



Base d'un
configurateur
de produit en
utilisant VueJS

Semaine 3 :

“authentication et
insertion/m-à-j/liste des
produits configurés”

supabase/ sql editor

Insertion dans Supabase

SQL editor

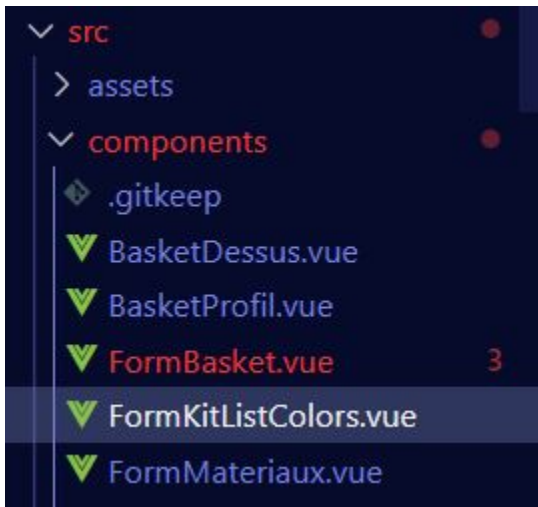
```
create table Basket (  
  id uuid not null default  
  uuid_generate_v4() ,  
  semelle text,  
  pointe text,  
  oeillet text,  
  bande text,  
  lacet text,  
  trimestre text,  
  empeigne text,  
  languette text,  
  Commander boolean not null default  
  false,  
  utilisateur uuid references auth.users  
  not null default uid(),  
  primary key (id) ) ;
```

Très important de bien écrire les parties de la chaussure en minuscule car on l'a écrit en minuscule dans la template svg et sur le formulaire!!!!

Commander en minuscule ou en majuscule comme vous préférez, ici en MaJ

Faire un composant du champ personnalisé listant les couleurs

Dans un fichier `src/components/FormKitListColors.vue` reprendre le code d'un champ listant les couleurs. Définir qu'il reçoit en paramètre `name` et `label` :



← FormKitListColors.vue

src/components/FormKitListColors.vue

Ici le but est de créer un composant automatisant l'appel des boutons avec champ personnalisé et le petit rond rouge autour à la sélection :

Pour cela : le name et le label appelleront directement la partie de la chaussure qui nous intéresse (semelle, empeigne, etc)

```
<script setup lang="ts">
import { colors } from "@/types";
defineProps<{
  name?: string;
  label?: string;
}>();
</script>
```

src/components/FormKitListColors.vue

Faire un composant du champ personnalisé listant les couleurs

```
<template>
  <FormKit
    :name="name"
    :label="label"
    value="#FFFFFF"
    type="radio"
    :options="colors"
    :sections-schema="{
      inner: { $el: null },
      decorator: { $el: null },
    }" input-class="peer sr-only" options-class="flex gap-1">
    <template #label="context">
      <div class="h-6 w-6 rounded-full border-2 peer-checked:border-red-600" style="{ backgroundColor: context.option.value}" />
      <span class="sr-only">{{ context.option.label }}</span>
    </template>
  </FormKit>
</template>
```

Puis l'utiliser dans `src/components/FormBasket.vue`:

Ici le but est de remplacer l'ancien formulaire (ci dessous) qui contenait dans chaque section du formulaire le code inscrit dans le composant FormKitList.vue

```
<FormKit name="oeillet" label="oeillet" value="#FFFFFF" type="radio" :options="colors" :sections-schema="{
  inner: { $el: null },
  decorator: { $el: null },
}" input-class="peer sr-only" options-class="flex gap-1" >
  <template #label="context">
    <div class="h-6 w-6 rounded-full border-2  peer-checked:border-red-600" :style="{ backgroundColor: context.option.value}" />
      <span class="sr-only">{{ context.option.label }}</span>
    </template>
  </FormKit>
```

Ancien formulaire comportant le code désormais dans FormKitList.vue

src/components/FormBasket.vue

Puis l'utiliser dans `src/components/FormBasket.vue`:

```
<script setup lang="ts">
  import type { Basket } from "@/types"
  import { ref } from "vue";
  import SvgProfil from "../BasketProfil.vue";
  import SvgDessus from "../BasketDessus.vue";
  import FormKitListColors from "../FormKitListColors.vue";

  const props = defineProps<{
    data?: Basket;
    id?: string;
  }>();
  const basket = ref<Basket>(props.data ?? {});
</script>
```

← Ne pas oublier d'importer
FomKitListColors.vue
dans le script

src/components/FormBasket.vue

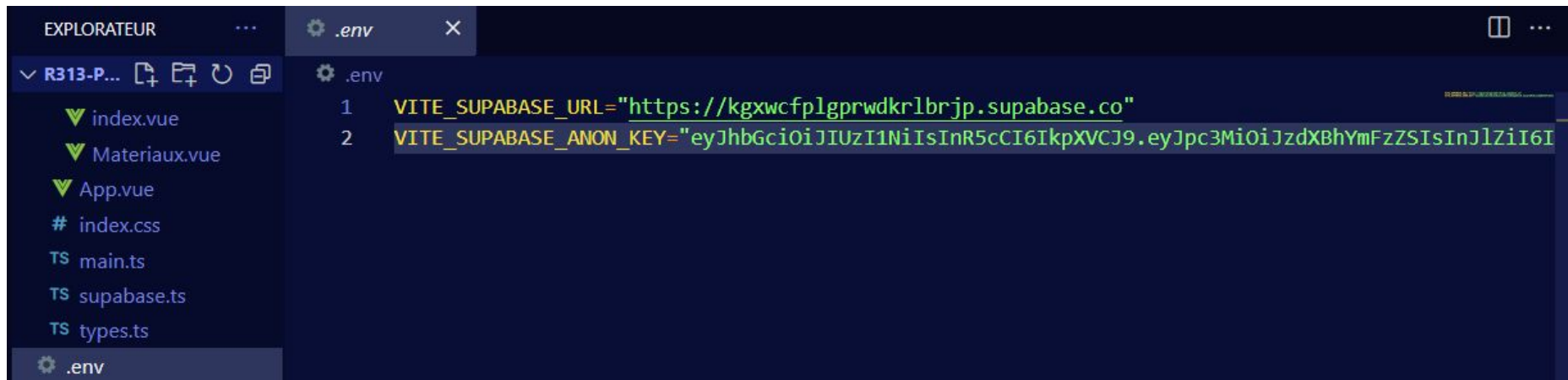
```
<FormKit type="form" v-model="basket">  
  <FormKitListColors name="semelle" label="semelle" />  
  <FormKitListColors name="empeigne" label="empeigne" />  
  <FormKitListColors name="pointe" label="pointe" />  
  <FormKitListColors name="oeillet" label="oeillet" />  
  <FormKitListColors name="bande" label="bande" />  
  <FormKitListColors name="languette" label="languette" />  
  <FormKitListColors name="lacet" label="lacet" />  
  <FormKitListColors name="trimestre" label="trimestre" />  
</FormKit>
```

← Importer ce FormKit à l'endroit de
l'ancien FormKit sur FormBasket.vue

N'oubliez pas de commit régulièrement votre travail



Renommer le .env-exemple en .env avec ses données supabase à récupérer dans settings → api

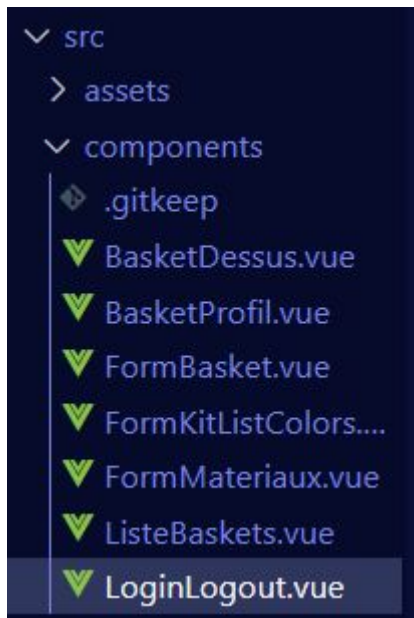


The screenshot shows a code editor interface with a sidebar on the left labeled 'EXPLORATEUR'. The sidebar contains a file tree for a project named 'R313-P...'. The files listed are: index.vue, Materiaux.vue, App.vue, index.css, main.ts, supabase.ts, types.ts, and .env. The .env file is selected and its contents are displayed in the main editor area. The editor shows two lines of code: line 1 is 'VITE_SUPABASE_URL="https://kgxwcfplgprwdkrlbrjp.supabase.co"' and line 2 is 'VITE_SUPABASE_ANON_KEY="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6I...'. The text in the editor is highlighted in green.

```
1 VITE_SUPABASE_URL="https://kgxwcfplgprwdkrlbrjp.supabase.co"  
2 VITE_SUPABASE_ANON_KEY="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6I...
```

src/components/LoginLogout.vue

Faire un composant `src/components/LoginLogout.vue`



← Créer composant LoginLogout.vue

src/components/LoginLogout.vue

Faire un composant `src/components/LoginLogout.vue`

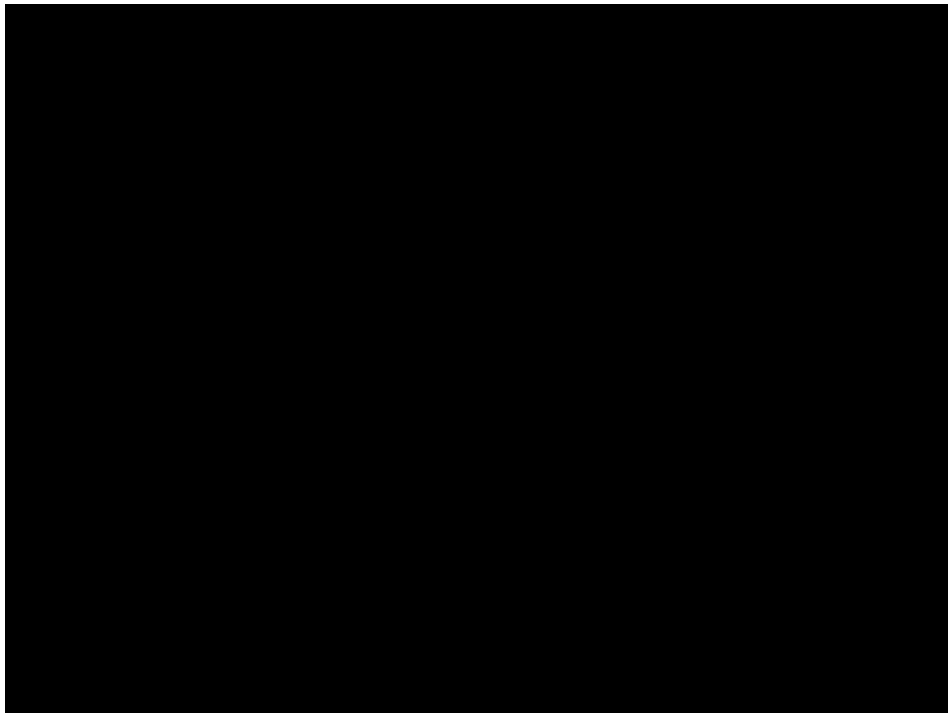
```
<script setup lang="ts">
import { supabase, user } from '../supabase';

</script>
<template>
<div>
  <button v-if="user"
    @pointerdown="supabase.auth.signOut()">
    Se déconnecter ({{user.email}})
  </button>
  <button v-else
    @pointerdown="supabase.auth.signIn({provider: 'github'})">
    Se connecter avec Github
  </button>
</div>
</template>
```

Authentication github par supabase

Vidéo expliquant la démarche pour se connecter depuis github géré par Supabase :

Ouvrir github et supabase au préalable

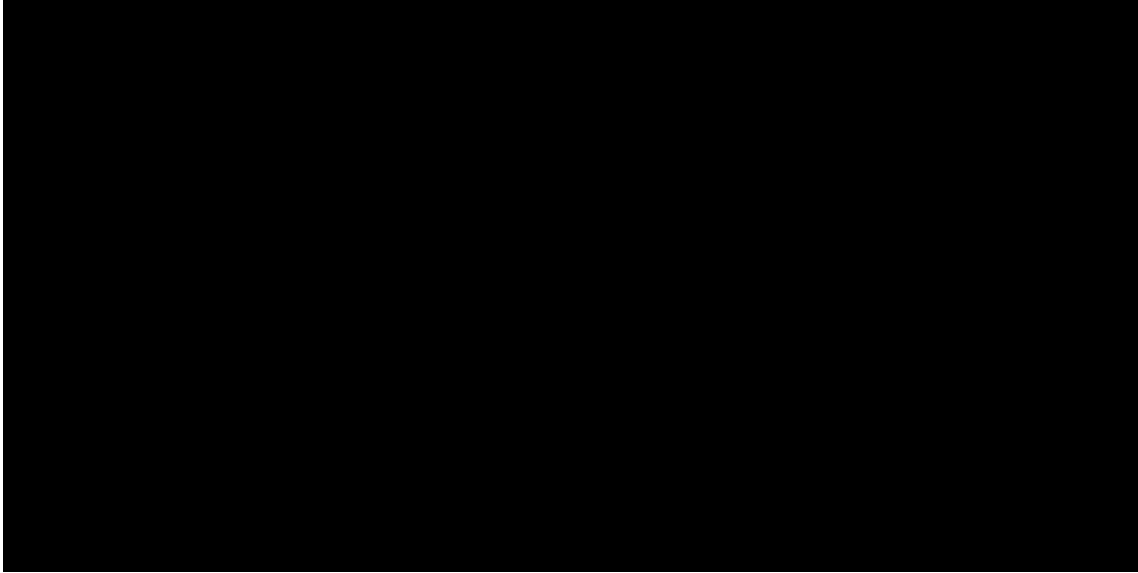


Puis l'utiliser le composant dans `src/App.vue` pour que les visiteurs puissent se connecter depuis toutes les pages du site.

```
▼ App.vue X
src > ▼ App.vue > {} template > nav > ul
1  <template>
2    <nav>
3      <h4 class="text-xl">
4        <Bars3Icon class="inline-block h-5 w-5 text-blue-500" />
5        menu (dans <code class="font-mono">/src/App.vue</code>)
6      </h4>
7      <LoginLogout />
8    <ul> ...
33  </ul>
34  </nav>
35
36  <!-- Affiche les pages -->
37  <Suspense>
38    <router-view class="m-2 border-2 p-2" />
39  </Suspense>
40  </template>
41
42  <script setup lang="ts">
43    import { Bars3Icon } from '@heroicons/vue/20/solid'
44    import LoginLogout from './components/LoginLogout.vue'
45  </script>
46
```

Importer le composant login logout avant le menu

Tester

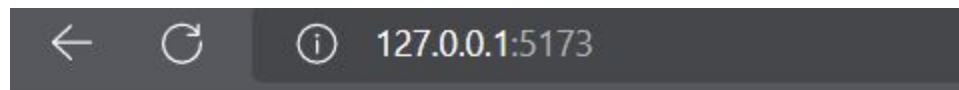


Puisque l'authentification
redirige vers le localhost...
attention à ce que ce soit le
BON localhost



Tester

Puisque l'authentification redirige vers le localhost... attention à ce que ce soit le BON localhost

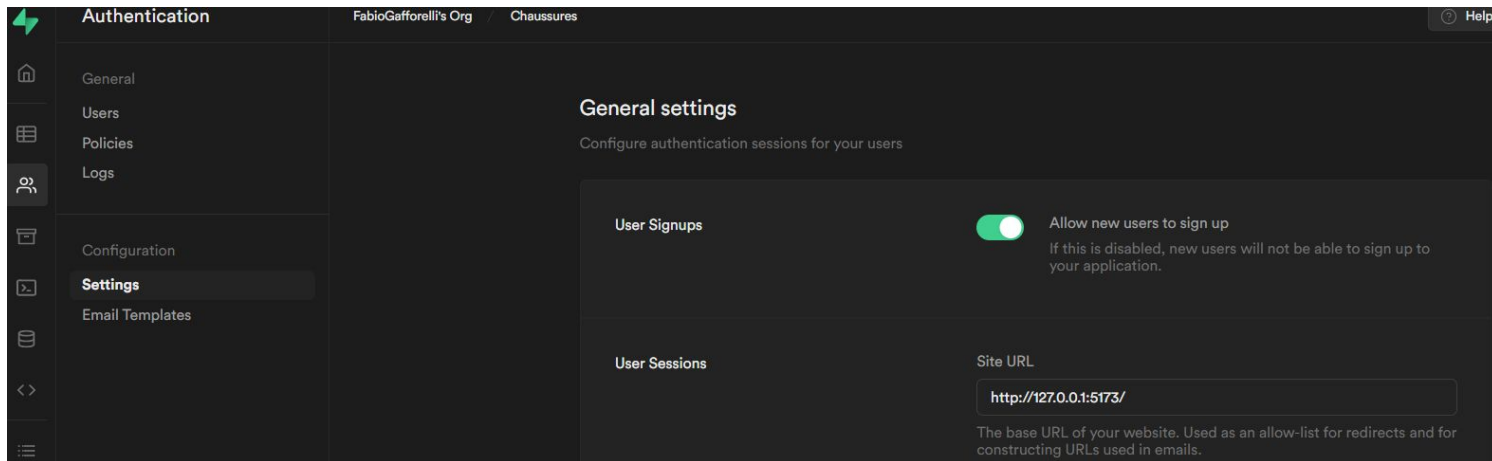


≡ menu (dans /src/App.vue)

Se déconnecter (fabio.gafforelli@edu.univ-fcomte.fr)

Ici je test dans le local host <http://127.0.0.1:5173/>

Tester



Dans supabase, authentication settings, je redirige donc vers <http://127.0.0.1:5173/> pour tester correctement la connexion!

Il est possible de remplacer se connecter par github par “se connecter par mail”

≡ menu (dans /src/App.vue)

Se déconnecter (fa.gafforelli009@gmail.com)

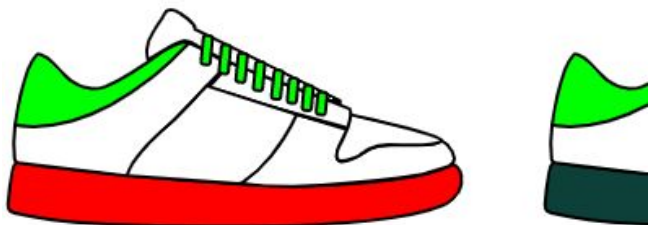
[lien vers /src/pages/index.vue](#)

[lien vers /src/pages/basket/index.vue](#)

[lien vers /src/pages/basket/new.vue](#)

[lien vers /src/pages/Materiaux.vue](#)

Exemples de Baskets



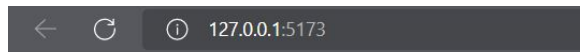
Il est possible de remplacer se connecter par github par “se connecter par mail”

```
<script setup lang="ts">
  import { ref } from "@vue/reactivity";
  import { supabase, user } from "../supabase";
  async function signIn(data, node) {
    const { user, error } = await (nv1Utilisateur.value
      ? supabase.auth.signUp(data)
      : supabase.auth.signIn(data));
    if (error) {
      console.error(error);
      node.setErrors([error.message]);
    }
  }
  const nv1Utilisateur = ref(false);
</script>
```

Il est possible de remplacer se connecter par github par “se connecter par mail”

```
<template>
  <div>
    <button v-if="user" @pointerdown="supabase.auth.signOut()">
      Se déconnecter {{ user.email }}
    </button>
    <FormKit
      v-else
      type="form"
      :submit-label="nv1Utilisateur ? 'S\'inscrire' : 'Se connecter'"
      @submit="signIn"
    >
      <FormKit name="email" label="Votre eMail" type="email" />
      <FormKit name="password" label="Mot de passe" type="password" />
      <formKit
        label="Nouvel utilisateur ?"
        name="nv1Utilisateur"
        type="checkbox"
        v-model="nv1Utilisateur"
      />
    </FormKit>
  </div>
</template>
```

Tester



☰ menu (dans /src/App.vue)

Votre eMail

Mot de passe

☐

Nouvel utilisateur ?

Se connecter

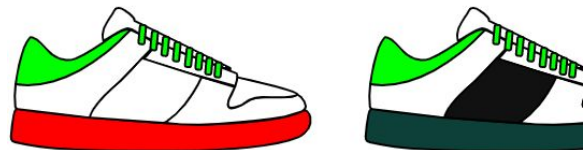
[lien vers /src/pages/index.vue](/src/pages/index.vue)

[lien vers /src/pages/basket/index.vue](/src/pages/basket/index.vue)

[lien vers /src/pages/basket/new.vue](/src/pages/basket/new.vue)

[lien vers /src/pages/Materiaux.vue](/src/pages/Materiaux.vue)

Exemples de Baskets



N'oubliez pas de commit régulièrement votre travail



src/components/FormBasket.vue

Insertion dans Supabase

On va ajouter au formulaire le gestionnaire d'événement `@submit` appelant une fonction `upsertBasket` qui permettra de créer sa chaussure dans supabase et de la modifier grâce au router-push `basket/edit/[id].vue`

```
<script setup lang="ts">
  import type { Basket } from "@/types"
  import { ref } from "vue";
  import { useRouter } from "vue-router";
  import SvgProfil from "@/BasketProfil.vue";
  import SvgDessus from "@/BasketDessus.vue";
  import FormKitListColors from "@/FormKitListColors.vue";

  const router = useRouter();
  const basket = ref({});
  const props = defineProps(["id", "Basket"]);
  if (props.id) {
    // On charge les données de la table Basket
    let { data, error } = await supabase
      .from("Basket")
      .select("*")
      .eq("id", props.id);
    if (error || !data)
      console.log("n'a pas pu charger la table Basket ", error);
    else basket.value = data[0];
  }
}
```




Insertion dans Supabase

On va ajouter au formulaire le gestionnaire d'événement `@submit` appelant une fonction `upsertBasket` qui permettra de créer sa chaussure dans supabase et de la modifier grâce au router-push `basket/edit/[id].vue`

```
    async function upsertBasket(dataForm, node) {  
      const { data, error } = await supabase.from("Basket").upsert(dataForm);  
      if (error) node.setErrors([error.message]);  
      else {  
        node.setErrors([]);  
        router.push({ name: "basket-edit-id", params: { id: data[0].id } });  
      }  
    }  
  }  
</script>
```

src/components/FormBasket.vue

Par rapport à l'ancien formulaire dans template, on rajoute seulement l'appel upsertBasket



```
<FormKit type="form" v-model="basket" @submit="upsertBasket">
  <FormKitListColors name="semelle" label="semelle" />
  <FormKitListColors name="empeigne" label="empeigne" />
  <FormKitListColors name="pointe" label="pointe" />
  <FormKitListColors name="oeillet" label="oeillet" />
  <FormKitListColors name="bande" label="bande" />
  <FormKitListColors name="languette" label="languette" />
  <FormKitListColors name="lacet" label="lacet" />
  <FormKitListColors name="trimestre" label="trimestre" />
</FormKit>
```

N'oubliez pas de commit régulièrement votre travail



Liste des produits

Faire un composant `src/components/ListeBaskets.vue` et l'utiliser sur la page `src/pages/basket/index.vue` qui listera tous les produits configurés.

```
<script setup lang="ts">
import ListeBaskets from "../../components/ListeBaskets.vue";
</script>
<template>
  <div class="p-2">
    <h2>Liste des chaussures perso de l'utilisateur</h2>
    <ListeBaskets />
  </div>
</template>
```

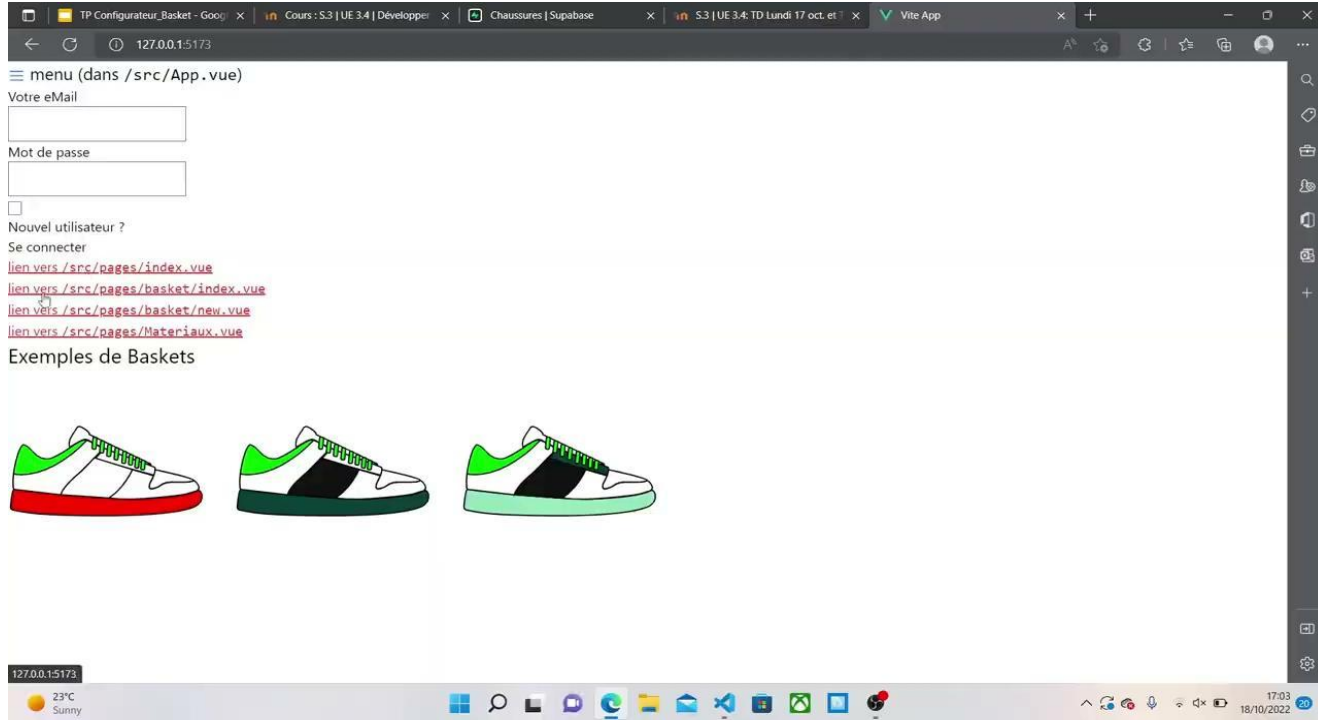
Liste des produits

```
<script setup lang="ts">
import { supabase } from "@supabase";
import BasketProfil from "../BasketProfil.vue";
const props = defineProps({
  max?: number;
})>();
let { data: Basket, error } = await supabase
  .from("Basket")
  .select("*")
  .limit(props.max ?? 100)
if (error) {
  console.log("n'a pas pu récupérer les basket", { error });
}
</script>
```

Liste des produits

```
<template>
  <ul>
    <li v-for="basket in Basket" :key="basket.id basket">
      <router-link
        :to="{ name: 'basket-edit-id', params: { id: basket.id } }"
      >
        <BasketProfil class="w-64" v-bind="basket" />
      </router-link>
    </li>
  </ul>
</template>
```

Tester



N'oubliez pas de commit régulièrement votre travail



Pour aussi afficher sur la page d'accueil (`src/pages/index.vue`) une partie (ici 3 maxi) des produits configurés par l'utilisateur (si connecté testé par `user` à importer de `'@/supabase'`)

```
<script setup lang="ts">
import BasketProfil from "@/components/BasketProfil.vue";
import type { Basket } from "@/types";
import { supabase, user } from '@supabase';
import ListeBaskets from '../components/ListeBaskets.vue';
```

Ne pas oublier d'importer supabase et le user from '@supabase' dans la page index.vue qui doit déjà contenir les anciens exemples

```
import { supabase, user } from '@supabase';
```

copiable pour les plus fainéants...

A la fin de la précédente section :

```
49     >
50     <BasketProfil class="w-64" v-bind="basket" />
51   </RouterLink>
52 </div>
53 </div>
54 </section>
55 <section v-if="user">
56 <h2>
57   un extrait de vos chaussures (<RouterLink
58     class="text-red-600 underline"
59     to="/basket"
60     >Toutes les voir</RouterLink
61   >)
62 </h2>
63 <ListeBaskets class="flex flex-wrap gap-2" :max="3" />
64 </section>
65 </template>
```

Fin de la section précédente

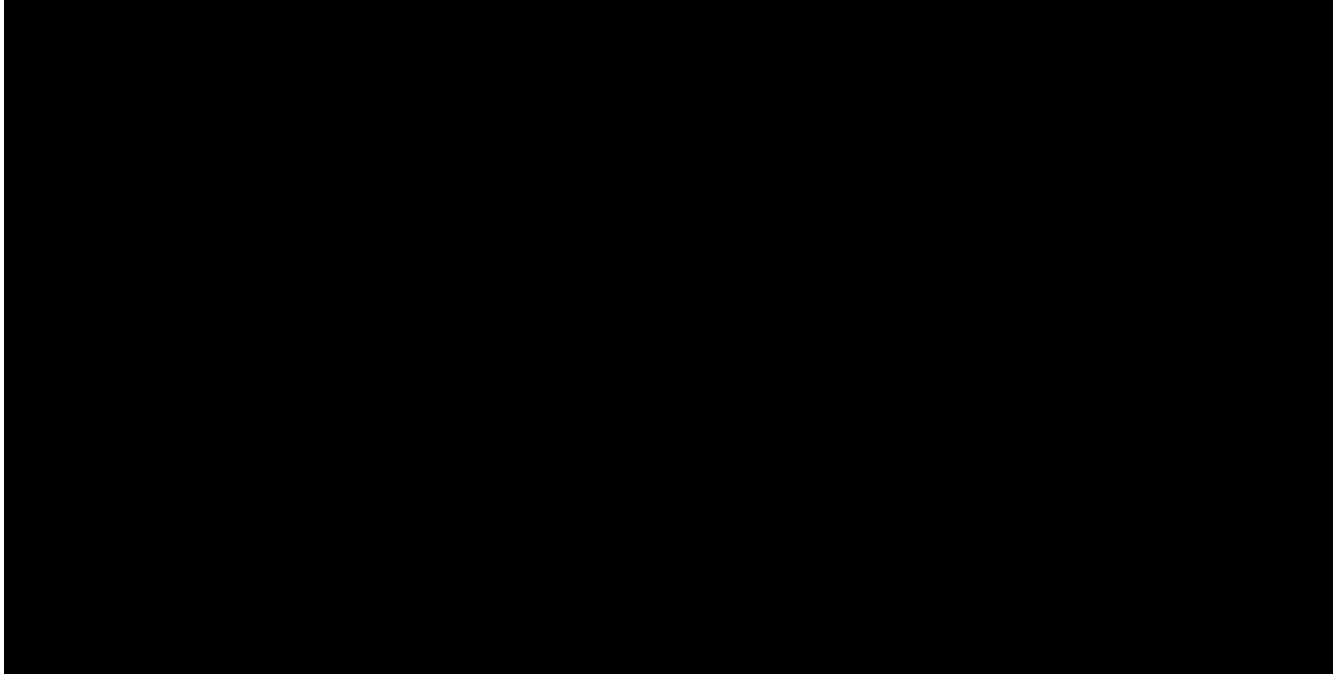
Ajout de la nouvelle [section](#)

A la fin de la précédente section :

```
<section v-if="user">  
  <h2>  
    un extrait de vos chaussures (<RouterLink  
      class="text-red-600 underline"  
      to="/basket"  
    >Toutes les voir</RouterLink  
  >)  
</h2>  
<ListeBaskets class="flex flex-wrap gap-2" :max="3" />  
</section>
```

Ajout de la nouvelle section

Tester



Pour aller plus loin:

- Ajouter le RLS pour que les utilisateurs ne voient et ne puissent modifier que leurs "baskets" !
 - S'assurer que l'id de l'utilisateur soit inséré comme valeur par défaut à la création d'un produit.
- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.
 - Optionnellement, insérer la date dans une colonne dédiée.
- Permettre de supprimer des configurations de produits.

Pour allez plus loin:

- Ajouter le RLS pour que les utilisateurs ne voient et ne puissent modifier que leurs "baskets" !
 - S'assurer que l'id de l'utilisateur soit inséré comme valeur par défaut à la création d'un produit.
- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.
 - Optionnellement, insérer la date dans une colonne dédiée.
- Permettre de supprimer des configurations de produits.

Pour aller plus loin:

- Ajouter le RLS pour que les utilisateurs ne voient et ne puissent modifier que leurs "baskets" !
 - S'assurer que l'id de l'utilisateur soit inséré comme valeur pas défaut à la création d'un produit.

Déjà fait à la création de la table en sql

The screenshot shows the 'Update column' dialog in Supabase. The column name is 'utilisateur' and it is from the 'Basket' table. The description is 'Optional'. The 'Is Primary Key' checkbox is unchecked. A foreign key relation is shown: 'utilisateur' points to 'auth.users.id'. The data type is 'T uuid'. The default value is 'uid()'. The dialog also includes a link to 'Learn more about data types' and a note that the default value can be a literal or an expression like 'uuid_generate_v4()'.

Pour aller plus loin:

- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.

Le booléen à également été inséré à la création de la table SQL!

Pour aller plus loin:

- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.

Il suffit d'ajouter une ligne formkit avec name:"commander"
label:"commander" et y rajouter un checkbox qui permettra de passer
à 'true' dans supabase lorsque l'utilisateur clique dessus, et rester
false si l'on ne coche pas la case avant d'envoyer le formulaire !

Pour allez plus loin:

- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.

Vous êtes perdu ?

Pour aller plus loin:

- Ajouter un Bouton permettant de commander un produit :
 - Changer un booléen pour interdire avec le RLS qu'on puisse changer le produit après l'avoir commandé.

Vous êtes perdu ?

```
<FormKit name="Commander" label="Commander" type="checkbox" />
```

Pour aller plus loin:

- Reste à faire :
 - Optionnellement, insérer la date dans une colonne dédiée.
- Permettre de supprimer des configurations de produits.

Ce tp à été réalisé à la brasserie 1801

