(/)

Menu principal

Page d'accueil (/index.php) | Hackathon (/index.php/hackathon)
Documentation (/index.php/menu-docs-sp-323795567) | S1 (/index.php/menu-cours-s1)
S2 (/index.php/menu-cours-s2) | S3 (/index.php/menu-cours-s3) | S4 (/index.php/menu-cours-s4)
S5 (/index.php/menu-lpsil) | Master IoT 1a (/index.php/master-iot-1a)
Wiki mathématique (https://cours-info.iut-bm.univ-fcomte.fr/pmwiki/pmwiki.php)

Connexion

Identifiant

Mot de passe

Se souvenir de moi

Connexion

Identifiant oublié ? (/index.php/component/users/?view=remind&Itemid=125)
Mot de passe oublié ? (/index.php/component/users/?view=reset&Itemid=125)

R4A.10 - Compléments Web (vuejs)

TP n°1 : Heroes & Vilains - axios + vuetify

Détails

Écrit par stéphane Domas
Catégorie : R4A.10 - Compléments web (vuejs) (/index.php/menu-cours-s4/menu-mmi4web)
Publication : 18 décembre 2020
Affichages : 1437

Préambule

- Le but de ce TP est de commencer une SPA en vuejs intitulée "Heroes & Vilains", qui permet de gérer des personnes (plus ou moins gentilles/méchantes) avec des super-pouvoirs, des équipes composées de ces personnes, et des organisations qui fédèrent des équipes.
- Toutes les données sont stockées dans une base de données non relationnelle mongdb, manipulable au travers d'une API proposant diverses routes, décrites ci-dessous.
- La SPA permet de s'exercer à accéder à cette API en utilisant axios et de visualiser les données ainsi récupérées grâce à une IG utilisant vuetify

- Les compétences/connaissances vuejs utiles pour ce TP sont celles qui ont été abordées dans le semestre précédent, plus celles du premier TD.

1°/ Modèle de données

- La BdD a été créée sur un serveur mongodb. Elle définit 3 collections : heroes, teams, organizations. Ces collections sont comme des tables en SQL, excepté qu'elles contiennent des documents dont la structure n'est pas celle du relationnel. Ces documents sont des objets JSON.
- Pour résumer ce que contiennent ces documents :
  - un héro a un nom public, un nom réel, des pouvoirs. Chaque pouvoir a un nom, un type (force, vitesse, endurance, magie, effrayant, furtivité, stupidité) et un niveau de 0 à 100.
  - une équipe a un nom et des membres qui sont des héros.
  - une organisation a un nom, un mot de passe secret et rassemble des équipes.

- Voici un exemple de document tiré de la collection heroes.

```
{ "_id" : ObjectId("614ddafcfda4c31dc9299352"), "publicName" : "super dupond", "realName" : "jean dupor
```

- On remarque qu'avec mongodb, tous les objets, y compris ceux qui sont à l'intérieur d'autres objets, sont référencés par un identifiant, sous la forme d'un ObjectId, qui contient un hashcode unique pour chaque objet de la BdD.

- Pour y accéder à la BdD, l'API utilise l'ORM Mongoose qui définit des schémas qui permettent de décrire la structuration des ces documents.
- Pour décrire les documents de heroes, il y a deux schémas dont le code simplifié est donné ci-dessous :

```
let PowerSchema = new Schema({
  name: {type: String},
  type: { type: Number },
  level: {type: Number}
}
```

```
let HeroeSchema = new Schema({
  publicName: {type: String, required: true},
  realName: {type: String},
  powers: [ PowerSchema ],
});
```

- On remarque que ces schéma correspondent bien aux documents stockés en BdD : un héro a des champs publicName, realName et powers, et ce dernier est bien un tableau contenant des objets dont la structure correspond à PowerSchema, avec des champs name, type, level.
- La seule différence est que les schémas n'ont pas besoin de contenir le champ _id dans tous les objets. C'est normal car mongoose se charge de les ajouter tout seul.

- Pour décrire les documents de teams et de organizations, les schémas sont les suivants :

```
let TeamSchema = new Schema({
  name: {type:String, required:true},
  members: [{type: Schema.Types.ObjectId, ref: 'Heroe'}],
}
```

```
let OrganizationSchema = new Schema({
  name: {type:String, required:true},
  secret: {type:String, required:true},
  teams: [{type: Schema.Types.ObjectId, ref: 'Team'}],
}
```

- Comme on le constate pour les champs members et teams, il est possible de créer des liens entre des documents, dans le genre one-to-many. Par exemple, le champ members est un tableau contenant les ids de documents de la collection heroes. C'est ce qui se rapproche du concept de clé étrangère en SQL.

2°/ l'API

- La racine commune à toutes les routes de l'API est : https://apidemo.iut-bm.univ-fcomte.fr/herocorp (https://apidemo.iut-bm.univ-fcomte.fr/herocorp)
- Quelle que soit la route demandée, l'API répondra avec un objet ayant toujours la même structure : `{error: num_error, status: http_status, data: données}`.
- S'il y a une erreur `num_error` sera >0 et data contiendra le message d'erreur. Sinon, `data` contient la réponse de l'API et son contenu diffère selon la route demandée.

- Ci-dessous, les routes sont décrites avec pour chacune le chemin, le type de requête http, les paramètres ou bien les informations à passer dans le corps de la requête. La description du résultat est celle du contenu du champ data mentionné ci-dessus, quand il n'y a pas d'erreur.

2.1°/ routes pour les héros

- `/heroes/getaliases` [GET] : permet d'obtenir l'identifiant BdD et le nom public de tous les héros.
  - aucun paramètre/corps de requête,
  - le résultat est un tableau contenant des objets au format `{ _id: ..., publicName: ...}`
- `/heroes/create` [POST] : permet de créer un héro
  - le corps de la requête est un objet au format `{ publicName: ..., realName: ..., powers: [ { name: ..., type: ..., level: ...}, {name: ..., type: ..., level: ...}, ... ] }`. Le champ `powers` est optionnel mais s'il est fourni, les objets de ce tableau doivent contenir obligatoirement les champs `name`, `type` et `level`, avec `type` entre 1 à 7 et `level` entre 0 et 100 (cf. section 1)
  - le résultat est un objet contenant la même chose que ce qui a été envoyé, plus l'identifiant _id du héro.
- `/heroes/update` [PUT] : permet de mettre à jour un ou plusieurs champ d'un héro existant
  - le corps de la requête est un objet au format `{ _id: ..., publicName: ..., realName: ..., powers: [ { name: ..., type: ..., level: ...}, {name: ..., type: ..., level: ...}, ... ] }`. Seul le champ _id est obligatoire et doit correspondre à un héro existant.
  - il faut également fournir le mot de passe secret d'une organisation dont le héro fait partie. Pour cela, soit une entête `org-secret` est ajoutée à la requête, soit la route est modifiée comme suivant : `/heroes/update?org-secret=...` (cf. exercices)
  - le résultat est un objet contenant le héro mis à jour.
- `/heroes/getbyid/:id` [GET] : permet de récupérer toutes les informations d'un héro existant
  - le paramètre :id doit être remplacé par l'identifiant d'un héro existant
  - il faut également fournir le mot de passe secret d'une organisation dont le héro fait partie. Pour cela, soit une entête `org-secret` est ajoutée à la requête, soit la route est modifiée comme suivant : `/heroes/getbyid/1234567abcd?org-secret=...` (cf. exercices)
  - le résultat est un objet contenant le héro demandé.

2.2°/ route pour les équipes

- `/teams/get` [GET] : permet d'obtenir la liste des équipes avec pour chacune, son identifiant BdD, son nom et le nombre d'organisations auxquelles elle est affiliée.
  - aucun paramètre/corps de requête,
  - le résultat est un tableau contenant des objets au format `{ _id: ..., name: ..., nbAffiliations: ...}`

- `/teams/create` [POST] : permet de créer une équipe, sans membres.
  - le corps de la requête est un objet au format `{ name: ...}`
  - le résultat est un objet représentant l'équipe créée en Bdd, avec son _id, son nom et un tableau des membres vide.
- `/teams/addheroes` [PATCH] : permet d'ajouter des héros à une équipe
  - le corps de la requête est un objet au format `{ idHeroes: [ ..., ...], idTeam: ...}`. idHeroes contient des identifiant de héros existant, et idTeam celui d'une équipe existante.
  - le résultat est un objet contenant l'équipe mise à jour.
- `/teams/removeheroes` [PATCH] : permet de supprimer des héros d'une équipe
  - le corps de la requête est un objet au format `{ idHeroes: [ ..., ...], idTeam: ...}`. idHeroes contient des identifiant de héros existant, et idTeam celui d'une équipe existante.
  - le résultat est un objet contenant l'équipe mise à jour.

REMARQUE : il n'existe pas de route permettant de récupérer une équipe en particulier avec ses membres. Pour cela, il faut passer par les routes gérant les organisations.

2.3°/ routes pour les organisations

- `/orgs/get` [GET] : permet d'obtenir l'identifiant BdD et le nom de toutes les organisations
  - aucun paramètre/corps de requête,
  - le résultat est un tableau contenant des objets au format `{ _id: ..., name: ...}`
- `/orgs/create` [POST] : permet de créer une organisation avec aucune équipe
  - le corps de la requête est un objet au format `{ name: ..., secret: ...}`. Les deux champs sont une chaîne de caractère.
  - le résultat est un objet représentant l'organisation créée avec une liste d'équipes vide.
- `/orgs/addteam` [PATCH] : permet d'ajouter une équipe à une organisation
  - le corps de la requête est un objet au format `{ idTeam: ...}`. La valeur du champ doit être celle d'une équipe existante.
  - il faut également fournir le mot de passe secret de l'organisation pour laquelle on veut ajouter une équipe. Pour cela, soit une entête org-secret est ajoutée à la requête, soit la route est modifiée comme suivant : `/orgs/addteam?org-secret=...` (cf. exercices)
  - le résultat est un objet contenant l'organisation mise à jour.
- `/orgs/removeteam` [PATCH] : permet de supprimer une équipe d'une organisation
  - le corps de la requête est un objet au format `{ idTeam: ...}`. La valeur du champ doit être celle d'une équipe existante.
  - il faut également fournir le mot de passe secret de l'organisation pour laquelle on veut supprimer une équipe. Pour cela, soit une entête org-secret est ajoutée à la requête, soit la route est modifiée comme suivant : `/orgs/addteam?org-secret=...` (cf. exercices)
  - le résultat est un objet contenant l'organisation mise à jour.
- `/orgs/getbyid/:id` [GET] : permet de récupérer toutes les informations d'une organisation existante
  - le paramètre :id doit être remplacé par l'identifiant d'une organisation existante
  - il faut également fournir le mot de passe secret de cette organisation. Pour cela, soit une entête org-secret est ajoutée à la requête, soit la route est modifiée comme suivant : `/orgs/get/123456789abcd?org-secret=...` (cf. exercices)
  - le résultat est un objet contenant l'organisation demandée, dont le tableau des équipes a été "complété". Cela veut dire qu'il contient des objets représentant des équipes, donc avec les identifiants des membres.

3°/ La SPA

3.1°/ Mise en place initiale

- L'objectif principal de cette application est de mettre en pratique vuetify ainsi que les principes d'accès à une API, notamment en utilisant les connaissances de cours et en s'inspirant de l'exemple fournit avec le cours de SAE (Connexion front & back end avec axios (/index.php/menu-cours-s3/sae-dev-appli-avec-bdd/2529-connexion-front-back-end-avec-axios)).
- Il faut donc avant tout créer un projet vuejs intégrant directement vuex, vue-router, vuetify et installer axios.

3.2°/ Les services

- Il faut créer les fichiers représentant les services, à savoir :
  - `axios.service.js` : contient la création d'une instance personnalisée d'axios, la méthode de traitement des erreurs, et des méthodes générales pour les requête GET, POST, PUT et PATCH.
  - `hero.service.js` : contient les 4 méthodes associées aux 4 routes de l'API pour gérer les héros.
  - `team.service.js` : contient les 4 méthodes associées aux 4 routes de l'API pour gérer les équipes.
  - `org.service.js` : contient les 5 méthodes associées aux 5 routes de l'API pour gérer les organisations.

- Comme précisé dans la description de l'API, certaines routes nécessitent d'envoyer avec la requête la phrase secrète d'une organisation.
- Pour tester facilement l'API, il est possible d'envoyer ce secret directement dans l'URL de la route, grâce aux variables de requête.
- Par exemple, si vous essayez de copier/coller l'URL suivante dans votre navigateur :

```
https://apidemo.iut-bm.univ-fcomte.fr/herocorp/orgs/getbyid/64762e41ad49469af2af0566?org-secret=nous%20
```

- L'API répondra :

```
{"error":0,"status":200,"data":[{"_id":"65b8affe644c2c12d725b2f9","name":"Vilaincorp","secret":"nous so
```

- Bien entendu, ce n'est absolument pas sécurisé d'avoir la phrase secrète dans l'URL.
- Dans le TD 2 sur l'authentification, nous verrons qu'il est possible de passer des informations d'authentification via les entêtes HTTP, grâce au principe d'intercepteur axios. C'est cette méthode qui sera utilisée dans le TP n°2 afin d'être plus "secure" !
- Mais que l'authentification soit sécurisée ou non, ce secret doit être stocké dans le store et mis à jour grâce à une des fonctionnalités de la SPA (cf. ci-dessous).

3.2°/ Le store

- Compte tenu des fonctionnalités exposées dans la section 3.4, il faut au minimum que le store fournisse un state et les actions/mutations associées pour gérer :
  - un mot de passe d'organisation,
  - la liste des alias des héros,
  - un héro dont on a récupéré toutes les informations et que l'on voudrait consulter/modifier
  - la liste des équipes,
  - une équipe que l'on voudrait consulter/modifier,
  - la liste des noms d'organisations,
  - une organisation que l'on voudrait consulter/modifier.

Conseils :

- utilisez des variables du state du type `currentHero`, `currentTeam`, `currentOrg` pour désigner le héro, l'équipe, l'organisation actuellement en cours de visualisation et/ou édition.
- si ces variables valent `null` alors qu'un composant veut les manipuler, cela indique qu'il y a un problème comme par exemple, quand on veut accéder à une organisation sans le bon mot de passe.

3.3°/ Les routes

- Compte tenu des fonctionnalités exposées dans la section 3.4, il faut au minimum que vue-router définisse des routes pour afficher :
  - le composant chargé de l'authentification,
  - le composant chargé de la liste des organisations,
  - le(s) composant(s) chargé(s) d'une organisation particulière,
  - le composant chargé de la liste des équipes,

- le(s) composant(s) chargé(s) d'une équipe particulière,

3.4°/ Les fonctionnalités

- L'apparence et la structuration des pages est laissée totalement libre, à part les contraintes suivantes :
  - utiliser vuetify comme bibliothèque de composants graphiques.
  - la page principale doit contenir un bandeau en haut, avec un icône permettant d'ouvrir un menu type tiroir, et un bouton permettant de s'authentifier (cf. ci-dessous).
  - le menu permet de suivre les différentes routes énoncées ci-dessus

3.4.1°/ phrase secrète

- Un composant doit permettre de saisir la phrase secrète d'une organisation, qui sera stockée dans le store. Cette phrase sera ensuite ajouté dans les entêtes de toutes les requêtes vers l'API (cf. service axios ci-dessus).
- Cela permet d'accéder à une organisation en particulier puisqu'il faut fournir son mot de passe pour la récupérer auprès de l'API.

3.4.2°/ liste des organisations

- Un composant doit afficher la liste des organisations, d'en sélectionner une en particulier, et de créer une nouvelle organisation.
- Pour cela, une fois le composant est monté dans le DOM, il doit utiliser l'action du store permettant de récupérer cette liste auprès de l'API.
- Son template doit donc utiliser les données du store pour créer la liste, par exemple en utilisant un v-data-table.
- En cas de sélection, on utilise la méthode du store pour récupérer les informations de l'organisation choisie puis on suit la route permettant d'afficher une organisation particulière (cf. ATTENTION ci-dessous)
- La création d'une nouvelle organisation doit être déclenchée par un bouton qui ouvre une boîte de dialogue, permettant de saisir le nom et le mot de passe de l'organisation, plus deux boutons permettant de valider ou annuler.
- En cas de validation, on doit utiliser la méthode du store permettant de créer une nouvelle organisation. Le dialogue doit être fermé et on revient à la liste, qui doit bien entendu être mise à jour automatiquement.
- En cas d'annulation, le dialogue se ferme et on revient à la liste.

3.4.3°/ consultation/modification d'une organisation

- Un composant (potentiellement utilisant des sous-composants) doit afficher l'organisation "courante", c.a.d. celle qui a été au préalable sélectionnée via la liste.
- Il est possible que ce composant ne puisse rien afficher, notamment lorsque la requête à l'API pour récupérer les informations a échoué (par ex, mauvais mot de passe) et/ou que `currentOrg` vaut `null`. Dans ce cas, le composant affiche une boite de dialogue signalant qu'il n'y a rien a afficher, avec un bouton Ok. Quand on clique sur le bouton, on retourne à la liste des organisations.

ATTENTION : Dans l'exemple du TD, toutes les actions du store sont écrites comme étant des fonctions asynchrones (mot-clé `async`). Cela permet d'attendre le résultat de la requête axios grâce à await. En revanche, cela rend asynchrone l'action elle-même, ce qui veut dire qu'un appel à cette action va se terminer tout de suite, ... sauf si on utilise await devant. Si une action permet de récupérer les informations d'une organisation, que l'on attend pas le résultat avec `await` et que l'on affiche le composant de consultation d'une organisation, il y a de grande chances que celui-ci considère qu'il n'y a pas d'organisation à afficher et que la boîte de dialogue s'ouvre. Cela sera à tort car c'est simplement la méthode du store qui n'a pas encore reçu la réponse de l'API et mis à jour la variable `currentOrg`. Il faut donc bien appeler la méthode du store avec `await` devant, afin d'attendre vraiment le résultat.

- Sinon, le composant affiche la liste des équipes, avec pour chacune deux boutons, un pour sélectionner, un pour supprimer.
- En cas de sélection, on utilise la méthode du store pour récupérer les informations de l'équipe choisie, puis on suit la route permettant d'afficher une équipe particulière. Attention : quand on récupère une organisation, on a accès aux équipes mais on a seulement les ids des membres de l'équipe. Il faut donc récupérer une à une les informations de chaque membre grâce à leur id.
- En cas de suppression, on affiche une boite de dialogue pour confirmer/annuler la suppression. En cas de confirmation, l'équipe est supprimée de l'organisation grâce à la méthode du store qui s'en occupe.
- Le composant affiche également un bouton permettant d'ajouter une équipe. Quand on clique sur ce bouton, une liste déroulante avec le nom de toutes les équipes recrutables (c.a.d. qui ne sont pas déjà dans l'organisation) apparaît, plus deux boutons : "valider" (inactif par défaut) et "annuler". Quand on sélectionne l'une d'entre elles, le bouton "valider" devient actif et si on clique dessus, on utilise la méthode du store pour ajouter une équipe à l'organisation. La liste doit

donc être mise à jour automatiquement et le bouton "valider" redevient inactif. La liste et les deux boutons disparaissent uniquement quand on clique sur "annuler"

3.4.4°/ liste des équipes

- Un composant doit afficher la liste des équipes, ainsi qu'un mécanisme permettant de créer une nouvelle équipe.
- Ce mécanisme est laissé libre quand à sa forme et son fonctionnement.
- La seule contrainte est que la liste des équipes doit se mettre à jour automatiquement en cas de création.

3.4.5°/ consultation/modification d'une équipe et de ses membres.

- Un composant (potentiellement utilisant des sous-composants) doit afficher, si elle existe, l'équipe "courante", c.a.d. celle qui a été au préalable sélectionnée via la liste des équipes d'une organisation.
- S'il n'y a pas d'équipe courante, rien ne s'affiche.
- Sinon, le composant affiche le nom de l'équipe et un bouton permettant d'ajouter un membre à l'équipe.
- Le composant doit également afficher la liste des membres, avec pour chacun toutes ses informations, plus un bouton permettant de modifier le membre, ou bien de le supprimer de l'équipe.
- La forme et le fonctionnement de l'ajout, modification, suppression de membres est laissé libre.

Précédent (/index.php/menu-cours-s4/menu-mmi4web/2463-td-n-5-electron-2)

Suivant (/index.php/menu-cours-s4/menu-mmi4web/2456-tp-n-1-un-premier-projet-vue-cli-sans-plugins-4)

Haut de page