



Stéphane

Hilaricus

Rapport de stage

Sommaire:

1. Introduction
2. Présentation de l'organisation
3. Présentation du service
4. Présentation de l'environnement réseaux et application
5. Description des missions

Introduction :

J'ai effectué un stage de 7 semaines, du 3 janvier au 18 février dans le service de la DSI de l'UTC de Compiègne

J'ai choisi de faire mon stage dans ce service pour plusieurs raisons :

- Pour en savoir plus dans le domaine de l'informatique,
- Acquérir de nouvelles connaissances,
- Parce que ce stage correspond à mon projet professionnel.

Ce stage m'a permis de faire la différence entre le monde du travail et mes études actuelles, il m'a aussi offert la possibilité de découvrir un aspect supplémentaire dans le domaine de l'informatique.

Mes horaires de stage étaient : 9h-12h30, 13h30-17h.

D'une manière générale, ils m'ont fait fortement confiance

J'ai particulièrement apprécié l'atmosphère qui était très conviviale,

L'équipe m'a donné du temps pour m'adapter au travail et ma tutrice et les techniciens ont répondu toujours avec plaisir à mes questions.

Il m'a fallu environ une semaine pour avoir le sentiment de faire partie de ce service.

J'ai beaucoup apprécié ce stage, tout le monde a été très gentil avec moi. Ma tutrice m'a apporté toute son aide possible pour mener à bien le projet et m'a aussi éclairée dans des situations que je ne comprenais pas.

Présentation de l'organisation :

L'organisation est une université technologique, il y a différents bâtiments dans Compiègne. J'ai effectué mon stage à UTC Centre de recherche Rue du Dr Schweitzer 60200 Compiègne.

À la fois université et école d'ingénieurs, l'UTC est construite sur une pédagogie de l'autonomie et une recherche technologique interdisciplinaire orientée vers l'innovation.

L'UTC forme des ingénieurs, masters et docteurs aptes à appréhender les interactions de la technologie avec l'homme et la société, et à évoluer dans un environnement concurrentiel mondial, dans un souci de développement durable. Les enseignants-chercheurs et ingénieurs de l'UTC donnent un sens à l'innovation, en permettant l'émergence de nouveaux axes à ce concept et en introduisant l'entrepreneuriat au cœur de leurs préoccupations.

Née de l'union entre un ingénieur informaticien et un diplômé d'HEC, la société AT Celtiweb est présente sur le marché des nouvelles technologies depuis 2006.

L'UTC, établissement public à caractère scientifique, culturel et professionnel, a été créée en 1972 pour être une université expérimentale de technologie.

L'UTC présente un modèle de formation où :

les sciences de l'ingénieur,

les sciences humaines et sociales,

les sciences économiques et politiques

sont intégrées harmonieusement au service de l'éducation de l'ingénieur, du scientifique, du manager du futur, innovant, humaniste, apte à maîtriser les enjeux de la complexité dans une société de l'information et de la communication.

L'UTC est une institution qui produit du sens pour nos sociétés, dans lesquelles peuvent dialoguer différentes cultures et différents modes d'appréhension du monde.

Fondateur UTC Compiègne :

Guy Deniélou, né le 14 juin 1923 à Toulon et mort le 12 décembre 2008 à Ermont, est un officier de la marine, ingénieur chercheur au CEA, qui fut président-fondateur de l'université de technologie de Compiègne (UTC)

Après des études secondaires classiques, les études de Guy Deniélou sont considérablement troublées par la guerre.



Sa préparation à l'école navale est interrompue le 15 juin 1940 où, le jour de ses 17 ans, il s'engage dans la Marine comme matelot de seconde classe pour gagner l'Angleterre ou Dakar. Cette équipée est interrompue à la bataille des Mers Aïd-el-Kébir le 3 juillet 1940 qu'il vit à bord du Commandant Test Rentré en France via Bizerte, il prépare à nouveau l'École où il est reçu en juin 1941

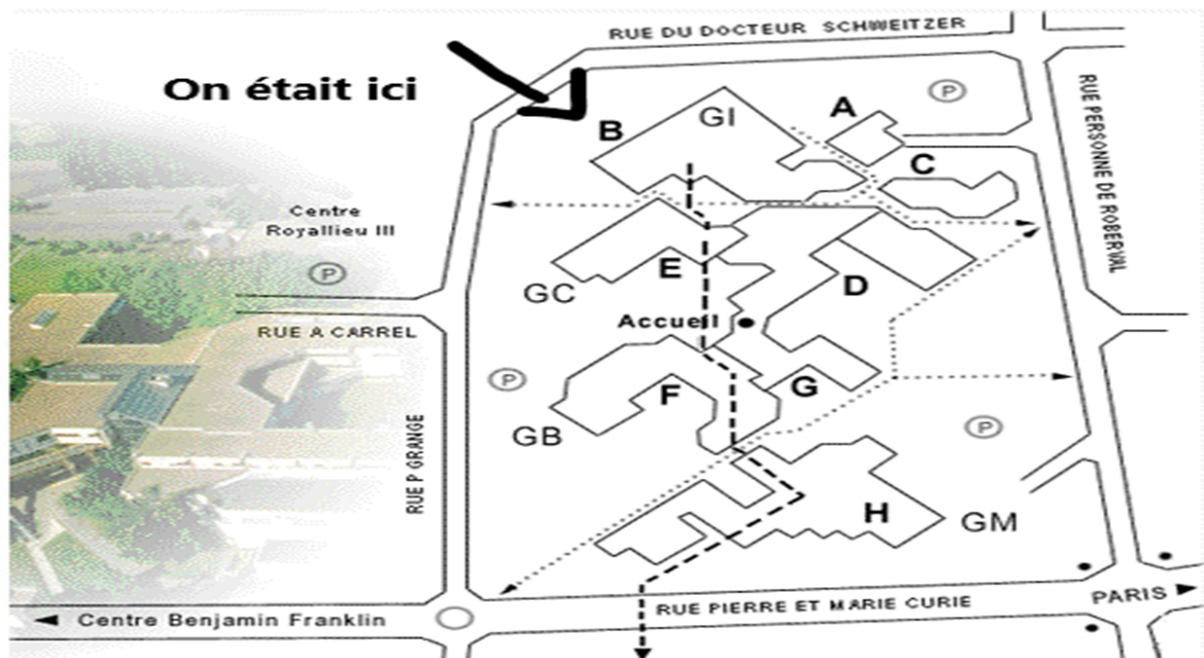
Guy Deniérou, vu son dossier, est désigné pour entrer à l'École polytechnique. Il doit alors passer une année à se perfectionner dans quelques matières au lycée de Parc. Les négociations entre l'armée et l'école polytechnique ayant échoué, Guy Deniérou choisit d'aller à Science Po où il entre en deuxième année. Sa scolarité sera interrompue par là .
Libération de Paris

Nommé chef du Corps Franc Marine, dans la région de Boulogne, Guy Deniérou participe en septembre 1944 au siège de Dunkerque dans l'armée britannique et l'armée tchécoslovaque. Il est ensuite nommé sur un dragueur de mines à La Rochelle en 1945.

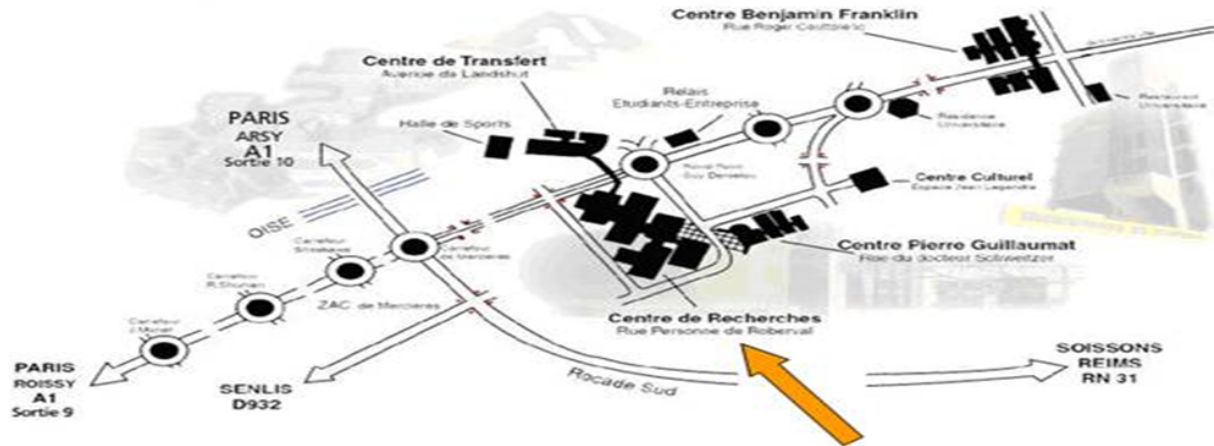
En 1946 et 1947 il participe à la guerre d'Indochine, en particulier au débarquement d'Hai Phong.

Guy Deniérou est recruté par le CEA en 1959 pour mener les études du réacteur Siloé au CEN de Grenoble. En même temps, il est chargé par Louis Néel de l'enseignement de neutronique et théorie des réacteurs à l'institut polytechnique de Grenoble

Le centre de recherche de Compiègne UTC se divise en plusieurs bâtiment



J'ai travaillé dans le bâtiment B



Présentation du service :

DSI : Direction de système information

La Direction des Systèmes d'Information :

Fédère l'ensemble des ressources informatiques, coordonne les différentes composantes de gestion globale des systèmes d'information, formalise les procédures et permet une meilleure prise en compte des besoins des utilisateurs.

Pour acquérir des logiciels, sauvegarder des données sur le serveur, connaître les trucs et astuces de sa messagerie, résoudre un problème de connexion depuis l'extérieur, découvrir les TICE...

Marchés informatiques 2021-2022

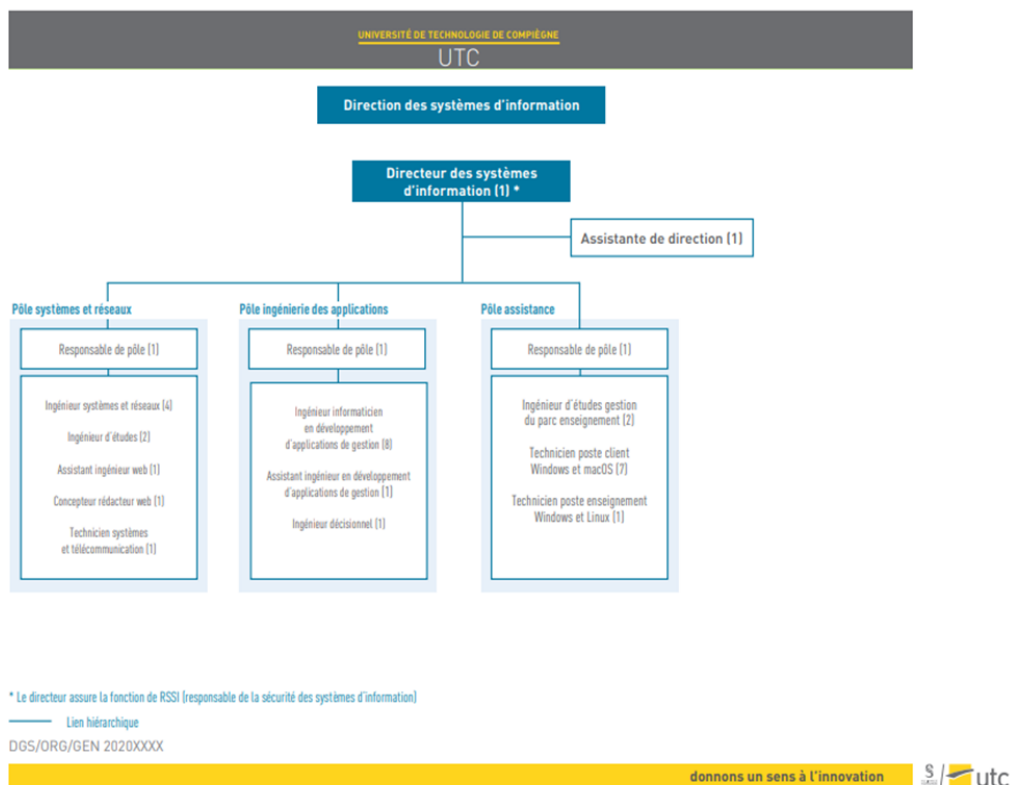
Certains achats de matériels informatiques sont soumis à des règles spécifiques d'acquisition identifiées dans le marché public ci-dessous.

Les autres achats, ceux ne rentrant pas dans le champ d'application dudit

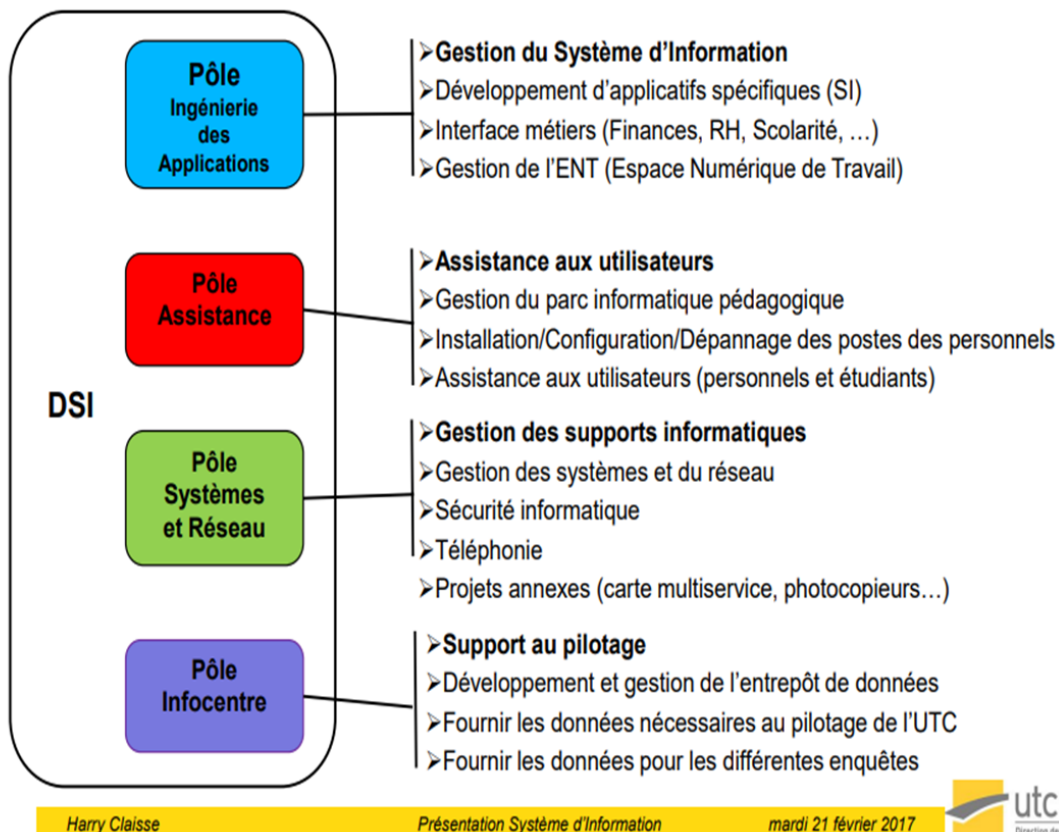
marché sont toutefois soumis aux règles de concurrence imposées par le code des marchés publics.

Il convient dès lors de prendre contact avec la DAF pour en connaître les modalités.

Organigramme DSI :



Organisation DSI :



Description des missions :

Lors de mon stage, on a pu me confier une assez grosse tâche. En effet j'ai eu le plaisir d'avoir pour mission de créer un site de covoiturage qui sera fonctionnelle seulement pour les membres de l'UTC, étudiant comme personnel de l'UTC. Pour se faire on effectue une première réunion pour bien se comprendre sur le projet, et après cette réunion le projet peut réellement !

On nous a aussi fourni un cahier des charges à compléter, donc on a commencé à le remplir. Mais dans un premier temps, j'ai regardé ce qu'il y avait comme application web libre de droit qui se faisait déjà, ensuite j'ai analysé chacune de leurs technologies. Après cette analyse j'ai fait un contre-rendu écrit dans le cahier des charges puis, oral à ma tutrice et à un technicien. J'ai donc pu voir que la plupart des applications qui existent sont en html, php et css

Nous avons dû faire l'arborescence du site web, au total nous aurons 9 pages accessibles depuis la barre de navigation. Ci dessous vous pouvez voir l'arborescence :



Voici les différentes fonctionnalités qui seront disponibles sur le site de covoiturage :

Accueil → Avant l'accès à cette page il faudra s'enregistrer sur le CAS UTC. L'accueil sera composé de champ de recherche avec lieu de départ et d'arrivée, l'heure et le lieu. Une carte avec tous les trajets disponibles sera affichée des repères seront sur cette dernière avec le nombre de départ possible dans cette ville.

Consulter mes trajets → Les trajets seront affichés dans un tableau où il est précisé si c'est un trajet Conducteur ou passager il sera possible de mettre un filtre pour avoir l'un ou l'autre d'afficher, une option pour valider le trajet sera évidemment mise en place

Supprimer mon trajet → Des radios boutons seront à sélectionner pour savoir si on veut avoir les trajets conducteurs ou passagers. Ensuite suivant la réponse on affichera dans une liste déroulante les trajets à modifier puis les détails de ce dernier seront affichés avec un bouton rouge supprimer

Mes annonces → Les annonces de l'utilisateur seront affichées par date (décroissant) une option de filtre pour trier par date (croissante/décroissante)

Déposer une annonce en conducteur ou passager → Un formulaire sera affiché avec les différents datagrid/listés déroulants à remplir. Mettre des boutons qui placent directement l'adresse. (Exemple : un bouton nommé "Centre de Recherche de l'UTC" et le bouton mettra automatiquement l'adresse du centre de recherche soit "Rue du docteur Schweitzer 60200 Compiègne")

Inscription/Connexion → Pour l'accès à l'application de covoiturage nous aurons une connexion au CAS* au préalable en suite nous récupérerons les informations que le CAS nous retourne (nom, prénom, adresse mail). Lors de la première connexion il y aura un formulaire à remplir qui obtiendra les données nécessaires à l'application

Modifier le compte → Les informations de l'utilisateur seront affichées dans un formulaire qu'il pourra modifier. Il pourra également rajouter son numéro de téléphone s'il ne l'a pas fait durant l'inscription.

Modifier une annonce → Une liste déroulante sera affichée pour sélectionner l'annonce à modifier, puis on sera redirigé vers une page qui nous affichera le formulaire de modification la suppression y sera aussi possible sur cette page

Vérification → Un datagrid devra être rempli avec un code de trajet qui aura été envoyé au(x) passager(s). Les informations des trajets et de la personne seront ensuite affichées

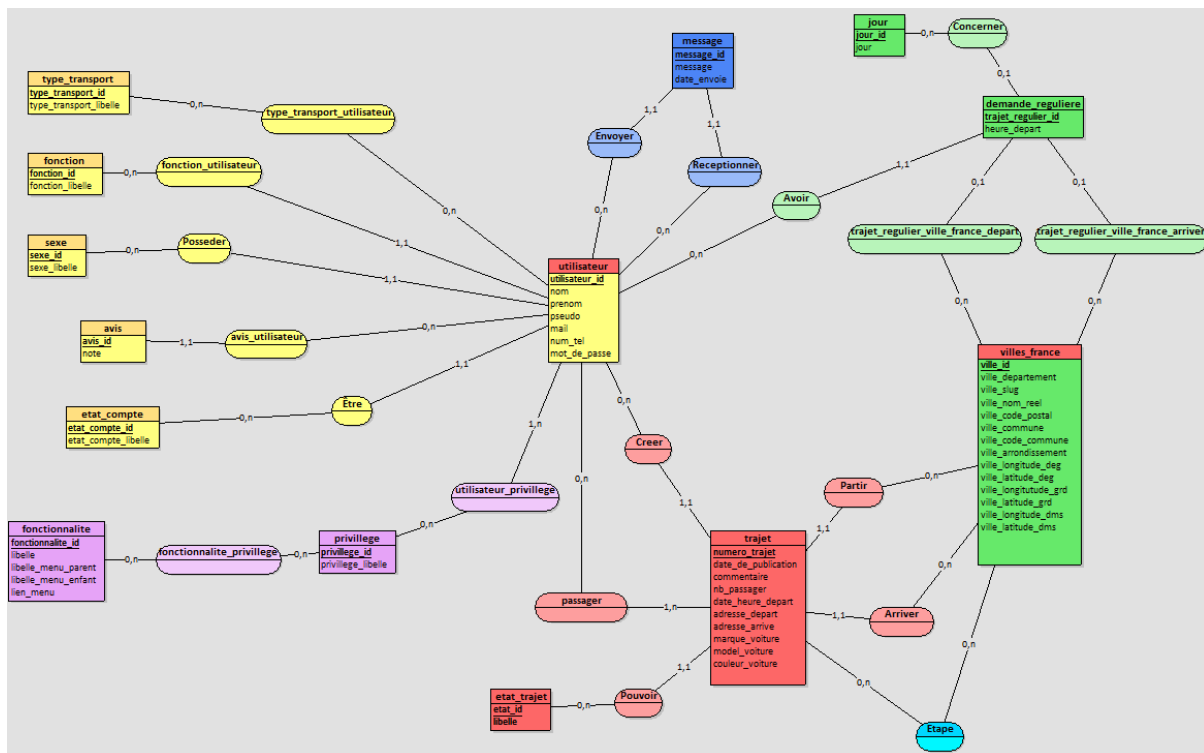
Envoyer un message → Une page afin de rédiger un message qui sera envoyé par mail ou un système de tchat (Liste déroulante avec les utilisateurs avec qui on a des covoiturages, après la sélection un mail ou un message depuis l'app sera envoyé) . Les messages seront stockés dans la base de données et sera supprimé après un délai enregistré

Consulter un message → La liste des utilisateurs des discussions sera affichée. Il suffira de cliquer sur l'utilisateur pour accéder à la conversation avec ce dernier

Planifier les trajets régulier → Un menu avec les jours de la semaine sera affiché à côté des jours il y aura un + pour ajouter un trajet, après le clic d'un des plus une redirection vers une page comportant en " titre " le jour, ensuite nous aurons des champs de sélection pour l'heure de départ (sélection d'heure uniquement <input type="time">), la ville de départ et la ville d'arrivée. Grâce à ça l'utilisateur aura un mail dès qu'un trajet correspondant à ses critères.

Modifier les trajets réguliers → L'utilisateur aura la liste de ses trajets réguliers, il y aura donc un bouton qui nous redirigera vers un formulaire afin de modifier le trajet sélectionné.

Ensuite après évaluation des besoins j'ai créer la base de donnée



Donc il y a 5 différentes parties. La partie jaune est liée à l'utilisateur, nous avons une table utilisateur qui contiendra id de l'utilisateur, son nom, prénom, pseudo, mail, numéro de téléphone son mot de passe et elle aura 3 clés étrangères. une première clé qui fait référence à l'id de la table fonction, elle contient les valeurs Étudiant et Personnelle de l'UTC. En suite la deuxième clé étrangère concerne la table sexe qui pointera sur l'id de cette table, la table sexe permet d'attribuer un sexe à l'utilisateur. Et pour finir l'état du compte qui lui va définir si le compte est actif, désactivé et suspendu. Il reste deux tables à étudier, la table type_transport et avis. La table type_transport a une relation n n avec la table utilisateur donc une autre table se crée, mais pourquoi faire ça ? Et bien tout simplement pour attribuer plusieurs types de transport à un utilisateur car il peut à la fois avoir une moto et une voiture. Et il reste la table avis elle va contenir l'avis sur un utilisateur, on ne stock pas l'auteur de l'avis car il n'aura pas la possibilité de saisir du texte il pourra juste mettre une note sur 5.

J'ai dû faire une maquette pour avoir un aperçu des besoins et pour être sûr qu'il n'y ait pas de quiproquo. (Lien de la maquette: <https://app.genial.ly/editor/61e5858253e2740d52bd1636>)



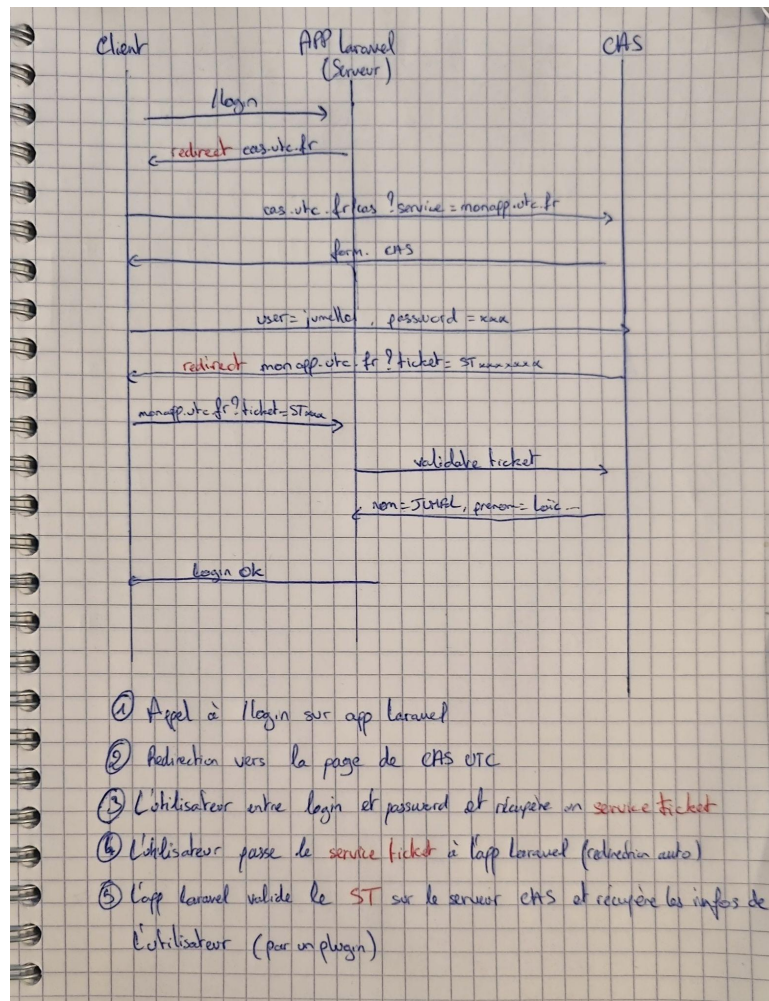
Voici la page d'accueil du site de covoiturage de l'UTC, UTCovoit.

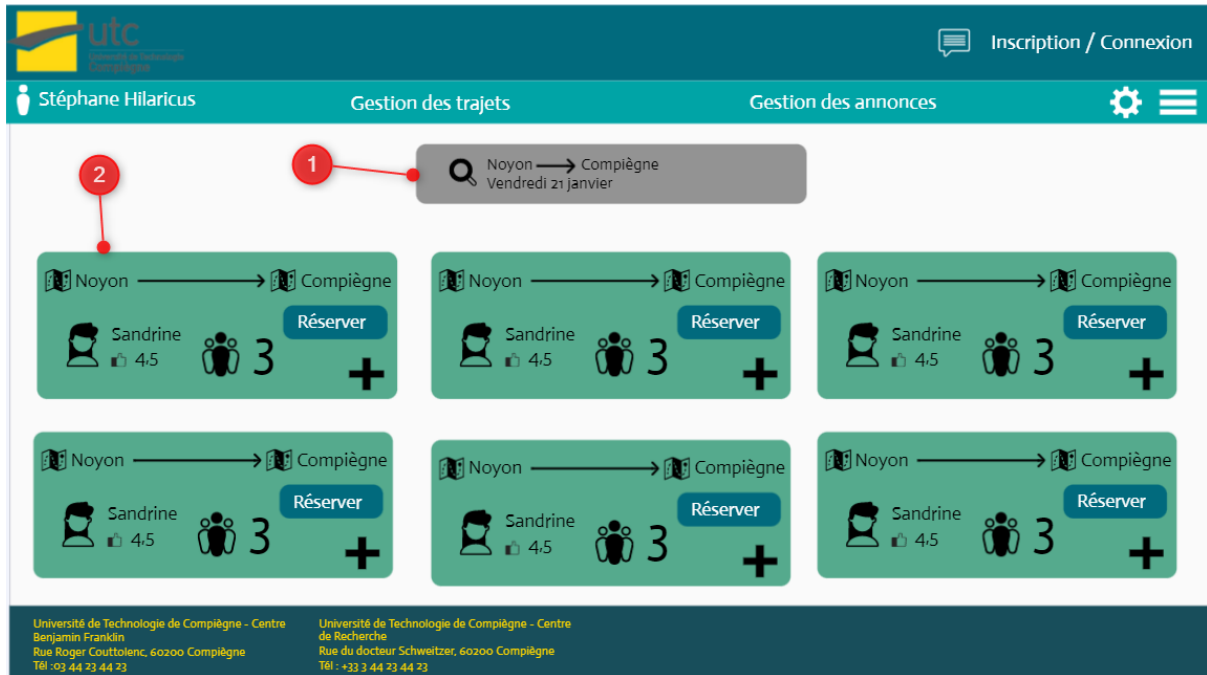
- 1- Le logo est cliquable il nous ramène à la page d'accueil
- 2- Ce bouton permet d'accéder à la fonction messagerie de l'application
- 3- Cette option est sur la maquette mais avec l'équipe on a pris la décision de ne pas l'intégrer au projet car pour qu'il n'y ai que les membre de l'UTC qui y ai accès on utilisera une de leurs application d'authentification j'en reparle plus en détails juste après !
- 4- Nous avons le nom et prénom de l'utilisateur qui est affiché dans la nav bar
- 5- Le bouton gestion des trajets permet d'afficher les fonctions consultation trajets à faire qui affiche le trajet à effectuer en tant que passager ou conducteur dans la semaine et après, nous avons vérifié le trajet qui permet de vérifier lors du covoiturage si la personne présente est bien la bonne. La vérification se fera en rentrant le numéro du trajet.
- 6- Le bouton gestion des annonces permet d'afficher les fonctions, ajout d'une annonce, consultation de mes annonces, modification et suppression de mon annonce.
- 7- Ce bouton permet d'accéder au paramètre du compte.
- 8- Les champs de texte permettent de pouvoir saisir une ville de départ/arrivée et son heure de départ.
- 9- Permet de rechercher avec les paramètres dans les champs de texte.
- 10- Le bouton "afficher tout" affiche toutes les annonces.

11- Dans le footer on peut retrouver différentes informations concernant l'UTC ou bien le site en lui-même.

Le CAS c'est quoi ?

Le cas est une interface d'authentification propre à l'UTC, il permet d'éviter l'accès aux applications de l'UTC au public. Grâce à cette connexion on récupère nom, prénom, login UTC (unique), email et population (personnel, étudiant). Voici un petit schéma :





1- Barre de recherche

2- Annonce de covoiturage

Après les présentations des derniers éléments on nous à demandé d'utiliser un framework php. J'ai choisi d'utiliser le framework **Laravel 8** pour plusieurs raisons. Les voici :

- Grande liberté d'organisation du code
- La boîte à outil qui s'occupe du back-end
 - Laravel prend en charge les backends de cache tels que Memcached et Redis prêts à l'emploi. Vous pouvez également configurer plusieurs configurations de cache
 - Sécurité et performance
 - (Votre application Web ne présente aucun risque d'injections SQL involontaires et cachées, certaines caractéristiques et fonctionnalités affectent les performances du site. Mais Laravel propose divers outils qui aident les développeurs à améliorer leurs performances.)
- La simplicité d'écriture
- La mise en place d'une API est simple
- Communication avec la base de données plus simple (ORM Eloquent)
- Documentation détaillée, appropriée et bien articulée
- Prise en charge de l'architecture MVC
- Meilleur moteur de template car il fonctionne mieux pour réutiliser le code
- Génération d'URLS
 - Laravel aide également à générer des URL
- Intégration aux services de messagerie
 - L'intégration du service de messagerie est un autre avantage fourni par Laravel.
- Créateur d'applications multilingues
- Rapidité de chargement de la page (60 millisecondes alors que symfony prend 250 millisecondes)
- Open Source et communauté de support dédiée
- Tests unitaires

○ Cela garantit que les nouvelles modifications apportées par le développeur n'interrompent pas l'application Web de manière inattendue. Le test unitaire est un type de test dans lequel chaque module ou composant de votre application Web est testé afin qu'aucune partie de votre site Web ne reste cassée.

- Applications prêtes pour l'avenir

○ La principale raison de l'immense popularité du framework Laravel est sa large gamme de fonctionnalités.

Cependant ce framework présente quelque **désavantage** comme :

- Ne prend pas en charge la fonction de paiement

- La qualité est parfois mitigée

○ Certains composants du cadre ne sont pas bien conçus. Par exemple, l'injection de dépendances devient parfois inutilement complexe. La documentation est également lourde. Vous devez apprendre beaucoup avant de vous lancer dans la création d'applications.

- Ne parvient souvent pas à fournir la richesse des applications mobiles

○ Les rechargements de pleine page peuvent être un peu lourds dans les applications mobiles par rapport aux sites Web. Dans de tels cas, les développeurs Web ont tendance à utiliser le framework comme API JSON backend uniquement.

Elle présente très peu de désavantages pour l'application qui sera développée. En effet l'application n'a pas pour but de rémunérer le conducteur des covoiturages donc on a pas besoin de fonction de paiement. Laravel 9 nous offre une assez grande liberté sur l'emplacement du code, il faut donc être méticuleux pour éviter d'avoir un code illisible.

Maintenant qu'on possède les besoins de l'application et qu'on a choisi les bonne technologies de développement on peut se répartir les tâches avec mon collègue et commencer à coder l'application.

Je vous rappelle les fonctions qui sont nécessaires, en rouge les indispensable, en orange les plus importante et en vert les dispensable.

Fonctions administrateur	Fonctions utilisateur
-Consultation des utilisateurs -Modification/Suppression des utilisateurs	-Ajout d'un utilisateur -Consultation des covoiturages -Ajout d'une annonce (conducteur/passager) -Modifier mon compte
-Consultation des annonces passé ou futur (d'un utilisateur) -Suppression des annonces	-Consult d'une annonce -Modification/Suppression d'une annonce -Consultation des trajets à faire -Vérification du trajet
	-Consultation/Ajout des messages -Ajout d'un trajet régulier -Consultation trajet régulier -Modifier un trajet régulier

Voilà comment on a décidé de se répartir les tâches :

Les tâches souligner sont les tâches qui concernent l'administration de l'application web.

Stéphane (temps estimé)→(temps réellement passé)	Temps de travail <i>Date et heure de début et de fin</i>	Etat (pas démarrer, en cours, fini)	Enzo (temps estimé)→(temps réellement passé)	Temps de travail <i>Date et heure de début et de fin</i>	Etat (pas démarrer, en cours, fini)
- Accueil (8h)	Debut 03/01/22	-En cours	- Consultation des covoiturages (3-4h)	Debut 03/02/2	En cours
- Ajout utilisateur (3h30)	10h00 → 12h,13h → 17h	-Fin	- Ajout d'une annonce (3h30)	022 14h30	-Pas démarrer
- <u>Consultation utilisateur (2h30)</u> (<i>temps réel</i> → 3h30)	Début 14/02/22 9h → 11h30	-Fin	- <u>Consultation de mes annonces (3-4h)</u>		-Pas démarrer
- <u>Modif/Supp utilisateur (4-5h)</u>	Debut 04/01/22 9h00 → 11h, 11h30→12h 30, 13h30 →14h	-Fin	- <u>Consultation des annonces passé ou futur d'un utilisateur (3h)</u>		-Pas démarrer
- Modifier mon compte (3-4h)		-En cours	- Modif/Supp d'une annonce (4h30)		-Pas démarrer
- Consultation des trajets à faire (1h30)		-Pas démarrer	- <u>Suppression des annonces (3h)</u>		-Pas démarrer
- Vérification du trajet (2h30)	Debut 15/02/22 14h→17h, 17/02 9h→	-Pas démarrer	- Ajout trajet régulier (2h30)		-Pas démarrer
- Ajout/Consultation de messages (4-5h)		-Pas démarrer	- Consultation trajet régulier (2h)		-Pas démarrer
- <u>Consultation de la messagerie d'un utilisateur (2-3h)</u>		-Pas démarrer	- Modif/Supp un trajet régulier(2h)		-Pas démarrer
Soit un total de environ 35h	Soit un total d'environ 28h30				

Je vais donc vous présenter seulement les fonctions que j'ai fini ou commencer.

Grâce au framework Laravel on peut directement générer les tables de la base de donnée dans le code donc pour chaque table nous allons utiliser la commande

```
php artisan make:model NomDuModel -m
```

Cette commande permet de créer à la fois le modèle et la migration, la migration permet de créer et définir les champs d'une table de la base de données.

Quand toutes les migrations sont toutes générées et remplies on peut exécuter la commande :

```
php artisan migrate
```

 pour ordonner au programme de créer les tables dans la base de données

Voici le contenu d'une migration :

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUtilisateursTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('utilisateurs', function (Blueprint $table) {
            $table->id(column: "id");
            $table->string(column: "nom");
            $table->string(column: "prenom");
            $table->string(column: "pseudo");
            $table->string(column: "mail");
            $table->string(column: "num_tel");
            $table->string(column: "mot_de_passe");

            $table->unique(['pseudo']);
            $table->unsignedBigInteger('sexe_id');
            $table->foreign('sexe_id')->references('sexe_id')->on('sexes');
            $table->unsignedBigInteger('fonction_id');
            $table->foreign('fonction_id')->references('fonction_id')->on('fonctions');
            $table->unsignedBigInteger('etat_compte_id');
            $table->foreign('etat_compte_id')->references('etat_compte_id')->on('etat_comptes');

            $table->timestamps();
        });
    }
}
```

Dans le premier encadré nous définissons les classes qui sont nécessaires pour utiliser certaines fonctions ce sont toutes des classes créées par Laravel. La deuxième bulle crée le champ 'id', pour définir le type et pour qu'il soit clé primaire on doit le préciser ici : `$table->id` et ensuite `(column: "id")`; indique le nom donné à la colonne. La 3ème bulle définit les autres champs de la table. Ensuite dans cette table on a une contrainte d'unicité sur la colonne pseudo et 3 clés étrangères. Pour définir la contrainte d'unicité on l'indique

grâce à la ligne suivante : `$table->unique(['nomDeLaColonne'])`; Et la clé étrangère on peut la définir comme ceci : `$table->unsignedBigInteger('sexe_id');`

`$table->foreign('sexe_id')->references('sexe_id')->on('sexes');`

La ligne `$table->unsignedBigInteger('sexe_id');` créer le champs `sexe_id` dans la table, et la ligne `$table->foreign('sexe_id')->references('sexe_id')->on('sexes');` defini la clé étrangère en référence à la colonne de la table en question.

Après la création de toutes les tables, il faut bien la remplir avec des valeurs nan ?

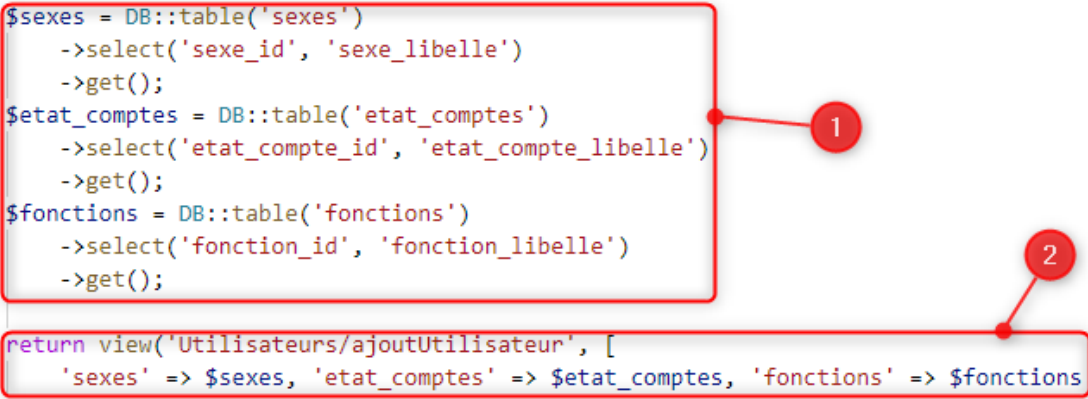
Donc j'ai pu créer la fonction d'ajout d'un utilisateur qui sera l'inscription sur le site de covoiturage en réalité.

Pour se faire dans un premier temps on creer le contrôler grace à la commande :

`php artisan make:controller UtilisateurController` qu'on entrera dans le terminal VisualStudio Code ou depuis l'exécuteur de commande windows si vous utiliser un autre IDE mais il faut que vous soyez placer sur la racine du projet avant d'exécuter la commande, puis après ça on se rend dans le controller. Que va contenir ma classe `UtilisateurController` ?

```
public function ajoutUtilisateur()
{
    #On récupère le contenu des tables sexes, etat_comptes et fonctions pour le transferer
    #dans la vu pour les listes déroulante
    $sexes = DB::table('sexes')
        ->select('sexe_id', 'sexe_libelle')
        ->get();
    $etat_comptes = DB::table('etat_comptes')
        ->select('etat_compte_id', 'etat_compte_libelle')
        ->get();
    $fonctions = DB::table('fonctions')
        ->select('fonction_id', 'fonction_libelle')
        ->get();

    return view('Utilisateurs/ajoutUtilisateur', [
        'sexes' => $sexes, 'etat_comptes' => $etat_comptes, 'fonctions' => $fonctions
    ]);
}
```



La première fonction permet de faire afficher dans la vue les différents utilisateurs qui sont dans la base données. On lui passe 3 objets afin d'afficher les valeurs dans 3 listes déroulantes. On va aller dans la vue `ajoutUtilisateur` pour voir ce qu'il s'y trouve !

```

@extends('Layouts.app')
@section('contenu')
<h6>Créer un nouvel utilisateur</h6>

<form method="POST" action="{{ route('ajoutUtilisateurTrait') }}">
    @csrf
    Nom: <input type="text" name="nom" >
    Prenom :<input type="text" name="prenom">
    Pseudo: <input type="text" name="pseudo">
    Mail: <input type="text" name="mail">
    Numéro de téléphone: <input type="text" name="num_tel">
    Mot de passe: <input type="text" name="mot_de_passe">

    <select name="sexes" id="sexes">
        @foreach ($sexes as $sexe)
            <option value="{{ $sexe->sexe_id }}" >{{ $sexe->sexe_libelle }}</option>
        @endforeach
    </select>
    <select name="fonction" id="fonction">
        @foreach ($fonctions as $fonctions)
            <option value="{{ $fonctions->fonction_id }}" >{{ $fonctions->fonction_libelle }}</option>
        @endforeach
    </select>
    <select name="etat_compte" id="etat_compte">
        @foreach ($etat_comptes as $etat_comptes)
            <option value="{{ $etat_comptes->etat_compte_id }}" >{{ $etat_comptes->etat_compte_libelle }}</option>
        @endforeach
    </select>

    <button type="submit">Créer</button>
</form>
@endsection

```

A la puce numéro une on fait étendre le Layouts qui contient le contenu qui est similaire pour toute les pages sont contenu :

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ asset('css/app.css') }}">
    <title>UTCovoit</title>
</head>
<body>
    @include('Partials.navbar')
    @yield('contenu')

    <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>

```

Il n'y a rien de bien spécial, on va y trouver le chemin du CSS, le titre de la page, la navbar qui elle est dans un autre fichier .blade.html et le chemin du scripte css. Revenons à la vue de l'ajout. Le contenu du rectangle 2 indique le début du code à écrire dans le body, le troisième rectangle contient les champs de saisie pour les différentes informations nécessaires. Dans le prochain rectangle on peut récupérer les information retourner par le controller lors de la redirection, on va les afficher dans des listes déroulante et enfin le dernier rectangle permet d'envoyer le formulaire et donc de remplir la base de donnée avec les informations qu'on a passé. Mais il faut indiquer au programme où sont les données à renvoyer c'est pour ça qu'il y a la fonction ajoutUtilisateurTrait(Request \$request)

```

public function ajoutUtilisateurTrait(Request $request)
{
    $utilisateur = new Utilisateurs();
    $utilisateur->nom = $request->nom;
    $utilisateur->prenom = $request->prenom;
    $utilisateur->pseudo = $request->pseudo;
    $utilisateur->mail = $request->mail;
    $utilisateur->num_tel = $request->num_tel;
    $utilisateur->mot_de_passe = bcrypt('$request->mot_de_passe');
    $utilisateur->sexe_id = $request->sexes;
    $utilisateur->fonction_id = $request->fonction;
    $utilisateur->etat_compte_id = $request->etat_compte;
    $utilisateur->save();

    return view('Utilisateurs/consultUtilisateurs', [
        'utilisateurs' => $utilisateur
    ]);
}

```

Dans la balise numéro 1, on indique quels champs sont à retourner et de quelle information il s'agit. Dans le dernier encadré la vue affichant les utilisateurs est retourné.

Voici la fonctionnalité ConsultUtilisateur, elle permettra de consulter tous les utilisateurs enregistrés dans la base de données. Dans le premier encadré nous avons le code qui générera la requête SELECT. Le deuxième encadré retourne la vue où tous les utilisateurs sont affichés. On passe les utilisateurs dans le return afin de pouvoir exploiter ses données dans la view.

```

public function consultUtilisateurs()
{
    #On se connecte à la base de donnée et on viens faire la requete SQL
    $utilisateurs = DB::table('utilisateurs')
        ->join('sexes', 'utilisateurs.sexe_id', '=', 'sexes.sexe_id')
        ->join('fonctions', 'utilisateurs.fonction_id', '=', 'fonctions.fonction_id')
        ->join('etat_comptes', 'utilisateurs.etat_compte_id', '=', 'etat_comptes.etat_compte_id')
        ->select(
            'utilisateurs.nom',
            'utilisateurs.prenom',
            'utilisateurs.pseudo',
            'sexes.sexe_libelle AS sexe',
            'fonctions.fonction_libelle AS fonction',
            'etat_comptes.etat_compte_libelle as etat_compte',
            'utilisateurs.id'
        )
        ->orderByRaw('nom')
        ->get();

    #renvoie la vue consultUtilisateur en lui passant les utilisateurs
    return view('Utilisateurs/consultUtilisateurs', [
        'utilisateurs' => $utilisateurs
    ]);
}

```

```

@extends('Layouts.app')

@section('contenu')
<h5>Liste utilisateurs</h5>
@if ($utilisateurs->count() > 0)
    @foreach($utilisateurs as $utilisateur)
        <div>
            <h6>Nom : {{ $utilisateur->nom }} Prénom : {{ $utilisateur->prenom }} Pseudo : {{ $utilisateur->pseudo }} Sexe : {{ $utilisateur->sexe }}
            Fonction : {{ $utilisateur->fonction }} État du compte : {{ $utilisateur->etat_compte }}</h6>
            <h6><a href="{{ route('modifSuppUtilisateur', ['id' => $utilisateur->id]) }}">Modifier l'utilisateur</a></h6>
        </div>
    @endforeach
@else
    <span>Aucun utilisateurs dans la base de données</span>
@endif
@endsection

```

Dans l'accolade A je fait un if, afin de pouvoir compter est afficher un message si les utilisateurs ne sont pas présent dans la base de donnée. L'encadré numéro 1 fait afficher les différents utilisateurs. Cependant, on doit parcourir le contenu de l'utilisateur, on utilise donc un foreach pour cela. Le deuxième encadré consiste à afficher un texte qui sera cliquable pour pouvoir accéder à la modification. Il apparaît comme ceci : [Modifier l'utilisateur](#)

Quand on clique sur le texte [Modifier l'utilisateur](#) on est redirigé vers la page de modification qui ressemble à ça :

La fonctionnalité n'a pas été terminée par manque de temps. Mais la modification s'effectue dans la base de données. Voici donc la fonction **modifSuppUtilisateur** :

```

#La fonction permet de récupérer un utilisateur en particulier grace à son id.
public function modifSuppUtilisateur($id)
{
    # / \   La fonction n'est pas encore complète, pour le moment
    # / | \  il n'y a juste la récupération de l'utilisateur grâce à son id.
    #/_°_ \ Il faut faire sa modification et sa suppression.

    $utilisateur = DB::table('utilisateurs')
        ->join('sexes', 'utilisateurs.sexe_id', '=', 'sexes.sexe_id')
        ->join('fonctions', 'utilisateurs.fonction_id', '=', 'fonctions.fonction_id')
        ->join('etat_comptes', 'utilisateurs.etat_compte_id', '=', 'etat_comptes.etat_compte_id')
        ->select(
            'utilisateurs.nom',
            'utilisateurs.prenom',
            'utilisateurs.pseudo',
            'sexes.sexe_libelle AS sexe',
            'fonctions.fonction_libelle AS fonction',
            'etat_comptes.etat_compte_libelle as etat_compte',
            'utilisateurs.id'
        )

        ->where('id', '=', $id)
        ->get();

    #On récupère le contenu des tables sexes, etat_comptes et fonctions pour le trasferer
    #dans la vu pour les listes déroulante
    $sexes = DB::table('sexes')
        ->select('sexe_id', 'sexe_libelle')
        ->get();
    $etat_comptes = DB::table('etat_comptes')
        ->select('etat_compte_id', 'etat_compte_libelle')
        ->get();
    $fonctions = DB::table('fonctions')
        ->select('fonction_id', 'fonction_libelle')
        ->get();

    //$utilisateur = $utilisateurs[$id] ?? 'L\'utilisateur n\'existe pas';
    #renvoi la vue modifSuppUtilisateur
    return view('Utilisateurs/modifSuppUtilisateur', [
        'utilisateur' => $utilisateur, 'sexes' => $sexes, 'etat_comptes' => $etat_comptes, 'fonctions' => $fonctions
    ]);
}

```

- 1- On récupère le compte à modifier dans la base de données
- 2- On récupère les différentes information à afficher dans les listes déroulante
- 3- On retourne les information qu'on a été récupéré pour les transmettre à la vue

```

@extends('Layouts.app')

@section('contenu')
<h5>Liste utilisateurs</h5>
@if ($utilisateur->count() > 0)
@foreach($utilisateur as $utilisateur)
<div>
<h6>Nom : {{ $utilisateur->nom }} Prenom : {{ $utilisateur->prenom }} Pseudo : {{ $utilisateur->pseudo }} Sexe : {{ $utilisateur->sexe }}
Fonction : {{ $utilisateur->fonction }} État du compte : {{ $utilisateur->etat_compte }}</h6>

<form method="POST" action="{{ route('modifSuppUtilisateurTrait',['id' => $utilisateur->id]) }}">
@csrf
Nom: <input type="text" name="nom" >
Prenom: <input type="text" name="prenom">
Pseudo: <input type="text" name="pseudo">
Mail: <input type="text" name="mail">
Numéro de téléphone: <input type="text" name="num_tel">
Mot de passe: <input type="text" name="mot_de_passe">

<select name="sexes" id="sexes">
@foreach ($sexes as $sexe)
<option value="{{ $sexe->sexe_id }}" >{{ $sexe->sexe_libelle }}</option>
</select>
<select name="fonction" id="fonction">
@foreach ($fonctions as $fonctions)
<option value="{{ $fonctions->fonction_id }}" >{{ $fonctions->fonction_libelle }}</option>
</select>
<select name="etat_compte" id="etat_compte">
@foreach ($etat_comptes as $etat_comptes)
<option value="{{ $etat_comptes->etat_compte_id }}" >{{ $etat_comptes->etat_compte_libelle }}</option>
</select>

<button type="submit">Créer</button>

</div>
</foreach>
@else
<span>Aucun utilisateurs dans la base de données</span>
</if>
</section>

```

Dans le premier encadré on affiche l'utilisateur récupérer dans la fonction **modifSuppUtilisateur**. Ensuite dans le deuxième encadré on affiche des comboBox et des listes déroulantes, les liste déroulante va contenir certaines informations récupérées dans la fonction **modifSuppUtilisateur**.

```

public function modifSuppUtilisateurTrait(Request $request, $id)
{
    $utilisateur = Utilisateurs::find($id);
    $utilisateur->nom = $request->nom;
    $utilisateur->prenom = $request->prenom;
    $utilisateur->pseudo = $request->pseudo;
    $utilisateur->mail = $request->mail;
    $utilisateur->num_tel = $request->num_tel;
    $utilisateur->mot_de_passe = bcrypt('$request->mot_de_passe');
    $utilisateur->sexe_id = $request->sexes;
    $utilisateur->fonction_id = $request->fonction;
    $utilisateur->etat_compte_id = $request->etat_compte;
    $utilisateur->save();

    return view('Utilisateurs/modifSuppUtilisateur', [
        'utilisateurs' => $utilisateur
    ]);
}

```

Le traitement reste le même pour l'ajout sauf que là on va récupérer l'id afin de faire les modifications sur le compte en question.

Conclusion :

Mon stage au sein de l'UTC m'a apporté beaucoup de savoir en plus. De surcroît j'ai appris à travailler avec des collègues qui n'ont pas les mêmes fonctions, j'ai donc dû adapter mon langage technique pour qu'il soit plus compréhensible. J'ai dû rendre des comptes sur le travail que j'effectuais donc j'ai réaliser plusieurs présentation comme ces trois ci :

<https://view.genial.ly/61e5858253e2740d52bd1636/interactive-content-maquettagesite-covoiturage>

<https://view.genial.ly/61f26fd310ec170019b77df1/interactive-content-presentation-laravel>

<https://view.genial.ly/620f5fba02538a0010070f3e/presentation-subject-presentation>

La première présentation est la maquette du site de covoiturage, la deuxième à pour but de présenter le framework Laravel et ensuite, le dernier diapositive présente le bout de l'application qui a été réalisé, afin qu'un autre stagiaire puisse prendre connaissance du projet.

J'ai développé un goût prononcé pour la gestion de projet, il est vrai je ne gérais pas complètement le projet mais au plus souvent je tenais à jour les différents documents de gestion et étais assez dynamique en globalité.

Si je doit regretter une chose durant ce stage c'est le temps mis avant de débiter le projet. Au départ j'étais dans le flou sur les attentes de mes supérieurs. Ensuite, quand tout était plus clair j'ai eu un peu de difficulté à lancer le projet. La création de cahier des charges,

l'évaluation des besoins et de ses priorités, etc.. Ces choses-là n'ont pas été réalisées avec rapidité car je n'avais pas encore fait ça.

Concernant les personnes qui m'ont encadré, ils ont tous répondu présent lors de mes différentes demandes, et toujours avec passion et sympathie.