MODIFICA 🡪 SIW-FOOD

1

Stampare il numero totale di ricette: sia tot sia il totale per categoria:

* in ricetteController

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

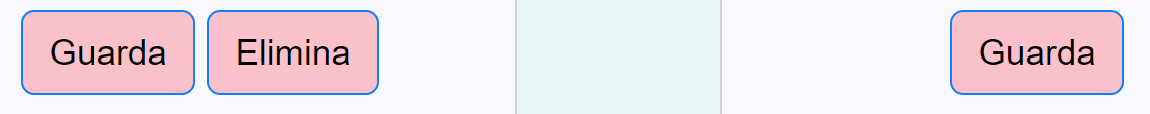
🡪

* Immagine che contiene testo, Carattere, linea, schermata

  Descrizione generata automaticamentein ricette.html
* In ricetteSingolaCategoria 🡪 nel /ricette/{categoria} già ho tutto quindi :

<div th:text="${ricette.size()}"></div>

2

Fare in modo che il bottone “Elimina” sia solo per le ricette di quell’user e quindi non che appare la scritta:

* In ricetteController:

@GetMapping("/ricette/{categoria}")

User current= **this**.userService.getUser(credentials.getUser().getId());

model.addAttribute("current", current);

* In cuoco/ricetteSingolaCategoria.html

<a th:if="${current.id==ricetta.user.id}"

th:href="@{'/deleteRicetta/'+ ${ricetta.id} + '/' + ${categoria}}"

class="btn btn-primary" style="background-color: pink;">Elimina</a>

Altro modo che avevo fatto per il tasto modifica:

* In controller:

String currentUsername = userDetails.getUsername();

model.addAttribute("currentUsername", currentUsername);

* In html:

<a th:if="${currentUsername == ricetta.user?.credentials?.username}"

3

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteIl tasto “Modifica” sta in tutti ma gli user non creatori della ricetta non possono modificare una ricetta non loro :

* In html tolgo la condizione di if per il tasto e aggiungo l’errore:

<!-- Visualizza il messaggio di successo se presente -->

<div th:if="${error}" style="color: red;">

<p th:text="${error}"></p>

</div>

4

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteAggiunta degli ingredienti in modo tale che qualsiasi che fa il match con le lettere maiuscole

* Oppure aggiungo: findbyNomeIgnoreCase

5

Stessa cosa per ogni ricetta ma avendo in validator metto:

ricetteRepository.existsByNomeIgnoreCase(ricetta.getNome())) {

6

Ingredienti in ordine di nome:

* @Query("SELECT i FROM Ingrediente i ORDER BY i.nome ")

List<Ingrediente> findAll();

**Codice per gli ingredienti:**

Immagine che contiene testo, schermata, Carattere, calligrafia

Descrizione generata automaticamente

I)

* <div th:each="rigaRicetta : ${ricetta.righeRicetta}">: Un ciclo Thymeleaf che itera su ogni rigaRicetta nella lista ricetta.righeRicetta.
* <select>: Un menù a discesa per selezionare gli ingredienti. Ogni opzione è generata da un ciclo interno th:each="ing : ${ingredienti}", e l'attributo th:selected viene utilizzato per pre-selezionare l'ingrediente corrente.
* <input>: Un campo di testo per inserire la quantità dell'ingrediente.
* <button>: Un pulsante per rimuovere l'ingrediente corrente dalla lista.

II)

* Il tag <template> è un elemento HTML speciale utilizzato per dichiarare frammenti di codice che non vengono renderizzati nel DOM finché non vengono esplicitamente utilizzati tramite JavaScript. In altre parole, tutto ciò che si trova all'interno del tag <template> non viene visualizzato immediatamente sulla pagina, ma può essere clonato e inserito dinamicamente nel DOM quando necessario.
* Il template viene utilizzato dinamicamente tramite JavaScript per aggiungere nuovi ingredienti alla ricetta. Ecco come:

🡪 Il tag <template> permette di definire un frammento di HTML che non viene visualizzato immediatamente.

Questo frammento contiene una struttura predefinita per aggiungere ingredienti: un menù a tendina per selezionare l'ingrediente, un campo di input per la quantità e un pulsante per rimuovere l'ingrediente.

Tramite JavaScript, il contenuto del template può essere clonato e inserito dinamicamente nel DOM, permettendo agli utenti di aggiungere quanti ingredienti desiderano senza dover ricaricare la pagina.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Utilizza document.addEventListener('DOMContentLoaded', ...) per assicurarsi che il codice all'interno della funzione venga eseguito solo dopo che l'intero documento HTML è stato completamente caricato. Questo è importante per garantire che tutti gli elementi del DOM siano disponibili quando il codice tenta di accedervi.
* addButton: Seleziona il pulsante con id="add-ingrediente", che viene cliccato per aggiungere nuovi ingredienti.
* ingredientiSection: Seleziona la sezione con id="ingredienti-section", dove verranno aggiunti i nuovi ingredienti.
* ingredientTemplate: Seleziona il contenuto del template con id="ingredient-template", che contiene la struttura HTML da clonare per aggiungere nuovi ingredienti.

Aggiunge un event listener al pulsante addButton. Quando il pulsante viene cliccato:

* document.importNode(ingredientTemplate, true): Crea una copia (clone) del contenuto del template. Il parametro true indica che viene clonata anche l'intera struttura figlia (deep clone).
* ingredientiSection.appendChild(clone): Aggiunge il clone alla fine della sezione ingredientiSection, rendendolo visibile sulla pagina.

Aggiunge un event listener all'intera sezione ingredientiSection per catturare i clic su eventuali pulsanti di rimozione.

* event.target.closest('.remove-ingrediente'): Controlla se l'elemento cliccato (o uno dei suoi antenati) ha la classe remove-ingrediente.
* Se sì, trova la riga (div) più vicina con la classe ingredienti-group, che rappresenta l'intero gruppo di input per un ingrediente.
* row.remove(): Rimuove questa riga dal DOM, eliminando così l'ingrediente dalla lista.

- **Deep Clone**: Il parametro true nel metodo importNode assicura che tutta la struttura HTML all'interno del template venga clonata, non solo il nodo di livello superiore.

- **Delegazione degli Eventi**: L'uso della delegazione degli eventi nella sezione ingredientiSection per gestire i clic sui pulsanti di rimozione è una tecnica efficiente. Invece di aggiungere un event listener a ogni singolo pulsante di rimozione, si aggiunge un solo listener alla sezione contenitore. Questo è particolarmente utile quando si aggiungono e si rimuovono dinamicamente elementi, poiché non è necessario aggiungere o rimuovere listener per ogni singolo elemento

**Differenza @{} - ${} e th:field – th:text**

* <p>Nome della ricetta: ${ricetta.nome}</p>

<input type="text" value="${ricetta.nome}">

* <a th:href="@{'/ricetta/' + ${ricetta.id}}">Guarda la ricetta</a>

Th:field 🡪 è usato per i form <input type="text" th:field="\*{nome}"> </div> <div> <button type="submit">Salva</button>

Th:text🡪 per visualizzare il contenuto dall’area utente <p th:text="${ricetta.nome}">Nome della ricetta</p>

7

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteFare in modo che la data di nascita sia compresa in un certo range:

* E in messages aggiungere il messaggio “ userdata= …..”

8

Fare in modo che un cuoco possa modificare i suoi stessi dati

9

Url del modifica non deve funzionare se incollato nel login di un altro utente cuoco

* Immagine che contiene testo, schermata, Carattere, Pagina Web

  Descrizione generata automaticamente
* In ricettaCuoco.html e ricettaAdmin.html :

<a th:if="${currentUsername == ricetta.user?.credentials?.username}"

th:href="@{'/ricetta/modifica/'+ ${ricetta.id}+'/'+${ricetta.user.id}}"

class="button">Modifica</a>

10

Stessa cosa per la modifica di un utente

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

11

Fare in modo che tutti vedano il tasto modifica ma solo chi è l’user stesso o l’admin possono modificare :

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

MODIFICA – ROMEXPLORER

1

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteEliminazione di un’attrazione:

* In AttrazioniController:
* In /admin/attrazioni.html

<a th:href="@{'/eliminaAttrazione'+'/'+${attrazione.id}}"

class="btn text-center btn btn-primary me-md-2 my-3" style="color:white">Elimina</a>

* In attrazioniRepository:

**void** deleteById(Long id);

* In AttrazioniService

**public** **void** deleteAttrazioneById(Long id) {

**this**.AttrazioneRepository.deleteById(id);

}

2

Modifica sarebbe da fare ma questa è la modifica del cuoco che va :

Immagine che contiene testo, schermata, documento, Carattere

Descrizione generata automaticamente

3

Fare in modo che l’url del pagamento sia accessibile solo all’user che effettua quella prenotazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

4

Contare le attrazioni :

* <div th:text="${monumenti.size()}"></div>

5

Attrazioni con piu prenotazioni:

* SELECT a.nome, COUNT(p) as numPrenotazioni

FROM Prenotazione p

JOIN Attrazione a ON p.attrazione = a.id

GROUP BY a.nome

HAVING count(p) = (

SELECT MAX(numPrenotazioni)

FROM (

SELECT COUNT(p2) as numPrenotazioni

FROM Prenotazione p2

GROUP BY p2.attrazione

) as prenotazioni\_per\_attrazione

)

ORDER BY numPrenotazioni DESC;

MODIFICA SIW-FOOD:

Ingredienti ordinati anche in ricetta/{id}

* **public** **interface** RigaRicettaRepository **extends** JpaRepository<RigaRicetta, Long> {

@Query("SELECT r FROM RigaRicetta r WHERE r.ricetta.id = :ricettaId ORDER BY r.ingrediente.nome ASC")

List<RigaRicetta> findByRicettaIdOrderByIngredienteNomeAsc(@Param("ricettaId") Long ricettaId);

}

* In ricettaService:

**public** Ricette findByIdRicetta(Long id) {

Ricette ricetta = ricetteRepository.findById(id).orElseThrow(() -> **new** RuntimeException("Ricetta

non trovata"));

List<RigaRicetta> righeRicetta =rigaRicettaRepository.findByRicettaIdOrderByIngredienteNomeAsc(id);

ricetta.setRigheRicetta(righeRicetta);

**return** ricetta;

}

**Annotazioni**

* orphanRemoval = true: per specificare che un'entità "orfana" deve essere rimossa automaticamente dal database 🡪 se un'entità figlia non è più associata a un'entità padre (è orfana), sarà automaticamente rimossa dal database
* @Transactional: garantisce che il metodo è eseguito all'interno di una transazione, gestendo automaticamente l'inizio, il commit o il rollback della transazione.
* @ModelAttribute: può essere utilizzato sia per passare i dati al modello prima che un controller venga chiamato, sia per legare i dati del form a un oggetto
* @PathVariable: lega la parte dell'URI specificata a un parametro del metodo🡪nei metodi del controller per ottenere i valori dalle parti dinamiche dell'URL
* @RequestParam: Estrae i parametri dalla query string dell'URL e li lega ai parametri del metodo del controller

**AuthConfiguration**

Fornisce un bean AuthenticationManager che gestisce l'autenticazione degli utenti.

 httpSecurity.csrf().and().cors().disable(): Disabilita la protezione CSRF e CORS.

 authorizeHttpRequests(): Configura le politiche di autorizzazione per le richieste HTTP.

* requestMatchers(HttpMethod.GET, ..., "/attrazione/{id}", "favicon.ico").permitAll(): Permette a chiunque di accedere alle risorse specificate con richieste GET.
* requestMatchers(HttpMethod.POST, "/register", "/login").permitAll(): Permette a chiunque di inviare richieste POST ai punti di accesso per la registrazione e il login.
* requestMatchers(HttpMethod.GET, "/admin/\*\*").hasAnyAuthority(ADMIN\_ROLE): Solo gli utenti con il ruolo di amministratore possono accedere alle risorse sotto /admin/\*\* con richieste GET.
* requestMatchers(HttpMethod.POST, "/admin/\*\*").hasAnyAuthority(ADMIN\_ROLE): Solo gli utenti con il ruolo di amministratore possono accedere alle risorse sotto /admin/\*\* con richieste POST.
* anyRequest().authenticated(): Qualsiasi altra richiesta richiede che l'utente sia autenticato.

 formLogin(): Configura il login basato su form.

* loginPage("/login").permitAll(): Specifica la pagina di login e permette l'accesso a tutti.
* defaultSuccessUrl("/success", true): Dopo un login di successo, reindirizza l'utente alla pagina /success.
* failureUrl("/login?error=true"): Dopo un fallimento nel login, reindirizza l'utente alla pagina di login con un parametro di errore.

 logout(): Configura il logout.

* logoutUrl("/logout"): Specifica l'URL per effettuare il logout.
* logoutSuccessUrl("/"): Dopo un logout di successo, reindirizza l'utente alla home page.
* invalidateHttpSession(true): Invalida la sessione HTTP al momento del logout.
* deleteCookies("JSESSIONID"): Cancella i cookie di sessione.
* logoutRequestMatcher(new AntPathRequestMatcher("/logout")): Matcher per la richiesta di logout.
* Immagine che contiene testo, schermata, Carattere, numero

  Descrizione generata automaticamenteclearAuthentication(true): Cancella le informazioni di autenticazione.

**Tipi di Operazioni Cascade**

Le operazioni di cascata si definiscono usando l'annotazione @Cascade su una relazione tra entità (ad esempio, @OneToMany, @ManyToOne, @OneToOne, @ManyToMany). I tipi di operazioni di cascata includono:

1. **CascadeType.PERSIST**:
   * Quando si persiste (salva) un'entità principale, tutte le entità correlate vengono automaticamente persistite.
   * Esempio: Se si salva un Order, tutte le OrderItem associate a quell'ordine saranno automaticamente salvate.
2. **CascadeType.MERGE**:
   * Quando si unisce (merge) un'entità principale, tutte le entità correlate vengono automaticamente unite.
   * Esempio: Se si aggiorna un Order, tutte le OrderItem associate a quell'ordine saranno automaticamente aggiornate.
3. **CascadeType.REMOVE**:
   * Quando si rimuove un'entità principale, tutte le entità correlate vengono automaticamente rimosse.
   * Esempio: Se si cancella un Order, tutte le OrderItem associate a quell'ordine saranno automaticamente cancellate.
4. **CascadeType.REFRESH**:
   * Quando si aggiorna (refresh) un'entità principale, tutte le entità correlate vengono automaticamente aggiornate.
   * Esempio: Se si ricarica un Order dal database, tutte le OrderItem associate a quell'ordine saranno automaticamente ricaricate.
5. **CascadeType.DETACH**:
   * Quando si stacca (detach) un'entità principale dal contesto di persistenza, tutte le entità correlate vengono automaticamente staccate.
   * Esempio: Se si stacca un Order dal contesto di persistenza, tutte le OrderItem associate a quell'ordine saranno automaticamente staccate.
6. **CascadeType.ALL**:
   * Propaga tutte le operazioni di cascata sopra menzionate.
   * Esempio: Salvataggio, aggiornamento, rimozione, aggiornamento dal database e distacco di un Order propagheranno automaticamente queste operazioni a tutte le OrderItem associate.

**Fetch**

* @OneToOne(fetch = FetchType.EAGER) // Caricamento immediato
* Semplice da usare quando sai che hai sempre bisogno dell'entità associata.
* Può causare un sovraccarico di memoria se ci sono molte associazioni o se l'associazione contiene molti dati.
* @OneToOne(fetch = FetchType.LAZY) // Caricamento su richiesta
* Carica i dati solo quando necessario, risparmiando memoria e migliorando le prestazioni iniziali.

**Metodo in UserService**

* @Transactional

**public** Optional<User> getUserByCredentials(UserDetails userDetails) {

String username = userDetails.getUsername();

**return** credentialsRepository.findByUsername(username).map(Credentials::getUser);

}

Qui si utilizza il credentialsRepository per trovare le Credentials associate a quel nome utente. Il metodo findByUsername restituisce un oggetto Optional<Credentials>, che rappresenta una possibile presenza o assenza di un valore non nullo.

AWS

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Idirizzo IP

[502 Bad Gateway (eba-weufihmg.eu-central-1.elasticbeanstalk.com)](http://siw-food-env-1.eba-weufihmg.eu-central-1.elasticbeanstalk.com/)