Sylvain Torrenti

Dossier projet

Concernant l’EPCF

du 29 juin 2023

A picture containing text, circle, diagram, design

Description automatically generated

# Remerciements

Pour commencer, je souhaite remercier la Maison Départementale des Personnes Handicapées des Bouches du Rhône, Cap Emploi, CT conseil et le centre 2ISA pour m’avoir permis cette reconversion professionnelle.

Je remercie vivement l’équipe pédagogique. Que ce soient les formateurs techniques, **Serge BOISSEAU**, **Théo OLLIVIER-TRIQUET**, **Fabien BELUGOU**, **Philippe PALAU** et **Hakim LASSOUED**, qui m’ont permis de mieux appréhender les différentes problématiques auxquelles j’ai été confrontées mais aussi les formateurs intervenus pour les compétences transverses, **Sarah KINSLEY, Annelies AANTJES, Dylan PEREZ** et **Alexandra TESTART.**

Je souhaite également remercier **Muriel ANDREO** et **Carole GALTIER** pour leur accueil au sein de l’établissement ainsi que tous mes collègues de la formation, qui ont permis un cadre propice à l’apprentissage.

Et enfin, merci à ma famille qui m’a soutenu et encouragé durant cette première étape de ma formation.

Table des matières

[Remerciements 2](#_Toc137537757)

[Liste des compétences couvertes par le projet 5](#_Toc137537758)

[Résumé 6](#_Toc137537759)

[Recherches en langue anglaise 7](#_Toc137537760)

[Introduction 8](#_Toc137537761)

[Prérequis du projet 8](#_Toc137537762)

[Mise en place du projet 9](#_Toc137537763)

[Développement du projet 10](#_Toc137537764)

[Création de la Base De Données 10](#_Toc137537765)

[Laravel 12](#_Toc137537766)

[Les routes 13](#_Toc137537767)

[Les vues 14](#_Toc137537768)

[La Page d’accueil 14](#_Toc137537769)

[Création de compte 15](#_Toc137537770)

[La liste des tickets 16](#_Toc137537771)

[La création des Tickets 17](#_Toc137537772)

[Le détail des tickets 18](#_Toc137537773)

[La création de message 19](#_Toc137537774)

[Le controller 20](#_Toc137537775)

[Le Model 21](#_Toc137537776)

[Les utilisateurs 21](#_Toc137537777)

[La création 21](#_Toc137537778)

[La route 21](#_Toc137537779)

[Le controller 22](#_Toc137537780)

[Le model 22](#_Toc137537781)

[Les vues 23](#_Toc137537782)

[Les Tickets 24](#_Toc137537783)

[La création 24](#_Toc137537784)

[La route 25](#_Toc137537785)

[Le controller 25](#_Toc137537786)

[Le model 26](#_Toc137537787)

[La vue 28](#_Toc137537788)

[L’affichage de tous les tickets 28](#_Toc137537789)

[La route 28](#_Toc137537790)

[Le controller 29](#_Toc137537791)

[Le model 29](#_Toc137537792)

[La vue 31](#_Toc137537793)

[Affichage d’un ticket unique 32](#_Toc137537794)

[La route 32](#_Toc137537795)

[Le controller 32](#_Toc137537796)

[Le model 33](#_Toc137537797)

[La vue 33](#_Toc137537798)

[Tests effectués 35](#_Toc137537799)

[Future évolution 35](#_Toc137537800)

[Conclusion 36](#_Toc137537801)

# Liste des compétences couvertes par le projet

* Maquetter une application.
* Développer des composants d’accès aux données.
* Développer la partie front-end d’une interface utilisateur web.
* Développer la partie back-end d’une interface utilisateur web.
* Concevoir une Base De Données.

# Résumé

Le projet consiste à créer une plateforme qui suit les pannes matériel des utilisateurs. Dans cette optique je commence par créer une maquette en utilisant l’outil FIGMA. Cet outil permet de créer l’enchainement des écrans disponibles en indiquant les liens à suivre et peut se partager en ligne sans contrainte.

Une fois satisfait de la maquette, je peux me lancer dans le conception. J’utilise VS Code comme IDE et Laravel comme Framework.

Ensuite je crée l’architecture du projet en commençant par les vues sans inclure le Back End. Je crée un Template pour que mes pages aient un fond identique avec des composants pour mon Header et mon Footer. Dans mon header, je change la disposition et ce qui est affiché si l’utilisateur est connecté ou non. Je crée les routes qui permettent d’accéder à ces vues et qui permettent un lien entre elles.

J’utilise Fortify pour gérer l’authentification et l’inscription.

En parallèle je crée ma Base De Données en utilisant DBeaver. J’utilise le MVC (Model, Vue, Controller) pour relier ma Base De Données avec mes vues. Je modifie les routes pour passer par un Controller. En cas de besoin de données présentes en Base De Données, j’utilise un Model qui récupère les données et les retourne au Controller qui, quant à lui, permet de les utiliser dans la vue qu’il retourne à l’utilisateur.

Mot Clé : FIGMA, maquette, conception, VS Code, IDE, Laravel, Framework, vue, Template, routes, Fortify, DBeaver, MVC, Controller, Model.

# Recherches en langue anglaise

Lors de la réalisation de mon projet, j’ai effectué différentes recherches dans la documentation de Laravel qui est seulement en anglais.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 1 : Documentation Laravel

Dans la figure 1, nous pouvons voir la marche à suivre pour créer des boucles. Si toutefois les informations trouvées via la documentation ne sont pas suffisantes, effectuer une recherche annexe avec les termes anglais permet de trouver des résultats plus pertinents que si la recherche est effectuée en français.

# Introduction

Durant la première partie de ma formation qualifiante Concepteur Développeur d’Application au sein de l’institut 2ISA, j’ai pu mettre en œuvre mes compétences techniques lors de la création d’un projet pour l’EPCF qui se déroule le 29 juin 2023.

Pour ce projet, qui se déroule en deux étapes importantes, j’ai dû mettre en œuvre une plateforme servant à la maintenance du parc informatique. Pendant la première étape, j’ai dû permettre la création d’utilisateurs qui, eux, peuvent créer des tickets d’incidence. Durant cette première période, je n’ai pas eu à gérer le matériel.

L’association 2ISA a été créé le 23 juin 2010. Elle propose des formations pour les personnes en situation de handicap mais possède également un organisme qui forme toutes personnes possédant le niveau Bac aux métiers du numérique.

# Prérequis du projet

Les prérequis techniques présents dans le projet ont été étudiés avec les différents formateurs techniques. Ils ont été présentés de façon à pouvoir effectuer le travail en continu. Ce projet a permis de mettre en pratique toutes les notions que nous avions abordées depuis le début de la formation.

Pour ce projet, et pour toute la formation, les éléments présents sur le site AMIO-FIT permettent de trouver les informations recherchées qu’elles soient transverses ou techniques.

# Mise en place du projet

Pour commencer je me suis servi de l’outil de maquettage en ligne FIGMA comme le montre la figure 2.

Une image contenant capture d’écran, texte, conception

Description générée automatiquement

Figure 2: Illustration des vues sous FIGMA

Le maquettage m’a permis d’avoir une vision globale de mon projet et ainsi de réfléchir en amont aux problématiques possibles.

# Développement du projet

## Création de la Base De Données

Dans un premier temps, j’ai créé la Base De Données sur laquelle je devais effectuer mon projet. J’ai utilisé DBeaver comme logiciel permettant l’administration et le requêtage de Base De Données. Pour la création des tables j’ai utilisé un Script SQL que j’ai créé (figure 3).

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3 : Exemple de SCRIPT

Comme nous pouvons le voir dans la figure 3, j’ai choisi de ne pas créer les clés étrangères pendant la création des tables pour éviter les éventuels conflits.

J’ai donc, par la suite, utilisé un autre script (figure 4).

A picture containing text, screenshot, font, line

Description automatically generated

Figure 4 : Script Alter Table

Ce script permet de modifier les tables existantes et d’y ajouter les clés étrangères.

Par la suite, pour toutes les requêtes du projet, j’ai d’abord utilisé DBeaver et son outil de Script SQL pour ainsi pouvoir vérifier l’exactitude de la requête. Je l’ai ensuite incorporée dans mon code en y effectuant les modifications nécessaires pour son bon fonctionnement.

Grâce à DBeaver nous pouvons également voir les liens entre nos tables. Comme le montre la figure 5, certaines tables ne sont pas liées à d’autres. Ce sont les tables créées par le Framework Fortify dont je parlerai par la suite.

A screenshot of a computer

Description automatically generated

Figure 5 : Diagramme BDD

La figure 5 nous montre que certaines tables ont été rajoutées et la table « User » a été modifiée. J’ai utilisé Fortify qui est une implémentation back-end à Laravel qui sert à l’authentification. Il n’est pas obligatoire de l’utiliser mais cela a permis de gérer facilement l’authentification. J’ai utilisé seulement la table « users » dans ce projet car les autres n’étaient pas nécessaires.

Il est indispensable de bien vérifier que les éléments nécessaires au bon fonctionnement sont présents. J’ai donc rajouté les colonnes « Prénom » et « Tel » dans la table « users ».

## Laravel

Pour ce projet, j’ai utilisé le Framework Laravel. Ce Framework est open-source et respecte le MCD (Model Vue Controller). Il utilise, entre autres, Blade qui facilite grandement le codage.

Grace à Blade, nous pouvons créer des Template que nous pouvons utiliser dans nos pages pour les rendre homogènes.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 6 : Header

La figure 6 représente le Template que j’ai utilisé sur l’ensemble de mes pages. Toutes mes pages auront cet élément comme Header.

L’affichage change si l’utilisateur est authentifié ou non. L’authentification est vérifiée grâce à la condition « @if (!auth()->guest()) » qui regarde si l’utilisateur est un visiteur. Le « ! » devant « auth() » signifie que nous voulons l’inverse du résultat de la condition.

Il permet de faire en sorte que le titre soit également un lien qui ramène vers la page d’accueil.

Une fois la condition vérifiée, le code « {{ auth()->user()->name }} » permet d’afficher la donnée qui se trouve dans la colonne « name » de la table « users » qui correspond à l’utilisateur authentifié.

## Les routes

Elles sont essentielles au bon fonctionnement de Laravel. Elles permettent de faire le liens entre les vues et d’utiliser les fonctions présentes dans un Controller (que nous verrons par la suite).



Figure 7 : Routes

La route (figure 7) utilise la fonction « displayTickets » présente dans « TicketController » qui permet l’affichage des tickets, le « middleware(‘auth’) » s’assure que l’utilisateur est authentifié. S’il ne l’est pas, il le renvoie sur la page de « Login ». Le code  « Name(‘nomDeLaRoute’) » permet de pouvoir utiliser facilement le nom de la route tout au long de notre code.

Il existe aussi la possibilité de créer des routes paramétrées (figure 8).

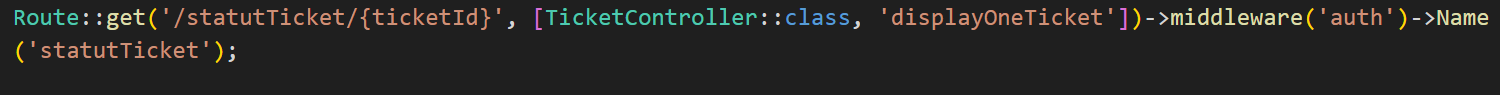


Figure 8 : Route paramétrée

Cette route permet de retourner la vue du détail d’un ticket grâce à la fonction « displayOneTicket » présente dans TicketController (figure 9).

A picture containing text, screenshot

Description automatically generated

Figure 9 : Fonction displayOneTicket

Cette fonction crée des instances de « class » des « model » créées (le détail des modèles sera abordé plus loin) dont elle a besoin pour fonctionner. Ici, tous les éléments nécessaires à l’affichage. Cette fonction fournit toutes ces données à la vue pour qu’elle puisse les utiliser, « ‘ticket’ => $ticket » signifie que dans la vue retournée nous pouvons utiliser « $ticket » pour accéder aux données du ticket.

## Les vues

### La Page d’accueil

La page d’accueil du site créé pour le projet est présentée dans la figure 10.

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Figure 10 : Accueil

Celle-ci vérifie si vous êtes authentifié. Si vous l’êtes, l’affichage n’est pas le même (figure 11).



Figure 11 : Accueil Log

Sur la page d’accueil vous avez la possibilité de vous identifier, si vous avez déjà un compte ou de le créer le cas échéant.

### Création de compte

Une image contenant texte, capture d’écran, Rectangle, Parallèle

Description générée automatiquement

Figure 12 : Création de compte

J’ai indiqué par une astérisque « \* », les champs obligatoires. Dans le code j’ai également indiqué qu’il devait obligatoirement être renseigné :

* L’Email et le Mot de Passe car ils permettent l’authentification et sans eux vous ne pouvez avoir accès à votre compte.
* Le Nom et le Prénom car ils sont nécessaires au bon fonctionnement du projet dans les cas que nous verrons plus tard.
* Un lien vers la page de Login est également présent si vous possédez déjà un compte et que vous êtes arrivé sur cette page par mégarde.

### La liste des tickets

Lorsque vous êtes authentifié avec un compte qui n’est pas administrateur vous visualisez uniquement vos tickets et vous avez également accès à un bouton qui mène vers la page de création du ticket (figure 13).

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Figure 13 : Liste Tickets (Utilisateur)

Dans cette liste, nous pouvons trouver les informations générales concernant les tickets. L’auteur n’y est pas indiqué car cette liste concerne seulement les tickets de l’utilisateur connecté. Nous pouvons également voir que le ticket 2 possède une information supplémentaire concernant la mise à jour. Celle-ci est rajoutée lorsque l’état initial du ticket est modifié (ajout de message, changement de statut).

La figure 14 nous montre la même vue mais provenant d’un compte administrateur.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Figure 14 : Liste Tickets (Administrateur)

Il y a deux différences importantes. La présence du nom et du prénom de l’émetteur du ticket car en tant qu’administrateur nous avons accès à tous les tickets. Ainsi que la disparition du bouton permettant de créer des tickets. J’ai fait le choix de ne pas permettre à un administrateur de créer un ticket.

### La création des Tickets

Une image contenant texte, capture d’écran, Rectangle

Description générée automatiquement

Figure 15 : Création de tickets

Grace à cette vue (figure 15), l’utilisateur peut créer un ticket. Il doit indiquer la matériel à changer (susceptible de changer lors de la 2ème partie du fil rouge ou nous devrons aussi gérer le matériel). Ces informations sont obligatoires car nous les utilisons dans différentes vues. Elles sont enregistrées en Base De Données.

### Le détail des tickets

La figure 16 montre le détail du ticket lorsqu’on est authentifié en tant qu’utilisateur.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 16 : Détail Ticket (Utilisateur)

Sur cette vue nous trouvons les détails du tickets. Un espace est réservé pour l’échange de messages. La date de chaque message est indiquée pour permettre un meilleur suivi. Le nom de l’auteur du message est aussi indiqué.

Un bouton permettant de revenir à la liste des tickets est également présent.

La figure 17 montre le détail du ticket lorsqu’on est authentifié en tant qu’administrateur.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Figure 17 : Détail Ticket (Administrateur)

Il y a certaines différences notables. Nous pouvons voir que les auteurs des messages ont changé pour correspondre au compte authentifié.

En tant qu’administrateur il est également possible de changer le statut du ticket, fonctionnalité qui n’est pas disponible si l’on est un simple utilisateur.

### La création de message

Cette vue ne change pas en fonction du rôle de l’utilisateur (figure 18). Cette fonctionnalité est disponible uniquement en passant par le détail du ticket car un message est lié à un ticket précis.

Une image contenant texte, capture d’écran, Rectangle, cadre photo

Description générée automatiquement

Figure 18 : Message

Dans cet espace vous pouvez indiquer votre message et cliquer sur « envoyer ». Il est ainsi enregistré en Base De Données avec les informations de l’auteur, du ticket concerné et la date d’émission.

## Le controller

En suivant la méthode MCD il faut obligatoirement passer par un Controller. Celui-ci est appelé par la route qui utilise également la fonction créée dans le Controller (figure 19).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 19 : Code d'un Controller

## Le Model

Pour finir, il y a aussi la présence du Model qui sert de lien entre le Controller et la Base De Données (figure 20).

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 20 : code Model

# Les utilisateurs

## La création

Pour créer des utilisateurs j’ai utilisé la méthode MVC (Model, Vues, Controller). Pour plus de facilité, j’ai utilisé l’outil Fortify qui peut s’utiliser avec Laravel. Cet outil permet de gérer l’authentification et la création des utilisateurs.

### La route

Fortify crée automatiquement les routes utilisées par la page de login et d’authentification. Pour changer la cible il nous faut la changer dans le fichier « RouteServiceProvider » (figure 21).



Figure 21 : Route login/auth

Ainsi j’ai pu indiquer que, quand l’utilisateur est authentifié, il est automatiquement redirigé vers la vue « logUser ».

### Le controller

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 22 : Create account

Le controller est créé par Fortify (figure 22). Il comporte toutes les fonctions nécessaires au fonctionnement de Fortify. La fonction de la figure 22, permet de stocker les données d’un utilisateur pour créer un compte. J’ai rajouté les colonnes qui ne sont pas définies de base dans Fortify pour être cohérent avec ma Base De Données.

### Le model

Le modèle est également créé automatiquement par Fortify. Des modifications sont nécessaires pour correspondre à ce que nous voulons. C’est aussi dans ce fichier que j’ai créé ma fonction « isAdmin » qui permet de savoir si l’utilisateur est un administrateur. Je reviendrai sur l’explication de cette fonction plus tard dans ce dossier.

Fortify permet une certaine protection lors de la création de compte. Il faut donc modifier une chose dans ce fichier (fichier 23).

A screen shot of a computer

Description automatically generated with medium confidence

Figure 23 : Fillable

Seuls les éléments présents dans cette liste seront enregistrés en Base De Données. C’est pourquoi j’ai rajouté « Prénom » et « Tel » qui sont les deux colonnes supplémentaires de ma Base De Données qui ne sont pas basiquement des Fortify. Attention, la casse doit être respectée.

### Les vues

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 24 : Vue code createAccount

La figure 24 montre le code utilisé dans la vue de création de compte.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 25 : Vue code login

La figure 25 montre le code utilisé pour gérer l’authentification.

# Les Tickets

## La création

Pour créer des tickets en suivant la méthode MVC il faut trois fichiers différents : le Model, la Vue et le Controller. L’ordre de création n’est pas fixe mais les trois doivent être présents.

### La route

La figure 26 montre la route utilisée quand l’utilisateur crée un ticket via le formulaire

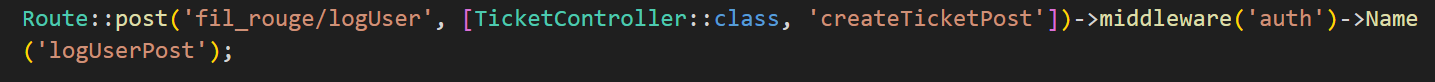


Figure 26 : Route création ticket

Cette route indique que le formulaire doit être utilisé avec une méthode POST. Elle utilise la fonction « createTicketPost » qui se trouve dans « TicketController ». Pour l’utiliser, il faut être authentifié. Je lui ai donné un nom pour faciliter son utilisation dans mon code.

### Le controller

Comme indiqué par notre route, elle utilise la fonction « createTicketPost » se trouvant dans le controller (figure 27).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 27 : Fonction crerateTicketPost

Dans la variable « $rules » nous indiquons les règles à suivre pour les différents éléments demandés. Les règles pour le champ « matériel » et « Sujet » sont :

* Le champ est requis
* Doit être de type string
* Doit faire minimum 3 caractères
* Doit faire maximum 255 caractères.

La variable « $validator » vérifie que les inputs rentrés correspondent aux règles définies. Elle utilise les éléments donnés en input grâce à la variable « $request » et à la variable « $rules ». S’il n’y a pas d’erreur, elle stocke tous les inputs dans une variable « $data ».

Une nouvelle instance de la « class » « TicketModel » est créée pour pouvoir utiliser les fonctions présentes dans le « model ». J’utilise la fonction « insert », avec comme paramètre la variable « $data » qui contient les inputs fournis.

Pour finir, j’ai établi une redirection vers ma route « statutTicket » avec comme paramètre l’« Id » de mon ticket présent dans la variable « $ticketId » et qui permet ainsi d’afficher directement les détails du ticket créé.

### Le model

Pour finir, il y a le « model » : il sert de lien entre la Base De Données et le controller.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 28 : Fonction insert

Cette fonction, présente dans le model, est appelée par le controller lorsque l’utilisateur crée un ticket (figure 28). Elle permet de stocker les données fournies par l’utilisateur dans la Base De Données et prend en paramètre les données récupérées dans le controller.

Les éléments qui constituent cette fonction sont :

**DB ::table(‘Ticket’)->insertGetId :** Signifie que j’ai fait une insertion en récupérant l’ Id  dans la table « Ticket ». J’ai utilisé « insertGetId » car l’Id de la table ticket est auto incrémenté.

**([…]) :** Les points sont remplacés par les données que nous voulons insérer.

**‘Sujet’ => $data[‘Sujet’] :** Signifie que dans la colonne « Sujet » j’insère les données de la clé « Sujet » présentes dans les données récupérées via le controller.

**‘Materiel’ => $data[‘materiel’] :** Signifie que dans la colonne « Materiel » j’insère les données de la clé « materiel » présentes dans les données récupérées via le controller.

**‘Status\_id’ => 1 :** Signifie que lors de la création d’un ticket son statut est défini sur 1 qui correspond à « en cours » dans la table Statuts.

**‘CreatedAt’=> Carbon ::now() :** Permet de récupérer la date lors de la création.

**‘User\_id’=> auth()->user()->id :** Récupère l’ « Id » de l’auteur et l’insère dans la colonne de la table Ticket.

### La vue

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 29 : Vue code createTicket

Pour la création de ticket, la vue (figure 29), indique que les champs « materiel » et « sujet » doivent être renseignés et qu’ils ont une taille maximum à respecter pour convenir à la Base De Données.

## L’affichage de tous les tickets

Comme pour la création, l’affichage nécessite aussi de suivre la méthode MVC, mais nous n’utilisons pas les mêmes fonctions.

### La route



Figure 30 : Route display ticket

Cette route utilise le TicketController pour appeler la fonction « displayTickets » (figure 30).

### Le controller

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 31 : fonction displayTickets

Cette fonction (figure 31) crée et utilise une instance de TicketModel pour récupérer et rendre visibles les données se trouvant en Base De Données et les retourne à la vue avec la possibilité pour celle-ci d’utiliser la variable $tickets et ainsi de pouvoir utiliser les données récupérées.

### Le model

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure 32 : Fonction getAll

Cette fonction contient une condition (figure 32) qui vérifie si l’utilisateur est administrateur ou non. En fonction de la réponse, l’affichage n’est pas le même. Elle permet de récupérer en Base De Données les informations nécessaires à l’affichage. J’ai inclus également la clause « order by » qui permet de choisir l’ordre de sélection en fonction d’un champ de la table. « desc » signifie que l’ordre doit être décroissant.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Figure 33 : Fonction isAdmin

Grace à cette fonction, nous pouvons voir si le rôle de l’utilisateur est « Administrateur » ou non (figure 33). Cette réponse est utilisée dans notre première fonction « getAll ». Si la personne est Administrateur tous les tickets sont récupérés sinon seuls ceux liés à l’utilisateur authentifié le seront.

Dans cette fonction, je demande de sélectionner le Label présent dans la table Rôle lié à l’Id de l’utilisateur authentifié. Pour y arriver, j’utilise plusieurs jointures entre mes tables.

Comme nous avons pu le voir au début du document, la table users n’est pas directement reliée à la table Role. Je dois donc faire plusieurs jointures entre mes tables. Comme je commence par la table Role, je dois la relier à la table UserRole via la clé étrangère IdRole présente dans la table UserRole. Ensuite, je dois relier la table users via la clé étrangère IdUser présente dans la table UserRole. Ainsi mes tables users et Role sont reliées. Dans ma requête, j’indique une clause where qui signifie que je veux uniquement les résultats qui correspondent à cette clause. J’utilise l’Id de l’utilisateur authentifié pour récupérer le Label de son rôle qui se trouve dans la table Role : si ce Label est Administrateur alors la fonctions renvoie « vrai » sinon elle renvoie « faux ».

### La vue

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 34: Code de la vue

Plusieurs éléments sont à noter sur la figure 34 :

**@forelse ($tickets as $ticket)** qui indique que pour chaque $tickets fourni par le controller, ils seront stockés dans une nouvelle variable $ticket.

**@if (auth()->user()->isAdmin())** qui vérifie si la personne est administrateur (comme vu précédemment) et ainsi afficher l’auteur du ticket et enlever la fonctionnalité de créer un ticket comme indiqué au début.

@if ($ticket->label\_status == ‘Résolu’) qui vérifie le label du statut. Si celui-ci est indiqué ‘résolu’ alors le message de mise a jour est remplacé par ‘Le ticket a été fermé le :’.

**@if ($ticket->UpdateAt != null && $ticket->label\_status == ‘En cours’)** qui vérifie si dans la Base De Données le champ UpdateAt n’est pas « null » et que le statut est ‘En cours’ (les deux conditions doivent être vraies en même temp). Si cette condition est vérifiée alors j’affiche le message ‘Dernière mise à jour faite le :’.

**$ticket->…** Les « … » sont remplacés par les noms des colonnes que nous souhaitons afficher.

## Affichage d’un ticket unique

### La route

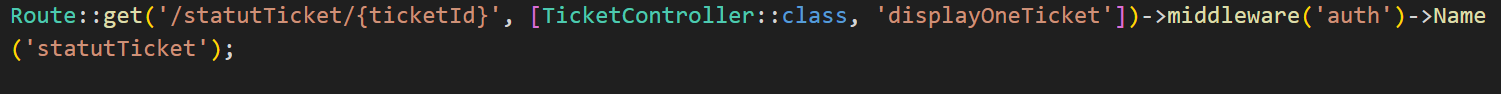


Figure 35: Route display one ticket

Cette route paramétrée est utilisée dans diverses situations (figure 35). Si l’utilisateur veut avoir accès aux détails d’un ticket précis ou quand un ticket est créé, elle prend en paramètre l’Id du ticket et ainsi a accès aux informations qui sont récupérées par la fonction displayOneTicket présente dans le TicketController.

### Le controller

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 36 : Fonction displayoneTicket

Grace à cette fonction (figure 36), nous récupérons toutes les informations nécessaires à notre vue. Nous pouvons voir que nous indiquons, en paramètre de la fonction, l’Id du ticket qui nous sert pour récupérer les informations qui lui sont liées comme les messages ou son statut. Nous utilisons aussi les model que nous avons créés pour les messages, le statut et l’utilisateur.

### Le model

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Figure 37 : Fonction get

Cette fonction (figure 37) permet de faire un « select » de tous les champs, grâce au signe « \* », de la table Ticket avec comme clause l’Id du ticket qui est passé en paramètre. Nous pouvons voir que cette requête utilise « selectOne » et pas simplement « select », cela nous permet de ne récupérer qu’une seule occurrence et non un tableau ce qui est plus logique car nous ne récupérons qu’un seul ticket.

L’écriture « where Id = ? ;’, [$ticketId]);  » permet d’éviter les injections SQL.

### La vue

Je vais vous présenter chaque élément important de la vue.

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 38 : Code vue statutTicket (message)

Dans la figure 38, nous pouvons voir comment les messages sont gérés. Tout d’abord, j’affiche les informations du ticket sélectionné. Puis arrive l’espace réservé aux messages. Si aucun message n’est présent, vérifié par la condition « if ($message == null) », le message affiché est « il n’y a pas encore de message ».

Si des messages sont présents, une autre condition est vérifiée, « if (auth()->user()->id == $message->User\_id) », elle permet de vérifier si l’auteur du message est l’utilisateur authentifié. Dans ce cas, l’auteur du message est remplacé par « Vous » pour permettre une meilleure vision de l’espace de message.

Nous voyons également que la date de création du message est présente ce qui permet un meilleur suivi de la conversation.

Il y a également la présence d’un bouton permettant la création de message utilisant une route paramétrée qui fournit l’Id du ticket en cours et ainsi permettre que le message créé soit lié au ticket lors de sa création. Nous verrons en détail la création des messages plus loin dans ce dossier.

C’est également dans cette vue que je gère si le ticket est résolu ou en cours. Si un administrateur a réglé le statut sur ‘résolu’ alors il est impossible de créer de nouveaux messages.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 39 : Code vue statutTicket (Administrateur)

Dans la figure 39, nous pouvons voir la gestion du rôle de l’utilisateur authentifié. L’auteur du ticket est affiché seulement si l’utilisateur authentifié est un administrateur. Dans ce cas, un bouton permettant de changer le statut du ticket est présent et actif. L’utilisation de ce bouton appelle la fonction « changeStatutTicket » qui récupère le statut du ticket via « Status\_id » et le change. Grâce à la requête utilisée pour afficher le détail du ticket, nous récupérons et affichons le nouveau statut.

# Tests effectués

Pour effectuer les différents tests j’ai fait appel à des personnes extérieures que je souhaite grandement remercier, **BRUNDU Roseline, TORRENTI Natacha et ROBERT Nicolas.**

Ils m’ont permis de mettre en évidence des problématiques que je n’avais pas envisagées et m’ont aussi prodigué des conseils au niveau du design et de l’ergonomie.

Pour que les tests soient le plus fidèles à la réalité, je ne leur ai fait qu’un rapide résumé de la plateforme sans donner d’indication sur la marche à suivre (mis à part qu’ils devaient créer un compte). Grâce à cela, j’ai pu mettre en lumière des dysfonctionnements de quelques fonctionnalités sous certaines conditions qui m’avaient échappé car je connaissais le code.

# Future évolution

Ce projet continuera d’évoluer durant la deuxième partie du Fil Rouge avec l’implémentation de nouvelles fonctionnalités, qui me sont, pour l’heure, inconnues.

Si ce n’est pas prévu, je souhaite mettre en place les services liés aux e-mails (mot de passe oublié, notification etc…).

# Conclusion

En conclusion, ce projet m’a permis de connaitre et d’apprendre à utiliser le Framework Laravel même si je suis conscient que je n’ai fait qu’en effleurer la surface.

Il m’a aussi permis de mieux appréhender les différentes problématiques que l’on pouvait rencontrer. Il me permettra de mieux appréhender la seconde partie du projet fil rouge en me basant sur des bases solides.

Il est également possible que des modifications future y soient apportées. Ces modifications peuvent porté sur la technique mais aussi le design.