

Développement d'une application VueJS permettant de générer des fichiers de configuration pour OpenADOM

Stage du 04/04/2022 au 29/07/2022 à l'INRAE Val de Loire

Antoine DELAHAYE



Table des matières

1	Remerciements	3
2	Introduction	4
3	Structure d'accueil	5
3.1	L'INRAE	5
3.2	L'unité INFOSOL	5
4	Environnement de travail	7
5	Sujet	9
5.1	Introduction	9
5.2	Structure du fichier de configuration	10
5.3	Maquettage	13
5.4	Développement	15
5.5	Internationalisation	25
5.6	Tests	25
5.7	Documentation	27
6	Bilan	29
7	Conclusion	30

1 Remerciements

Je souhaite tout d'abord remercier mon maître de stage, Monsieur TCHERNIATINSKY Philippe, de m'avoir permis de rejoindre son équipe afin de travailler sur un projet très intéressant et complet en termes applicatif.

Je remercie Madame BOUKIR HAKIMA, ma collègue de bureau, de m'avoir supporté pendant quatre mois.

Je remercie aussi toute l'équipe INFOSOL et le personnel de l'INRAE avec lesquels j'ai pu travailler ou discuter.

Enfin je remercie Monsieur DABROWSKI Frédéric, mon tuteur de stage pour son suivi.

2 Introduction

Étudiant en troisième année de licence Informatique à l'Université d'Orléans, j'ai réalisé un stage de quatre mois, du 4 avril au 29 juillet, à l'INRAE Val de Loire à Ardon, au sein de l'unité INFOSOL.

Mon rôle dans cette équipe consistait à développer une application web permettant l'aide à la création du fichier de configuration pour les applications du SI ORE (Observatoire de Recherche en Environnement).

3 Structure d'accueil

3.1 L'INRAE

L'INRAE, l'Institut National de la Recherche Agronomique Environnementale est né le 1er janvier 2020. Il est issu de la fusion entre l'INRA, Institut National de la Recherche Agronomique et IRSTEA, Institut national de Recherche en Sciences et Technologies pour l'Environnement et l'Agriculture.

L'INRA a été créé en réponse aux pénuries alimentaires provoquées par la fin de la Seconde Guerre mondiale : *combiner science et technologie pour améliorer les techniques agricoles et d'élevage*. Une fois cette question résolue, les objectifs de l'INRA ont changé et plusieurs missions ont été confiées à l'organisme. Ces missions couvrent un large éventail de domaines tels que les sols, le changement climatique, le carbone renouvelable, les systèmes agricoles à haute performance environnementale, l'eau, la biodiversité, l'alimentation humaine, les biotechnologies végétales et les maladies émergentes.

Aujourd'hui l'INRAE regroupe dix-huit centres régionaux (en plus de son siège à Paris) répartis sur plus de 150 sites et emploie environ 8200 titulaires ainsi que de nombreux doctorants (environ 500) et nom permanents (environ 2600).

Pour ce qui est de l'INRAE Val de Loire, elle compte 800 agents dont 632 titulaires et regroupe quatre pôles :

- Dynamique des sols et gestion de l'environnement
- Biologie intégrative des arbres et organismes associés
- Infectiologie et one health
- Biologie animale intégrative et durabilité des systèmes d'élevage

3.2 L'unité INFOSOL

J'ai donc réalisé mon stage au sein de l'équipe Éco-Informatique ORE, cette dernière fait partie de l'unité INFOSOL.

INFOSOL est une unité de services. Ses activités s'exercent dans le cadre de la participation de l'INRAE à un Groupement d'Intérêt Scientifique Sol (GIS Sol) qui propose un

ensemble de programmes nationaux pour faciliter et encourager une gestion patrimoniale et durable des sols.

L'unité INFOSOL, qui gère un SI Sol depuis l'acquisition jusqu'à la valorisation de données sols, en passant par leur traitement, s'est vue confiée la mission de réaliser une application destinée à gérer les données environnementales issues des ORE.

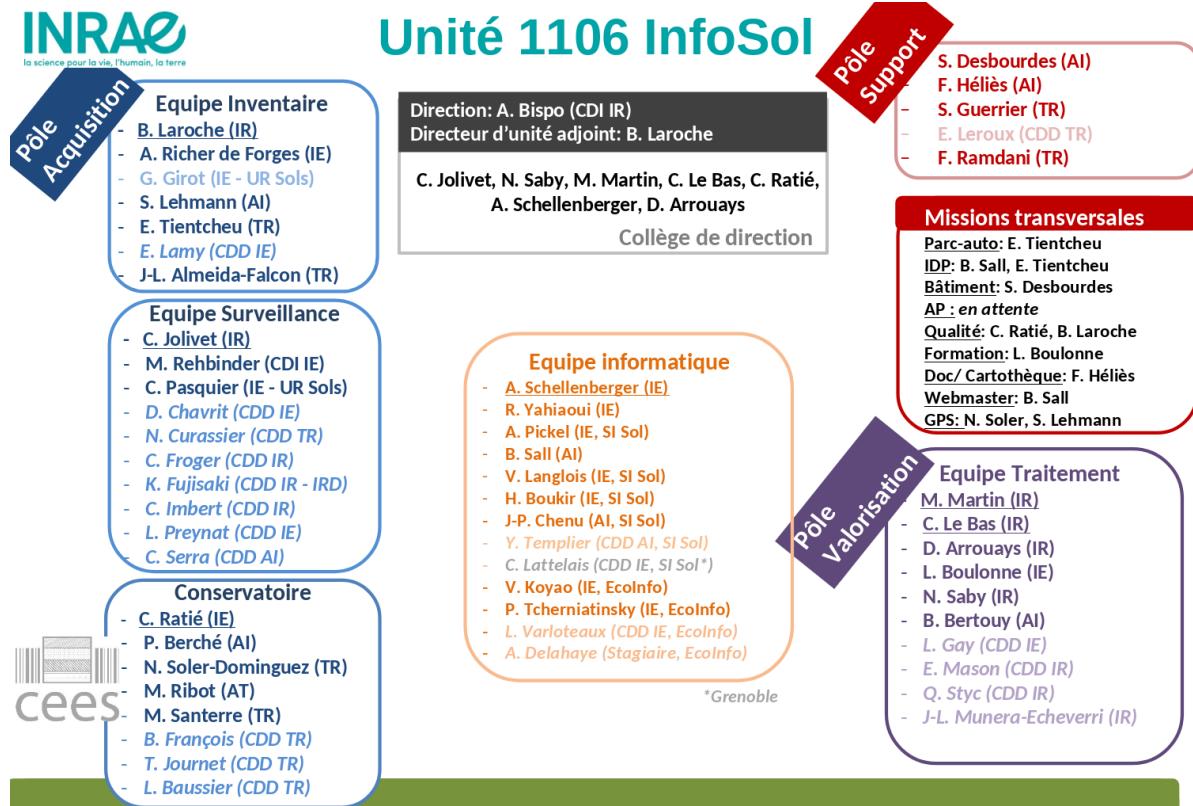


Figure 3.1: Organigramme de l'unité

4 Environnement de travail

Afin de pouvoir travailler dans de bonnes conditions, j'ai partagé pendant ma période de stage un bureau de travail avec Madame BOUKIR Hakima, développeuse pour l'équipe SI Sol.



Figure 4.1: Mon bureau de travail

Concernant mon outil de travail, il s'agit d'un ordinateur portable Dell de 15,6 pouces sous Linux afin d'installer tous les paquets nécessaires pour faire du développement d'application web. J'ai aussi à disposition une station d'accueil à mon bureau afin de pouvoir brancher un écran externe de 27 pouces, un clavier et une souris, tout en alimentant mon ordinateur avec un seul cable.



Figure 4.2: Mon ordinateur

5 Sujet

5.1 Introduction

Comme expliqué précédent, la mission générale de l'unité INFOSOL est de constituer et de gérer un système d'information à vocation nationale et européenne sur les sols, par rapport à leur distribution spatiale, leurs propriétés et l'évolution de leur qualité. La mission de l'unité a été élargie aux données environnementales et aux données des pratiques culturelles. De ce fait, afin de répondre à ces exigences, l'équipe des ORE a été chargée de développer outil permettant de gérer ces données. Ainsi, à la suite d'un audit sur la précédente application des SI ORE, le développement d'une nouvelle application a été initié. C'est de là qu'est né OpenADOM (Application for Data Organization and Management).

OpenADOM est donc avant tout une API REST (Interface de Programmation d'Ap-plication REpresentational State Transfer), c'est-à-dire un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web permettant de générer une application pour stocker, gérer et visualiser des données à partir d'un fichier de configuration. Ces données sont stockées dans des fichiers CSV. Ce fichier de configuration permettant de décrire les données à stocker, les référentiels, leurs types, les relations entre les référentiels et entre les données et les référentiels. Le format utilisé par ce fichier est le YAML, il s'agit d'un format de représentation de données simple et facile à comprendre.

Cependant, l'écriture d'un tel fichier est très fastidieuse. En effet, ce dernier est relativement complexe étant donné qu'il est chargé de décrire la structure d'une application, il est possible de se retrouver avec des fichiers faisant plusieurs milliers de lignes.

```
version: 1
application:
  defaultLanguage: fr
  internationalizationName:
    fr: SOERE mon SOERE avec dépôt
    en: SOERE my SOERE with repository
  name: MONSORE
  version: 1
compositeReferences:
  sites:
```

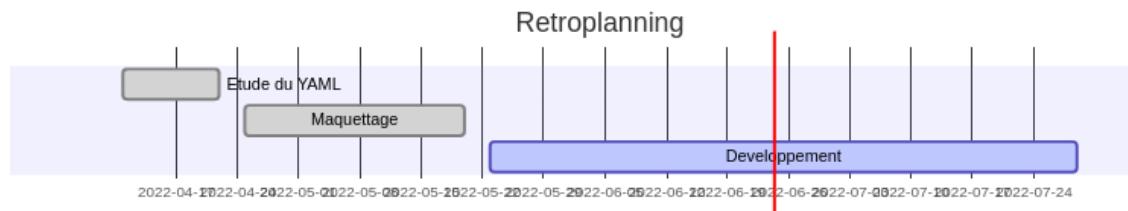
```

components:
  - reference: type_de_sites
    parentRecursiveKey: null
  - parentKeyColumn: tze_type_nom
    parentRecursiveKey: zet_chemin_parent
    reference: sites
projet:
  components:
    - reference: projet

```

C'est donc à partir du constat que la création d'un tel fichier se révèle trop complexe, qu'il m'a été demandé de développer une application permettant de guider pas à pas l'utilisateur dans la création de ce fichier. Ainsi, lors de l'une des réunions hebdomadaires du SI ORE, il a été décidé avec mon maître de stage et notre chef de projet Monsieur SCHELLENBERGER Antoine d'établir un rétroplanning afin de gérer au mieux les tâches à réaliser dans le temps imparti de mon stage. De ce fait, il a été convenu de me laisser :

- jusqu'à la mi-avril afin d'étudier et de comprendre le fonctionnement d'un fichier de configuration pour être capable d'en construire un par moi-même
- jusqu'à la mi-mai pour concevoir des maquettes
- jusqu'à la fin de mon stage pour développer et écrire la documentation de l'application.



5.2 Structure du fichier de configuration

Afin de comprendre correctement le sujet du stage, il est nécessaire d'apporter du vocabulaire et de détailler la structure d'un fichier. En effet, le fichier contient cinq parties permettant de décrire l'application :

- version

La version de l'analyseur du fichier de configuration (0, 1, 2, ...). Cette partie est propre au format YAML. En effet, tout fichier YAML comporte cette "clé" qui permet de savoir

quelle version du format est utilisée étant donné que le format YAML peut évoluer au fil du temps. Dans le cadre de OpenADOM, la version 1 est la version courante.

```
version: 1
...

```

- application

Cette partie sert à la description de l'application avec notamment, le nom de l'application. Une section optionnelle d'internationalisation du nom de l'application, la langue par défaut et la version du fichier de configuration. Lorsque l'on apporte des modifications au fichier, on incrémente la version du fichier.

```
application:
  defaultLanguage: fr
  internationalizationName:
    fr: Ma première application
    en: My first application
  name: application_nom
  version: 1

```

- references

Il s'agit d'un ensemble d'informations permettant de préciser le contexte de la mesure ou de l'observation, fournies sous la forme de fichier CSV. Chaque fichier représentant un référentiel.

On décrit un référentiel de données en y listant le nom des colonnes souhaitées dans la partie `columns` en fonction de celles présente dans le fichier CSV, les colonnes qui forment la clé naturelle sont renseignées dans `keyColumns` et on pourra aussi décrire des règles de validations sur une ou plusieurs colonnes dans une section `validations` qui sera détaillée plus tard. De plus, des sections d'internationalisation optionnelles peuvent être détaillées.

```
references:
  especes:
    internationalizationName:
      fr: Espèces
      en: Species
    internationalizedColumns:
      esp_definition_fr:
        fr: esp_definition_fr
        en: esp_definition_en
    internationalizationDisplay:
      pattern:
```

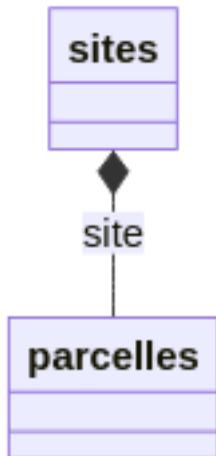
```

    fr: '{esp_nom}'
    en: '{esp_nom}'
keyColumns:
  - esp_nom
columns:
  esp_nom: null
  esp_definition_fr: null
  esp_definition_en: null
  colonne_homonyme_entre_referentiels: null

```

- compositeReferences

Une référence composite permet de définir une hiérarchie entre différentes données de référence.



```

compositeReferences:
localizations:
components:
  - reference: sites
  - reference: parcelles
  parentKeyColumn: 'site'

```

- dataTypes

Un type de donnée se compose :

- d'une section **data** permettant de décrire le schéma des données enregistrées en base
- d'une section **format** pour déclarer le mapping entre le fichier de données (CSV) et le schéma déclaré dans **data**

- des autorisations
- les validations de chaque ligne

```

dataTypes:
  flux_meteo_dataResult:
    authorization:
      ...
    timeScope:
      component: day
      variable: Date
    data:
      ...
  format:
    headerLine: 2
    firstRowLine: 4
    columns:
      ...

```

5.3 Maquettage

J'ai utilisé Figma pour concevoir des maquettes de l'application rapidement.

Dans un premier temps, je suis parti sur des maquettes sans règles de design spécifiques mis à part de respecter la charte graphique de l'INRAE. La disposition était épurée avec un entête, au milieu le contenu de page sur un fond blanc et entre l'entête et le contenu, une barre de navigation.

Cependant, je me suis rapidement rendu compte de ses principaux défauts qui sont de ne pas suivre de règles d'ergonomie et d'avoir une interface beaucoup trop simpliste qui ne permettent pas d'évoluer rapidement. J'ai donc décidé après avoir produit trois maquettes de partir sur un nouveau design de maquettes respectant des règles de conception.

Après de longues recherches, j'ai décidé d'utiliser Material Design afin d'unifier le style de l'application. Material Design est un ensemble de règles de design fourni par Google afin de concevoir des interfaces de qualité plus rapidement. C'est donc grâce à des composants de Material Design que j'ai pu reprendre le maquettage de mon application et repartir sur de bonnes bases.

J'ai pu, suite aux retours qui ont été fait lors de la première présentation des maquettes, retravailler ces panneaux, notamment par rapport au nommage des pages et à la navigation. Même si la structure reste la même avec un entête aux couleurs de l'INRAE et une carte centrale avec le contenu de la page, j'y ai ajouté une barre de navigation latérale et supprimée celle entre le contenu de la page et l'en-tête afin de bien délimiter le contenu de la navigation.

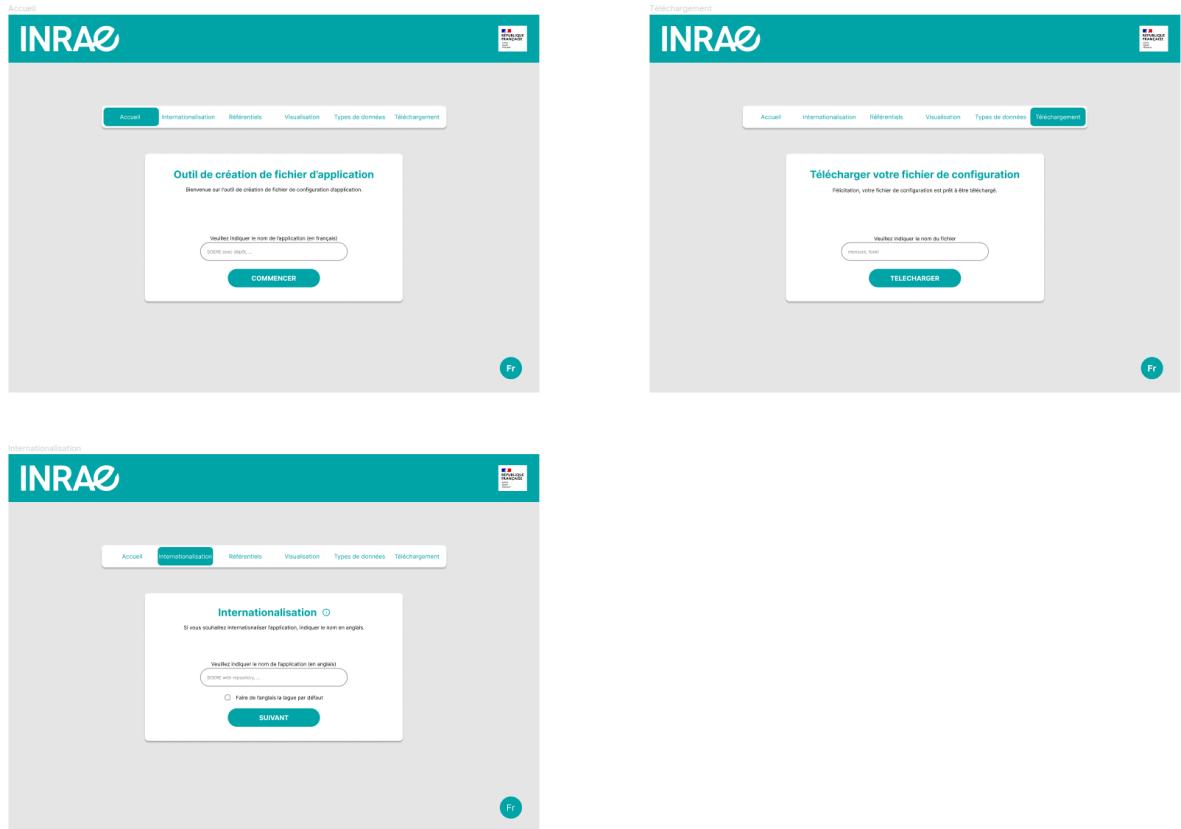


Figure 5.1: Première version des maquettes

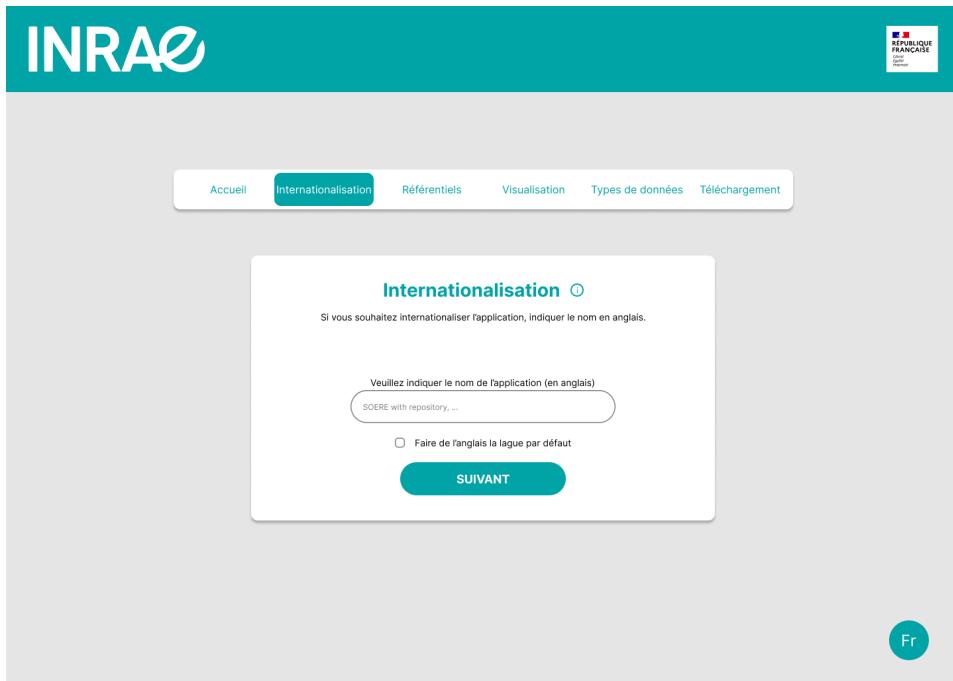


Figure 5.2: Page d'internationalisation

Sur la page d'accueil, on peut choisir si l'on veut créer un nouveau fichier ou bien importer un fichier existant, la page application qui contient deux champs de textes afin d'internationaliser le nom de l'application et une boîte à cocher si on veut que l'anglais soit la langue par défaut.

Pour ce qui est de la page des référentiels, nous avons un tableau affichant le nom des référentiels, le nombre de colonnes, les colonnes clés et des boutons d'actions.

Après avoir terminé le design de la page des référentiels, je me suis attelé à concevoir une boîte de dialogue pour ajouter des référentiels. En effet, je souhaitais avoir une boîte de dialogue pour ajouter des référentiels et non une nouvelle page pour ça. Cependant, ayant déjà une idée de la disposition général qu'allait prendre cette page et n'ayant pas de visibilité sur la partie technique de l'application et donc de ce qui allait être possible de faire, j'ai décidé de commencer le développement d'un premier prototype afin de présenter une première version fonctionnelle des maquettes.

5.4 Développement

Avant de pouvoir commencer à développer un premier prototype, une phase de recherche concernant le framework UI à utiliser a été nécessaire, ce framework doit bien évidemment respecter les règles de Material Design.

The figure displays three wireframe mockups of application screens for INRAE:

- Application Configuration Screen:** Shows a sidebar menu with "Application", "Référentiels", "Types de données", "Visualisation", and "Téléchargement". The main area is titled "Application" with a sub-section "SOERIE avec dépôt... Optimisé". It includes a checkbox for "Faire de l'anglais la langue par défaut" and a "SUIVANT" button.
- Referential Management Screen:** Shows a sidebar menu with "Référentiels" selected. The main area is titled "Référentiels" and contains a table with columns: Nom du référentiel, Nombre de colonnes, Colonne clé, and Actions. The table lists six entries: test1 (2 columns, test1_nom), test2 (3 columns, test2_nom), test3 (1 column, test3_nom), test4 (2 columns, test4_nom), test5 (5 columns, test5_nom), and test6 (4 columns, test6_nom). Below the table are buttons for "+ IMPORTER" and "+ RÉFÉRENTIEL".
- CSV Import Screen:** Shows a sidebar menu with "Référentiels" selected. The main area is titled "Que souhaitez-vous faire ?" and contains a message about creating or modifying a configuration file. It includes buttons for "+ NOUVEAU" and "ENVOYER".

Figure 5.3: Deuxième version des maquettes

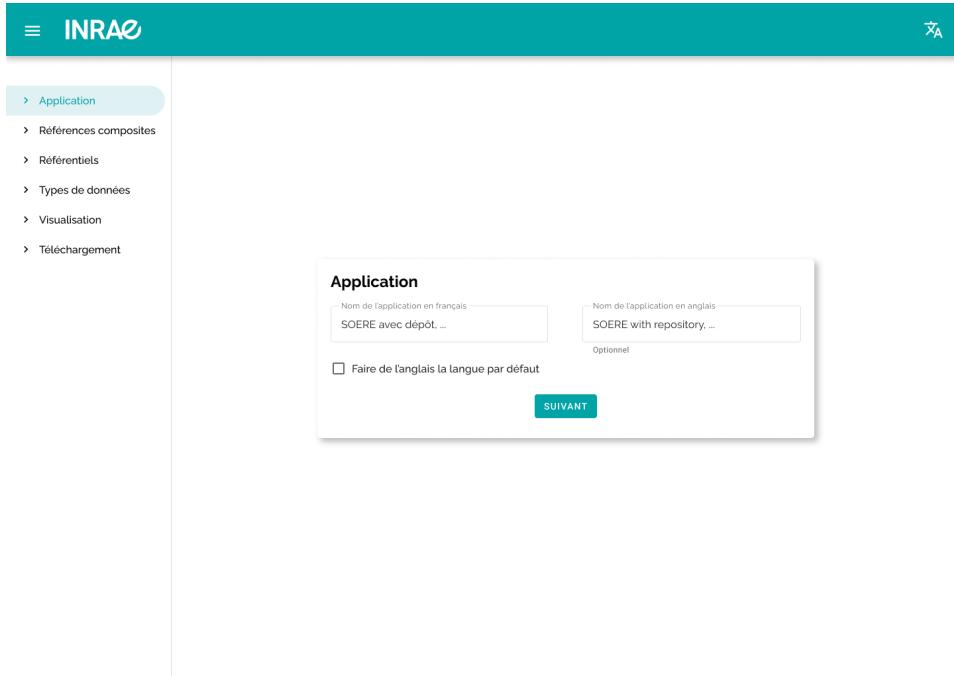


Figure 5.4: Page de la section application

Un framework UI est un ensemble de composants qui permettent de développer des interfaces utilisateur. En effet, utiliser un framework UI respectant Material Design va me permettre d'avoir à disposition des composants déjà construits afin de faciliter le développement, c'est ce que propose Vuetify, le framework UI le plus populaire sous VueJS, ce dernier est libre, possède une communauté très active, est mis à jour chaque semaine et possède des versions avec un support long terme.

Un des autres aspect technique important est le fait d'utiliser VueJS étant donné que OpenADOM utilise déjà VueJS et qu'il est nécessaire que cette application soit accessible partout sans avoir à installer un logiciel. Quant à VueJS, il s'agit d'un framework JavaScript qui permet de construire des interfaces utilisateur et des applications web mono-page. Pour ce qui est de la version utilisée, il s'agit de la version 3 de VueJS, cette dernière apporte des améliorations notables sur les performances, la taille de l'application et la facilité de développement.

De plus, au lieu de d'utiliser Vue CLI qui permet simplement de construire le projet j'ai utilisé Vite, un outil de développement de dernière génération permettant d'améliorer la vitesse construction du projet, par exemple lorsqu'on modifie un fichier qui influe sur la disposition de l'interface, ces modifications sont directement visibles et de plus, Vite intègre des mécanismes d'optimisation afin de précompiler certaines ressources et importer uniquement celle nécessaire.

Concernant le versionnage, l'application a été versionnée avec le GitLab de l'INRAE. Le dépôt contient une branche `main` où il n'est possible de pousser des modifications qu'en

effectuant une demande de fusion avec une autre branche (merge request) devant être acceptée par mon maître de stage ainsi que mes branches permettant de développer les différentes fonctionnalités.

Concernant les outils de développement, j'ai utilisé :

- WebStorm comme IDE étant donné qu'il s'agit d'un projet JavaScript
- NodeJS en version 16 LTS (Long Term Support) comme plateforme de développement
- NPM comme gestionnaire de paquets

J'ai donc dans un premier temps installé Vite ainsi que VueJS afin d'avoir la base de mon application, cela se fait très simplement en suivant la documentation de Vite. La partie un peu plus ardue de l'installation viens de Vuetify. En effet, un framework UI nécessite un peu de configuration pour fonctionner correctement, notamment avec Vite afin que ce dernier puisse optimiser Vuetify pour le rechargement des pages.

Par exemple, ce bout de code va permettre de construire un thème personnalisé pour l'INRAE avec la définition de la charte graphique.

```
export const inrae: ThemeDefinition = {
    dark: false, // On définit si le thème est sombre ou non
    colors: {
        background: '#ffffff', // Couleur de fond
        surface: '#ffffff', // Couleur de surface
        primary: '#00a3a6', // Couleur primaire
        error: '#df463a' // Couleur d'erreur
    }
}
```

Ainsi, après avoir défini le thème, ce dernier est importé dans un fichier nommé `main.js`, c'est lui qui va permettre de charger et paramétriser les différents frameworks utilisés.

Une fois cela fait, j'ai peu m'attaquer à la structure de mon application. Lors d'une précédente réunion, il a été défini que l'application avait besoin de sept pages :

- La page application pour le nom et la langue
- Une page pour les référentiels
- Une pour les types de données
- Les références composites dans une autre
- Une page permettant de visualiser l'évolution du fichier
- Et une permettant de télécharger le fichier

De ce fait, sept composants ont été créés, un pour chacune pages. Un composant VueJS correspond à un composant HTML contenant le code HTML, JavaScript et CSS pour une page en question et il va être possible par la suite d'appeler une ou plusieurs instances

du composant indépendant entre eux afin d'afficher ces derniers. Avec cela, on va donc installer Vue Router afin de faire correspondre une route à un composant et donc de permettre de naviguer entre les différentes pages. Par exemple, si on veut aller sur la page des référentiels, on va avoir https://mon_application.fr/references ou references correspond à la route.

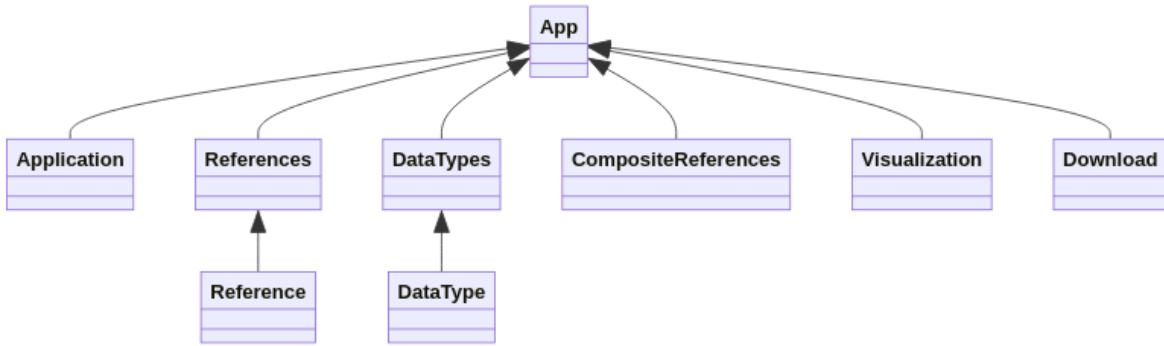
Ce code permet de lier une route à un composant.

```
const router = createRouter({
  history: createWebHistory(),
  routes: [
    {path: '/', component: Home},
    {path: '/application', component: Application},
    {path: '/references', component: References},
    {path: '/data-types', component: DataTypes},
    {path: '/composite-references', component: CompositeReferences},
    {path: '/visualization', component: Visualization},
    {path: '/download', component: Download}
  ]
})
```

Après avoir configuré le projet pour avoir une application fonctionnelle, j'ai pu développer l'interface grâce aux composants cités précédemment. En effet, chaque composant possède trois parties, une partie pour le code JavaScript, une partie pour le code HTML et une pour le code CSS. Cela permet de tout regrouper dans un seul et même fichier et ainsi chaque composant aura son code qui lui ait propre.

En plus des composants décrits précédemment, un composant App est présent dans le projet, Ce dernier intègre les différents composants grâce à Vue Router de plus, la réutilisabilité étant un des concepts clé de VueJS, cela va permettre de réutiliser des éléments présents sur chaque page, comme la barre navigation ou l'entête sans avoir à réécrire le même code à chaque fois.

Ce diagramme représente la structure de l'application. Reference et DataType seront détaillés plus tard.



Comme expliqué précédent, je me suis d'abord attardé sur l'aspect ergonomique plutôt que la partie fonctionnelle de l'application. C'est-à-dire que j'ai fait en sorte que l'on puisse naviguer dans l'application afin de pouvoir présenter un aperçu de l'application et d'avoir un retour sur cette dernière.

En effet, l'aspect ergonomique est plus important dans notre cas étant donné que le but premier de l'application est de rendre la création du fichier le plus transparent possible.

Une fois que le visuel validé, j'ai pu m'atteler à la partie fonctionnelle de l'application. Ici cela fait référence au fait de stocker les données du fichier, pour ce faire le format JSON a été utilisé. Le JSON est un format standard utilisé pour représenter des données structurées de façon semblable aux objets Javascript. Étant donné qu'il existe des librairies pour convertir du YAML en JSON et inversement, cela permet d'avoir un objet que l'on peut manipuler pour ajouter nos données et qu'on pourra donc par la suite convertir en YAML, de même pour importer un fichier existant que l'on pourra convertir en JSON.

Afin de stocker cet objet et qu'il soit accessible dans chaque composant, j'ai utilisé Pinia. Il s'agit d'un gestionnaire d'état et une bibliothèque pour des applications VueJS. Il sert de zone de stockage de données centralisée pour tous les composants dans une application. C'est donc grâce à Pinia que qu'ont défini un store, ce store va représenter notre objet JavaScript. Dans ce store, on va aussi pouvoir ajouter des getters afin de récupérer des données depuis le store et des actions afin d'agir avec l'objet.

```

export const useYamlStore = defineStore({
  id: 'yaml',
  state: () => ({
    application: {
      defaultLanguage: 'fr',
      internationalizationName: {
        fr: null,
        en: null
      },
      name: null,
    }
  })
})

```

```

        version: 1
    },
    compositeReferences: {},
    references: {},
    dataTypes: {}
}),
getters: {
    getYaml() {
        return {...}
    }
},
actions: {
    setYaml(yaml) {
    },

    resetYaml() {
    },

    addReference(index, reference) {
    }
}
)
}

```

Ainsi, nous avons la page d'accueil qui permet de créer un nouveau fichier ou bien de charger un fichier existant.

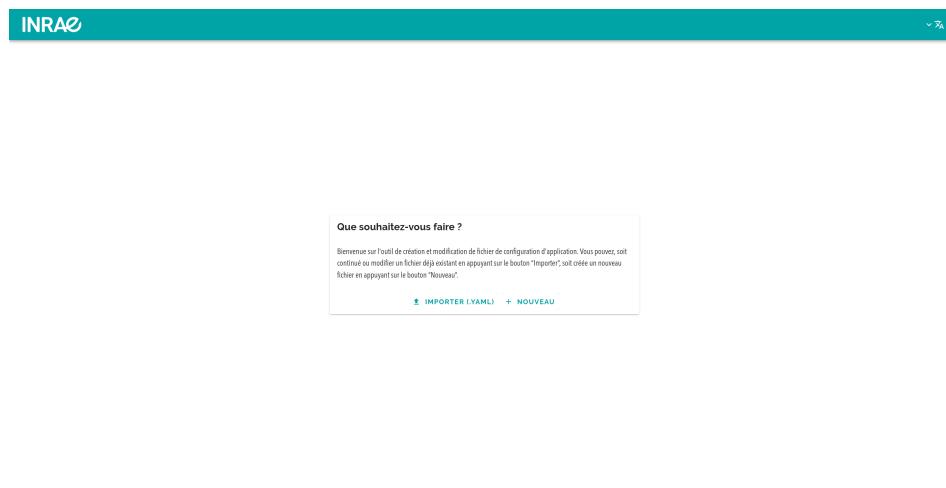


Figure 5.5: Page d'accueil

La page d'application quant à elle permet de donner un nom à l'application et de définir la langue par défaut. Chaque champ de texte est directement relié à l'objet JSON, ce qui

permet de modifier en temps réel les données sans avoir besoin de bouton de validation.

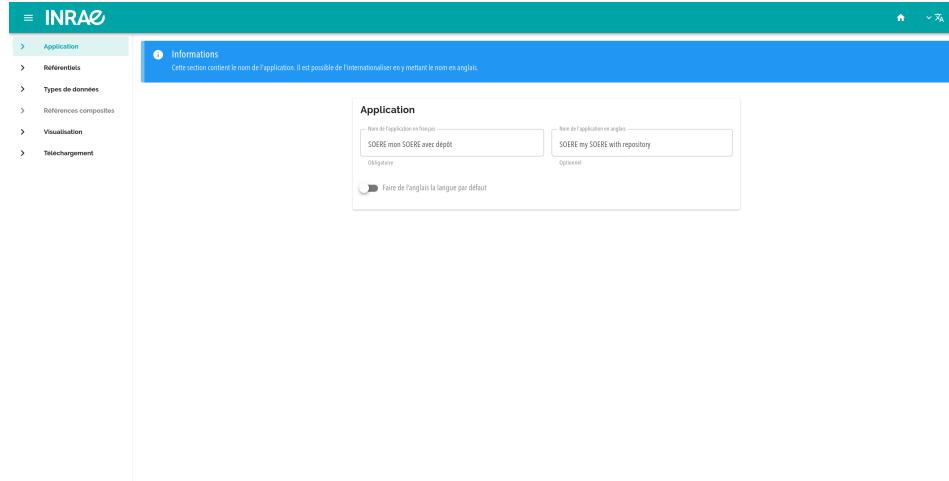


Figure 5.6: Page de l'application

Une des pages les plus importantes, cette dernière va permettre de voir, éditer et supprimer des référentiels, un référentiel possède des colonnes, des contraintes et des colonnes clés, on affiche ces informations pour chaque référence dans un tableau. Il y a aussi possibilité d'éditer ou de supprimer un référentiel. Un bouton permettant d'ajouter un nouveau référentiel est aussi présent en dessous du tableau.

The screenshot shows the 'Référentiels' management page. The sidebar menu is identical to Figure 5.6. The main content area has a blue header 'Informations' with the sub-section 'Référentiels'. A note says 'Les références sont un ensemble d'informations permettant de préciser le contenu de la mesure ou de l'observation. Pour décrire un référentiel, on y liste les noms des colonnes visuelles et précise la liste des colonnes qui forment la clé naturelle.' Below is a table titled 'Référentiels' with columns: 'Nom du référentiel', 'Nombre de colonnes', 'Nombre de contraintes', 'Colonnes clé', and 'Actions'. The table lists various entities like 'Espace', 'Projet', 'Site', etc., each with specific details. At the bottom are buttons 'IMPORTER (.YAML)' and '+ RÉFÉRENTIEL'.

Figure 5.7: Page des référentiels

Afin ajouter ou éditer un référentiel, on utilise une boite de dialogue comprenant tous les champs nécessaires pour cette dernière. Cette boite de dialogue étant la même pour ajout et édition, un composant nommé Reference a été créé comme on peut le voir sur le diagramme reprenant la structure de l'application. On va donc pouvoir remplir les champs nécessaires comme le nom du référentiel, les colonnes ou même les contraintes afin ajouter un référentiel.

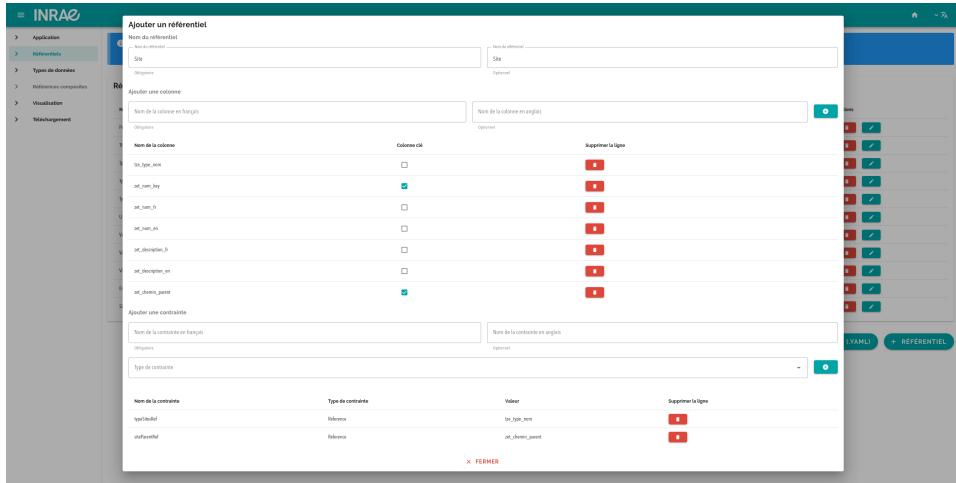


Figure 5.8: Boite de dialogue pour l'ajout ou l'édition d'un référentiel

Comme décrit précédemment, chaque champ texte qui modifie directement une donnée du fichier de configuration est relié à cette dernière dans l'objet enregistré dans le store, cela permet de modifier en temps réel les données dans l'objet. Ici par exemple, nous avons l'attribut `v-model` d'un composant de champ de texte qui permet de lier sa valeur d'entrée à la donnée de l'objet.

```
<v-text-field id="referenceName" :rules="[rules.required]"
    :label="t('reference.name', ['en français', 'in French'])"
    :placeholder="t('reference.frPlaceholder')"
    variant="outlined" color="primary"
    :hint="t('hint.required')" persistent-hint
    v-model="reference.internationalizationName.fr"
/>
```

Un aspect important de l'application est la présence de validateur permettant de vérifier les contraintes. Il s'agit d'une fonctionnalité mise à disposition par Vuetify. Il est possible d'inclure des règles de validation dans chaque composant afin d'afficher une erreur lorsque les règles ne sont pas respectées. On peut le voir ci-dessus avec l'attribut `rules` qui fait appelle à une règle qui a été déclarée dans la section script du composant Vue.

La règle en question. Si la valeur du champ est vide alors on affiche une erreur.

```
rules = {
    required: v => !!v || this.t('rule.required')
}
```

Une des fonctionnalité qui avait été évoqué en réunion est le fait de pouvoir visualiser le fichier en cours de construction afin d'avoir une vue d'ensemble sur le fichier et notamment

Figure 5.9: Les validateurs

de pouvoir revenir plus facilement sur des points spécifiques en cliquant sur un élément. Pour l'instant, il est juste possible de visualiser l'objet, mais pas d'interagir, il s'agit d'une des fonctionnalités qui sera ajoutée par la suite.

Figure 5.10: La page de visualisation

La page de téléchargement va permettre de télécharger le fichier créé grâce à l'application. Le principe de fonctionnement est simple, lorsque l'on clique sur le bouton de téléchargement, on va récupérer l'objet contenu dans le store et on va utiliser une librairie qui se charge de convertir du JSON en YAML. Une fois le travail effectué, on génère un fichier avec l'extension YAML contenant les données.

Concernant la page des types de données, à l'heure actuelle cette dernière est fonctionnelle, mais n'est pas pourvu de toutes les fonctionnalités, de même pour la page des références composites qui n'est pas encore accessible. Ces fonctionnalités seront développées durant la suite de mon stage en juillet.

5.5 Internationalisation

Un aspect important de l'application est la gestion de la langue. Il est possible de changer la langue de l'application depuis le bouton prévu à cet effet en haut de la page. Deux langues sont disponibles, français comme langue par défaut et l'anglais. Si on change la langue, un cookie est créé afin de garder la langue en mémoire. Pour cela, on utilise Vue i18n et Vue3 Cookies. Vue i18n est une API permettant de gérer l'internationalisation de l'application. Cette dernière va fonctionner avec Vite afin de précompiler les fichiers de traduction et de ce fait, cela permet de changer la langue à la volée sans recharger la page.

Les données de traduction sont stockés dans des fichiers JSON qui sont importés par Vue i18n et mis à disposition dans les composants de Vue. Étant donné qu'il s'agit du format JSON, les traductions fonctionnent par système de clé/valeur. Par exemple, on veut un bouton fermer. Dans ce cas, on a créé une clé `close` et on lui associe la traduction en fonction de s'il s'agit du JSON en français ou en anglais.

```
{  
  "close": "Fermer"  
}
```

```
{  
  "close": "Close"  
}
```

Après avoir défini la traduction, on peut utiliser cette clé dans un composant grâce à l'attribut `t`.

```
<v-btn @click="closeDialog">{{ t('close') }}</v-btn>
```

5.6 Tests

Un autre aspect important du projet concerne les tests à réaliser. En effet, il est nécessaire de vérifier par exemple que le fichier YAML qui sera généré est correct. Pour ce faire, plusieurs fichiers test ont été générés. Ensuite, grâce à OpenADOM et son API, il est possible de tester si un fichier est valide.

D'autres tests sont bien évidemment nécessaires pour vérifier que l'application fonctionne correctement. Pour ce faire Cypress a été intégré au projet. Il s'agit d'un outil qui permet de tester l'application en utilisant un navigateur embarqué afin de tester la navigation et les fonctionnalités de l'application. Pour ce faire, on va écrire un test qui va décrire les

actions à effectuer dans le navigateur, puis on va lancer le test et si ce dernier passe cela veux dire qu'il a été possible d'effectuer toutes les actions demandées.

Ce test permet par exemple de simplement vérifier qu'il est possible d'importer un fichier existant et de le charger.

```
describe('Testing', () => {
  it('Navigate to the home page', () => {
    cy.visit('http://localhost:3000/')
  })

  it('Import a existing file', () => {
    cy.get('#home').click()
    cy.get('#import').invoke('show').selectFile('cypress/fixtures/foret.yaml')
    cy.get('.v-navigation-drawer__scrim').click('center')
    cy.get('#applicationName').contains('foret')
  })
})
```

Pour aller plus loin, des tests continus ont été mis en place grâce à GitLab. Cela va permettre de tester automatiquement l'application dès qu'un changement est fait sur le dépôt.

Grâce à un fichier YAML nommé `.gitlab-ci.yml`, on va décrire comment exécuter les tests.

```
stages:
  - test

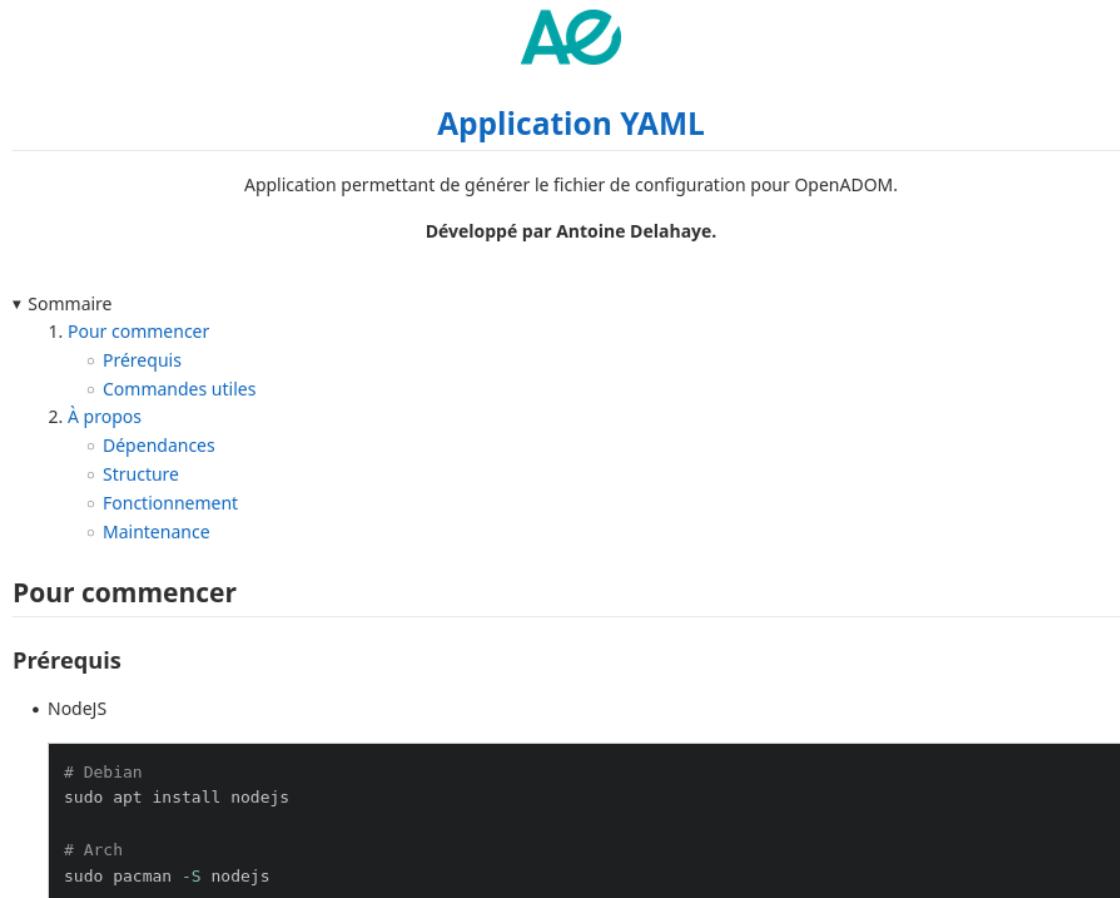
cache:
  key: ${CI_COMMIT_REF_SLUG}
  paths:
    - node_modules/
    - .npm/

test:
  image: cypress/browsers:node17.8.0-chrome99-ff97-slim
  stage: test
  script:
    # install dependencies
    - npm ci
    # start the server in the background
    - npm run dev &
    # run Cypress tests
```

```
- npx cypress run --browser firefox
artifacts:
  when: always
  paths:
    - cypress/videos/**/*.{mp4}
    - cypress/screenshots/**/*.{png}
  expire_in: 1 day
```

5.7 Documentation

Pour ce qui est de la documentation, il s'agit du fichier README présent sur le dépôt, ce fichier étant au format Markdown, il a été possible d'y intégrer du code HTML afin de styliser la documentation.



The screenshot shows a documentation page for an application named "Application YAML". The page features a logo consisting of the letters "Ae" in a teal color. Below the logo, the title "Application YAML" is displayed in a bold, dark blue font. A horizontal line separates the header from the main content. The main content area contains the following text:

Application permettant de générer le fichier de configuration pour OpenADOM.
Développé par Antoine Delahaye.

A navigation sidebar on the left side of the content area lists sections:

- ▼ Sommaire
 - 1. Pour commencer
 - Prérequis
 - Commandes utiles
 - 2. À propos
 - Dépendances
 - Structure
 - Fonctionnement
 - Maintenance

Pour commencer

Prérequis

- NodeJS

```
# Debian
sudo apt install nodejs

# Arch
sudo pacman -S nodejs
```

Figure 5.11: La documentation

Nous avons donc un sommaire qui présente les six parties de la documentation, à savoir :

- Les prérequis, c'est-à-dire les logiciels nécessaires au lancement de l'application et comment les installer
- Les commandes utiles à l'application
- Les dépendances du projet, cela concerne tout framework et librairie utilisé
- La structure du projet afin de savoir comment est construit l'application et où trouver certaines ressources
- Le fonctionnement de l'application qui détaille les données stockées, comment les récupérer, les utiliser et comment est construit l'application
- La maintenance afin de garder une application stable et de pouvoir la faire évoluer en mettant à jour les dépendances ou en ajoutant de nouvelles fonctionnalités par exemple

6 Bilan

Travailler pendant quatre mois à l'INRAE a été une expérience très enrichissante pour moi, aussi bien sur le plan humain que technique. En effet j'ai beaucoup appris sur le monde en entreprise et dans le domaine de l'informatique grâce aux différentes personnes que j'ai côtoyées et avec lesquelles j'ai pu travailler. Ces dernières m'ont beaucoup appris grâce à leur expérience. Cela a donc été très enrichissant pour moi d'autant plus que j'ai pu suivre tout mon stage en présentiel contrairement à mon précédent stage.

Ce stage m'a donc grandement apporté sur le plan technique et organisationnel. Cela est dû à l'autonomie que mon maître de stage m'a donné et au fait d'être derrière moi en cas de besoin. En effet j'ai su monter en compétence dans des technologies et pratiques que je ne connaissais pas forcément. C'est le cas de VueJS où je n'avais eu que peu d'occasion de travailler sur ce framework et Figma, un outil très puissant que j'ai toujours voulu utiliser.

7 Conclusion

Comme expliqué précédemment, ce stage m'a beaucoup apporté. Cela est pour moi une expérience très enrichissante. J'y ai rencontré des personnes formidables et très intéressantes, j'ai dû faire face à des difficultés, m'adapter dans certaines situations et apprendre à communiquer avec les gens avec lesquels j'ai pu travailler. Au-delà de ça, j'ai eu la chance de travailler sur un sujet très complet, ce qui a été pour moi quelque chose de très intéressant, cela m'a permis de toucher à beaucoup de domaine de la programmation web et de la conception d'interface.

En fin de compte, je suis pour l'instant très satisfait de mon stage, il me reste encore un mois à travailler sur le projet et ce dernier avance bien. Toutes les fonctionnalités ne seront peut-être pas disponible à la fin, mais je suis confiant pour la suite grâce aux bases solide de ce dernier.