. L'ordinamento utilizza OrderBy seguito dal nome del campo e Asc o Desc.

Spring deduce automaticamente la query SQL corretta per ciascuno di questi metodi basandosi esclusivamente sul nome. Esistono diverse keyword per costruire query complesse. And permette di combinare condizioni multiple, Or consente alternative, Is o Equals verifica l'uguaglianza, Between cerca valori in un intervallo, LessThan e LessThanEqual cercano valori minori, GreaterThan e GreaterThanEqual cercano valori maggiori.

Before e After funzionano con le date, IsNull e NotNull verificano i valori nulli, Like, NotLike, StartingWith, EndingWith e Containing gestiscono pattern testuali, Not

verifica disuguaglianza, In e NotIn verificano l'appartenenza a collezioni, True e False cercano valori booleani, IgnoreCase effettua confronti case-insensitive.

Oltre a First e Top, esistono altri modificatori speciali. Distinct rimuove duplicati con findDistinctByName, Count conta i risultati invece di restituirli con countByAge, Delete o Remove eliminano i record con deleteByAge, Exists verifica l'esistenza di record con existsByEmail.

Andare a un indirizzo significa eseguire un metodo, quindi quello che vediamo nel browser è il ritorno di quel metodo. Le pagine HTML devono essere posizionate nella cartella templates. Le pagine HTML ora caricheranno dati da stampare, consumeranno dati caricandoli dalle API. L'indirizzo è stato modificato perché le API devono essere distinte dalle pagine web.

La creazione di una pagina HTML che funziona come client per l'API rappresenta il caso dell'OperatorHome. L'operatore è Angelo che si trova in strada sul posto dell'incidente e deve decidere dove portare il paziente. L'HTML deve caricare i dati dalla API, ovvero da /emergency/api/hospitals/fastest.

Nel body abbiamo due elementi con id specifici: hospitalname e hospitaladdress. Questi id sono fondamentali perché funzionano come etichette identificative che JavaScript userà per trovare e modificare quegli elementi specifici. Il processo si articola in tre fasi: fetch fa la richiesta HTTP all'API Spring Boot, then converte la risposta JSON in oggetto JavaScript, e infine i dati vengono stampati nella pagina usando DOM manipulation per sostituire Loading con i dati reali.

Ora si deve scrivere un controller per mappare questo file HTML a un indirizzo, dopodiché la pagina si prenderà i dati. Prima esisteva un controller che produceva dati grezzi, ora si scriverà un controller che restituirà una pagina web. Si utilizza Controller invece di RestController.

La classe è annotata con Controller, non RestController. Questo significa che i metodi non restituiscono JSON, ma nomi di viste, ovvero pagine HTML che verranno renderizzate dal motore di template di Spring, di solito Thymeleaf. In pratica, questo controller è dedicato a servire l'interfaccia grafica, la pagina operatorhome.html, non i dati.

Alcuni controller producono dati grezzi, altri producono pagine web. Solo i controller con annotazioni come Controller o RestController sono mappati agli endpoint HTTP. La differenza fondamentale è che RestController restituisce dati

serializzati in JSON mentre Controller restituisce nomi di viste HTML. Un metodo annotato con GetMapping viene eseguito automaticamente quando arriva una richiesta GET a quell'URL specifico.

Il controller OperatorController mappato su /emergency/operator restituisce la stringa operatorhome, che Spring Boot interpreta come nome logico di una vista. Il ViewResolver cerca automaticamente il file operatorhome.html nella cartella src/main/resources/templates e lo serve al browser.

Questa pagina HTML contiene elementi con id specifici come hospitalname e hospitaladdress inizialmente impostati su Loading. Un blocco JavaScript eseguito nel browser utilizza la Fetch API per chiamare l'endpoint REST /emergency/api/hospitals/fastest del controller RestController.

La chiamata fetch funziona in tre fasi. Prima fetch invia la richiesta HTTP asincrona e restituisce una Promise. Il primo then riceve la risposta HTTP grezza e converte il body JSON in un oggetto JavaScript contenente i dati dell'ospedale. Il secondo then riceve questo oggetto e utilizza document.getElementById per trovare l'elemento HTML con quell'id e sostituire il testo Loading con il nome effettivo dell'ospedale. La stessa operazione viene ripetuta per l'indirizzo. Il browser aggiorna immediatamente la visualizzazione sostituendo i placeholder con i dati reali.

Quando l'operatore accede all'URL /emergency/operator, il controller MVC serve la pagina HTML iniziale che contiene JavaScript. Questo codice si esegue automaticamente caricando i dati dall'API REST e aggiornando dinamicamente la pagina. Il risultato visibile è il nome dell'ospedale consigliato, l'indirizzo e il numero di pazienti in coda, calcolati dal backend e visualizzati dal frontend senza ricaricare la pagina.

Quando si carica un sito web si ricevono oppure pagine web complete oppure dati grezzi in formato JSON. La prima richiesta dell'operatore va all'indirizzo /emergency/operator, mappato a un controller normale annotato con Controller che restituisce il nome del template operatorhome.html. Spring Boot legge questo file dalla cartella templates e lo invia al browser come pagina web completa.

Il file HTML contiene JavaScript che fa una seconda richiesta verso /emergency/api/hospitals/fastest, gestita da un controller RestController che fornisce dati JSON puri. JavaScript riceve questi dati e popola dinamicamente la pagina sostituendo i placeholder con le informazioni reali dell'ospedale. L'HTML

rappresenta il frontend visibile all'utente mentre le API costituiscono il backend fornitore di servizi e dati.

All'interno della pagina, il codice JavaScript esegue un'operazione fetch che significa prendere dati dall'indirizzo API. Questa chiamata fetch va verso il controller RestController che espone l'endpoint REST specifico. Il controller REST chiama il metodo findFastest del repository HospitalRepository, riceve l'oggetto Hospital contenente i dati dell'ospedale più veloce e lo serializza automaticamente in formato JSON.

HTML e JavaScript costituiscono il frontend, ovvero la parte dell'applicazione che gira nel browser dell'utente e consuma i servizi. Rappresentano l'interfaccia che l'operatore vede e con cui interagisce, richiedendo dati tramite chiamate fetch. Le API rappresentano il backend, ovvero la parte del sistema che gira sul server e fornisce i servizi. È il fornitore di dati e logica di business, accessibile tramite gli endpoint REST gestiti dai controller annotati con RestController.

Le frecce bidirezionali mostrano il flusso di comunicazione: il frontend invia richieste HTTP verso il backend e riceve in risposta dati JSON strutturati. Questa separazione tra frontend consumatore e backend fornitore è il pattern standard delle applicazioni web contemporanee.

Una pagina web è composta da molteplici file HTML forniti dai controller annotati con Controller, che mappano gli URL alle viste nella cartella templates. La freccia che sale dal blocco HTML verso il Controller mostra la prima richiesta HTTP: quando l'utente digita l'indirizzo, il Controller restituisce il nome del template HTML che Spring Boot serve al browser.

Successivamente il JavaScript contenuto nell'HTML fa una seconda richiesta verso il blocco API. Questa richiesta va al RestController che gestisce gli endpoint REST e fornisce dati strutturati in formato JSON. I due tipi di controller hanno responsabilità distinte ma complementari: il Controller gestisce la presentazione, ovvero l'HTML frontend, mentre il RestController gestisce i dati, ovvero l'API backend. Insieme formano l'architettura completa dell'applicazione web.

L'annotazione Service serve a indicare che una classe appartiene allo strato di servizio dell'applicazione, ovvero contiene la logica di business del dominio specifico. Tecnicamente è una specializzazione dell'annotazione Component, quindi viene rilevata automaticamente dalla scansione del classpath di Spring durante l'avvio dell'applicazione.

Spring crea un'istanza singleton di questa classe e la registra come bean gestito nel container IoC, ovvero Inversion of Control, rendendola disponibile per l'iniezione delle dipendenze tramite Autowired in altri componenti come controller o repository. Nel caso della classe QueueRepository, l'annotazione Service permette ai controller REST di iniettare questa classe e utilizzare il metodo list per ottenere le informazioni sulle code degli ospedali.

L'annotazione Autowired è il meccanismo di Dependency Injection di Spring che permette di iniettare automaticamente le dipendenze necessarie in un controller, servizio o qualsiasi altro componente gestito dal container IoC.