AIRNEIS

ASI 3-24 DEV A

Équipe chargée du développement du projet :

HAMEL Julien
Zweird-958 — julienhamel.h@gmail.com

CARIOU Léon
theghost013 — cariouleon@gmail.com

GOMES DE SOUSA Vitor
rd-xx — vitor.gdsousa28@gmail.com

GOMES PEIXOTO Bruno
SeRiice — bruno.gomespeixoto1@gmail.com

Document rédigé le 28 juin 2024.

Accès au code

Pour pouvoir accéder au code, il suffit de se rendre sur ce lien GitHub : https://github.com/Zweird-958/airneis.
Vous y trouverez les informations nécessaires à la mise en place de celui-ci dans le fichier « README.md ». Toute autre documentation sera dans le dossier « docs », dossier qu'on parlera un peu plus en détail dans la partie architecture.

Table des matières

Introduction

À la suite de la demande du client nous avons réalisé un site de ventes de meubles. Dans cette section, nous vous expliquerons comment nous avons procédé le développement de ce projet.

Qu’est-ce que ça implique et comment avons-nous procédé ?

Avant de commencer à développer nous avons mis en place un système de tickets avec différentes tâches à faire comme des user stories, bugs et tâches techniques ce qui permet de pouvoir mieux prioriser et assigner les tâches afin de garantir une organisation structurée au sein de l’équipe, mais cela permet aussi d’avoir un historique des tâches ce qui permettrait à une autre équipe de reprendre le développement très facilement.

Il faut savoir que chaque bug ou amélioration à réaliser sur certaines fonctionnalités est répertorié dans la catégorie « Issues » de notre projet. De plus, nous avons développé en priorité l’application web au lieu de mobile, car selon notre équipe, l’application web peut obtenir plus d’utilisateurs que l’application mobile.

Pour commencer, nous avons créé la base du projet dans un « monorepo » (tous les éléments du site et du mobile ne sont pas séparés, mais au même endroit) car comme demandé nous avons réalisé une application web responsive, avec un back office, ainsi qu’une application mobile.

Une de nos priorités au début du projet était de mettre en place le système de traduction au plus vite afin de faire la traduction au fur et à mesure qu’on ajoutait les fonctionnalités pour nous éviter une grosse charge de travail une fois le projet terminé.

Puis nous avons mis un système de CI/CD en place afin de garantir que tout le code soit vérifié avant une mise en production. La CI/CD, ou Intégration Continue et Déploiement Continu, est un processus automatisé qui nous permet de tester et de valider régulièrement les modifications de code apportées par les développeurs. Grâce à ce système, chaque modification est automatiquement testée pour s'assurer qu'elle n'introduit pas de nouveaux problèmes, et une fois validée, elle peut être déployée de manière sécurisée et rapide en production.

Ensuite, nous avons commencé par développer les différentes fonctionnalités. Nous avons d'abord travaillé sur ce que le client peut voir, comme l'affichage des produits, puis nous avons développé les parties administratives. Cela nous a permis d'anticiper plus efficacement toutes les problématiques éventuelles et les besoins. Et nous avons fini par le système de paiement une fois que toutes la partie acheteur avais été fini.

Technologies

Nous énumérerons ici toutes les technologies utilisées et donnerons également des explications sur celle-ci.

Cette catégorie sera divisée en trois parties : une partie front-end, une back-end et pour finir une partie mobile. Chaque sous-catégorie contiendra les technologies relatives à son domaine pour une meilleur structure du document. Il y a néanmoins une exception à cette règle.

Le langage de programmation est commun aux deux sous-catégories. Pendant cette année scolaire, notre groupe a principalement étudié le JavaScript. De nombreux projets ont été réalisé avec ce langage et naturellement, c'est celui avec lequel nous sommes le plus confortable. Il nous a semblé donc logique de le choisir pour ce projet. Cependant, au vu de la nature du langage qu'il ne soit pas typé, ce n'est pas forcément un choix idéal pour le travail en équipe.

Que sont les types au juste ?

Ce sont des annotations qui définissent la nature des données d'une variable. Ce que ça signifie concrètement c'est que, premièrement, dans n'importe quel langage de programmation, on retrouve des entiers, des chaînes de caractère, etc. En JavaScript ce n'est pas différent, en revanche, il n'y a aucune vérification au moment de la compilation pour s'assurer que ces variables contiennent les données attendues, et cela peut provoquer des erreurs difficiles à détecter.

C'est précisément ici que TypeScript entre en jeu. TypeScript est une surcouche de JavaScript qui introduit ce concept de types. On bénéficie donc d'une vérification de ces types lors de la compilation mais également lors du développement. C'est particulièrement avantageux dans le travail en équipe car chaque membre du groupe sait à quel type de donnée s'attendre. Cela réduit grandement les erreurs de puisque le code écrit est automatiquement plus robuste ainsi que maintenable.

Pour ces raisons-là, nous avons opté pour écrire tout notre projet en TypeScript. Tout le groupe était déjà familier avec TypeScript, donc, là à nouveau, cela nous a semblé un choix logique.

Front-end

Cette sous-catégorie comporte un nombre considérable de technologies, donc nous avons opté pour en mentionner que certaines, les plus importantes selon nous.

Le front-end concerne uniquement la partie visible du site web, tout ce que l'utilisateur verra ainsi que

Next.js

Next.js est un framework pour React, qui est lui-même un framework pour JavaScript. Le but de Next.js est de faciliter le développement d'applications web. Ce framework nous permet aussi d'écrire notre code back-end. Cette partie sera expliquée plus en détails dans la catégorie back-end.

Next.js inclus également beaucoup d'utilitaires pour optimiser nos applications web. Par exemple, il fournit un composant pour les images qui se chargera automatiquement d'optimiser l'image selon le client qui la demande. Concrètement, si le navigateur du client qui demande l'image supporte les images au format « webp », alors Next.js s'occupera de transmettre ladite image, et naturellement, si le navigateur du client ne supporte pas ce type d'images, alors Next.js transmettra une image au format classique.

TailwindCSS

TailwindCSS est un framework CSS très facile de prise en main qui nous permet de styliser nos applications très facilement. TailwindCSS est très fléxible et répond parfaitement à tous nos besoins de manière efficace, que ce soit en termes de palettes, gestion des formats d'écran mobile ou ordinateur.

Une fois à nouveau, tous les membres de l'équipe maîtrisaient déjà TailwindCSS donc il nous a paru logique de choisir cette technologie.

Radix UI

Radix UI est une librairie de composants headless pour React. Les composants headless sont des composants sans aucun style ou très brut. Le but de ces composants c'est qu'ils possèdent énormément de fonctionnalités, sont modulables et accessibles.

C'est la raison pour laquelle nous avons choisis Radix UI. La facilité d'utilisation nous a permis de développer notre application très rapidement tout en stylisant ces composants avec TailwindCSS.

Back-end

Comme mentionné dans la catégorie front-end, notre back-end est fait avec Next.js, ou plutôt « lié » à Next.js.

tRPC

« Liés » parce que concrètement, l'API est faite avec tRPC, qui ensuite fait la liaison avec le système d'API de Next.js.

tRPC est outil de création d'APIs qui brille de par ses capacités de typages. Grâce à lui, on peut définir toutes nos routes API avec très grande facilité, définir les entrées de chaque route et bien sûr les données de sorties de chaque route.

Une fois tout cela défini, tRPC propose un utilitaire pour interagir avec son API, qui une fois bien configuré, permet d'avoir accès à toutes les routes définies dans l'application, ainsi que pour chaque route, avoir accès à ses types d'entrée et ses types de sorties.

Cela augmente considérablement notre rapidité et donc productivité puisque nous utilisons TypeScript a son plein potentiel et donc on connait le type des données qui transitent.

tRPC offre aussi une intégration avec une librairie appelée zod qui permet de vérifier ses données lors de l'exécution de l'application, plutôt que lors du développement grâce à TypeScript.

Une fois ces deux-là couplés, on peut créer une API robuste et prédictible, rapidement et avec assurance.

Stripe

Stripe est un service de paiement en ligne. Il nous permet de gérer tout ce qui est paiements sur le site web.

Nous avons choisi Stripe parce qu'il propose une offre gratuite assez généreuse mais aussi parce que nous avions jugés que l'intégration de cette technologie au sein de notre stack déjà existante serait triviale. C'était le cas.

Leur dashboard est très simple et facile d'utilisation. Ils proposent aussi un mode de tests où on peut effectuer autant de faux paiements que l'on juge nécessaires, tout en fournissant de fausses combinaisons de cartes bancaires pour chaque cas. Exemple : une carte bancaire qui causera un paiement approuvé, une autre qui causera un paiement échoué pour fonds insuffisants ou bien une autre qui causera une erreur de banque tout simplement.

Cela nous a permis de construire un système de paiements robuste et préparé à toutes éventualités.

S3

Pour stocker toutes nos images, nous avons choisis Simple Storage Service (S3) par Amazon.

S3 est devenu un standard dans le stockage de fichiers en masse et énormément de services sont conçus pour lui. Cela fait de lui un choix logique selon nous.

Meilisearch

En matière de recherche, nous avions le choix entre Elasticsearch ou bien Meilisearch. Nous avons opté pour Meilisearch pour sa facilité d'utilisation principalement, mais aussi pour sa rapidité.

Meilisearch fournis officiellement un client Node.js pour gérer tout ce qui insertion de nouveaux items dans les recherches ainsi que la recherche bien évidemment.

Il est très configurable, notamment en ce qui concerne la tolérance aux erreurs ou bien quels paramètres exclure des recherches.

D'après nos tests, les recherches effectuées avec Meilisearch sont très pertinentes donc nous avons choisis de le garder.

Redis

Redis est conçu pour garder des données en mémoire, on appelle ça le cache. Redis est un standard dans ce milieu donc nous avons optés pour l'utiliser.

Sa facilité d'utilisation et sa rapidité ont été grandement appréciés par tout le groupe.

Mobile

Expo

Nous connaissons React, donc à nouveau, choisir React Native nous a semblé logique.

React Native, est, comme son nom l'indique, React mais pour les plateformes natives, comme (inclus mais n'est pas limité à) : iOS et Android. Le code React écrit sera transformé en code natif pour les respectives plateformes. Pour iOS, du code Swift sera généré et pour Android, du code Kotlin sera généré.

Cela nous permet de bénéficier des avantages des langages natif sans jamais apprendre un nouveau langage, tout en gardant nos connaissances de React.

Le problème étant que, React Native à lui seul est assez cru. Nous avons choisi de développer l'application mobile avec le framework Expo.

Expo nous permet, entre autres, d'accéder à des fonctionnalités des plateformes natives en une seule API unifiée, comme la caméra, les notifications, etc.

NativeWind

NativeWind est ce qui nous permet de styliser notre application mobile avec les classes CSS de TailwindCSS.

C'est une librairie dont on ne peut pas se passer puisqu'elle nous permet très facilement personnaliser nos composants mobiles. De plus, grâce à cette dernière le code de nos composants sera similaire que ce soit pour l'application web ou mobile.

Architecture

Afin de travailler de la meilleure des façons, nous avons opté pour un monorepo divisé en quatre parties.

En premier, nous avons le dossier « apps », celui-ci est tout simplement où se trouve l'application web ainsi que l'application mobile, ces deux derniers ne peuvent pas communiquer ensemble et sont donc indépendant.

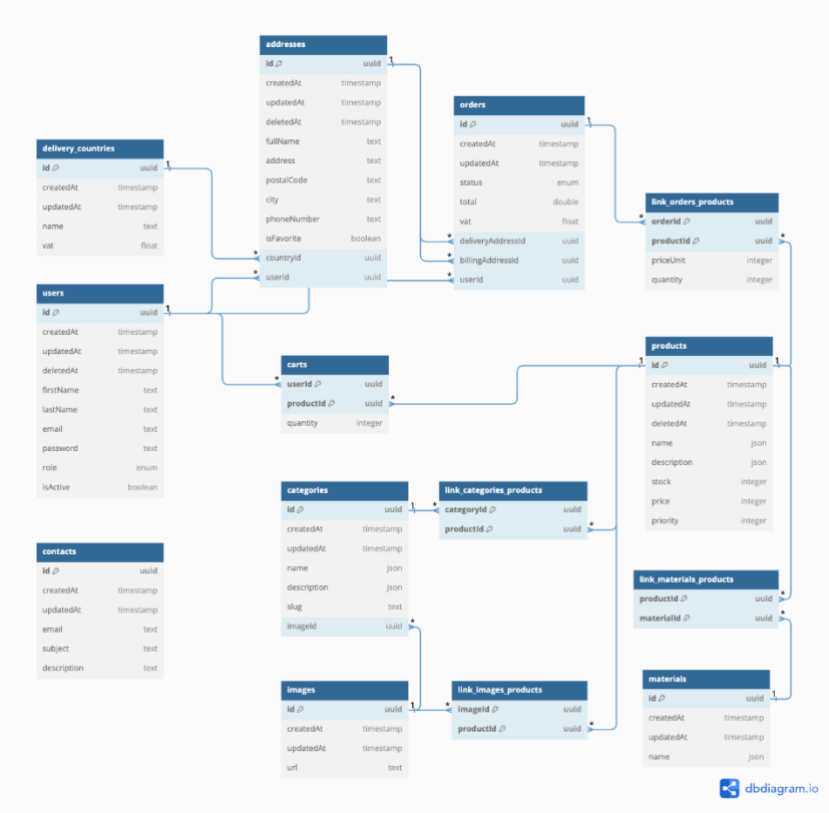
Ensuite vient le dossier « packages », ici nous retrouvons tous nos outils que nous pouvons utiliser que ce soit dans l'application web ou mobile. Ce dossier est extrêmement important, car il nous évite d'écrire le même code deux fois, en effet si nous avons créé par exemple un type pour notre application web, mais qu'il est nécessaire également pour le mobile nous pourrons tout de même l'utiliser, ce dossier offre donc un gain de temps et un code plus propre en évitant les doublons.

Pour continuer, nous avons le « tooling », il ne s'agit pas d'un dossier mais de nos fichiers de configurations, ces derniers se trouvent à la racine du projet directement. Nous avons donc une configuration globale, cependant, si dans une application ou un package nous avons la nécessité de modifié la configuration, ceci est totalement possible, ainsi ce dernier possèdera sa configuration en héritant de la globale.

Pour terminer, vient le dossier « docs », dans celui-ci vous retrouverez toute la documentation technique que ce soit toutes les routes API ou bien le modèle physique de données, qu'on reparlera plus tard. Grâce à ce dossier, le code pourra être repris par une autre équipe très facilement et sans se perdre dans le fonctionnement de ce projet.

Nous avons donc opté pour ce choix d'architecture pour avoir le plus de code réutilisable possible et également avoir une belle structure dans notre projet.

Modèle physique de données



Ci-dessus, vous trouverez le modèle physique de données (MPD) élaboré par notre équipe, il est également disponible à l'adresse suivante : https://github.com/Zweird-958/airneis/blob/main/docs/db/mpd.png

Pour répondre aux besoins et aux exigences fixés par Airneis, notre équipe a opté pour une approche relationnelle. Concrètement, nos tables sont interconnectées par différents types de relations : un à un, un à plusieurs, plusieurs à un, ou plusieurs à plusieurs. Définir correctement les relations entre chaque table est crucial puisque cela empêche la redondance des données et garantit leur intégrité. De plus, les modèles relationnels sont réputés pour offrir une gestion structurée, sécurisé et efficace des données, ce qui en fait une solution idéale dans notre cas.

En se référant à notre MPD, on distingue facilement deux types de tables.

En effet, le premier type de table peut être comparé à des entités, que l'on peut définir comme étant la représentation d'un objet dans lequel on va stocker des informations relatives à celui-ci. C'est le cas des tables : *users, addresses, orders, products, categories, carts, images, materials, contacts* et *delivery_countries*. À l'exception de *carts*, ces tables reposent sur une même structure de base incluant les *champs id, createdAt* et *updatedAt*, qui servent à identifier chaque ligne et de suivre l'évolution des données. Bien entendu, chaque table dispose ensuite de ses propres champs pour définir au mieux le sujet auquel elle se réfère.

Puis, il y a les tables de liaison, qui correspondent à notre deuxième type. Ce sont elles qui vont nous permettre de définir au mieux les relations plusieurs à plusieurs entre deux tables. Prenons par exemple la table *link_categories_products* : une catégorie peut contenir plusieurs produits, et inversement, un produit peut appartenir à plusieurs catégories. De ce fait, il est indispensable de créer une table intermédiaire pour connecter les tables *categories* et *products*. Sans cela, nous aurions eu plusieurs lignes similaires dans chacune des tables avec seulement un champ dont la valeur aurait changé.

Difficultés rencontrées

Dans cette partie, les différentes difficultés seront expliquées, nous expliquerons comment ces dernières ont été rencontrées et ainsi résolues.

Mobile

La partie développement mobile a rencontré énormément de problèmes. À plusieurs reprises il était impossible d'exécuter le code sur nos machines locales, pour ne pas aider, chaque erreur différenciait entre les membres de l'équipe. Si un membre corrigeait le problème de son côté cela ne garantissait pas que celui-ci ne serait plus présent pour les autres membres de l'équipe.

Après plusieurs investigations, nous avons constaté qu'il s'agissait d'une différence de versions entre les membres. Après avoir regardé tous ensemble, chaque membre a fait en sorte d'avoir les mêmes versions des outils utilisés dans le projet.

Internationalisation

D'après le souhait du client, nous avons mis en place l'internationalisation, ainsi tout le contenu du site sera traduit, que ce soit le front-end, back-end, le site côté admin ou client. Pour le moment, uniquement le français et l'anglais sont pris en charge.

Nous avions ainsi, développer un système de traduction par nos soins, de plus celui-ci s'accordait parfaitement avec TypeScript. Cependant, on devait penser au référencement, il était donc primordial que le texte soient traduit aux yeux des moteurs de recherche. Au cours du projet nous avons donc migrer de méthode pour gérer ces traductions, ce dernier est plus optimisé et maintenable, ce qui n'est pas un point à ignorer.

Les délais

A plusieurs reprises, les revues de code prenaient plus de temps que prévu, alors les délais étaient courts et presque non respectés, pour remédier à cela on a dû obligatoirement se dépêcher sur la fin de quelques sprints pour que la tâche soit validée à temps.

Ces retards nous ont permis d'être plus réactifs et réaliser les revues de code en temps et en heure sans la précipitation.

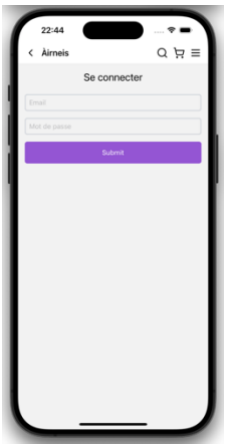
La mise en ligne d'images

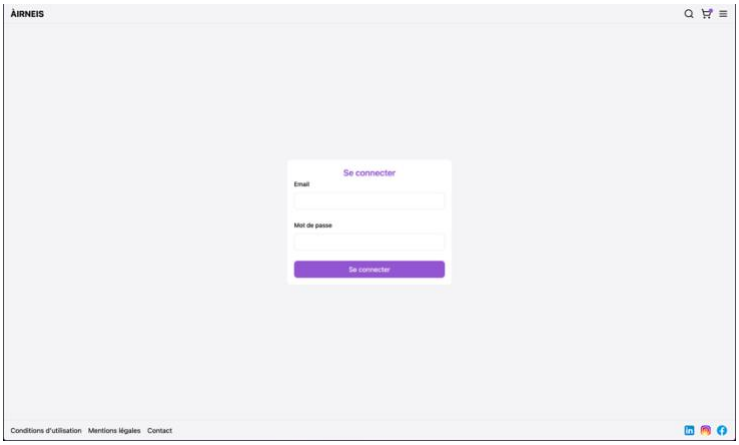
Mettre en ligne une image est toujours délicat, encore plus avec l'utilisation de tRPC. Après plusieurs recherches, nous avons opté par un système en deux temps, c'est-à-dire une qui utilisera tRPC et l'autre non, grâce à cela les images seront stockées sur S3 de façon optimale.

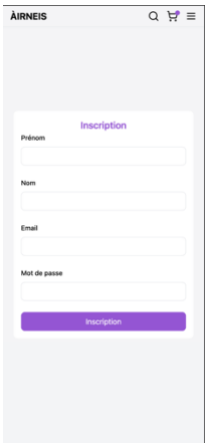
Les fonctionnalités

À travers cette section, vous pourrez observer les principales fonctionnalités développées avec pour chacune l'affichage ordinateur et mobile pour l'application web, ainsi que l'application mobile si la fonctionnalité est présente.

L'authentification



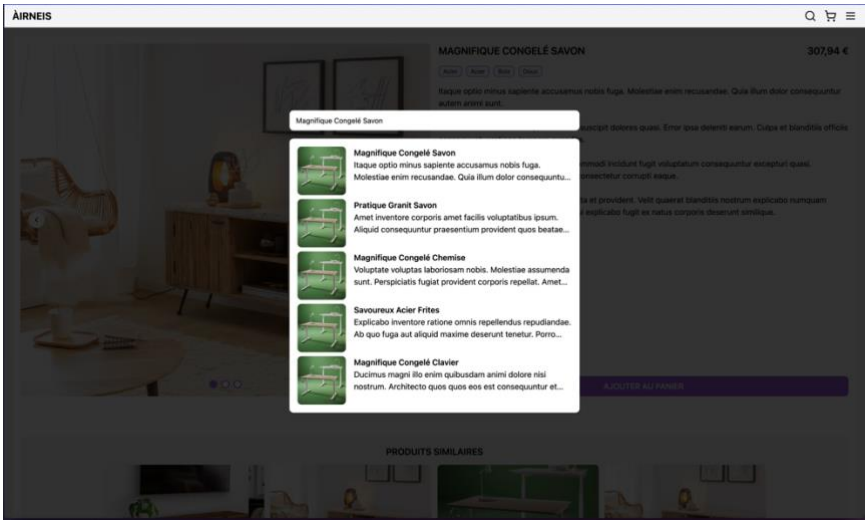




La fonctionnalité primordiale est bien entendu l'authentification, afin de garantir une sécurité aucun mot de passe de sauvegarder en claire dans la base de données, uniquement le hash de chacun est stocké, obtenu à l'aide de la fonction de hachage « bcrypt ». Si jamais une fuite de données se produit alors aucun mot de passe ne sera révélé.

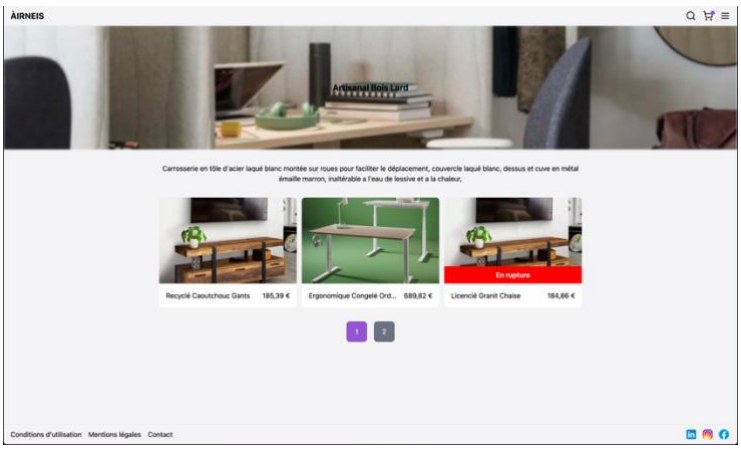
Chaque utilisateur possède un rôle, soit utilisateur soit administrateur, cela permet de gérer les permissions de l'accès au panel d'administration.

La recherche



Pour améliorer l'expérience utilisateur au niveau de la navigation nous avons intégré la recherche, grâce à cette dernière l'utilisateur peut trouver un produit en rentrant son nom, évidemment il n'a pas besoin d'écrire le nom du produit à la perfection. Notre recherche possède de la tolérance donc il peut se tromper sur l'écriture mais trouver tout de même son produit, de plus écrire les accents ne sont pas nécessaire, grâce à cette fonctionnalité, l'utilisateur pourra retrouver ce qu'il souhaite le plus rapidement possible.

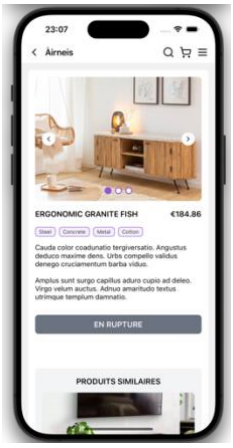
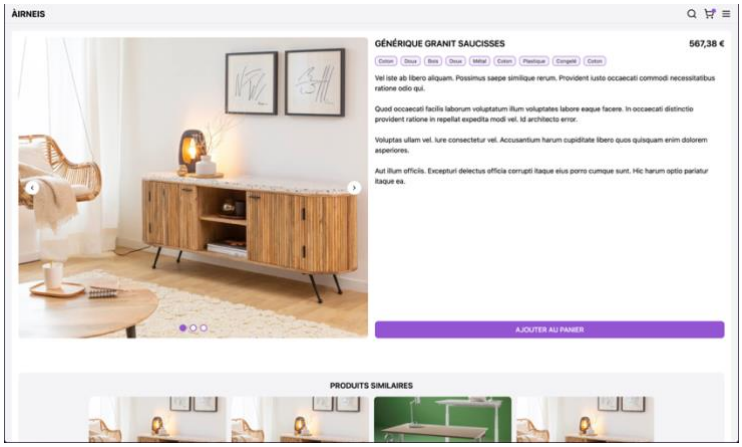
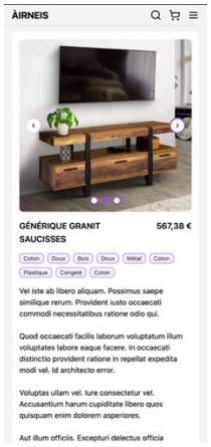
Les catégories

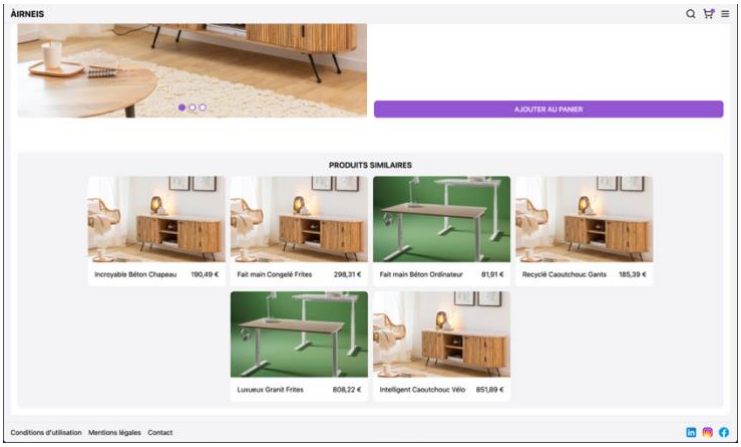

Afin d'accéder à un produit spécifique l'utilisateur soit doit passer par la recherche expliquée précédemment ou bien par la page des catégories. Chaque catégorie possède son image, son nom et sa description. Les produits appartenant à cette dernière sont ensuite listés, avec une limite de douze produits par page.

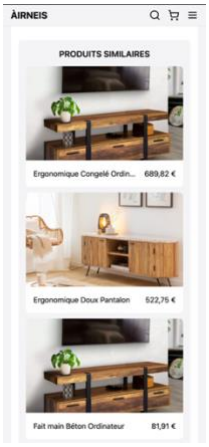
L'ordre d'affichage des produits est géré par un algorithme, en premier vous aurez les produits en stock trié par leurs priorités, en second les produits sans priorité et en dernier ceux en rupture de stock.

Les produits

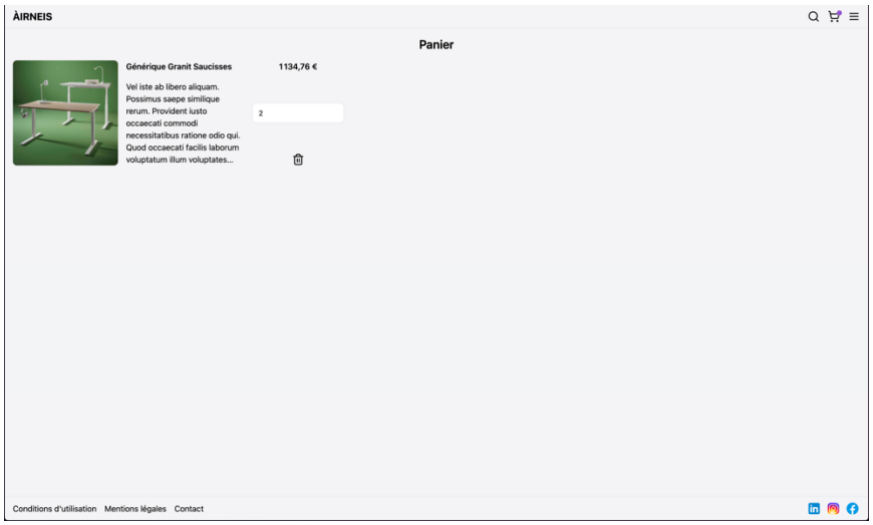
Chaque produit possède sa propre page, elle offre des informations supplémentaires que la page catégorie. Tout d'abord, pour ce qui est des images cette fois-ci nous avons un carrousel, ainsi l'utilisateur peut défiler parmi toute les images. Nous avons également la description et tous les matériaux de ce dernier. Pour finir, depuis cette page l'ajout au panier est disponible, si le produit est en rupture de stock, ce bouton enlevé est remplacé par un texte informatif.

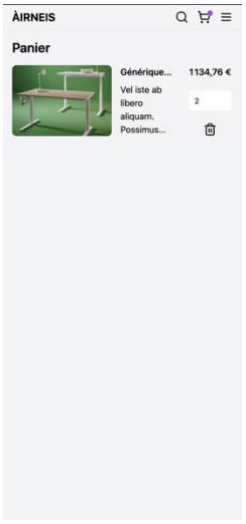




La fonctionnalité importante de cette page est « les produits similaires ». Nous listons ainsi six produits qui sont de la même catégorie, et en stock de préférences.

Le panier

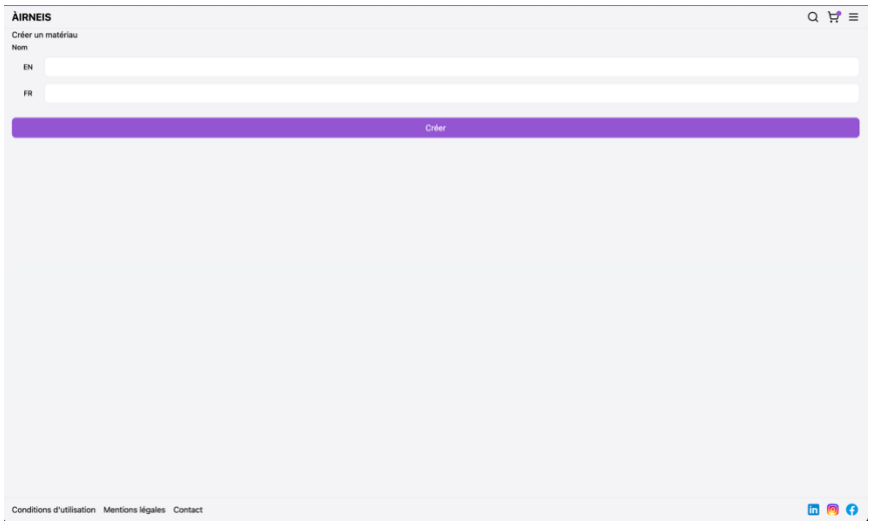


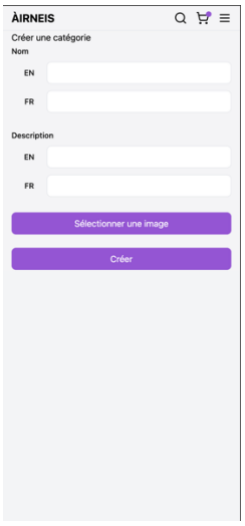


Évidemment que serait un site de vente sans panier, c'est pour cela que cette fonctionnalité était l'une des plus importantes selon notre équipe. Il faut noter que comme demandé, l'utilisateur peut posséder un panier qu'il soit connecté au nom. Le panier est sauvegardé dans notre bases de données si connecté sinon dans le navigateur de l'utilisateur. Lorsque l'utilisateur se connecte il ajoute les produits de son panier de son navigateur dans celui de la base de données.

L'utilisateur a également la possibilité de modifier la quantité de chaque produit ou bien de le supprimer de son panier actuel.

Administration





Pour chaque entité à créer, que ce soient les matériaux ou catégories, l'administrateur doit remplir les champs pour chaque langue que le site prend en charge. Pour certains cas, une image est nécessaire.

Il aura accès à chacune de ses pages depuis son interface d'administration.