TP VueJS 3 - Front du projet API Spring Boot

Avant toute chose...

Dans votre controller de votre projet SpringBoot :

- Ajouter : `import org.springframework.web.bind.annotation.CrossOrigin;` dans vos imports
- Ajouter, sous chaque annotation XXXMapping, la ligne `@CrossOrigin(origins = "*")`

Cela permettra au serveur VueJS d'accéder sans problème à l'API.

Fichiers à installer

- NodeJS : https://nodejs.org/en/download/
- VueCLI : dans un cmd, tapez :

```
npm install -g @vue/cli
```

Télécharger le projet template

- Télécharger le zip fourni
- Lancer un cmd dans le dossier du projet
- Tapez `npm install`

Le serveur est maintenant prêt à être utiliser. Pour le démarrer, il faudra taper dans la cmd, toujours dans le dossier, la commande `npm run serve` (lance un serveur en mode développement).

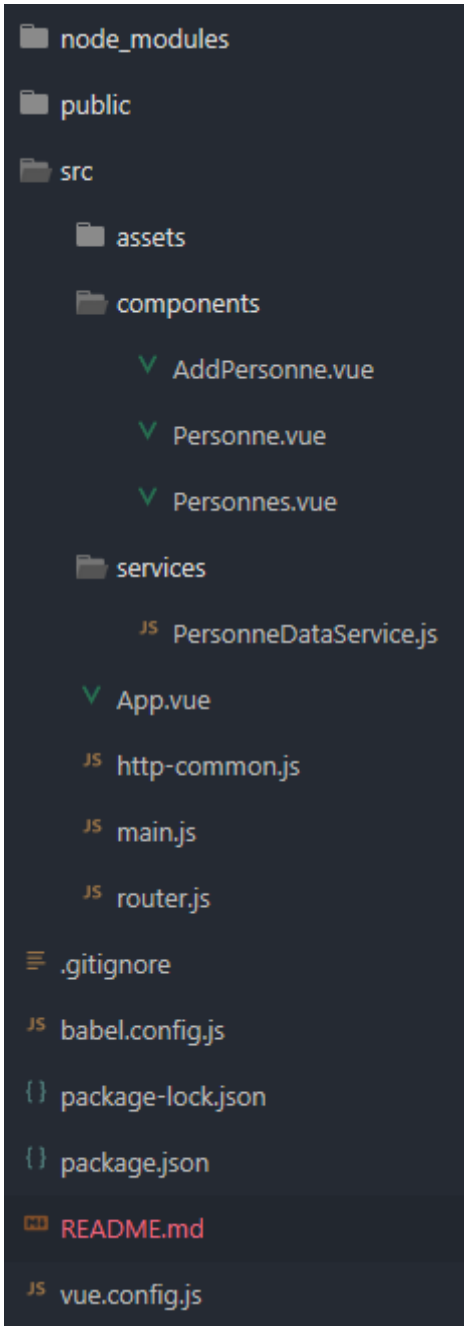
Contenu du projet fourni

- Le composant `App` sert de container pour l'application. Il contient le menu qui redirige vers les différentes pages, et affichera le contenu des autres pages.
- `Personnes` récupère et affiche la liste des personnes de l'annuaire.
- `Personne` permet d'éditer une personne selectionnée (choix grâce à l'ID)
- `AddPersonne` permet de créer une nouvelle personne
- Chaque composant appelle le service `PersonneDataService` qui contient les méthodes utilisant la libraire `axios` qui éxécutera les requêtes HTTP et recevra les réponses.

Librairies pré-installées

- vue 3 : Framework Javascript utilisé pour notre application
- vue-router 4 : Permet de relier chaque composant à une route
- axios 0.21.1 : Gestion des requêtes HTTP
- bootstrap : Librairie graphique vue en première séance

Structure du projet



- package.json qui contient les déclarations des 4 librairies citées précédemment : vue, vue-router, axios, bootstrap.
- Les composants Personne, Personnes et AddPersonne.
- router.js qui définit les routes pour chaque composant
- http-common.js qui initialise axios avec le lien de l'API et les entêtes.
- PersonneDataService contient les méthodes pour envoyer les requêtes HTTP vers l'API.
- vue.config.js permet de configurer le port sur 8081 (ou autre) : en effet, VueJS et SpringBoot utilisent le même port, il faut donc les différencier.

Initialisation du router

Dans le fichier `router.js`, on retrouve un exemple de route.
Chaque route est définie par :

- path: L'URL sur laquelle on va taper
- name: nom de la route
- component: composant relié à la route

Dans `src/main.js`, on importe le router à notre application (voir ligne 7)

```
createApp(App).use(router).mount('#app')
```

Finalisation du menu dans App.vue

La navbar est fournie, mais elle peut être modifiée par vos soins.
La seule chose qu'il reste à faire, c'est d'ajouter en lignes 4, 7, et 10 les liens vers vos routes. (4 et 7 sont identiques pour moi)

On définit la navbar dans ce fichier car c'est le container principal de notre application, c'est dans cette page que les autres composants vont s'afficher.

Initialisation d'Axios

L'initialisation d'Axios se passe dans le fichier `http-common.js`. Il vous suffit, pour ce TP, de modifier `baseURL` avec votre lien (normalement http://localhost:8080/).

Création du service PersonneDataService

Notre service utilisera axios, initialisé précédemment, pour envoyer des requêtes HTTP.
Pour cela, dans le fichier `PersonneDataService.js`, on doit réaliser l'import de notre fichier `http-common.js`.

Une fois réalisé, on peut définir nos fonctions qui appellent l'API.
Ci-dessous, vous trouverez les 3 premières fonctions à mettre en place (déjà présente dans le fichier) :

```
//Fonction pour récupérer l'ensemble des personnes (GET)
getAll() {
    return http.get("/entree");
}

//Fonction pour récupérer une personne via son ID (GET)
get(id) {
    return http.get(`/entree/${id}`);
}

//Fonction pour créer une nouvelle personne (PUSH)
//data représente les données du formulaire (nom, prénom, téléphone, ville)
create(data) {
    return http.post("/entree", data);
}
```

En se basant sur ces fonctions, il ne vous reste plus qu'à faire les deux fonctions `update` et `delete` pour avoir l'ensemble des fonctionnalités développées sur SpringBoot.

Création des composants

Un composant, c'est ni plus ni moins qu'une page HTML.
La seule différence, c'est qu'au lieu de commencer par `<html>`, le composant commence par `<template>`. En effet, la balise html est déjà présente dans App.vue, et comme indiqué précédemment, c'est App.vue qui sert de container. Donc, on vient simplement charger le template Personnes dans App.
Dans App, en ligne 16, vous retrouvez `<router-view />`. C'est cette balise qui sait, en fonction de l'URL et donc de votre route, quel composant elle doit afficher.

> Petit conseil avant de vous lancer de la création des vues finales : Faites d'abord le fonctionnel !
> Il est plus important que l'application soit fonctionnelle plutôt que jolie. Un truc beau qui marche pas, c'est pas fort utile..
> Donc focus sur le fonctionnel, et après on fait des trucs beaux ! ;)

Personnes.vue

Ce composant permet de lister l'ensemble des personnes de l'annuaire, et lorsqu'on clique sur une de ces personnes, on affiche ses informations.
Pour cela, il faut donc :

- Faire appel à `getAll()`

Pour afficher la liste des personnes, il faut d'abord écrire un peu de Javascript. Ce code JS permettra de stocker les résultats de l'appel d'API dans une variable qu'on pourra utiliser pour générer la liste.
On importe en premier lieu notre service, et ensuite on vient remplir le code du export.
Dans data(), on vient déclarer toutes les variables nécessaires au fonctionnement de notre page :

- `personnes: []` : tableau qui contiendra le résultat de l'appel d'API
- `currentPersonne: null` : variable qui nous servira lors du clic sur un item de la liste
- `currentIndex: -1` : variable qui nous servira lors du clic sur un item de la liste

Ensuite, dans methods, on vient définir notre fonction pour faire notre appel d'API : `getPersonnes()`.
Cette fonction fait appel à la fonction `getAll` de notre service `PersonneDataService`. Deux cas possibles :

- L'appel d'API est concluant : on entre dans le .then . On stocke le résultat de la requête dans la variable `personnes`.
- L'appel d'API échoue : on affiche l'erreur dans la console du navigateur.

On revient maintenant dans la balise template. A cet endroit, on doit créer notre code HTML avec une liste (balise `<ul>`). Dans cette liste, on doit venir générer nos items `<li>` en fonction des résultats de notre requête.
Pour cela, on vient utiliser du code VueJS.

```
<li class=""
  :class="{ active: id == currentIndex }"
  v-for="(personne, id) in personnes"
  @click="setActivePersonne(personne, id)"
>
    {{ personne.surname }} {{ personne.name }}
</li>
```

Détaillons tout cela :

- l'attribut `class` : un grand classique, c'est là où on définit nos classes Bootstrap
- l'attribut `:class` : c'est déjà autre chose.. En effet, on entre dans le vif du sujet. C'est avec cet attribut qu'on pourra définir la couleur sur l'élément sélectionné.
  On précise ici que l'on souhaite rajouter la classe `active` si et seulement si id == currentIndex.
  Qu'est ce que c'est `id` et `currentIndex`? `id` correspond à l'ID de votre personne renvoyé

par l'API (chaque personne en possède un), et `currentIndex` correspond à l'index d'item de la liste sur lequel on clique.
- v-for : C'est la boucle qui va permettre de générer chaque item de la liste avec les informations d'une personne.
- @click : C'est l'attribut qui nous permet de définir la personne active et d'afficher ses informations.
- Dans le contenu de chaque item, on vient afficher le nom et prénom de la personne grâce aux balises {{ }}.

Venons en au fonctionnement de setActivePersonne et de :class.
Souvenez-vous un peu plus haut, on définit par défaut currentIndex à -1, et currentPersonne à null. C'est logique, comme nous n'avons cliqué sur aucun élément de liste, on ne peut donc pas avoir choisi qui que ce soit.
Ce qu'on veut, c'est qu'en cliquant sur un des items de la liste, celui-ci devienne bleu et on affiche les informations détaillées.
Pour cela, on précise sur notre item qu'au clic, on lance la fonction setActivePersonne, et cette fonction va simplement dire que currentPersonne devient la personne sur laquelle on a cliqué, et currentIndex prend comme valeur l'index de la personne sur qui on a cliqué.
De ce fait, id devient égal à currentIndex, la condition de :class est donc respecté, et donc l'item deviendra bleu instantanément.

Maintenant que la liste est fonctionnelle, on doit afficher les valeurs détaillées d'une personne.
Pour les afficher, il faut qu'on s'assure qu'une personne soit sélectionnée.
Pour cela, on utilise un v-if.

```
<div v-if="currentPersonne">
    <!-- On affiche le contenu -->
</div>
<div v-else>
    <!-- On affiche le message indiquant de choisir une personne -->
</div>
```

Grâce à ce v-if, on vérifie que currentPersonne soit différent de `null` ou pas :

- currentPersonne différent de `null` : on affiche les informations grâce aux balises {{ }}
- currentPersonne est égal à `null` : on affiche le petit message d'explications

On trouve enfin une balise `<router-link>` qui génère un bouton vers la route d'édition d'une personne.

Voilà :) Le principal est expliqué !
J'ai été gentil, je vous ai mis le principal du code pour cette page, il ne vous reste plus qu'à faire l'interface graphique !

A vous de jouer

Les règles sont simples.
Afin d'utiliser le TP SpringBoot, il faut créer en VueJS une application permettant d'intéragir avec l'API.

Il s'agit de réaliser des interfaces simples et fonctionnelles.
Vous êtes libre sur les choix graphiques, il est demandé d'utiliser correctement Bootstrap et de bien indenté son code pour qu'il soit lisible.

Pour la vue `Personne.vue` (édition / suppression):

- Compléter les fonctions
  - `getPersonne` : récupération d'une personne par son ID
    - En cas de réussite, collez cette ligne `this.currentPersonne = response.data;`
  - `updatePersonne` : modification d'une personne via l'envoi du formulaire
    - En cas de réussite, collez cette ligne : `this.message = 'Personne modifiée avec succès!';`
  - `deletePersonne` : suppression d'une personne par son ID
    - En cas de réussite, collez cette ligne `this.$router.push({ name: "ROUTE_VERS_LISTE_PERSONNES" });`
- Créer le formulaire HTML d'édition d'une personne
  - Vérifier que currentPersonne soit différent ou égal à `null`
  - Créer la structure du formulaire si currentPersonne est différent de `null`
    - Chaque input sera de cette forme : `<input type="text" class="..." id="..." v-model="currentPersonne.XXX" />` (XXX correspondant à la variable à afficher)

Pour la vue `AddPersonne.vue` (ajout)

- Compléter la fonction `creerPersonne`
  - dans `var data`, ajouter tous les champs nécessaires en se basant sur l'exemple fourni sur id.
  - sur le principe de la fonction `update` créée auparavant, réaliser l'appel d'API permettant de créer une personne
    - En cas de réussite, collez les lignes `console.log(response.data);` `this.submitted = true;`
- Créer ensuite le formulaire
  - /!\ : Ne pas utiliser de balise `form`, mettre les input directement dans des div !
  - Chaque input prendra cette forme :

```
<input
    type="text"
    class="..."
    id="XXX"
    required
    v-model="personne.XXX"
    name="XXX"
  />
```

    - (XXX représentant la variable à compléter dans le formulaire)

Pour rendre le code, il faudra utiliser Github.
Pour cela, il vous suffit de créer un compte, et de suivre les instructions vues ensemble en cours.