IMPLEMENTATION DE L'INTERFACE D'ADMINISTRATION

Objectifs

Fournir à l'administateur une interface de gestion des ingrédients. L'interface de gestion des utilisateurs sera ajoutée ultérieurement.

Description de la démarche globale

Nous allons procéder par les étapes suivantes :

1. Mise à jour de l'API côté backend pour traiter l'ensemble des opérations **CRUD** idem du côté frontend au niveau du service.
2. Création des composants qui seront intégrés dans le composant page `ingredient-manager-page`.
3. Ajout des constantes Bootstrap à notre projet (nécessaire pour l'étape 5) et création d'un objet ingrédient littéral pour fournir les valeurs par défaut.
4. Implémentation du code des composants.
5. Amélioration de l'interface en intégrant le formulaire de saisie/modification des ingrédients dans un mmode de **Bootstrap**.

1/ Mise à jour de l'API

1.1/ Mise à jour de l'API côté backend

Le contrôleur `IngredientController` est incomplet et ne nous permettra pas dans sa version actuelle de pouvoir tester l'interface d'administration mise en place dans le frontend.

Dans **IntelliJ** ouvrir le fichier `IngredientController` afin que le code corresponde à celui-ci :

```
package org.ldv.savonapi.controller

import org.ldv.savonapi.model.dao.IngredientDAO
import org.ldv.savonapi.model.entity.Ingredient
import org.springframework.http.HttpStatus
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.*



/**
 * Contrôleur REST pour la gestion des ingrédients.
 *
 * Fournit des endpoints pour effectuer des opérations CRUD (Create, Read, Update,
Delete).
 */
@RestController
@CrossOrigin
```

```
@RequestMapping("/api-savon/v1/ingredient")
class IngredientController (val ingredientDAO: IngredientDAO){

    /**
     * Récupère la liste de tous les ingrédients disponibles.
     *
     * @return Liste d'ingrédients.
     */
    @GetMapping
    fun index(): List<Ingredient> {
        return this.ingredientDAO.findAll()
    }

    /**
     * Récupère un ingrédient spécifique par son ID.
     *
     * @param id Identifiant de l'ingrédient.
     * @return L'ingrédient s'il est trouvé, sinon retourne `404 Not Found`.
     */
    @GetMapping("/{id}")
    fun show(@PathVariable id: Long): ResponseEntity<Ingredient> {
        val ingredient = ingredientDAO.findById(id)
        return if (ingredient.isPresent) {
            ResponseEntity.ok(ingredient.get())
        } else {
            ResponseEntity.notFound().build()
        }
    }

    /**
     * Enregistre un nouvel ingrédient dans la base de données.
     *
     * @param ingredient L'ingrédient à enregistrer (données envoyées en JSON).
     * @return L'ingrédient créé avec un code HTTP 201 Created.
     */
    @PostMapping
    fun store(@RequestBody ingredient: Ingredient): ResponseEntity<Ingredient> {
        val savedIngredient = ingredientDAO.save(ingredient)
        return ResponseEntity.status(HttpStatus.CREATED).body(savedIngredient)
    }

    /**
     * Met à jour un ingrédient existant.
     *
     * @param id Identifiant de l'ingrédient à mettre à jour.
     * @param updatedIngredient Nouvelles valeurs de l'ingrédient.
     * @return L'ingrédient mis à jour ou un code 404 si l'ingrédient n'existe
pas.
     */
    @PutMapping("/{id}")
    fun store(
        @PathVariable id: Long,
        @RequestBody updatedIngredient: Ingredient
    ): ResponseEntity<Ingredient> {
```

```
        return ingredientDAO.findById(id).map { existingIngredient ->
            existingIngredient.nom = updatedIngredient.nom
            existingIngredient.iode = updatedIngredient.iode
            existingIngredient.ins = updatedIngredient.ins
            existingIngredient.sapo = updatedIngredient.sapo
            existingIngredient.volMousse = updatedIngredient.volMousse
            existingIngredient.tenueMousse = updatedIngredient.tenueMousse
            existingIngredient.douceur = updatedIngredient.douceur
            existingIngredient.lavant = updatedIngredient.lavant
            existingIngredient.durete = updatedIngredient.durete
            existingIngredient.solubilite = updatedIngredient.solubilite
            existingIngredient.sechage = updatedIngredient.sechage
            existingIngredient.estCorpsGras = updatedIngredient.estCorpsGras

            ResponseEntity.ok(ingredientDAO.save(existingIngredient))
        }.orElse(ResponseEntity.notFound().build())

    }

    /**
     * Supprime un ingrédient de la base de données.
     *
     * @param id Identifiant de l'ingrédient à supprimer.
     * @return Code HTTP 204 No Content si la suppression réussit, sinon 404 Not
Found.
     */
    @DeleteMapping("/{id}")
    fun delete(@PathVariable id: Long): ResponseEntity<Void> {
        return if (ingredientDAO.existsById(id)) {
            ingredientDAO.deleteById(id)
            ResponseEntity.noContent().build()
        } else {
            ResponseEntity.notFound().build()
        }
    }

    /**
     * Supprime tous les ingrédients de la base de données.
     *
     * @return Code HTTP 204 No Content si la suppression réussit.
     */
    @DeleteMapping("/all")
    fun deleteAll(): ResponseEntity<Void> {
        ingredientDAO.deleteAll()
        return ResponseEntity.noContent().build()
    }


}
```

1.2/ Mise à jour du service API

Selon le déroulement de votre projet vous avez créé un ou deux service(s) pour assurer la liaison avec l'API du backend. Le fichier du service prenant en charge la liaison API pour les ingrédients doit comporter l'ensemble des opérations **CRUD**.

A FAIRE :

1. Identifier votre fichier service prenant en charge la récupération des ingrédients.
2. Dans le fichier concerné, remplacer le contenu de la classe (corps de la classe, pas le fichier entier) du service par le code suivant :

```
private apiUrl = 'http://localhost:8080/api-savon/v1/ingredient';

    constructor(private http: HttpClient) {}

    /**
     * Récupère tous les ingrédients depuis l'API.
     * @returns Observable contenant la liste des ingrédients.
     */
    getAllIngredients(): Observable<Ingredient[]> {
        return this.http.get<Ingredient[]>(this.apiUrl);
    }

    /**
     * Récupère un ingrédient spécifique par son ID.
     * @param id - Identifiant de l'ingrédient.
     * @returns Observable contenant l'ingrédient correspondant.
     */
    getIngredientById(id: number): Observable<Ingredient> {
        return this.http.get<Ingredient>(`${this.apiUrl}/${id}`);
    }

    /**
     * Ajoute un nouvel ingrédient à la base de données.
     * @param ingredient - L'objet Ingredient à ajouter.
     * @returns Observable contenant l'ingrédient ajouté.
     */
    addIngredient(ingredient: Ingredient): Observable<Ingredient> {
        return this.http.post<Ingredient>(this.apiUrl, ingredient);
    }

    /**
     * Met à jour un ingrédient existant dans la base de données.
     * @param id - Identifiant de l'ingrédient à mettre à jour.
     * @param ingredient - L'objet Ingredient mis à jour.
     * @returns Observable contenant l'ingrédient modifié.
     */
    updateIngredient(id: number, ingredient: Ingredient): Observable<Ingredient> {
        return this.http.put<Ingredient>(`${this.apiUrl}/${id}`, ingredient);
    }

    /**
     * Supprime un ingrédient de la base de données.
```

```
     * @param id - Identifiant de l'ingrédient à supprimer.
     * @returns Observable indiquant le succès ou l'échec de la suppression.
     */
    deleteIngredient(id: number): Observable<void> {
        return this.http.delete<void>(`${this.apiUrl}/${id}`);
    }

    /**
     * Supprimer TOUS les ingrédients.
     */
    deleteAllIngredients(): Observable<void> {
        return this.http.delete<void>(`${this.apiUrl}/all`);
      }
```

2/ Création des composants

Si vous avez récupérez et travaillez avec la correction de vos professeurs, votre projet **Angular** comporte déjà un dossier `shared`. Dans le cas contraire nous vous invitons à le faire.

Puis vous allez pouvoir générer les 4 composants suivants :

1. Liste des ingrédients : `ng generate component shared/ingredient-list`
2. Formulaire création/modification : `ng generate component shared/ingredient-form`
3. Import/export CSV : `ng generate component shared/ingredient-import-export`
4. Modale partagé pour la boite de dialogue de confirmation : `ng generate component shared/modal-box-confirmation`

3/ Ajout des constantes Bootstrap & Objet ingredient pour initialisation

3.1/ Constantes Bootstrap

Nous allons utiliser des **Modals Bootsrap** pour améliorer l'expérience utilisateur. Les **Modals** comporte à la fois du code **HTML/CSS**, mais également du code **JavaScript**. Afin de pouvoir gérer ce code **JS** directement dans les fichiers **TypeScript** de nos composants, nous aurons besoin des constantes associées aux **Modals**.

Dans le terminal de **VsCode**, exécuter la commande suivante : `npm install bootstrap @types/bootstrap --save`

3.2/ Objet ingrédient d'initialisation

Plusieurs composants ont besoin de définir un objet `Ingredient` pour fonctionner. Cependant **TypeScript** n'autorise pas par défaut d'initialiser un objet avec `null`. Nous sommes donc dans l'obligation de fournir des valeurs d'initialisation au moyen d'un objet littéral Ingredient contenant justement des valeurs d'initialisation. Cette tâche s'avère pénible surtout quand elle devient répétitive. Afin d'alléger le code à implémenter, nous allons définir un objet littéral constant que nous pourrons utiliser pour initialiser les objets `Ingredient` des différents composants.

Marche à suire :

1. Créer un dossier `constants` dans le dossier `shared` (`../src/app/shared/constants`).
2. Créer un nouveau fichier `ingredient.constants.ts` (`src\app\shared\constants\ingredient.constants.ts`).
3. Ouvrir le fichier `ingredient.constants.ts` et placer le code suivant :

```
import { Ingredient } from "../../models/Ingredient";

export const DEFAULT_INGREDIENT: Ingredient = {
    id: null,
    nom: '',
    iode: 0,
    ins: 0,
    sapo: 0,
    volMousse: 0,
    tenueMousse: 0,
    douceur: 0,
    lavant: 0,
    durete: 0,
    solubilite: 0,
    sechage: 0,
    estCorpsGras: true,
    ligneIngredients: []
};
```

Vous pourrez voir par la suite, qu'on utilisera plusieurs fois cette classe en l'important et en utilisant l'instruction : `{ DEFAULT_INGREDIENT }`

4/ Implémentation du code

Composant `modal-box-confirmation`

Fichier de la logique `modal-box-confirmation.component.ts` :

```
import { Component, Input } from '@angular/core';
import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';

@Component({
  selector: 'app-modal-box-confirmation',
  templateUrl: './modal-box-confirmation.component.html',
  styleUrl: './modal-box-confirmation.component.css'
})
export class ModalBoxConfirmationComponent {

    // Titre du modal :
    @Input()
    titre: string = "Le titre du modal par défaut";

    // Message à afficher dans le corps du modal :
    @Input()
```

```
    message: string = "Texte du corps du modal par défaut.";

    // Texte associé au bouton d'action :
    @Input()
    btnText: string = "Appliquer";

    // Couleur du bouton d'action :
    @Input()
    btnColor: string = "info";  // Valeurs possibles : primary; secondary;
success; danger; warning; info; light; dark

    constructor(public activeModal: NgbActiveModal) {}
}
```

Fichier de la vue `modal-box.confirmation.component.html` :

```
<!-- ---------------------------->
<!--  MODALE DE CONFIRMATION  -->
<!-- ---------------------------->

    <!-- Entête du modal  -->
    <!-- ---------------- -->
    <div class="modal-header">
        <!-- <h5 class="modal-title">Confirmer la suppression</h5> -->
        <h5 class="modal-title">{{ titre }}</h5>
        <button type="button" class="btn-close" aria-label="Fermer"
                (click)="activeModal.dismiss('Cross click')">
        </button>
    </div>

    <!-- Corps du modal : -->
    <!-- --------------- -->
    <div class="modal-body">
        {{ message }}
    </div>

    <!-- Pied du modal : -->
    <!-- --------------- -->
    <div class="modal-footer">
        <!-- Bouton annulation -->
        <button type="button" class="btn btn-secondary"
                (click)="activeModal.close('Close click')">
            Annuler
        </button>

        <!-- Bouton d'action -->
        <button type="button" class="btn btn-{{ btnColor }}"
        (click)="activeModal.close('execute')">
            {{ btnText }}
```

```
        </button>
    </div>
```

Composant `ingredient-list`

Fichier de la logique `ingredient-list.component.ts` :

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Ingredient } from '../../models/Ingredient';
import { NgbModal } from '@ng-bootstrap/ng-bootstrap';
import { ModalBoxConfirmationComponent } from '../modal-box-confirmation/modal-
box-confirmation.component';

@Component({
  selector: 'app-ingredient-list',
  templateUrl: './ingredient-list.component.html',
  styleUrl: './ingredient-list.component.css'
})
export class IngredientListComponent {

    // Récupération des valeurs de la vue et du composant parent :
    // ------------------------------------------------------------
    @Input()
    ingredients: Ingredient[] = [];


    // Envoi de valeurs vers la vue et au composant parent :
    // -----------------------------------------------------
    @Output()
    edit = new EventEmitter<Ingredient>();

    @Output()
    delete = new EventEmitter<number>();

    @Output()
    deleteAll = new EventEmitter<void>(); // Événement pour supprimer TOUS les
ingrédients


    // ------------------------------------------------------------
    // Gestion de la suppression d'un ingrédient avec un modal :
    // ------------------------------------------------------------
    ingredientToDelete: Ingredient | null = null;

    constructor(private modalService: NgbModal) {}

    // Méthodes NgbModal pour ouvrir et configurer les modals :
    // --------------------------------------------------------

    /**
```

```
     * Modal de suppression de tous les ingrédients :
     */
    openDeleteAllModal() {
        const modalRef = this.modalService.open(ModalBoxConfirmationComponent,
{centered: true});
        modalRef.componentInstance.titre = "Suppression de tous les ingrédients";
        modalRef.componentInstance.message = "Êtes-vous sûr de vouloir supprimer
TOUS les ingrédients ? Cette action est irréversible.";
        modalRef.componentInstance.btnText = "Supprimer TOUT !";
        modalRef.componentInstance.btnColor = "danger";

        // Gestion des actions :
        modalRef.result.then((result: string) => {
            if (result === 'execute') {
                this.deleteAll.emit(); // Exécution de la suppression de tous les
ingrédients
            }
        })
    }

    /**
     * Modal de suppression d'un ingrédient :
     */
    openDeleteOneModal(ingredient: Ingredient) {
        const modalRef = this.modalService.open(ModalBoxConfirmationComponent,
{centered: true});
        modalRef.componentInstance.titre = "Suppression d'un ingrédient";
        modalRef.componentInstance.message = `Êtes-vous sûr de vouloir supprimer
l’ingrédient ${ ingredient?.nom } ?`;
        modalRef.componentInstance.btnText = "Supprimer !";
        modalRef.componentInstance.btnColor = "danger";

        // Gestion des actions :
        modalRef.result.then((result: string) => {
            if (result === 'execute') {
                // Exécution de la suppression de l'ingrédient
                this.ingredientToDelete = ingredient;
                this.delete.emit(this.ingredientToDelete.id!);
                this.ingredientToDelete = null;
            }
        })
    }

    /**
     * Émet un événement pour éditer un ingrédient.
     * @param ingredient L'ingrédient sélectionné.
     */
    editIngredient(ingredient: Ingredient): void {
        this.edit.emit(ingredient);
    }

}
```

Fichier de la vue `ingredient-list.component.html` :

```
<div class="row mt-4">
    <h2>Liste des ingrédients :</h2>
</div>



<!-- Bouton pour supprimer TOUS les ingrédients -->
<div class="row col-3 d-flex justify-content-between align-items-center ms-4 mt-3
mb-3">
    <button class="btn btn-danger" (click)="openDeleteAllModal()">Supprimer TOUS
les ingrédients</button>
</div>


<!-- ---------------------------->
<!-- Tableau des ingrédients : -->
<!-- ---------------------------->

<table class="table table-bordered table-striped mt-3" *ngIf="ingredients.length >
0">
    <thead class="table-dark">
    <tr>
        <th>ID</th>
        <th>Nom</th>
        <th>Corps Gras</th>
        <th>Ind. sapo.</th>
        <th>Ind. INS</th>
        <th>Ind. Iode</th>
        <th>Lavant</th>
        <th>Douceur</th>
        <th>Duretée</th>
        <th>Solubilité</th>
        <th>Séchage</th>
        <th>Vol. Mousse</th>
        <th>Tenue Mousse</th>
        <th>Actions</th>
    </tr>
    </thead>

    <tbody>
    <tr *ngFor="let ingredient of ingredients">
        <td>{{ ingredient.id }}</td>
        <td>{{ ingredient.nom }}</td>
        <td>{{ ingredient.estCorpsGras }}</td>
        <td>{{ ingredient.sapo }}</td>
        <td>{{ ingredient.ins }}</td>
        <td>{{ ingredient.iode }}</td>
        <td>{{ ingredient.lavant }}</td>
        <td>{{ ingredient.douceur }}</td>
```

```
        <td>{{ ingredient.durete }}</td>
        <td>{{ ingredient.solubilite }}</td>
        <td>{{ ingredient.sechage }}</td>
        <td>{{ ingredient.volMousse }}</td>
        <td>{{ ingredient.tenueMousse }}</td>

        <td>
            <button class="btn btn-warning btn-sm me-2"
(click)="editIngredient(ingredient)">
                Modifier
            </button>
            <!-- <button class="btn btn-danger btn-sm"
(click)="confirmDelete(ingredient)" data-toggle="modal" data-
target="#deleteModal"> -->
                <button class="btn btn-danger"
(click)="openDeleteOneModal(ingredient)">
                    Supprimer
                </button>
        </td>
    </tr>
    </tbody>
</table>
```

Composant `ingredient-form`

Fichier de la logique `ingredient-form.component.ts` :

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Ingredient } from '../../models/Ingredient';
import { DEFAULT_INGREDIENT } from '../constants/ingredient.constants';

@Component({
  selector: 'app-ingredient-form',
  templateUrl: './ingredient-form.component.html',
  styleUrl: './ingredient-form.component.css'
})
export class IngredientFormComponent {

    // Champs de saisie des caractéristiques : sapo; ins; iode; lavant; douceur;
dureté; solubilité; séchage; vol. mousse et tenue mousse
    ingredientFields: (keyof Ingredient)[] = [
        'sapo', 'ins','iode', 'lavant',  'douceur',  'durete',
        'solubilite', 'sechage','volMousse', 'tenueMousse',

      ];


    // Récupération des valeurs de la vue et du composant parent :
    // ------------------------------------------------------------
```

```
    @Input()
    ingredient: Ingredient = { ...DEFAULT_INGREDIENT };


    @Input()
    isEditing: boolean = false;


    // Envoi de valeurs vers la vue et au composant parent :
    // -----------------------------------------------------
    @Output()
    save = new EventEmitter<Ingredient>();
    @Output()
    cancelEdit = new EventEmitter<void>();


    // Méthodes sur le formulaire :
    // ----------------------------
    saveIngredient(): void {
        this.save.emit(this.ingredient);
    }

    cancel(): void {
        this.cancelEdit.emit();
    }
}
```

Fichier de la vue `ingredient-form.component.html` :

```
<!-- ------------------------------------------- -->
<!-- Formulaire de saisie d'un nouvel ingrédient : -->
<!-- ------------------------------------------- -->

<form (ngSubmit)="saveIngredient()" #ingredientForm="ngForm" class="mt-3">
    <div class="row">
        <div class="col-6 mb-3">
        <label for="nom" class="form-label">Nom de l'ingrédient</label>
        <input type="text" id="nom" [(ngModel)]="ingredient.nom" name="nom"
class="form-control" required>
        </div>
    </div>


    <!-- Champs de saisie des caractéristiques : sapo; ins; iode; lavant; douceur;
dureté; solubilité; séchage; vol. mousse et tenue mousse -->
    <div class="col-lg-2 mb-3" *ngFor="let field of ingredientFields">
        <label [for]="field" class="form-label">{{ field | titlecase }}</label>
        <input type="number" [id]="field" [(ngModel)]="ingredient[field]"
[name]="field" class="form-control" required>
```

```
    </div>



    <!-- Checkbox pour indiquer si c'est un corps gras -->
    <div class="form-check mb-3">
        <input type="checkbox" id="estCorpsGras"
[(ngModel)]="ingredient.estCorpsGras" name="estCorpsGras" class="form-check-
input">
        <label for="estCorpsGras" class="form-check-label">Cet ingrédient est un
corps gras</label>
    </div>


    <!-- Boutons -->
    <button type="submit" class="btn btn-primary">
        {{ isEditing ? 'Mettre à jour' : 'Ajouter' }} l'ingrédient
    </button>
    <button type="button" class="btn btn-secondary ms-2" (click)="cancel()"
*ngIf="isEditing">
        Annuler
    </button>
  </form>
```

Composant `ingredient-import-export`

Fichier de la logique `ingredient-import-export.component.ts` :

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Ingredient } from '../../models/Ingredient';
import { IngredientService } from '../../services/ingredient.service';

@Component({
  selector: 'app-ingredient-import-export',
  templateUrl: './ingredient-import-export.component.html',
  styleUrl: './ingredient-import-export.component.css'
})
export class IngredientImportExportComponent {

    @Input()
    ingredients: Ingredient[] = []; // Liste des ingrédients reçue du parent

    @Output()
    importComplete = new EventEmitter<void>(); // Emet un événement après l'import

    message: string = '';   // Message de confirmation
    error: boolean = false  // Flag pour le message d'erreur


    constructor(private ingredientService: IngredientService) {}
```

```
    /**
     * Exporte les ingrédients au format CSV et génère un fichier téléchargeable.
     */
    exportToCSV(): void {
    if (!this.ingredients || this.ingredients.length === 0) {
        this.error = true;
        this.message = "Aucun ingrédient à exporter.";
        return;
    }

    // Génération des lignes du CSV
    const csvData = this.ingredients.map(ing =>

`${ing.id},${ing.nom},${ing.iode},${ing.ins},${ing.sapo},${ing.volMousse},${ing.te
nueMousse},${ing.douceur},${ing.lavant},${ing.durete},${ing.solubilite},${ing.sech
age},${ing.estCorpsGras}`
    );

    // Ajout de l'en-tête au CSV
    const csvHeader = "ID,Nom,Indice Iode,Indice INS,Indice de
Saponification,Volume de Mousse,Tenue de Mousse,Douceur,Pouvoir
Lavant,Dureté,Solubilité,Séchage,Est Corps Gras";
    const csvContent = [csvHeader, ...csvData].join("\n");

    // Création et téléchargement du fichier CSV
    const blob = new Blob([csvContent], { type: 'text/csv' });
    const a = document.createElement('a');
    a.href = URL.createObjectURL(blob);
    a.download = "ingredients.csv";
    a.click();

    this.message = "Exportation réussie !";
    this.clearMessageAfterDelay();
    }

    /**
     * Importe des ingrédients depuis un fichier CSV.
     * @param event - L'événement déclenché par l'input file.
     */
    importFromCSV(event: any): void {
        const file = event.target.files[0];
        if (!file) return;

        const reader = new FileReader();
        reader.onload = (e) => {
            const text = e.target?.result as string;
            const rows = text.split("\n").slice(1); // Ignore l'en-tête
            rows.forEach(row => {
                const [id, nom, iode, ins, sapo, volMousse, tenueMousse, douceur,
lavant, durete, solubilite, sechage, estCorpsGras] = row.split(",");

                const newIngredient: Ingredient = {
                    id: null, // Laisse null pour générer un ID côté backend
```

```
                    nom: nom.trim(),
                    iode: parseFloat(iode),
                    ins: parseFloat(ins),
                    sapo: parseFloat(sapo),
                    volMousse: parseFloat(volMousse),
                    tenueMousse: parseFloat(tenueMousse),
                    douceur: parseFloat(douceur),
                    lavant: parseFloat(lavant),
                    durete: parseFloat(durete),
                    solubilite: parseFloat(solubilite),
                    sechage: parseFloat(sechage),
                    estCorpsGras: estCorpsGras.trim().toLowerCase() === "true",
                    ligneIngredients: []
                };


                this.ingredientService.addIngredient(newIngredient).subscribe({
                    next: () => {
                      this.importComplete.emit(); // Notifie le parent pour
rafraîchir la liste
                      this.error = false;
                      this.message = "Importation réussie !";
                    },
                    error: (error) => {
                        this.error = true;
                        this.message = `Erreur lors de l'import d'un ingrédient :
${error}`
                    }
                  });


                this.clearMessageAfterDelay();

            });
        };
        reader.readAsText(file);
    }

    /**
     * Efface le message après 3 secondes.
     */
    clearMessageAfterDelay(): void {
    setTimeout(() => {
        this.message = "";
        this.error = false;
    }, 3000);
    }

}
```

Fichier de la vue `ingredient-import-export.component.html` :

```
<!-- ----------------- -->
<!-- Import/Export CSV -->
<!-- ----------------- -->

<!-- Boutons d'import & export -->
<div class="row ms-2">
    <!-- Bouton d'import CSV : -->
    <label class="col-md-4 btn btn-primary m-3">
      Importer depuis un fichier CSV
      <input type="file" (change)="importFromCSV($event)" hidden>
    </label>

    <!-- Bouton d'export CSV : -->
    <button class="col-md-4 btn btn-success m-3" (click)="exportToCSV()">Exporter
en CSV</button>
  </div>


<!-- Message de confirmation -->
<div *ngIf="message && error" class="alert alert-danger mt-2">
    {{ message }}
</div>

<div *ngIf="message && !error" class="alert alert-success mt-2">
    {{ message }}
</div>
```

Composant `ingredient-manager-page`

Fichier de la logique `ingredient-manager-page.component.ts` :

```
import { Component, OnInit } from '@angular/core';
import { Ingredient } from '../../../models/Ingredient';
import { DEFAULT_INGREDIENT } from
'../../../shared/constants/ingredient.constants';
import { IngredientService } from '../../../services/ingredient.service';

@Component({
  selector: 'app-ingredient-manager-page',
  templateUrl: './ingredient-manager-page.component.html',
  styleUrl: './ingredient-manager-page.component.css'
})

export class IngredientManagerPageComponent implements OnInit {

    // Attributs du composant :
    // ------------------------

    ingredients: Ingredient[] = [];
```

```
    // message: string = '';
    isEditing: boolean = false; // Permet de savoir si on est en mode édition
    //ingredientToEditId: number | null = null;


    // Objet Ingredient de travail :
    // -----------------------------
    selectedIngredient: Ingredient = { ...DEFAULT_INGREDIENT };


    // Constructeur avec injection du service :
    // ----------------------------------------
    constructor(private ingredientService: IngredientService) {}


    // Fetch avec le service à l'initialisation :
    // ------------------------------------------
    ngOnInit(): void {
        this.fetchIngredients();
    }


    // ____________________________________________________
    //
    // METHODES SUR LES FORMULAIRES
    // ____________________________________________________

    /**
     * Récupère tous les ingrédients via l'API.
     */
    fetchIngredients(): void {
        this.ingredientService.getAllIngredients().subscribe({
        next: (data) => this.ingredients = data,
        error: (error) => console.error('Erreur chargement des ingrédients:',
error)
        });
    }


    /**
     * Gère la mise à jour de la liste des ingrédients après un import CSV.
     */
    handleImportComplete(): void {
        this.fetchIngredients(); // Recharge la liste après l'importation
    }


    /**
     * Ajoute ou met à jour un ingrédient.
     * @param ingredient - L'ingrédient à ajouter ou mettre à jour.
     */
    saveIngredient(ingredient: Ingredient): void {
        if (this.isEditing && ingredient.id !== null) {
        // Mode modification
```

```
        this.ingredientService.updateIngredient(ingredient.id,
ingredient).subscribe({
            next: () => {
                //this.message = 'Ingrédient mis à jour avec succès !';
                this.isEditing = false;
                this.selectedIngredient = { ...DEFAULT_INGREDIENT };
                this.fetchIngredients();    // Recharge la liste
                //this.resetForm();
            },
            error: (error) => {
            console.error('Erreur mise à jour ingrédient:', error);
            //this.message = 'Erreur lors de la mise à jour de l'ingrédient.';
            }
        });
        } else {
        // Mode ajout
        this.ingredientService.addIngredient(ingredient).subscribe({
            next: (newIngredient) => {
            this.ingredients.push(newIngredient);
            this.selectedIngredient = { ...DEFAULT_INGREDIENT };
            this.isEditing = false;
            //this.message = 'Ingrédient ajouté avec succès !';
            //this.resetForm();
            },
            error: (error) => {
            console.error('Erreur ajout ingrédient:', error);
            //this.message = 'Erreur lors de l'ajout de l'ingrédient.';
            }
        });
        }
    }

    /**
     * Prépare la modification d'un ingrédient en remplissant le formulaire avec
ses données.
     * @param ingredient L'ingrédient à modifier.
     */
    editIngredient(ingredient: Ingredient): void {
        this.selectedIngredient = { ...ingredient }; // Clone l'objet pour éviter
les modifications directes
        //this.ingredientToEditId = ingredient.id;
        this.isEditing = true;
    }

    /**
     * Supprime un ingrédient via l'API.
     * @param id - Identifiant de l'ingrédient à supprimer.
     */
    deleteIngredient(id: number | null): void {
        if (id === null) return;
        //if (!confirm("Voulez-vous vraiment supprimer cet ingrédient ?")) return;

        this.ingredientService.deleteIngredient(id).subscribe({
        next: () => {
```

```
            this.ingredients = this.ingredients.filter(ing => ing.id !== id);
            //this.message = 'Ingrédient supprimé avec succès.';
        },
        error: (error) => {
            console.error('Erreur suppression ingrédient:', error);
            //this.message = 'Erreur lors de la suppression de l’ingrédient.';
        }
        });
    }


    /**
     * Supprime TOUS les ingrédients via l'API
     */
    deleteAllIngredients(): void {
        this.ingredientService.deleteAllIngredients().subscribe({
          next: () => {
            this.ingredients = []; // Vider la liste après suppression
          },
          error: (error) => console.error('Erreur suppression de tous les
ingrédients:', error)
        });
      }



    /**
     * Réinitialise le formulaire après soumission.
     */
    resetForm(): void {
        this.selectedIngredient = { ...DEFAULT_INGREDIENT };
        this.isEditing = false;
    }


    // _____________________________________________________
    //
    //  METHODES POUR L'IMPORT & EXPORT DES INGREDIENTS
    // _____________________________________________________

    /**
     * Gestion de l'importation CSV.
     */
    importFromCSV(event: any): void {
    // Logique d'importation depuis le sous-composant
    }

    /**
     * Gestion de l'exportation CSV.
     */
    exportToCSV(): void {
    // Logique d'exportation depuis le sous-composant
    }
```

```
}
```

Fichier de la logique `ingredient-manager-page.component.html` :

```
<div class="container">
    <h1>Gestion des Ingrédients</h1>

    <!-- Formulaire d'ajout/modification -->
    <app-ingredient-form
        [ingredient]="selectedIngredient"
        [isEditing]="isEditing"
        (save)="saveIngredient($event)"
        (cancelEdit)="resetForm()">
    </app-ingredient-form>

    <!-- Liste des ingrédients -->
    <app-ingredient-list
        [ingredients]="ingredients"
        (edit)="editIngredient($event)"
        (delete)="deleteIngredient($event)"
        (deleteAll)="deleteAllIngredients()">
    </app-ingredient-list>

    <!-- Import / Export CSV -->
    <app-ingredient-import-export
        [ingredients]="ingredients"
        (import)="importFromCSV($event)"
        (export)="exportToCSV()"
        (importComplete)="fetchIngredients()">
    </app-ingredient-import-export>
</div>
```

5/ Ajout d'un modal pour le formulaire de l'ingrédient

Nous vous demandons de procéder aux modifications et ajouts nécessaire pour que le formulaire s'ouvre sous la forme d'un **Modal**. L'ouverture sera enclenchée par l'appui d'un bouton "Créer ingrédient" de la page `ingredient-manager-page.component.html` et/ou appuyer du bouton "Modifier" de la liste des ingrédients.