Création du site de l’association les chats libres de La Rochette

Projet professionnel

Elisa GYURICZA

Formation développeur web 2021

# 

# Remerciements

Je tiens à remercier Béatrice Vottero, présidente de l’association les Chats Libres de la Rochette de m’avoir permis de travailler sur ce projet et de m’avoir fait confiance afin de répondre au mieux aux attentes et besoins de l’association. J’ai eu la chance de travailler avec une équipe de bénévole très sympathique et forte de proposition pour la création du site internet de l’association.

Je souhaite remercier l’école de l’Adrar qui a cru en moi et sans qui je n’aurai pas eu l’impulsion pédagogique pour m’accompagner dans mon projet de reconversion professionnelle et qui m’a ainsi donné ma chance. L’équipe pédagogique qui la compose m’a permis d’acquérir de nombreuses compétences dans les domaines du développement et de la conception. Je souhaite remercier chacun de mes enseignants pour le temps qu’ils m’ont accordé dans ce processus d’apprentissage très riche, pour leurs accompagnements, bienveillances et disponibilités.

Je tiens à remercier tout particulièrement mes camarades de promotion, avec lesquels nous nous sommes motivés au quotidien, nous avons partagés nos connaissances et compétences respectifs. Un remerciement tout particulier à mon camarade Jérôme qui nous a partagé son savoir, accompagné et supporté tout au long de la formation.

Je tiens à remercie pour finir, ma famille et mon compagnon qui m’ont accompagné et soutenu tout au long de ma formation.

# Abstract

I did my internship in the association « Les Chats Libres de la Rochette ». I decided to work for them because I have been a volunteer for this association for over a year. We are taking care of street cats, feeding and finding them a new family.

I offered my services because I knew they had the project to develop a website for the purpose of sharing their activities and bringing to the fore all cats who are looking for a forever home. For the moment they’re visible on social media which is great but not sufficient.

I was aware of the requirements because we had a few appointments to discuss on this project. The project looks like a blog. There are 2 parts, one for visitors and another one for an administrator where volunteers will be able to add articles and share information.

As I started a web developer training, I decided to create the website using the tools I had learned at school such as, HTML, CSS, JS for the front-end part and PHP, laravel framework for the back-end.

The project isn’t finished yet but I hope it will in September. I dealt with some difficulties, first of all I worked alone so I had to manage the organization and scheduling of different tasks on my own. Then I’m a junior developer so I spend a lot of time learning how to use languages especially back-end ones.

I’m quite proud of myself until now because a few functionalities are working okay, and it’s motivating me to move on.

After passing my IT degree I would like to continue my studies and find a part-time job to gain practice and experience.

[Remerciements 3](#_Toc74239230)

[Abstract 4](#_Toc74239231)

[I. L’association 7](#_Toc74239232)

[1. Présentation de l’association 7](#_Toc74239233)

[2. Contexte et besoins 7](#_Toc74239234)

[3. Présentation du projet 8](#_Toc74239235)

[a. Partie administration : 8](#_Toc74239236)

[b. Partie visiteur : 8](#_Toc74239237)

[4. Compétences couvertes par le projet 9](#_Toc74239238)

[II. Cahier des charges 10](#_Toc74239239)

[1. Expression des besoins du client 10](#_Toc74239240)

[a. Les fonctionnalités indispensables 10](#_Toc74239241)

[b. Les fonctionnalités informatives 10](#_Toc74239242)

[c. Les fonctionnalités additionnelles 10](#_Toc74239243)

[2. Charte graphique 10](#_Toc74239244)

[3. Contraintes Techniques 12](#_Toc74239245)

[III. Spécifications fonctionnelles 13](#_Toc74239246)

[1. Diagramme des cas d’utilisations 13](#_Toc74239247)

[2. Premier cas : Authentification du rédacteur web 15](#_Toc74239248)

[a. Diagramme d’activité 15](#_Toc74239249)

[b. Diagramme de séquence 16](#_Toc74239250)

[c. Maquettage 18](#_Toc74239251)

[3. Deuxième cas : Créer un chat « famille d’accueil » 21](#_Toc74239252)

[a. Diagramme d’activité 21](#_Toc74239253)

[b. Diagramme de séquence 22](#_Toc74239254)

[c. Maquettage 24](#_Toc74239255)

[4. Troisième cas : Afficher une liste de chat par catégorie. 26](#_Toc74239256)

[a. Diagramme d’activité 26](#_Toc74239257)

[b. Diagramme de séquence 26](#_Toc74239258)

[c. Maquettes 26](#_Toc74239259)

[5. Diagramme de classe 27](#_Toc74239260)

[IV. Spécifications techniques 28](#_Toc74239261)

[1. Base de données 28](#_Toc74239262)

[a. Dictionnaire de données 28](#_Toc74239263)

[b. MCD 31](#_Toc74239264)

[c. MLD 33](#_Toc74239265)

[2. Choix Techniques 35](#_Toc74239266)

[a. Environnement de travail et langages 35](#_Toc74239267)

[V. Arborescence 36](#_Toc74239268)

[1. Arborescence utilisateur 36](#_Toc74239269)

[2. Arborescence administrateur : 36](#_Toc74239270)

[VI. Implémentation 37](#_Toc74239271)

[1. Création du projet sous Laravel 8 37](#_Toc74239272)

[a. Présentation de Laravel et de son ORM Eloquent 37](#_Toc74239273)

[1. Framework 37](#_Toc74239274)

[2. Composer 37](#_Toc74239275)

[3. MVC 37](#_Toc74239276)

[4. Architecture Laravel 38](#_Toc74239277)

[5. Eloquent 39](#_Toc74239278)

[6. Template blade.php 39](#_Toc74239279)

[a. Création du projet sous Laravel 8 40](#_Toc74239280)

[b. Création d’une table avec Laravel 8 41](#_Toc74239281)

[1. Premier cas : Authentification 43](#_Toc74239282)

[1. Routeur (web.php) 43](#_Toc74239283)

[2. Modèle (users.php) 44](#_Toc74239284)

[3. Contrôleur (loginController.php) 45](#_Toc74239285)

[4. Vue (connexion-page.blade.php) 47](#_Toc74239286)

[5. Front-end 47](#_Toc74239287)

[2. Deuxième cas : Créer chat famille d’accueil. 50](#_Toc74239288)

[1.Routeur (web.php) 50](#_Toc74239289)

[2. Model (cat.php) 50](#_Toc74239290)

[3. Controller (CatController.php) 50](#_Toc74239291)

[4. Vues (create-articles-cat-foster.blade.php) 52](#_Toc74239292)

[5. Front-end 52](#_Toc74239293)

[6. Démonstration visuelle 52](#_Toc74239294)

[VII. Conclusion du projet 58](#_Toc74239295)

[VIII. Bilan 58](#_Toc74239296)

[IX. Annexes 58](#_Toc74239297)

[1. Maquettes desktop et mobile. 59](#_Toc74239298)

[2. Mobile 62](#_Toc74239299)

# L’association

## Présentation de l’association

L’association les chats libres de la Rochette a été fondé en 2013 par Béatrice Vottero dans le but de prendre en charge les chats de la rue.

Ses principales missions sont de soigner, nourrir, sociabiliser et ensuite de faire adopter les chats et chatons lorsque cela est possible. L’association est composée d’une dizaine de bénévoles qui tous les jours, dans leurs temps libres, travaillent sur différentes missions pour faire vivre cette association et prendre en charge un maximum de chats.

La prise en charge des chats ne se limitent pas aux trappages. Il y a en arrière-plan toute une gestion très complexe, de nombreuses démarches administratives, de l’organisation entre les différents acteurs : bénévoles, familles d’accueils, familles adoptantes, vétérinaires et toutes les personnes qui nous contactent pour avertir l’association de la présence de chats errants dans leurs communes.

Jusqu’à présent l’association utilise uniquement les réseaux sociaux pour communiquer avec les personnes externes à l’association qui sont sensibles à cette cause, pour rechercher des familles d’accueils ou faire des demandes de dons par exemple.

Ils utilisent comme moyens de communication interne des fichiers partagés sur Google Drive ainsi que de nombreux groupes de discussions WhatsApp.

## Contexte et besoins

L’association prend de l’envergure et a du mal à optimiser la gestion des prises de contacts avec les personnes qui souhaitent participer à la vie associative. L’utilisation des réseaux sociaux comme point de contact avec les personnes atteint aujourd’hui ses limites.

Malgré un travail quotidien sur l’organisation de l’association dans la transmission des informations entre les différents acteurs, Il y a un constat d’une perte de temps dans l’automatisation des prises de contacts par exemple au niveau des personnes qui souhaitent devenir famille d’accueil ou famille adoptante.

Le besoin majeur serait de pouvoir centraliser sur une plateforme les propositions des visiteurs en créant des modèles de formulaire uniques pour chaque prise de contact et de pouvoir conserver une trace écrite de chacun de ses contacts afin que tous les bénévoles puissent suivre leurs évolutions.

Les réseaux sociaux seraient ensuite utilisés comme plateformes de relais qui permettraient de promouvoir les différents articles et informations mis à jour quotidiennement sur le site internet ou pour les communications urgentes.

Plusieurs communes subventionnent l’association et des magasins spécialisées dans les animaux offrent régulièrement des dons. Le site internet serait une façon de pouvoir mettre en avant les différents acteurs qui accompagnent l’association afin de rendre ses aides pérennes.

## Présentation du projet

Le projet est de créer un site web dynamique pour promouvoir l’association, immerger les visiteurs au cœur de l’association à travers des articles présentant les différentes interventions sur lesquelles l’association travaillent au quotidien et enfin, mettre en avant les différents chats à sauver en fonction de leurs caractéristiques afin de leur trouver la famille qui leur correspondra le mieux à l’aide des formulaires de prise de contact.

Le site sera partagé en 2 parties principales.

### Partie administration :

La page d’administration sera accessible uniquement aux bénévoles qui auront la gestion du site internet via une page de connexion. Des droits spécifiques seront nécessaires pour y accéder (adresse e-mail et mot de passe). Plusieurs fonctionnalités seront disponibles sur cet espace.

* Créer, modifier, afficher, supprimer les postes de toutes les catégories qui alimenteront du site.
* Télécharger les formulaires complétés par les visiteurs sous format PDF. Ces formulaires seront aussi stockés dans la base de données.
* Avoir un tableau de bord fonctionnel permettant de suivre les indicateurs de performance du site.

### Partie visiteur :

La partie visiteur sera accessible par tout le monde. Elle sera composée de plusieurs pages :

* **Pages statiques :** 
  + Présentation de l’association
  + Les critères pour devenir une famille d’accueil
* **Pages dynamiques :** 
  + Pages des postes de la catégorie « Articles » (2 sous-catégories : nos conseils, nos sauvetages)
  + Pages des postes de la catégorie « Chats » (3 sous-catégories : chats à l’adoption, chats en attente de famille d’accueil et les chats sauvés).
  + Pages formulaires de contact. Il y aura 2 formulaires différents. (Devenir famille d’adoption et devenir famille d’accueil).

## Compétences couvertes par le projet

Travailler sur ce projet m’a permis de valider plusieurs compétences qui sont les suivantes :

Développement de la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité. Cette partie contient :

* Le maquettage de l’application
* La réalisation d’une interface utilisateur statique et adaptable
* Le développement d’une interface utilisateur web dynamique
* La réalisation d’une interface utilisateur avec une solution de gestion de contenu.

Développement de la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité. Cette partie contient :

* La création d’une base de données
* Le développement des composants d’accès aux données
* Le développement de la partie back-end d’une application web
* L’élaboration et la mise en œuvre des composants dans une application de gestion contenu.

# Cahier des charges

Le cahier des charges a été créé au début du stage après plusieurs rendez-vous téléphonique entre la présidente de l’association, ma tutrice, bénévole et moi-même.

Je connaissais très bien l’association avant le début de mon stage ayant un membre de ma famille bénévole très active de l’association et l’étant moi-même également. J’avais donc connaissance avant le début de mon contrat de stage des différents besoins et les difficultés que pouvaient rencontrer l’association. C’est pour cela que je leur ai proposé mes services pour créer le site internet.

## Expression des besoins du client

Nous avons établi dans un premier temps, une liste des fonctionnalités avec leurs niveaux d’importances et de nécessités.

### Les fonctionnalités indispensables

* Mettre en avant les chats en attente de famille d’accueil.
* Mettre en avant les chats qui sont à l’adoption.
* Pouvoir permettre aux visiteurs du site internet de remplir un formulaire pour nous faire part de leurs désirs de devenir famille d’accueil ou famille adoptante en répondant aux diverses questions.
* Rendre ses formulaires complétés téléchargeables sous format PDF pour améliorer le suivi et la gestion.

### Les fonctionnalités informatives

* Présenter les chats qui ont été sauvés par l’association. (Historique)
* Rédiger des articles qui permettront aux visiteurs d’être informés des différents sauvetages qu’a pu réaliser l’association.
* Rédiger des articles pour partager les « conseils » de l’association sur divers sujets.

### Les fonctionnalités additionnelles

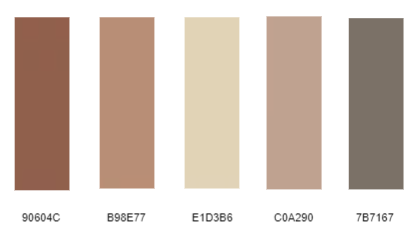
* Tableau de bord pour suivre les indicateurs de performance du site internet
* Evaluer la pertinence des formulaires complétés par une note d’appréciation. (Admin)
* Offrir la possibilité aux visiteurs de commenter les articles.
* Mettre en place un système de newsletter informatives.

## Charte graphique

L’association me laisse carte blanche sur l’aspect graphique général du projet, cependant il y a quelques contraintes à prendre en considération :

* Conserver le logo de l’association
* Choisir des couleurs sobres et modernes correspondant à l’identité d’une association de protection animal.

Voici la palette de couleur que j’ai proposé à l’association et qui a été validé unanimement par l’ensemble des bénévoles.



## Contraintes Techniques

**Accessibilité :**

Le site internet devra fonctionner sur n’importe quel type d’ordinateur et sur tous les navigateurs. Il devra également être adaptable aux différents écrans (ordinateur, tablette et mobile)

**Hébergement :**

Le site sera hébergé par un hébergeur mutualisé puisqu’il s’agit en général des hébergements les plus économique.

J’ai fait le choix de ce type d’hébergement car il est facile à configurer et à utiliser ayant déjà hébergé un autre projet sur ce type d’hébergeur.

**Planning :**

L’association ne m’a pas indiqué de date de livraison du projet, cependant il sera très apprécié que le projet soit finalisé d’ici la fin du mois d’aout 2021.

**Budget :**

Le projet est à destination d’une association qui n’a pas de moyen financier extensible. Il faudra donc que je limite l’utilisation d’outils payants et que je choisisse un hébergement qui soit d’une part adaptée aux besoins et d’autres parts qui soit en cohérence avec les moyens financiers de l’association.

# Spécifications fonctionnelles

## Diagramme des cas d’utilisations

Le cas d’utilisation regroupe l’ensemble des actions qui sont réalisables par le système de mon projet en interaction avec les rôles de chacun des acteurs.

Les rôles qui composent le cas d’utilisation sont :

* **Le visiteur**
* **Le rédacteur web**
* **Le rédacteur web connecté**
* **L’admin**

Des relations d’héritages sont visibles entre les acteurs, l’administrateur va hériter des fonctionnalités du rédacteur web connecté qui lui-même va hériter de la fonctionnalité attribuée au rédacteur web.

Pour simplifier l’utilisation pour les visiteurs, aucune création de compte et de connexion ne sera demandée pour pouvoir envoyer un formulaire.

* **Fonctionnalités du visiteur :**

- Consulter l’ensemble des pages du site ne nécessitant aucune demande de connexion. (Pages statiques, pages dynamiques)

- Envoyer un formulaire pour se proposer en tant que famille d’accueil

- Envoyer un formulaire pour se proposer en tant que famille adoptante.

* **Fonctionnalités du rédacteur web :**

- S’authentifier en renseignant e-mail et mot de passe afin d’accéder à la partie administrateur du site.

* **Fonctionnalités du rédacteur web connecté :**

- Créer, modifier, supprimer les articles et les chats

- Afficher la liste des articles par catégorie (tri)

- Afficher la liste des chats par catégorie (tri)

- Consulter les formulaires complétés.

- Générer sous format PDF les formulaires complétés.

- Fermer sa session

* **Fonctionnalités de l’admin :**

- Créer un nouveau compte pour ajouter un rédacteur web

- Créer en cas de besoin une catégorie supplémentaire

## Premier cas : Authentification du rédacteur web

### Diagramme d’activité

Ce cas d’utilisation intervient au moment où le rédacteur web souhaite s’authentifier.

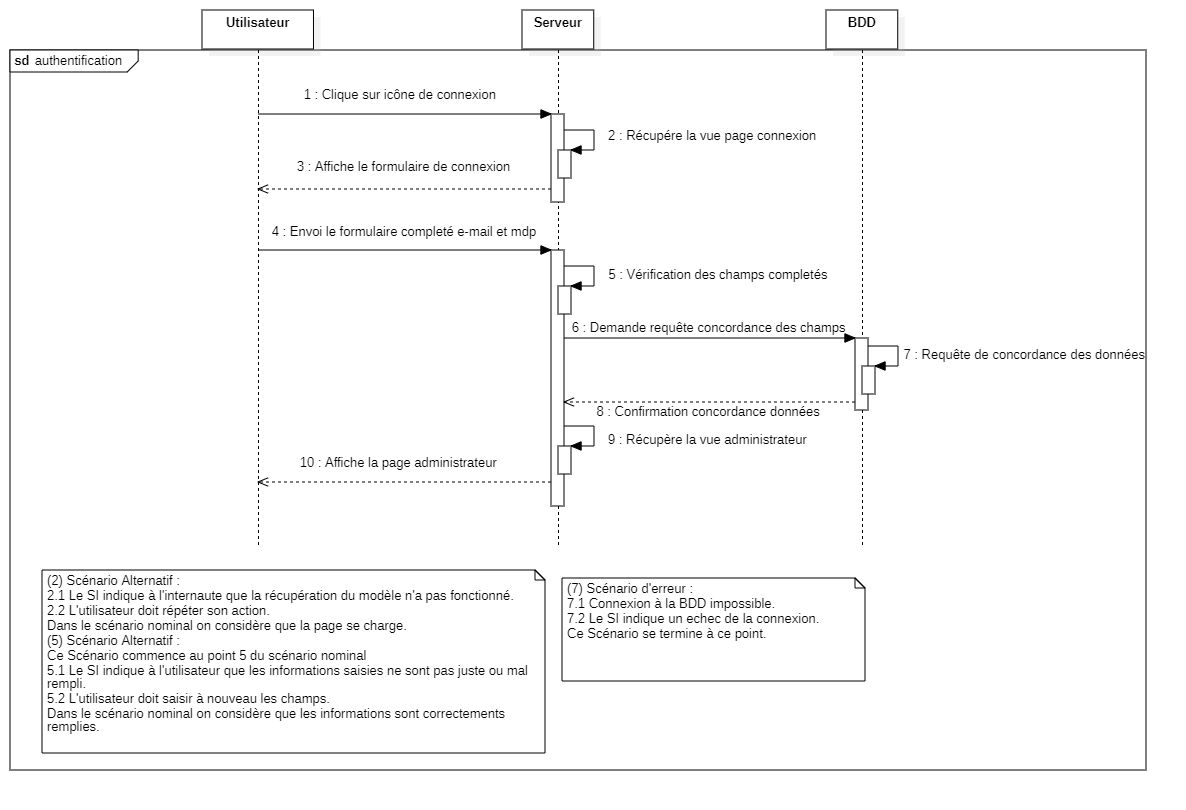
**Cliquer sur le bouton de connexion :** le rédacteur web demande l’accès à la page d’authentification.

**Compléter les champs et envoyer la demande :** il doit renseigner son e-mail et son mot de passe.

**Vérifier les champs complétés :** à ce niveau, il y a une 1ère vérification qui a lieu, la vérification des champs complétés. Si l’un des champs obligatoires est vide/nulle alors un message d’erreur apparaitra. Si la vérification est validée alors il y a exécution de la requête d’insertion dans la base de données.

**Exécuter la requête :** Ici ce trouve la 2ème vérification, une fois la demande de la requête exécutée il y a un nœud de décision, soit la requête a fonctionné ce qui finalise le diagramme d’activité, soit la requête n’a pas fonctionné et dans ce cas le serveur affiche un message d’erreur.

### Diagramme de séquence

Le diagramme de séquence est la représentation graphique des interactions entre les acteurs et le système en prenant en considération l’enchainement des actions par ordre chronologique. Les actions sont numérotées et se succèdent dans une dimension vertical représentant leurs successions dans le temps.

**Scénario nominal :**

1. Le rédacteur demande l’affichage de la vue correspondant au formulaire de connexion
2. Le formulaire se charge : Le Controller charge la vue.
3. La page s’affiche : La vue apparait sur l’interface utilisateur.
4. Le formulaire est complété et envoyé par le rédacteur web.
5. Vérification des champs par le serveur, les champs obligatoires doivent être renseignés.
6. Tous les champs sont renseignés la demande de requête est lancée.
7. Exécution de la requête demande de concordance des données.
8. Le modèle envoie une réponse au Controller : traitement des données effectué
9. Récupération de la vue
10. Affichage du tableau de bord. Connexion réussie.

Dans le scénario nominal on considère que l’ensemble des actions se déroulent sans cas d’erreur.

**Scénario alternatif :**

2.1. Le serveur ne récupère par la vue du formulaire. Le rédacteur doit répéter son action.

5.1. Le serveur indique à l’utilisateur que les informations saisies sont mal remplies. Le rédacteur web doit revoir les champs et ajuster en remplissant tous les champs demandés.

**Scénario cas d’erreur :**

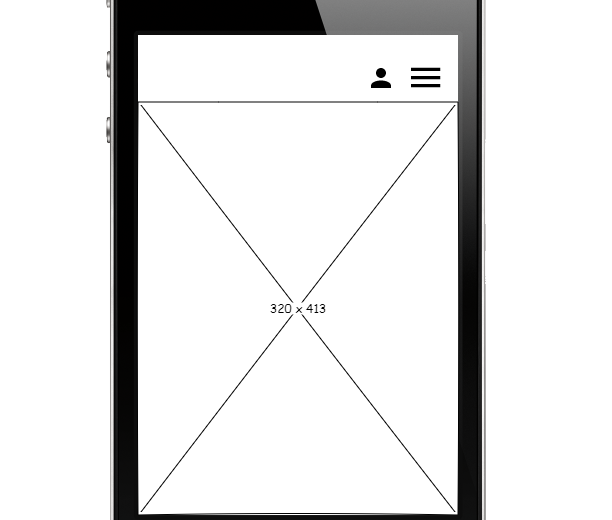
7.1. Connexion à la base de données est impossible, le serveur indique un échec de la connexion et donc la requête ne peut pas s’exécuter.

### Maquettage

**Version desktop :**



**Version mobile :**



**Version desktop et mobile :**



## Deuxième cas : Créer un chat « famille d’accueil »

### Diagramme d’activité

Ce cas d’utilisation intervient au moment où le rédacteur web connecté souhaite ajouter un chat dans la catégorie « famille d’accueil ».

**Cliquer sur le lien « ajouter un chat famille d’accueil » :** le rédacteur web demande la page comprenant le formulaire de création d’un chat famille d’accueil.

**Afficher le formulaire de création du chat :** la vue demandée se charge et apparait.

**Envoyer le formulaire complété :** le rédacteur web doit compléter l’ensemble des champs qui composent le formulaire et doit cliquer sur le bouton « créer ».

**Vérifier les champs complétés :** a ce niveau, il y a une 1ère vérification qui a lieu, la vérification des champs complétés. Si l’un des champs obligatoires est vide alors un message d’erreur apparaitra. Si la vérification est validée alors il y a exécution de la requête d’insertion dans la base de données.

**Exécuter la requête :** Ici ce trouve la 2ème vérification, une fois la demande de la requête exécutée il y a un nœud de décision, soit la requête a fonctionné ce qui finalise le diagramme d’activité, soit la requête n’a pas fonctionné et dans ce cas le serveur affiche un message d’erreur.

### b. Diagramme de séquence

**Scénario nominal :**

1. Le rédacteur demande l’affichage de la vue correspondant au formulaire de création.
2. Le modèle de création se charge : Le Controller charge la vue.
3. La page s’affiche : La vue apparait sur l’interface utilisateur.
4. Le formulaire est complété et envoyé par le rédacteur web.
5. Vérification des champs par le serveur, les champs obligatoires doivent être renseignés.
6. Tous les champs sont renseignés la demande de requête est lancée.
7. Exécution de la requête enregistrement des nouvelles données.
8. Le modèle envoie une réponse au Controller : traitement des données effectué
9. Affichage sur l’interface utilisateur du message envoyé par le Controller comme quoi la création a été effectué.

Dans le scénario nominal on considère que la demande d’affichage de la vue du formulaire se charge correctement et que les champs complétés par l’utilisateur soient valides. Nous partons du principe que le rédacteur web est authentifié ce qui lui donne l’accès aux fonctionnalités du rédacteur web connecté.

**Scénario alternatif :**

2.1. Le serveur ne récupère par la vue du modèle de création. Le rédacteur doit répéter son action.

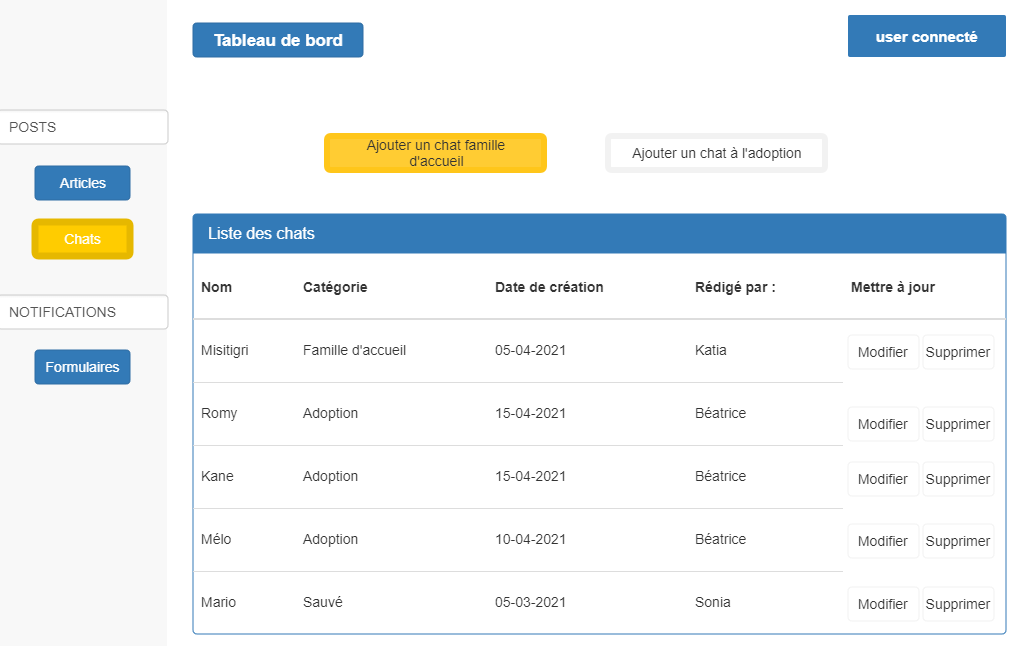
5.1. Le serveur indique à l’utilisateur que les informations saisies sont mal remplies. Le rédacteur web doit revoir les champs et ajuster en remplissant tous les champs demandés.

**Scénario cas d’erreur :**

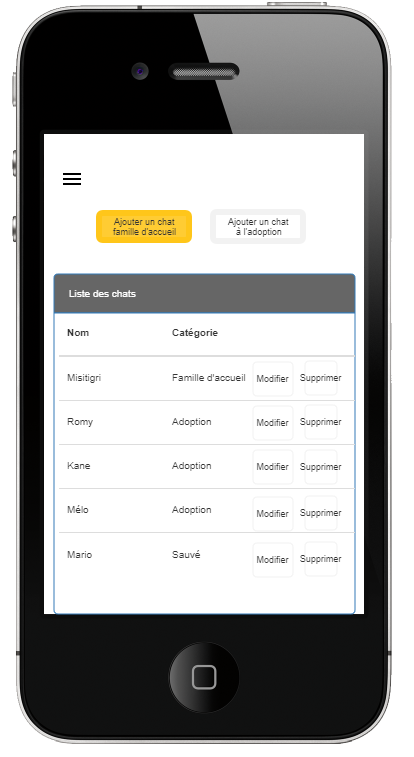
7.1. Connexion à la base de données est impossible, le serveur indique un échec de la connexion et donc la requête ne peut pas s’exécuter.

### c. Maquettage

Ci-dessous la maquette de la partie administration du site. L’utilisateur a déjà cliqué sur le bouton sur la gauche « Chats » ce qui affiche la liste des chats qui ont déjà été ajouté dans la base de données, toutes catégories confondues.

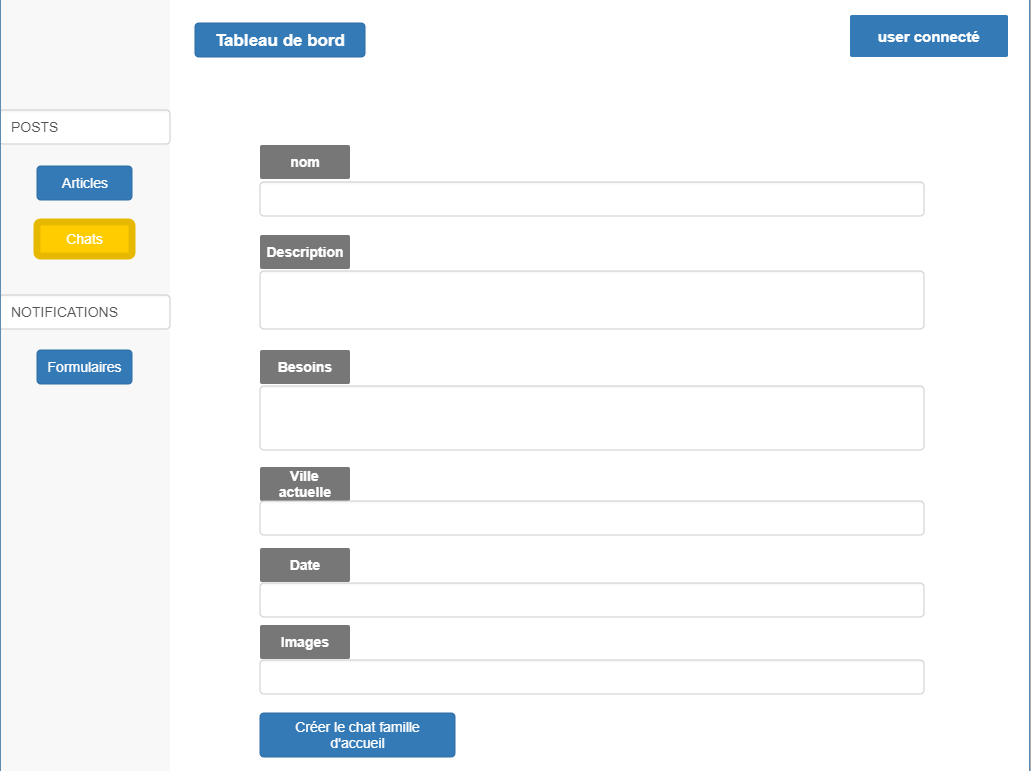
**Version desktop :**

**Version mobile :**

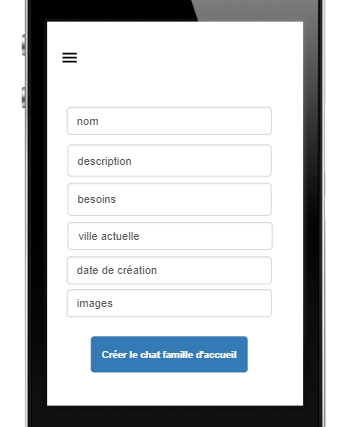


Ci-dessous la maquette du formulaire de création du chat famille d’accueil.

**Version desktop :**

****

**Version mobile :**



## 5. Diagramme de classe

Les utilisateurs se retrouvent dans la base de données et dans le diagramme de classes parce qu’ils ont des actions sur le site et il y a aussi un stockage de leurs données dans la base de données.

Mettre les classes Model + les classes Controller

Mettre les relations entre les classes, héritages, agrégations compositions, association, avec le verbe qui qualifié la relation. Ensuite générer le setter et getter.

# Spécifications techniques

## Base de données

### Dictionnaire de données

Afin de m’aider dans la conception de la base de données relationnelles, j’ai réalisé le dictionnaire des données. Il s’agit du document qui regroupe toutes les données qui seront stockées dans la base de données. J’ai décidé de créer 8 entités dans lesquelles j’ai répertorié les données utiles à enregistrer.

* User : correspond aux comptes d’accès des rédacteurs web.
* Cat : correspond aux fiches d’identités des chats de l’association.
* Category\_cat : les chats seront répertoriés par catégorie (3 catégories)
* Article : correspond aux contenus des articles.
* Category\_article : les articles seront répertoriés par catégorie (2 catégories)
* Image : Les chats et les articles possèderont des images qui leurs seront propres.
* Form\_fa : correspond aux formulaires complétés par les visiteurs qui souhaiteraient devenir famille d’accueil (fa = famille d’accueil)
* Form\_adoption : correspond aux formulaires complétés par les visiteurs qui souhaiteraient devenir famille adoptante.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| name | Pseudo de l’utilisateur | VARCHAR | (50) |  |
| email | E-mail de l’utilisateur | VARCHAR | (255) | Adresse unique |
| password | Mot de passe de l’utilisateur | VARCHAR | (255) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cat | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| name | Prénom du chat | VARCHAR | (255) |  |
| slug | Slug du chat | VARCHAR | (255) |  |
| teaser | Mots clés qualificatif du chat | TEXT |  | Valeur nulle autorisée |
| description | Description du chat | TEXT |  |  |
| requirement | Besoins du chat | TEXT |  | Valeur nulle autorisée |
| city | Ville actuelle du chat | VARCHAR | (255) |  |
| adoption\_date | Date adoption du chat | DATE |  |  |
| available\_date | Date disponibilité du chat | DATE |  |  |
| is\_sos | Chat à placer en urgence | BOOLEAN |  |  |
| visible | Visibilité du chat | BOOLEAN |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category\_cat | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| name | Nom de la catégorie | VARCHAR | (255) |  |
| slug | Slug de la catégorie | VARCHAR | (255) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Article | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| title | Titre de l’article | VARCHAR | (255) |  |
| slug | Slug de l’article | VARCHAR | (255) |  |
| content | Contenue de l’article | LONGTEXT |  |  |
| visible | Article visible ou pas | BOOLEAN |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category\_article | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| name | Nom de la catégorie | VARCHAR | (255) |  |
| slug | Slug de la catégorie | VARCHAR | (255) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Image | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| name | Nom de l’image | VARCHAR | (255) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Form\_fa | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| gender | Civilité | BOOLEAN |  |  |
| firstname | Prénom | VARCHAR | (255) |  |
| lastname | Nom | VARCHAR | (255) |  |
| mail | Adresse e-mail | VARCHAR | (255) |  |
| phone | Téléphone | VARCHAR | (255) |  |
| adress | Adresse | VARCHAR | (255) |  |
| postal | Code postale | VARCHAR | (255) |  |
| city | Ville | VARCHAR | (255) |  |
| already\_cat | Ont-ils déjà eu un chat ? | BOOLEAN |  |  |
| already\_fa | Ont-ils déjà été famille d’accueil ? | BOOLEAN |  |  |
| description\_fa | Si oui, décrire l’expérience | TEXT |  |  |
| family\_animal | Ont-ils en animal en ce moment ? | BOOLEAN |  |  |
| animal | Si oui, détailler | TEXT |  |  |
| room | Ont-ils une pièce pour isoler ? | BOOLEAN |  |  |
| room\_details | Si oui, décrire la pièce | TEXT |  |  |
| more\_space | Libre accès à la maison ? | BOOLEAN |  |  |
| questions | Ont-ils des questions ? | TEXT |  |  |
| is\_read | Formulaire lu ou pas encore. | BOOLEAN |  |  |
| Is\_rgpd\_ok | Autorisation enregistrement donnée | BOOL |  | REQUIRED |
| created\_date | Date de création du formulaire | DATE |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Form\_adoption | | | | |
| Nom | Désignation | Type | Taille | Remarques |
| gender | Civilité | BOOLEAN |  |  |
| firstname | Prénom | VARCHAR | (255) |  |
| lastname | Nom | VARCHAR | (255) |  |
| mail | Adresse e-mail | VARCHAR | (255) |  |
| phone\_1 | Téléphone 1 | VARCHAR | (255) |  |
| phone\_2 | Téléphone 2 | VARCHAR | (255) |  |
| house\_member | Nombre membres du foyer | VARCHAR | (255) |  |
| adress | Adresse | VARCHAR | (255) |  |
| postal | Code postale | VARCHAR | (255) |  |
| city | Ville | VARCHAR | (255) |  |
| family\_animal | Ont-ils en animal en ce moment ? | BOOLEAN |  |  |
| animal | Si oui, détailler | TEXT |  |  |
| room | Maison ou appartement | BOOLEAN |  |  |
| area | Superficie totale du logement | TEXT |  |  |
| have\_garden | Jardin ou pas ? | BOOLEAN |  |  |
| is\_working | Activité pro ou pas ? | BOOLEAN |  |  |
| is\_missing | Durée absence par jour | TEXT |  |  |
| questions | Ont-ils des questions ? | TEXT |  |  |
| Is\_read |  | BOOL |  |  |
| Is\_rgpd\_ok | Autorisation enregistrement donnée | BOOL |  |  |
| created\_date | Date de création du formulaire | DATE |  |  |

Les champs « created\_at » et « updated\_at » sont des champs automatiquement ajoutés par le framework que j’ai utilisé (Laravel 8) ils permettent de pouvoir ordonnés l’apparition des données (1 affiché => dernier crée ou inversement).

J’ai décidé de créer une table catégorie\_cat pour répertorier les chats, cela me permettra par la suite de pouvoir trier facilement les données en fonction de la catégorie attribuée au chat. Il existera donc 3 données dans l’entité « category\_cat ». J’ai fait de même pour les articles.

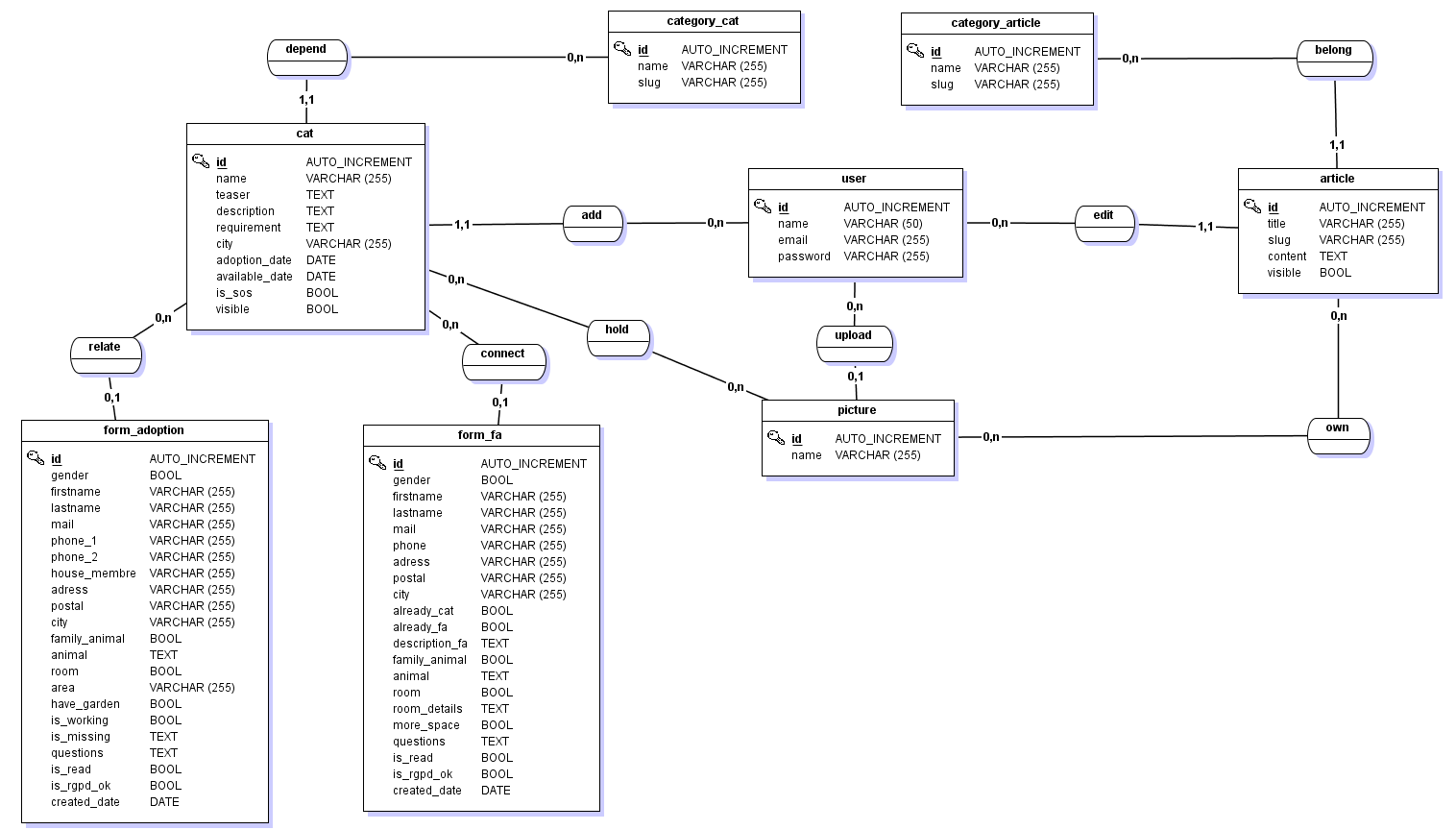
Concernant les formulaires, puisque les informations demandées entre les deux formulaires sont différentes, j’ai été contrainte de créer deux tables différentes pour chaque questionnaire.

### MCD

Le modèle conceptuel de données ci-dessous représente les données du dictionnaire de données de façon schématique. Nous retrouvons dans chaque entités les différents attributs qui constitueront les données.

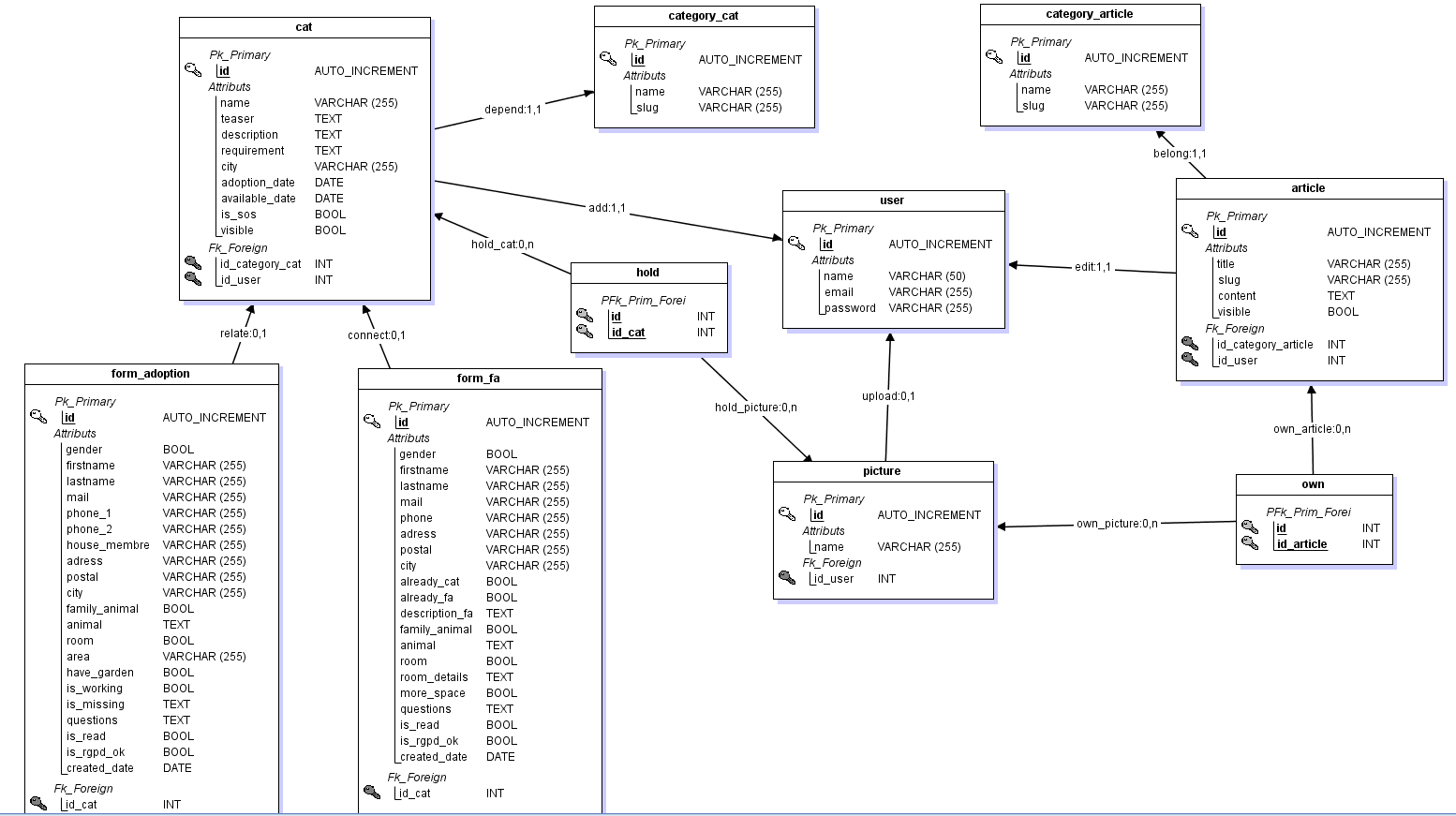
Chaque entité possède un attribut « id », cet attribut représente l’identifiant de l’entité. Il permettra d’identifier une instance de cette entité grâce à son auto incrémentation. A chaque instanciation l’identifiant augmentera de 1. De ce fait, il permet de rendre unique chaque instance.

Les entités sont reliées entre elles par des relations d’associations auxquelles j’ai attribué des cardinalités.



### MLD

Le modèle logique de données est la représentation des données du système d’information. Il est



## Choix Techniques

### a. Environnement de travail et langages

* **Système d’exploitation**

Le système d’exploitation utilisé est Windows 10.

* **Editeur de code**

L’éditeur de code utilisé est Visual Studio Code.

* **Système de versionnage**

Pour versionner les fichiers j’ai utilisé GitHub sur un repository privée pour respecter la confidentialité du projet.

* **Outils de conception**

StarUML : Diagramme cas d’utilisation, diagrammes d’activité, diagrammes de séquence

Jmerise : MCD, MLD

* **Wireframes**

Pour réaliser les wireframes j’ai utilisé le logicien Pencil.

**Langage de programmation développé pour les projets web.**

* **Php 7.4**
* **HTML CSS**
* **Javascript**
* **MySQL**

**Frameworks utilisés pour ce projet :**

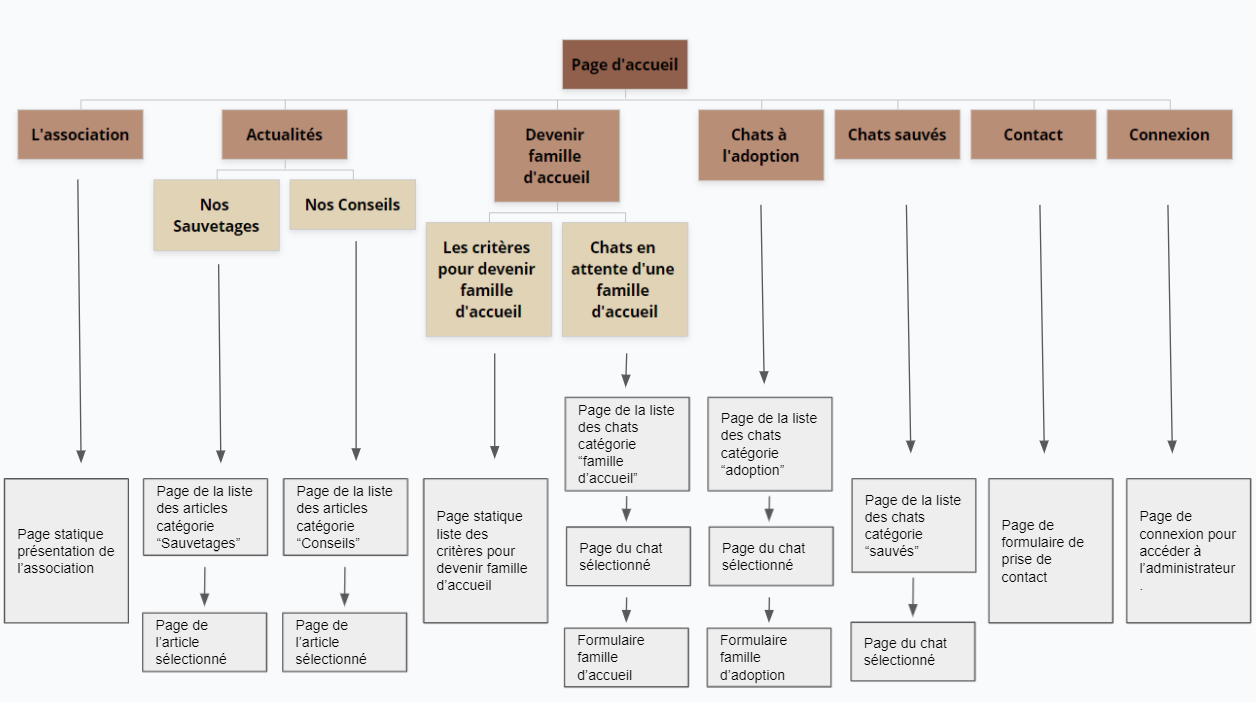
* **Bootstrap 5**
* **Laravel 8**

**Serveur local :**

* **Laragon**

# Arborescence

L’arborescence d’un site internet est une représentation schématique qui définit l’architecture du site. Elle permet de structurer et d’organiser son contenu et offrir une vision globale des données accessibles.

1. Arborescence utilisateur:

On retrouve dans cet arborescence le menu du site ainsi que toutes les pages qui en découlent. L’onglet « Connexion » correspond au bouton qui permettra aux rédacteurs web d’accéder à la partie administrateur.

Il sera visible par tous les utilisateurs mais pour pouvoir y accéder il faudra impérativement renseigner les champs requis pour valider la connexion en accord avec les données stockées dans la base de données.

## Arborescence administrateur :

# Implémentation

## 1. Création du projet sous Laravel 8

### a. Présentation de Laravel et de son ORM Eloquent

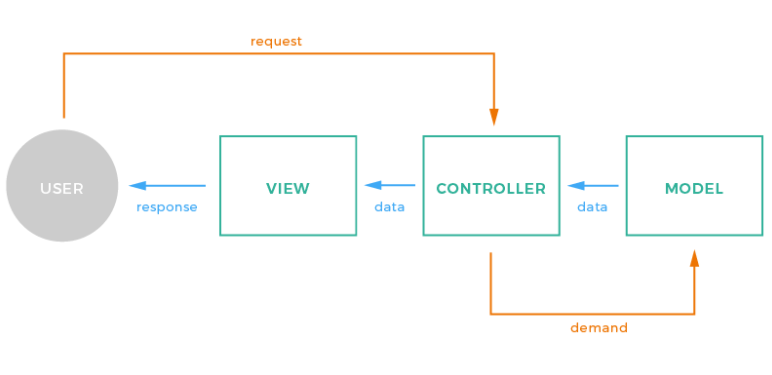
#### Framework

Laravel 8 est un framework qui va offrir une structure de base solide au projet. Il nous met à disposition des librairies intégrées de programmation en PHP nous permettant de gagner du temps, mais surtout de partir d’une base fiable puisque ces librairies sont toujours validées et mises à jour très régulièrement. Il nous sera possible d’installer plusieurs librairies et de les faire cohabiter ensemble grâce à Laravel.

#### Composer

Laravel utilise comme outil de gestion Composer. Il s’agit d’un logiciel gestionnaire de dépendances libre codé en PHP. Toutes ces dépendances sont répertoriées dans le fichier « composer.json » qui se trouve à la racine du projet. On indiquera à Composer quels paquets ou bibliothèques nous souhaitons gérer/intégrer au projet et automatiquement le logiciel va l’exécuter. Ce logiciel sera donc utilisé très régulièrement en fonction des besoins du projet. Laravel ne peut pas fonctionner sans Composer. Toutes les librairies du vendor sont chargées par composer. C’est d’ailleurs à l’aide de composer que nous allons installer laravel sur notre projet.

#### MVC



Laravel respecte l’architecture MVC.

* **Model** : contient les données et leur logique
* **View** : contient la présentation visuelle à renvoyer à l’utilisateur
* **Controller** : traite les actions utilisateurs (via des requêtes http), il demande au modèle d’effectuer les opérations, puis passe les données à la vue pour l’affichage.

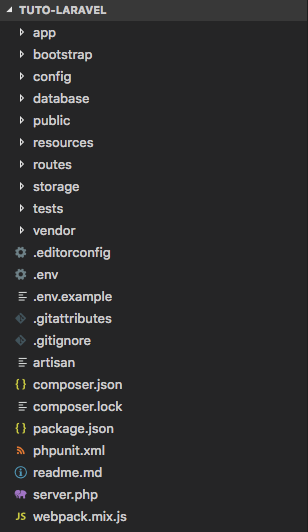
Toutes les actions de l’utilisateur passent par le contrôleur qui envoie des demandes de manipulations d’un objet au modèle. Le modèle effectue les modifications de l’objet et le renvoie au contrôleur qui à son tour l’affichera à la vue. Le contrôleur permet de faire le lien entre le modèle et la vue. Il gère les demandes de l’utilisateurs et le traitement des données.

*Dans le fichier Model on retrouvera les champs de la table et ses relations qu’il y a avec les autres tables et c’est grâce à sa lorsque je fais une migration, que les données vont se créer en prenant en considération les configurations de création de la base de données.*

*Dans le fichier Controller on y retrouve les fonctions qui appelleront les tables de la base de données.*

#### Architecture Laravel

Voici l’architecture des dossiers qu’installe Laravel. Je vais vous présenter les dossiers que j’ai utilisé pour mon projet.

****

**App** : Dans le dossier app nous allons retrouver les dossiers Models, Controllers, Requests.

**Database** : Fichiers liés à la base de données

* **Factories et seeders** : ajouter des données factices
  + - **Migrations** : Création des tables et de leurs champs

**Public** : Dossier accessible depuis le serveur (images, vidéos, css, js)

**Resources** : Fichiers css et js et le dossier views qui comporte tous nos fichiers blade.php [(cf 6. Template blade.php)](#_Template_blade.php)

**Routes** : fichier web.php qui correspond au routeur.

**.env** : fichier de configuration de la base de données et l’environnement de l’application.

**Composer.json** : Fichier de gestion des dépendants

**Webpack.mix.js** : préprocesseur CSS et JS courants.

Une fois Laravel installée, pour pouvoir accéder au projet j’utilise la commande :

**Php artisan serve**

Cette commande permet d’exécuter le fichier artisan qui se trouve également dans la structure de base de Laravel. En exécutant cette commande j’ai accès à mon site tout comme si j’y accédais en tapant l’url localhost :8888/chatslibres/public.

Une 2ème commande est nécessaire à exécuter afin de voir les modifications visuelles apportées sur le site instantanément :

**Npm run watch**

#### Eloquent

Laravel dispose d’un ORM qui se nomme Eloquent. Celui-ci va permettre de traiter les données relationnelles. En effet les tables qui composeront la base de données seront souvent liées les unes avec les autres par des relations. Eloquent va me permettre d’accéder et de traiter les données en prenant en considération ses relations sur toutes les opérations CRUD par exemple. Eloquent propose des fonctions prédéfinies de requêtes SQL. *(Abstraction avec la base de données (interface) et me permet aussi de récupérer la base de données en synchronisant ses données à l’intérieur du modèle.)*

#### Template blade.php

Blade est le moteur de template utilisé par Laravel. Son but est de permettre d’utiliser le php sur notre vue. Pour chaque fichier qui va constituer notre vue il faudra ajouter l’extension «.blade.php ». Cela signifie que nous utiliserons le template.

L’utilité du moteur de template blade est de pouvoir manipuler des données comme sur un fichier php. La structure de contrôle varie cependant à celle de php. Les conditions et les boucles sont annoncées avec un « @ ». Les appels de données se font à l’aide des {{ }}.

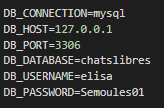
Il est possible également de créer des sections de contenu que l’on ouvre grâce à @section() et que l’on ferme avec le @endsection(), sous forme de layout.

### Création du projet sous Laravel 8

Pour créer le projet j’ai utilisé composer en utilisant la commande **create-project chatslibres** en indiquant le dossier dans lequel je veux créer mon application. Une fois le projet crée, lorsque j’ouvre le dossier de mon projet sur VS Code je constate que toute l’architecture de dossier Laravel est disponible depuis le dossier de mon projet.

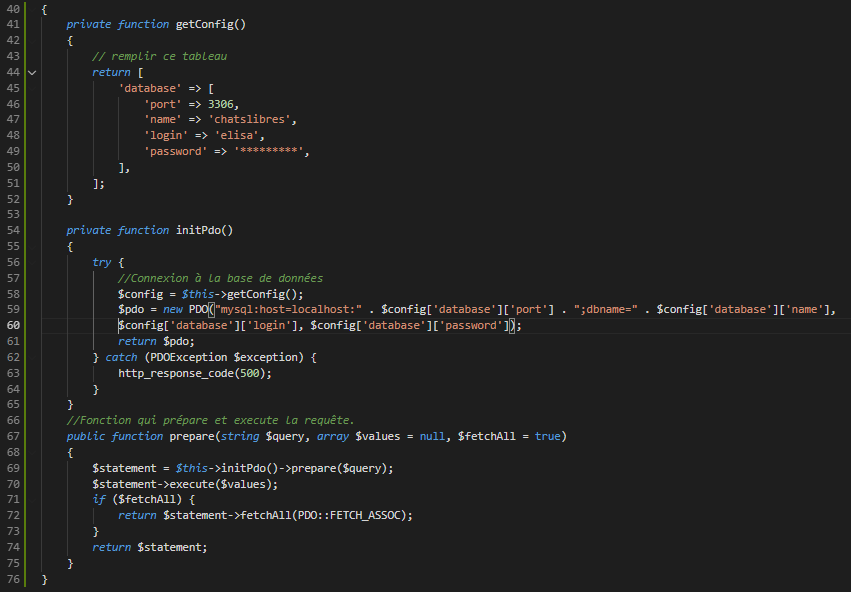
Une fois mon projet lancé, je vais dans un 1er temps le configurer afin que celui-ci soit connecté à la base de données.

C’est dans le fichier «.env » que je vais configurer la base de données.



Je renseigne ici le nom de la base de données le nom de l’utilisateur et le mot de passe associé.

*Si je n’avais pas utilisé Laravel, voici comment j’aurais procédé pour la connexion avec la base de données.*



### Création d’une table avec Laravel 8

Pour créer la base de données de mon projet j’ai utilisé l’outil d’administration de base de données HeidiSQL. Il possède un éditeur de langage SQL et un constructeur de requête.

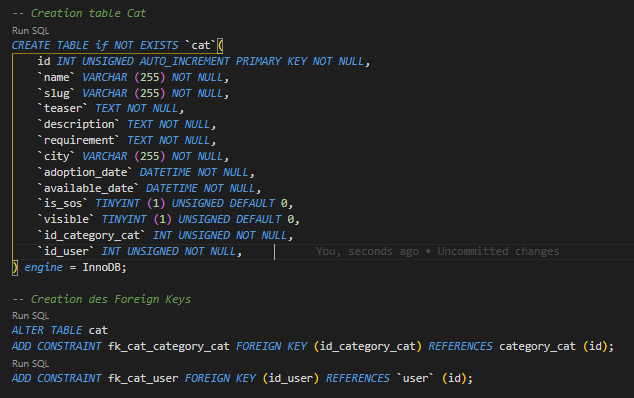
J’ai dans un 1er temps, à partir du gestionnaire de session, crée la session que j’utiliserai pour la gestion de la base de données de mon projet. Une fois la session créée je peux dans un 2eme temps créer la base de données et l’ensemble des tables qui la composera.

Aillant utilisé le framework Laravel pour mon projet, j’ai utilisé les outils qu’il propose pour créer mes tables, cependant je vais vous présenter ici comment je l’aurais fait si je n’avais pas utilisé Laravel.

Création de la base de données :

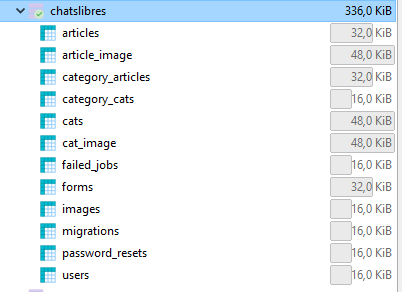


Création de la table « Cat » avec ses Foreign Keys :

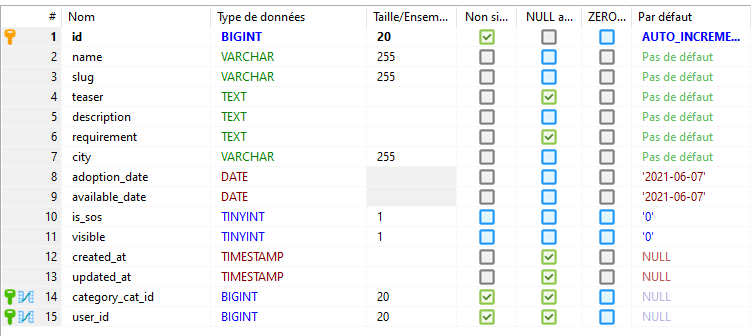


Je répète cette opération pour chaque table. Une fois mon fichier .sql complet, je l’intègre à mon outil HeidiSQL.

Voici un aperçu des tables que j’ai créé. On constate que toutes les tables sont au pluriel. Puisque j’ai utilisé Laravel, par convention, le nommage des tables doit être au pluriel et en anglais. J’ai donc passé toutes mes tables au pluriel.



Voici ci-dessous un aperçu de la table « cats » sur HeidiSQL. Nous retrouvons 2 champs qui n’étaient pas présents lors de la création de la table qui correspondent à des champs créés par la méthode timestamps de Laravel. « created\_at » et « updated\_at » qui me permettront de pouvoir ordonner mes données par leur date de création ou date de modification.



Sur Laravel pour créer des tables dans la base de données je vais utiliser les Migrations. Le dossier migration se trouve dans le dossier database. Le dossier migration est composé de plusieurs fichiers php qui correspondent aux tables que l’on souhaite insérer dans la base de données.

3 classes sont utilisées :

* Migrations
* Facade schema
* Class builder

Maintenant que mon projet est relié à la base de données, je vais pouvoir créer les tables avec leurs champs qui composeront la base de données. Pour cela je vais essentiellement utiliser l’outils de gestion des taches en ligne Artisan. Il existe différentes commandes qui permettent de créer des fichiers.

J’ai pour ma part utilisé majoritairement cette commande :

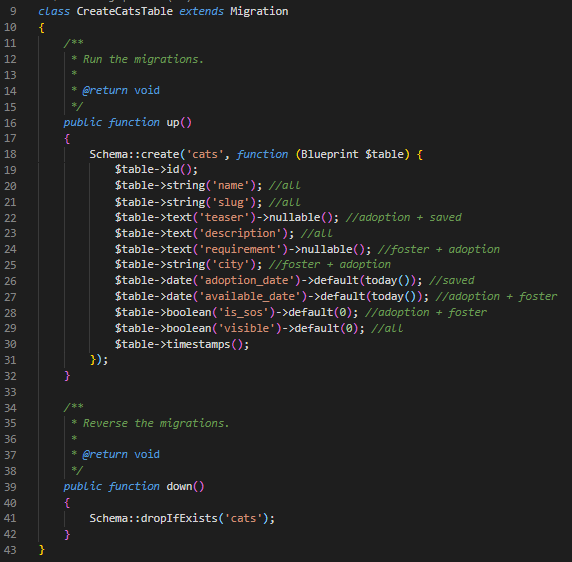
**Php artisan make : « nom de la table » -mcr**

La commande me permet de créer pour chaque table un fichier Migrate, Controller et Model.

Pour créer la table nous allons compléter le fichier de migration.

Il y a 2 fonctions

* Fonction up() : ajouter des éléments
* Fonction down() : drop des éléments



**Point 3**

**Point 2**

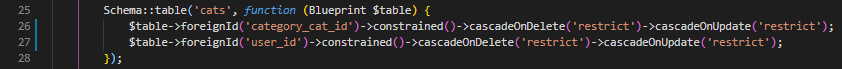
**Point 1**

**Point 1** : On appelle la façade schema qui me permet de gérer le schéma de la base de données et on exécute la fonction create().

**Point 2** : On indique tous les champs à ajouter dans la table en lui indiquant le type et la valeur par défaut si besoin.

**Point 3** : la fonction down() permet en cas de migration de pouvoir supprimer la table que l’on indique dans la fonction drop() et de pouvoir ensuite créer la nouvelle table.

Afin d’éviter les conflits au moment de l’import de mes tables, j’ai décidé de créer un fichier pour les « foreign keys » à part, dans lequel j’indique toutes les clés étrangères de mes tables.



On peut voir que j’ai ajouté des options à mes clés étrangères :

* Constrained() : cette fonction utilise des conventions pour déterminer le nom de la table et de la colonne référencées. Si le nom de la table ne correspond pas aux conventions alors on peut ajouter comme argument le nom de la table.
* cascadeOnDelete(‘restrict’) : en cas de suppression on applique la restriction « restrict ». Cela signifie qu’on empêche la suppression d’une catégorie ou d’un utilisateur.
* cascadeOnUpdate(‘restrict’) : en cas de modification on applique la restriction « restrict »

Une fois les fichiers entièrement complétés afin de l’ajouter dans la base de données il suffit d’utiliser la commande.

**Php artisan make :factory « nom du fichier » --model= « nom du fichier »**

Cette commande me permet de créer le fichier factories en important les données de la table dans laquelle je souhaite ajouter des données fictives.

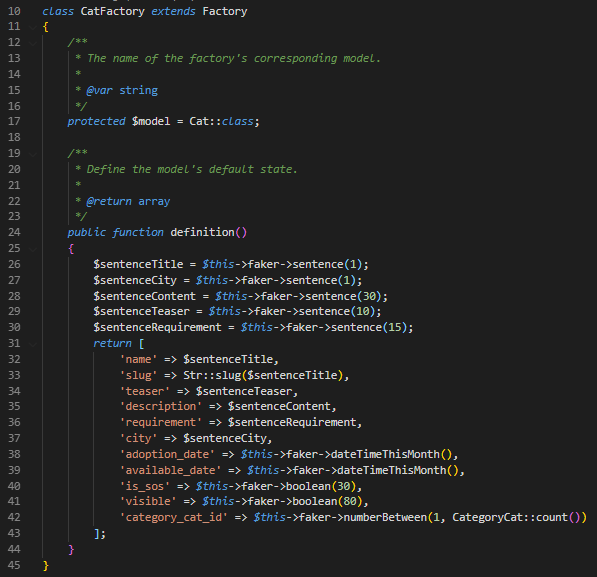
### Ajout de données fictives avec Laravel 8

Au début du projet, j’ai utilisé la méthode de Laravel pour ajouter des données fictives à mes tables afin de pouvoir travailler sur l’affichage de mes données sans avoir à utiliser des requêtes d’insertions. Pour cela j’ai utilisé les Seeders et les factories.

**Php artisan migrate**

Dans un 1er temps je vais compléter le fichier factories qui e permet de configurer les données de chaque champ. Pour cela j’utilise la librairie **faker** afin de générer diverses valeurs aléatoires.

La fonction sentence() détermine le nombre de mot que je souhaite intégrer dans la donnée.



Une fois le fichier factories complété, je m’occupe du fichier databaseseeder.php. J’utilise la fonction factory() que j’importe du fichier Factories appartenant à l’ORM Eloquent.

Je renseigne le nombre de « chat » que je souhaite créer et puisqu’un chat doit obligatoirement posséder une catégory je lui assigne une catégorie avec l’id 1. (j’ai au préalable également utilisé l’ajout de données pour la table « categoryCat ».) Et pour finir j’applique la fonction create() afin de l’ajouter dans la base de données.



Une fois mes fichiers completés j’utilise la commande suivante qui permet d’annuler toutes les précédentes migrations et de les mettre à jour.

**Php artisan migrate :refresh --seed**

## Premier cas : Authentification

L’authentification étant une tâche commune à de nombreux site, Laravel met à disposition directement dans le squelette de base la table « users » qui sert de référence pour la fonction d’authentification.

Je vais vous présenter ci-dessous l’ensemble des fichiers que j’ai utilisé pour la fonctionnalité authentification.

#### Routeur (web.php)

Construction du routeur :

J’ai donc 3 routes qui me permet de gérer l’authentification

* 1ère route : fonction index() me permet d’accéder à la page de connexion
* 2ème route : fonction login() Permet de gérer la connexion avec la soumission du formulaire.
* 3ème route : fonction index() me permet d’afficher le dashboard une fois la connexion réussie.

Chaque route est construite de la même façon.

**Point 1** : On remarque que 2 méthodes sont utilisées :

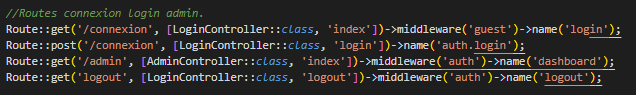
* GET = permet d’obtenir des données, les paramètres sont écrites dans l’url
* POST = permet de gérer la soumission de formulaire, paramètres non visibles dans l’url.

**Point 2** : l’url de la route

**Point 3** : On appelle la fonction que l’on souhaite utiliser et on nomme la classe du contrôleur dans lequel elle se trouve.

**Point 4** : Middleware : vérification supplémentaire entre les requêtes http qui permet de vérifier si l’utilisateur est bien authentifié ou pas. S’il ne l’est pas il n’a donc pas accès à la route.

* ‘guest’ : comme son nom l’indique tout le monde peut accéder à cette page.
* ‘auth’ seul les utilisateurs authentifiés peuvent accéder à cette page.

**Point 5** : on nomme la route c’est ce nommage que l’on va utiliser dans les vues pour faire les liens entre chaque page.

**3**

**5**

**44**

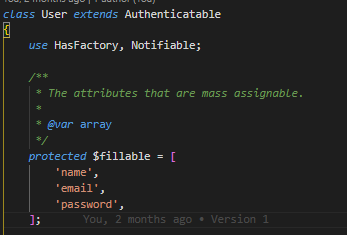
**2**

**1**

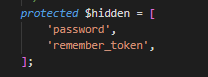
#### Modèle (users.php)

Le fichier modèle est celui dans lequel je vais renseigner les champs qui vont composer ma table ici nous avons pour l’utilisateur 3 champs ‘name’ ‘email’ ‘password’. Pour l’authentification 2 champs sont requis, l’adresse e-mail et le mot de passe.

Dans ce fichier nous pouvons également y renseigner les relations que la table peut avoir avec d’autres tables.



Il regroupe également les champs avec la propriété « $hidden » cela signifie que ces champs seront cachés lors de leurs récupérations pour une demande d’affichage d’une liste d’utilisateurs par exemple. Le token correspond à la clé de récupération pour le mot de passe.



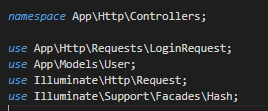
#### Contrôleur (loginController.php)

Dans mon contrôleur j’ai créé la fonction login() qui permet l’authentification de l’utilisateur. Il y a également la fonction logout() qui permet la déconnexion de l’utilisateur.

La fonction login possède un paramètre « LoginRequest ». On constate que pour mettre en place ma fonction login je récupère en amont grâce aux imports plusieurs fonctionnalités dont :

* Class **LoginRequest** (2 fonctions) :
  + authorize() contrôle de sécurité sur l’émetteur
  + rules() : les champs obligatoires nécessaires pour autoriser l’authentification.
* Model **User** :
  + Récupération des champs qui composent la table user.
* Class **Request** :
  + Utilisé pour la fonction Logout.
* Class **Hash** :
  + Possède la fonction qui permet de hasher le mot de passe.

Différence entre la classe « LoginRequest » et la classe « Request ». LoginRequest a toutes les fonctionnalités de Request et certaines autres fonctionnalités comme la validated().



**Point 1** : La vérification des champs du formulaire peut être établi de 2 manières similaires soit utiliser la fonction validated() soit utiliser la façade Validator ::make.

**Point 2** : 1ère vérification : Je vérifie si l’adresse e-mail renseigné dans le formulaire existe dans la BDD. Si elle existe alors je la récupère en 1er avec la fonction First(). Elle va permettre de récupérer une collection de plusieurs éléments, et donc le first va permettre de récupérer le 1er élément de la collection.

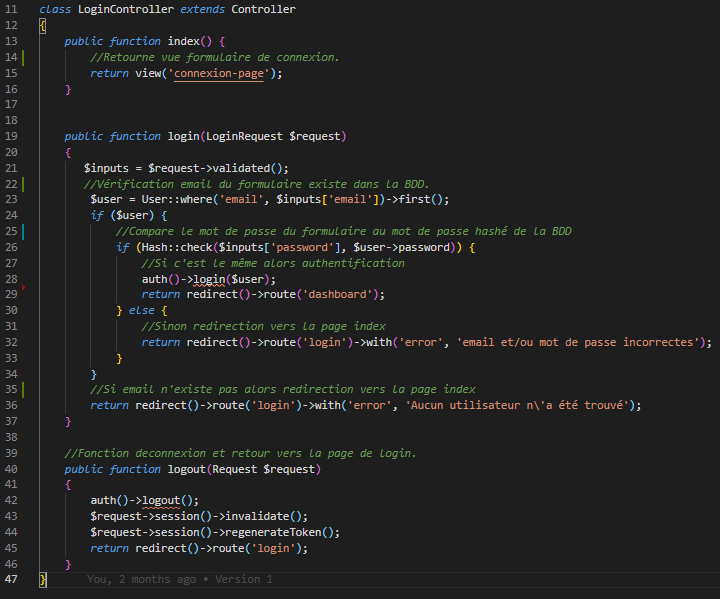
Si l’adresse e-mail existe et n’est pas nulle alors on passe à la 2ème vérification qui est celle du mot de passe. Si ce n’est pas le cas on bascule sur la ligne 37 avec la redirection vers la route du formulaire avec un message d’erreur indiqué.

**Point 3** : 2ème vérification : Je compare le mot de passe du formulaire (en le hachant) au mot de passe stocké dans la BDD. Si ce n’est pas le cas alors je bascule à la ligne 33 qui redirige vers la route « login » du formulaire en affichant le message d’erreur indiqué.

Hash::check -> prend le mot de passe du formulaire il va le hasher et quand je fais le hash ::check, le mot de passe renseigné sera d’abord hashé avant de le vérifier puisque le mot de passe enregistré sur la base de données est stocké hashé.

Si le mot de passe concorde alors je passe au point 4.

**Point 4** : Authentification de l’utilisateur récupéré en 1er avec la fonction auth(). Et je redirige l’utilisateur authentifié vers la route qui se nomme « dashboard » qui est la partie administrateur du site.



**Fonction déconnexion de la session.**

**Point 4**

**Point 3**

**Point 2**

**Fonction : affiche la vue du formulaire**

**Point 1**

#### Vue (connexion-page.blade.php)

Ci-dessous le code du formulaire d’authentification.

Dans la balise <form> nous retrouvons la méthode POST permettant la soumission du formulaire et en action la route vers laquelle l’utilisateur sera redirigé si l’authentification aboutit.

Sur la ligne 8 il y a « @crsf » qui doit être intégré dans chaque formulaire. Il s’agit d’un filtre qui permet de protéger contre les attaques CSRF (Cross Site Request Forgery). L’attaque peut-être dans notre cas, de transmettre à l’utilisateur authentifié une requête http falsifiée.

Sur les lignes 10-11, 15-16 et 19 permet de positionner l’endroit où les messages d’erreurs s’afficheront sur la page. (Voir la démonstration visuelle)



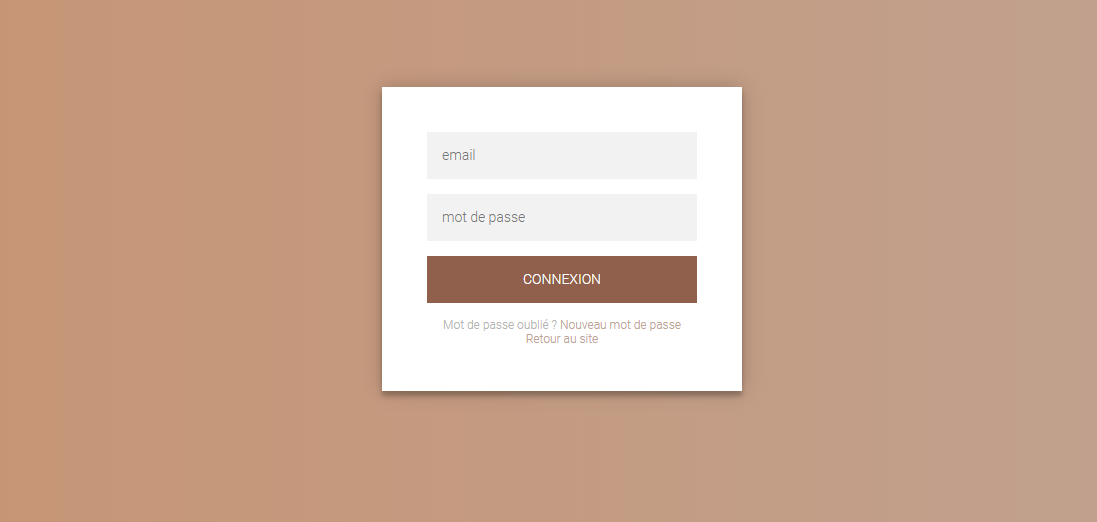
#### Front-end

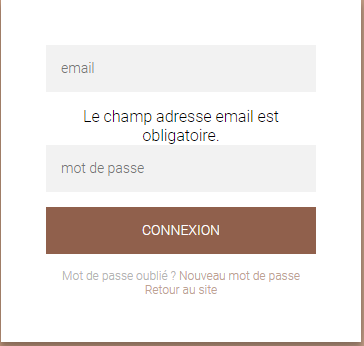
Concernant la partie front-end, j’ai décidé de réaliser la partie visible par les utilisateurs en HTML CSS et quelques librairies JS pour animer l’affichage tandis que pour la partie administrateur afin de gagner du temps j’ai décidé d’utiliser bootstrap.

Pour se connecter : cliquer sur l’icône en haut à droite



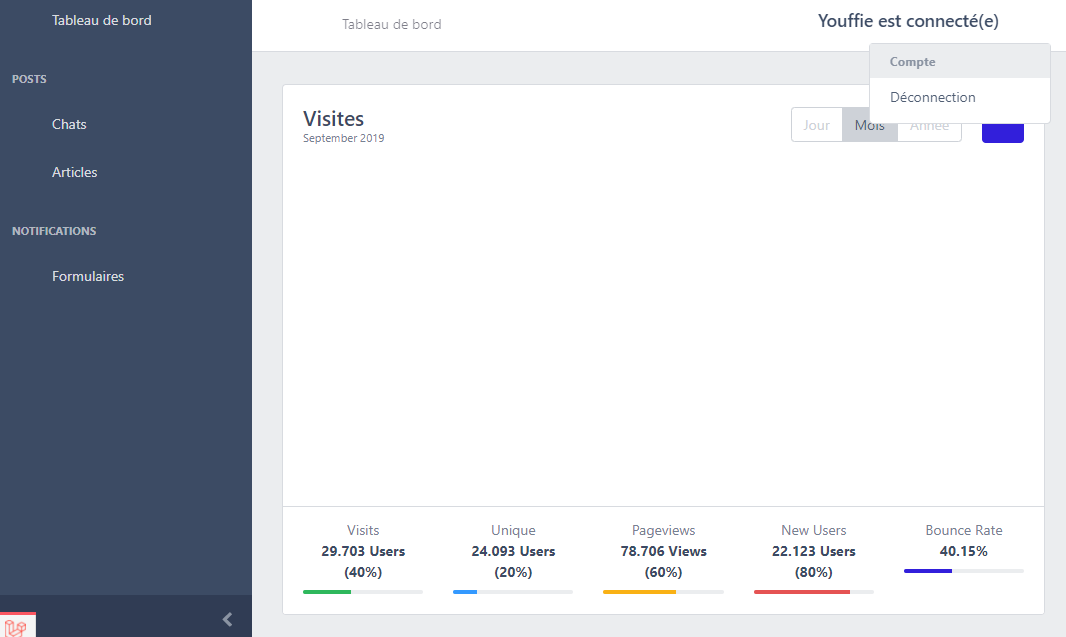
Accès au formulaire de connexion. Renseigner e-mail et mot de passe.



**Cas d’erreur 1**  **Cas d’erreur 2**

Authentification réussie -> accès au « dashboard » autorisé.

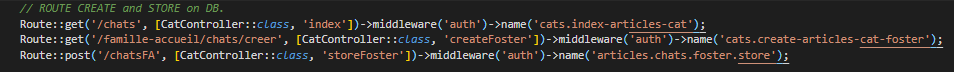


## Deuxième cas : Créer chat famille d’accueil.

#### 1.Routeur (web.php)

Construction du routeur :

J’ai donc 2 routes qui me permet de gérer l’authentification

* 1ème route : fonction create() permet d’afficher le formulaire de création
* 2ème route : fonction store() permet de vérifier les champs et d’enregistrer la création de données.

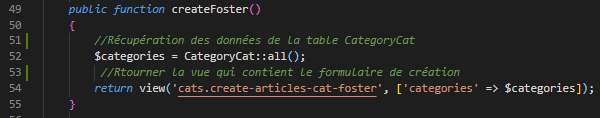
#### 2. Model (cat.php)

La table « Cats » est composée de 10 champs et les relié à 3 tables avec des relations différentes.

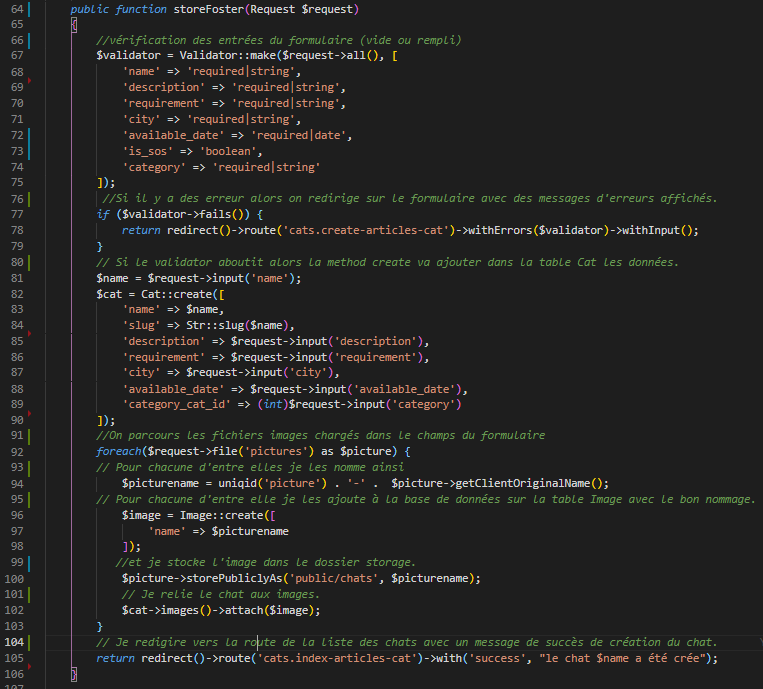
* Relation **BelongsTo** : Le chat appartient à une seule catégorie. (La table catégorie de son coté aura la relation hasMany puisqu’une catégorie peut avoir plusieurs chats.
* Relation **BelongsToMany** : Le chat possède plusieurs images et inversement. Une table pivot images\_cat va se créer.
* Relation **HasMany** : Le chat peut être relié à plusieurs formulaires ce qui justifie la relation HasMany.



#### Controller (CatController.php)



La fonction createFoster() me permet tout d’abord de récupérer les données de la table « CategoryCat » et ensuite me retourne la vue du formulaire.



La fonction StoreFoster() est la fonction qui me permet d’insérer dans la base de données le chats dans la catégorie « famille d’accueil ». On distingue 3 sections de code

**Section 1** : On utilise la facade Validator en indiquant toutes les entrées ($request->all()) et les règles de validation. Si la validation des champs échoue « fails » alors on redirige avec un message d’erreur.

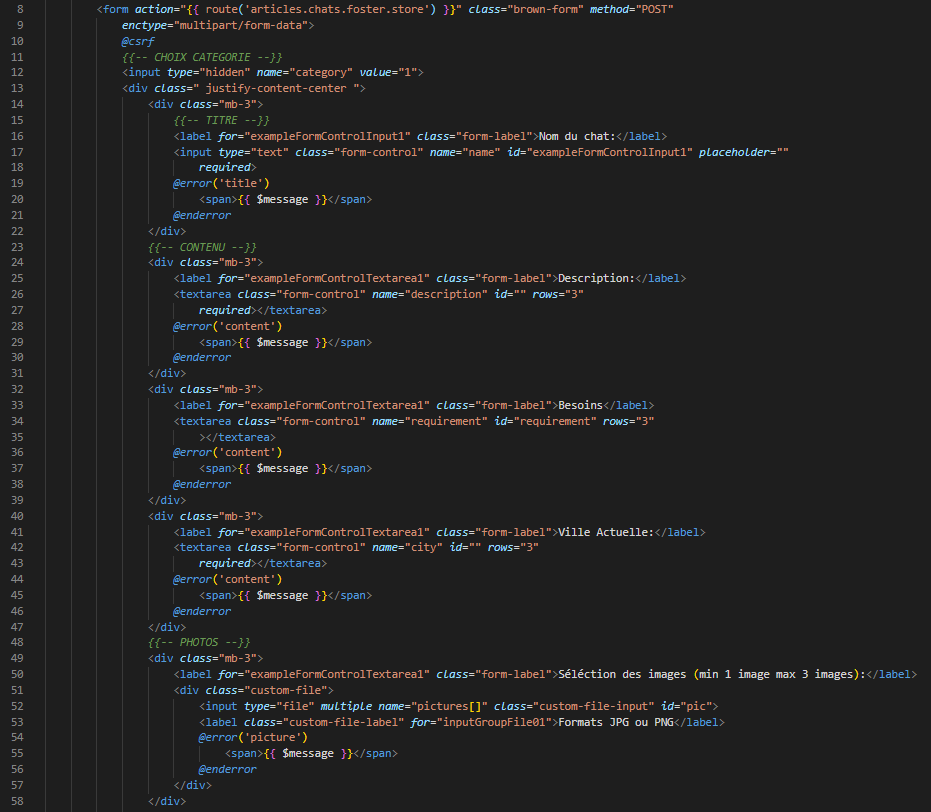
**Section 2** : On utilise la fonction create() qui va enregistrer dans la base de données la nouvelle données avec tous les champs qui la compose.

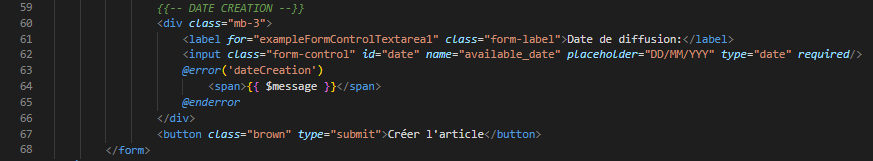
**Section 3** : Pour chaque fichier image qui a été ajouté dans l’input « pictures » nous allons

* Modifier son nom avec la fonction uniqid suivi du nom qui a été indiqué par l’utilisateur
* Ajouter cette image dans la base de données dans la table Image.
* Stocker cette image dans le dossier « storage » en public.
* Associer cette image au chat qu’elle représente.

**Voici comment j’aurai réalisé la requête de création en SQL :**

#### Vues (create-articles-cat-foster.blade.php)

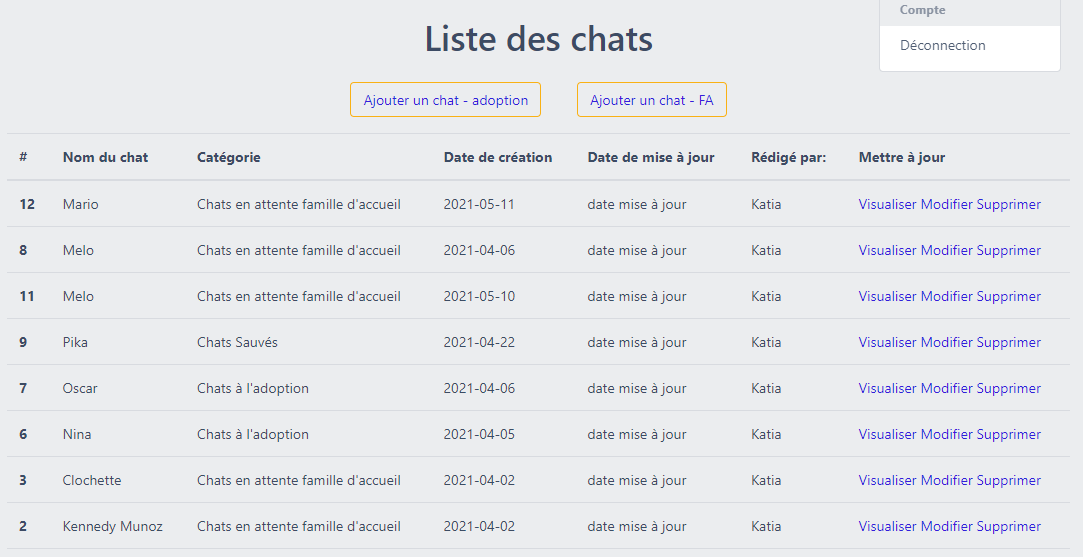




Le formulaire de création possède une méthode POST et une action qui redirige en cas de succès vers la route de la fonction storeFoster().

#### Front-end

#### Démonstration visuelle





CRUD signifie en anglais Create, Read, Update, Delete qui sont les 4 opérations principales d’une base de données (Insérer, Afficher, Mettre à jour et supprimer). Je vais vous présenter comment j’ai réalisé le CRUD avec le framework laravel 8 et vous montrer comment je l’aurai réalisé sans ce framework.

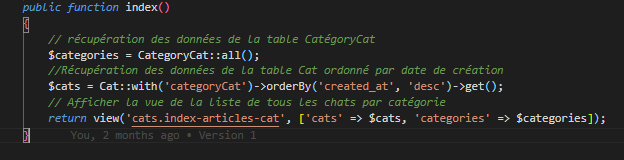
Pour mettre en place les 4 opérations, nous devons utiliser 7 fonctions différentes :

* index : afficher une liste de données
* create : afficher le formulaire pour la création d’une donnée
* store : créer une donnée
* show : afficher une donnée
* edit : afficher le formulaire pour la modification une donnée
* update : modifier une donnée
* delete : supprimer une donnée

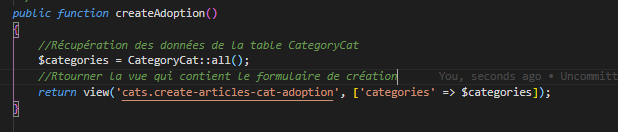
Afin de diriger chaque fonction vers les bonnes vues, nous devons créer 7 routes. Chaque route correspondant à une fonction du contrôleur. J’ai utilisé 2 méthodes la méthode GET et la méthode POST. La méthode GET permet de demander l’affichage d’un formulaire et la méthode POST permet de soumettre le formulaire par exemple.

|  |  |  |  |
| --- | --- | --- | --- |
| Méthodes | URL | Fonction du contrôleur | Nom de la route |
| GET | /chats | index() | cats.index-articles-cat |
| GET | /adoption/chats/creer | createAdoption() | cats.create-articles-cat-adoption |
| POST | /chatsAdoption | storeAdoption() | articles.chats.adoption.store |
| GET | /chats/{cat:slug}/edit | editAdoption() | cats.edit-articles-cat |
| POST | /chats/{cat:slug} | updateAdoption() | cats.update-articles-cat-adoption |
| GET | /chats-a-ladoption/{cat:slug} | showCatAdoption() | cat.show |
| GET | /chats/{cat:slug}/delete | destroy() | cat.delete |

La fonction **index()** va me permettre d’afficher la liste de tous les chats toutes catégories confondues ordonnés par leur date de création.

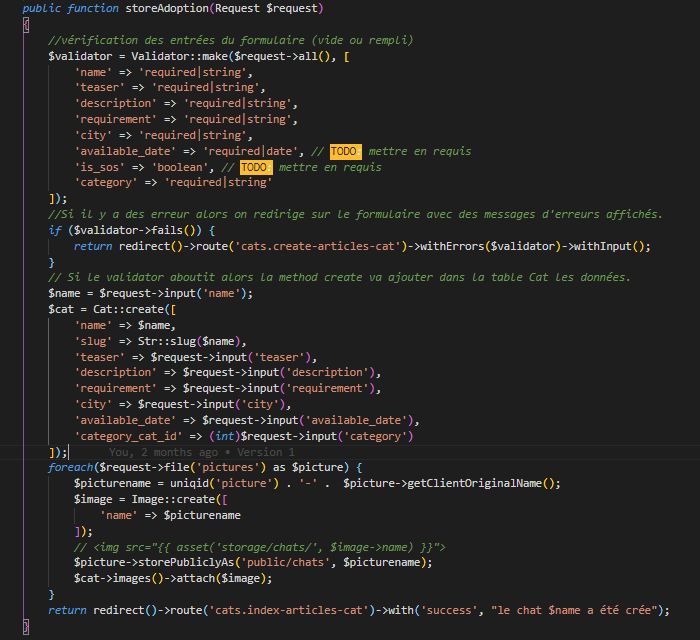


La fonction **createAdoption()** va me permettre dans un 1er temps dans ma situation de récupérer les données de la table « CategoryCat » afin de pouvoir sélectionner la catégorie du chat à créer puis dans un 2eme temps retourner la vue du formulaire de création en me donnant l’accès aux données de la table CategoryCat.

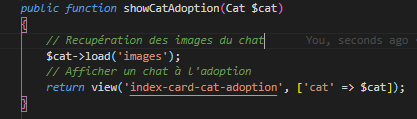


La fonction **storeAdoption()** est divisé en 2 sections.

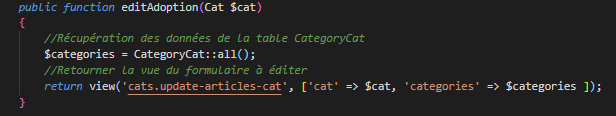
* Section Validator : La facade Validator permet de vérifier toutes les entrées du formulaire, avant de faire la requête de création. Si la validation n’aboutit pas alors le validateur génère et affiche des messages d’erreurs à l’utilisateur.
* Section fonction create() : Les champs sont validés et maintenant la requête d’insertion va s’exécuter. La donnée va être ajoutée à la base de données.



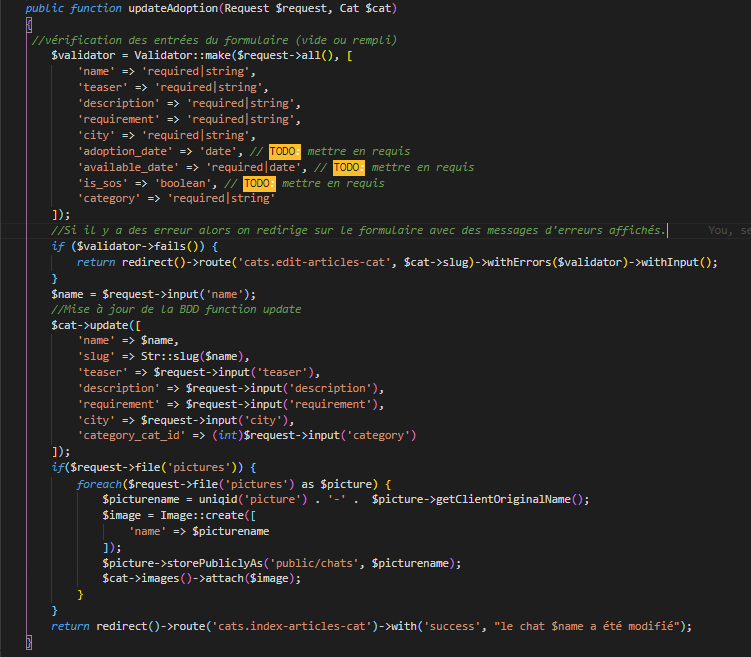
La fonction **showCatAdoption()** permet d’afficher une seule donnée, ici je lui demande de m’afficher un chat à l’adoption. Je charge les images qui sont liées à ce chat, puis je retourne la vue du chat.



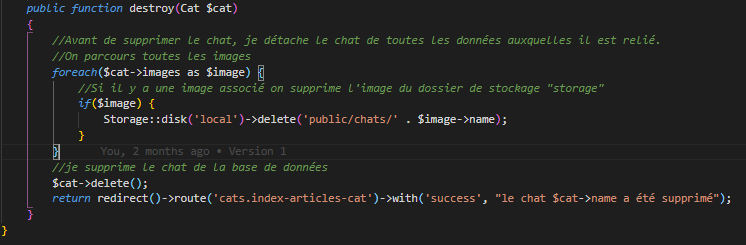
La fonction **editAdoption()** est similaire à la fonction create() puisqu’elle récupère dans un 1er temps dans mon exemple les données de la table CategoryCat puis nous retourne la vue du formulaire à éditer.



La fonction **updateAdoption()** est similaire à la fonction store() puisqu’elle est construite de la même façon avec une 1ere étape de vérification et validation des données renseignées dans le formulaire puis en cas de succès, la mise à jour de la base de données s’effectue avec la méthode update().



La fonction **destroy()** me permet de supprimer une donnée de la base de données. Ici le chat est relié à des données de la table Images. Avant de supprimer le chat, je vais d’abord vérifier si il y a des images reliées au chat, si c’est le cas alors je supprime dans un 1er temps les images puis je supprime le chat avec la fonction delete().

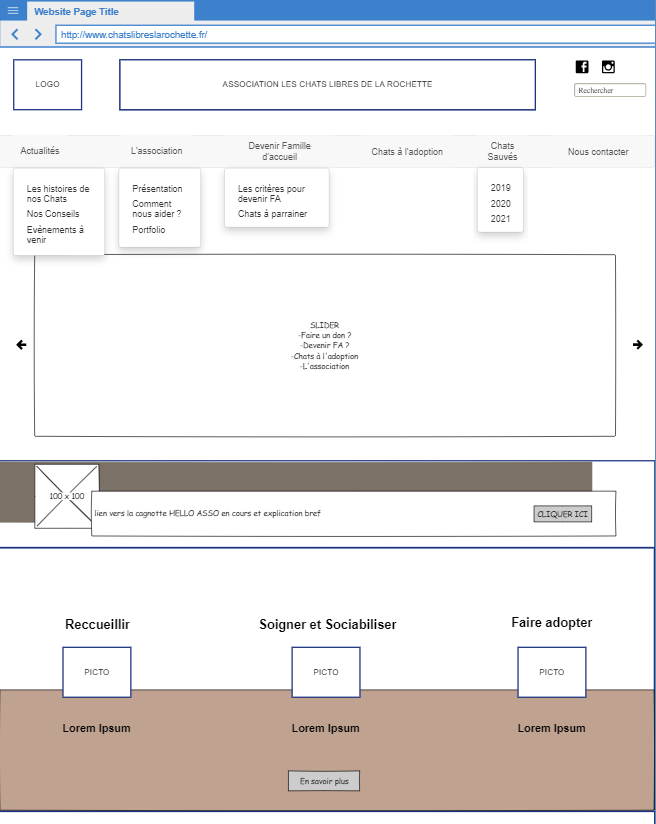


# Conclusion du projet

# Bilan

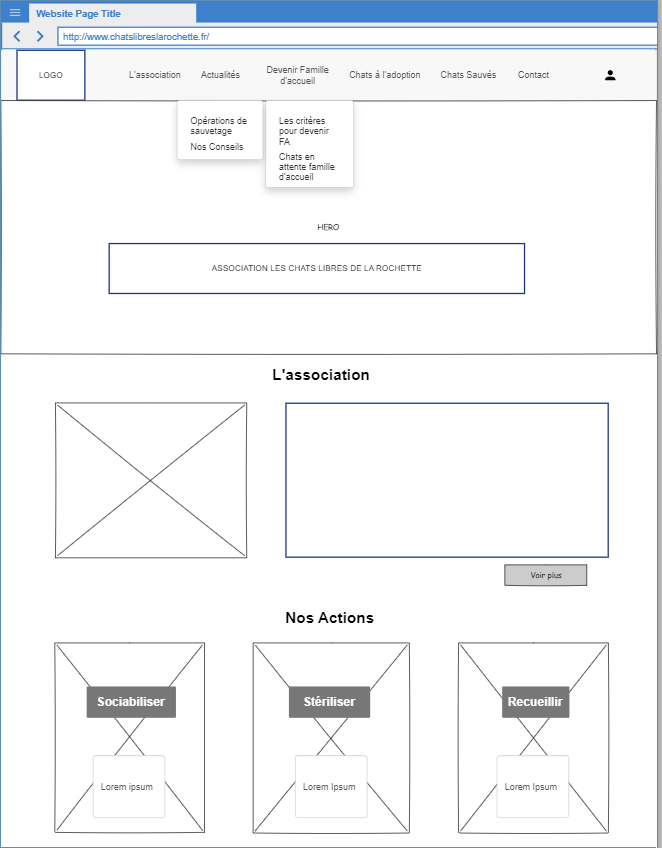
# Annexes

## Maquettes desktop et mobile.

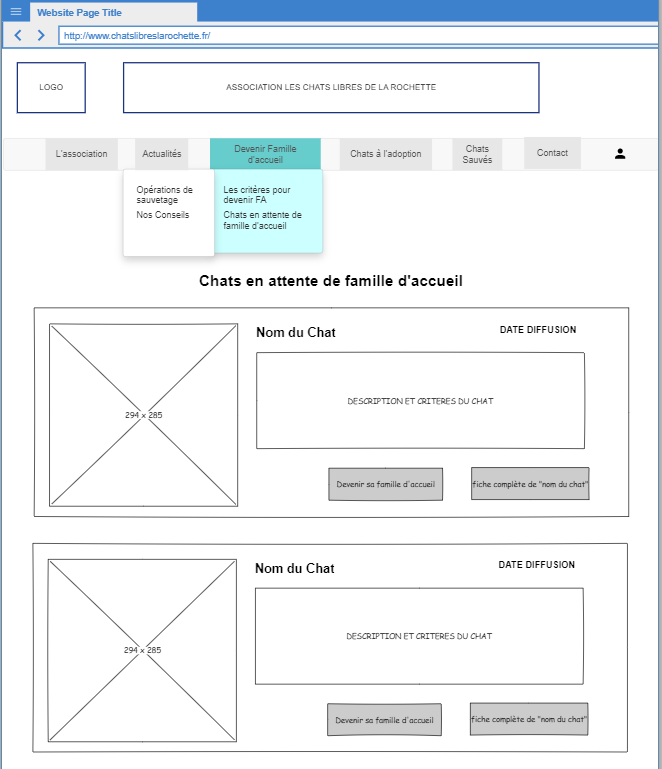
**1ère version de la page d’accueil du site.**

**Version finale de la page d’accueil :**

Après avoir échangé avec l’équipe de l’association nous avons décidé de simplifier la page d’accueil et de mettre uniquement les informations les plus importante et utile pour ne pas perdre l’utilisateur.

Pour une question de rapidité du chargement de la page d’accueil j’ai pris la décision de retirer le carrousel et de plutôt mettre un Hero.

**Page chats en attente de famille d’accueil :**

****Cette page correspond à la liste des chats qui sont en attente de famille d’accueil. Chaque chat possèdera une carte composée d’une photo et d’un texte descriptif. L’utilisateur pourra soit cliquer sur le bouton fiche complète pour accéder à plus d’informations correspondant au chat ou bien sur le bouton devenir sa famille d’accueil pour accéder au formulaire.

## Mobile