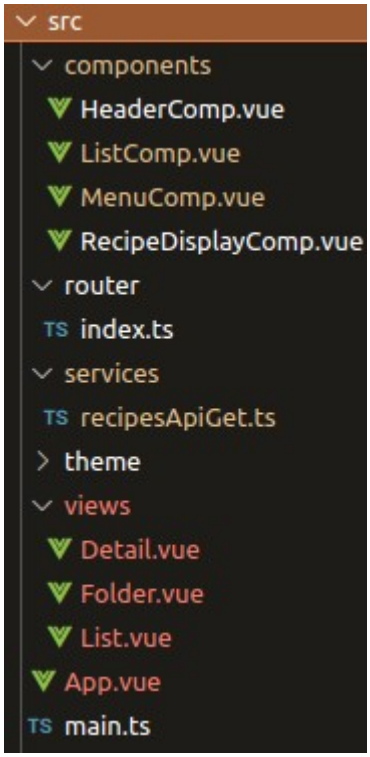
/src : structure du code source

Le projet est structuré comme le montre l'image ci-dessous avec le dossier « src » comme racine.



Il contient les 3 pages dans le dossier « view » (nommés Folder, Detail et List) qui sont rendues par App.vue. L'index.ts permet la redirection entre ces pages les acheminant les unes vers les autres. Le main.ts est la configuration initiale de l'application qui permettra aux libreries de fonctionner dans toute l'application. Les 4 composants du dossier « composants » facilitent certaines des fonctionnalités clés de certaines ou de toutes les pages.

/src/router/index.ts

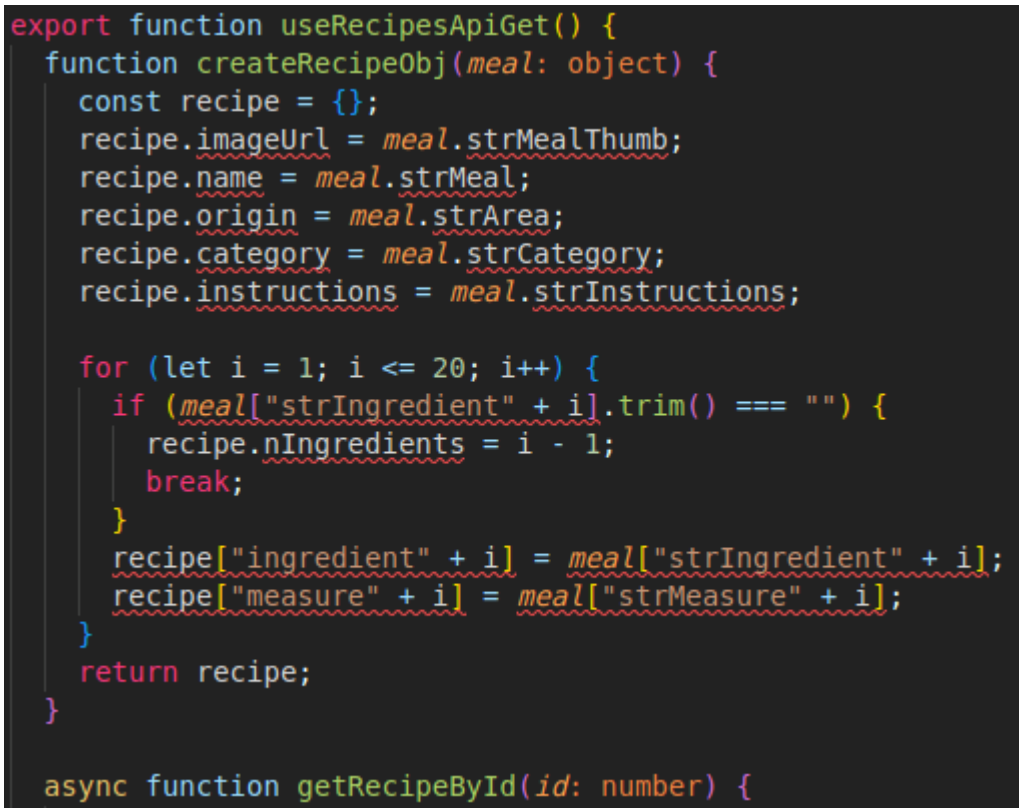
```ts
import { createRouter, createWebHistory } from "@ionic/vue-router";
import { RouteRecordRaw } from "vue-router";
import Detail from "../views/Detail.vue";
import Folder from "../views/Folder.vue";
import List from "../views/List.vue";

const routes: Array<RouteRecordRaw> = [
  {
    path: "/",
    redirect: "/folder",
  },
  {
    path: "/folder",
    name: "Folder",
    component: Folder,
  },
  {
    path: "/list/:category",
    name: "List",
    component: List,
  },
  {
    path: "/detail/:id",
    name: "Detail",
    component: Detail,
  },
];
```

L'application est composée de 3 pages, où Folder est la page d'accueil et affiche immédiatement une recette aléatoire. La page List affiche la liste des recettes d'une catégorie (lorsqu'elle est sélectionnée dans le menu latéral qui apparaît lorsque l'on clique sur l'icône sandwich). Le fonctionnement de la page Détail est similaire à celui de la page Folder, sauf qu'au lieu d'afficher une recette aléatoire, il s'agit d'une recette sélectionnée dans la page List.

/src/services/recipesApiGet.ts

L'application habilitée par le module de service RecipesApiGet. Il est chargé de récupérer les données des recettes et de les transformer afin qu'elles soient lisibles par les composants ionic-vue puis affichées à l'utilisateur.



```
export function useRecipesApiGet() {
  function createRecipeObj(meal: object) {
    const recipe = {};
    recipe.imageUrl = meal.strMealThumb;
    recipe.name = meal.strMeal;
    recipe.origin = meal.strArea;
    recipe.category = meal.strCategory;
    recipe.instructions = meal.strInstructions;

    for (let i = 1; i <= 20; i++) {
      if (meal["strIngredient" + i].trim() === "") {
        recipe.nIngredients = i - 1;
        break;
      }
      recipe["ingredient" + i] = meal["strIngredient" + i];
      recipe["measure" + i] = meal["strMeasure" + i];
    }
    return recipe;
  }

  async function getRecipeById(id: number) {
```

La fonction createRecipeObj, première fonction du module, n'est pas exportée par défaut, mais elle est utilisée dans les fonctions asynchrones pour structurer une recette individuelle récupérée. Il est utilisé par les fonctions qui récupèrent soit une recette aléatoire, soit une recette spécifique.

```
async function getRecipeById(id: number) {
  let recipe = {};

  await fetch("https://www.themealdb.com/api/json/v1/1/lookup.php?i=" + id)
    .then((rawData) => rawData.json())
    .then((data) => data.meals[0])
    .then((meal) => {
      recipe = createRecipeObj(meal);
    });
  return recipe;
}

async function getRandomRecipe() {
  let recipe = {};

  await fetch("https://www.themealdb.com/api/json/v1/1/random.php")
    .then((rawData) => rawData.json())
    .then((data) => data.meals[0])
    .then((meal) => {
      recipe = createRecipeObj(meal);
    });
  return recipe;
}

async function getRecipesOfCategory(category: string) {
  return await fetch(
    "https://www.themealdb.com/api/json/v1/1/filter.php?c=" + category
  )
    .then((rawData) => rawData.json())
    .then((data) => data.meals);
}

async function getCategories() {
  return await fetch("https://www.themealdb.com/api/json/v1/1/categories.php")
    .then((rawData) => rawData.json())
    .then((data) => data.categories);
}

return {
  getRecipeById,
  getRandomRecipe,
  getRecipesOfCategory,
  getCategories,
};
```

Les 4 fonctions principales de ce module sont celles qui récupéreront les données à afficher sur le front-end. Comme leurs noms l'indiquent déjà, les 4 fonctions asynchrones obtiennent un seul objet de recette par identifiant, un objet de recette aléatoire, une collection de recettes par catégorie, ou une collection de catégories respectivement.

App

App.vue est l'endroit où toutes les pages seront rendues.

```vue
<template>
  <ion-app>
    <ion-split-pane content-id="main-content">
      <MenuComp :categories="categories" />
      <ion-router-outlet id="main-content"></ion-router-outlet>
    </ion-split-pane>
  </ion-app>
</template>
  <script setup lang="ts">
import {
  IonApp,
  IonRouterOutlet,
  IonSplitPane,
  loadingController,
} from "@ionic/vue";
import MenuComp from "@/components/MenuComp.vue";
import { onMounted, ref } from "vue";
import { useRecipesApiGet } from "@/services/recipesApiGet";

const { getCategories } = useRecipesApiGet();
const categories = ref([]);

onMounted(async () => {
  const loading = await loadingController.create({
    message: "Attendre SVP...",
  });

  await loading.present();
  categories.value = await getCategories();
  loading.dismiss();
});
</script>
```

Cela habilitera le composant MenuComp sur chaque page comportant une icône de bouton sandwich (pages Dossier et List). Il est possible de l'afficher sous forme de menu latéral par IonSplitPane lorsque la fenêtre est suffisamment petite.

Folder

La page Dossier permet d'afficher les détails d'une recette aléatoire lorsque l'utilisateur démarre l'application.

```vue
<template>
  <ion-page>
    <HeaderComp />
    <ion-content :fullscreen="true">
      <RecipeDisplayComp :recipe="recipe" />
    </ion-content>
  </ion-page>
</template>

<script setup lang="ts">
import {
  IonContent,
  IonPage,
  loadingController,
  onIonViewWillEnter,
} from "@ionic/vue";

import RecipeDisplayComp from "@/components/RecipeDisplayComp.vue";
import HeaderComp from "@/components/HeaderComp.vue";
import { ref } from "vue";
import { useRecipesApiGet } from "@/services/recipesApiGet";

const { getRandomRecipe } = useRecipesApiGet();

const recipe = ref({
  imageUrl: null,
  name: null,
  origin: null,
  category: null,
});

onIonViewWillEnter(async () => {
  const loading = await loadingController.create({
    message: "Attendre SVP...",
  });

  await loading.present();
  recipe.value = await getRandomRecipe();
  loading.dismiss();
});
</script>
```

Son contenu principal est le composant RecipeDisplayComp qui prend une variable « recipe » comme objet pour afficher la recette aléatoire obtenue avec la fonction getRandomRecipe.

List

La page Liste affichera une liste de toutes les recettes d'une catégorie sélectionnée sur le menu de la barre latérale.

```vue
<template>
  <ion-page>
    <HeaderComp />
    <ion-content :fullscreen="true">
      <ListComp :recipes="recipes" />
    </ion-content>
  </ion-page>
</template>

<script setup lang="ts">
import {
  IonContent,
  IonPage,
  loadingController,
  onIonViewWillEnter,
} from "@ionic/vue";
import HeaderComp from "@/components/HeaderComp.vue";
import ListComp from "@/components/ListComp.vue";
import { ref } from "vue";
import { useRoute } from "vue-router";
import { useRecipesApiGet } from "@/services/recipesApiGet";

const { getRecipesOfCategory } = useRecipesApiGet();
const recipes = ref([]);
const route = useRoute();

onIonViewWillEnter(async () => {
  const loading = await loadingController.create({
    message: "Attendre SVP...",
  });

  await loading.present();
  recipes.value = await getRecipesOfCategory(route.params.category);
  loading.dismiss();
});
</script>
```

Le contenu principal de cette page est le composant ListComp. Il affichera la liste complète des recettes obtenues avec la fonction getRecipesOfCategory. Il utilisera le paramètre « category » de la page comme entrée pour obtenir la collection de recettes qui seront utilisées comme objets dès que selectionée et redirectioné.

Detail

La page Detail permet d'afficher les détails d'une recette selectionée sur la categorie de recettes affichée sur la page List.

```vue
<template>
  <ion-page>
    <HeaderComp :category="recipe.category" />
    <ion-content :fullscreen="true">
      <RecipeDisplayComp :recipe="recipe" />
    </ion-content>
  </ion-page>
</template>

<script setup lang="ts">
import {
  IonContent,
  IonPage,
  loadingController,
  onIonViewWillEnter,
} from "@ionic/vue";
import RecipeDisplayComp from "@/components/RecipeDisplayComp.vue";
import HeaderComp from "@/components/HeaderComp.vue";
import { ref } from "vue";
import { useRoute } from "vue-router";
import { useRecipesApiGet } from "@/services/recipesApiGet";

const { getRecipeById } = useRecipesApiGet();

const route = useRoute();

const recipe = ref({
  imageUrl: null,
  name: null,
  origin: null,
  category: null,
});

onIonViewWillEnter(async () => {
  const loading = await loadingController.create({
    message: "Attendre SVP...",
  });

  await loading.present();
  recipe.value = await getRecipeById(route.params.id);
  loading.dismiss();
});
</script>
```

Son contenu principal est le composant RecipeDisplayComp qui prend une variable « recipe » comme objet pour afficher la recette selectionée obtenue avec la fonction getRecipeById, qui prendra le paramètre « id » de la page en tant qu'input pour obtenir les données de la recette spécifique de l'API. . Cela fonctionne de manière similaire à la page Folder, mais le composant HeaderComp ne permet pas d'accéder directement au menu de la barre latérale. Au lieu de cela, l'utilisateur doit revenir à la page List par le composant standard BackButton.

HeaderComp

Le HeaderComp s’adaptera et changera ses propres étiquettes et ses icônes des boutons en fonction de la page dans laquelle il se trouve, rendant disponible le menu latéral ou le bouton de retour.

```vue
<template>
  <ion-header :translucent="true">
    <ion-toolbar id="appHeader" color="primary">
      <ion-buttons v-if="$route.name == 'Detail'" slot="start">
        <ion-back-button :default-href="/list/ + category" />
      </ion-buttons>
      <ion-buttons v-else slot="start">
        <ion-menu-button auto-hide="false"></ion-menu-button>
      </ion-buttons>
      <ion-title v-if="$route.name == 'List'" size="large">{{
        $route.params.category
      }}</ion-title>
      <ion-title v-else size="large">{{
        $route.name == "Folder" ? "Recette du moment" : "Recette"
      }}</ion-title>
    </ion-toolbar>
  </ion-header>
</template>

<script setup>
import {
  IonHeader,
  IonTitle,
  IonToolbar,
  IonButtons,
  IonBackButton,
  IonMenuButton,
} from "@ionic/vue";

defineProps({
  category: {
    type: String,
    required: false,
  },
});
</script>
```

L'en-tête est présent dans toutes les pages, mais il change en fonction de la route/page, affichant ainsi les titres et les boutons en fonction de la page donnée.

RecipeDisplayComp

Ce composant est utilisé pour afficher tous les détails d'une recette donnée (comme l'origine, la catégorie, les ingrédients et les instructions). Il est utilisé sur les pages/routes Folder et Detail.

```vue
<template>
  <ion-img :src="recipe.imageUrl" style="width: 90%; margin: auto"></ion-img>
  <h3>{{ recipe.name || "recipe name" }}</h3>
  <ul id="non-decorated">
    <li><strong>Origine:</strong> {{ recipe.origin || "recipe origin" }}</li>
    <li>
      <strong>Catégorie:</strong> {{ recipe.category || "recipe category" }}
    </li>
  </ul>
  <h3>Ingrédients</h3>
  <ul>
    <li v-for="i in recipe.nIngredients" :key="i">
      {{ recipe["measure" + i] }} {{ recipe["ingredient" + i] }}
    </li>
  </ul>
  <h3>Instructions</h3>
  <p id="instructions">{{ recipe.instructions || "recipe instructions" }}</p>
</template>

<script setup>
import { IonImg } from "@ionic/vue";

defineProps({
  recipe: {
    type: Object,
    required: true,
  },
});
</script>
```

Il prend essentiellement un objet de recette comme « prop », puis affiche certaines de ses propriétés de manière formatée.

MenuComp

Le travail du composant MenuComp est de lister les catégories.

```vue
<template>
  <ion-menu content-id="main-content" type="overlay">
    <ion-content>
      <ion-list>
        <div id="border">
          <h1>Mes recettes</h1>
          <p>202197834@CollegeAhuntsic.qc.ca</p><p>Bienvenue, Flavio Weinstein Silva</p>      You, 1 sec
        </div>
        <ion-menu-toggle auto-hide="false">
          <ion-item href="/folder/">
            <ion-icon slot="start" :icon="home" :md="home"></ion-icon>
            <ion-label>Accueil</ion-label>
          </ion-item>
          <ion-item v-for="c in categories" :href="'list/' + c.strCategory" :key="c.idCategory">
            <ion-thumbnail slot="start"><img :src="c.strCategoryThumb" alt="meal cat" /></ion-thumbnail>
            <ion-label>{{ c.strCategory }}</ion-label>
          </ion-item>
        </ion-menu-toggle>
      </ion-list>
    </ion-content>
  </ion-menu>
  <ion-router-outlet id="main-content"></ion-router-outlet>
</template>
<script setup lang="ts">
import {
  IonRouterOutlet,
  IonContent,
  IonList,
  IonItem,
  IonLabel,
  IonMenu,
  IonMenuToggle,
  IonIcon,
  IonThumbnail,
} from "@ionic/vue";
import { home } from "ionicons/icons";

defineProps({
  categories: {
    type: Object,
    required: true,
  },
});
</script>
```

La liste des catégories affichées aura un lien vers la même route List, mais avec un paramètre « category » différent, donc List saura à quelle catégorie faire une requête API GET. Avec ces informations, la page List affichera une liste de recettes correspondant au paramètre de « category ». Le prop « category » utilisé pour ce composant sera utilisé dès que la liste des catégories sera obtenue dans App.vue.

ListComp

Le composant ListComp est celui qui heberge les recettes d'une catégorie donnée et les affiche.

```
<template>
  <ion-list id="container">
    <ion-item v-for="r in recipes" :href="'detail/' + r.idMeal" :key="r.idMeal">
      <ion-thumbnail slot="start">
        <img :src="r.strMealThumb" alt="" />
      </ion-thumbnail>
      <ion-label>{{ r.strMeal }}</ion-label>
    </ion-item>
  </ion-list>
</template>

  <script setup>
import { IonLabel, IonItem, IonList } from "@ionic/vue";

defineProps({
  recipes: {
    type: Object,
    required: true,
  },
});
</script>
```

Ce composant, comme le MenuComp, permettra de sélectionner un paramètre différent pour un même route/page. Dans ce cas, la route est la page de Detail, qui recevra un identifiant de recette différent à appeler et à afficher par RecipeDisplayComp.