

I. Le moteur de Template EJS

Quand on crée une application Node, un système d'affichage autre que le format json est parfois nécessaire.

`res.send()` permet de renvoyer du code HTML, par contre si le message est une page html complexe, ce sera vite fastidieux et illisible.

Comme dans d'autres framework, il est possible avec express d'utiliser un moteur de templates

A. Qu'est-ce qu'un moteur de template

Un moteur de template sert à :

- Séparer la logique et la présentation :
 - Le moteur de template permet de séparer le code de la logique (JavaScript) du code de la présentation (HTML).
 - Cela facilite la maintenance et la modification du code.
- Générer des pages dynamiques :
 - Le moteur de template permet d'injecter des données dynamiques dans les pages HTML.
 - Cela permet de générer des pages web personnalisées en fonction des données fournies par l'application.
- Réutiliser le code :
 - Les moteurs de template offrent la possibilité de créer des layouts, des partials (blocs réutilisables) et des helpers.
 - Cela permet de réduire la duplication du code et d'améliorer la cohérence visuelle de l'application.
- Améliorer l'abstraction et la portabilité :
 - L'utilisation d'un moteur de template permet d'abstraire la génération de pages HTML.
 - Cela rend l'application plus portable et facile à déployer sur différents environnements.

Les principaux moteurs de template populaires dans Express.js sont Handlebars, Pug (anciennement Jade), EJS et Mustache. Chacun a ses propres syntaxes et fonctionnalités, mais ils partagent tous l'objectif de faciliter la création de pages web dynamiques dans une application Express.

B. [EJS](#) (Embedded JavaScript template)

EJS est un moteur de template simple et léger qui permet d'intégrer facilement du code JavaScript dans des fichiers HTML.



1. Principales caractéristiques d'EJS :

- Syntaxe minimale et intuitive
 - Possibilité d'inclure du code JavaScript directement dans les fichiers HTML
 - Génération dynamique de pages web
- Prise en charge des layouts et des partiels pour la réutilisation du code
 - Facilité d'utilisation et d'apprentissage
 - Bonne documentation et communauté active

Avec EJS on va pouvoir appeler des fichiers HTML complets avec ou sans passage de paramètres depuis notre code Node.js, à la suite de l'exécution d'une requête. Écrire du HTML brut renvoyé par la fonction *res.send* devient vite fastidieux.

EJS nous simplifie aussi l'inclusion d'en-têtes ou pieds-de-page génériques (layout).

C. Installation

- Depuis le terminal, si ce n'est pas fait, positionnez-vous dans votre dossier backend
- Installez ejs

```
npm i ejs
```

- Si vous ne l'aviez pas fait au TP 7, il est encore temps d'ajouter la ligne suivante dans la section « scripts » du package.json pour préciser à npm la commande à lancer en mode dev
`"dev": "nodemon app.js"`

Maintenant nous pouvons lancer notre serveur avec la commande suivante

```
npm run dev
```

Pour pouvoir utiliser ejs, il faudra le définir comme moteur de template

- Dans votre fichier "app.js", placez le code suivant avant la définition des routes

```
app.set("view engine", "ejs"); //on utilise le moteur de template ejs
```

D. Les vues

- Pour afficher les vues, créez 2 nouvelles routes, au début de votre routeur

GET /about → about

GET / → index

L'affichage du template se fera avec la fonction res.render()

```
// page d'accueil
router.get("/", function (req, res) {
  res.render("index");
});
```

```
// page à propos
router.get("/about", function (req, res) {
  res.render("about");
});
```

Par défaut ejs, va chercher les pages dans le dossier **/views**, ceci peut être modifié par l'instruction
`app.set("views", "./views");` //on définit le dossier des vues

Les modèles utilisés par ejs sont des fichiers avec l'extension « **.ejs** »

- Créez le dossier **views** et créez à l'intérieur 2 fichiers index.ejs et about.ejs

Exemple de contenu HTML

○ about.ejs

```
<!DOCTYPE html>
<html lang="fr"></html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>À propos de KazABurger</title>
</head>
<body class="bg-gray-100 text-gray-800">
  <header class="bg-blue-600 text-white p-4">
    <h1 class="text-3xl font-bold">À propos de <a href="/">KazABurger</a></h1>
  </header>
  <main class="container mx-auto p-4">
    <section class="bg-white p-6 rounded-lg shadow-md">
      <h2 class="text-2xl font-semibold mb-4">Qui sommes-nous</h2>
      <p class="mb-4">Nous sommes KazABurger, une équipe de passionnés dédiée à fournir les meilleurs burgers possibles. Notre mission est de livrer des produits de haute qualité qui répondent aux besoins de nos clients.</p>
      <h2 class="text-2xl font-semibold mb-4">Nos valeurs</h2>
```

```

<ul class="list-disc list-inside mb-4">
  <li>Intégrité</li>
  <li>Innovation</li>
  <li>Satisfaction client</li>
  <li>Travail d'équipe</li>
</ul>
<h2 class="text-2xl font-semibold mb-4">Notre histoire</h2>
<p>Fondée en 2025, KazABurger est passée d'une petite startup à une entreprise leader dans le secteur des burgers. Notre parcours a été marqué par une croissance continue et un engagement envers l'excellence.</p>
</section>
</main>
<footer class="bg-blue-600 text-white p-4 mt-4 text-center">
  <p>&copy; 2025 KazABurger. Tous droits réservés.</p>
</footer>
</body>
</html>

```

○ index.ejs

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>KazaBurger - Accueil</title>
</head>
<body class="bg-gray-100">
  <header class="bg-blue-600 text-white p-4">
    <h1 class="text-3xl font-bold">Bienvenue chez KazaBurger</h1>
  </header>
  <main class="container mx-auto mt-8">
    <section class="bg-white p-6 rounded-lg shadow-lg">
      <h2 class="text-2xl font-semibold mb-4">À propos de nous</h2>
      <p class="text-gray-700">KazaBurger est votre destination pour les meilleurs burgers de Limoges. Nous utilisons des ingrédients frais et de qualité pour vous offrir une expérience culinaire inoubliable.</p>
      <p class="mt-4"><a href="/about" class="text-blue-600 underline">En savoir plus sur nous</a></p>
    </section>
    <section class="bg-white p-6 rounded-lg shadow-lg mt-6">
      <h2 class="text-2xl font-semibold mb-4">Nos Burgers</h2>
      <ul class="list-disc list-inside text-gray-700">
        <li>Classic Burger</li>
        <li>Cheese Burger</li>
        <li>Bacon Burger</li>
        <li>Veggie Burger</li>
      </ul>
    </section>
    <section class="bg-white p-6 rounded-lg shadow-lg mt-6">
      <h2 class="text-2xl font-semibold mb-4">Contactez-nous</h2>
      <p class="text-gray-700">Pour toute question ou commande, veuillez nous contacter au <a href="tel:+1234567890" class="text-red-600">+123 456 7890</a>.</p>
    </section>
  </main>
  <footer class="bg-blue-600 text-white p-4 mt-8 text-center">
    <p>&copy; 2025 KazaBurger. Tous droits réservés.</p>
  </footer>
</body>
</html>

```

- Dans le dossier racine du backend, créez 2 dossiers "src" et "public"
- Dans le dossier public, créez 2 sous dossiers "css" et "images"
- Placez dans images, le logo de l'application, cf TP6
- Installez les modules tailwind et @tailwind/cli
- Dans src, ajoutez le fichier index.css du TP6
- Via le terminal, générez le fichier css de sortie dans public/css/styles.css
- Modifiez les pages pour prendre en compte le fichier styles.css
- Afin qu'express serve les fichiers statiques, ajoutez l'instruction suivante, dans le serveur

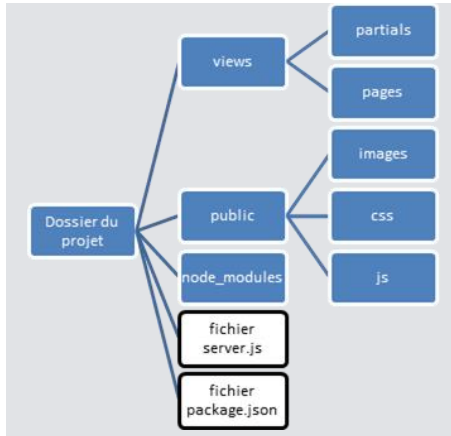
```
app.use(express.static("public")); //on définit le dossier des fichiers statiques
```

- Consultez le site depuis le navigateur <http://localhost:4501/>

E. Utilisation de « Templates »

EJS nous permet aussi d'exploiter des fichiers HTML génériques (comme les en-têtes, pieds de page) identiques à toutes nos pages, pour éviter les redondances et les erreurs de copier-coller.

Afin de profiter de cette modularité, il est conseillé de réorganiser les dossiers d'un projet (Node.js + express.js + EJS) comme suit :



- Dans « public », on mettra les fichiers accessoires qui seront envoyés au client : les css, les images
- Dans « views », on organisera nos templates (fichier .ejs), en 2 catégories :
 - Le dossier « partials » contiendra les fichiers génériques : les portions communes à toutes nos pages (en-tête, pied de page, menu, ...),
 - et le dossier « pages » contiendra les fichiers construisant les différentes pages (page d'accueil, login, ...)

1. Les partials

- Dans « views/partial », créez les fichiers suivants :
 - « head.ejs » qui servira de patron pour tous les débuts de nos pages :

```

<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>KazaBurger</title>
<link href="css/styles.css" rel="stylesheet" />
</head>

```

- « footer.ejs » qui servira à mutualiser le pied de page

```

<footer class="bg-blue-600 text-white p-4 mt-8 text-center">
  <p>&copy; 2025 KazaBurger. Tous droits réservés.</p>
</footer>

```

- Modifier les pages index et about, pour utiliser les vues partielles

```

<!DOCTYPE html>
<html lang="fr"></html>
<%- include ('../partials/head.ejs') %>
<body class="bg-gray-100 text-gray-800">
  <header class="bg-blue-600 text-white p-4">
    <h1 class="text-3xl font-bold">À propos de <a href="/">KazABurger</a></h1>
  </header>
  <main class="container mx-auto p-4">
    <section class="bg-white p-6 rounded-lg shadow-md">
      <h2 class="text-2xl font-semibold mb-4">Qui sommes-nous</h2>
      <p class="mb-4">Nous sommes KazABurger, une équipe de passionnés dédiée à fournir les
      <h2 class="text-2xl font-semibold mb-4">Nos valeurs</h2>
      <ul class="list-disc list-inside mb-4">
        <li>Intégrité</li>
        <li>Innovation</li>
        <li>Satisfaction client</li>
        <li>Travail d'équipe</li>
      </ul>
      <h2 class="text-2xl font-semibold mb-4">Notre histoire</h2>
      <p>Fondée en 2025, KazABurger est passée d'une petite startup à une entreprise leader
    </section>
  </main>
  <%- include ('../partials/footer.ejs') %>
</body>
</html>

```

NB:

- `<% xxxx %>` permet d'exécuter du code JS (js, fonction, composant)
- `<%= maVariableJS %>` permet d'afficher le contenu de la variable
- `<%-` permet d'inclure du code html. Ici celui de la vue partielle, récupéré avec `include`

Référez-vous à la [doc](#) pour les autres tags

En observant la partie `<header>` des 2 pages, on pourrait déjà penser à mutualiser la mise en forme et utiliser une variable pour gérer le texte

```

<header class="bg-blue-600 text-white p-4">
  <h1 class="text-3xl font-bold">À propos de <a href="/">KazABurger</a></h1>
</header>

```

- Créez la vue partielle "header"

```

<header class="bg-blue-600 text-white p-4">
  <h1 class="text-3xl font-bold"><%= locals?.title || "Accueil" %></h1>
</header>

```

Ici on vérifie l'existence de la variable locale "title" pour l'afficher. Dans le cas contraire, on affiche "Accueil"

Code équivalent :

```

<h1 class="text-3xl font-bold">
  <% if(typeof locals.title !== "undefined") {%>
    <%= title %>
  <% }else{ %>
    Accueil
  <% } %>
</h1>

```

- La page appelante doit passer le contenu de la variable `title` à la vue partielle

```

<%- include ('../partials/header.ejs', {title: 'À propos de <a href="/">KazABurger</a>'}) %>

```

Nb : EJS étant un interpréteur, si on ne prend pas garde, du code JS peut être envoyé à la [page](#). Il convient de bien contrôler les données avant de les fournir aux locals

2. Les layouts

L'utilisation d'include est pratique, mais cette méthode est fastidieuse lorsque l'on souhaite inclure systématiquement des parties comme le header et le footer sur toutes les pages d'un site.

Le package [express-ejs-layouts](#) permet de pallier ce problème, en faisant en sorte que les vues utilisent un(des) layout(s) spécifique(s) sans avoir à ajouter include dans chaque vue.

- Installez le module

```
npm i express-ejs-layouts
```

Pour l'utiliser dans notre application, nous définirons le layout souhaité dans un fichier, par exemple nommé layout.ejs, que nous mettrons dans un sous-dossier layouts du dossier views.

- Dans le dossier « views », créez un sous dossier « layouts »
- Créez-y un fichier main.ejs qui contiendra notre code commun

```
<!DOCTYPE html>
<html lang="fr">
  <%- include ('../partials/head.ejs') %>
  <body>
    <div id="root">
      <div class="container">
        <%- include ('../partials/header.ejs') %>
        <main class="mx-auto p-4 flex flex-col flex-grow justify-start">
          <%- body %>
        </main>
        <%- include ('../partials/footer.ejs') %>
      </div>
    </div>
  </body>
</html>
```

NB : <%- body %> permet de préciser l'emplacement du contenu spécifique

- Modifiez les vues partielles pour ne conserver que le code spécifique, les sections
- Modifier le serveur pour utiliser express-js-layout, après les instructions ejs

```
const expressLayouts = require("express-ejs-layouts");//on importe ejs-layouts
```

```
app.use(expressLayouts); //on utilise ejs-layouts
```

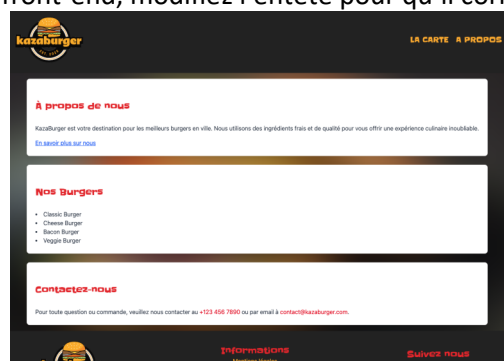
```
app.set("layout", "layouts/main"); //on définit le layout par défaut
```

A ce stade la page on, a un souci sur le header.

- Modifiez le routeur pour envoyer la variable title aux vues

```
router.get("/", (req, res) => {
  res.render("pages/index", {title:"Bienvenue sur <a href='/'>Kaz 'A burger</a>"});
});
```

- En vous inspirant du front-end, modifiez l'entête pour qu'il corresponde au visuel



NB : L'attribut className de react doit être remis en class

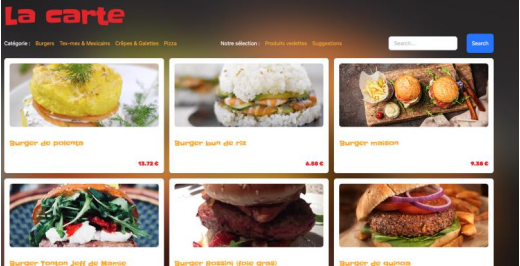

Le lien sur le logo renverra vers la page « / »

Le lien « A propos » renverra vers la page « /about »

II. Sans filet

Afin d'afficher les différentes informations en html, via l'adresse <http://localhost:4501/>
Vous devrez créer de nouvelles routes , en utilisant la fonction [render](#) de l'objet response,

NB : Le lien « La carte » renverra vers la page /html/product

Actions	Verbe	URL	Retour
Liste des produits	GET	/html/product /html/product/?family=xxx	<p>Liste de 9 produits</p>  <p>Au niveau de la barre de recherche :</p> <ul style="list-style-type: none"> Catégories : Affichez la liste des catégories de produit (family) pour permettre un filtrage Notre sélection : Afficher 2 liens de sélection <ul style="list-style-type: none"> Produits vedettes : /html/featured Nos suggestions : /html/suggest La zone de recherche permettra de filtrer les produits sur leur titre <p>Dans la liste :</p> <ul style="list-style-type: none"> Le clic sur une vignette de produit affichera la fiche détaillée du produit page /html/product/xxxx
Détail d'un produit	GET	/html/product/xxx	<p>Fichée détaillée d'un produit + les témoignages liés</p> 
Liste des produits en vedette	GET	html/product/?featured=true	Idem liste des produits
Liste des suggestions (démo)	GET	html/product/?suggest=true	Idem liste des produits
Liste des témoignages (groupes 4b/5a)*	GET	/html/testimony	<p>Tableau des 10 derniers témoignages</p> <p>La sélection d'une ligne affichera la page d'édition du témoignage</p>
Détail d'un témoignage	GET	/html/ testimony /xxx	Formulaire de modification/suppression
Liste des suggestions (groupes 4a/5b)*	GET	/html/suggest	<p>Tableau des suggestions</p> <p>La sélection d'une ligne affichera la page de modification de la suggestion</p>
Détail d'une suggestion	GET	/html/ suggest /xxx	Formulaire de modification/suppression
Liste des utilisateurs	GET	/html/user	<p>Tableau de 10 utilisateurs</p> <p>La sélection d'une ligne affichera la page de modification de l'utilisateur</p>
Détail d'un utilisateur	GET	/html/user/xxx	Formulaire de modification/suppression

* : Attention à l'inversion des groupes par rapport au TP7 .

Si au TP 7 , vous avez codé les routes pour les suggestions, vous pouvez créer des témoignages via l'api <https://kazaburger.net/api/sec/testimony>

Si vous avez développé les routes pour les témoignages vous créerez vos suggestions à partir de l'api <https://kazaburger.net/api/sec/suggest>

Struture de l'entité family

```
_id: ObjectId('67857189667f58558151f287')
name : "Burgers"
picture : "https://cdn.stoneline.de/media/b0/c4/37/1721744157/Smash-Burger.png?ts..."
alias : "burger"
```

NB : L'attribut à utiliser pour le filtrage est l'alias

Conseil : Vous pouvez créer dans les vues, par exemple, un dossier « product » qui contiendra les 2 vues (liste , détail)

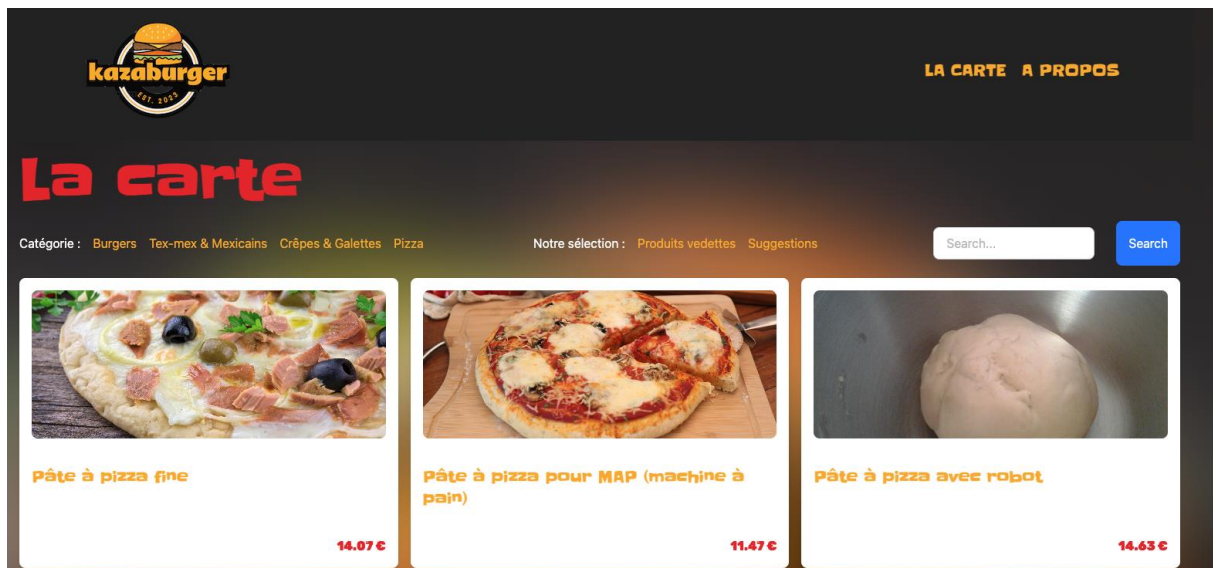


Figure 1: Liste des produits

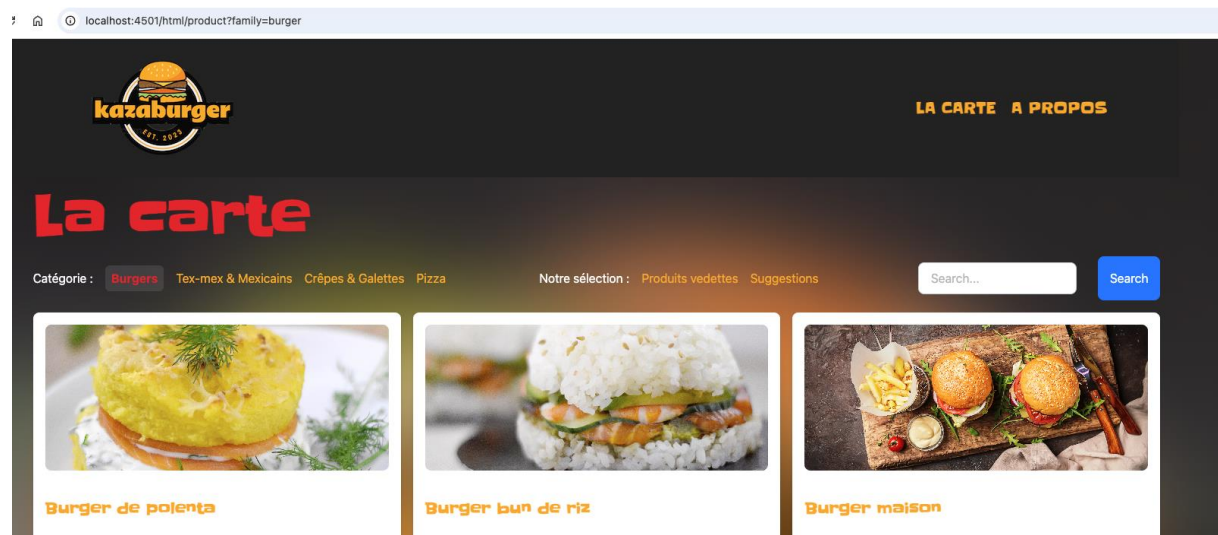


Figure 2: Filtrage des produits par famille

NB : Remarquez le changement de style sur le lien actif, ici« Burgers »

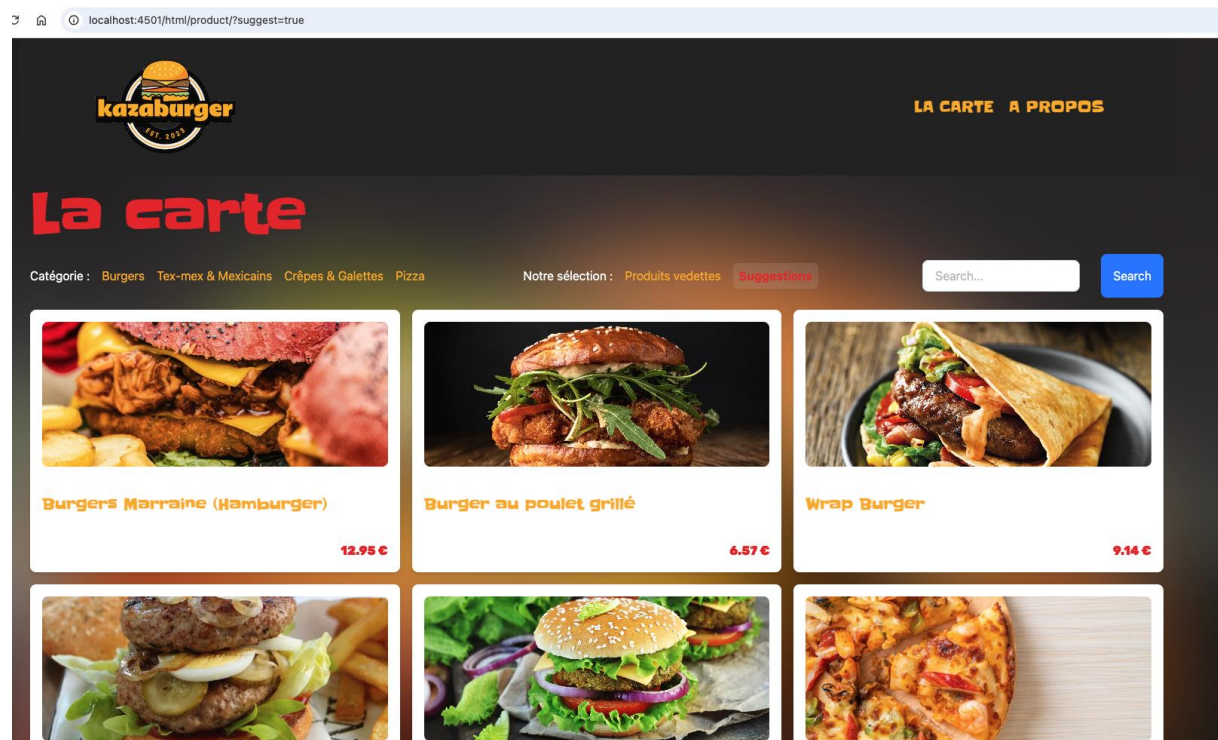


Figure 3: Affichage des suggestions

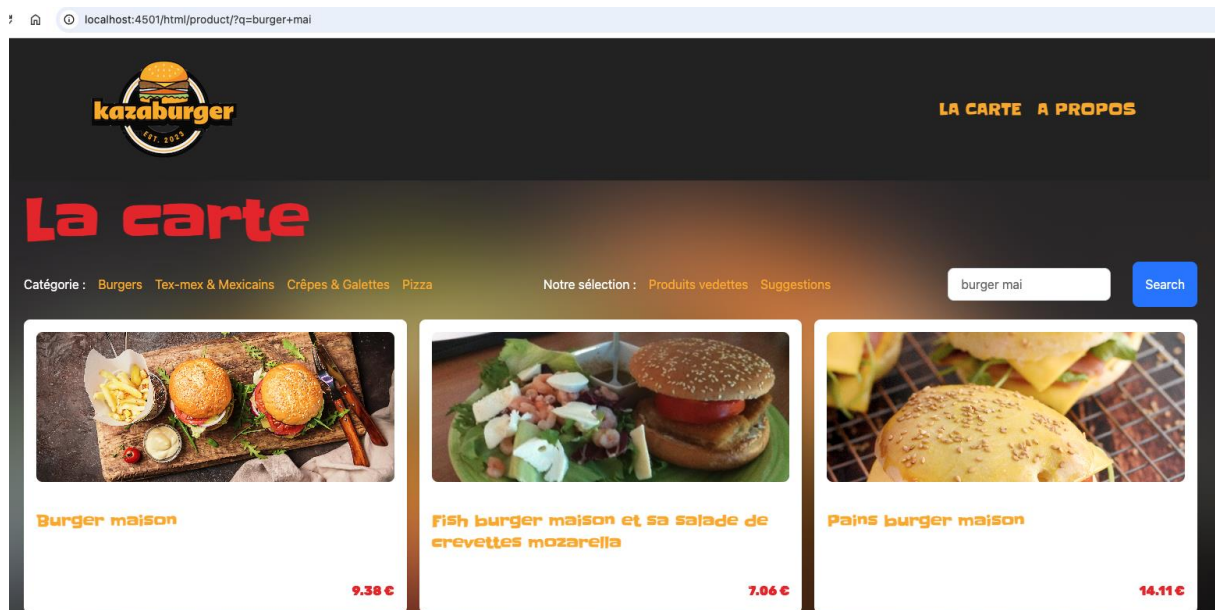


Figure 4: Recherche d'un produit par son titre

NB : Remarquez la zone de texte, conserve la valeur recherchée

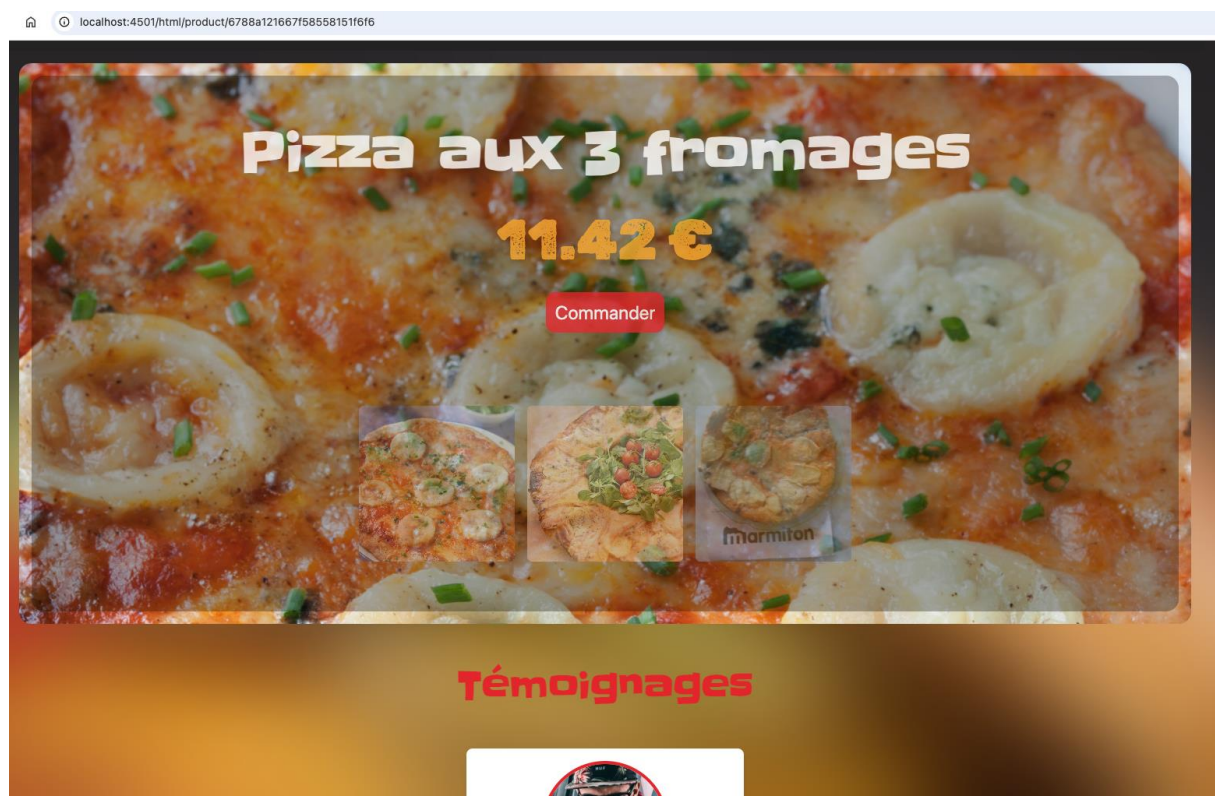


Figure 5 : Vue détaillée d'un produit 1

NB : Remarquez la liste des images en vignette et la zone des témoignages (s'il y en a)

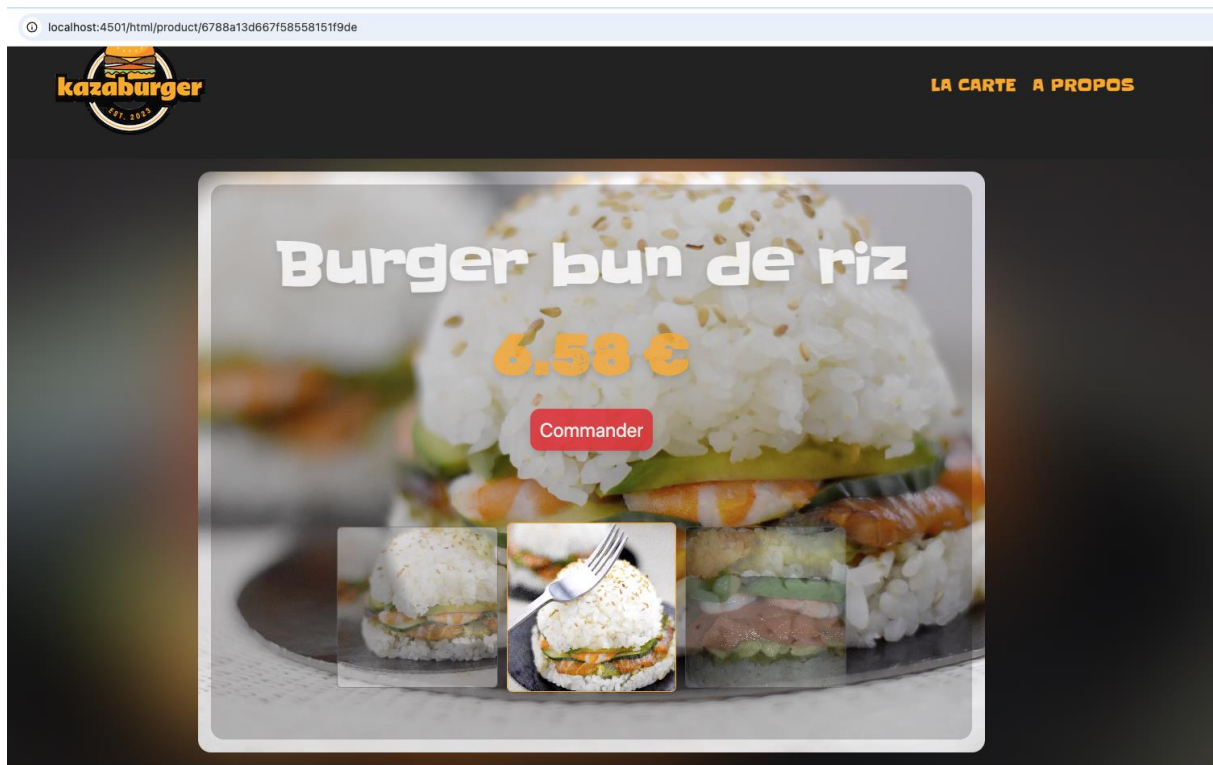


Figure 6 : Fiche détaillée d'un produit – 2

NB : Remarquez le changement de style au survol d'une image + l'absence de témoignages

Figure 7: Fenêtre d'authentification

Figure 8: Fenêtre d'inscription

La gestion de l'état connecté ou pas , doit être conservé et transmises aux pages. Vous pouvez recourir aux sessions en installant le module express-session

npm i express-session

```
app.use(
  session({
    secret: process.env.SESSION_SECRET, //on définit la clé secrète
    resave: false,
```

```

    saveUninitialized: true,
    cookie: { secure: false },
  })
};

```

A partir de là vous pouvez récupérer et modifier la session dans les fonctions middleware à partir de `req.session`

Exemple d'utilisation, dans la route d'authentification de l'utilisateur

```

router.post("/", (req, res, next) => {
  const { login, password, email } = req.body;

  const id = login || email;
  if (!id || !password) {
    req.message = "Login ou mot de passe absent";
    return res.redirect("/login");
  }

  return (
    id &&
    password &&
    getData("/user/?login=" + id, req.session, "GET")
      .then((user) => {
        if (user.length)
          bcrypt.compare(password, user[0].password, function (err, result) {
            if (result) {
              req.session.user = user[0];
              res.redirect("/");
            } else res.redirect("/login");
          });
        else throw new Error("Bad user or password");
      })
      .catch((err) => next(err))
  );
});

```

Nb : Remarquez, l'envoi de la variable de session aux requêtes fetch, pour déterminer si on appelle les requêtes en mode sécurisé ou pas ☺ . Utile pour les témoignages et suggestions ☺



Figure 9: Liste des témoignages – utilisateur connecté

NB : Les bouton d'actions ne sont visibles que pour les utilisateurs connectés

kazaburger EST. 2025

LA CARTE SUGGESTIONS **TÉMOIGNAGES** A PROPOS CONNEXION

Nos clients témoignent

Search... Search

PRODUIT	AVIS
LE BURGER AUVERGNAT	Ella DEUTRANCHE - 16/02/2025 - 5/5 J'ai essayé plusieurs restaurants de hamburgers dans la région, mais celui-ci est de loin le meilleur. Les hamburgers sont énormes et savoureux et le service est excellent. Je reviendrai à coup sûr !
PIZZA ORIENT EXPRESS : SAUCISSES, POIVRONS, CURRY	Clément P. - 29/01/2025 - 3/5 La pizza était bonne, mais j'ai trouvé la croûte un peu trop épaisse. - Clément P.
BURGER ROULÉ	Emma B. - 05/01/2025 - 3/5 Le burger était bon, mais j'ai trouvé le pain un peu sec. - Emma B.
PIZZA AU CHILI (OU COMMENT CUISINER UN RESTE DE CHILI)	Thomas D. - 06/12/2024 - 5/5 Un burger exceptionnel ! La viande était cuite à la perfection, le pain était moelleux et les frites étaient croustillantes. Un vrai régal ! - Thomas D.

Figure 10: Témoignages - utilisateur non connecté

kazaburger EST. 2025

LA CARTE A PROPOS CONNEXION

Témoignages

em Search

PRODUIT	AVIS
BURGER ROULÉ	Emma B. - 05/01/2025 - 3/5 Le burger était bon, mais j'ai trouvé le pain un peu sec. - Emma B.
PIZZA MÈMÉ À LA CHAIR À SAUCISSE	Emma M. - 10/07/2024 - 4/5 Une pizza savoureuse avec une bonne garniture. J'ai apprécié la variété des ingrédients. - Emma M.
PIZZA AUX POMMES	Emma B. - 18/07/2023 - 3/5 Le burger était bon, mais j'ai trouvé le pain un peu sec. - Emma B.

Nb : Remarquez les boutons servant à éditer et supprimer un témoignage

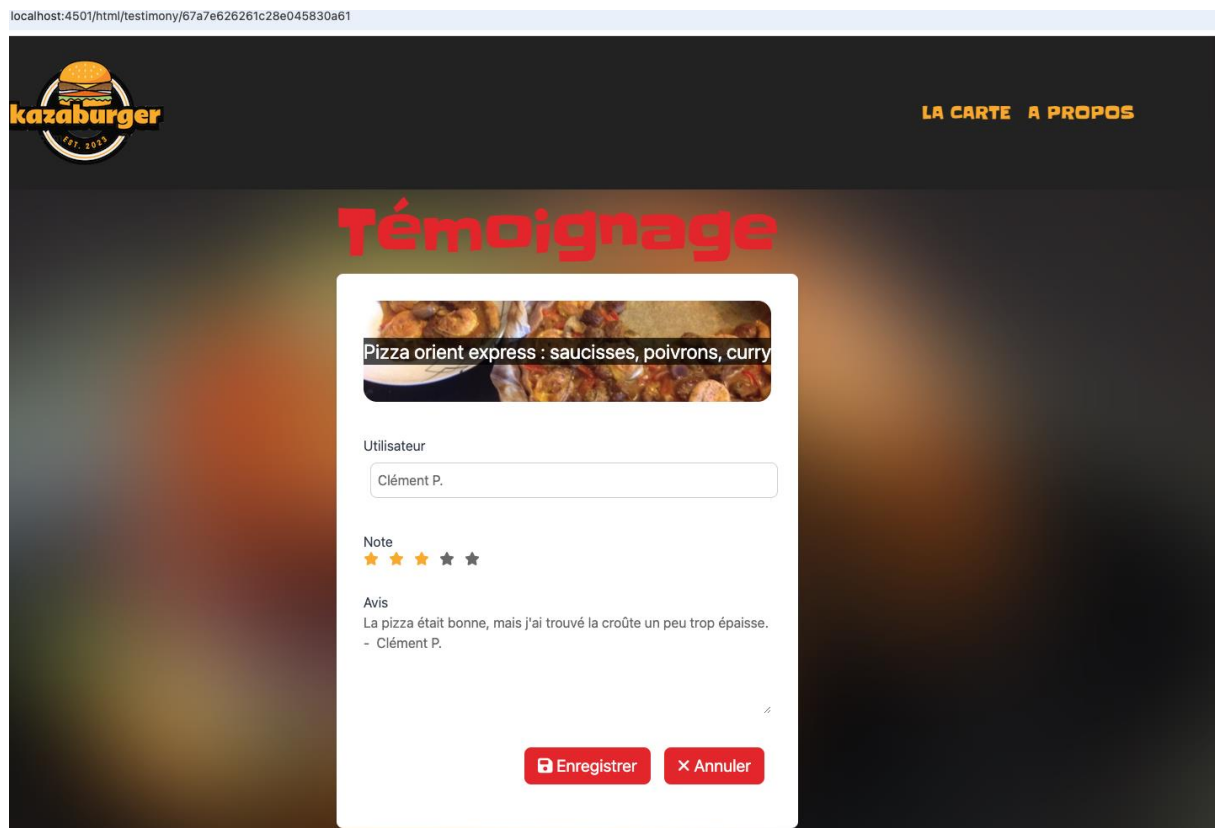


Figure 11: Edition d'un témoignage

NB : Remarquer la modification du vote en cliquant sur les étoiles

L'enregistrement pourra utiliser la route définie en json

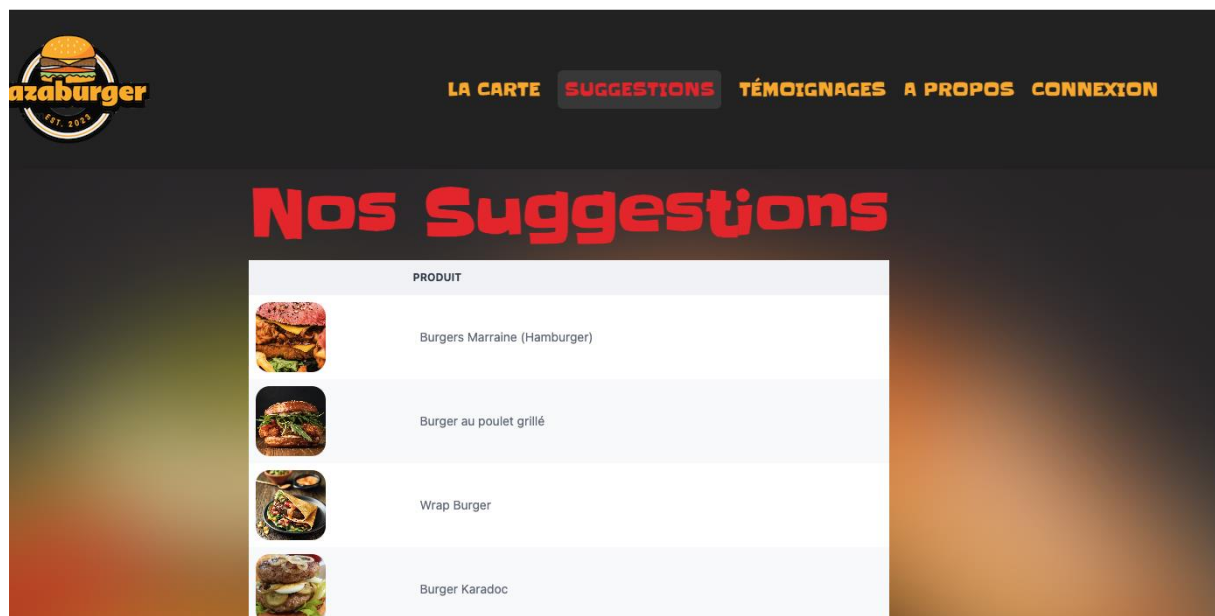


Figure 12: Liste des suggestions - Utilisateur non connecté

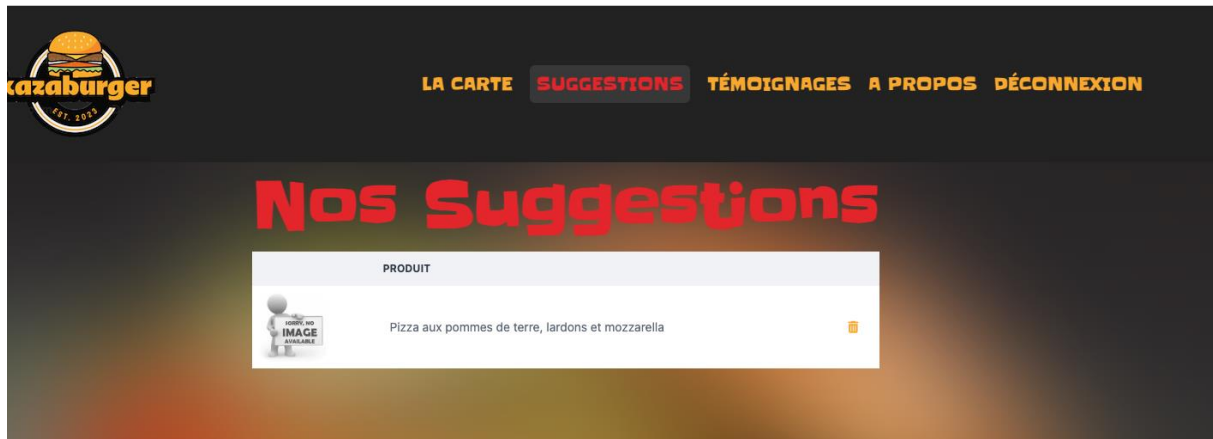


Figure 13: Liste des suggestions. - utilisateur connecté

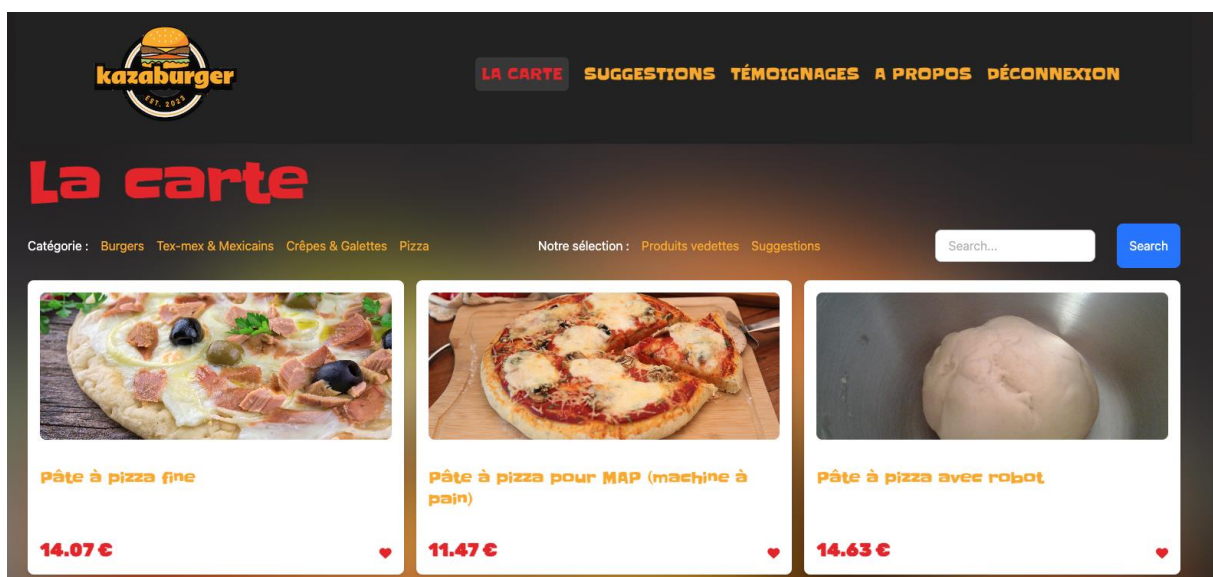


Figure 14: Ajout des suggestion depuis la liste des produits