WOOFPAL

DOCUMENTATION

8 février 2024

PROJET INTÉGRATEUR

Flavio Peter Weinstein Silva
Samuel Bouchereau

CONNEXION/ENREGISTREMENT

Les pages de connexion et d'enregistrement sont en réalité sur une seule page (view). On utilise des conditions *v-if* et *v-else-if* pour afficher soit le formulaire de connexion pour les utilisateurs existants, soit le formulaire d'identification initiale pour les nouveaux utilisateurs. La page d'enregistrement comporte également un bouton pour revenir au menu de connexion si l'utilisateur possède déjà un compte enregistré dans la base de données.

```
<div class="login-section ion-padding" v-if="$route.params.page == 'login'">
  <div class="heading">
    <h1>Bienvenue!</h1>
    <p>Connectez-vous pour continuer</p>
  </div>
```

```
<div
  class="login-section ion-padding"
  v-else-if="$route.params.page == 'signup'"
>
  <div class="heading">
    <h1>Créer un compte</h1>
    <p>Offrez une qualité de vie exceptionnelle à votre compagnon</p>
  </div>
```

Nous avons utilisé des fonctions associées aux bouton réactifs pour envoyer et valider les requêtes d'authentification directement dans le script.

```
<div class="action-button ion-padding padding-adjust">
  <ion-button @click="createUserData" size="large" class="login-button"
    >let's go!</ion-button
  >
  <p>
    Déjà inscrit?
    <a href="/home/login" style="text-decoration: underline"
      >Se Connecter</a
    >
  </p>
</div>
```

La première fonction majeure récupère les informations des comptes existant dans l'API MongoDB Atlas dans le cloud pour vérifier si les informations fournies correspondent à un profil déjà présent dans

```
async function getUserData() {
  if (user.name != "" && user.password != "") {
    await fetch("http://localhost:8082/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },

      body: JSON.stringify(user),
    })
      .then((response) => {
        if (!response.ok) {
          toast.error("User doesn't exist");
          throw new Error("Network response was not ok. Maybe API not running");
        }
        return response.json();
      })
      .then((data) => {
        user.typeOfAnimal = data.typeOfAnimal;
        user.breed = data.breed;
        user.weight = data.weight;
        user.age = data.age;
        user.weeklyActivity = data.weeklyActivity;
        user.lastVisitVet = data.lastVisitVet;
        user.lastVisitToi = data.lastVisitToi;
        user.medicalCondition = data.medicalCondition;

        storeUserInfo();

        clearUser();
      });
  } else {
    toast.warning("Login: Either name or password are missing");
  }
}
```

la base de données. Si ce n'est pas le cas, une notification *toastification* nous informe que les clés d'authentification sont invalides. Si les informations correspondent, nous sommes redirigés vers la page accueil de l'application qui présente les fonctionnalités et leur fonctionnement.

La seconde fonction asynchrone de la page de connexion est la fonction chargée d'enregistrer les informations rentrées par l'utilisateur lors de son inscription. Elle enregistre l'utilisateur dans la base de données MongoDB Atlas et redirige l'utilisateur vers la page de connexion pour pouvoir procéder à une nouvelle authentification. La procédure de connexion est alors la même que pour les utilisateurs existants. Nous avons également ajouté une notification *toast* pour confirmer la création du compte.

Les données sont enregistrées dans le local storage par la fonction ci-dessous. Nous avons ajouté des fonctions pour effaces les champs de saisie de texte lorsqu'une requête a été complétée.

```
function storeUserInfo() {
  localStorage.setItem("user", JSON.stringify(user));
}

function clearUser() {
  user.name = "";
  user.password = "";
}
```

```
async function createUserData() {
  if (newUser.name != "" && newUser.password != "") {
    await fetch("http://localhost:8082/signup", {
      method: "POST",
      headers: { "Content-Type": "application/json" },

      body: JSON.stringify(newUser),
    }).then((response) => {
      if (!response.ok) {
        throw new Error("Network response was not ok. Maybe API not running");
      }
      toast.success(`User ${newUser.name} created successfully`);
      clearNewUser();
    });
  } else {
    toast.warning("SignUp: Either name or password are missing");
  }
}
```

TABLEAU DE BORD

```
<ion-item class="section-title">Informations</ion-item>
<ion-item class="alert-container">
  <ion-input
    class="ion-text-end"
    label="Nom du chien"
    v-model="newUser.name"
  ></ion-input>
  <ion-input
    class="ion-text-end"
    label="Race"
    v-model="newUser.breed"
  ></ion-input>
  <ion-select class="ion-text-end" label="Poids" v-model="newUser.weight">
    <ion-select-option
      v-for="weightValue in weightOptions"
      :key="weightValue"
      :value="weightValue"
    >
      {{ weightValue }} lbs
    </ion-select-option>
  </ion-select>
  <ion-select
    class="ion-text-end"
    label="Âge"
    type="number"
    v-model="newUser.age"
  >
    <ion-select-option
      v-for="ageValue in ageOptions"
      :key="ageValue"
      :value="ageValue"
    >
      {{ ageValue === 1 ? "1 an" : ageValue + " ans" }}
    </ion-select-option>
```

La fonctionnalité principale du tableau de bord étant les niveaux d'activité n'est pas encore reliée à un horodateur, ce qui rend la fonctionnalité incomplète. Cependant, la section « Informations » est opérationnelle et interagit avec les données enregistrées au compte de l'utilisateur connecté. Lorsqu'on modifie les champs *ion-input*, en appuyant sur le bouton d'enregistrement, les données sont automatiquement sauvegardées pour le programme du ChatBot. Il est possible de procéder ainsi grâce à des *v-model* ainsi qu'une fonction *storeUserInfo()* qui enregistre les modifications.

```
function storeUserInfo() {
  localStorage.setItem("user", JSON.stringify(user));
}

async function getUserData() {
  if (newUser.name != "" && newUser.password != "") {
    await fetch("http://localhost:8082/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },

      body: JSON.stringify(newUser),
    })
      .then((response) => {
        if (!response.ok) {
          toast.error("User doesn't exist");
          throw new Error("Network response was not ok. Maybe API not running");
        }
        return response.json();
      })
      .then((data) => {
        newUser.typeOfAnimal = data.typeOfAnimal;
        newUser.breed = data.breed;
        newUser.weight = data.weight;
        newUser.age = data.age;
        newUser.weeklyActivity = data.weeklyActivity;
        newUser.lastVisitVet = data.lastVisitVet;
        newUser.lastVisitToi = data.lastVisitToi;
        newUser.medicalCondition = data.medicalCondition;

        storeUserInfo();

        clearNewUser();
      });
  } else {
    toast.warning("Login: Either name or password are missing");
  }
```

CHATBOT

```
onIonViewWillEnter(async () => {
  restoreUserInfo();
  messages.value.push({
    role: "user",
    content: `Meet ${user.value.name}, a ${user.value.breed} weighing ${user.value.weight} pounds and aged ${user.value.age} years old. ${user.value
  });
});
```

Le Chatbot utilise un API fournit par OpenAI. Ce dernier envoie des requêtes avec un prompt personnalisé d'abord, puis la question entrée dans le champ de saisie de texte. Cela permet d'informer constamment l'assistant des détails pertinents concernant l'animal concerné, sans que l'utilisateur n'ait besoin de réécrire en permanence ces informations. Il s'agit d'une assistant personnel conçu pour garder en mémoire ces informations pour conseiller de façon plus éclairée.

Le prompt envoyé initialement prend en compte tous les détails sensibles mentionnés par l'utilisateur. Ces détails sont récupérés directement à partir de la base de données. Les composants *ion-select* assurent un formatage des données et donc facilite leur exploitation par le reste des programmes de l'application.

```
const sendMessage = async () => {
  messages.value.push({
    role: "user",
    content: currentMessage.value,
  });

  try {
    const answer = await fetch("http://localhost:8083/chatbot", {
      method: "POST",
      headers: { "Content-Type": "application/json" },

      body: JSON.stringify({
        chatHistory: messages.value,
        userInput: currentMessage.value,
      }),
    }).then((response) => {
      return response.json();
    });
    messages.value.push({
      role: "assistant",
      content: answer.content,
    });
    currentMessage.value = "";
  } catch (error) {
    console.error("Error sending message:", error);
  }
};
```

CARTE

```
async function placeMarkers() {
  const loading = await loadingController.create({
    message: "Attendre SVP. On cherche les endroits...",
  });

  await loading.present();

  const apiUrl = `http://localhost:8080/locations?category=${category.value}&latitude=${lat.value}&longitude=${lon.value}`;
  await fetch(apiUrl)
    .then((response) => {
      if (!response.ok) {
        throw new Error("Network response was not ok");
      }
      return response.json();
    })
    .then((data) => {
      markerRefList.value.forEach((m) => m.remove());
      markerRefList.value = [];
      console.log(data);
      data.forEach(async (e) => {
        const formattedAddress = e["address"]
          .replace("E,", "Est")
          .replace("O,", "Ouest")
          .replace("St", "")
          .split(" ")
          .slice(0, e["address"].split(" ").length - 2)
          .join(" ");
        console.log(encodeURIComponent(formattedAddress));
        const locationsJsonList = await fetch(
          `http://localhost:8080/location/${encodeURIComponent(
            formattedAddress
          )}`
        )
          .then((response) => response.json())
          .catch((error) => console.error("Address location not found", error));
```

Le programme envoie également une requête mais à un API différent, celui d'OpenWeatherMap, permettant d'obtenir la géolocalisation de l'utilisateur ainsi que de fournir une carte interactive avec multiples fonctionnalités intégrées. La carte est vierge lorsqu'on arrive d'abord sur la page. Lorsqu'on sélectionne une catégorie, une requête est envoyée avec les données de longitude et latitude actuelles de l'utilisateur pour récupérer les lieux correspondants à proximité, c'est-à-dire dans les délimitations graphiques de la carte affichée.

Lorsque les emplacements ont été récupérés, une fonction affiche alors tous les détails pertinents sur le lieu dans une fenêtre s'ouvrant uniquement si l'utilisateur appuie sur le lieu en question. Le programme comporte une fonction chargée de recalculer la position de l'utilisateur à chaque intervalle de 30 secondes.

```
console.log("this is", locationsJsonList);

if (
  locationsJsonList.hasOwnProperty("lat") &&
  locationsJsonList.hasOwnProperty("lon")
) {
  // Object has both lat and lon properties
  const { lat, lon } = locationsJsonList;

  const marker = L.marker([lat, lon])
    .addTo(map.value)
    .bindPopup(
      `<img src="${e["imageUrlNeeded"]}" style="height: 25vh; width: 30vw;"/>
      <br/>
      ${e["establishmentName"]}
      <br/>
      ${e["address"]}
      <br/>
      ${e["closingTime"]}
      <br/>
      ${e["phoneNumber"]}
      <br/>
      ${e["status"]}
      <br/>
      ${e["rating"]}
      <br/>
      ${e["comments"]}`
    );
  markerRefList.value.push(marker);
} else {
  // Object is missing either lat or lon properties
  console.log("Object is missing either lat or lon properties.");
}
});
```

MENU DÉROULANT

À la suite de l'authentification, un menu déroulant devient disponible à l'utilisateur à la gauche de l'écran. Il affiche les onglets représentant les fonctionnalités de l'application ainsi que la page d'accueil présentant cette dernière. Ce menu fait partie des composants externes tout comme le *header* étant chargé d'afficher les titres des pages en tout temps.

Pour l'ensemble du projet nous avons utilisés des composant ion pour faciliter l'intégration. Les librairies ion nous ont particulièrement aidées pour organiser les divers éléments, telles qu'*ion-grid, ion-icon* et *ion-select*.

```
<template>
  <ion-header :translucent="true">
    <ion-toolbar>
      <ion-buttons slot="start">
        <ion-menu-button color="primary"></ion-menu-button>
      </ion-buttons>
      <ion-title>{{ $route.name }}</ion-title>
    </ion-toolbar>
  </ion-header>
</template>
```

```
const selectedIndex = ref(0);
const appPages = [
  {
    title: "Accueil",
    url: "/intro",
    iosIcon: chatboxOutline,
    mdIcon: homeSharp,
  },
  {
    title: "ChatBot",
    url: "/chatbot",
    iosIcon: chatboxOutline,
    mdIcon: chatboxSharp,
  },
  {
    title: "Tableau de bord",
    url: "/dashboard",
    iosIcon: clipboardOutline,
    mdIcon: clipboardSharp,
  },
  {
    title: "Carte",
    url: "/map",
    iosIcon: mapOutline,
    mdIcon: mapSharp,
  },
];
```