DOCUMENTATION TECHNIQUE

BILL–CUTTING FRONTEND

3PROJ

RYAN DORDAIN / SERAPHIN DUBAIL / TRISTAN TOURBIER / THOMAS LAPERE

Table des matières

1- Choix des technologies :

Pour mener à bien ce projet, nous avons décidé de nous orienter sur le Framework Vue.JS 3 en composition API ainsi que la bibliothèque de composant vue Quasar.

1.1- Avantages :

Vue3 représente des avantages dans sa simplicité syntaxique et sa manière d'organiser le code, les hooks intégrés comme les 'computed' ou les variables réactives avec le mot clé 'ref ' ou encore le 'v-if' représentent aussi un avantage dans l'utilisation de composants réactifs. Vue3 optimise aussi la manière dont les composants sont créés et appelés.

Quasar est une bibliothèque de composants basée sur Vue permettant également d'initialiser un projet avec toutes les dépendances requises pour la création d'une application web complète et offrant une expérience utilisateur optimale.
Quasar étant open source permet aussi d'installer des dépendances supplémentaires en fonction de nos besoins ce qui nous a permis d'implémenter un système de popup ou ne notifications toast très facilement.

Les documentations de Vue3 et Quasar sont très bien fournies ce qui optimise encore le temps passé à développer.

```vue
<template>
    <h1>{{ counter }}, {{ doubleCounter }} </h1>
    <button @click="incrCounter">Click Me</button>
    <button @click="increaseByTwo">Click Me For 2</button>
</template>
<script>
export default {
    data() {
        return {
            counter: 0,
            doubleCounter: 0
        }
    },
    methods: {
        incrCounter: function() {
            this.counter += 1;
        },
        increaseByTwo: function() {
            this.doubleCounter += 2;
        }
    }
}
</script>
```

```vue
<template>
    <h1>{{ counter }}, {{ doubleCounter }} </h1>
    <button @click="incrCounter">Click Me</button>
    <button @click="increaseByTwo">Click Me For 2</button>
</template>

<script setup>
    import { ref } from 'vue'

    let counter = ref(0);
    const incrCounter = function() {
        counter.value += 1;
    }

    let doubleCounter = ref(0);
    const increaseByTwo = function() {
        doubleCounter.value += 2;
    }
</script>
```

Figure 1 Comparaison Option API (Vue2) et Composition API (Vue3)

1.2- Inconvénients :

Vue3 est très récent, et l'utilisation de l'api composition n'est pas encore assez répandu car nouveau pour les développeurs habitués à du Vue2 ou encore du React tout comme Quasar qui est passé dans sa version publique depuis très peu de temps.

Ce qui peut poser problème dans le manque de forums d'entraide sur le sujet dans le cas de problèmes spécifiques

2- Avant utilisation

Afin de pouvoir exécuter le Front de notre solution, vous devrez définir quelques variables d'environnement :

Dans le dossier FrontEnd/Quasar vous trouverez un fichier nommé .env.example
Dupliquez le et renommez le en .env en prenant soin d'y ajouter vos variables.

```
URL_BACKEND =
URL_PROD =
```

- Entrez l'url de votre API pour URL-BACKEND, veillez à ce qu'elle termine bien par '/'
- Entrez l'url de production de votre frontend pour URL-PROD, veillez à ce qu'elle termine bien par '/#/'

3- Fonctionnement général

3.1- Routeur

Cette version de Quasar utilise un routeur 'Hash', ce qui implique que toutes nos routes contiennent un '#' ex : https://bill-cutting.com/#/login

Le routeur se trouve dans src/router/routes.ts

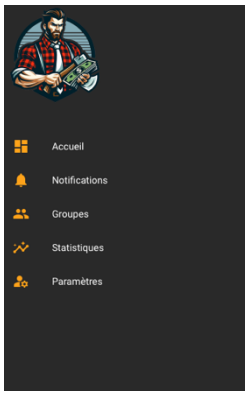
Une route se présente comme ceci :

```
{
  path: '/groups',
  component: () => import('layouts/MainLayout.vue'),
  children: [{ path: '', component: () => import('pages/GroupListPage.vue') }],
},
```

Nous avons :

- Path : url concernée.
- Component : un composant qui est global à toutes les pages, un layout de manière générale avec Quasar qui peut inclure un header etc...
- Children : composant enfant du layout donc la page.

3.2- Main Layout



Le main Layout est un composant vue qui est appelé avant l'instanciation de chaque page, on y importe ici le composant de volet de navigation latéral qui a le rôle d'un header.

La logique qui permet de vérifier si un utilisateur est authentifié se trouve aussi sur ce composant, à l'initialisation de celui-ci, il va vérifier si un token est bien disponible dans le session storage, si aucun token n'est présent, l'utilisateur sera redirigé vers la page de login.

Une vérification exceptionnelle s'effectue si l'utilisateur est arrivé sur la page concerné dans le but de rejoindre un groupe, si celui-ci n'est pas authentifié, on sauvegarde le token d'invitation du groupe dans un cookie pour le réutiliser une fois connecté.

```
async function intilalizeData(){
  try {
    const tokenGroup = route.params.token;
    const token = sessionStorage.getItem( key: 'userToken');
    if (!token){
      if(tokenGroup){
        setCookie( name: 'join-group', tokenGroup,  minutes: 5);
      }
      router.push('/login');
    }
    else {
      isLog.value = true;
    }
  } catch (error) {
    router.push('/login');
    console.error(error);
  }
}
```

Pour les pages qui peuvent accéder au site sans être connectés, (Login, Register etc..) on utilise le layout EmptyLayout qui ne contiens pas toute la logique de gestion de token et le panneau latéral.

4- Communication avec l'API

4.1- Axios

Quasar permet d'initialiser plusieurs dépendances lors de la création d'un projet, nous avons donc utilisé Axios afin de simplifier notre communication avec l'api.

Nous avons par la suite créé des variables globales permettant d'optimiser nos requêtes.

Nous y configurons également un interceptor qui va se charger d'envoyer le token utilisateur à chaque appel.

Nous pouvons ensuite requêter notre API avec un simple api.get

5- Stores

5.1- Présentation

L'un des avantages de Vue est l'utilisation des Stores, ils permettent de gérer des états ou des variables globales de l'application, ils contiennent des getters et des setters pour pouvoir traiter proprement le contenu qu'ils contiennent. Nous les utilisons par exemple pour gérer l'utilisateur connecté, les notifications ou encore pour des fonctions qui peuvent être appelés à plusieurs endroits du code.

5.2- UserStore

Le store utilisateur peut être retrouvé dans src/stores.
Il contient plusieurs fonctions permettant de récupérer l'utilisateur actuellement connecté, le mettre à jour si besoin ou le déconnecter.

L'utilisateur est vide par défaut, si un de nos composants à besoin d'une information sur l'utilisateur, il appellera la fonction getUser().

Cette fonction va vérifier si l'objet user initialisé avec les valeurs par défaut contient un id, si il n'en contient pas on appelle la fonction getUserData() qui va se charger d'acquérir les données.
Les données de l'utilisateur sont généralement perdus au refresh de la page mais restent durant la navigation.

```
async function getUserData(): Promise<void> {
  const token : string | null  = sessionStorage.getItem( key: 'userToken');

  if (!token) {
    console.error("Pas de token");
    return;
  }

  try {
    const payload : string  = token.split( separator: '.')[1];
    const decodedPayload = JSON.parse(atob(payload));

    const response : AxiosResponse<any, any>  = await api.get( url: `users/${decodedPayload.userId}`,  config: {
      headers: {
        Authorization: `Bearer ${token}`
      }
    });

    const userData = response.data;
    user.value.id = decodedPayload.userId;
    user.value.firstname = userData.firstname;
    user.value.lastname = userData.lastname;
    user.value.username = userData.username;
    user.value.email = userData.email;
    user.value.birth_date = new Date(userData.birth_date).toISOString().split( separator: 'T')[0];
    user.value.profile_picture = userData.profile_picture;
  } catch (error) {

    console.error("Erreur lors de la récupération des données utilisateur :", error);
    disconnectUser();
  }
}
```

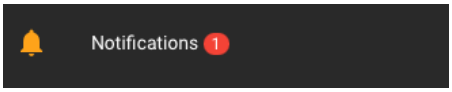
La fonction getUserData() permet d'acquérir les données d'un utilisateur grâce au token obtenu lors de la connexion. Ce token va être parsé afin d'obtenir l'id de l'utilisateur et pouvoir requêter l'api pour récupérer les informations de l'utilisateur.
La fonction prend également en charge le cas où le token est inexistant ou périmé ce qui va appeler la méthode de déconnexion qui supprimera le token du local storage.
Ce qui conduira à un retour sur la page de connection.

5.3- Autres stores

Nous avons également un store de notifications et de groupe qui fonctionnent sur le même principe.

```
async function getNotificationCounts() : Promise<void>  {
  try {
    const response : AxiosResponse<any, any>  = await api.get( url: 'notifs/count');
    count.value = response.data.count;
  } catch (error) {
    console.error(error);
  }
```

Par exemple cette fonction qui permet de récupérer le nombre de notifications non-lues qui sera affiché dans une bulle sur le volet latéral.



Ou cette fonction permettant de récupérer les informations sur un membre en particulier dans un groupe :

```
function getUserGroupData(id:number){  Show usages
  return group.value.Users.find(user => user.id == id);
}
```

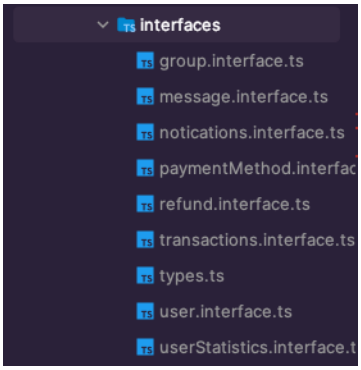
Pour les fonctions plus universelles, nous avons créé un store global contenant par exemple cette fonction qui permet de prendre une somme et de la formatter afin d'avoir un rendu plus propre à l'écran.

```
export function formatNumber(num) : string  {
  const roundedNum : number  = Math.round( x: (num + Number.EPSILON) * 100) / 100;
  const formattedNum : string  = roundedNum.toFixed( fractionDigits: 2);
  return formattedNum;
}
```

Cette fonction va transformer 30 en 30.00 mais aussi 3.3333333333 en 3.33 afin de limite le plus possible les valeurs sortant des calculs.

6- Interfaces

6.1- Présentation



Pour optimiser notre utilisation de TypeScript, nous utilisons des interfaces. Les interfaces permettent de définir à quoi doit ressembler un objet, on y retrouve une liste des différentes variables ainsi que leur types, ce qui permet derrière de faciliter l'utilisation des différentes fonctions et autres car nous savons toujours ce qui doit être retourné ou envoyé dans une fonction, ceci permet également une meilleur maintenabilité du code. Elles peuvent être retrouvés dans le dossier src/interfaces.

On retrouve une interface pour presque tous les objets renvoyés par l'API.
Voici un exemple avec l'interface utilisateur

```ts
export interface IUser {
  id?: number
  username?: string;
  email?: string;
  firstname?: string;
  lastname?: string;
  birth_date?: Date;
  token?: string;
  profile_picture?: [string];
}

export function DefaultUser(): IUser {
  return {
    id: 0,
    username: '',
    email: '',
    birth_date: new Date(),
    firstname: '',
    lastname: '',
    token: '',
  };
}
```

On y retrouve l'interface avec toutes les variables et les types de celle-ci.

On y créé également une fonction avec des valeurs par défaut qui sont ensuite hydratés avec un retour API ou des actions utilisateurs.

Par exemple le formulaire d'inscription est initialisé de la sorte afin d'avoir des valeurs vides par défaut et d'avoir les différents v-model de chaque champ relié à une variable déjà existante.

7- Composants généraux

L'un des principaux avantages de Vue réside dans sa structuration basée sur les composants, nous avons utilisé la bibliothèque de quasar et nos propres composants afin d'optimiser le fonctionnement général du site.

7.1- Composants Quasar

Les composants Quasar font partie intégrante du projet, voici les principaux composants utilisés :

7.1.1- Q-Input

```vue
<q-input
  class="input"
  outlined
  v-model="userName"
  icon="user"
  label="Nom d'utilisateur"
  dark
  color="secondary">
  <template v-slot:prepend>
    <q-icon name="person"/>
  </template>
</q-input>
```

Un des principaux composants utilisés est le composant <q-input/> on peut lui ajouter plusieurs variables qui dirigeront son comportement et un v-model qui va lier celui-ci à une variable du code.
L'input ci-contre sers à entrer le nom d'utilisateur sur l'écran de connexion, on y ajoute des icones par exemple mais la personnalisation peut aller beaucoup plus loin en ajoutant des règles sur la valeur du model.
Voici la documentation complète des inputs pour en savoir plus : https://quasar.dev/vue-components/input#input-types

7.1.2- Q-btn

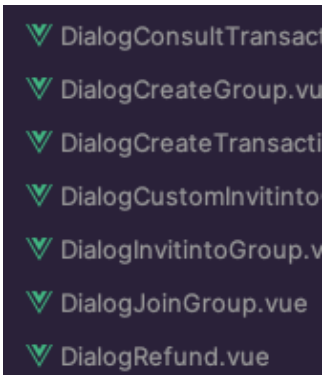
```
<q-btn
  class="btn"
  color="secondary"
  text-color="white"
  unelevated
  label="Se connecter"
  type="submit"
  :loading="loading"
/>
```

Le bouton à un fonctionnement similaire à celui des inputs avec ses petites spécifités comme le statut loading dans l'exemple ci-contre qui va désactiver le bouton et y afficher un spinner.
L'attribut loading est relié à une variable qui passe sur true au moment ou une certaine fonction est appelée, dans le cas présent lorsque la fonction de connexion est appelée.
Voici tout ce qui peut être fait par un bouton quasar :
https://quasar.dev/vue-components/button#progress-related

7.1.3- Q-Dialog

Nous avons beaucoup utilisé les boites de dialogue popup, elles permettent d'être appelés à n'importe quel endroit de l'application sans perturber l'utilisateur. C'est par exemple le cas du popup de création d'un groupe qui peut être aussi bien appelée sur la page d'accueil que sur la page 'Groupes'.

L'appel d'une fonction permet d'ouvrir ou de fermer cette boite de dialogue, nous pouvons lui passer des variables en paramètres ou extraire des données qui ont pu y être entrés.




```
function openDialogCreate() {  Show usages
  dialogCreate.value = true;
  $q.dialog( opts: {
    component: DialogCreateGroup,
    componentProps: {
      isOpen: dialogCreate,
    },
  }).onDismiss( callbackFn: () => {
    getGroups();
  });
}
```

Voici un exemple d'appel de popup avec une ecouteur sur l'évenement onDismiss qui correspond à la fermeture du popup qui va actualiser la liste des groupes dans le cas présent.

7.1.4- Variables de style globales

Quasar permet d'utiliser d'autres variables que vous pourrez souvent comme 'q-pa-md' qui ajoute un padding de taille moyenne à l'élément sur lequel il est ajouté dans les classes. La variable 'q-mx-auto' permet d'aligner au centre un élément, il remplace un {margin : 0 auto}

Les couleurs de l'application sont elles aussi définies dans des variables stockées dans le fichier src/css/quasar.variables.css

```scss
$primary   : #292929;
$secondary : #ffa31a;
$accent    : #808080;

$dark : #131313;

$dark-page : #141332;

$positive  : #21BA45;
$negative  : #dc0707;
$info      : #31CCEC;
$warning   : #F2C037;
```

Cette utilisation permet de modifier facilement la palette de couleurs du front.

7.2- Composants personnalisés

Nous avons également eu besoin d'implémenter nos propres composants comme sur la page d'accueil par exemple pour le composant indiquant combien a été dépensé ce mois-ci :

Les 2 éléments ci-dessous sont en fait un même composant qui adopte un comportement différent en fonction de comment il a été instancié.

```vue
onMounted( hook: async () => {
  User.value = await getUser()
  try {
  loading.value = true;
    if (props.etat !== EtatTotalPaidComponent.Positive) {
      displayColor.value = 'red';
      titleText.value = 'Reste à payer';
      icon.value = 'north_east';
      const response = await api.get( url: `users/${User.value.id}/transactions/totalBalance`);
      montantTotal.value = response.data.amount;
      operations.value = -1;

    } else {
      displayColor.value = 'green';
      titleText.value = 'Total payé ce mois-ci';
      icon.value = 'south_west';
      const response = await api.get( url: `users/${User.value.id}/transactions/thisMonth`);
      montantTotal.value = response.data.amount;
      operations.value = response.data.transactions;
    }
    loading.value = false;
  } catch (error) {
    console.error("Une erreur s'est produite lors de la récupération des données :", error);
  }
```

Les variables sont ensuite récupérées à l'api en fonction de l'état passé en props.

Tous les composants de la page d'accueil ont un comportement similaire

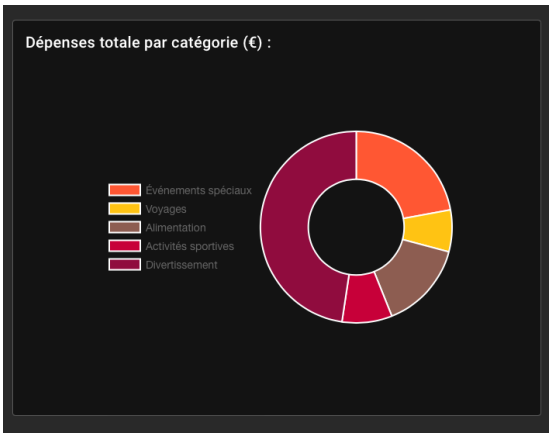
8- Statistiques

Pour les statistiques, nous avons décidés d'utiliser la librairie ChartJs pour l'affichage des statistiques.

Nous récupérons tous les statistiques de l'utilisateur depuis l'API puis nous les ajoutons à des composants Doughnut ou des composants Bar en ayant défini des préréglages de forme de couleurs et de forme au préalable, le tout dans un composant q-card quasar.

```
<q-card  flat dark  bordered class="card-chart">
  <div class="q-pa-md q-mx-auto">
    <q-item-label class="text-h6">Quantité de dépenses par catégorie :</q-item-label>
    <Doughnut class="q-mx-auto chart" :data="chartCountCatdata" :options="options"/>
  </div>
</q-card>
```

Voici le résultat obtenu :



Il suffit de passer le curseur sur une partie du diagramme pour afficher les chiffres qui y correspondent.