Koekelnerd

42 Avenue des nerds
1081 Bruxelles
(+32) 123 45 67 89

ImmauWeb Project

3 Novembre 2025

Plateforme d’annonces immobilières de ventes et de locations de biens particuliers

L’objectif premier est de faciliter l’accès aux annonces de biens mises en ventes ou mises en locations, avec une recherche personnalisée des biens souhaités.

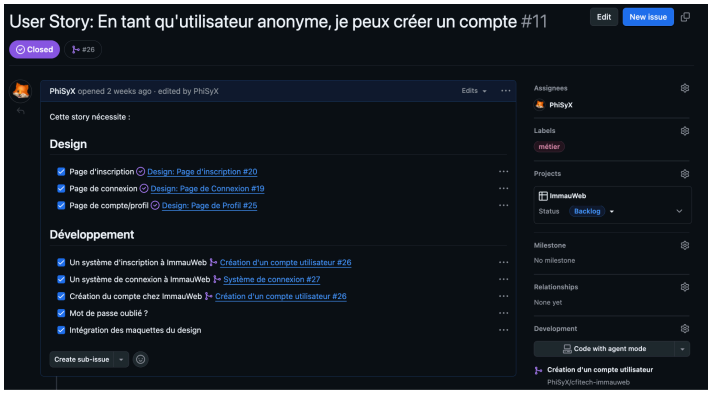
Conception

Le projet a été réalisé par mes propres soins et a été versionné dans un GitHub.

Gestion du projet

Le projet a été segmenté en tâches afin d’optimiser sa conception, après une longue réflexion sur les parcours des divers profils utilisateurs de la plateforme.

Chaque tâche a été développée dans une branche Git dédiée, et le suivi a été centralisé dans GitHub Projects sur GitHub.



2

Architecture et stack technique

- Base de données **MySQL**
- Backend **Symfony** avec PHP et une architecture MVC.

  L'architecture MVC est un modèle de conception logicielle qui sépare une application en trois parties interconnectées :

  1. Le Modèle (logique métier et données)
  2. La Vue (interface utilisateur)
  3. Le Contrôleur (traitement des entrées de l'utilisateur et interaction entre le modèle et la vue).

  Cette séparation permet d'avoir une bonne organisation, une bonne maintenabilité pour un projet non complexe ou novice.

  Pour ce backend, étant donné qu'il y a très peu de règles métiers, où l'accès aux données se fait avec **API Platform**, cette architecture me semble adéquate.
  Ce backend va principalement servir d'API afin d'alimenter l'application frontend.

- Frontend **Angular** avec TypeScript et une architecture N-tiers

  L'architecture N-tiers est une approche logicielle qui divise une application en plusieurs couches indépendantes, chacune ayant un rôle spécifique comme la présentation, la logique métier ou l'accès aux données. Cette séparation permet d'améliorer la modularité, la réutilisabilité et la capacité à faire évoluer l'application de manière indépendante.

  Pour ce projet, le frontend est la partie où le code est plus fournis et où il y a plus de logiques. Le choix de cette architecture m'a semblé adéquat.

Persona

L'utilisateur anonyme

Il s'agit de l'utilisateur non inscrit ou non connecté qui visite notre plateforme.

L'utilisateur client

Il s'agit de l'utilisateur inscrit sur notre plateforme, et qui est intéressé par un ou plusieurs biens.

Cet utilisateur obtient l'étiquette de `**Client**` **dès l'inscription** à notre plateforme. Cette étiquette de **Client** est représentée sous le nom de rôle `**ROLE_CLIENT**` en Symfony et dans la base de données.

L'utilisateur propriétaire

Il s'agit de l'utilisateur inscrit sur notre plateforme, et qui a publié des annonces de ventes ou de locations de ses biens.

Cet utilisateur obtient l'étiquette de `**Propriétaire**` **dès la publication** d'annonce sur notre plateforme. Cette étiquette de **Propriétaire** est représentée sous le nom de rôle `**ROLE_PROPRIETAIRE**` en Symfony et dans la base de données.

L'administrateur

Il s'agit de l'administrateur inscrit sur notre plateforme, et qui gère les clients et propriétaires ainsi que les messages de contact envoyés par ces différents persona's.

Cet administrateur obtient l'étiquette d'`**Admin**` **par le webmaster** autrement dit moi-même. Cette étiquette d'**Admin** est représentée sous le nom de rôle `**ROLE_ADMIN**` en Symfony et dans la base de données.

Sécurité d’accès aux données et aux pages

Certaines requêtes sont sécurisées via des **JWT** (**JSON Web Token**).
Lors de l’authentification, le serveur émet un jeton signé contenant l’identité, les rôles et une date d’expiration du jeton d'1h. Ce jeton est sauvegardé dans l’espace de stockage de session du navigateur.
Le frontend envoie ensuite dans l’en-tête `Authorization: Bearer JETON` sur les requêtes ayant besoin d’une sécurité. Le backend vérifie la signature et la validité du jeton avant d’autoriser l’accès aux ressources sécurisantes.

Chaque rôle mentionné dans la précédente section se voit recevoir des autorisations et des restrictions pour l’accès aux données, l’accès aux pages.

C’est-à-dire que l’utilisateur dont le rôle est **Client** NE PEUT PAS éditer l’annonce d’un utilisateur dont le rôle est **Propriétaire**.
C’est-à-dire que l’utilisateur dont le rôle est **Propriétaire** NE PEUT éditer que SES propres annonces, et pas celles des autres utilisateurs ayant le rôle **Propriétaire**.

Les administrateurs ont quant à eux une liberté totale sur ces restrictions, pour des questions de bon sens.

Ces contrôles sont mis en place côté backend et reflétés dans l’interface.