Vue.js

Application de démonstration

- Démo Vue.js (CSR) *(Code source)*

Lien vers la documentation officielle

- Vue.js

Outils

- Vite (compilation)
- Vue Router (routing)
- Pinia (store management)
- Vitest (tests unitaires)
- Vue Test Utils (tests unitaires pour composants Vue)
- Cypress (tests e2e)
- Playwright (tests e2e)

UI

- daisyUI
- shadcn-vue

Data Fetching

- TanStack-Query

Exercices Vue.js

Pour travailler sur ces exercices, vous devez forker le projet de démonstration et créer une nouvelle branche.

https://github.com/opac-teach/vue-demo

Une fois que vous avez ouvert le projet sur votre machine, vous pouvez lancer les commandes suivantes:

```bash
# Installer les dépendances du projet

npm install

# Lancer le serveur de développement
npm run dev
```

Rendu

Une fois votre travail terminé, vous devrez Commit/Push votre travail sur votre fork, et créer une pull request sur le projet d'origine pour valider votre travail.

Exercices de base

Ces exercices sont à commencer dans le composant `src/components/exercices/Exercice.vue`

Fondamentaux

Boucles et conditions

Creer une liste d'articles d'objets à la vente, les afficher, puis n'afficher que ceux dont le prix est supérieur à 50€

Composants

Creer un composant qui affiche un seul article, et l'utiliser dans la liste precedente pour afficher chaque article en le passant en prop.

Styles

Afficher en rouge le prix des objets >100€ et en vert si <100€

Lifecycle

Afficher une alerte lorsque l'utilisateur quitte la page d'exercices

---

Routing

Pages

Creer une nouvelle page et l'ajouter dans le menu de navigation

Layout

Ajouter votre nom dans le Footer

State

Articles en state

- Transformer la liste d'articles en variable reactive
- Transformer le filtre sur les prix > 50€ qui etait fait avec un v-if en une nouvelle variable calculée (computed)

Ajout d'article

Ajouter un formulaire pour ajouter un nouvel article à la liste.

Data fetching et filtres

Récupérer les données sur le endpoint https://api.sampleapis.com/rickandmorty/characters et les afficher.

Creer un menu d'options pour filtrer les personnages affichés ("All", "Human", "Alien")

Global State

Créer un formulaire pour définir un nom d'utilisateur. Stocker celui-ci dans un store pinia, et l'afficher dans le menu de navigation.

Exercices avancés

Page Memecoin

- Créer une nouvelle page `/memecoins`
- Dans cette page, ajouter une section qui affiche une liste de memecoins
  - Récupérer la liste de memecoins depuis l'adresse
  - `https://nuxt-demo-blush.vercel.app/api/get-memecoins`
  - Les afficher dans la page
- Dans cette page, ajouter un formulaire pour créer un memecoin
  - Entrées du formulaire :
    - `name`, 4-16 caractères, obligatoire
    - `symbol`, 2-4 caractères, obligatoire, uniquement des majuscules
    - `description`, 0-1000 caractères, pas obligatoire
    - `logoUrl`, 0-200 caractères, pas obligatoire, doit être une URL valide
  - Envoyer sous forme de requête POST à cette adresse

    - `https://nuxt-demo-blush.vercel.app/api/create-memecoin`
- Gérer la validation des entrées dans l'interface
    - Empêcher l'envoi du formulaire si les conditions ne sont pas respectées
    - Afficher des messages d'erreurs de validation
- Afficher le résultat, et gérer le cas des erreurs API
- Rafraîchir la liste des memecoins affichée
- Bonus : Récupérer et stocker la liste des memecoins dans un store Pinia

---

Authentification

- Ajouter une page d'authentification
    - Créer un formulaire qui demande juste un mot de passe
        - `password` non vide
    - Envoyer ce mot de passe en POST à
        - `https://nuxt-demo-blush.vercel.app/api/login`
        - *(le bon mot de passe est `admin123`)*
    - Stocker le token JWT retourné dans un store Pinia
        - Stocker aussi son userId retourné
        - Injecter le token JWT dans les prochaines requêtes à l'API
        - Bonus: stocker le token dans le localStorage pour qu'il soit chargé si on relance l'app
    - Changer l'URL de la requête pour créer un memecoin vers
        - `https://nuxt-demo-blush.vercel.app/api/create-memecoin-protected`
        - Vérifier que la création fonctionne bien et que l'owner du memecoin créé est bien le nôtre
    - Faire évoluer l'UI en fonction de l'état connecté
        - Si on est connecté :
            - Afficher un bouton pour se déconnecter dans la navbar
            - Si l'utilisateur affiche la page de login, ne pas l'afficher et rediriger ailleurs

- Si on est déconnecté :
  - Afficher un bouton pour se connecter dans la navbar
  - Ne pas afficher le formulaire de création de memecoin, mais un bouton pour se connecter à la place

---

Tests

- Créer des tests unitaires pour :
  - Le(s) store(s) Pinia
  - Le composant qui affiche la liste des memecoins
  - Le formulaire de création de memecoin
- Créer un test d'intégration (e2e) qui va créer un memecoin et vérifier qu'il apparaît bien dans la liste

---

TanStack Query

Remplacer les appels API pour récupérer la liste des memecoins et créer un memecoin par TanStack Query.