



**POLYTECHNIQUE  
MONTRÉAL**

UNIVERSITÉ  
D'INGÉNIERIE

# LOG2440 – Méthod. de développ. et conc. d'applic. Web

## Travail pratique 4

Chargés de laboratoire:

Minh-Tri Do

Gourdigou Junior Patrice Kolani

Hiver 2022

Département de génie informatique et génie logiciel

# 1 Objectifs

Le but de ce travail pratique est de vous familiariser avec le développement d'un serveur Web et la communication réseau à travers le protocole HTTP. Pour se faire, vous allez utiliser l'environnement NodeJS ainsi que la librairie Express pour créer votre propre serveur. Vous allez vous familiariser avec le module *FS (File System)* afin de gérer des fichiers ou des informations côté serveur. Vous allez également utiliser l'API *Fetch* pour construire les requêtes HTTP envoyées par votre site web.

Plus particulièrement, à partir du client, vous aurez à faire des requêtes HTTP dans le but de communiquer avec un serveur. Ce dernier vous transmettra alors les informations souhaitées afin de fournir les données demandées.

Contrairement aux travaux pratiques précédants, votre serveur Web remplacera le serveur de contenu statique (HTML, CSS, JS, images, etc.) *lite-server* et sera également un serveur dynamique qui est en mesure de répondre aux requêtes HTTP qui lui sont envoyées.

## 2 Introduction

Lors des trois derniers travaux pratiques, vous avez mis en place un site web à l'aide de fichiers HTML, CSS et JavaScript qui étaient tous récupérés grâce à un serveur HTTP statique utilisant l'outil *lite-server*. Votre site web était également capable de charger dynamiquement des recettes et de les sauvegarder localement sur l'entrepôt local *LocalStorage*.

Toutes ses étapes se déroulaient uniquement du côté client. Ceci est peu efficace, car les informations ne sont pas partagées entre plusieurs utilisateurs ou plusieurs navigateurs.

Habituellement, le contenu du client tel que les fichiers HTML, CSS, JavaScript, les images et ainsi de suite sont hébergés sur un serveur web. Lorsqu'un client accède à un URL, le navigateur fait une requête HTTP avec la méthode GET au serveur pour récupérer les fichiers correspondants. Le client peut également envoyer d'autres types de requêtes qui sont interprétées de manière différente par le serveur résultat en une action quelconque. Par exemple, une requête POST est souvent utilisée pour envoyer de l'information qui sera traitée et/ou sauvegardée par le serveur tandis qu'une requête DELETE se comporte comme une demande de suppression d'une ressource quelconque spécifiée dans l'URI ciblé par la requête.

## 3 Travail à réaliser

Premièrement, à partir du code fourni, vous aurez à faire circuler des données entre le client et votre serveur local. Lorsqu'un client accède à votre serveur web (par défaut : *localhost :5000*), ce dernier devra renvoyer la page *index.html* ou toute autre page spécifiée par le chemin de la requête. De plus, la gestion des recettes aura lieu sur le serveur et le client devra alors faire des requêtes `GET` pour récupérer une recette ou une liste des recettes (filtrée par catégorie, ingrédient ou rien). Pareillement, pour ajouter une recette, le client doit effectuer une requête `POST` dont le corps contient un objet qui représente la nouvelle recette.

Deuxièmement, une nouvelle fonctionnalité consiste à permettre à l'utilisateur du site d'envoyer un message aux administrateurs à travers la page de Contacts. Ce contact est ajouté à travers une requête `POST` et sauvegardé dans le fichier *contacts.json* dans *server/data* qui contient tous les messages déjà transmis.

Finalement, une page d'administration "cachée" est disponible seulement à travers le chemin */admin*. Cette page permet d'afficher toutes les recettes ainsi que tous les contacts qui existent sur le serveur et de les supprimer. L'affichage de l'interface vous est déjà fourni, mais vous devez implémenter la logique qui permet de supprimer une recette ou un contact spécifiques à l'aide de requêtes `DELETE`. La fonctionnalité de réinitialisation de la liste des recettes est également seulement disponible sur la page d'administration et permet de revenir à la liste initiale de recettes à travers une requête `PATCH`.

### 3.1 Syntaxe CommonJS et librairie Nodemon

La particularité de NodeJS est que la syntaxe CommonJS est utilisée pour la gestion des différents modules du logiciel au lieu d'ESM. Contrairement à ESM, CJS utilise le mot clé *require* pour importer un module et *exports* pour exporter les éléments d'un module.

Pour vous aider dans votre développement, vous utiliserez l'utilitaire *nodemon*. Nodemon permet d'exécuter du code JS, similairement à la commande *node*, mais il surveille également pour des changements dans le code source du projet et le relance automatiquement (*hot-reload*). Dans votre cas, ceci permettra d'automatiquement relancer le serveur lorsque vous modifiez son code source sans que vous ayez à le faire à la main.

Pour lancer le serveur, vous pouvez utiliser la commande `npm start` dans votre terminal avec : **npm start**. Par défaut, ce serveur est accessible à l'adresse : "http ://localhost :5000".

**Note :** le nom **localhost** est un raccourci pour l'adresse IP **127.0.0.1**. Le lancement du serveur *lite-server* affichera les adresses auxquelles le serveur est accessible.

## 3.2 Tests automatisés

Plusieurs tests vous sont déjà fournis dans le répertoire `server/tests` et vous permettront de valider votre travail et les différentes requêtes au serveur. Certains tests vérifient le bon comportement des réponses HTTP de votre serveur en simulant des requêtes à l'aide de la librairie *Supertest*. D'autres se concentrent sur la logique de gestion des ressources sur le serveur. Dans les deux cas, la librairie *Jest* est utilisée.

Pour lancer les suites de tests, vous n'avez qu'à aller à la racine du répertoire `server` avec un terminal et de lancer la commande **npm test**. Les tests d'API lanceront votre serveur et feront plusieurs requêtes avant de le fermer à la fin de la suite. Une commande pour la couverture de code (**npm run coverage**) est également à votre disposition.

Il est fortement recommandé de regarder les tests fournis et leur description avant d'écrire votre code pour vous aider avec le résultat final attendu. Vous pouvez ajouter vos propres tests si vous considérez que c'est pertinent.

## 3.3 Éléments à réaliser

La première étape consiste à implémenter les fonctionnalités nécessaires pour rendre le client et le serveur fonctionnels. Avant de débiter, lancez le serveur et accédez aux différentes routes pour s'assurer que la navigation se fait correctement.

**Rappel :** Les fonctionnalités primaires consistent à implémenter les routes pour les différentes pages du client, manipuler les données JSON à partir du serveur, ajouter des contacts sur le client et gérer les nouvelles fonctionnalités de l'administrateur.

## 3.4 Gestion des ressources statiques

Cette fonctionnalité permet de rendre le serveur capable de gérer des ressources statiques, notamment des pages HTML. Une route de gestion d'une requête GET pour chaque page du site vous est fournie. Vous devez remplacer ces routes spécifiques par une seule route `/*` qui est capable de gérer l'accès à n'importe quelle page du site Web à travers un GET.

Vous devez également gérer le cas d'une demande pour une page qui n'existe pas dans les ressources statiques du répertoire `/public/pages`. Dans un tel cas, la page `error.html` doit être retournée comme réponse au client ayant fait la requête erronée.

La route à implémenter se trouve dans le fichier `server.js`. Lors de la remise, elle doit être la seule route qui gère une requête vers la racine du serveur. Les routes fournies et spécifiques à chaque page doivent être supprimées.

## 3.5 Gestion des recettes sur le serveur

Contrairement aux TPs 2 et 3, vous devez désormais gérer l'état des recettes sur votre serveur et non seulement à travers le *LocalStorage*. Vous aurez donc à implémenter la gestion des recettes sauvegardées dans le fichier `recipes.json` dans le répertoire `server/data`.

Vous aurez à compléter certaines méthodes de la classe `RecipeJsonManager`, notamment `getRecipeByID`, `getRecipesByCategory`, `addNewRecipe` et `deleteRecipeByID`. Lisez bien les en-têtes de fonctions fournies dans le code pour mieux comprendre les entrées, sorties et fonctionnement nécessaires pour chaque méthode.

La classe `RecipeJsonManager` est utilisé par le *Router* défini dans le fichier `recipes.js`. Ce routeur implémente la logique de gestion pour les différentes requêtes en lien avec la gestion des recettes. Le routeur est accessible à travers tout chemin relatif qui commence par `/api/recipes` tel que défini dans la configuration de l'application Express dans `server.js`.

Vous aurez à compléter les gestionnaires de requêtes définies dans les sous-sections suivantes. Notez que le chemin indiqué ici est relatif du chemin du routeur `/api/recipes`.

### 3.5.1 GET /

Retourne toutes les recettes présentes dans le fichier `recipes.json` dans un objet JSON.

La réponse HTTP doit contenir un code de retour pertinent.

### 3.5.2 POST /

Vous devez vérifier la présence d'un corps sur la requête et, si présent, sauvegarder son contenu de la requête comme une nouvelle recette dans le fichier `recipes.json`.

La réponse HTTP doit contenir un code de retour pertinent dans le cas d'un corps invalide, d'une sauvegarde réussie ou d'une erreur quelconque sur le serveur.

### 3.5.3 GET /:id

Retourne la recette spécifiée par l'attribut `id` dans le chemin dans un objet JSON.

La réponse HTTP doit contenir un code de retour pertinent dans le cas d'un attribut `id` valide et dans le cas d'un attribut invalide.

### 3.5.4 DELETE /:id

Supprime la recette spécifiée par l'attribut `id` dans le chemin et met à jour le fichier.

Exemple : `DELETE /12` supprime la recette ayant l'`id` 12 si présente dans le fichier.

La réponse HTTP doit contenir un code de retour pertinent dans le cas d'une suppression réussie ou non et dans le cas d'une erreur quelconque sur le serveur.

### 3.5.5 GET /category/ :category

Retourne toutes les recettes ayant la même catégorie que le paramètre `category` dans le chemin à partir du fichier `recipes.json` dans un objet JSON.

Exemple : `GET /category/keto` retourne toutes les recettes ayant la catégorie `keto`.

La réponse HTTP doit contenir un code de retour pertinent.

### 3.5.6 GET /ingredient/ :ingredient[?matchExact]

Retourne toutes les recettes ayant au moins un ingrédient qui correspond au paramètre `ingredient` dans le chemin à partir du fichier `recipes.json` dans un objet JSON. Cette requête peut avoir un paramètre de *query* optionnel : `matchExact`. Si ce paramètre est présent, la recherche par ingrédient se fait avec une comparaison exacte avec la chaîne de caractères `ingredient`, sinon la recherche se fait en cherchant seulement si la chaîne `ingredient` est une sous-chaîne d'un des ingrédients de la recette.

Exemple :

- `GET /ingredient/oignon?matchExact=true` retourne toutes les recettes qui ont l'ingrédient "oignon".
- `GET /ingredient/oigon` retourne toutes les recettes qui ont un ingrédient qui contient la chaîne "oignon" ("oignon vert" étant un ingrédient valide).

La réponse HTTP doit contenir un code de retour pertinent.

### 3.5.7 PATCH /admin/reset

Cette requête réinitialise la liste des recettes avec les recettes par défaut. Le fichier `recipes.json` est remplacé par `default.json`. Aucune valeur n'est retournée.

La réponse contient un code de retour 200 en cas de succès et 500 dans le cas d'une erreur.

Cette requête est déjà implémentée pour vous.

## 3.6 Gestion des contacts sur le serveur

Pour ce travail pratique, vous aurez également à gérer les contacts sauvegardés dans le fichier `contacts.json` dans le répertoire `server/data`. Les utilisateurs peuvent soumettre un message de contact avec un formulaire ayant les champs suivants : `name`, `email` et `message`. Les contacts peuvent être lus et supprimés par les administrateurs du site.

Vous aurez à compléter deux méthodes de la classe `ContactJsonManager` : `addNewContact` et `deleteContactById`. Lisez bien les en-têtes de fonctions fournies dans le code pour mieux comprendre les entrées, sorties et fonctionnement nécessaires pour chaque méthode.

La classe `ContactJsonManager` est utilisée par le *Router* défini dans le fichier `contact.js`. Ce routeur implémente la logique de gestion pour les différentes requêtes en lien avec la gestion des contacts. Le routeur est accessible à travers tout chemin relatif qui commence par `/api/contacts` tel que défini dans la configuration de l'application Express dans `server.js`.

Vous aurez à compléter les gestionnaires de contacts du routeur définies dans les sous-sections suivantes. Notez que le chemin indiqué ici est relatif du chemin du routeur `/api/contacts`.

### 3.6.1 GET /

Gestion d'une requête `GET` pour le chemin `/`.

Retourne tous les contacts présents dans le fichier `contacts.json` dans un objet JSON.

La réponse HTTP doit contenir un code de retour pertinent.

### 3.6.2 POST /

Vous devez vérifier la présence d'un corps sur le contact et, si présent, sauvegarder son contenu de la requête comme un nouveau contact dans le fichier `contacts.json`.

La réponse HTTP doit contenir un code de retour pertinent dans le cas d'un corps invalide, d'une sauvegarde réussie ou d'une erreur quelconque sur le serveur.

L'ajout d'un nouveau contact se fait par un formulaire grâce à l'attribut `method` de la balise `<form>`. Un ajout de contact réussi doit rediriger l'utilisateur vers le chemin `/contact` après la sauvegarde. Cette redirection doit être gérée par le serveur.

### 3.6.3 DELETE / :id

Supprime le contact spécifié par l'attribut `id` dans le chemin et met à jour le fichier.

Exemple : `DELETE /2` supprime le contact ayant l'`id` 2 si présent dans le fichier.

La réponse HTTP doit contenir un code de retour pertinent dans le cas d'une suppression réussie ou non et dans le cas d'une erreur quelconque sur le serveur.

## 3.7 Gestion des recettes sur le client

Puisque la gestion des recettes est maintenant faite par le serveur, le site web doit communiquer avec en utilisant le protocole HTTP pour gérer des recettes. Le site web doit pouvoir effectuer les mêmes opérations que les TPs 2 et 3, mais sans utiliser *LocalStorage*.

La classe `StorageManager` utilisée par `RecipeManager` a été remplacée par la classe `HTTPManager` qui est responsable de la communication avec le serveur. Vous devez compléter plusieurs fonctions de cette classe afin d'implémenter la communication avec votre serveur.

Les fonctions à compléter contiennent le décorateur **@todo** dans leur en-tête de commentaires. Lisez bien les en-têtes de fonctions fournies dans le code pour mieux comprendre les entrées, sorties et fonctionnement nécessaires pour chaque méthode. Certaines méthodes vous sont déjà fournies, notamment la réinitialisation des recettes sur le serveur.

Pour vous aider à démarrer, un objet `HTTPInterface` vous est fourni dans le fichier `http_manager.js`. Cet objet implémente plusieurs fonctions utilitaires qui utilisent l'API `Fetch` pour envoyer des requêtes HTTP. Certaines méthodes qui envoient des paramètres contiennent l'en-tête *Content-Type : application/json* qui n'est pas toujours nécessaire. Vous devez enlever cet en-tête pour les requêtes qui n'en ont pas besoin.

## 3.8 Page d'administration du site

Une nouvelle fonctionnalité à ajouter pour le site web dans ce travail est une page d'administration du site qui permet d'afficher et supprimer les différentes recettes et contacts disponibles sur le serveur. Il est aussi possible de réinitialiser les recettes sur le serveur pour les recettes par défaut (anciennement disponible dans le formulaire d'ajout de recettes).

Cette page est accessible sur l'URL relative `/admin` (`localhost :5000/admin`). Contrairement aux autres pages, il ne doit pas avoir un lien dans le menu de navigation vers cette page dans le reste du site.



La page contient 3 boutons : "Afficher Recettes", "Afficher Contacts" et "Réinitialiser Recettes". Les 2 premiers boutons permettent d'afficher les recettes ou les contacts dans une liste de la page. Les éléments affichés sont récupérés à travers des requêtes `GET` au serveur.

Chaque élément de la liste contient un bouton de suppression, affiché seulement lorsque l'utilisateur met sa souris par-dessus. Un click sur ce bouton envoie une requête `DELETE` avec le paramètre `id` de l'élément à supprimer. La liste doit être mise à jour après la suppression.

Le dernier bouton permet de réinitialiser la liste des recettes à sa valeur initiale. Un click sur le bouton envoie une requête `PATCH` vers le serveur. La liste des recettes est mise à jour après la réinitialisation.

Le code HTML et une grande partie du code vous sont déjà fournis. Vous devez implémenter les fonctions `deleteRecipe`, `deleteContact` et `resetRecipes` de `AdminManager` qui communiquent avec la classe `HTTPManager` pour rendre la communication avec le serveur fonctionnelle. Lisez bien les en-têtes de fonctions fournies dans le code pour mieux comprendre les entrées, sorties et fonctionnement nécessaires pour chaque méthode.

## **Conseils pour la réalisation du travail pratique**

---

1. Séparez bien la logique des requêtes de la logique de gestion des fichiers.
  2. Lisez bien les tests fournis pour vous aider avec les fonctionnalités demandées.
  3. Consultez les exemples de serveurs NodeJS/Express sur le [GitHub du cours](#).
  4. Exécutez les tests fournis souvent afin de valider votre code.
  5. Respectez la bonne sémantique pour les codes de retour.
  6. Assurez-vous de gérer les cas d'erreur pour les différentes requêtes HTTP.
  7. Respectez la convention de codage établie par *ESLint*. Utilisez la commande `npm run lint` pour valider cet aspect.
-

## 4 Remise

Voici les consignes à suivre pour la remise de ce travail pratique :

1. Le nom de votre entrepôt Git doit avoir le nom suivant : **tp4\_matricule1\_matricule2** avec les matricules des 2 membres de l'équipe.
2. Vous devez remettre votre code (*push*) sur la branche **master** de votre dépôt git. (pénalité de 5% si non respecté)
3. Le travail pratique doit être remis avant **23h55**, le **31 mars**.

**Aucun retard** ne sera accepté pour la remise. En cas de retard, la note sera de **0**.

Le navigateur web **Google Chrome** sera utilisé pour tester votre site web. Votre serveur doit être déployable avec la commande **npm start**.

## 5 Évaluation

Vous serez évalués sur le respect des exigences fonctionnelles de l'énoncé, ainsi que sur la qualité de votre code JS et sa structure. Le barème de correction est le suivant :

Exigences	Points
Implémentation du serveur web et site web	
Implémentation de la route dans <code>server.js</code>	1
Implémentation du <i>Routeur</i> <code>recipes.js</code>	3
Implémentation du <i>Routeur</i> <code>contacts.js</code>	2
Implémentation de <code>RecipeJsonManager</code>	3
Implémentation de <code>ContactJsonManager</code>	2
Implémentation de <code>HTTPManager</code> et <code>AdminManager</code>	5
Qualité du code	
Structure du code	2
Qualité et clarté du code	2
Total	20

L'évaluation se fera à partir de la page d'accueil du site, soit `index.html`. À partir de cette page, le correcteur devrait être capable de consulter toutes les autres pages de votre site web et interagir avec les différents éléments du site.

L'évaluations des tests se fera à partir des tests unitaires dans le projet `server`. Tous les tests fournis doivent obligatoirement passer. Tous les tests unitaires additionnels écrits par vous (si lieu) doivent obligatoirement passer.

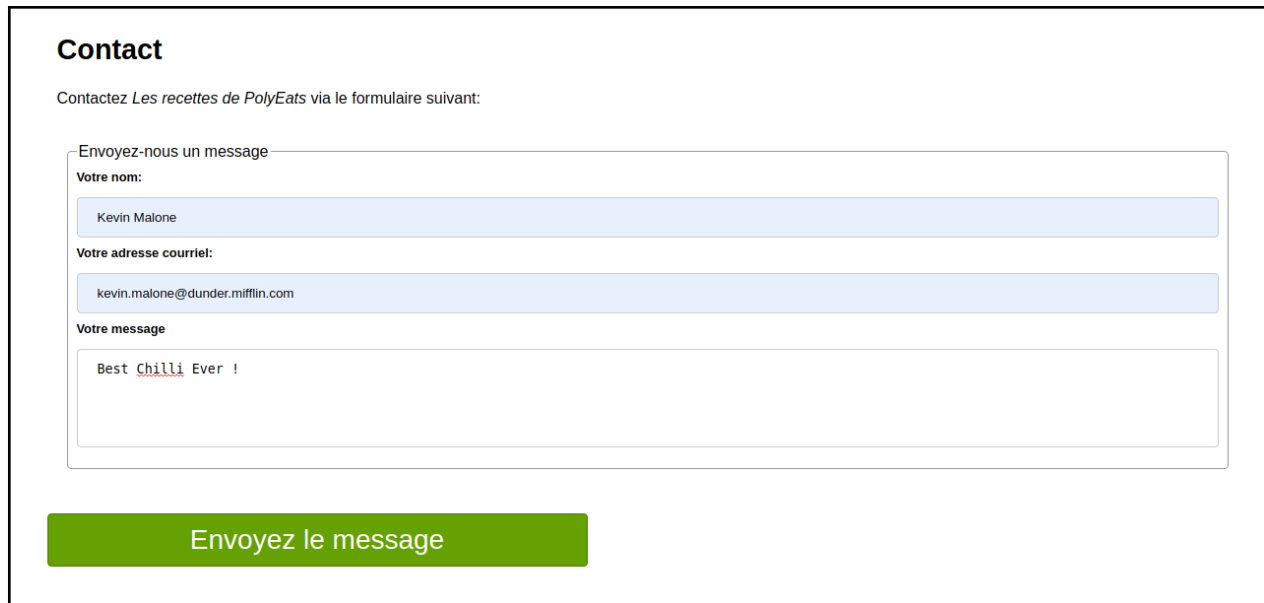
Le code doit respecter les règles établies par *ESLint*. La commande *lint* ne doit pas indiquer des erreurs dans la console après son exécution.

Ce travail pratique a une pondération de **8%** sur la note du cours.

## 6 Questions

Si vous avez des interrogations concernant ce travail pratique, vous pouvez poser vos questions sur MS Teams ou contacter votre chargé de laboratoire.

## Annexe



**Contact**

Contactez *Les recettes de PolyEats* via le formulaire suivant:

Envoyez-nous un message

Votre nom:

Kevin Malone

Votre adresse courriel:

kevin.malone@dunder.mifflin.com

Votre message

Best Chilli Ever !

Envoyez le message

FIGURE 1 – Formulaire pour un nouveau contact

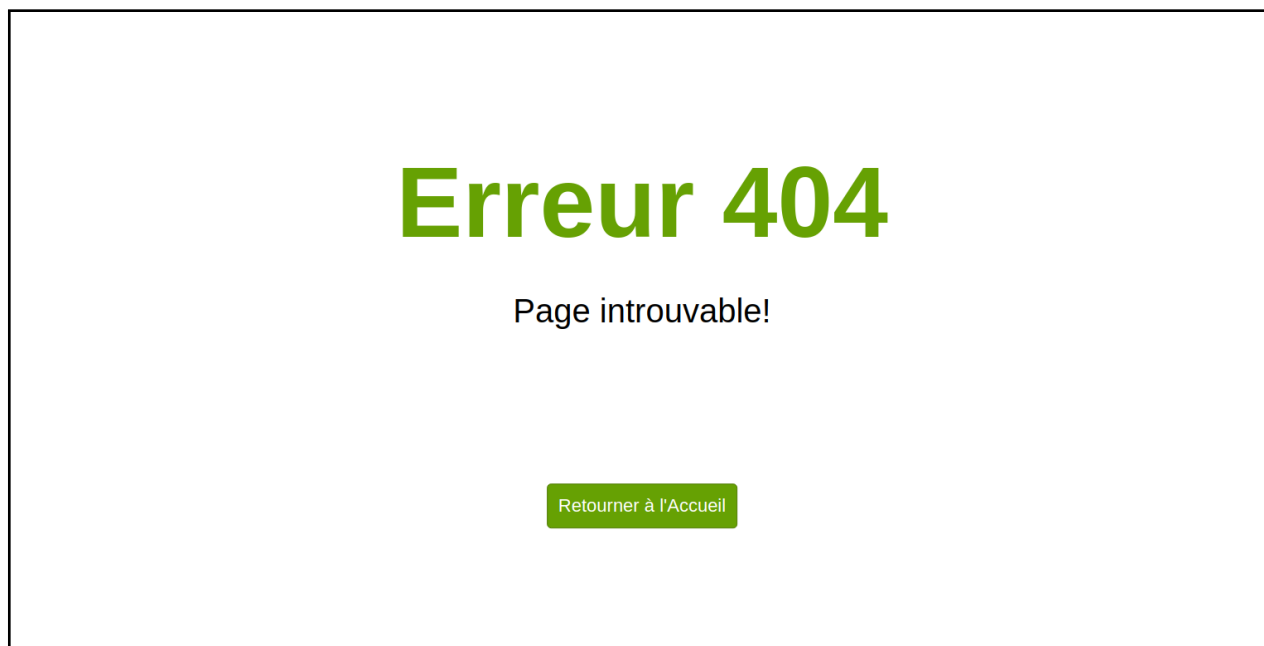


FIGURE 2 – Page d'erreur pour une page invalide

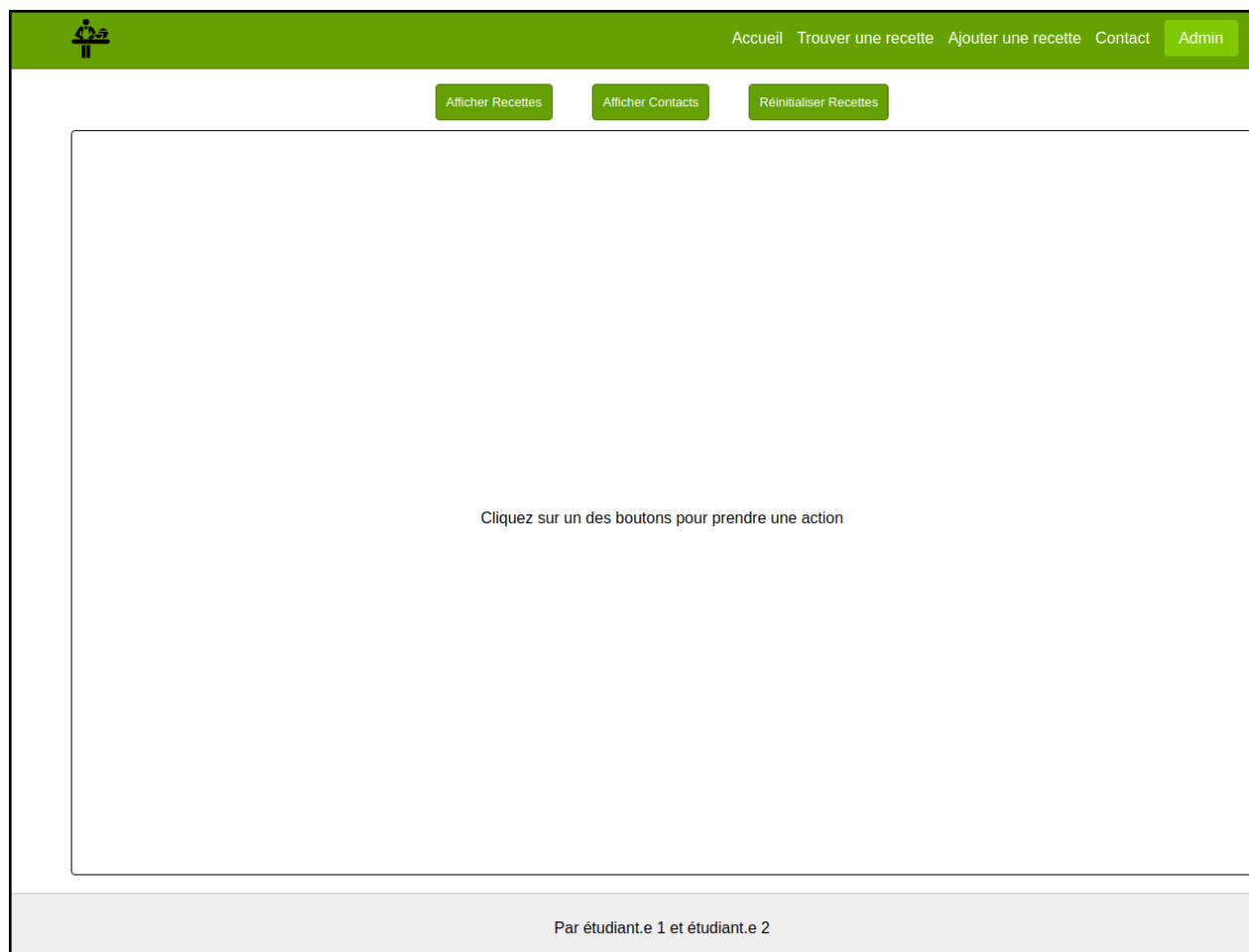


FIGURE 3 – Page administrative par défaut sur `http://localhost:5000/admin`

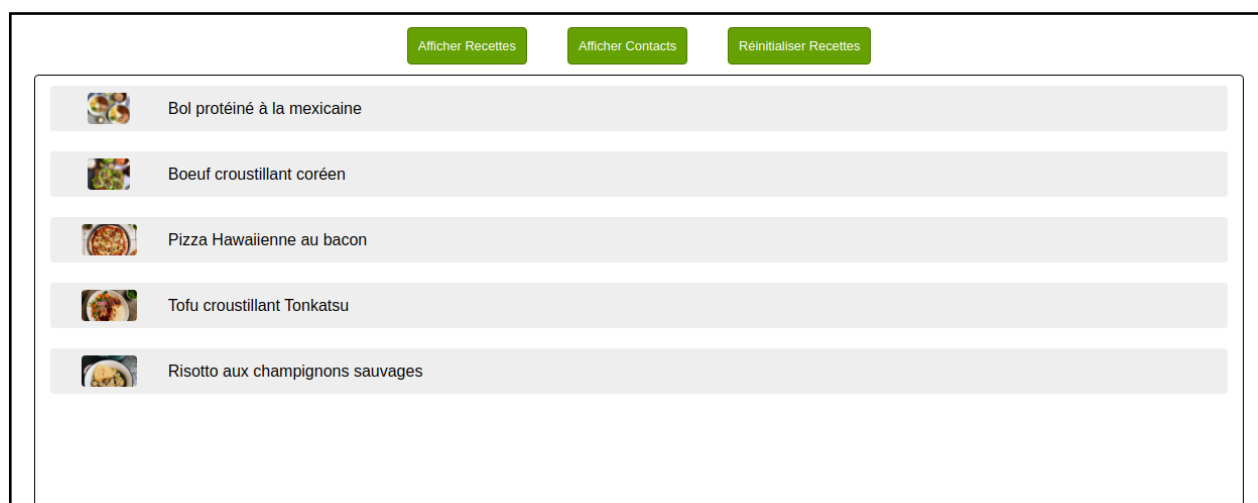


FIGURE 4 – Affichage des recettes sur la page d'administration

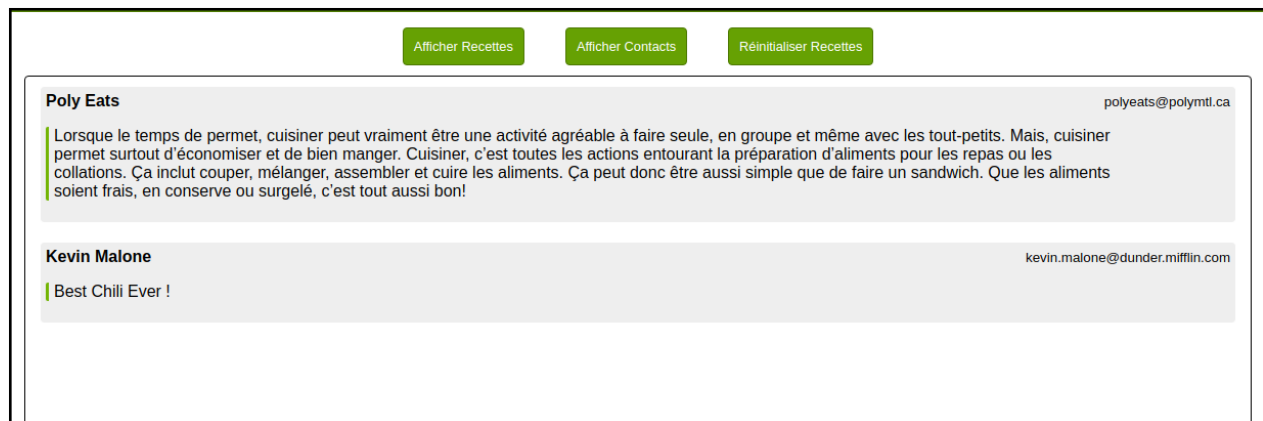


FIGURE 5 – Affichage des messages des contacts sur la page d'administration