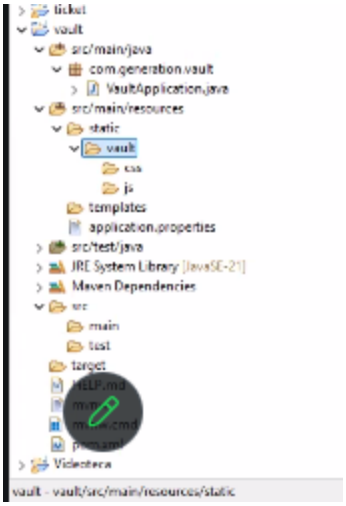
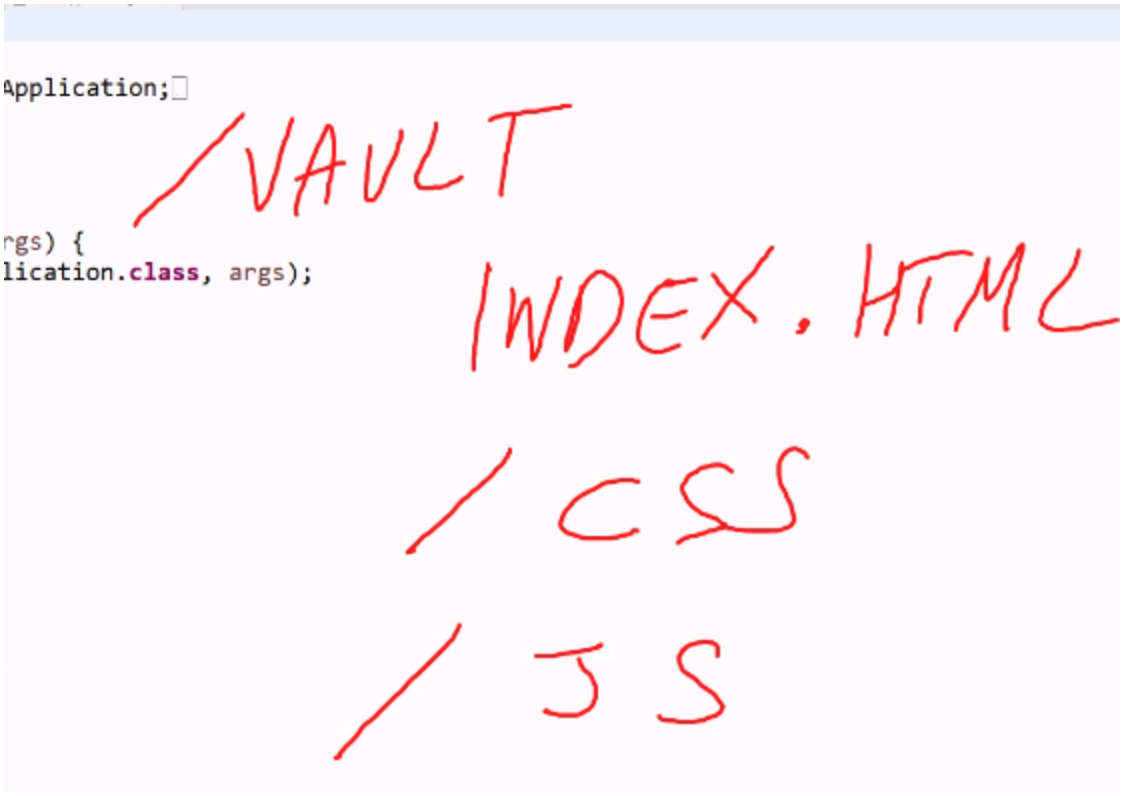
Appunti 05-02-26





VAULT PROJECT

La struttura del progetto VAULT Spring Boot, creato con Maven, prevede all'interno della directory static la creazione di una cartella che prende il nome del sito, in questo caso "vault". All'interno di questa cartella principale si collocano i file HTML, come ad esempio index.html. Inoltre vengono definite altre due cartelle, css e js, destinate rispettivamente a contenere i fogli di stile e la logica dei componenti. Questa suddivisione organizzativa ricorda da vicino la struttura dei progetti Angular.

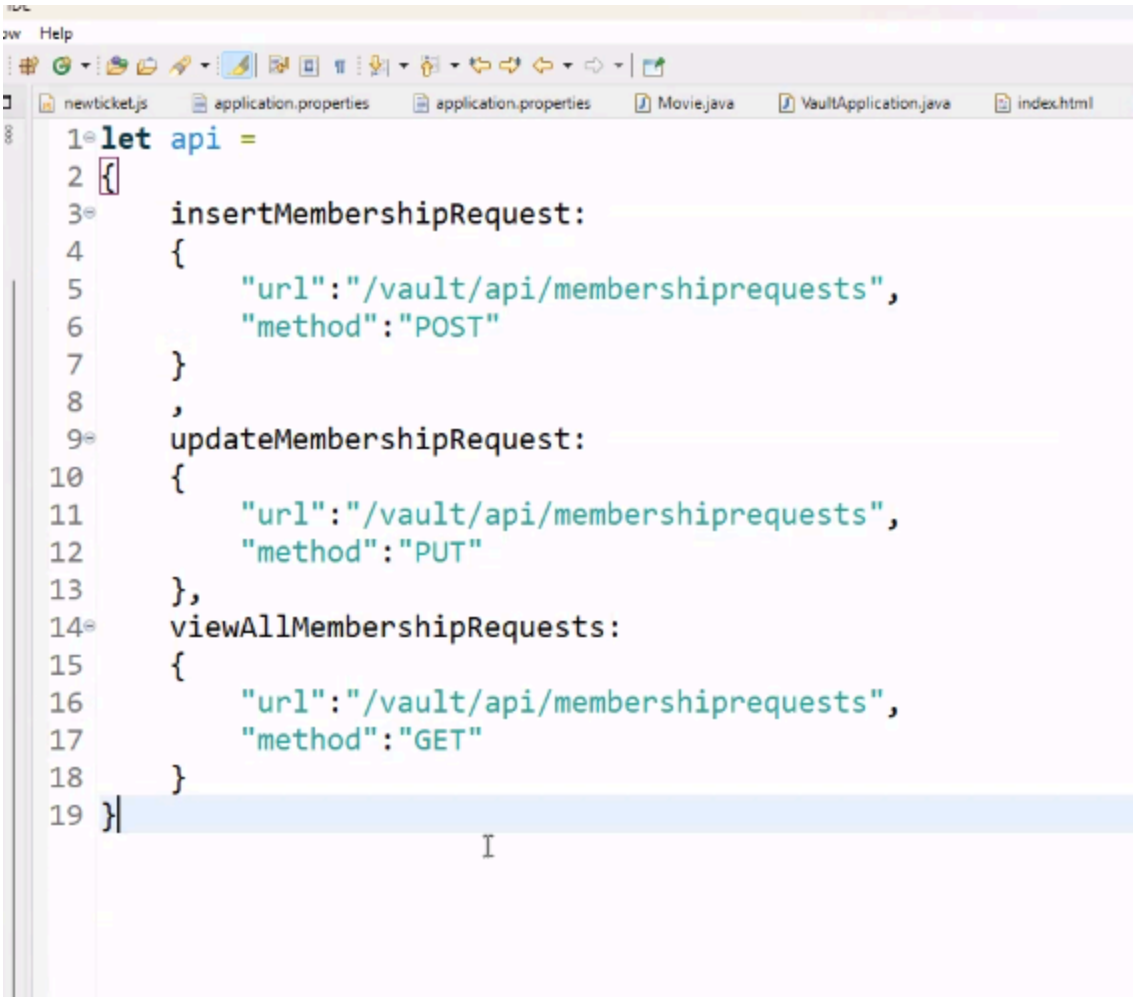
Il primo passo consiste nella creazione, all'interno della directory static, di una cartella che assume il nome del sito, come "vault". In questa cartella principale vengono collocati i file HTML, tra cui index.html.

Il secondo passo riguarda la creazione del file index.html vero e proprio.

È importante sottolineare che quando si apre una pagina web non è possibile aprirla direttamente come file HTML dalla macchina locale, ma occorre sempre accedervi tramite il server. Nel caso specifico, l'accesso avviene attraverso localhost sulla porta 8080, utilizzando l'URL http://localhost:8080/vault/index.html.

Il terzo passo prevede la creazione del file CSS, nel quale vengono definite le regole generali per il sito.

Il quarto passo consiste nel collegare il foglio di stile al file HTML. Poiché ci si trova nella cartella principale e occorre raggiungere la sottocartella css, il percorso da specificare nel collegamento sarà css/styles.css.



Il quinto passo prevede la creazione, all'interno del file JavaScript, di un file denominato api.js nel quale vengono inserite tutte le API necessarie. Per l'operazione insertMembershipRequest non ci si limita a inserire semplicemente il valore, ma si specifica anche l'URL e si definisce il metodo HTTP, che in questo caso sarà POST. Analogamente, per updateMembershipRequest si utilizza l'URL "/vault/api/membershiprequests" con il metodo PUT, mentre per viewAllMembershipRequests si impiega lo stesso URL "/vault/api/membershiprequests" ma con il metodo GET.

Il sesto passo consiste nella creazione del primo file HTML vero e proprio, denominato requestmembership.html. Si parte copiando la struttura di index, ma all'interno di questo file sarà presente la form con tutto ciò che ne consegue. Questa pagina avrà probabilmente il suo foglio di stile CSS dedicato, con una grafica specifica, e avrà anche il suo controller JavaScript, chiamato

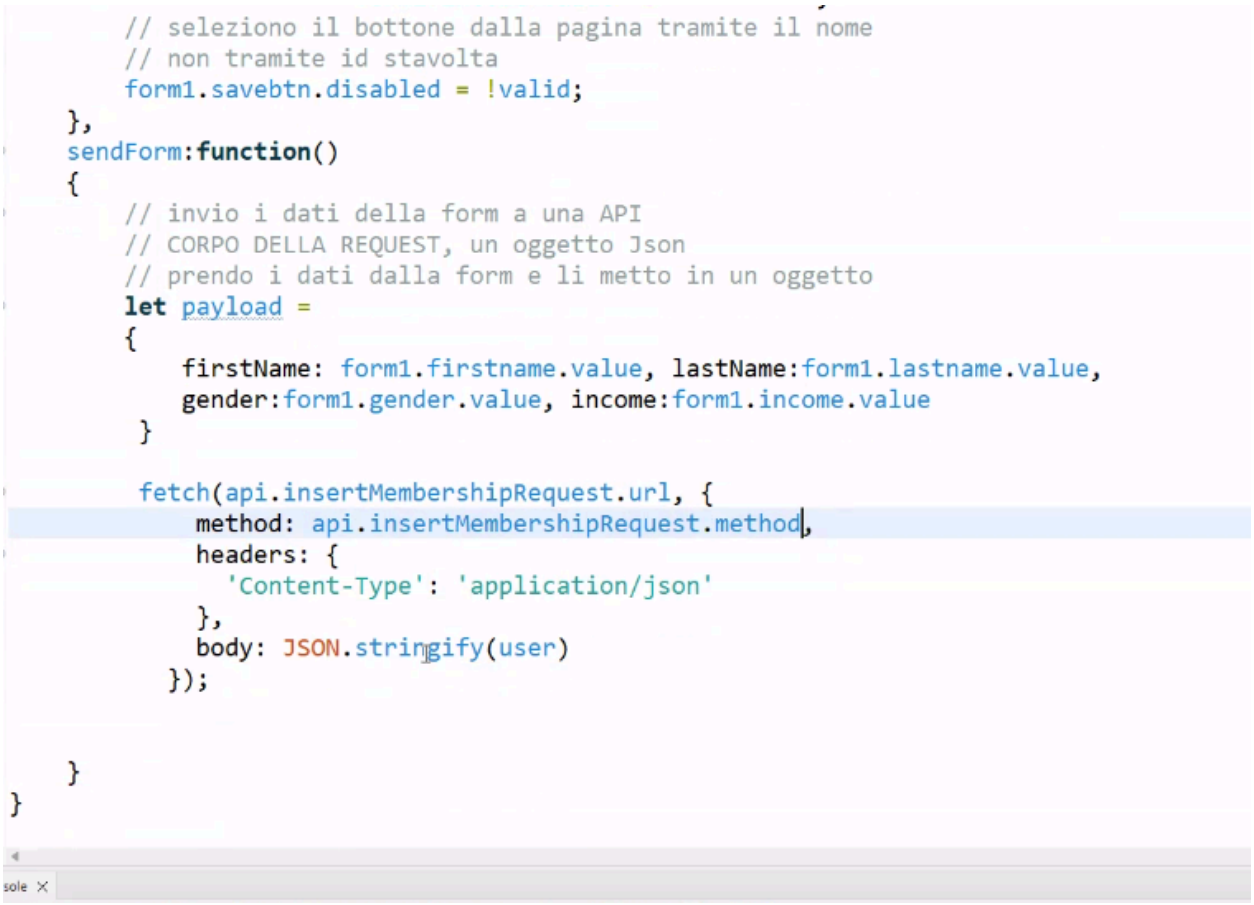
requestmembership.js. Ogni pagina seguirà questa struttura, disponendo del proprio file HTML, del proprio controller e del proprio foglio CSS.

Approfittando della situazione, si carica subito il file contenente l'indirizzo delle API. Successivamente si procede alla scrittura completa del codice HTML. Gli elementi di interesse includono l'inserimento di un div e di un h1, eventualmente accompagnati da paragrafi di descrizione. Si passa quindi alla creazione del form, ovvero il modulo dove vengono inseriti i dati. I campi richiesti sono il nome, il cognome e il gender, per il quale vengono previste due opzioni tramite l'elemento option.

Successivamente si inseriscono vari input e un bottone con la scritta "Register now for your future".

Quando si sviluppa per il web, è necessario aprire l'indirizzo sul browser e accedere alla console degli sviluppatori, in particolare alla sezione Rete. Qui è possibile verificare se alcuni file non sono stati caricati correttamente. Il debug frontend permette di individuare errori come il mancato caricamento del CSS specifico o del file requestmembership.js, che restituiscono un errore 404 not found, indicando che tali file non sono stati creati. Durante lo sviluppo del frontend è fondamentale tenere sempre aperta la console del browser per monitorare questi aspetti.

L'ottavo passo consiste nella creazione del controller della pagina, che gestisce la logica. Si desidera implementare due comportamenti, il primo dei quali prevede che il bottone si abiliti solamente quando tutti i campi sono stati compilati. Si procede quindi in JavaScript creando un nuovo file, il controller requestmembership.js, che rappresenta un oggetto improvvisato per gestire tutto ciò che serve alla pagina, includendo i metodi e le proprietà necessarie. Si inizia scrivendo una funzione denominata checkForm, riferita al form1. Per comodità si assegna un nome al bottone, in modo da poterlo recuperare tramite questo identificativo. La logica prevede che se checkForm restituisce false, allora il bottone rimane disabilitato.



```
        // seleziono il bottone dalla pagina tramite il nome
        // non tramite id stavolta
        form1.savebtn.disabled = !valid;
    },
    sendForm:function()
    {
        // invio i dati della form a una API
        // CORPO DELLA REQUEST, un oggetto Json
        // prendo i dati dalla form e li metto in un oggetto
        let payload =
        {
            firstName: form1.firstname.value, lastName:form1.lastname.value,
            gender:form1.gender.value, income:form1.income.value
         }

         fetch(api.insertMembershipRequest.url, {
             method: api.insertMembershipRequest.method,
             headers: {
               'Content-Type': 'application/json'
             },
             body: JSON.stringify(user)
           });


    }
}
```

A questo punto è necessario effettuare una fetch con il metodo POST. In precedenza non sono state scritte le API vere e proprie, ma gli indirizzi delle API, proprio per poterli utilizzare in questo momento. La funzione fetch si occupa di inviare una richiesta HTTP e di comunicare con le API. Si scrive quindi api.insertMembershipRequest.url per specificare l'indirizzo da contattare.

L'header serve a comunicare alle API che ciò che viene inviato è in formato JSON. Successivamente si utilizza il metodo then, che viene eseguito quando l'operazione è stata completata, attraverso il meccanismo delle promise.

Il file HTML deve essere collegato al controller. Il bottone deve essere configurato con l'attributo onclick che richiama la funzione sendForm.

Ora si passa al backend, dove si procede alla creazione delle entità, dei repository e dei rest controller. La prima operazione consiste nella creazione di una nuova classe all'interno del package com.generation.vault.model.entities.

Si sta seguendo questo metodo partendo dal frontend per motivi didattici, ma tipicamente lo sviluppo inizia dalle API. Si procede quindi con la stesura dell'entità e dell'interfaccia, che implementa JpaRepository. A questo componente è sufficiente indicare cosa deve gestire.

A questo punto si scrive il rest controller, al quale viene assegnata una mappatura di base. Si utilizza l'annotazione @PostMapping perché il metodo POST serve per creare nuovi oggetti. L'annotazione @RequestBody indica che i dati che arrivano sono in formato JSON e devono essere trasformati nell'oggetto newMembership. In questo modo i dati provenienti dalla form diventano un oggetto newMembership.

1. **Quando si effettua una fetch è necessario inviare sempre una stringa in formato JSON, per cui si utilizza la funzione stringify.**
2. **JSON.stringify è una funzione di JavaScript che prende un oggetto o un valore JavaScript e lo converte in una stringa nel formato JSON**

Sul server poi Spring MVC (con @RequestBody) prende quella stringa JSON e la riconverte automaticamente nell'oggetto Java MembershipRequest. È un passaggio obbligatorio per comunicare tra frontend JavaScript e backend Java