

Rapport de projet PC3R

-
HungerSpiele

-
Belkacem GUELIANE
Jae-Soo LEE

1. Description de l'Application

L'application que nous avons choisie pour le projet PC3R est une application de cuisine et de recettes. l'idée nous est venue il y a longtemps, nous pensions qu'il serait bon d'avoir une application web qui nous permette de rechercher librement des recettes et des idées de repas, ainsi que de tenir un inventaire de tous les ingrédients que nous avons à la maison, et ce projet était l'occasion parfaite de concrétiser cette idée.

L'application permet donc à l'utilisateur d'utiliser la page d'accueil pour rechercher n'importe quel mot clé se rapportant à une recette, qu'il s'agisse du nom de la recette, d'un ingrédient ou d'une cuisine nationale.

Dans la section inventaire, l'utilisateur peut s'inscrire et consulter la liste des produits qu'il a à sa disposition, ainsi qu'une alarme de date d'expérimentation pour espérer l'inciter à utiliser ces ingrédients et limiter le gaspillage.

L'application est composée d'un côté client écrit en JavaScript à l'aide de la bibliothèque React, et d'un côté serveur écrit en GoLang, nous parlerons des deux en détail plus loin.

2. l'API Web

Pour ce projet, nous avons utilisé 2 APIs WEB :

- a. **spoonacular recipe and food API** (<https://spoonacular.com/food-api>).

Il s'agit d'une API très riche contenant plus de 5000 recettes, 2600 ingrédients, 115k articles de menu et 90k produits. Cette API peut être utilisée de nombreuses façons, par exemple la planification des repas pour les écoles, le suivi des informations nutritionnelles et des régimes, la budgétisation...

Dans notre cas, nous avons utilisé la fonctionnalité "Complex Search" qu'ils proposent (<https://spoonacular.com/food-api/docs#Search-Recipes-Complex>) pour obtenir simplement les recettes correspondant au mot clé qu'un utilisateur recherche (qu'il s'agisse d'un ingrédient, du nom d'une recette ou d'une cuisine).

Voici un exemple typique de requête :

```
rqt := fmt.Sprintf("https://api.spoonacular.com/recipes/complexSearch?  
apiKey=%s  
&query=%s  
&number=12  
&fillIngredients=true  
&addRecipeInformation=true  
&instructionsRequired=false"  
, foodAPIKey, "beef")
```

le résultat est une liste de 12 recettes maximum, voici un exemple des informations utiles de l'une d'entre elles

```
[  
{  
  "vegetarian": false,  
  "pricePerServing": 135.58,  
  "extendedIngredients": [  
    {  
      "id": 23572,  
      "aisle": "Frozen;Meat",  
      "image": "beef-cubes-raw.png",  
      "consistency": "solid",  
      "name": "beef",  
      "nameClean": "beef",  
      "original": "3/4 pound beef, tenderloin",  
      "originalString": "3/4 pound beef, tenderloin",  
      "originalName": "beef, tenderloin",  
      "amount": 0.75,  
      "unit": "pound",  
      "meta": [],  
      "metaInformation": [],  
      "measures": {  
        "us": {  
          "amount": 0.75,  
          "unitShort": "lb",  
          "unitLong": "pounds"  
        },  
        "metric": {  
          "amount": 340.194,  
          "unitShort": "g",  
          "unitLong": "grams"  
        }  
      }  
    }  
  ],  
}
```

```

.....],

    "id": 634698,
    "title": "Beef Tataki",
    "readyInMinutes": 45,
    "servings": 4,
    "sourceUrl": "http://www.foodista.com/recipe/6R7JGFRL/beef-tataki",
    "image": "https://spoonacular.com/recipeImages/634698-312x231.jpg",
    "imageType": "jpg",
    "summary": "Beef Tataki might be just the main course you are searching for. One serving contains <b>259 calories</b>, <b>18g of protein</b>, and <b>17g of fat</b>. This recipe serves 4 and costs $1.36 per serving. This recipe is liked by 1 foodies and cooks. Head to the store and pick up beef, ginger, daikon, and a few other things to make it today. To use up the lemon you could follow this main course with the <a href='\"https://spoonacular.com/recipes/lemon-shortbread-cookies-with-lemon-icing-a-tribute-to-aunt-roxanne-487814\"'>Lemon Shortbread Cookies with Lemon Icing {A Tribute to Aunt Roxanne}</a> as a dessert. All things considered, we decided this recipe <b>deserves a spoonacular score of 42%</b>. This score is solid. Try <a href='\"https://spoonacular.com/recipes/beef-tataki-for-two-551512\"'>Beef Tataki for Two</a>, <a href='\"https://spoonacular.com/recipes/beef-tataki-41576\"'>Beef Tataki</a>, and <a href='\"https://spoonacular.com/recipes/beef-tataki-with-ponzu-sauce-41488\"'>Beef Tataki With Ponzu Sauce</a> for similar recipes.",
    "cuisines": [],
    "dishTypes": [
        "lunch",
        "main course",
        "main dish",
        "dinner"
    ],
    "diets": [
        "gluten free",
        "dairy free"
    ],
    "occasions": [],
    "spoonacularSourceUrl": "https://spoonacular.com/beef-tataki-634698",
    "usedIngredientCount": 0,
    "missedIngredientCount": 8
}

.....]

```

Après que l'utilisateur a soumis sa recherche, le mot clé est envoyé au serveur qui envoie la requête à l'API, reçoit la réponse, filtre les informations et renvoie les parties utiles au client, ce dernier affichant les résultats.

nous voulions également ajouter un affichage du prix pour chaque recette, au départ nous voulions prendre la liste des ingrédients et la faire passer hypothétiquement par une "API Carrefour" qui nous donnerait une estimation précise du prix pour un utilisateur français, malheureusement ni Carrefour ni aucune autre chaîne ne propose d'API à ces fins, Nous avons donc choisi d'utiliser le prix fourni par l'API Spoonacular elle-même (bien qu'il ait tendance à être déraisonnablement élevé puisqu'il inclut le prix de l'équipement), cependant ce prix est en USD, et donc pour convertir en Euros nous avons utilisé notre deuxième API.

b. Free Currency Converter API (<https://www.currencyconverterapi.com/>).

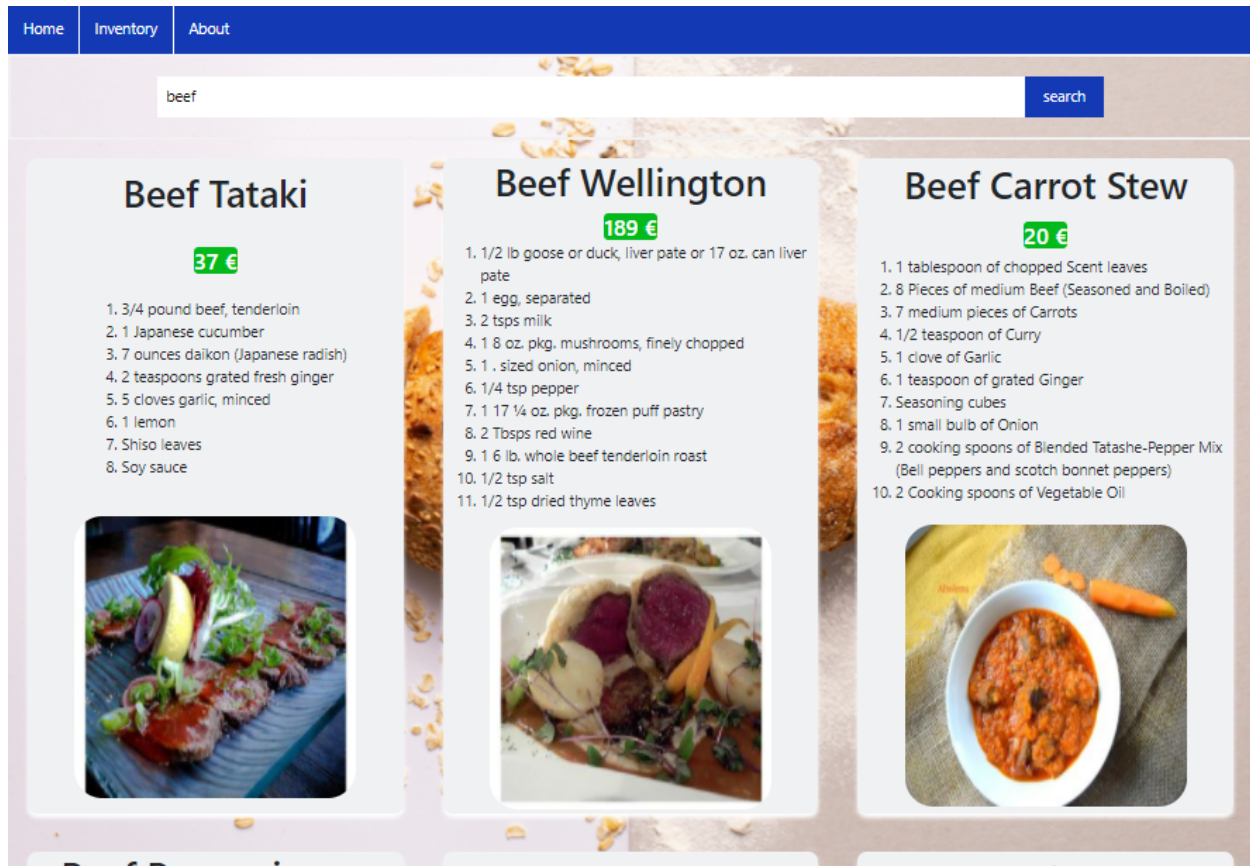
Cette API fournit les taux de change de diverses devises.

l'API met à jour les taux de change toutes les 30 minutes, et pour l'utiliser, nous lançons une goroutine au démarrage du serveur, qui envoie une requête pour le taux de change des dollars en euros, met à jour la constante correspondante sur le serveur, puis dort pendant une heure.

```
func updateExchangeRate() {
    for {
        rqt :=
fmt.Sprintf("https://free.currconv.com/api/v7/convert?q=USD_EUR&compact=ultra&apiKey=%s", currencyAPIKey)
        res, _ := http.Get(rqt)
        body, _ := ioutil.ReadAll(res.Body)
        var exch ExchangeRate
        json.Unmarshal(body, &exch)
        dollarToEuro = exch.USDEUR
        fmt.Println(dollarToEuro)
        time.Sleep(3600 * time.Second)
    }
}
```

Nous utilisons le constant dollarToEuro pour ajouter le champ "priceeuro" dans le JSON que nous renvoyons au client, cette valeur est simplement le prix en dollars multiplié par le taux de change, cette valeur est ensuite affichée pour chaque recette.

Voici l'exemple d'un utilisateur qui cherche "beef":



3. Les Fonctionnalités

page d'accueil

- barre de recherche textuelle
- recherche flexible (saisie du nom de la recette, d'un ingrédient, d'une cuisine nationale ...)
- affichage de tous les résultats dans des panneaux séparés
- affichage du titre, des ingrédients, d'une image et du prix estimé pour chaque recette
- titre et image cliquables pour accéder à la description détaillée de la recette
- une barre de navigation en haut de page pour accéder aux autres pages du site web

page d'inventaire

- authentification (accès au compte utilisateur)
- une barre de recherche textuelle permettant de rechercher les différentes catégories saisies par l'utilisateur (fromage, vin, condiments...)
- un bouton d'ajout de produit
- un panneau d'ajout de produit pliable contenant un formulaire avec la catégorie, le nom, la quantité et la date d'expiration du produit.
- un affichage des produits de la catégorie recherchée
- une fonctionnalité d'avertissement qui fait passer la date de péremption en rouge si elle est inférieure à 3 jours de l'heure actuelle.
- un bouton de suppression pour chaque produit
- un bouton de déconnexion à droite de la barre de navigation.

page d'accueil

- une description de notre mission

remarque :

le système d'authentification que nous utilisons est un peu "unique", il combine à la fois l'inscription et la connexion en une seule fonctionnalité, en résumé, il fonctionne comme suit :

- Si le nom d'utilisateur existe déjà et que le mot de passe est correct, alors le login est accepté
- Si le nom d'utilisateur existe déjà et que le mot de passe est incorrect, la connexion est rejetée.
- Si le nom d'utilisateur est nouveau, le système acceptera n'importe quel mot de passe, il mémorise le nom d'utilisateur et le mot de passe, et il procédera à la connexion.

Nous stockons également le dernier utilisateur dans la mémoire locale, ainsi la session est maintenue jusqu'à la déconnexion (comme Facebook par exemple).

4. Cas d'Utilisations

- Belkacem arrive sur la page d'accueil de l'application. il tape "cheese cake" dans la barre de recherche. Il obtient tous les résultats et choisit la recette dont le prix est le plus bas. il clique sur le titre pour voir plus de détails et les instructions de préparation.
- Jae-Soo se rend sur la page d'inventaire. Il se connecte en utilisant son nom d'utilisateur et son mot de passe. il tape du fromage dans la barre de recherche pour voir quels ingrédients il lui reste. il remarque que la "mozzarella" qu'il a ajoutée il y a une semaine expire demain. il va sur la page d'accueil et tape mozzarella et choisit une des recettes.
- Léa achète du vin. Elle vient sur le site et va sur la page d'inventaire. elle est déjà connectée donc elle est dirigée directement vers son inventaire sans authentification. elle clique sur le bouton "+". Elle remplit le formulaire avec la catégorie vin, le nom du vin. le nombre de bouteilles et la date d'expiration. elle soumet et l'inventaire montre son nouveau produit.
- Morgan vient sur le site pour la première fois. Elle va sur la page d'inventaire et écrit son nom d'utilisateur et son mot de passe. Elle se connecte et elle reçoit un message d'avertissement lui demandant de sauvegarder son nom d'utilisateur et son mot de passe car c'est la première fois qu'elle se connecte avec ces coordonnées. Elle remplit ensuite son inventaire avec tous les ingrédients qu'elle a. elle clique sur le bouton de déconnexion.
- Laurent a fait une recette. il va sur sa page d'inventaire. Il recherche les produits qu'il a consommés et les supprime. Il va sur la page "about" et remercie Gottin-sama pour le bon repas.

5. Les Données

Pour garder la trace de nos données, nous avons utilisé principalement une structure de données, que nous avons mémorisée sous la forme d'un JSON. Nous gardons la trace du nom d'utilisateur, du mot de passe et de la liste d'inventaire de chaque utilisateur organisée par catégorie. la structure que nous avons utilisée est la suivante:

```

type Users struct {
    UserList []User `json:"userlist"`
}

type User struct {
    UserName  string    `json:"username"`
    Password  string    `json:"password"`
    Inventory []Category `json:"inventory"`
}

type Category struct {
    Name      string    `json:"name"`
    Products []Product `json:"inventory"`
}

type Product struct {
    PName      string    `json:"pname"`
    Number     int       `json:"number"`
    Expiration time.Time `json:"expiration"`
}

```

Cette structure de données est chargée sur le serveur à son démarrage, elle est ensuite mise à jour lorsqu'un utilisateur effectue une opération comme la première connexion ou l'ajout ou la suppression de produits.

voici un exemple du contenu de data.json

```

{
  "userlist": [
    {
      "username": "belkacem",
      "password": "1998",
      "inventory": [
        {
          "name": "cheese",
          "inventory": [
            {
              "pname": "mozzarella ",
              "number": 3,
              "expiration": "2021-06-29T22:00:00Z"
            }
          ]
        }
      ]
    }
  ]
}

```



```

    },
    {
      "pname": "fetta",
      "number": 3,
      "expiration": "2021-05-30T22:00:00Z"
    }
  ],
  {
    "name": "wine",
    "inventory": [
      {
        "pname": "vino verde",
        "number": 2,
        "expiration": "2021-05-28T14:20:41.675Z"
      }
    ]
  }
]
}
]
}

```

bien que cette structure de données fonctionne parfaitement avec l'application, et qu'elle soit protégée contre les accès simultanés, nous la considérons comme une lacune de projet, puisque nous n'avons pas eu le temps d'implémenter une véritable base de données, elle manque donc beaucoup de rapidité d'accès et de sécurité, problèmes qui deviendraient certainement apparents si le serveur est sous tension en étant utilisé simultanément par un grand nombre d'utilisateurs.

comme pour le contact avec les API externes, les données de l'API spoonacular ne sont pas du tout conservées sur le serveur, cependant la variable "dollarToEuro" est mise à jour régulièrement avec le taux de change actuel obtenu à partir de "Free Currency Converter API".

6. Le Serveur

Nous avons choisi d'écrire notre serveur en GoLang (**l'approche ressources**).

Le serveur est composé d'un routeur qui route chaque lien contacté vers une handler fonction qui récupère les ressources dont le client a besoin pour fabriquer son contenu.

voici la liste des points de contact que le serveur utilise :

```

r.HandleFunc("/recipes/search/{rname}", getRecipes).Methods("GET")
r.HandleFunc("/users/search/{uname}/{rsearch}", getUserData).Methods("GET")
r.HandleFunc("/users/login/{uname}/{upassword}", getUser).Methods("GET")
r.HandleFunc("/users/submit", addProduct).Methods("POST")
r.HandleFunc("/users/update", updateProduct).Methods("POST")

```

- **getRecipes** contacte l'API Spoonacular pour obtenir la liste des recettes correspondantes et il construit ensuite la structure de réponse JSON contenant les informations utiles (déjà expliquées ci-dessus) qu'il renvoie au client.

- **getUserData** reçoit l'identifiant de l'utilisateur et le mot clé de recherche que l'utilisateur tape lorsqu'il consulte son inventaire, et renvoie la structure JSON suivante contenant la liste des produits de cette catégorie:

```

type UserDataResponse struct {
    UserName string    `json:"username"`
    Category string    `json:"category"`
    Answer    []user.Product `json:"answer"`
}

```

- **getUser** reçoit le nom d'utilisateur et le mot de passe que l'utilisateur utilise pour se connecter, si le nom d'utilisateur existe déjà dans la base de données, il compare le mot de passe et envoie un true ou false selon qu'il correspond ou non, sinon il crée une nouvelle entrée dans la base de données en sauvegardant le nouveau nom d'utilisateur et le mot de passe.

La structure JSON de la réponse est la suivante :

```

type LoginResponse struct {
    Answer    bool `json:"answer"`
    FirstTime bool `json:"firsttime"`
}

```

La première fois est utilisée pour indiquer si le nom d'utilisateur et le mot de passe ont été enregistrés pour la première fois, si c'est le cas, un message d'avertissement apparaîtra pour en informer l'utilisateur.

- **addProduct** reçoit le JSON suivant du client lorsque l'utilisateur ajoute un produit:

```
type AddProd struct {  
    User      string    `json:"user"`  
    Category  string    `json:"category"`  
    Name      string    `json:"name"`  
    Number    int        `json:"number"`  
    MyDate    time.Time `json:"date"`  
}
```

Il utilise ensuite l'information pour trouver l'utilisateur dans la base de données et ajouter le produit dans son inventaire sous la bonne catégorie.

- **updateProduct** reçoit le même JSON que addProduct, elle est contactée par le client lorsqu'un utilisateur essaie de supprimer le produit de son inventaire, la fonction localiser le produit et supprime l'entrée de la base de données, bien que le temps ne l'ait pas permis, nous allons également ajouter une fonctionnalité permettant de modifier les détails du produit.

et enfin, nous avons la goroutine **updateExchangeRate** déjà mentionnée que nous utilisons pour mettre à jour le taux de change.

7. Le Client

notre client est écrit en JavaScript à l'aide de la bibliothèque React.

Le site est principalement composé de 3 pages :


- **la page d'accueil** : où l'on peut rechercher des recettes.

[Home](#)
[Inventory](#)
[About](#)

Brownie Swirl Cheese Cake

24 €


- 1 (8 oz.) pkg. brownie mix
- 2 (8 oz.) Philadelphia cream cheese, softened
- 2 eggs
- 1 c. milk chocolate pieces, melted
- 1/2 c. sugar
- 1 tsp. vanilla



Mascarpone & Ricotta Cheese Cake

30 €


- 1/4 cup almonds, ground
- 1 tablespoon butter, melted
- 16oz whipped cream cheese, room temperature
- 4 large eggs, room temperature
- 2 tbsp flour
- 1 1/2 cups graham crackers
- 1 tbsp lemon juice
- 8oz mascarpone cheese, room temperature
- 8oz ricotta cheese
- 1/2 tsp salt
- 1 1/4 cups sugar
- 2 tbsp sugar
- 1 tsp vanilla extract



Cheesecake with cranberries

38 €

- Dash allspice
- 1 cup butter, softened
- Dash cloves
- 250g McVitie's Wholewheat Digestive cookies, crumbled
- 1 Tbs cornstarch
- 226g cranberries, fresh or frozen
- 450g cream cheese, softened
- 2 eggs, lightly beaten
- 1/4 tsp orange extract
- 1/4 cup sour cream
- 1/2 cup sugar
- 1/4 cup water



- **l'inventaire** : où nous pouvons suivre les produits disponibles.



Username

Enter Username

Password

Enter Password

Login

search

+

cheese

- **mozzarella** Number: 3 Date Of Expiration 30/06/2021
✕
- **fetta** Number: 3 Date Of Expiration 31/05/2021
✕
- **parmigiano** Number: 1 Date Of Expiration 04/12/2021
✕
- **pecorino** Number: 2 Date Of Expiration 12/10/2021
✕

[Home](#)
[Inventory](#)
[About](#)

[belkacem](#) | [Logout](#)

Enter Your Product:

Category

Product name

Number of Products

Date of Expiration

wine

- Number: 2

- **la page à propos** : pour quelques informations supplémentaires


[Home](#)
[Inventory](#)
[About](#)

About Us

We are a couple of food lovers devoted to Gottin-sama the goddess of the hungry.

By all means, please search for any recipe on our home page, and register your ingredients in the inventory .

Our Team



Gottin-sama

GOD

She feeds the hungry.

gottin@holy.com

nous allons maintenant donner quelques détails sur les composants que nous avons utilisés :

Index.js: c'est le principal point d'entrée du site et il renvoie un objet RoundPoint

RoundPoint.js: c'est le routeur que nous utilisons pour naviguer dans les 3 pages de l'application

```
<Switch>
  <Route path="/about"><About /></Route>
  <Route path="/login"><Login /></Route>
  <Route path="/"><App /></Route>
</Switch>
```

Il affiche également la barre de navigation du haut.

APP.js: c'est le composant qui affiche la page d'accueil.

Lorsqu'un utilisateur soumet le contenu de la barre de recherche, **getRecipes** est appelé sur le serveur et renvoie une liste de recettes, App affiche ensuite les objets **Recipe.js** en utilisant les données JSON qu'il a reçues du serveur.

About.js c'est une page à propos très simple.

Login.js : c'est le composant le plus compliqué du client.

Le composant affiche d'abord la boîte de connexion où l'utilisateur peut entrer son nom d'utilisateur et son mot de passe, puis appelle **getUser** sur le serveur qui renvoie si la connexion est acceptée.

si la connexion est acceptée, la page renvoie l'inventaire en utilisant un objet **User.js** qui appelle **getUserData** et affiche une liste d'objets **Product.js** en utilisant les données qu'il reçoit, l'ajout d'un produit se fait en utilisant le composant **ProductInput.js**, qui appelle **addProduct** sur le serveur, la suppression d'un produit appelle **updateProduct**.

8. requêtes HTTP

Notre serveur prend le rôle d'une API REST que le client contacte pour obtenir les données nécessaires à l'affichage de ses éléments, les requêtes HTTP qui sont utilisées pour chaque fonction du serveur sont les suivantes

getRecipes (App.js)

```
const response = await
fetch(`http://localhost:8000/recipes/search/${cleanSearch}`);
```

getUser (Login.js)

```
const response = await
fetch(`http://localhost:8000/users/login/${userNameV}/${passwordV}`);
```

getUserData (Login.js)

```
const response = await
fetch(`http://localhost:8000/users/search/${userNameV}/${cleanSearch}`);
```

addProduct (ProductInput.js)

```
const requestOptions = {
  method: 'POST',
  body: JSON.stringify({user:userID, category: inCat, name:inName,
number:inNum, date:inDate });});

fetch('http://localhost:8000/users/submit', requestOptions)
```

updateProduct (Product.js)

```
const requestOptions = {
  method: 'POST',
  body: JSON.stringify({user:usr, category: cat, name:prod.pname});
  fetch('http://localhost:8000/users/update', requestOptions);
  callback();
}
```

Nous avons déjà discuté des structures de réponse dans la section Serveur.

9. Le développement

Le plus gros problème que nous avons rencontré au début du développement était que nous n'avions aucune expérience avec JavaScript et le développement Web en général, nous avons donc dû construire un petit projet de test pour apprendre les fonctionnalités de base et s'habituer à la logique et aux limitations de JavaScript et React.

Une fois que cela a été fait, nous avions déjà une idée pour l'application web, nous avons donc commencé à construire notre application web en commençant par la page d'accueil, en décidant de l'API externe, et en construisant les éléments d'affichage et les objets de base.

Nous avons ensuite appris à "fetch" les données de l'API et à les filtrer. C'est à ce moment-là que nous avons commencé à construire le serveur et à apprendre comment router les différents appels du client et comment traiter les requêtes et former une réponse.

Notre choix de base de données était un fichier .json que nous avons utilisé comme solution temporaire jusqu'à ce que nous construisions le reste du site web, mais malheureusement nous n'avons pas eu le temps de mettre en œuvre une solution permanente (une base de données complète).

Nous sommes revenus au côté client et avons terminé la construction de la page d'accueil. Nous avons aussi rapidement construit une page "A propos" et ajouté un routeur pour les différentes pages du site.

enfin nous avons construit la page d'inventaire qui a pris le plus de temps car elle était beaucoup plus riche en fonctionnalités. Nous avons construit la boîte de Login, puis l'affichage de l'inventaire par catégorie, l'élément Product et le bouton de suppression, ainsi que le formulaire ProductInput pour ajouter des produits, tout en travaillant sur le serveur pour permettre toutes ces fonctionnalités.

a. quelques choix de design

- nous avons combiné les fonctionnalités de connexion et d'inscription dans la même boîte (comme décrit précédemment), si le nom d'utilisateur est nouveau, la connexion sera acceptée et l'utilisateur recevra une notification que ses données ont été sauvegardées, si le nom d'utilisateur existe, le rejet ou l'acceptation dépendra de la validité du mot de passe
- nous avons gardé l'interface utilisateur minimaliste et propre avec un thème blanc et des bleus.
- nous avons décidé de donner un minimum d'informations sur les recettes (le titre, le prix, les ingrédients et une image) pour pouvoir les afficher dans de petits panneaux, puisque nous avons la possibilité de cliquer sur la recette pour plus de détails
- nous gardons l'utilisateur connecté tant qu'il ne se déconnecte pas, car l'application ne contient pas d'informations particulièrement privées et il est très ennuyeux de devoir se connecter à chaque fois.
- nous gardons l'inventaire dans des catégories pour éviter un désordre lors de l'affichage de tous les produits à la fois sans ordre.
- nous avons beaucoup utilisé `useState()` pour les données locales, et `useEffect()` pour réagir aux différents changements.

b. les points forts

- notre application est entièrement réactive et ne recharge pas la page pour quelque opération que ce soit (AJAX)
- l'application maintient les requêtes http au minimum (requêtes externes et internes)
- l'application est totalement stable et sans bug
- la fonctionnalité permettant d'avertir l'utilisateur de l'expiration imminente d'un produit
- nous avons conçu l'application de manière à ce qu'elle soit entièrement compatible avec les appareils mobiles ayant un affichage dynamique.

c. le défaut

- l'implémentation de notre base de données n'est pas optimale puisqu'elle repose sur un seul fichier
- la sécurité est "primitive" (pas de hachage pour les mots de passe et pas de cryptage de la base de données).

10. Conclusion

ce projet a été le tout premier travail que nous avons fait en développement web pour nous deux, aucun de nous n'avait jamais écrit une ligne de JavaScript avant ce projet, c'est pourquoi nous l'avons trouvé de la plus haute importance et nous y avons mis beaucoup de temps, car l'expérience que nous avons acquise sera certainement utile dans un environnement d'entreprise. nous n'avons pas seulement eu une introduction à JavaScript, mais nous avons également approfondi notre expérience avec GoLang, nous avons appris à comprendre les échanges entre les côtés Client et Serveur et le formatage des données et l'échange entre les différents composants.

Le code que nous avons écrit pourrait certainement être amélioré, en particulier la base de données dont l'implémentation est très insuffisante, nous pourrions également ajouter quelques fonctionnalités dans le futur, mais nous pensons avoir produit une application web qui est utile et que nous utiliserons probablement dans le futur dans nos aventures culinaires!