Paris EmberJS Lab #6

Novembre 2019

people doc

Antonin Messinger

Projet :

ember-cli-deploy-index-json: Générer un json destiné à être consommé par le backend pour construire l'index.html


@mr_meuble


@amessinger

people doc

Guillaume Gérard (aka GreatWizard)

Projets :

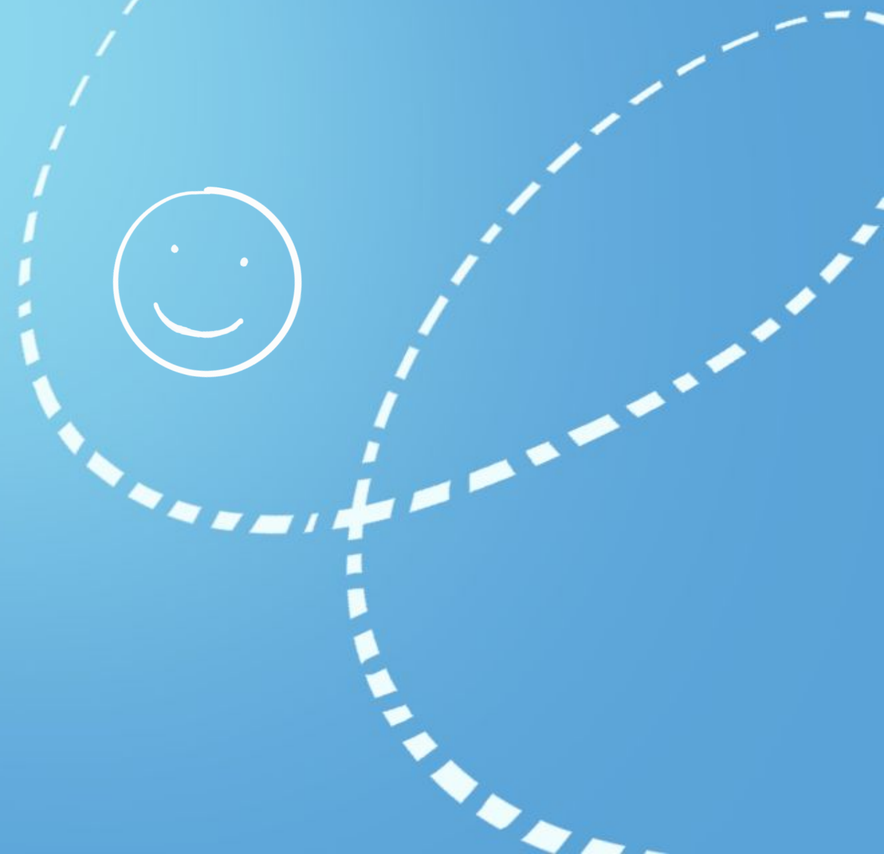
- GreatWizard/ember-circleci : Générer votre configuration CircleCI pour app ou addons
- GreatWizard/ember-service-worker-unregistration : Désenregistrer vos services workers
- peopledoc/ember-feature-controls : Activer / désactiver vos feature flags avec une interface d'administration et permettre de garder l'état dans le localStorage.

@ggerard88

@GreatWizard

Taking Over Server-side Rendering Websites

- Antonin Messinger
- Guillaume Gérard



people doc

PeopleDoc il y a 3 ans

- Application SaaS en Django (server-side rendering)
- Templating avec Bootstrap 2/3
- Zéro API
- jQuery pour les quelques interactions asynchrones
- Une équipe de trois devs JS
  - Objectif : Faire les projets avec un framework JS moderne *(spoiler: EmberJS)*
  - Le legacy est énorme

Cas #1 : SPA embarquée

Cas #1 : SPA embarquée

Prend en charge une *partie* du DOM.

Utile lorsque la page est préexistante et rendue par le serveur.

Sujets :

1. DOM
2. Routage
3. Feature Flags
4. API
5. Assets

Cas #1 : SPA embarquée - DOM

L’application prend en charge une *partie* du DOM.

“By default, your application will [...] attach [...] to the document's body element. [...] You can [...] append the application template to a different element by specifying its rootElement property”

— Ember guides, Changing the Root Element

Nous avons donc choisi de faire correspondre rootElement à modulePrefix.

Cas #1 : SPA embarquée - Routage

Nous voulons une application *stateful* en tirant avantage du routeur. Problème, notre application est rendue à une URL de base que nous (frontend) ne maîtrisons pas.

“[...] indicate to the router what the root URL for your Ember application is [...] by configuring the rootURL property [...]”

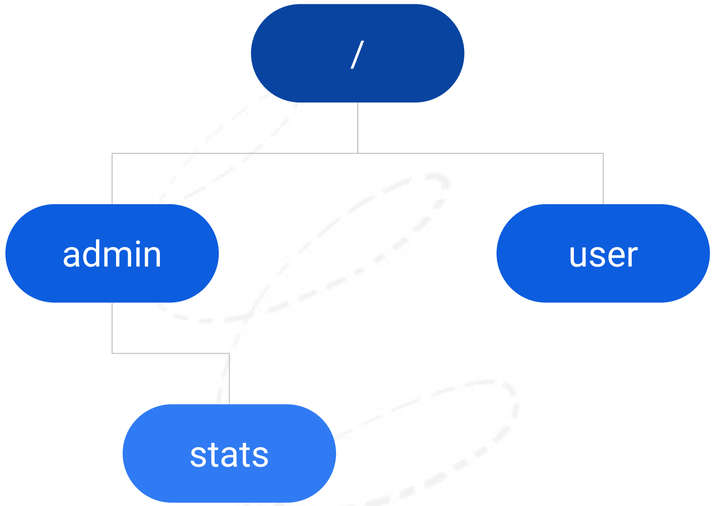
— Ember guides, Specifying a Root URL

Notre application va donc recevoir du backend cette URL de base pour que son routeur n’intervienne qu’à *partir* de celle-ci.

Cas #1 : SPA embarquée - Breadcrumbs

Cette problématique de routage ne se limite pas à l'URL :

Notre application se trouve dans une arborescence
qu'elle ne connaît pas à priori.



Etant servi à l'URL /admin/stats, l'application voudra
afficher sa position dans l'arborescence via des breadcrumbs.
"Home > Admin > route_actuelle_de_l_app"

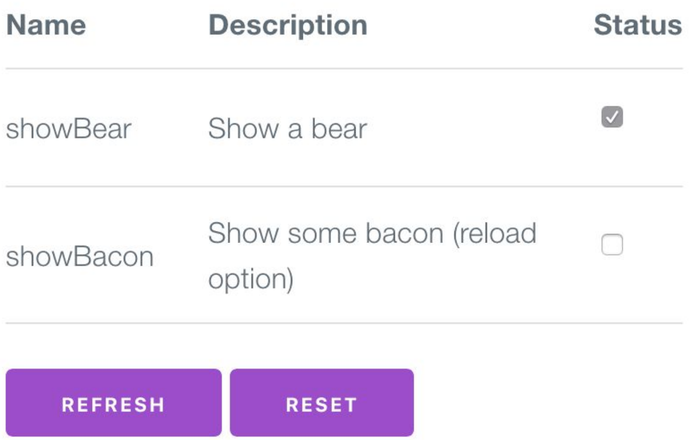
Le backend va donc la lui indiquer via un paramètre de la forme [{ label: Home', url: '/' }, { label: Admin, url: '/admin' }] (que nous passons par la suite à poteto/ember-crumbly).

Cas #1 : SPA embarquée - Feature Flags

Nous “feature flaggons” toutes nos fonctionnalités (avec kategengler/ember-feature-flags).

Le backend est responsable de leur activation, c’est pourquoi il passe une liste de la forme `{ flagName1: true, flagName2: false }` à notre application.

Side note : nous avons un addon pour activer nos feature flags directement depuis l’app peopledoc/ember-feature-controls



Cas #1 : SPA embarquée - API

Pour communiquer avec notre API, notre application a besoin de certaines d'informations :

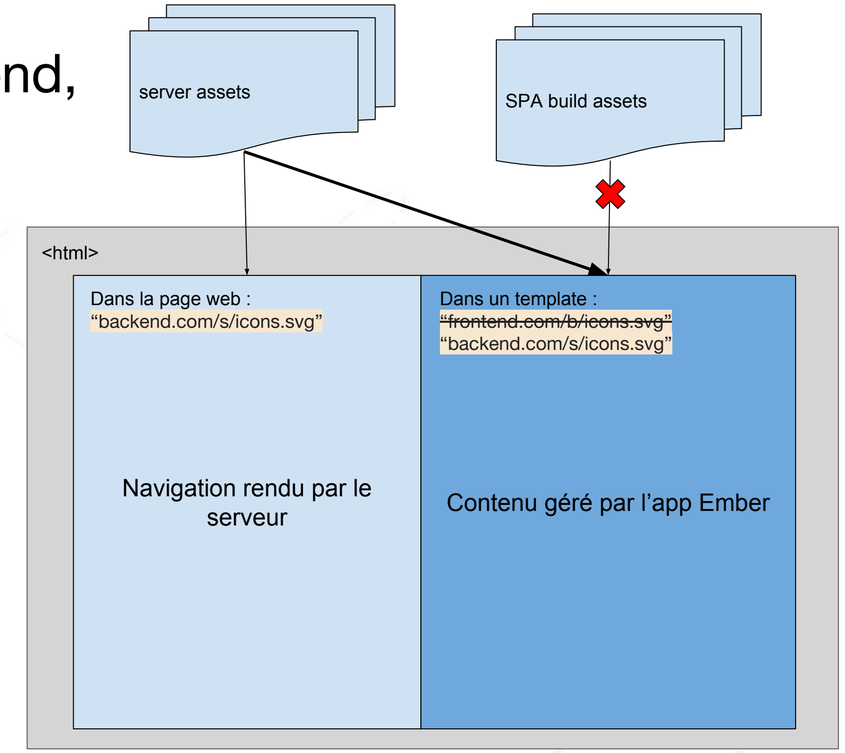
- **hostname** (standard)
- **namespace** (standard)
- **token** (notre utilisateur a déjà une session, demandons au backend un token à l'initialisation !)

Si la destination des 2 premiers paramètres suit la logique du framework, token est pour sa part non standard et nous le passons à un authenticator de simplabs/ember-simple-auth.

Cas #1 : SPA embarquée - Réutilisation des Assets

Notre application peut partager certaines assets avec le backend,

pour gagner un peu de cache entre autres.



Solution : un paramètre passé du backend au frontend.

Ex. iconsPath: “backend.com/assets/icons.svg”

Attention, le fait d’avoir un markup HTML venant de deux applicatifs distincts peut poser des problèmes de compatibilité si l’un et l’autre visent des versions différentes du framework UI.

Cas #1 : SPA embarquée - Configuration et démarrage de l'App

L'addon xcambar/ember-cli-embedded permet de :

- contrôler quand démarre l'application Ember
- lui passer un objet de configuration au démarrage

```
<body>
  <div id="my-app"></div>
  <script>
  MyApp.start({
    rootUrl: "/admin/stats",
    breadcrumbs: [ ... ]
  })
  </script>
</body>
```

Comment ?

"By default, the router will begin [working] once the browser emits the DOMContentReady event. If you need to defer [this], you can call the application's deferReadiness() method. Once [it] can begin, call the advanceReadiness() method."

— Ember API Documentation, Application

La configuration pour sa part est stockée dans un service.

Cas #1 : SPA embarquée - Déploiement et Fin

Déployer une application embarquée diffère d'une application standard : plus d'index.html pour indiquer au navigateur les ressources (versionnées !) qu'il doit charger.

Le backend doit être en mesure de générer son équivalent. Nous mettons donc à sa disposition un index.json à chaque déploiement, listant pour chaque ressource son équivalent versionné.

Il y a un addon pour ça peopledoc/ember-cli-deploy-index-json

Merci !

Cas #2 : SPA avec redirections

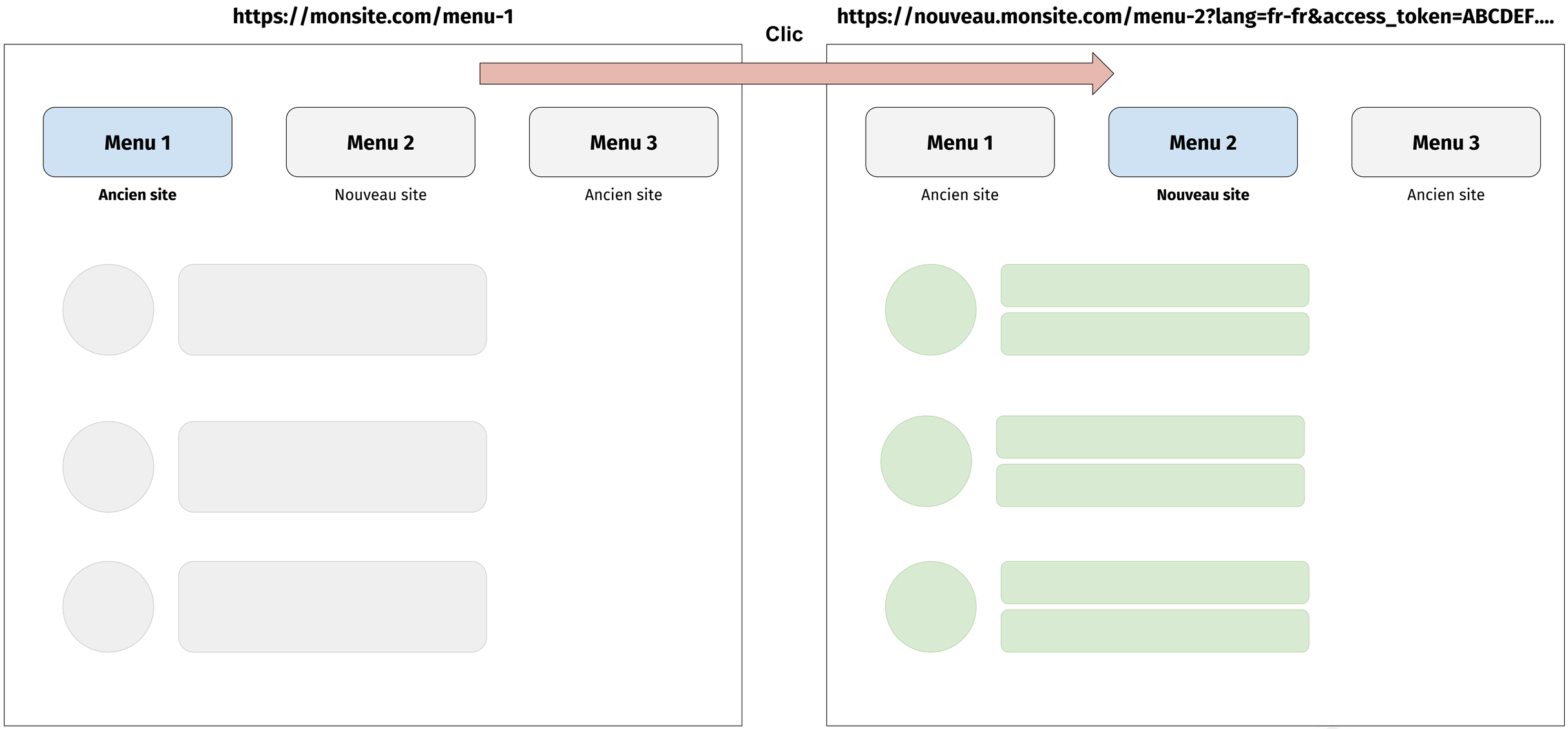
Cas #2 : Objectifs

- Remplacer l’existant par une SPA (au sens strict du terme)
- Utiliser des API
  - comprendre les fonctionnalités actuelles
  - entreprendre la création des API
  - authentification et gestion des droits (Oauth2)
- Avoir une transition visuellement cohérente entre les 2 applications

Cas #2 : Méthodologie

- SPA sur une nouvelle infrastructure
- Utilisation de l’ancien site pour le login/logout
  - évite de changer l’infra actuelle
  - évite de changer l’URL d’accès aux utilisateurs
- Migration progressive
  - Reprise de l’existant page par page
  - Utilisation de l’ancien site pour les pages non migré
  - Refonte UX avec prise en compte du mobile

Cas #2 : Redirection ancien site vers SPA



Cas #2 : Redirections

Il faut configurer l’ancien site pour rediriger vers la SPA (Menu 2) et faire évoluer les redirections au fil du temps.

La redirection vers la SPA se fait avec des query-params:

- **langue courante**
- **token d’accès OAuth2**

La SPA récupère au démarrage des informations de configuration par un call API sur le domaine courant

- Infos d’affichage de menu
- URLs de l’ancien site pour le Menu-1 et Menu-3
- Feature flags à activer/désactiver
- Hostname et namespace pour communiquer avec l’API

Cas #2 : Cohérence de design

Possibilité de naviguer d’un site à l’autre sans se rendre compte d’une différence visuelle.

Design cohérent entre l’ancien site et la SPA.

Nouvelle contrainte : mobilité (CSS responsive, PWA…)

Si vous avez des thèmes clients, il faut un service dédié pour délivrer les CSS/logos...

Cas #2 : Feature Flags

Nous “feature flaggons” toutes nos fonctionnalités (avec kategengler/ember-feature-flags).

Le backend est responsable de leur activation, c’est pourquoi il passe une liste de la forme `{ flagName1: true, flagName2: false }` à notre application.

Hors-sujet : nous avons un addon pour activer nos feature flags directement depuis l’app https://peopledoc.github.io/ember-feature-controls

Pareil que cas #1

Cas #2 : API

Pour communiquer avec notre API, notre application a besoin d'un certain d'informations :

- **hostname** (standard)
- **namespace** (standard)
- **token** (notre utilisateur a déjà une session, demandons au backend un token à l'initialisation !)

Si la destination des 2 premiers paramètres suit la logique du framework, token est pour sa part non standard et nous le passons à un authenticator de simplabs/ember-simple-auth.

Pareil que cas #1

Cas #2 : Fin