

Dossier de projet

Sabah Lorente

Formation : Développeur Web & Web Mobile 2022

Site : Afpa de Saint Etienne du Rouvray

Maître de stage : Yann Petit

Lieu du stage : Cleyrop

Période de stage : 19 septembre au 25 novembre 2022



Atlas
by cleyrop

Table des matières

INTRODUCTION	3
Compétences acquises	3
Cleyrop	3
Projet de stage :	4
Logiciels et outils utilisés pour ce projet	4
Pour coder	4
Langages, Frameworks	5
Outils backend :	7
Outils de référence pour le suivi du projet :	8
Mise en place du projet d'entreprise	9
Frontend	9
Installation SvelteKit :	9
Atomic design :	9
Conception :	11
Librairie :	24
Axes d'amélioration du projet en front.....	26
Backend	27
Préparation de la base de données :	27
Développement de l'API.....	30
Axe d'amélioration du projet en back.....	33
Les variables d'environnement	34
Outils de communications en entreprise et autres.....	35
Conclusion et remerciements	36
Annexes	38

INTRODUCTION

Compétences acquises

Front :

- Maquetter une application.
- Réaliser une interface utilisateur web statique et adaptable.
- Développer une interface utilisateur web dynamique.
- Réaliser une interface utilisateur avec une solution de gestion de contenu.

Back :

- Créer une base de données.
- Développer les composants d'accès aux données.
- Développer la partie backend d'une application web ou web mobile.
- Elaboration et mise en œuvre des composants dans une application de gestion de contenu.

Cleyrop



Présentation :

CLEYROP est une start-up née en 2021. Elle prône un DATAHUB souverain et européen. Elle propose à ses clients de valoriser leurs données et de les aider à tirer profit de son expertise.

HEMERA est le produit client, c'est une plateforme sur laquelle les données du client sont stockées. Hemera offre une interface unique qui permet de gouverner, cataloguer, orchestrer, superviser et monitorer les données et leurs traitements, dans un cadre sécurisé et industriel. Elle offre une étude minutieuse de ces dernières afin d'en tirer le meilleur parti. Ses clients sont d'horizons divers tels que la santé (les hôpitaux) et les finances (banques).



Projet de stage :

Le produit interne de **CLEYROP** est la plateforme **ATLAS**, c'est sur celle-ci que se porte mon projet de stage. CLEYROP a besoin de fournir à son équipe une visibilité en temps réel des différents environnements de travail déployés par **HEMERA**. Le site a donc pour but de répertorier les projets en cours.

Le menu permet de naviguer vers des liens externes et privés :

- Zammad pour le support
- Gitlab : Cleyrop-org
- Cleyrop.sharepoint

Le menu mène également sur la page d'accueil ainsi que sur la page « environnements ». Cette dernière donne une visibilité sur l'avancement de chaque projet en détails :

- Statut du projet
- Nom du projet
- Client
- Type (dev ou prod etc.)
- Responsable du projet
- Version
- Bastion
- Etat du projet (en favori ou non selon le choix de l'utilisateur)

A la charge de l'équipe **ATLAS** de créer un site avec un menu comportant la redirection vers les pages citées ci-dessus.

Puisque les pages externes existent déjà, il nous a fallu créer la page d'accueil **ATLAS** ainsi que la page des environnements.

Logiciels et outils utilisés pour ce projet

Pour coder :

Visual studio code est un **éditeur de code extensible développé par Microsoft pour Windows, Linux et MacOS**. J'ai personnalisé mon installation en ajoutant des extensions comme **Svelte pour VS code**, **Svelte 3 Snippet** afin de bénéficier de l'autocomplétions, **Git Graph** est une extension VSC que j'ai aussi utilisée afin de visualiser le repository. (cf. annexe 1 et 2)

Storybook est un **outil open source** qui permet de **développer les composants UI en complète isolation**, il est compatible avec la plupart des frameworks front-end ([Vue](#), [React](#), Angular ...). (cf. annexe 3)

Lors de notre projet **ATLAS**, Storybook m'a été très utile. Il me permettait de visualiser ce que je codais. Sa puissance émane aussi du fait que l'on puisse ajouter des options sur chaque composant afin d'observer leur comportement et choisir exactement la composition visuelle la plus appropriée.

Git est un logiciel de gestion de versions qui permet aux équipes de développeurs de travailler sur une fonctionnalité sans gêner les autres. Git permet de garder un historique des versions du projet.

Les principales commandes sont :

git switch -c afin de créer une branche et se positionner sur celle-ci.

git switch pour changer de branche.

git fetch --all récupère toutes les informations du serveur et nous permet de voir si nous sommes à jour.

git pull met à jour la branche locale depuis le serveur.

git rebase met à jour sur ce qu'a fait le reste de l'équipe et garde nos changements.

git add (+nom du fichier) pour ajouter son code.

git commit -m pour faire un commit (commande à laquelle on ajoute le commentaire).

git commit --amend modifie le dernier commit.

git push pour pusher le code sur le serveur. Ainsi le code peut-être en review par les personnes désignées qui le valideront (merge avec la branche principale) ou alors feront des commentaires pour des modifications futures.

Ceci est une liste non exhaustive de ce que **GIT** peut faire. Il est possible de forcer une suppression, de récupérer un code perdu etc. Je vous propose ci-dessus les principales commandes que j'ai utilisées durant mon stage.

[Langages, Frameworks](#)

Langages :

CSS

- Les « feuilles de style en cascade », également appelées **CSS**, sont un langage de conception simple destiné à simplifier le processus de création de pages Web présentables.
- CSS gère l'aspect, le style d'une page Web. CSS permet de contrôler la couleur du texte, le style des polices, l'espacement entre les paragraphes, la taille et la disposition des colonnes, les images ou les couleurs d'arrière-plan utilisées, les conceptions de mise en page, les variations d'affichage pour différents appareils et tailles d'écran, ainsi qu'une variété d'autres effets comme des animations.
- CSS est facile à apprendre et à comprendre, mais il offre un contrôle puissant sur la présentation d'un document HTML. Le plus souvent, CSS est combiné avec les langages de balisage HTML ou XHTML.

CSS est un incontournable pour gérer toute la partie liée au style.

Pour mon projet, j'ai utilisé SvelteKit, chaque composant possède son propre style CSS.

Le SCSS regorge de fonctionnalités avancées comme l'utilisation de variables afin de raccourcir le code et c'est justement ce que j'ai utilisé ce qui m'a permis de gérer toute la partie « esthétique » de la plateforme.

Le nesting sert justement à faciliter l'organisation du code :

```
<style lang="scss">
  @import "../_ions/radius";

  p {
    display: none;
    padding: 1em;
    line-height: 24px;
    letter-spacing: 0.0012em;

    font-family: 'Roboto', sans-serif;
    font-size: 16px;
    font-weight: 400;

    background-color: var(--branding-500);
    color: var(--gray-50);
  }

  @include radius-md;

  &.visible {
    display: flex;
  }

  .icon {
    display: flex;
    align-items: center;
    margin-right: 0.5em;
  }
</style>
```

Annotations:

- `@import` permet d'appeler le fichier dans lequel les radius sont répertoriés et convertis en variables.
- Variables de couleurs répertoriées dans le dossier static/color.css
- Ici, on utilise le symbole « & » afin de concaténer les propriétés

SVELTE

❖ Sa petite histoire...

Svelte est un framework JavaScript orienté composants et réactivité. Il a été créé par Rich Harris en novembre 2016. Son but était d'obtenir un framework qui produise du code compilé ultra léger et rapide à l'exécution.

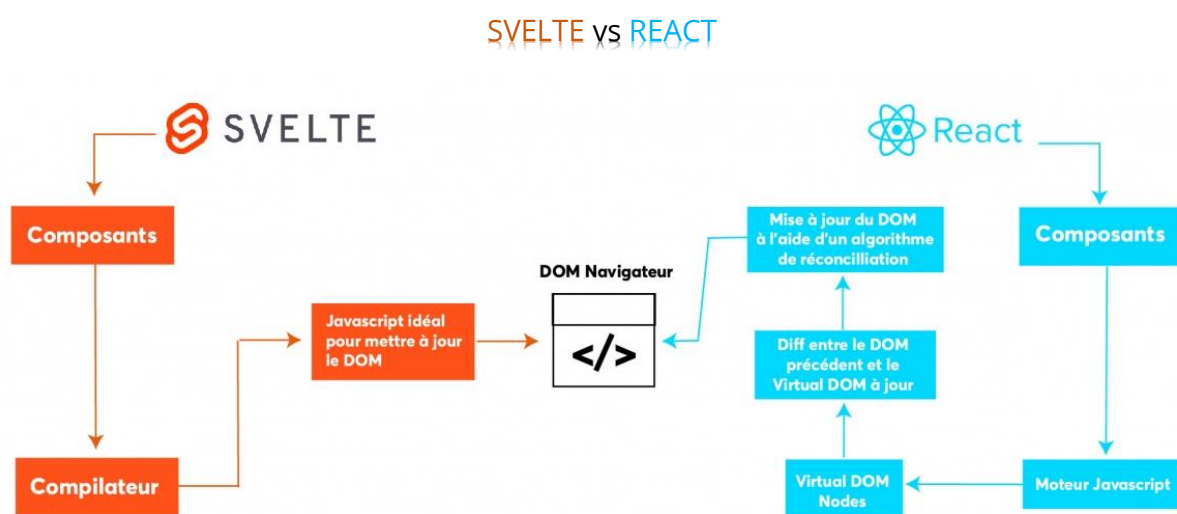
Actuellement, c'est la version 3 sortie en 2019 que j'utilise.

SvelteKit est aussi un framework, cette fois full-stack, il est encore dans sa version bêta depuis mars 2021. Une extension VS Code est maintenue par l'équipe Svelte. On peut également compter sur la communauté très active sur Discord ou StackOverflow.

❖ Svelte, sa vélocité :

La différence principale est que Svelte n'interprète pas le code à l'exécution, comme le font les autres frameworks JavaScript. Svelte, au moment du build, compile tout son code en code Vanilla JS. Ce qui signifie qu'il n'a pas besoin de moteur d'interprétation pour lire le code à la volée, c'est le moteur de rendu du navigateur web qui est utilisé.

Svelte est un compilateur qui prend les composants et les transcrit en Javascript pur.



A noter que l'équipe **HEMERA** utilise du REACT pour la partie front de son produit.

Outils backend :

RethinkDB fournit une interface Web qui nous a permis de gérer notre base de données. Il est possible de faire des *imports* pour charger les données existantes dans la base de données RethinkDB. Il est possible de lire des fichiers JSON ou CSV. ReQL est le langage de requête de RethinkDB, il permet de manipuler les documents JSON.

Docker est une plate-forme de containerisation qui regroupe l'application et ses dépendances dans un container afin que l'application fonctionne de manière transparente dans n'importe quel environnement.

Il s'agit d'un outil conçu pour faciliter la création, le déploiement et l'exécution d'applications à l'aide de containers. Les containers Docker sont légers, alternatives aux machines virtuelles et ils utilisent le système d'exploitation hôte ce qui permet de ne pas pré-allouer de RAM dans les containers comme on le fait dans les machines virtuelles.

Grâce à Docker, les développeurs peuvent créer, packager, livrer et exécuter des applications en toute simplicité, sous forme de containers légers, portables et autonomes, qui peuvent fonctionner pratiquement n'importe où. Les containers permettent aux développeurs de conditionner une application avec toutes ses dépendances et de la déployer en une seule unité. En fournissant des containers d'applications prédéfinis et autonomes, les développeurs peuvent se concentrer sur le code de l'application et l'utiliser sans se soucier du système d'exploitation sous-jacent ou du système de déploiement.

Outils de référence pour le suivi du projet :



Cet outil permet de déployer l'agilité en entreprise afin de travailler avec efficacité et fluidité. Tuleap possède des outils de gestion de projet AGILE Scrum qui permettent à chaque équipe de s'organiser comme elle le souhaite. Dans notre équipe, nous utilisons Tuleap pour créer des tâches (Team_enablers) ainsi que des sous-tâches (enabler_tasks).

Voici comment je procédais :

Sur la maquette Figma, je choisisais un groupe de composants, je listais les composants liés et créais des tâches associées en détaillant au maximum : propriétés CSS, utilité, comportement attendu etc. Ensuite, je prenais le temps d'évaluer l'effort estimé, son temps de création et son degré de priorité. Puis j'assignais chaque tâche à une personne de l'équipe.

Le tableau (Agile Dashboard) permet ensuite de visualiser le statut du travail à effectuer. (cf annexe 4)

Statut des tâches :


- READY : lorsque celle-ci n'attend plus que d'être codée.
- WAITING : lorsque la tâche est en attente.
- ON GOING : lorsque celle-ci est en cours de codage.
- IN REVIEW : quand la tâche est pushée et attend que la personne assignée la revoie et/ou la corrige.

Scrum recommande de nommer un **scrum master** (dans mon cas, c'était mon maître de stage). Son rôle est de garantir la mise en œuvre de l'outil agile (Tuleap) et de piloter les quatre étapes d'un sprint scrum : planification, daily meeting, sprint review et sprint rétrospective. Élément central pour le bon fonctionnement de l'équipe projet, le scrum master est aussi le garant de la fluidité des échanges et de la productivité du travail. A ce titre, il identifie les points de blocage et anime les brainstormings pour cerner les solutions.



Toute l'équipe utilise GitLab qui est une plateforme de développement collaborative open source éditée par la société américaine du même nom. Elle couvre l'ensemble des étapes du DevOps. Se basant sur les fonctionnalités du logiciel Git, elle permet de piloter des dépôts de code source et de gérer leurs différentes versions.

On peut lier Gitlab et Tuleap en paramétrant les plateformes et en utilisant une convention de nommage spécifique lors des commits par exemple.

 **Figma** Cleyrop possède son équipe d'UX-designers qui s'est chargée de nous fournir toutes les maquettes nécessaires à notre projet. La version sur laquelle nous travaillons se nomme **DS Athena 1.0** mais l'équipe fera dès 2023 une migration vers la **version 1.1**. Sur Figma, il est pratique de pouvoir communiquer avec cette équipe en postant des questions sur le composant concerné directement à l'emplacement de ce dernier. Il est également possible à l'équipe UX-designers de nous laisser des messages pour faciliter notre travail. (cf annexe 5 et 6)

Mise en place du projet d'entreprise

Frontend

Installation SvelteKit :

La documentation SvelteKit est très bien expliquée.

Voici les commandes d'installation:

1. `pnpm create svelte@latest my-app` permet de créer le dossier.
2. `cd my-app` : cette commande permet de faire un « change directory », donc de se positionner dans le dossier que l'on vient de créer.
3. C'est avec `pnpm install (ou pnpm i)` que l'on installe le node.module avec toutes les dépendances dont svelteKit (ex : builder vite).
4. `pnpm run dev` sert à lancer l'application.

**Mon tuteur de stage m'a vivement encouragée à utiliser la commande « pnpm », gestionnaire de packages plus léger et plus performant selon lui.*

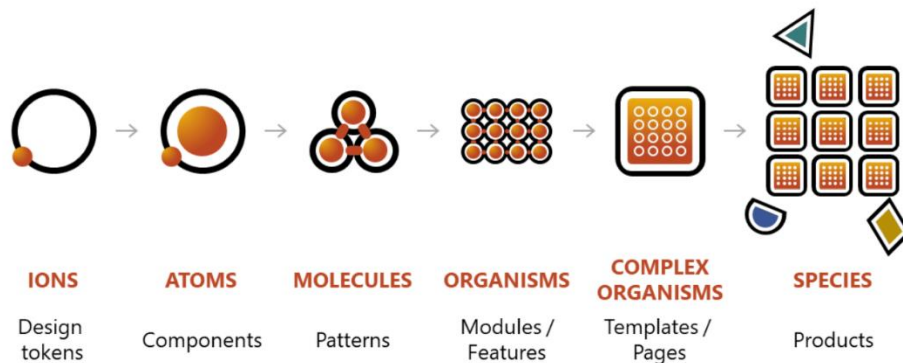
Atomic design :

Il se réfère à une hiérarchie de taille, tel qu'on peut l'imaginer en physique atomique : ions, atome, molécule, puis cellules ou organismes, le tout permettant finalement de concevoir des templates et des pages.

L'idée est de réduire un site à ses plus petits composants, lesquels peuvent ensuite être assemblés pour constituer des modules de plus grande taille. Ces modules peuvent eux-

mêmes être accolés pour constituer l'ossature d'une page puis d'un site, qui s'affiche plus correctement sur tous types d'écrans.

Les cinq états de la conception atomique sont :



Pourquoi dit-on *Atomic Design* ?

Ce lien thématique avec la chimie décrit parfaitement les atouts de cette méthode :

- ❖ Chaque élément peut être indépendant.
- ❖ Chaque composant supérieur repose sur des composants plus petits.
- ❖ En combinant des éléments, on obtient d'autres éléments encore plus complexes et plus fonctionnels.
- ❖ Les combinaisons d'éléments sont infiniment variées.

Avantages :

Ce sont de petits composants réutilisables par différentes équipes dans différents projets donc gain de temps sur le long terme.

Inconvénients :

Le projet met plus de temps à se concrétiser car il faut prendre soin de faire un code réutilisable donc comprenant des affichages dynamiques et non en durs.

Il faut ensuite prendre le temps de comprendre la mécanique d'assemblage pour créer une page avec tous les éléments nécessaires.

Pour finir :

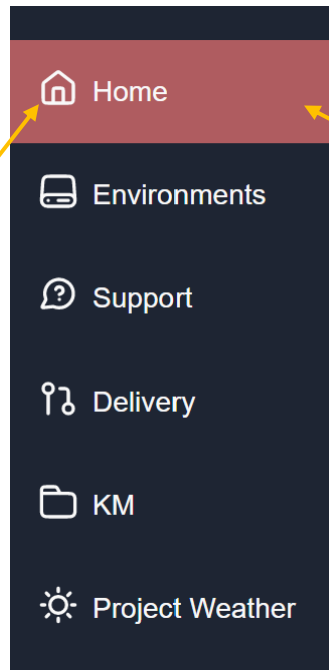
C'est un bon système, l'organisation est bien définie, les tâches de chacun également. Cela permet de puiser dans les forces de chaque membre de l'équipe pour atteindre l'objectif fixé. L'Atomic design permet d'avoir un code qui perdure et est un réel atout lorsqu'un nouveau développeur doit prendre le relai.

Conception :

Commençons par le **Menuitems.svelte** qui est un composant du **Menu.svelte** :



Menuitem.svelte est une molécule. Ici on retrouve ce composant 6 fois, il représente chaque ligne du menu, il est donc répété de manière dynamique. Il comporte un **logo**, un **label** ainsi que le **lien cliquable**.



&.selected en SCSS permet de modifier la couleur de l'item quand le lien qu'il comporte est identique au chemin dans la barre de navigation.

CODE :

`<li class="selected">` `` est un raccourci svelte.

```
<template>
  <li class:selected>
    <a href="{href}" target="{target}">
      <span class="icon"
        >{#if icon}<Icon icon="{icon}" />{/if}</span>
    </a>
  </li>
</template>
```

class :selected nous servira pour le SCSS, il applique la class « **selected** » si la variable **selected = true**.

```
<style lang="scss">
  &.selected {
    background-color: #ff7373b2;
  }
```

```

<script>
  import Icon from '../_ions/Icon.svelte';
  export let label = '';
  export let href = '';
  export let icon = '';
  export let selected = false;
  export let target = '';
</script>

```

Selected est un boolean, il est donc : *TRUE* ou *FALSE*.

Ici, par défaut il est sur *FALSE*. La logique et le changement de statut se font sur le fichier **+layout.svelte** :

(inside)/+layout.svelte :

```

<template>
  <div class="container">
    <Menu>
      <MenuItem icon="home-02"
        label="{${t('home.menu.home')}}"
        href="/"
        selected="{${page.url.pathname === '/'}}"/>
    </Menu>
  </div>
</template>

```

\$page est un store de svelte qui donne accès à toutes les infos de la page courante.

```

{
  error: null,
  params: {},
  route: { id: '/(inside)' },
  status: 200,
  url: URL {
    href: 'http://127.0.0.1:5173/',
    origin: 'http://127.0.0.1:5173',
    protocol: 'http:',
    username: '',
    password: '',
    host: '127.0.0.1:5173',
    hostname: '127.0.0.1',
    port: '5173',
    pathname: '/',
    search: '',
    searchParams: URLSearchParams {},
    hash: ''
  },
  data: { '0': undefined, user: undefined },
  form: null
}

```

On récupère ainsi toutes informations nécessaires afin de construire notre logique dans la console.

Qu'est-ce qu'un store ?

Ici :

```
import {page} from '$app/stores';
```

Un store est un objet qui contient des données et accède à ces données à partir de différentes parties de l'application. Les stores Svelte sont une fonctionnalité intégrée qui facilite la gestion de l'état de l'application. Ils sont réactifs, ce qui signifie qu'ils mettent à jour les composants chaque fois que les données changent dans l'application. En bref, les

stores sont des données globales qui contiennent des valeurs ou un objet. Les composants s'abonnent aux stores et reçoivent des notifications lorsque leurs valeurs changent.

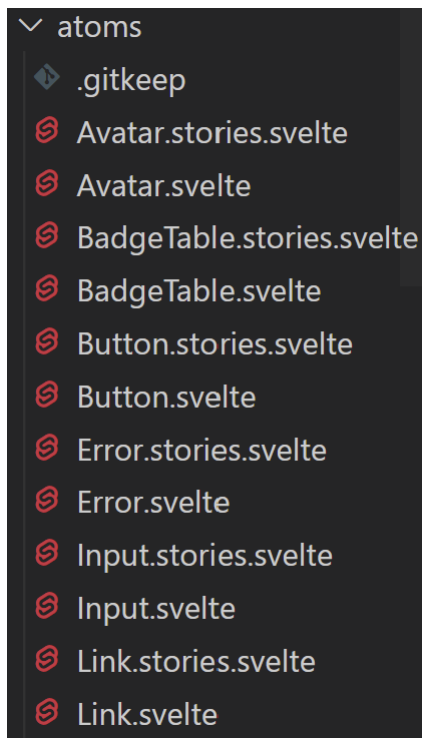
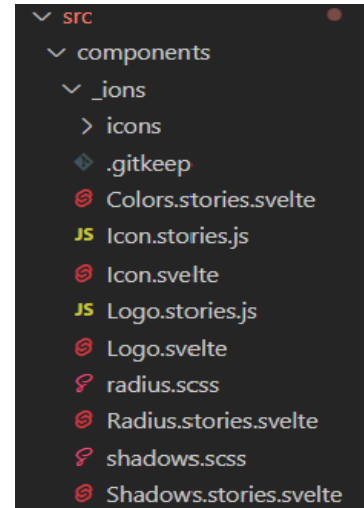
Création du Menu.svelte :

Reprenons le modèle d'Atomic design :

Les ions sont les composants les plus petits.

Dans notre projet, le dossier ions comprend :

- ✓ Les variables CSS: color, shadow, radius
- ✓ Les icônes
- ✓ Le logo

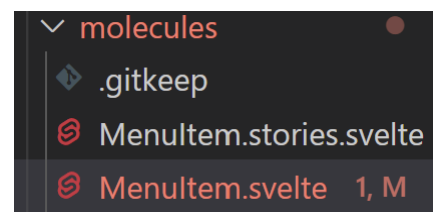


- Les atomes :

L'**atome** suit l'ion en ordre de grandeur, il correspond à n'importe quel élément de base d'une interface. Ce sont par exemple les inputs (cases d'éditations), les boutons, les labels, etc.

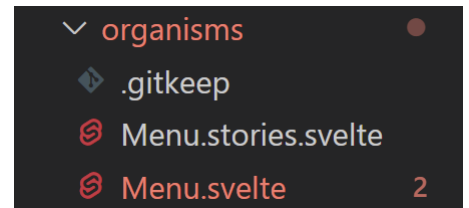
- Les molécules :

Une **molécule** est tout simplement l'assemblage de plusieurs atomes.



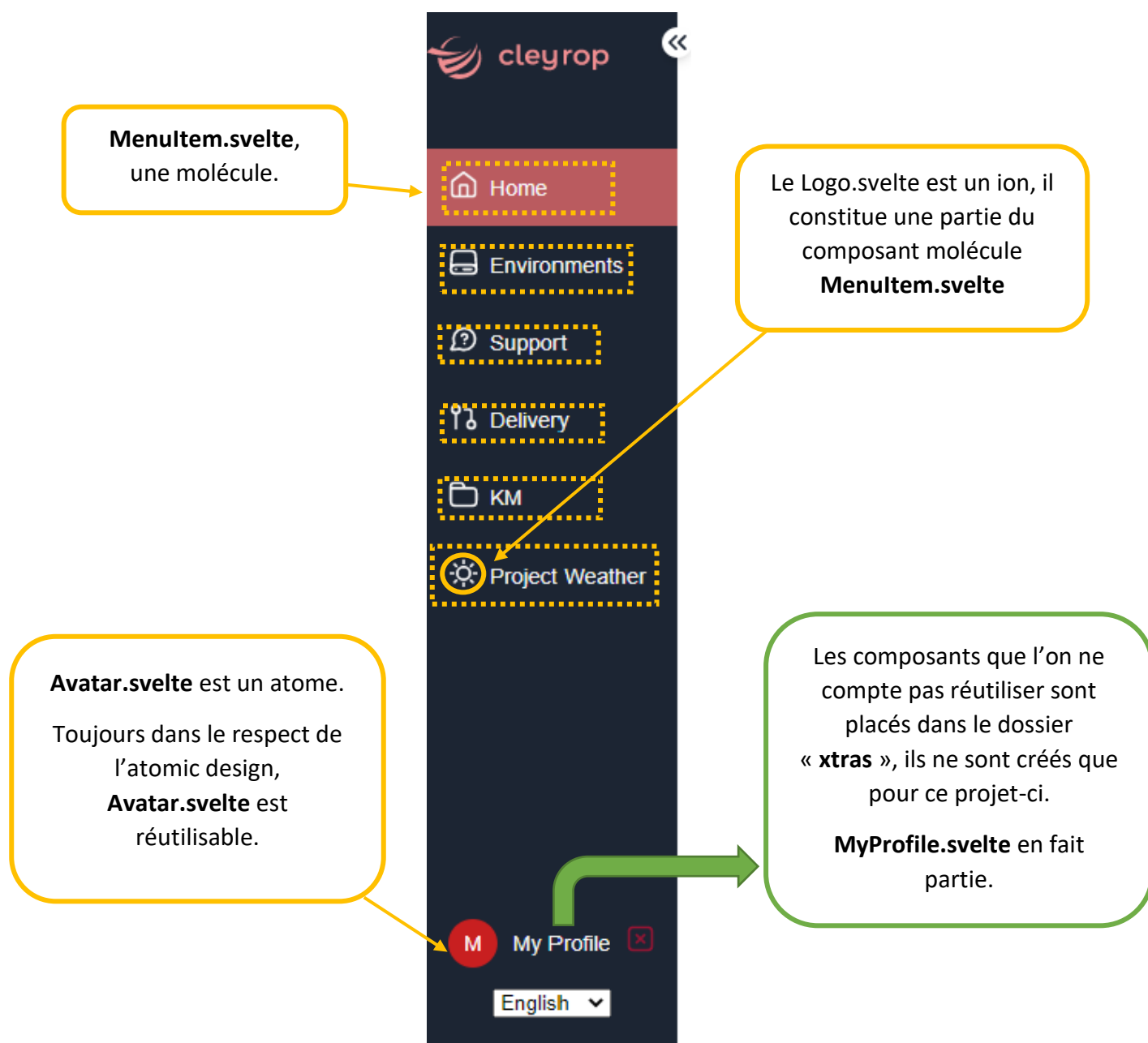
- Les organismes :

Les organismes sont un assemblage d'ions, d'atomes et de molécules, ils définissent des composants d'interfaces complexes. Ces organismes vont permettre de créer des interfaces très rapidement. À la différence des molécules, ils n'ont pas forcément un seul objectif. Par exemple, un header peut permettre : la navigation sur un site, effectuer une recherche et retourner sur la page d'accueil en appuyant sur le logo du site web.



- Les organismes dans notre projet : le **Menu.svelte**





Puis vient ce que l'on nomme le **+layout.svelte** qui se situe dans le dossier `routes/(inside)/`. **+layout.svelte** est à la racine de `inside` qui correspond à l'ensemble des pages internes quand on est connecté.

+layout.svelte réunit tous les éléments qui constitueront enfin la « **page** » que l'on veut afficher.

Dans la logique Svelte, on retrouve principalement trois types de balises :

- `<script></script>` englobent les « import », « export » et la logique JS.
- Le HTML : ici, l'équipe a configuré le code pour que tout notre HTML soit toujours entre ces balises `<template></template>`.

- Le CSS se trouve entre les balises <style></style>.

```

<script>
  import {page} from '$app/stores';
  import Menu from '../components/organisms/Menu.svelte';
  import MenuItem from '../components/molecules/MenuItem.svelte';
  import LanguageSelector from
  '../components/xtras/LanguageSelector.svelte';
</script>

<template>
  <div class="container">
    <Menu >
      <MenuItem icon="home-02"
        label="{${('home.menu.home')}}"
        href="/"
        selected="{${page.url.pathname === '/'}}"/>
      <MenuItem icon="server-03"
        label="{${('home.menu.env')}}"
        href="/envs"
        selected="{${page.url.pathname === '/envs'}}"/>
      <MenuItem icon="message-question-circle"
        label="{${('home.menu.support')}}"
        href="{ZAMMAD_URL}"
        target="_blank"/>
      <MenuItem icon="git-pull-request"
        label="{${('home.menu.delivery')}}"
        href="{GITLAB_URL}"
        target="_blank"/>
      <MenuItem icon="folder" label="{${('home.menu.km')}}" href="{KM_URL}"
target="_blank"/>
      <MenuItem icon="sun"
        label="{${('home.menu.weather')}}"
        href="{WEATHER_URL}"
        target="_blank"/>
      <div class="lang" slot="lang"><LanguageSelector /></div>
    </Menu>
    <slot />
  </div>
</template>

<style lang="scss">
  .container {
    display: flex;
  }
</style>

```

Le <slot/> correspond à la page que l'on appelle.

Le Menu sera toujours présent sur le côté de chaque page interne à laquelle s'ajoute la page que l'utilisateur souhaite afficher : la page sur laquelle on navigue et qui correspond à l'URL.

Rendu sur la page web :



La page « environnements » :

La page « environnements » est composée de la **+page.svelte** qui se situe dans le dossier « **envs** » ainsi que du **+layout.svelte** qui lui se trouve dans **routes/(inside)/+layout.svelte**. Le **+layout.svelte** est constitué du **menu** et du **<slot/>** qui lui renvoie à **routes/(inside)/envs/+page.svelte**.

Status	Name	Customer	Type	Owners	Version	Bastion	Favorite
backupt	ENVDEV1		QA		develop	bastion-dev	☆
backupt	ENVDEV2		dev		develop	bastion-dev	☆
backupt	ENVDEV3		dev		develop	bastion-dev	☆
unknown	ENVDEV4		dev		develop	bastion-dev	☆
unknown	ENVDEV5		dev		develop	bastion-dev	☆
error	ENVDEV6		dev		develop	bastion-dev	☆
backupt	ACPR01	ACPR	dev		1.2	bastion-dev	☆
backupt	ACPR02	ACPR	dev		1.2	bastion-dev	☆
backupt	ACPR03	ACPR	dev		1.2	bastion-dev	☆
backupt	ATT-Dev	ATT	dev		1.2	bastion-dev	☆
backupt	ATT-Prod	ATT	prod		1.2	bastion-dev	☆
backupt	ATT-Staging	ATT	staging		1.4	bastion-dev	☆
backupt	HDH-Dev	HDH	dev		1.2	bastion-dev	☆
backupt	Invest-Dev	Invest	dev		1.2	bastion-dev	☆
backupt	MIFIH-Dev	MIFIH	dev		1.2	bastion-dev	☆

Récapitulatif de ce que les fichiers contiennent :

Ions	Atomes	Molécules	Organismes	Pages
Icon	Link	MenuItem	Menu	+page.svelte Chemin : routes/(inside)/envs/+page.svelte
	Favorite	Header	Table	+layout.svelte Chemin : routes/(inside)/+layout.svelte
	Sorted	Line		
	Filtered			
	Status			
	Language selector			

Le code du composant **<Table>** :

Entre les balises `<script>`, on retrouve les importations nécessaires à l’affichage.

Entre les balises `<template>`, nous retrouvons la balise `<table>` car ce fichier affiche précisément un tableau. Nous faisons appel au composant **<EnvBackupHeader/>** qui est l’entête du tableau (cf. illustration ci-dessous). La boucle **{#each}** dans une syntaxe propre à svelte est présente afin de boucler sur les informations du fichier `json = envlist.json`.

{#each environments as env} permet donc de boucler sur le `json` afin d’afficher chaque information (cf. l’extrait JSON ci-dessous). Pour chaque élément dans la boucle, un nouveau composant **<EnvBackupLine/>** est appelé. Et nous lui donnons en attribut toutes les informations de l’environnement.

status="{env.status}" : permet d’envoyer le statut de manière dynamique : « **backupid, error ou unknow** » en fonction des valeurs passées : `true`, `false` ou rien du tout.

Table.svelte :

```
<script>
  import environments from './envlist.json';
  import EnvBackupHeader from './EnvBackupHeader.svelte';
  import EnvBackupLine from './EnvBackupLine.svelte';
</script>

<template>
  <table>
    <EnvBackupHeader/>
    {#each environments as env}
      <EnvBackupLine status="{env.status}"
        name="{env.name}"
        url="{env.url}"
        customer="{env.customer}"
        type="{env.type}"
        owners="{env.owners}"
        version="{env.version}"
        bastion="{env.bastion}" />
    {/each}
  </table>
</template>
```

Envlist.json :

```
[
  {
    "id": 279528,
    "name": "HDH Dev",
    "description": "",
    "type": "dev",
    "customer": "HDH",
    "owners": ["Franck"],
    "status": true,
    "next": "2022-10-03T02:30:00.000Z",
    "url": "https://cleyrop.demo-sante.cleyrop.tech/",
    "bastion": "bastion-dev",
    "version": "1.2"
  },
]
```

Environments

Here you can find the list of our currently running environments and their backup status.

http://localhost:8888/q/swagger-ui/#/Environment%20Controller/get_atlas

Status	Name	Customer	Type	Owners	Version	Bastion	Favorite
- All -	- All -	- All -	- All -	- All -	- All -	- All -	
✓ backedup	HDH Dev	HDH	dev	Franck	1.2	bastion-dev	★
✓ backedup	MIPIH Dev	MIPIH	dev	Franck	1.2	bastion-dev	★
✓ backedup	ACPR03	ACPR	dev	Franck	1.2	bastion-dev	☆
✓ backedup	ACPR02	ACPR	dev	Franck	1.2	bastion-dev	☆
✓ backedup	ACPR01	ACPR	dev	Franck	1.2	bastion-dev	☆
✓ backedup	Invest Dev	Invest	dev	Lauren	1.2	bastion-dev	★
⚠ error	ENVDEVELOP		dev	Luis Joan	develop	bastion-dev	☆
⚠ unknown	ENVDEV4		dev	Luis Joan	develop	bastion-dev	★
⚠ unknown	ENVDEV5		dev	Luis Joan	develop	bastion-dev	☆
⚠ unknown	ENVDEV6		dev	Luis Joan	develop	bastion-dev	☆
✓ backedup	ENVDEV3		dev	Luis Joan	develop	bastion-dev	☆
✓ backedup	ENVDEV2		dev	Luis Joan	develop	bastion-dev	★
✓ backedup	ENVDEV1		QA	Lauren Jean	develop	bastion-dev	★
✓ backedup	ATF Dev	ATF	dev	Romain NicolasC	1.2	bastion-dev	★
✓ backedup	ATF Prod	ATF	prod	Sandie	1.2	bastion-atf	☆
✓ backedup	ATF Staging	ATF	staging	Sandie	1.4	bastion-atf	★

Src/routes/(inside)/envs/+page.svelte :

```

<script>
  import Textzone from '../../../components/atoms/Textzone.svelte';
  import Table from '../../../components/xtras/EnvBackupTable.svelte';
  import {t} from '../../../../translation/i18n';
</script>

<template>
  <div class="container">
    <h1>{t('env.title')}</h1>
    <Textzone text="{t('env.text')}" />
    <Link href="http://localhost:8888/q/swagger-ui/#/Environment%20Controller/get_atlas" target="_blank" />
    <div class="scrollable">
      <Table />
    </div>
  </div>
</template>

<style lang="scss">
  .container {
    margin: 0 auto;
    .scrollable {
      overflow-y: auto;
      max-height: 800px;
    }
  }
</style>

```

Entre les balises <script> nous faisons appel aux différents composants nécessaires : le <Textzone/>, <Link/>, <Table/> et {t} qui correspond à l'internationalisation.

Ici, le titre en <h1> est traduit de manière dynamique avec le **i18n** intégré. Le paragraphe est en fait le composant <Textzone/>, ensuite le lien <Link/>. Enfin, le composant <Table/> qui représente le tableau des environnements.

Le Login :

Voici le formulaire de connexion sécurisé pour accéder à la plateforme Atlas.

Son fonctionnement :

Emplacement :

- routes/(outside)/login/+page.svelte

```
let username = '';  
<Input type="text"  
  name="username"  
  placeholder={`$t('login.username')}`  
  focused  
  required  
  bind:value="{username}" />
```

Cela permet de lier une variable à un élément HTML et de mettre à jour la valeur de {username}.

Dans l'action du formulaire, on soumet la route « /login », puis il suffit d'ajouter « use:enhance » pour que le navigateur fasse tout ce qu'il doit faire sans recharger la page.

```
import {enhance} from '$app/forms';  
<form method="post" action="/login" use:enhance>
```

Si l'un des deux champs est vide alors le bouton est **disabled**. Disabled sert à désactiver le ou les champs.

\$: permet de réévaluer la valeur et de la modifier. C'est ce que l'on nomme une « déclaration réactive » (« reactive déclaration » en anglais). Si une variable dans l'expression ci-dessous change, le \$: permet de mettre à jour la valeur de « **disabled** ».

```
<script>
$: disabled = username.length == 0 || password.length == 0;
</script>
```

```
<template>
<Button label="{${t('login.login')}}" size="md" disabled="{disabled}" />
</template>
```

- *routes/(outside)/login/+page.server.js*

```
src > routes > (outside) > login > JS +page.server.js > ...
1  import {invalid, redirect} from '@sveltejs/kit';
2
3  export const actions = {
4    default: async ({request, cookies}) => {
5      const data = await request.formData();
6      // FIXME use real backend auth here
7      if (data && 'Atlas' === data.get('username') && 'Cleyrop' === data.get('password')) {
8        cookies.set('session', 'user', {
9          path    : '/',
10         httpOnly : true,
11         sameSite : 'strict',
12         secure   : process.env.NODE_ENV === 'production',
13         maxAge   : 60 * 10
14       });
15
16         throw redirect(307, '/');
17       }
18       return invalid(401, {invalid: true});
19     }
20   };

```

Méthodes de Svelte

En ligne 3, on exporte des actions pour récupérer les données du formulaire envoyées en **POST**.

En ligne 5, pour récupérer le contenu de la requête, on utilise un « **await request.formData()** ». « **Const data** » donne le résultat de la requête passée par le formulaire et comme c'est un objet avec toutes les infos dedans, on utilise **data.get('username')** pour récupérer chaque information à l'intérieur. En faisant donc le « **data.get('username')** » et « **data.get(password)** », on récupère les infos passées en string pour mieux traiter les données.

En ligne 7, on pose une condition qui traduit donnerait : 'Si dans les données on récupère la valeur exacte du **'username'** et **'password'** alors on crée une session cookie'.

En ligne 16, on indique une redirection en cas de succès vers la page d'accueil.

En ligne 18, le formulaire devient invalide et affiche une erreur si le **'user'** ou le **'password'** est invalide.

L'utilisation du « hook » :

```
src > JS hooks.server.js > [x] handle
1  const redirect = (location) =>
2    new Response(null, {
3      status : 307,
4      headers : {location}
5    });
6
7  export const handle = async ({event, resolve}) => {
8    const session = event.cookies.get('session');
9    if (!session && event.url.pathname !== '/login' && event.url.pathname !== '/logout') {
10     return redirect('/login');
11   }
12   if (session && event.url.pathname === '/login') {
13     return redirect('/');
14   }
15   return await resolve(event);
16 };
```

En ligne 7, la fonction « **handle** » est utilisée dans le **hook**. Cette fonction est appelée lors de chaque événement qui a lieu dans notre application (changement de page ou requête).

En ligne 8, on va chercher dans les cookies la présence d'un cookie nommé « **session** ». On récupère la valeur de la clé session si le cookie est présent sinon la valeur devient « **undefined** ».

En ligne 10, si la session est inexistante (**!session c'est-à-dire session=undefined**) et que l'URL est différente de **/login** ou **/logout** alors on est redirigé vers le **/login**.

En ligne 13, si la session est trouvée et qu'on va sur la page **/login** alors on est redirigé vers la page d'accueil.

Déconnexion de l'utilisateur :

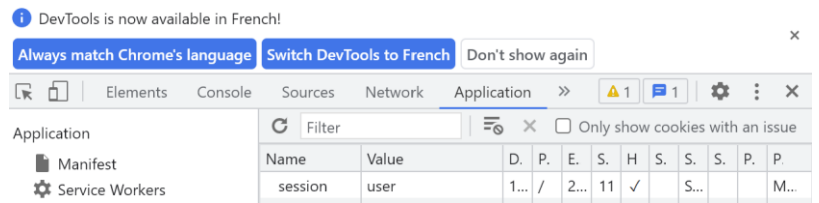
```
import {redirect} from '@sveltejs/kit';

export const load = async ({cookies}) => {
  cookies.set('session', '', {
    path : '/',
    expires : new Date(0)
  });
  throw redirect(307, '/login');
};
```

La fonction '**load**' est une fonction qui est appelée à chaque fois que la route **/logout** est chargée. Elle redéfinit le cookie nommé « **session** » en lui attribuant une valeur nulle et lui attribue une date d'expiration négative ce qui entraîne l'effacement du cookie.

Où sont les cookies ? 🍪

On peut récupérer les cookies générés sur le navigateur en suivant le chemin ***inspecter/application/cookies***.



Les failles XSS :

Il faudra également rester attentif aux **failles XSS** qui font partie des « *vulnérabilités par injection de code* ». Il s'agit pour un attaquant d'injecter du code malveillant via les paramètres d'entrée côté client (ex : input).

L'avantage est que Svelte échappe automatiquement, c'est-à-dire qu'il affiche les chaînes de caractères en texte brut.

Exemple : Cette phrase contient du `HTML!!!`

En revanche, il reste possible d'interpréter du HTML avec Svelte en utilisant le TAG `@html` (à mettre avant le nom de la variable dans les accolades), il faudra dans ce cas veiller à échapper nous-mêmes les entrées utilisateur en remplaçant la balise `<` par `<` et `>` par `>`.

Librairie :

Internationalisation, **svelteKit-i18n** :

Dans mon projet, il est possible de traduire le site en anglais et en français. Ici, je prendrai l'exemple de la page `/login`.

translation/i18n.js :

```
import i18n from 'sveltekit-i18n';
const config = {
  fallbackLocale : 'en',
  loaders : [
    {
      locale : 'en',
      key : 'login',
      loader : async () => (await import('./lang/en/login.json')).default
    },
    {
      locale : 'fr',
      key : 'login',
      loader : async () => (await import('./lang/fr/login.json')).default
    }
  ]
};
export const {t, locale, loadTranslations} = new i18n(config);
```

« i18n » est une librairie

On crée une constante config dans laquelle on met :

fallbackLocale est en anglais, c'est la traduction par défaut si aucune autre langue n'est choisie.

Loader donne le chemin du fichier JSON à récupérer pour la traduction en fonction de la langue choisie.

Ensuite, on exporte la constante avec :

- **t** qui contient toutes les traductions.
- **locale** qui permet de sélectionner la langue souhaitée.
- **i18n** est donc une librairie que l'on l'instancie en lui donnant en paramètre le fichier de config. `(new i18n(config));`.

{ } login.json :

Ce sont des fichiers de traductions que l'on récupère selon la langue choisie (un fichier par page et par langue).

```
{
  "welcome": "Welcome to Atlas",
  "username": "Username",
  "password": "Password",
  "login": "LOGIN",
  "error": "You have entered the wrong credentials."
}
```

C'est à l'aide du composant **LanguageSelector.svelte** que l'utilisateur peut changer la langue du site.

```
src > components > extras > LanguageSelector.svelte > script
1  <script>
2    import {locale} from '../translation/i18n';
3
4    const languages = [
5      {code: 'en', name: 'English'},
6      {code: 'fr', name: 'Français'}
7    ];
8
9    $: selected = locale.get();
10
11    const handleOnChange = () => {
12      document.cookie = `language=${selected};path=/`;
13      locale.set(selected);
14    };
15  </script>
16
17  <template>
18    <select bind:value="{selected}" on:change="{handleOnChange}">
19      {#each languages as {code, name}}
20        <option value={code}>{name}</option>
21      {/each}
22    </select>
23  </template>
```

{locale} permet d'enregistrer la langue souhaitée par l'utilisateur. On peut utiliser les méthodes **locale.get()** et **locale.set()** pour accéder à la valeur actuelle ou la modifier.

Le **select** est par défaut sur « en » (anglais). Quand on change sa valeur, on appelle la fonction « **handleOnChange** » qui crée un cookie nommé « language » et qui pour valeur la valeur de notre **select**. La valeur de la langue est alors actualisée dans **locale**.

Il faut alors charger les fichiers de traduction pour chacune des pages, on utilise pour cela le **layout**.

+layout.js :

```
import {loadTranslations, locale} from '../translation/i18n';

export async function load({url}) {
  return await loadTranslations(locale.get() ||
    document.cookie.replace('language=', ''), url);
}
```

Situé à la racine du dossier **routes**, ce fichier est commun à toutes les pages du site.

On importe la langue voulue depuis **{locale}**.

La fonction **load** est appelée à chaque chargement de page et prend l'**{url}** en paramètre. Cela permet de renvoyer le chargement des traductions avec soit **locale.get** (langue par défaut si aucun cookie n'a encore été créé) soit la valeur dans le cookie si celui-ci est existant.

{url} permet d'atteindre le fichier JSON qui correspond à la page appelée.

L'affichage dynamique de la langue sur chaque page est rendu grâce à des variables.

+page.svelte :

```
<script>
import LanguageSelector from
'../../../../../components/xtras/LanguageSelector.svelte';
import {t} from '../../translation/i18n';
</script>
```

On importe le fichier de traduction avec le **{t}**.

Puis on utilise le « **\$t** » à chaque élément que l'on veut traduire.

Par exemple :

```
<h2>{$t('login.welcome')}</h2>
```

Axes d'amélioration du projet en front

Il est certain que la sécurité n'est pas effective sur ce projet interne et c'est normal car c'est un produit en cours de construction néanmoins l'utilisation de Keycloak est fortement envisagée en ce qui concerne l'authentification utilisateur. Les autorisations peuvent être gérées à partir de la console d'administration de Keycloak et nous donnera la possibilité de définir exactement les politiques dont nous aurons besoin. Cette solution permettra aux utilisateurs de gérer leur propre compte, mettre à jour leur profil, modifier leur mot de passe et activer l'authentification à deux facteurs.

Le site n'est pas responsif (absence de media queries en CSS) et les normes d'accessibilité ne sont pas respectées. La prochaine version d'ATLAS prendra en compte ces aspects.

Le back n'étant pas développé, le fichier JSON devra être remplacé par les données d'un backend codé en Quarkus et Kotlin ce qui permettra d'obtenir un affichage dynamique des données.

Backend

Pour le backend, la tâche qui nous incombait était de créer une API. Pour ce faire, j'ai mis en place des traqueurs conçus pour recueillir certaines données utilisateurs. Ceci aura pour but de faire de l'a/b testing sur notre site ce qui consiste à comparer deux versions d'une page web ou d'une application afin de vérifier laquelle est la plus performante. Ces variations, dénommées A et B, sont présentées de manières aléatoires aux utilisateurs.

Il nous faut donc :

1. Enregistrer le comportement de l'utilisateur au clique (bouton, icone, liens).
2. Enregistrer le comportement de l'utilisateur à la navigation.
3. Enregistrer le comportement de l'utilisateur sur la connexion et la déconnexion (savoir quand il se connecte, se déconnecte et s'il y a des erreurs).
4. Enregistrer les appels au backend.

Les quatre tables qui enregistreront les comportements de l'utilisateur sont (cf. annexe 7):

- Clicks
- Logs
- Navigation
- Backend

Je ne traiterai que du point 3. Les autres points étant quasi similaires en termes de code, il n'était pas judicieux de tous les expliquer en détails.

Préparation de la base de données :

Afin de préparer la base de données, il a fallu installer « **Docker Desktop Installer.exe** ». J'ai donc effectué une recherche sur le web afin de trouver le lien et les instructions d'installation. Les mots clé de ma recherche sur Google étaient « installation docker ». J'ai suivi le lien officiel de docker docs pour une installation sur un système d'exploitation Windows (<https://docs.docker.com/desktop/install/windows-install/>). La documentation étant en anglais, j'ai dû la traduire de l'anglais au français.

Voici le résultat en anglais :

Install Docker Desktop on Windows

Install interactively

1. Double-click **Docker Desktop Installer.exe** to run the installer.

If you haven't already downloaded the installer ([Docker Desktop Installer.exe](#)), you can get it from [Docker Hub](#). It typically downloads to your [Downloads](#) folder, or you can run it from the recent downloads bar at the bottom of your web browser.

2. When prompted, ensure the **Use WSL 2 instead of Hyper-V** option on the Configuration page is selected or not depending on your choice of backend.

If your system only supports one of the two options, you will not be able to select which backend to use.

3. Follow the instructions on the installation wizard to authorize the installer and proceed with the install.
4. When the installation is successful, click **Close** to complete the installation process.
5. If your admin account is different to your user account, you must add the user to the **docker-users** group. Run **Computer Management** as an **administrator** and navigate to **Local Users and Groups > Groups > docker-users**. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

Traduction en français

Installer Docker Desktop sur Windows

Installation interactive

Double-cliquez sur **Docker Desktop Installer.exe** pour exécuter le programme d'installation.

Si vous n'avez pas encore téléchargé le programme d'installation (Docker Desktop Installer.exe), vous pouvez l'obtenir à partir de Docker Hub. Il se télécharge généralement dans votre dossier **Téléchargements**, ou vous pouvez l'exécuter à partir de la barre des téléchargements récents au bas de votre navigateur Web.

Lorsque vous y êtes invité, assurez-vous que l'option Utiliser WSL 2 au lieu de Hyper-V sur la page de configuration est sélectionnée ou non en fonction de votre choix de backend.

Si votre système ne prend en charge qu'une seule des deux options, vous ne pourrez pas sélectionner le backend à utiliser.

Suivez les instructions de l'assistant d'installation pour autoriser le programme d'installation et procéder à l'installation.

Lorsque l'installation est réussie, cliquez sur **Fermer** pour terminer le processus d'installation.

Si votre compte *administrateur* est différent de votre compte *utilisateur*, vous devez ajouter l'utilisateur au groupe **docker-users**. Exécutez la **Gestion de l'ordinateur** en tant qu'**administrateur** et accédez à **Utilisateurs et groupes locaux > Groupes > docker-users**. Faites un clic droit pour ajouter **l'utilisateur au groupe**. *Déconnectez-vous et reconnectez-vous* pour que les modifications prennent effet.

Afin de déployer RethinkDB sur Docker, il suffit d'exécuter la commande suivante (cf. annexe 8):

```
$ docker run -d -P --name NomContainer rethinkdb
```

Pour sécuriser l'accès à la base de données il a fallu ajouter un mot de passe à l'utilisateur **admin**.

Voici la requête que j'ai utilisée dans l'interface web de RethinkDB :

```
r.db('rethinkdb').table('users').get('admin').update({ password: 'Atlas2022'})
```

Pour utiliser la connexion à la base de données sur toutes les pages, on utilise un **Hook**.

Le hook traite l'événement de chaque page et ajoute la connexion de la bd dans locals, à chaque événement (chaque requête), il est possible de retrouver dans l'événement l'objet locals contenant la connexion à la bd

Il faut également installer la librairie de RethinkDB dans mon projet :

```
npm install rethinkdb
```

C'est grâce à cette librairie que je pourrai exécuter les requêtes.

Le fichier **hooks.server.js** sert à mettre RethinkDB dans **{locals}**.

hooks.server.js:

```
import r from 'rethinkdb';

export const handle = async ({event, resolve}) => {
  let rethinkdb = null;
  try {
    rethinkdb = await r.connect({host: 'localhost', port: 49154, db: 'atlas',
    user: 'admin', password: 'Atlas2022'});

    event.locals.rethinkdb = rethinkdb;

  } catch (err) {
    console.log('RethinkDB is not reachable, tracking system disabled');
  }
  const response = await resolve(event);
  if (rethinkdb) {
    rethinkdb.close();
  }
  return response;
};
```

Pour l'instant, rethinkdb est déclaré mais n'a pas de valeur.

On crée un objet qui permet de se connecter à la base de données.

Ceci est le message d'erreur si la connexion ne se fait pas.

Enfin, on ferme la connexion et on obtient la réponse.

Objet de connexion récupéré
lors d'un console.log(event):

```
locals: {
  rethinkdb: TcpConnection {
    host: 'localhost',
    port: 49154,
    db: 'atlas',
    authKey: '',
    timeout: 20,
    ssl: false,
    outstandingCallbacks: {},
    nextToken: 1,
    open: true,
    closing: false,
    buffer: <Buffer >,
    _events: [Object: null prototype] {},
    _eventsCount: 0,
    _closePromise: null,
    rawSocket: [Socket]
  }
}
```

Développement de l'API

Envoyer les informations à l'API lors d'une connexion de l'utilisateur :

La **+page.svelte** contient un formulaire. Les informations de celui-ci sont envoyées vers la **+page.server.js** qui exporte l'action par défaut. Celle-ci contient une fonction qui traite les informations reçues par le formulaire. Cette fonction a pour rôle de créer un cookie, de permettre la connexion au site et de rediriger vers la page d'accueil. C'est dans cette fonction que le **fetch** est ajouté, ce dernier permet d'envoyer les logs de connexion à l'API pour le système de **tracking**.

La méthode POST sert à envoyer des informations à l'API sur le endpoint : **/track/logs** :

- On envoie le type d'action : login ou logout.
- On envoie la date de connexion qui renvoie à un timestamp.
- Le nom de l'utilisateur.
- Et si la connexion est 'OK' sinon cela renvoie un message d'erreur 'Wrong credentials'.

(outside)/login/+page.server.js :

```
fetch('/track/logs', {
  method : 'POST',
  headers : {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    type : 'Login',
    user : data.get('username'),
    time : Date.now(),
    status : credentials ? 'OK' : 'Wrong credentials'
  })
});
```

Objet regroupant les informations de connexion d'un utilisateur.

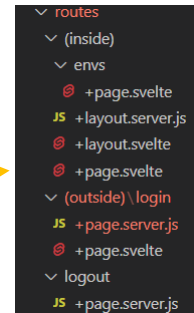
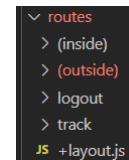
J'ai utilisé le même procédé sur le composant **logout** afin d'enregistrer la déconnexion de l'utilisateur.

Création de l'API :

Le routeur de SvelteKit :

Ce système de fichiers sont les routes de notre application (le chemin URL auxquels les utilisateurs peuvent accéder). C'est par le répertoire de notre base de code que ceux-ci sont définis. La racine du routeur est **src/route**. On ajoute à la cette racine le dossier sur lequel on désire faire arriver dynamiquement l'utilisateur.

Chaque répertoire **route** contient un ou plusieurs fichiers route, qui peuvent être identifiés par leur préfixe +.



Pour créer la **route/track/logs**, j'ai du créer un dossier **track** qui contient lui-même un dossier **logs** et c'est à cet emplacement que je crée mon fichier **+server.js**.

Le fichier **+server.js** exporte les fonctions correspondant aux méthodes http tels que GET, POST, PATCH, PUT and DELETE qui prennent pour argument un événement lié à la requête (**RequestEvent**) et qui retournent une réponse à la requête (**objet Response**) .

GET sert à envoyer les données, POST les reçoit et peut renvoyer une réponse, PUT reçoit les changements et les applique, DELETE supprime la donnée.

Track/logs/+server.js :

Récupération de rethinkdb depuis « locals ».

{locals} fait référence au « locals » du Hook.

```
export async function POST({locals, request}) {
  const {rethinkdb} = locals;
  let response = 'Database unavailable';
  let status = 503;

  if (rethinkdb) {
    await r
      .table('logs')
      .insert(await request.json())
      .run(rethinkdb, function (err, res) {
        if (err) throw err;
      });
    response = 'Log added to the database';
    status = 201;
  }

  return new Response(response, {status});
}
```

Message d'erreur si rethinkdb est introuvable.

Code erreur qui indique que la base de données est indisponible.

Cette requête sert à ajouter les informations dans la table « logs ».

La requête a été créée avec succès.

On envoie la réponse et le statut.

Méthode **GET** de track/logs/+server.js :

```
import r from 'rethinkdb';

export async function GET({locals}) {
  const {rethinkdb} = locals;
  let response = 'Database unavailable';
  let status = 503;

  if (rethinkdb) {
    await r.table('logs').run(rethinkdb, function (err, cursor) {
      if (err) throw err;
      cursor.toArray(function (err, result) {
        if (err) throw err;
        response = JSON.stringify(result, null, 2);
        status = 200;
      });
    });
  }

  return new Response(response, {status});
}
```

On récupère les logs dans la table en question et on les affiche.

On affiche les informations récupérées en JSON.

Code indiquant que la requête est traitée avec succès (code OK).

Réponse sur le navigateur en JSON :

```
[
  {
    "id": "cddf1d1c-b4a6-47b2-95d7-65de9d051b66",
    "status": "Wrong credentials",
    "time": 1669031079921,
    "type": "Login",
    "user": "Atlas"
  },
  {
    "id": "320b4854-e1db-4340-9144-cd25ccf18b6c",
    "time": 1669031066426,
    "type": "Logout",
    "user": "Atlas"
  },
  {
    "id": "4b84fe94-e9ed-41af-bdab-5904ffa03111",
    "status": "OK",
    "time": 1669031090880,
    "type": "Login",
    "user": "Atlas"
  }
]
```

Les injections NoSQL, bien que souvent méconnues sont, au même titre que les injections SQL, particulièrement dangereuses en raison de l'exposition des données.

Voici un exemple d'injection NoSQL :

Code de base :

```
JS test > ...
1 db.employes('users').find({
2   "user": req.query.user,
3   "password": req.query.password
4 });
```

Code malveillant :

```
JS test2 > ...
1 db.employes('users').find({
2   "user": "sabah",
3   "password": {"&ne": ""}
4 });
```

L'injection NoSQL « **&ne** » signifie « *non équivalent* ». Ce qui voudrait dire : Si le mot de passe n'est pas équivalent à rien (non nul) alors l'accès est autorisé.

Quelques pratiques à suivre pour sécuriser sa base de données :

- Vérifier les entrées utilisateur, en particulier le type, afin d'empêcher un utilisateur malveillant d'injecter des objets JSON au sein de requêtes.
- Tous les nœuds doivent être stockés derrière un pare-feu connecté à la base de données afin de sécuriser l'accès à vos informations sensibles.

Axe d'amélioration du projet en back

Les méthodes **DELETE** et **PUT/PATCH** seront à ajouter dans les versions futures d'**ATLAS**. Pour le moment, elles n'étaient pas nécessaires. Mais admettons qu'il faille effectuer une requête **PUT** sur une entrée précise, voici la syntaxe appropriée :

```
r.db('atlas').table('logs').filter(r.row('time').eq(1669031066426)).update({
  "id": "320b4854-e1db-4340-9144-cd25ccf18b6c",
  "time": 1669031066426,
  "type": "login",
  "user": "Atlas"
});
```

Dans un premier temps, on cible une entrée spécifique dans la table « log » en filtrant par sa date de création.

Ensuite, la méthode **PUT** permet de mettre à jour les données. Cette méthode implique de renvoyer l'entité entière sur laquelle une ou plusieurs modifications ont pu être effectuées à l'inverse de la méthode **PATCH** qui nous permet de ne renvoyer que les valeurs modifiées.

Concernant le **DELETE**, cette méthode permet de supprimer une donnée. Pour ce faire, il faudrait utiliser la syntaxe suivante :

```
r.db('atlas').table('logs').filter(r.row('time').eq(1669031066426)).delete();
```

Les variables d'environnement

Les variables d'environnement permettent de stocker les données sensibles qu'il ne faut pas afficher en dur dans le code.

```
• .env
1 DB_USER="admin"
2 DB_PASSWORD="Atlas2022"
```

La création d'un fichier **.env** à la racine du projet permet de protéger ces données qui peuvent être des identifiants de connexion à la base de

données, une clé API, etc.

La création d'un tel fichier permet notamment de ne pas envoyer ces données lors d'un push sur Gitlab. Il faut néanmoins vérifier que le fichier **.env** est bien présent dans le **.gitignore**.

```
• .gitignore
1 .DS_Store
2 node_modules
3 /build
4 /.svelte-kit
5 /package
6 .env
7 .env.*
8 !.env.example
9 .vite
10 dist
11 .eslintcache
12
```

Code dans le **hook.server.js** :

```
import { DB_USER, DB_PASSWORD } from '$env/static/private';

rethinkdb = await r.connect({host: 'localhost', port: 49154, db: 'atlas', user: DB_USER, password: DB_PASSWORD});
```

Outils de communications en entreprise et autres

Lors de mon intégration chez Cleyrop, j'ai dû me familiariser les outils mis en place. J'avais mes habitudes avec certains d'entre eux tandis que d'autres m'étaient inconnus.

Puisque la start-up prône le « *full-remote* », il lui est donc indispensable de fournir les outils qui facilitent la communication de tout à chacun.

En ce qui concerne la communication avec les employés, nous utilisons **Slack**, des channels dédiés aux différents besoins sont créés (cf. annexe 9). Nous utilisons également cet outil pour y rédiger notre « Daily » hebdomadaire.

Discord est exclusivement utilisé chez Cleyrop pour les partages d'écrans et salons mis à disposition afin de faciliter le travail collaboratif. Aucune ligne de code n'est permise sur la plateforme pour des raisons de sécurité.

Nous faisons nos « daily meeting » sur **Microsoft Teams**.

Factorial est à la fois un gestionnaire de congés et calendrier d'équipes. Il permet de rappeler les événements importants du mois en cours notamment.

Okarito est une solution de gestion de déplacements. Je suis donc passée par cette plateforme pour réserver mes billets de train et d'hôtel durant les « Caravan Days ».



Les « **Caravan Days** » sont une institution chez la jeune start-up. Il s'agit de réunir tous les membres de Cleyrop durant deux jours. Cela facilite la cohésion de groupe, permet de rencontrer les nouveaux venus et d'aborder des sujets plus sensibles tels que l'intégration des femmes dans les métiers de la Tech. Des ateliers sont mis en place.

Conclusion et remerciements

Cette reconversion professionnelle s'est imposée à moi très rapidement, le projet a donc mûri très vite. Jamais je n'aurais cru que le domaine du développement web était ouvert aux femmes. Je suis heureuse d'avoir pris ce chemin même si les doutes se sont installés de nombreuses fois durant cette formation.

La charge de travail fut lourde, le temps précieux et l'engagement important.

Tout au long de cette formation j'ai pu découvrir le monde du web, accroître mes connaissances à ce sujet. J'ai connu la frustration des erreurs dans la console, les longs moments de solitude à rechercher un point-virgule dans mon fichier CSS.

Au fil du temps et notamment durant mon stage, j'ai appris de nouvelles technologies et compris les choix qui pouvaient nous pousser vers l'une ou l'autre. J'ai découvert un autre aspect du travail en équipe, l'entraide, l'engagement et le plaisir partagé lors d'une victoire.

J'ai fait de belles rencontres qui ont su me rasséréner lors de mes périodes difficiles.

J'ai compris en partie la grandeur du code, l'envers du décor et de la multitude de possibilités encore existantes. Rien n'est figé et créer est toujours possible.

Mon stage m'a permis de comprendre les étapes de création de projet, les difficultés humaines, technologiques, financières et temporelles. L'importance de la collaboration, le bénéfice de s'écouter et de partager.

J'ai maintenant une vision globale d'une gestion de projet en AGILITE au sein d'une entreprise et c'est une chance pour la suite.

Je remercie tout d'abord mes enfants qui m'ont soutenue tout au long de cette période. Ils ont su croire en moi lorsque je doutais.

Je remercie Tidiane Sow pour ses conseils avisés et son soutien au quotidien.

Je remercie Jennifer Leroux et Yann Petit qui ont cru en moi et m'ont donné l'opportunité d'intégrer leur équipe au sein de Cleyrop. Yann a su élargir mes connaissances de manière considérable. Il a fait preuve de patience et de compréhension.

Je suis reconnaissante à l'équipe Cleyrop pour le soutien, les conseils et la bienveillance dont ils ont fait preuve durant mon stage. Je pense particulièrement à Ludovic Delahaye, Joan Bordeaux, Arnaud Muller et Franck Amiot.

Je remercie nos formateurs, Jérôme Lesaint et Guillaume Delacroix. Leurs bons conseils ainsi que la qualité de leur accompagnement étaient indispensables.

Je remercie également @Les Formés, camarades de ma promotion avec qui j'ai passé de très bons moments, nous nous sommes soutenus et cela restera une belle expérience.

Je tiens à remercier la Région Normandie qui a rendu cette reconversion professionnelle atteignable.

L'apprentissage reste la clé dans ce domaine qui demande d'être en veille permanente. La curiosité et l'investissement personnel restent une condition essentielle pour être une bonne développeuse. Les technologies évoluent rapidement et la concurrence entre les différents acteurs numériques reste importante.

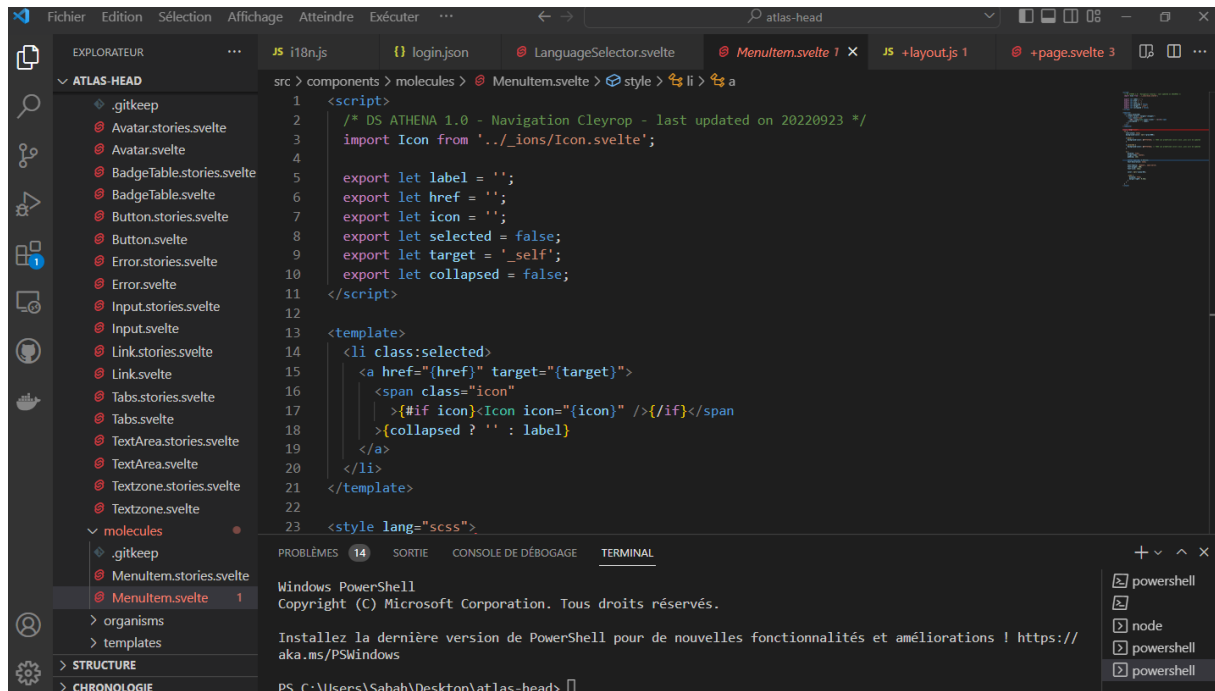
Ce parcours fut instructif, long, sinueux parfois mais m'a permis de découvrir un domaine qui m'était inconnu jusqu'alors et que j'affectionne désormais. C'était le but de cette reconversion, exercer un métier que j'aime. Il est une citation que j'affectionne et qui m'a guidée dans cette démarche :

« Choisis un métier que tu aimes et tu n'auras pas à travailler un seul jour de ta vie. »

Confucius.

Annexes

VISUAL STUDIO CODE (annexe 1) :

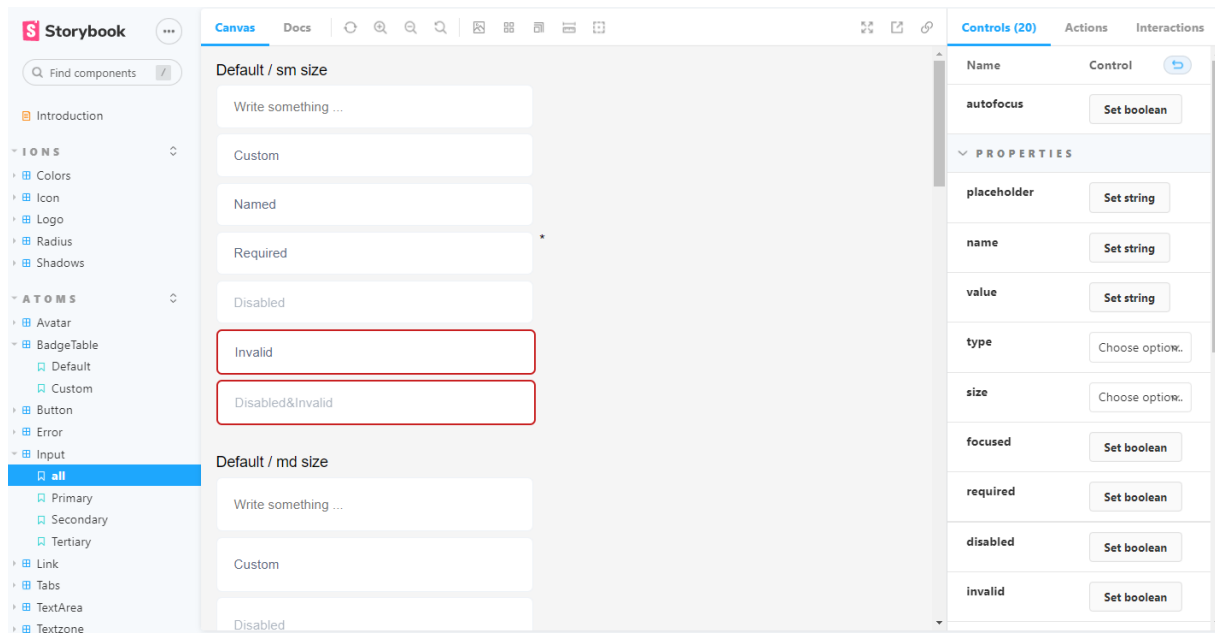


GIT GRAPH (annexe 2) :

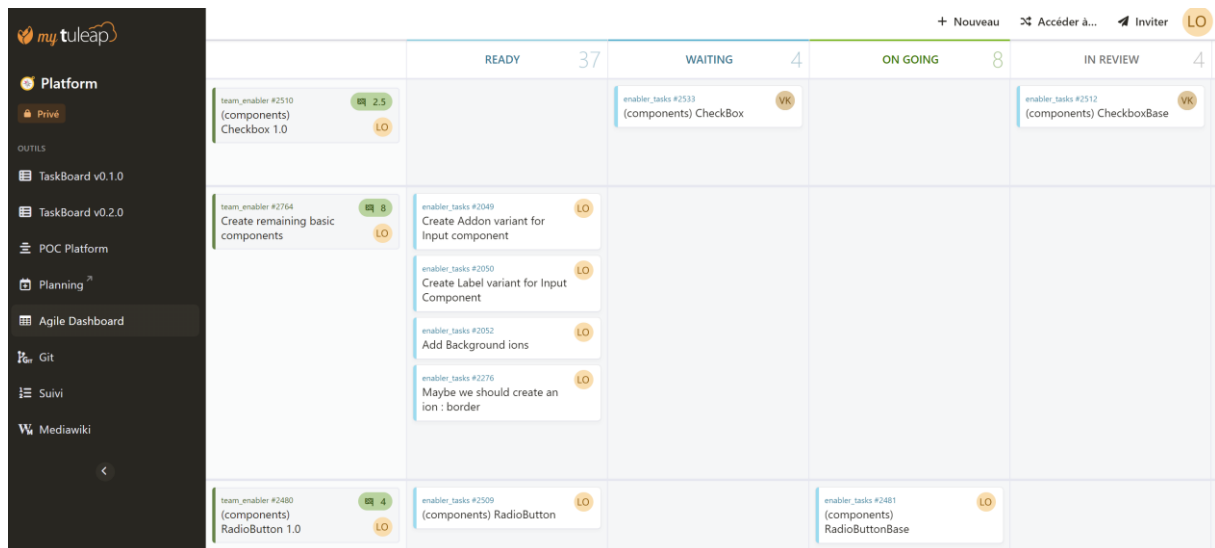
Branches: Show All

Show Remote Branches

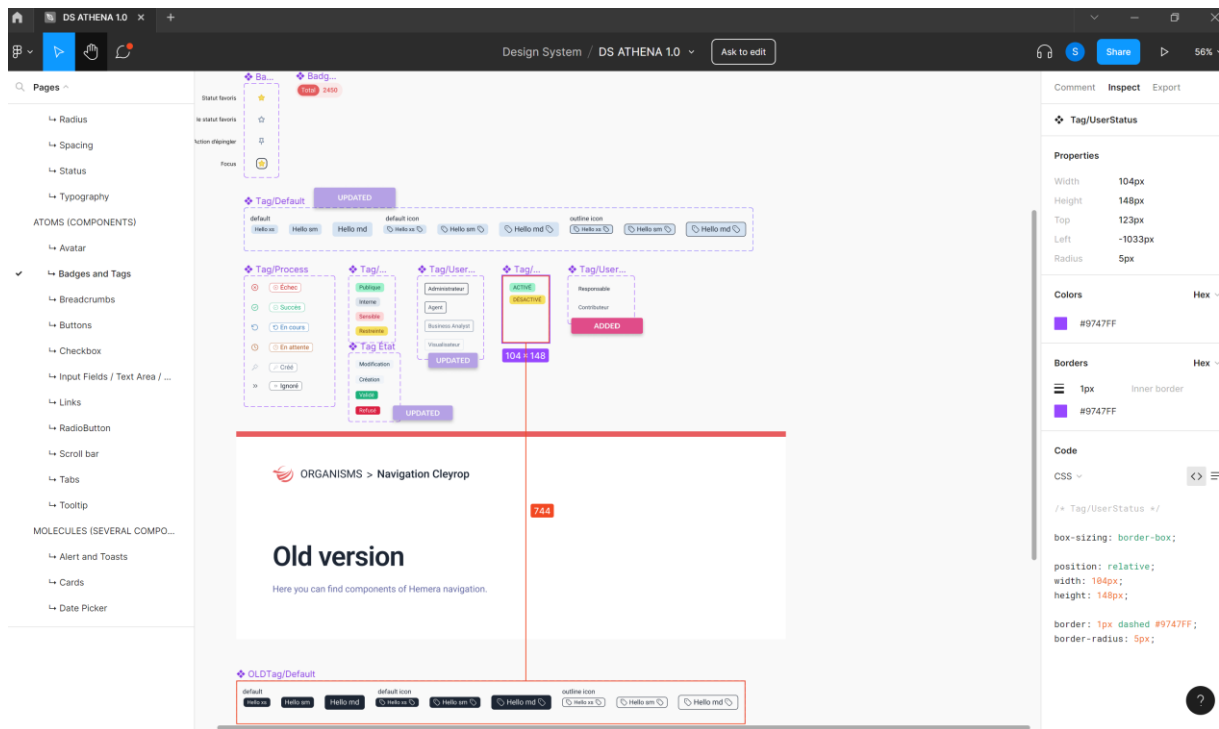
STORYBOOK (annexe 3) :



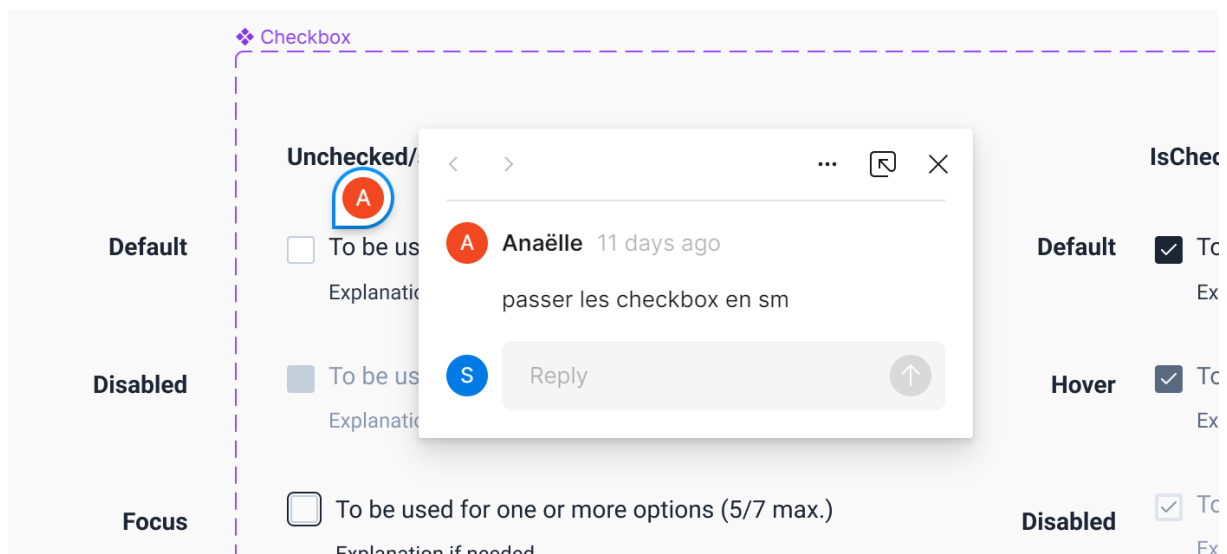
TULEAP (annexe 4) :



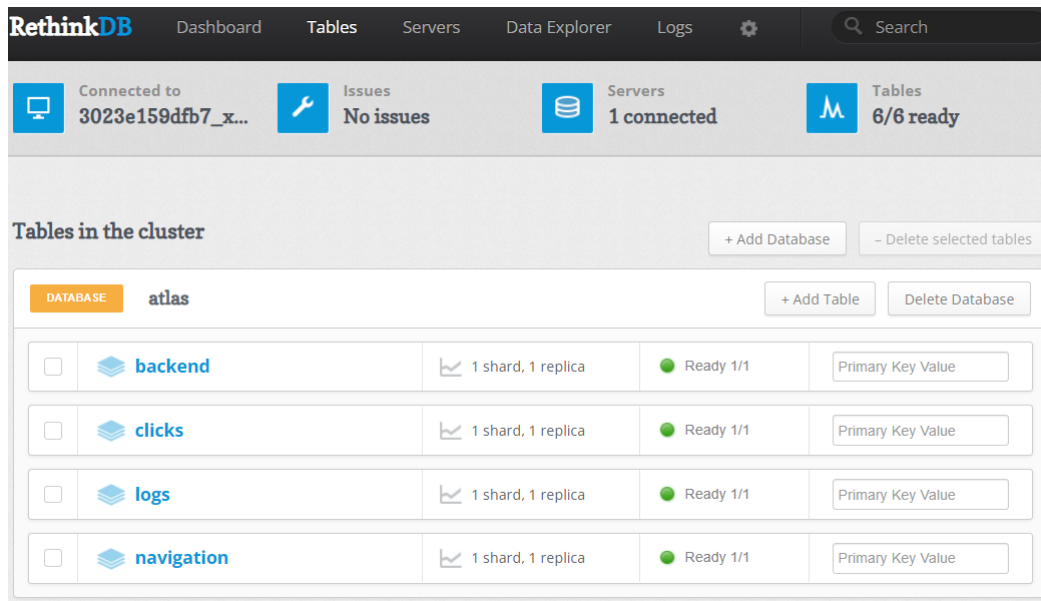
FIGMA (annexe 5) :



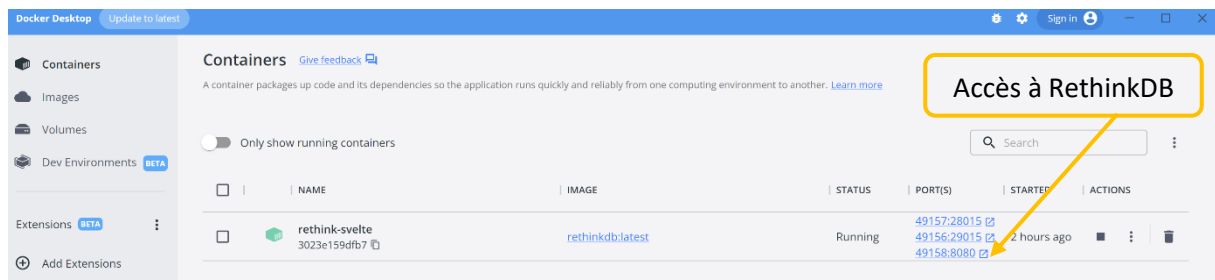
Commentaires dans FIGMA (annexe 6) :



Interface de **RethinkDB**, après la création de la base de données dans un conteneur (annexe 7) :



Capture de l'interface **DOCKER** une fois **RethinkDB** installée (annexe 8) :



SLACK (annexe 9) :

