1 – INTRO

* PRESENTATION

Je m’appelle Christopher Chiarandini. J’ai fait des études dans le commerce et après avoir obtenue mon BAC STT Action et Communication Commerciale, j’ai dû entrer dans la vie active.

Après différents petits jobs, j’ai obtenu un poste de commerciale sédentaire pour une filiale automobile durant 2 ans qui a suivi par un poste de courtier en formation professionnelle pour une start up basé à Paris.

Après quelques années passées dans cette structure j’ai décidé de partir de Paris pour m’installer à Toulouse où j’ai travaillé dans l’agro-alimentaire qui m’a permis de devenir Responsable Manager.

J’ai toujours eu un attrait pour le monde du numérique, j’ai suivi quelques formations en ligne sur mon temps libre pour en apprendre plus sur le sujet. Au fur et à mesure, je me suis rendu compte que c’est ce que je voulais faire.

Après quelque temps de réflexion et n’étant plus en phase avec ce métier, j’ai décidé de tout arrêter pour suivre cette voie. Après quelque mois de recherche, j’ai trouvé la formation de Développeur d’Application Web et Web Mobile proposé par l’ADRAR et ai donc passé les tests afin d’intégrer la session.

Lors de mon stage de fin d’étude, j’ai été initié aux pratiques en entreprise qui m’a apporté énormément de connaissances que ce soit sur les langages utilisés comme sur le travail en équipe sur un même projet.

* REMERCIEMENT

Avant de continuer ce mémoire, je tiens à remercier l’ADRAR et tout particulièrement Sophie POULAKOS qui a eu confiance en moi lors des phases de test et aussi Mathieu MITHRIDATE pour son soutient et ses nombreux conseils lors de la formation.

Je tiens aussi à remercier l’ensemble de l’équipe des formateurs pour l’apport de connaissances qu’ils nous ont apporté.

Je souhaite aussi remercier l’ensemble de ma promotion pour les moments d’échanges et de qualités passés ensemble durant ces 10 mois de formation.

Je remercie, également Florian CHEVALLIER, mon maitre de stage durant ces 2 mois chez ATECNA pour le temps qu’il m’a accordé et pour m’avoir donné énormément d’informations et m’a beaucoup aidé sur les bonnes pratiques en entreprise et la vision optimale de la conception d’un projet. Je remercie aussi l’ensemble de l’équipe pour leur accueil et toute les informations qu’ils ont pu me donner durant ce stage.

Et pour finir, merci à Julie CHIARANDINI BOLIOLI, mon épouse, pour tout le soutient qu’elle m’a apportée durant les phases de doute et de difficultés

2 ANALYSE DU BESOIN

* PRESENTATION DU PROJET

Pour pouvoir valider notre titre, nous avons dû, au début de notre formation, choisir un projet fil rouge sur lequel avancé tout au long de l’année en vue de l’écriture de ce mémoire et de notre présentation orale.

J’ai décidé de travailler sur un sujet qui me tient à cœur : le réchauffement climatique.

La période que nous vivons et un moment charnière pour l’environnement et ceux qui en dépendent.

Nous utilisons de plus en plus d’équipement ayant besoin d’énergie (électricité, énergie fossile etc…)

La fast-fashion et la consommation de masse n’est pas sans reste ce qui entraine un réchauffement anormal et rapide des températures sur l’ensemble du globe et menace de plus en plus la faune comme la flore.

* CONTEXTE

Le projet sera porté par un site vitrine qui aura pour objectif d’apporter à l’utilisateur des informations données de façon simple et ludiques

Il tournera sur 3 axes majeurs :

* **Sensibiliser**: par l’apport de chiffres recensés par le DATALAB Edition 2023 du ministère de la transition écologique
* **Informer**: l’utilisateur de bonnes pratiques pour réduire nos émissions de CO2
* **S’entraider**: au travers d’un forum, qui permettra aux utilisateurs de discuter entre eux et de s’apporter des éléments de réponses.
* LANGAGES UTILISES

Étant un site vitrine constitué d’un forum et d’une messagerie, le produit a été réalisé avec les langages suivants :

Pour la partie Front End : HTML5, Javascript.

Le CSS est géré par le framework Bootstrap (ou Tailwind) qui permet de travailler le CSS d’une façon plus simple, rapide et facilement modulable.

En ce qui concerne la partie Back End : PHP sera le modèle de langage le plus adapté aux besoins.

* CONTRAINTES TECHNIQUES

Dans une ère où plus de 70% du trafic internet s’effectue sur mobile, le produit a été pensé en Mobile First et pourra ( /devra) s’adapté à tous les supports, le produit sera donc responsive.

Par soucis d’accessibilité à l’ensemble des utilisateurs, le produit aura différentes options d’accessibilités :

* Modifications de la police d’écriture pour les personnes atteintes de dyslexie
* Grossissement des caractères
* Modification des couleurs pour les personnes atteintes de daltonisme
* Mise en place de synthèse vocale pour les personnes malvoyantes
* OUTILS UTILISES
* LOGICIEL

Maquettage

[Logo Figma] Figma : Figma est une application de conception d’Interface Utilisateur (UI) et d’Expérience Utilisateur (UX) largement utilisé par les designers.

Elle permet de concevoir des projet grâce à une interface intuitive et de créer des prototypes interactifs.

Conception

[Logo starUML] StarUML : Star UML est un outil de modélisation UML (Unified Modeling Language). Il permet de créer des use cases (cas d’utilisation), des diagrammes de classes, de séquences et d’activités afin de concevoir et documenter des systèmes logiciels. Il est très utilisé par les développeurs et les architectes logiciel.

[Logo Looping] Looping : Looping est un logiciel de modélisation conceptuel de données entité-association. Il va permettre la conception de l’architecture d’une base de données.

Éditeur de code

[Logo Vs Code] Vs Code : C’est un éditeur de code source léger développé par Microsoft.

Il permet la création de code pour de nombreux langages de programmation comme Javascript, Python, PHP et bien d’autre.

Serveur Locale et Conception SQL

[Logo XAMPP] XAMMP : C’est un logiciel qui fournit un environnement de développement web local regroupant des outils comme Apache, MySQL, PHP et Perl permettant de créer, développer et test des applications web directement sur son ordinateur sans avoir besoin d’une connexion internet et d’un serveur distant.

[Logo Workbench] : Workbench est un outil graphique de modélisation de Base de Données et d’administration MySQL. Il permet de générer du code SQL pour créer ou modifier des bases de données relationnelles. Il permet de gérer des connexions de base de données ainsi que l’exécution de requête SQL.

* LANGAGES

[Logo HTML]HTML (Hypertext Markup Language) : C’est un langage de balisage permettant de créer et structurer le contenu d’une page web. Il va permettre de structurer de façon logique le contenu textuel d’une page grâce à un système de balise (< >) permettant de hiérarchiser les éléments tel que les titres, les textes, les images, …

[Logo CSS] CSS (Cascading Style Sheets) : C’est un langage base sur des feuille de style afin de mettre en forme et de styliser le code HTML. Il va permettre de contrôler l’apparence visuelle des éléments comme la couleur, la taille et la disposition grâce à l’application de règle de style à l’aide de sélecteur et de propriétés CSS.

[Logo Javascript] Javascript : C’est un langage de programmation côté client utilisé afin de rendre les pages interactive et dynamique. Il permet la manipulation de contenu HTML , l’interaction avec l’utilisateur et la gestion des évènements.

[Logo PHP] PHP : C’est un langage côté serveur utilisé permettant de se connecter et d’interagir avec les bases de données pour la création et la récupération d’informations. Il va permettre de générer du contenu HTML contribuant à la création d’application web dynamiques et interactives

* ARBORESCENCE DU PROJET

[Screen de l’arborecence]

L'arborescence d'un site web est une représentation de sa structure organisationnelle, décrivant comment les différentes pages et sections sont organisées et liées entre elles. Elle est souvent représentée sous forme de diagramme ou de liste hiérarchique, montrant les pages principales, les sous-pages et les liens qui les connectent. Cette arborescence permet de visualiser la navigation du site et aide à planifier la disposition et l'organisation du contenu de manière logique et intuitive pour les utilisateurs.

3- SPECIFICATION FONCTIONNELLES

* USE CASE

Le Use Case (Cas d’Utilisation en français) est une modélisation qui permet de décrire les interactions entre les utilisateur (acteur) et le system d’information.

Il définit les différentes actions que l’utilisateur peut entreprendre sur le système pour atteindre un objectif spécifique.

Il est représenté sous forme de schéma qui montre les différents acteurs, les cas d’utilisations et la relation entre eux. Il permet de comprendre les fonctionnalités du système du point de vue de l’utilisateur.

[img acteur] Ceci est la représentation de l’acteur dans le schéma, ici le modérateur

[img heritage] Ceci est un héritage. Le système d’héritage est une relation entre uncas d’utilisation spécifique et un cas d’utilisation plus général.

Elle indique que le cas spécifique hérite des caractéristiques du cas générale.

Elle va permettre de structurer et de simplifier la lecture du diagramme en regroupant les fonctionnalités communes à un cas d’utilisation plus général.

Ici, dans l’exemple l’utilisateur hérite des fonctionnalités du visiteur.

[img cas d’utilisation] Ceci est un cas d’utilisation. C’est une représentation d’une action que l’acteur sera autorisé à faire sur le system d’information.

[img include] L’include (ou inclusion) est une relation obligatoire entre cas d’utilisation. Elle est symbolisé par une flèche pointant vers le cas inclus avec la mention <<include>>.

Ex  sur le use case : Le modérateur peut effacer un message sur une discussion ce qui inclus d’afficher les réponses à la discussion.

[img extend] L’extend ou extension, quant à elle, est une relation optionnelle entre cas d’utilisation. Elle est symbolisée par une flèche qui pointe ver le cas étendu avec la mention <<extend>>

* DIAGRAMME D’ACTIVITE

Le diagramme d’activité est diagramme UML de modélisation comportementale qui a pour but de renseigner les flux de contrôle et d’activité dans un système.

Il permet de représenter graphiquement, les différentes étapes liées au déroulement d’un processus.

Pour une meilleure lisibilité, le diagramme d’activité utilise différents symboles ayant chacun leur propre action :

[screen etat initial] L’état initial : il représente le début du processus.

[screen accept signal] Accept Signal : Il représente l’étape qui reçoit un signal entrant ou un évènement externe qui lui permet de pouvoir continuer son exécution.

[screen action] Action : représentes les différentes tâches de chaque étapes du processus.

[screen nœud de décision] Le Nœud de décision : C’est un point de décision dans le processus où le flux d’activités peut diverger en fonction d’une condition ou d’une décision.

[Screen fin de flot] Le nœud de fin de flow représente la fin d’un flux d’activités spécifique à une branche, il n’a aucune incidence sur le reste des flux actifs de l’activité.

[Screen send signal] Il représente une action qui déclenche l’envoi d’un signal ou d’un évènement à un autre élément du system.

[Screen etat final] L’état final représente, quant à lui, la fin du processus

Pour le bien de mon mémoire et eu reste de la présentation, j’ai pris le parti de considérer que l’utilisateur est déjà connecté et ai décidé de présenter **l’affichage et l’envoi de message dans une messagerie interne.**

[screen du diagramme d’activité]

* DIAGRAMME DE SEQUENCE

Un diagramme de séquence UML est une modélisation qui montre comment différents objets ou composants interagissent les uns avec les autres dans un ordre chronologique et spécifique.

Il est utile pour comprendre et communiquer la logique des interactions et des échanges de messages entre les différentes parties d'une application.

A l’instar du diagramme d’activité, le diagramme de séquence représente un seul scénario.

Pour une meilleure lisibilité, le diagramme de séquence utilise aussi différents symboles ayant chacun leur propre action :

[screen Lifeline] : La Lifeline représente l’existence temporelle d’un objet ou d’une entité participant à l’interaction. En la suivant, on peut observer son état et ses interactions avec les autres objets au fil du temps ce qui va aider à la compréhension du déroulement du scénarios d’utilisation.

[screen message synchrone] Le message synchrone est une message où l’émetteur attend une réponse du système.

[screen self message] Le self message est un message envoyé par un objet à lui-même. Il indique que l’objet en question déclenche une action qui n’interfère avec aucun autre objet.

[screen reply message] Le reply message est le message retour envoyé par le système en réponse au message envoyé par l’émetteur.

Le scénario alternatif représente une séquence d’action qui se produit à des conditions spécifique qui diffèrent du scénario initial. Il permet de prendre en compte les différentes possibilités et le cas exceptionnels dans le comportement du système en offrant une vue complète et détaillé des interactions.

[screen scenario alternatif] Dans cette exemple , le Si informe que la discussion n’existe pas. L’utilisateur a le choix de choisir une autre discussion et le scénario nominal peut reprendre.

Le scénario d’erreur représente une séquence d’action qui se produit lorsque des erreurs ou des exceptions se produisent pendant le scénario principal. Il est utilisé pour modéliser les réponses du système a des situations exceptionnelles ou a des conditions d’erreur telle que les erreurs de traitement, les cas non gérés ou les défaillances matérielles

[screen scenario d’erreur] Dans cette exemple, le scénario d’erreur définit qu’une défaillance lié au serveur s’est produit. U. message de retour est renvoyé. L’utilisateur ne peut rien faire pour résoudre le problème

Ci-dessous, le digramme de séquence représentant le scénario nominal de l’affichage et l’envoi d’un message.

Le scénario nominal est définit par le fait que le système suit un **chemin normal** sans rencontré de **scénario alternatif** ni de **scénario d’erreur.**

[screen diagramme de séquence]

4- CONCEPTION

* MCD

Le MCD (Modèle de Conception de Données) est la première étape clés dans la conception, il est une représentation formelle des concepts et des relations d’un système d’information.

Il est utilisé afin de décrire les entités, les attributs, les associations et les contraintes qui compose le modèle de données.

Il constitue le cahier des charges pour la conception d’une base de données.

En résumé, le MCD est une représentation graphique structuré des données facilitant la compréhension de leur gestion.

Définissons les différents éléments d’un MCD :

[screen Entité] **Entité** : Elle représente un objet ou un concept du « monde réel » sur lequel on souhaite stocker des informations (**Attributs**) dans la base de données. C’est une unité élémentaire qui se suffit à elle-même, dans notre exemple : un message.

**Attribut** : Un attribut est une caractéristique ou une propriété qui décrit l’**entité**. Chaque entité peut avoir plusieurs attributs qui vont informer sur ses caractéristiques, dans notre exemple : l’entité message a 8 attributs : id\_messages qui est un attribut de type **COUNTER** (qui s’auto-incrémente) et **PRIMARY\_KEY** (qui permet d’identifier de façon unique un attribut), text-message de type **TEXT** (chaine de plus de 255 caractères), data\_message de type **DATATIME** (qui renvoi la date et l’heure) **sender** et **recipient** qui sont de type **INTEGER** (un nombre), id \_talk de type **VARCHAR(100)** (chaine de 100 caractères) ainsi que read-messages et is\_hidden\_messages de type **TINYINT(0)** (un type de booléen qui renvoit 0 ou 1)

[screen relation cardinalité] Association : Une association représente une relation/connexion entre deux ou plusieurs entités. Elle sert à décrire comment les entités sont liées entre elle. **Les cardinalités** vont jouer un rôle important dans l’association qui permettre de mieux comprendre la relation entre les tables.

**Les cardinalités** : Les cardinalités définissent le nombres d’occurrences d’une entité qui peuvent être associés à une autre entité au travers de la relation, dans notre exemple nous avons une relation 1,1 de messages vers Users (C’est-à-dire qu’un message doit être attribué a, au minimum (1) et maximum (1), un user. Dans l’autre cas nous avons une relation de 0,n de Users vers messages, ce qui veut dire qu’un user peut ne pas avoir écrit de messages (0) mais a pu aussi en écrire plusieurs (n).

À la suite de ses précisions, voici une représentation complète du MCD du projet :

[screen du MCD]

* MLD

Les MLD (Modèle Logique de Données) est une représentions structurés des données d’un système où les entités, attributs et relations sont spécifié de manière détaillé sous forme d’un schéma des bases de données relationnelles.

A contrario du MCD qui se concentre sur les concepts et les relations entre le données , le MLD se focalise sur la manière dont les données seront stockées et organisées de façon concrète dans la base de données.

Le schéma va nous données plusieurs informations comme :

* Les Clé Primaire : représenté par un **attribut de couloir noir, en gras et surligné**, elle identifie de manière unique chaque occurrence de l’entité dans une table de la base de données.
* Les clé Étrangère : représenté par *un attribut de couleur bleu en italique*, elle identifie, quant à elle, une clé primaire d’un autre entité , établissant une relation entre les deux entités dans la base de données.

Le sens des flèches est aussi important dans la compréhension du schèma :

* Une flèche pointant vers l’extérieur d’une entité signifie que l’entité a une clé étrangère qui fait référence à une clé primaire d’un autre entité.
* Un flèche pointant vers l’intérieur d’un entité signifie qu’un clé étrangère dans une autre entité est lié à la clé primaire de cette entité.

Pour faire plus simple, le sens de la flèche indique la direction de la relation entre les entités et comment les données sont oganisées en base de données relationnelle.

Ci-dessous, la représentation du MLD du projet :

[screen MLD]

* LE MAQUETTAGE

Le maquettage est un processus de conception visuelle utilisé pour créer des représentation graphiques ou interactive d’un système avant sa mise en développement.

Il permet au concepteur de visualiser et de communiquer les idées, les fonctionnalités et les interactions du projet. Il va permettre aussi de recueillir des informations des parties prenantes avant la mise en développement du projet.

Le maquettage suit différentes étapes :

* Le Zoning : Le zoning va permettre de définir la structure globale de l’interface du projet et organiser les différents éléments fonctionnels.

[screen du zoning a définir]

* Le Wireframe : Le wireframe est une représentation visuelle simplifié de l’interface utilisateur (UI), il se concentre principalement sur la structure et la disposition des éléments de l’interface sans se soucier, pour le moment, des détails esthétiques (couleur, images, polices, etc..). Il va permettre d’architecturer l’information, la navigations, les zones fonctionnelles et les interaction de bases donnant une base solides pour les prochaines étapes du maquettage.

[screen du wireframe à définir]

* Le prototype : le prototype est une version interactive du wireframe. Il offre une simulation réaliste des fonctionnalités et des interactions. Il permet de tester les fonctionnalités et d’avoir un retour d’expérience utilisateur (UX) afin de pouvoir, le cas échéant, modifier certains élément avant de passer à la conception visuelle finale.

[screen prototype avec les intéraction]

* Le Mockup : le mockup est la phase finale du maquettage, il va donner la représentation graphique finale et précise du projet. Il inclut, l’ensemble des détails esthétique comme les couleurs (reprise de la charte graphique), les polices, les images et les éléments graphiques.

Il va servir afin de faire la présentation finale pour validation auprès des clients avant la mise en développement du projet.

[screen mockup à définir]

* SQL

Afin de créer les différentes tables de la base de données dont aura besoin le projet, j’ai utilisé le logiciel Workbench.

Pour rappel, Workbench est un outil de modélisation de base de données et d’administration MySQL

Pour me permettre d’accéder à mon administration, j’utilise en parallèle l’outil XAMPP qui me permet de créer une connexion l’administration de MySQL grâce à un serveur local.

Pour le bien de ce mémoire, la présentation s’axera sur la création de la Base de données « nw\_change\_ton\_climat » ainsi que sur la création de la table « messages » et ses foreign key.

[screen create database] La commande **CREATE DATABASE nw\_change\_ton\_climat** permet de crée une base de donnée qui aura pour nom « nw\_change\_ton\_climat ».

La commande **USE nw\_change\_ton\_climat** va permettre de se place dans la base de données afin d’y créer les tables

[screen create table messages] La commande **CREATE TABLE messages ()** va créer la table messages. Entre le parenthèse, nous allons pouvoir insérer nos différents attributs, ici :

* **Id\_messages INT AUTO\_INCREMENT NOT NULL PRIMARY KEY** nous crée un attribut qui prend différents paramètres **INT** pour indique qu’il s’agira d’un entier, **AUTO\_INCREMENT** pour indique que la colonne id\_messages doit doit insérer a chaque nouvelle ligne une nouvelle valeur, **NOT NULL** informe que la colonne de doit jamais avoir la valeur **NULL** et **PRIMARY\_KEY** informe que cette colonne doit avoir une valeur unique.
* **text\_messages TEXT** crée un attribut de type T**EXT** qui correspond à une chaine de caractères de plus de 255 caractères.
* **date\_messages** **DATETIME** crée un attribut de type **DATETIME** qui insère à la BDD une date et une heure.
* **sender** **INT** et **recipient** **INT** crée des attributs de type **ENTIER** (elle représente les foreign key)
* **read\_messages TINYINT(1) DEFAULT 0** et **is\_hidden\_messages TINYINT(1) DEFAULT** **0** crée un attribut de type **TINYINT** c’est-à-dire un petit entier, **DEFAULT 0** indique que par default l’entier sera 0 (nous nous serviront de cette attribut comme d’un booléen qui renverrait true ou false)
* **idtalk\_message INT** crée aussi un entier.

**Engine= InnoDB**, referme la création de la table en indiquant le moteur de stockage à utiliser. Ici, on utilisera InnoDB qui permet plusieurs fonctionnalités avancées.

[screen foreign key] la Commande **ALTER TABLE messages** indique sur quelle table nous voulons travailler , **ADD CONSTRAINT** ajoute une contrainte à notre table qui permet de définir des conditions spécifiques qui doivent être respecté pour l’intégrité des données. **FOREIGN KEY** fait référence à la clé étrangère dans la table et **REFERENCES** permet de lié la clé étrangère à la colonne id\_users de la table users (users(id\_users))

-FRONT

Le projet a été créée sous le modèle **MVC (Model View Controller).** C’est un type d’architecture logicielle qui divise une application en 3 composants interconnectés que nous détailleront ci-dessous.

Ce choix a été fait car ce modèle permet une meilleure lisibilité du code. Il facilite aussi la gestion, la maintenance et l’évolutivité de l’application. C’est un modèle largement utilisé dans le développement d’application web et logiciel.

[screen 0-MVC]

**App**: Ce dossier regroupe toute la logique de l’application.

**Controller**: Dans ce répertoire, on retrouve l’ensemble des fichiers qui sont responsables de la gestion des requêtes des utilisateurs, de la coordination des interactions entre les modèles et les vues et de la génération des réponses appropriées.

**Model :** Dans ce répertoire, on retrouve l’ensemble des model servant à représenter la structure des données et la logique métier de l’application. Le model va réagir avec la base de données afin de récupérer, stocker mettre à jour ou supprimer les informations nécessaires à l’application (CRUD)

**Utils :** Dans ce répertoire, on retrouve l’ensemble des fichiers de codes pouvant être réutilisable de façon indépendante au model, au view ou au controller. Ici dans ce projet, il sera utilisé pour créer une fonction qui permettra de nettoyer chaque entrée des formulaires. On y retrouvera aussi une fonction qui permettra la connexion à la base de données ce qui permettra de ne pas dupliquer ce code plusieurs fois dans l’application.

Vue : Ce répertoire contient l’ensemble des fichiers responsables de l’interface utilisateur et de l’affichage des données. Elle présente les informations aux utilisateurs de manières à ce qu’elles soient facilement compréhensibles et interactives. La vue va récupérer les données du model envoyer par le controller pour les afficher à l’utilisateur.

**Public :** Ce répertoire va contenir l’ensemble des éléments nécessaire au bon affichage sur le navigateur. Il va contenir l’ensemble des images, des fichiers **CSS** (pour l’habillage de l’application) ainsi que le fichiers **Javascript** (pour l’interactivité et le dynamisme de l’application)

.htaccess : Ce fichier est un fichier de configuration pour le serveur. Il est utilisé pour définir des règles et des directives qui affectent le comportement du serveur tel que les réécritures d’url, la gestion des erreurs etc…

Index.php : C’est la racine de notre application. C’est le fichier qui permettra d’afficher l’ensemble des éléments dès que l’utilisateur se connectera à l’application.

* PRESENTATION DU CODE HTML/CSS/JS: Carte interactive

[screen 0.5 HTML] Dans un premier temps, j’ai initialisé le **controller chiffreController** qui mettra en place l’ensemble de la logique des information envoyer au navigateur de l’utilisateur.

* La commande **namespace** va nous permettre d’instancier le chemin de notre fichier, ici chiffreController.
* La commande **use** va appeler le fichier Template qui aura pour rôle de mettre en place l’aspect de la page.
* La **class chiffreController** prend la **function public getChiffre** qui, elle, renvoi l’ensemble des informations au fichier Template grâce à la méthode Template ::render() . Celle méthode renvoi :
  + Le fichier contenant la navigation (navbar.php)
  + Le titre de la page
  + Le fichier contenant la partie inforrmative de la page (vueChiffre.php)
  + Le Footer de la page (footer.php),
  + L’initialisation de la variable d’erreur, au cas où il y ai une erreur a afficher à l’utilisateur
  + Les fichiers javascript et css .

[screen 1-HTML-Head] La première partie est envoyé au Template qui nous sert de page HTML. Il est composé en 2 parties :

* Le <**head>** : Le head est une partie réservée au navigateur, qui n’est pas visible par l’utilisateur, afin de lui donner des informations supplémentaire sur la page. Dans notre cas, je lui passe plusieurs données
  + Le **charset** qui va permet d’informer sur l’encodage des caractères, de base UTF-8 est le plus utilisé car il prend en charge la plupart des caractères et langues les plus utilisés dans le monde.
  + Les balises **<link>** vont permettre d’initialiser les fichiers CSS dont j’aurais besoin, ici je délivre le CDN de Bootstrap qui me permettra d’utiliser l’ensemble des outils proposer par ce Framework. On peut remarquer une boucle **foreach** qui appelle des fichiers. Celle-ci appelle les fichiers donnés au controller.
  + Les balises **<script>** on le même rôle de la balise **<link>** mais pour appeler des fichiers javascript.
  + Pour finir, la balise <title> va permettre d’afficher le titre de la page dans l’onglet du navigateur (et va servir au référencement). Ici le titre est lui aussi passer par l’intermédiaire du Controller présenté auparavant.
* Le **<body>** : le body est la partie visible par l’utilisateur par le biais de son navigateur. Dans notre cas le body comporte 3 balises qui sont passé par l’intermédiaire du Controller :
  + Les balises < ?= $navbar ?> et < ?= $footer ?> sont présenter en Annexe (NUMERO ANNEXE A INDEXER)
  + La balise < ?= $content  ?> est présenter ci-dessous.

A noter que l’instanciation « < ?= $ex\_contenu ?> » permet d’insérer du code PHP au sein de code HTML.

[screen 4-HTML-Body] Dans cette partie, je vous présente le code qui me permet d’afficher une carte interactive sur le navigateur de l’utilisateur.

Cette partie est divisé en 2 balises **<article>**, la première va servir à afficher :

* Une **<div> « Tooltip »** qui, au chargement n’est pas visible mais sera appeler grâce à du code Javascript que nous verrons ultérieurement. Cette balise comporte des paragraphes dans lesquelles des informations seront injectés dynamiquement.
* Une **<div> « worldmap »** qui comprend une Mappemonde au **format SVG**. J’ai choisi ce format pour plusieurs raisons : la première est que le format SVG agit comme un format vectoriel, il n’y a aucune pixellisation de l’image en cas d’agrandissement en fonction de l’écran. La seconde est que ce format permet la récupération d’information au sein même de l’image. Ici je récupérer une information concernant le nom des pays, mais nous y reviendrons plus tard dans le code Javascript.

La seconde balise **<article>** servira à afficher :

* Une **<div>** qui affichera dynamiquement un compteur.
* La seconde affiche un paragraphe simple.
* PRESENTATION DU CSS

J’ai choisi d’utilisé **Boostrap** car c’est une **Framework CSS Open Source** qui est très utilisé par les développeurs. Il offre une collection de composants et de styles prédéfinis facilitant la création d’interface. Il permet un gain de temps conséquent.

Il permet aussi une personnalisation des éléments afin de pouvoir l’adapter à nos besoins.

Vous trouverez le code personnalisé dans l’Annexe [INDEX ANNEXE A INFORMER]

* PRESENTATION DU JAVASCRIPT

Dans cette partie je vous présente comment j’appelles les différents éléments HTML dont j’ai besoin dans le code Javascript ainsi que toute la logique pour le renvoi des informations.

Ce code a pour but d’afficher 2 compteurs en « temps réel » lorsque l’utilisateur cliquera sur la mappemonde et 1 compteur qui s’affiche sous la carte et se lance au chargement de la page.

[screen 5-JS-init const] Je commence par initialiser plusieurs variables en récupérant les élément du **DOM (Document** ) la méthode **parseInt** permet de transformer une chaine de caractère en un entier.

La variable « tInstant » prend la méthode **New Date()** (ligne 26) qui permet d’instancier la date du jour. En suivant je récupère l’année actuelle dans la variable « currentYear » grâce à la méthode let **getFullYear().** Une fois l’année récupérer, j’instancie une nouvelle date au 1er janvier de l’année en cours avec une new Date qui prendra en paramètre l’année (currentYear), le mois (0 *ndlr => index de la première valeur d’un tableau)* et le jour (1).

Nous continuons sur l’initialisation des chiffres par pays regroupé au format **objet (nom de la variable : { clé : valeur } )** . La logique est la même pour chaque pays instancié. Les chiffres sont obtenue grâce au tableau des émissions transmit par l’ ADEME [ANNEXE A INDEXE]

[screen 6- JS ]

Pour finir, j’instancie 2 variables qui serviront un peu plus tard.

L’écouteur d’évènement window.**addEventListener()** (ligne 151) va permettre de lancer les fonctions **loopFr** qui prend en paramètre les chiffres par pays instancier auparavant) et **emSinceNYear** qui vont lancer 2 compteurs distinct au chargement de la pages **(load)**

Un second écouteur d’évènement (ligne 157) sur l’objet map est mis en place. Celui-ci ce déclenchera au clic **(click)** sur l’objet **map** *( ndlr : on retrouve ici l’intérêt de la mappemonde en SVG)* il prend en paramètre l’évènement **(evt)** qui sera utiliser pour cibler les éléments cliqués comme dans l’instanciation de la variable cible qui récupère grâce à la méthode .target (Ligne 159) l’id de l’élément (evt) cliqué.

(Le console.log ici affichera cible => ex : // FR)

Avant de lancer les compteur je réinitialise l’ensemble des fonctions de compteur pour les remettre à 0 avec la méthode **clearInterval()** et vide les élément du DOM pour avoir des paragraphe vide (ligne 166 – 167 -168)

Une fois que le formatage est fait, je (re)lance les 2 fonctions compteurs avec en paramètre les informations du pays concerné (récupérée par la variable cible) grâce à un **switch**

[screen 7 - js ] Toujours dans notre écouteur d’évènement, je récupère les coordonnées x et y du curseur de la souris (ligne 236 -237) grâce à la méthode **clientX/Y** et **scrollX/Y**  qui me permet en suivant d’afficher la div « tooltip » à l’endroit précis du clic. La condition **if/else** (ligbe 240) permet, quant à elle d’afficher ou d’enlever la div du DOM

Maintenant que nos fonction sont appeler dynamiquement sur le DOM, je crée mes fonctions :

La fonction **cumulUser** (ligne 254) qui prend 2 paramètres en compte permet d’afficher le compteur d’émission de CO2 par habitant. Je commence par instancier des variables qui ne serviront que dans le scope de la fonction (*ndlr : c’est-à-dire qu’elles ne peuvent pas être appeler en dehors de la fonction*)

J’ai décidé, pour le compteur de détailler le grammage pour avoir une interaction plus intéressante pour l’utilisateur car les chiffres étant extrêmement petit (on parle, par exemple de 0.2gr de CO2 /sec pour un Français).

La fonction renvoi dynamiquement, grâce à la méthode **setInterval**, à la div récupérée dès le départ grâce à la variable **showResultPerson,** le résultat du compteur chaque seconde..

La fonction carbonCountry (ligne 280) fonctionne sur le même principe, a défaut qu’ici, les chiffre sont bien plus élevé (ex : 13.6 To de CO2/sec pour la France) donc je divise par 10 la quantité de C02 étant donné que je renvoi le résultat à **showResultCountry** toutes les centièmes de secondes.

La fonction **loopFr**, lancé au chargement de la page, reprend exactement le même schéma que **cumulUser**.

La fonction emSinceYear, quant à elle renvoi les émissions de CO2 depuis le début de l’année :

J‘instancie le taux à 0 (ce qui correspond aux émission le 1er janvier à 0h00). Ensuite, je calcule le nombre de jour écoulé depuis le début de l’année qui est données en milliseconde je le convertie en jour grâce au calcul **parseInt((tInstant.getTime() – startYear.getTime())/1000/3600/24 )**

Je calcul le taux de C02 rejeter grâce au calcul (ligne 327) que je retourne grâce à une incrémentation dans un **setInterval** initialisé à la seconde (ligne 328).

[screen resultat final] Dans les annexes [ANNEXES A INDEXER] le résultat final au chargement de la page et au clic sur un pays.

* PRESENTATION DU CODE BACK END

Dans cette partie, je vous présente la messagerie interne au projet, comment un utilisateur peut envoyer un message à un utilisateur et comment ce dernier peut y répondre.

Pour réaliser cette fonction du site, j’ai eu besoin de 4 fichiers distinct :

* **Messages.php** : fichier basé dans les **Model**, il va servir à la logique métier et va permettre la relation avec la base de données.
* **messagerieController.php** : fichier basé dans le **Controller**, il va permettre de gérer les requêtes de l’utilisateur. Il sera le lien entre les fichiers **Vue** et **Model**.
* **vueMessagerie.php** et **vueReception.php** : fichier basé dans la **Vue**, ils vont permettre l’affichage sur le navigateur des élément nécessaires pour l’utilisateur. Ces fichiers, essentiellement basé sur du Front-End seront présentés dans les annexes [ANNEXE A PRECISER]

* Messages.php

[Screen 1 Model messagerie]

Le **namespace** App/Model va permettre d’organiser et de regrouper les classes. Cette instanciation va permettre de faire comprendre au système le chemin à utiliser pour utiliser les classes.

Je continue en utilisant la méthode use. Cette méthode va me permettre de faire des appels à d’autre fonction se situant dans d’autre fichier. Ici j’utilise la fonction déportée de la connexion à la base de données : **App\utils\BddConnect** (Annexe [A PRECISER])et j’appelle le Model User qui me sera nécessaire pour référencer l’expéditeur et le réceptionnaire du message : **App\Model\Users**.

Une fois les appels recensés je créer la class Messages qui héritera des méthodes de la class BddConnect grâce à la méthode **extends**.

Cette notion d’héritage permet la réutilisation des méthodes publiques créés dans la classe parente comme si ces dernières avaient été créer au sein de la classe enfant.

Je crée ensuite les attributs qui me seront nécessaire dans la class Messages. Ces attributs font références aux colonnes de la tables messages en base de données et sont déclaré en **private** pour informer le système qu’ils doivent être encapsuler au sein de la class et ne doivent pas en sortir.

J’initialise un **constructeur** qui va me permettre de crée un nouveau sender ou reciper, grâce à la configuration de la class User.

[Screen 2 ou 3] Je continue sur la création des **getter** et **setter** de l’ensemble de mes **attributs** en mode public.

Les getter et setter sont des fonctions qui vont permettre **la récupération, la manipulation et la modification** des données d’attributs de la class. Grâce au typage des function **:int / :void / :string,** on informe le système quel type de données il est censé recevoir ou retourner,ça permet un renfort de la sécurité des données et de documenter le code (si toutefois une autre personnes est amené à intervenir sur le code) .

La suite des getter et setter sont représenter Annexe [ANNEXE A PRECISER]

Une fois que j’ai fini d’instancier l’ensemble des mes attributs, je peux créer mes méthodes.

Les méthodes sont des fonctions qui vont servir de logique pour la lecture et l’écriture des données dans la base de données.

Pour, mon explication, je vais vous présenter la méthode d’insertion de message en base de données. Toutes les autres méthodes de cette class sont présentées Annexe [ANNEXES A PRECISER]

[screen 4] Dans cette fonction qui permet l’enregistrement d’un message en base de données (et qui ne promet rien en retour **:void**), j’utilise un bloc ‘**try-catch’.** Ce bloc va permettre de prévenir des potentielles erreurs qui peuvent survenir pendant l’exécution du bloc de code.

Dans le bloc « **Try** », je vais placer l’ensemble du code qui peut amener des erreurs. Si une erreur se produit, l’exécution du programme passe immédiatement au bloc « **Catch** » et exécute le code qui y est placé, ici (et dans la major partie du temps) un message d’erreur avec la nature de l’erreur qui permettra une meilleure résolution.

Le **die**, quant à lui, va permettre l’arrêt immédiat de la fonction.

Dans le bloc **Try**, je commence par stocker les données récupéré (*ndlr : dans le formulaire côté front*) dans des variables locales faisant références à mes attributs de class.

Une fois fais, j’instancie une requête **$req** qui va appeler **connexion()** (*ndlr : fonction hérité de la class BddConnect*) et **prepare** la requête SQL .

Afin de passer mes données stockées dans les variables à la base de données via ma requête SQL, j’utilise bindParam qui va permettre le traitement des données de manières sécurisé et flexible ce qui est accentuer par l’argument **\PDO ::PARAM\_STR** qui indique ma nature de la donnée, cela va permettre de prévenir des éventuelles injections SQL et garantir le traitement correct des valeurs par la BDD.

Une fois les données « bindées », j’exécute le bloc grâce à la méthode **$req->execute()**

Et voilà, nos données sont enregistrées dans la base de données.

* messagerieController.php

Après avoir décrit comment passer les données à la BDD, je vais maintenant vous décrire comment les données sont envoyer à la class **Messages**

[screen 11 messagerie] Je commence par instancier mon **namespace**, les **use**. Le fonctionnement est exactement le même qui celui décris plus tôt.

La classe **messagerieController** étend (**extends**) la class Messages afin de pouvoir accéder à l’ensemble des getter et setter public.

$error = ‘’ permet d’instancier une variable **$error** qui pourrait être nécessaire pour afficher une erreur sur le navigateur de l’utilisateur.

LA variables $users va, elle, me permettre de récupérer l’ensemble des utilisateurs inscrit en BDD grâce à la function **getAllUsers**() présentes dans le model **Users.**

Cette fonction est un ensemble d’imbrication de **If ..Else** afin de permettre de gérer un ensemble de cas de figures et de pouvoir répondre le cas échéant.

Pour commencer les variables **$\_POST[‘nom\_de\_linput\_côté\_front’]** permet d’indiquer en PHP que nous somme sur une méthode **POST** (qui envoie des informations).

Je débute par vérifier si le bouton est bien définie grâce à la méthode **isset(),** cette méthode va renvoyer **true** si la variable est définie (ici si le bouton est bien cliqué). Dans le cas contraire, elle renvoie **false** et rien ne se passe.

Une fois cette première étape passée, je vérifie avec **!empty($\_POST[« reciper »]** que l’ensemble des champs ne soit pas vides. Si le cas se présente, le programme sort de la condition et renvoi vers le **else**, qui affichera, grâce à **$error,** un message d’erreur, ici « Merci de remplir les champs ».

Si tous les champs sont bien remplis, le programme va maintenant procéder à 2 vérifications que le destinataire est bien différents du champs « pseudo » (*ndlr : pseudo est la valeur du champ vide*) **$\_POST[‘reciper’] != « pseudo »** et qu’il est différent de la variable de login **$\_POST[‘reciper’] != $\_SESSION[‘id’]**. Si la condition n’est pas rempli, le programme renvoi un message d’ereur, ici : « Veuillez renseigner un destinataire »

Si le destinataire est bien différent, le programme continue et vient nettoyer les champs grâce à la fonction **Utilitaire ::CleanInput** (présenter à l’Annexe [A PRECISER])

Une fois les champs nettoyés, je passe aux différents attributs de la classe Messages, les variables récupérer dans le formulaire (**$this->getSender()->setId($\_Session[‘id’]**) et je lance la fonction **insertMessages** ( **$this->insertMessage()** ) je profites de la variable  «error »  pour afficher un message de succès.

[SCREEN RENDU FINAL] Le rendu final est en Annexe [ANNEXE A PRECISER]

CONCLUSION

Ce projet m’a permis d’approfondir l’ensemble de connaissances que j’ai pu acquérir tout au long de ma formation au sein du Pôle Numérique de l’ADRAR et m’a aussi conforté mes choix de reconversion professionnelle.

Je connaissais déjà mes sensibilités concernant la partie Front End mais au plus le projet avancé, au plus je découvrais la partie Back End et, de ce fait mon appétence pour ce langage qui me plait aussi énormément.

En relisant mon code fait il y a plusieurs mois, et mes connaissances actuelles, je me rends compte que plusieurs parties peuvent être optimisé, que ce soit en termes de lisibilité, de simplicité ou de choix de méthode, et c’est aussi ça qui me plait dans ce métier car rien n’est figé et tout peut-être optimisé.

Mon stage m’aura, aussi apporté énormément de compétences grâce aux nouvelles technologies découvertes mais aussi par le travail en équipe et les difficultés que cela pourrait amener.

J’ai décidé de continuer sur la formation de Concepteur Développeur d’Application afin de pouvoir continuer à apprendre les différents langages informatiques. Cette formation, je souhaite la faire en Alternance afin de pouvoir aussi monter en compétence au sein d’une entreprise qui pourra me tutorer tout au long de l’année.