*Mon projet Sponsor par Elan Formation*



MA PROJET APPLICATION WEB Symfony

EN PHP



***TA.Walid***

15 rue Wattu - 6700 STRASBOURG

 07 69 34 40 70  waledo.1997@hotmail.com

www.elan-formation.fr

**SOMMAIRE**

[I. Introduction de projet 3](#_Toc78448443)

[II. MCD 3](#_Toc78448444)

[III. Modèle Vue Contrôleur (MVC) 4](#_Toc78448445)

[IV. Des Entity (Des Objet De la classe => viens de la table mcd) 7](#_Toc78448446)

[1- Relation (ASSO) de table 7](#_Toc78448447)

[2- Des Valeure de chaque table 8](#_Toc78448448)

[3- Controller 9](#_Toc78448449)

[5- Base de donnes 15](#_Toc78448450)

[15](#_Toc78448451)

# Introduction de projet

Ce projet permet de insère la voiture, le marque et le modèle par le client qu’il va inscrite déjà pour être connecté sur le site

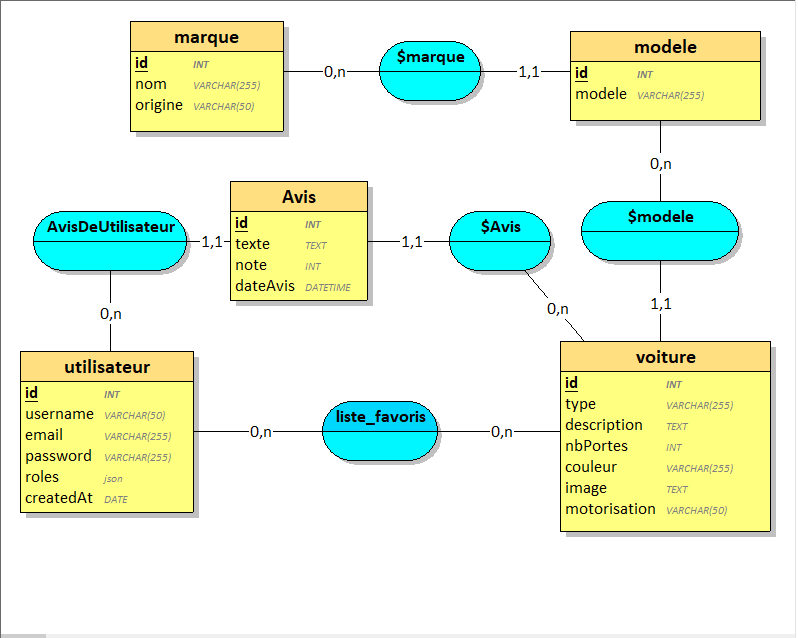
À la suite,

Le site se permette de mettre l’information des voitures avec une avis,

On peut voir les autres voitures en ligne avec son avis aussi.

# MCD

**Ça présent les type des communications entre des valeurs identifient et les déférent valeurs des tableaux et ça clé propre de ASSO qui permet faire des connexions entre eux**.

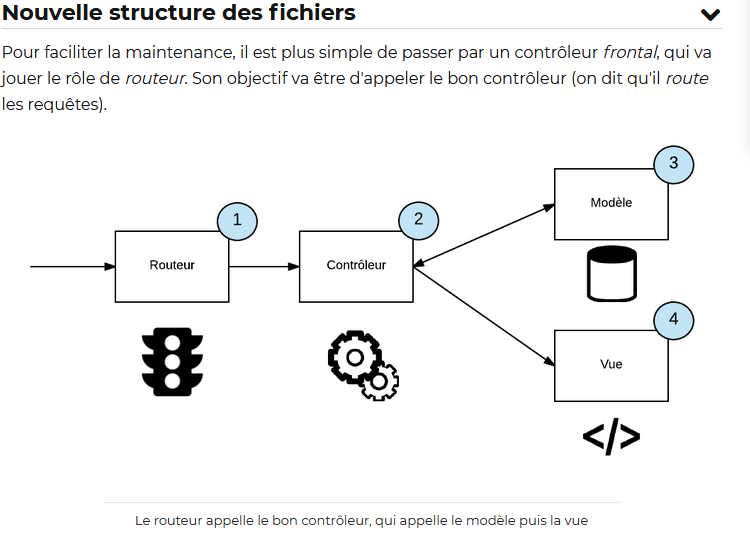


# Modèle Vue Contrôleur (MVC)

**Modèle-vue-contrôleur** ou **MVC** est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

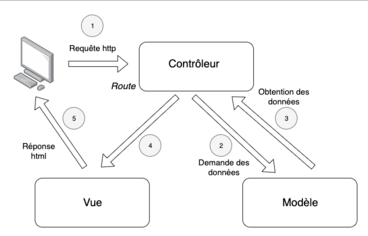
* Un modèle (Model) contient les données à afficher.
* Une vue (View) contient la présentation de l'interface graphique.
* Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.

En utilisant des fonctions de classe, Chaque action des Controller d’une table ou (plusieurs tables) avait un Controller avec des index de chaque un.

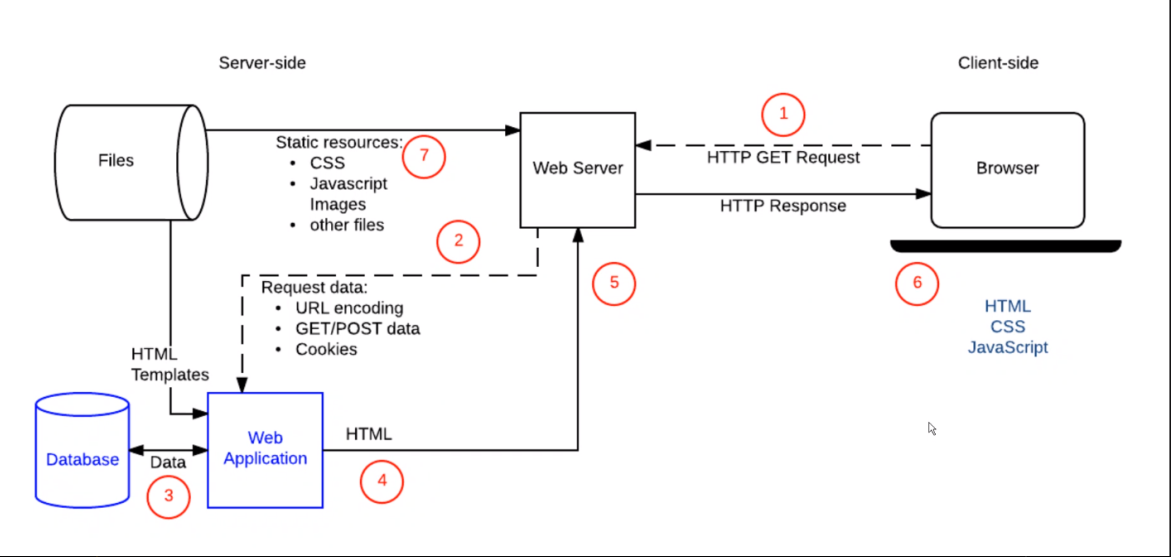


Symfony utilise la méthode de MVC pour crée le projet :

Ce MVC ce présent le chemin d’opération de nos parti et l’ordi parti jusqu’à l’affichage de résultat.



Un Autre Example qui est plus détaillé :

 Ainsi, le moyen de communication entre le client et le serveur (le protocole HTTP) représente en quelque sorte l'intermédiaire, l'interprète des deux parties lors de leurs échanges respectifs.

Dans l'exemple ci-dessus, le client soumet une requête au serveur qui se compose de plusieurs éléments présents dans l'URL : http://monsite.com/liste.php?page=2

**Le protocole employé Le nom de domaine du La ressource (fichier) Un paramètre de requête**

### pour la communication serveur sur le réseau Internet demandée "page" avec pour valeur = 2

Donc on va utiliser ce type de schéma par default de Symfony

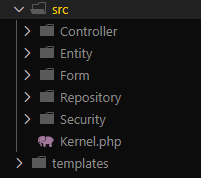
Ou moment de faire d’action le Controller viens de faire le routage au chemin d’accès de chaque fonction qui va présenter avec cette manier en haut dans le navigateur (URL).

Et voila c’est le mvc par default :

Entity ,form , repository remplace le ( MODELE )

Controller c’est Le Controller

Templates remplace le (VEU)



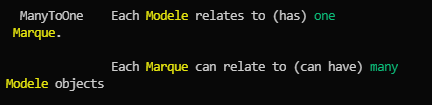
# Des Entity (Des Objet De la classe => viens de la table mcd)

**CONSOLE INTEGREE A SYMFONY:**



## 1- Relation (ASSO) de table

**Dans Class Marque Il y a OneToMany $modeles , et ( dans class Modele il y a ManyToOne $marque ) =>**



**Dans Class Modele Il y a OneToMany $voiteures , ( et dans class Voiteur il y a ManyToOne $modele ) =>**



**Dans Class Voiture Il y a OneToMany $Avis, (et dans class Avis il y a ManyToOne $voiture) =>**



**Dans Class User Il y a OneToMany $AvisDeUtilisatures, (et dans class Avis il y a ManyToOne $user) =>**



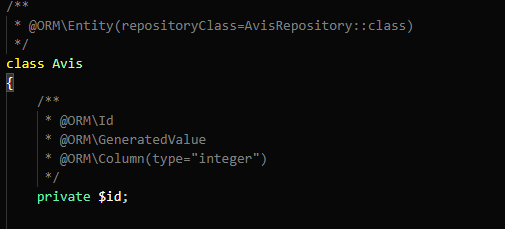
**Dans Class User Il y a ManyToMany $voitures, (et dans class Voiture il y a ManyToMany $users)**

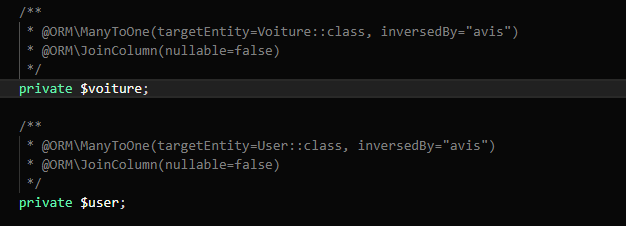
**Je vais appeler le Relation $listFavoirs comme dans le mcd =>**



## Des Valeure de chaque table

Example Class avis :





Avec la commande php/bin make :entity : il permet de faire le entity avec sa repository

Comme est présenter en haut chaque classe avait sa Respositry

Et sa id par default crée par Symfony.

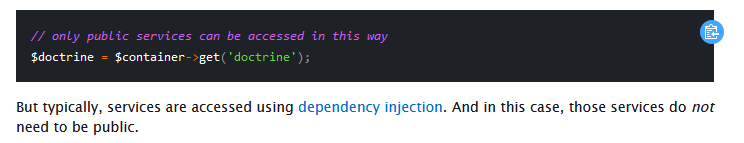
Chaque classe avait plusieurs entity qui ont créé par nous comme on veut

Et je vais mentionne plus de détaille sur les entity utiliseés,

Au moment de présenter le base SQl

Avant passe au controller ici en petite example  (( private $valeur ))

« When defining a service, it can be made to be public or private. If a service is public, it means that you can access it directly from the container at runtime. For example, the doctrine service is a public service:



Et avec la valeur de la classe il n’y a pas besoin de être public alors c’est créé par default en mode service private par la command de Symfony,

Private $user cest une valeur ManyToOne de la classe avis comme on a expliqué avant et cette valeur ne peut pas être Nullable (null = vide).

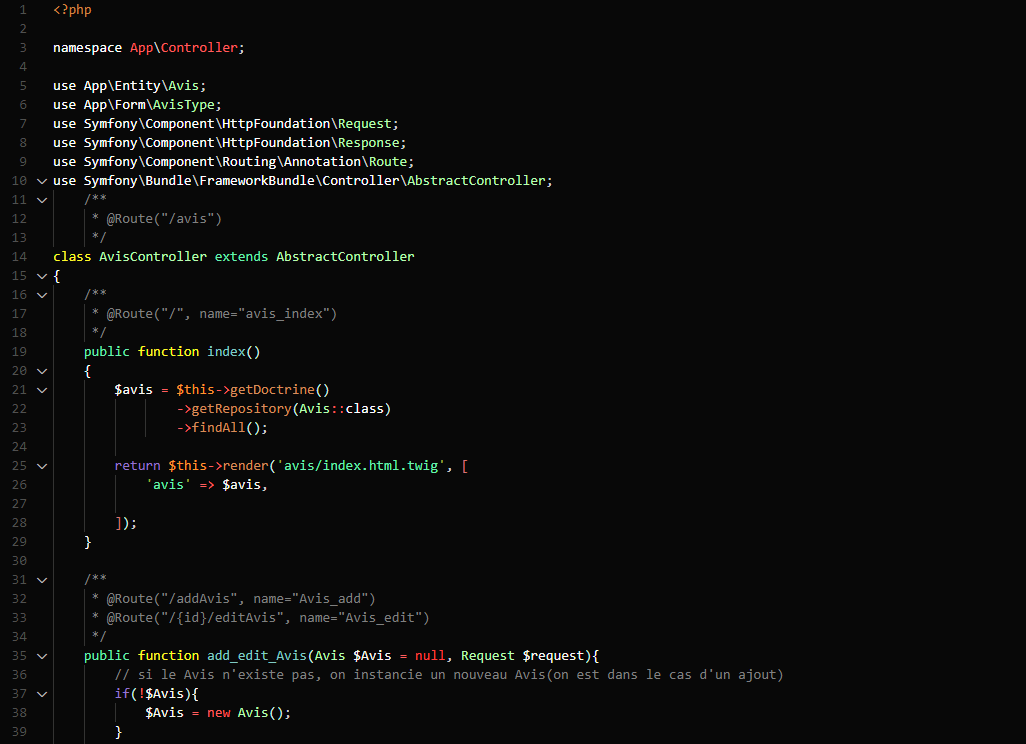
## 3- Controller

Example AvisController : comme est-elle écrite c’est importé que le $valeur écrite en

Minuscule et le Nom de Controller la première lettre en Majuscule

Et la première lettre du deuxième mot c’est aussi en Majuscule

Même le Nom de Controller est 1er lettre Maj et le reste de les lettres minuscule



*  Ligne 1 c’est le commande PHP: de ici la code est en PHP.
*  Ligne 3 c’est le NameSpace: Dans la définition la plus large, les espaces de noms sont un moyen d'encapsuler des éléments. Cela peut être considéré comme un concept abstrait dans de nombreux endroits. Par exemple, dans n'importe quel système d'exploitation, les répertoires servent à regrouper les fichiers liés et servent d'espace de noms pour les fichiers qu'ils contiennent. A titre d'exemple concret, le fichier foo.txt peut exister à la fois dans le répertoire /home/greg et dans /home/other,

Et ici App/Controller.

*  Ligne 5,6 use de package Symfony: on utilise App/Entity/Avis, App/Form/AvisType
*  Ligne 7,8,9 use de package Symfony: composent de symfony HttpFondation

: Response, Request, Route.

*  Ligne 10 use package Symfony: AbstactController qui permet faire le sécurité

Et il y a plusieurs Fonction dedans.

* De  Ligne 16 au 18 : Routage dans ce example /avis.
* De  Ligne 19 : Fonction index de AvisController.
* De  Ligne 21 au 26 : un var avis qui est égal aller utiliser le service Symfony de AbstractController qui permet de manipuler le doctrine : getdoctrine() : donne-moi le tout le doctrine suite en passe une autre méthode de service AbstractController qui avait une méthode getRespotory() : donne-moi le respotory de la classe avis et trouver tout, Route a la Template avis/ qui avait un indix.html.twig donc le route sera avis/index.html.twig et après en déclare le route avis qui est égal a la avis (de trouver tout) que on a déclaré déjà .
* De  Ligne 35 : Fonction add\_edit\_avis qui addition les avis et le modifie.

Un autre Example des fonctions utiliser pour crée le site :



*  Ligne 42 au 43: /addMarque : c’est le Manier de trouver le fonction par le navigateur donc c’est une parti de routage , Marque\_add c’est le Manier de trouver quelle fonction est-il demande par le navigateur ici il y a deux un c’est Marque\_add et un est Marque\_edit Alors cest au moment un tab aller au formollaire qui permet si on ajoute le marque Marque\_add est applleé et si on modifie le marque Marque\_edit et bien sur au moment on utilise le modification on a besoin de connaitre le quelle marque au base de donne on parle et ici c’est le job de les ID qui sont créé par default et utilise comme est-il en haut au routage pour savoir le quelle entity on parle donc

/{id}/editMarque qui est-il affiché au URL comme on a expliquée en avant.

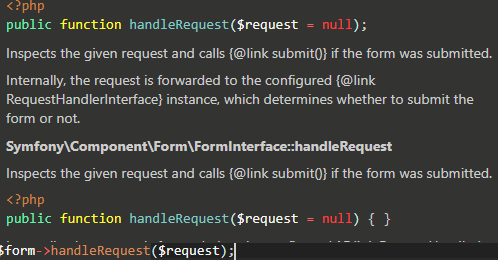
* De  Ligne 46 : Fonction add\_edit\_Marque c’est le nom de fonction qui addition les marque et le modifie, On utilise le classe Marque et on mit une valeur $marque pour mettre le donne dedans au mettre le en Null donc est-il vide et on utilise ici le méthode Request qui vient du service de AbstractController qui est créé par Symfony et cree valeur $request Null (vide).
* De  Ligne 48 au 50 : c’est un langue PHP un méthode if qui permet dire si une condition (est vrai ou false ça débande l’utilisation) on faire une action et ici

Si le Maque n’existe pas il faut crée un en utilise la classe marque pour les donnes qui est dedans le entity marque et le mettre aux valeurs.

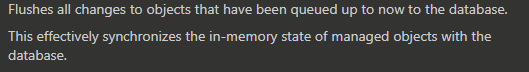
* De  Ligne 52 au 54 : il faut créer un Marque au préalable avec la fonction de service symfony en utilisant le commande : php bin/console make form qui permet faire cette forme alors $form est un valeur crée par nos pour mettre les donnes dedans $this est un référence de le objet courant donc this objet on le prends suit on cree le form avec createForm qui avait deux paramaitre 1- un classe de MaqueType en utilisant en commande de symfony
* php bin/console make:form

Alors ma forme de Marque qui Jai crée déjà est appelle MarqueType qui Avit des donnes avec chaque type le donne par default. 2- le valeur de marque et c’est appelé ici $Marque

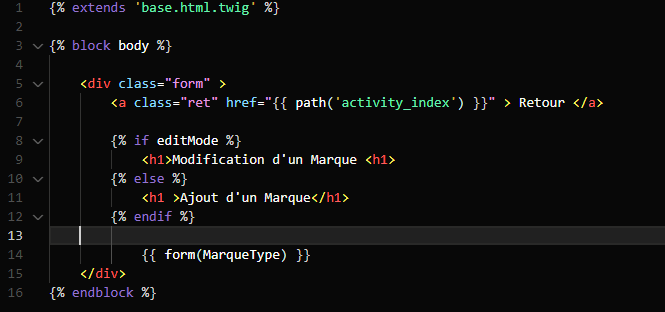
Et line 54 :



En crée un valeur $form qui est Null (vide) et on mit le $request dedans en utilisant des donnes de la fonction handleRequest qui avit un paramètre $request dedans donc handleRequest permit voir et inspictier le lien submit et vois si la form est-il submit et mettre le réponse (ture ou false =>vrai ou faux) dans le paramètre de handleRequest et on prend le donnes de ce paramètre et on le mis dans notre valeur $request et comme ça on voir si notre donnes $form est-il submit au pas , bien sûr le handleRequst est un fonction écrite déjà par le service Symfony dans AbstractController et utilise ici.

* De  Ligne 56 au 66 : un condition simple If qui permit de dire si on soumet le formulaire et que le forme est valide isSubmitted un fonction de Symfony qui permit voir le si est submit alors le résultat est true(vrai) et avec le donne de forme $form qui avait déjà just avant le résultat de HandleSubmit il compare et si est true le condition est vrai il passe comme ici : ( true && qqc ) && permit dire ( And ) alors c’est (et) en français , en suit on a déjà un vrai , Il nos manque le résultat de la deuxième condition isValid aussi un fonction de Symfony de AbstactController qui vérifie si le forme est valide avec le même type entre le entity et sql (base de donne) et nous donne un réponse true vrai ou false non pas valide et bien sûr ici c’est true alors ça va nous donner deux condition ( vrai && vrai ) et avec le crocher c’est le dépôt de notre if donc si le deux sont vrai en applique le condition .
* De  Ligne 58 : on récuprère les données du formulaire avec la fonction getData() et on le mettre dans le variable $Marque
* De  Ligne 60 au 62 : on ajoute le nouveau Marque en manipulation de doctrine alors getDoctrine() un fonction qui permet faire un shortcuts (chemin d’access court) au service de régénération doctrine et getManager() permet de prends un objet qui avait un nom et on mis ce objet dans $entityManger et on prend ce valeur et le faire une mangement et persister après comme,
* Les trois images en dessus après le persistent on le fluch avec un fonction flush
* 
* 
* //on ajoute le nouveau Marque
* $entityManager = $this->getDoctrine()->getManager();
* $entityManager->persist($Marque);
* $entityManager->flush();
* Et comme ça notre nouvelle marque est bien ajoutée aux bases de donnes alors après click valider dans le formulaire tous ces opérations sont appliquées et en suit bien sure on a besoin de redirect le client dans un page home ou un page accueil etc...
* Donc le  Ligne 64 : $this->redirectToRoute() avec ce fonction qui redirect le client au nom de la Rout existé entre ( ) et bien sure on mit le nom entre gimé comme ("home")
* Et voilà la fin de notre if
* De  Ligne 66 au 70 : Notre fonction doit return qqc car toute la fonction dois retourner qqc alors au moment de appeler le fonction on rendre le client soit au ajouter le nouveau marque soit on le modifie donc ces lien c’est un redirection au Template qui est la vue de formulaire et prends le marque permettent de voir
* 
* Ça ID en avance est est-il Null (vide -> existe) ou pa.
* Et voilà dans ce cas c’est au Template qui existe au activity et la page c’est add\_edit\_Marque.html.twig.

## Template code



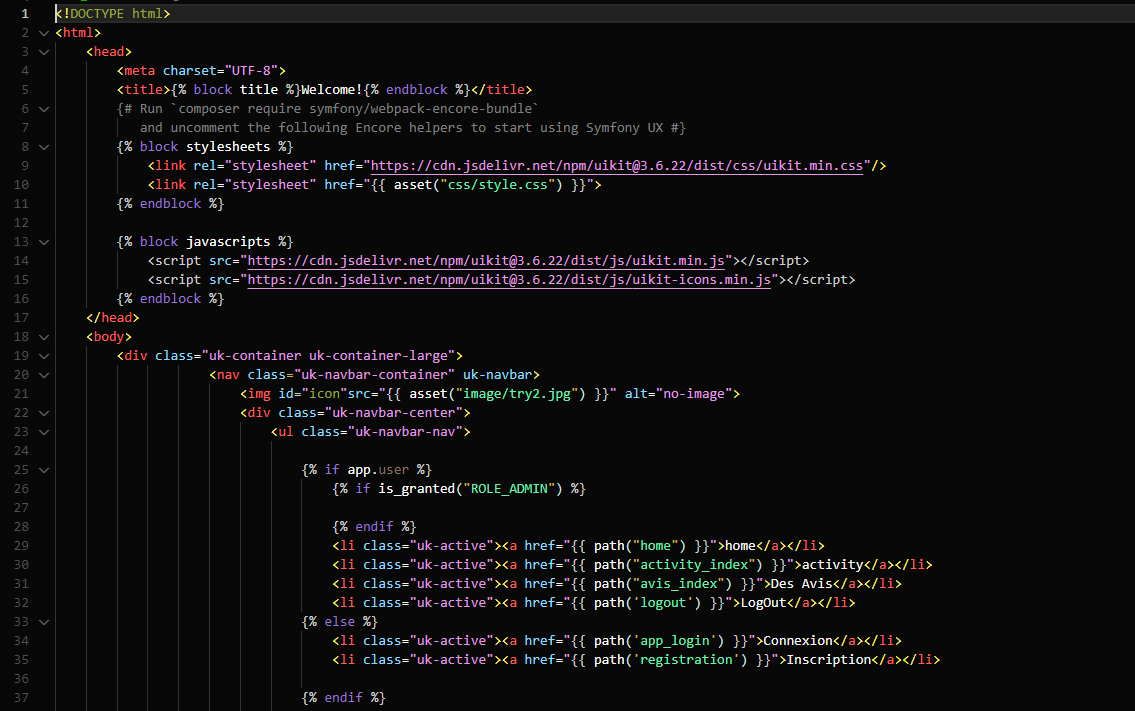
*  Ligne 1: on extends (mis un page dans un autre page donc le page de mère avait un page qui est dans le body) de la page de: base.html.twig
*  Ligne 2: le dépôt de la block
*  Ligne 5: un div -> div: c’est un manier de mettre un block (qaré) qui peut avait un id ou un classe donc le défection sera un élément de [HTML](https://fr.wikipedia.org/wiki/Hypertext_Markup_Language) qui crée un niveau logique.
*  Ligne 6: le balise (L'élément) HTML <a> avec son attribut href, crée un lien hypertexte vers des pages Web, des fichiers, des adresses e-mail, des emplacements dans autre pages etc...

Path() est un méthode utilise pour le page de Twig pour écrire le nom de chemin d’access .

Class : est un manier de appelle des balises défièrent uniquement ou en plusieurs.

*  Ligne 8: le dépôt de if en langue html.Twig
*  Ligne 9: le balise <h1> c’est un header de premier niveau.
*  Ligne 10: un else de if faut dire si non.
*  Ligne 12: le fin de if
*  Ligne 14: la place de forme
*  Ligne 15: le fin de la balise div
*  Ligne 16: le fin de la block

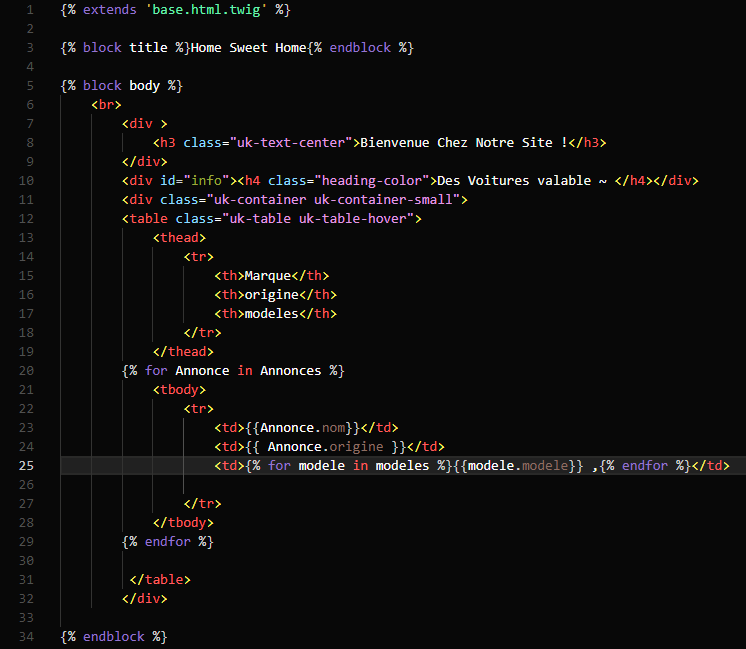
On va à la page de base :

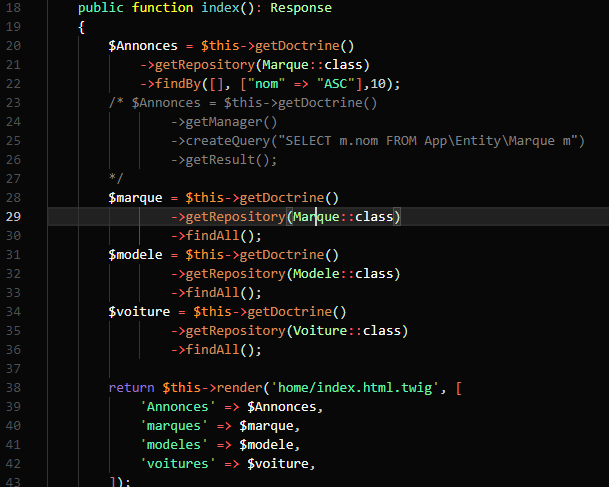


*  Ligne 1: la page est définit en html comme un langue utilisé.
*  Ligne 2: la balise html commence ici.
*  Ligne 3: L'élément <head> est un conteneur de métadonnées (données sur les données) et est placé entre la balise <html> et la balise <body>. ... Les métadonnées définissent généralement le titre du document, le jeu de caractères, les styles, les scripts et d'autres méta-informations.
*  Ligne 4: on utilise le meta UTF-8 ici
*  Ligne 5: définition de titile comme -> Welcome!
*  Ligne 8: le dépôt de block de css de style
*  Ligne 9: Ukit c’est un style utiliser par nous qui est écrite par le server Ukit
*  Ligne 10: notre ficher de style
*  Ligne 11: le fin de block de css de style
*  Ligne 13 et 16: le dépôt et la fin de block de Javascript
* Appliquée par nous ecrit par Ukit.
*  Ligne 14 et 15: le liens vers le script de Ukit.
*  Ligne 17: la fin de balsie head.
*  Ligne 18: le dépôt de balise body.
*  Ligne 19: le dépôt de balise div qui avait un style de ukit
*  Ligne 20: le dépôt de balise de navigation donc un list de navigation est commencer
* Styler par le Ukit
*  Ligne 21: un balise image qui est présenter dans le nav.
*  Ligne 22: deuxième balise de div qui est centre par le style de Ukit.
*  Ligne 23: le dépôt de balise UL <ul> permit faire un list.
*  Ligne 25 et 26: condition de if qui dit si le user (utilisateur) est-il un admin ici le condition sera a rien car dans mon site le admin et le utilisateur normal , font qu’ils veulent car c’est un site de avis a la fin J’ai mis le condition pour savoir que il existe ce type de chose dans le codage on peut mettre le site sécuriser et garantie pour l’admin uniquement.
*  Ligne 29: le premier élément de la liste <ul>.
* Chaque élément va avoir un lien bien sur car c’est un navigateur.
*  Ligne 29 a 35: ce sont les élément de la liste.
*  Ligne 35 a 40: on ferme tous le balise utilise.

Et voilà après cette page je veux vous montrez le page de index c’est un Template aussi mais pour afficher les donnes de la base de donnes.

Ex :



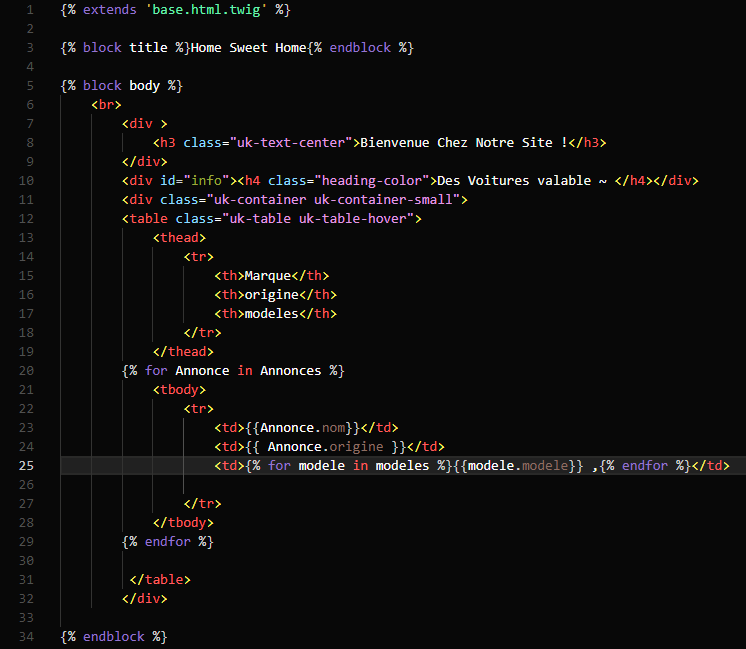
*  ce page est appeler index.html.twig est-il dans un dossier home
* Le dossier home avait sa propre controller qui avait un fonction index qui affiche le 10 dernière voiture ajouter. En utilisant la valeur ici c’est $Annonces
* 

En utilisant les trois tables on peut récupère toutes les infos de base de donnes.

Alors je vais présenter le page index en détail,

Avec la connexion entre le Controller et de la vue

Allons-y : ~ ~



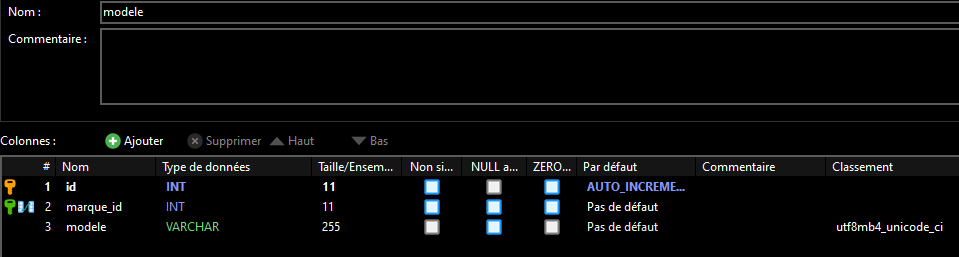
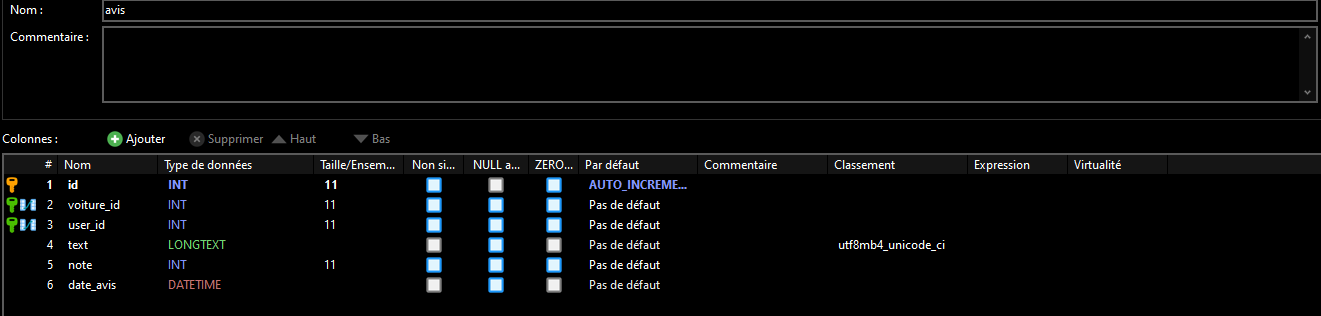
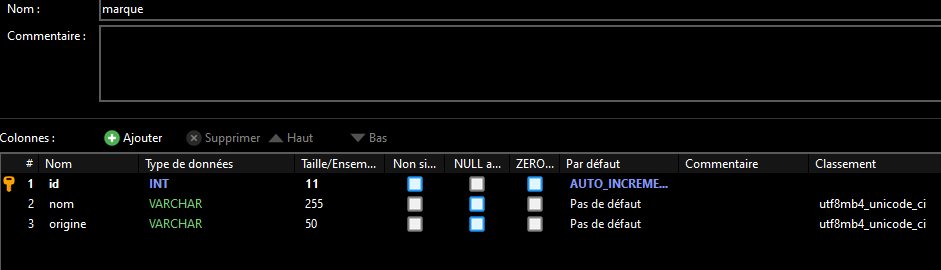
*  Ligne 1 on extends (mis un page dans un autre page donc le page de mère avait un page qui est dans le body) de la page de: base.html.twig.
*  Ligne 3: titre.
*  Ligne 5: dépôt body.
*  Ligne 6: <br> c’est un balise de une line vide
*  Ligne 7 et 9: dépôt et fin la balise div.
*  Ligne 10: div qui contient un header niveau 4 et le div est style par un id appelle info
*  Ligne 11: div qui est style comme un container (div mère) Par ukit.
*  Ligne 12: dépôt de balise table stylée par le ukit.
*  de Ligne 13 au 19: dépôt balise thead qui est le tète de table.
*  Ligne 14 et 18: dépôt balise tr qui permet défini la colonne de table.
*  Ligne 20: un for (boucle) qui permet boucles sur les id de valeur $Annonces
* Et la valeur vient de HomeController au page 16 en haut qui est dans le index et permet avoir un contenu (des donnes->ici sont des marques) les 10 dernières voitures en ordre Alphapitique (ASC) et return ces donnes dans un valeur appelle Annonces été on utilise Annonces comme un valeur max de la boucle et on boucle là-dessus pour obtenir une valeur que on a créé Annonce <- singulaire de numéro 1 de 10 ici dans cet exemple
*  Ligne 21 et 28: dépôt balise tbody qui permit dire que c’est le body de table
*  Ligne 23 et 24 et 25: dépôt balise td qui permit dire la colonne de body.

 exp : Ligne 23 Annonce.nom -> nom cest le nom qui est defini dans le entity de la classe marque .

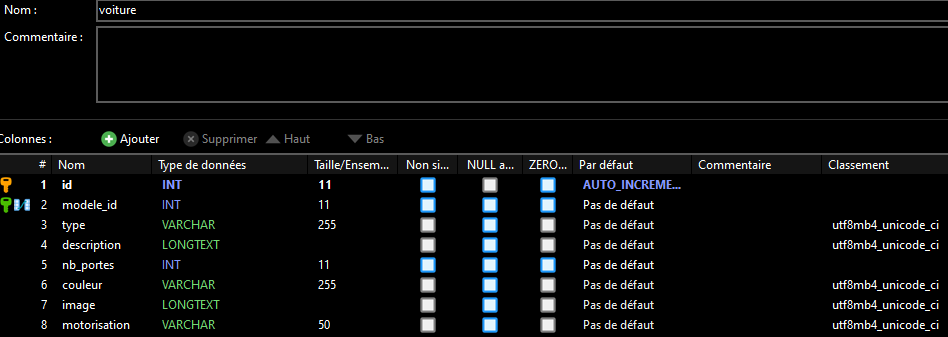
*  Ligne 25: deuxième boucle sur les modeles et voilà comme Annonces modeles vient la fin de la entity de la classe modele et modele.modele permit dire le modele de la voiture et ici le modele des 10 dernier voiture ajouter dans les ordre Alphapitique (ASC).
* Ligne 29 et 34 : la fin de for, la fin de block body.

## 5- Base de donnes

La 2 capture suivante présente HeidiSQL qui est notre base des données de coute serveur :

* Les paramètres****

## 

****