1 – INTRO

* PRESENTATION

Je m’appelle Christopher Chiarandini. J’ai fait des études dans le commerce, j’ai obtenu mon BAC STT Action et Communication Commerciale, et suis entré dans la vie active.

Après différents petits jobs, j’ai occupé un poste de commercial sédentaire pour une filiale automobile durant 2 ans, puis un emploi de courtier en formation professionnelle pour une start up basée à Paris.

Quelques années plus tard, j’ai quitté la région parisienne pour m’installer à Toulouse où j’ai travaillé dans l’agro-alimentaire jusqu’à évoluer au poste de 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 dans ce domaine d’activité que je souhaite exercer.

Après quelque temps de réflexion et n’étant plus en phase avec mon poste dans l’agro-alimentaire, j’ai décidé de tout arrêter pour suivre cette voie. Quelques mois de recherche plus tard, j’ai trouvé la formation de Développeur d’Application Web et Web Mobile proposée par l’ADRAR et ai donc passé les tests afin d’intégrer la session de Juillet 2023.

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

* REMERCIEMENT

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

Je tiens également à remercier l’ensemble de l’équipe des formateurs pour l’ensemble des connaissances qu’ils nous ont apportées.

Je remercie les membres de ma promotion pour les moments d’échanges et de qualité passés ensemble tout au long de ces 10 mois.

Je remercie chaleureusement Florian CHEVALLIER, mon maître de stage durant ces 2 mois chez ATECNA pour le temps qu’il m’a accordé. Il m’a transmis énormément d’informations et m’a beaucoup aidé à appréhender les bonnes pratiques en entreprise et une vision optimale de la conception d’un projet. Merci aussi à l’ensemble de l’équipe pour son accueil et tous les éléments qui m’ont été partagés lors de ce stage.

Pour finir, merci à Julie CHIARANDINI BOLIOLI, mon épouse, pour tout le soutien qu’elle m’a apporté durant les phases de doutes 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 avancer 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, énergies fossiles etc…)

La fast-fashion et la consommation de masse ne sont pas en reste en ce qui concerne leur empreinte carbone, 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 ludique.

Il tournera sur 3 axes majeurs :

* **Sensibiliser**grâce à l’apport de chiffres recensés par le DATALAB Édition 2023 du Ministère de la Transition Écologique,
* **Informer** les utilisateurs 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 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’adapter à tous les supports, il sera donc responsive.

Par souci d’inclusion de 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 utilisée par les designers.

Elle permet de concevoir des projets 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 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’autres.

Serveur Local et Conception SQL

[Logo XAMPP] XAMPP 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 tester 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 Bases de Données et d’administration MySQL. Il permet de générer du code SQL pour créer ou modifier des bases de données relationnelles. Il permet également de gérer des connexions de bases 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 agencer le contenu d’une page web. Il rend possible la structuration logique du contenu textuel d’une page grâce à un système de balises (< >), hiérarchisant ainsi les éléments tel que les titres, les textes, les images, …

[Logo CSS] CSS (Cascading Style Sheets) : Ce langage, basé sur des feuilles de styles, permet de mettre en forme et de styliser le code HTML. Il va permettre de définir l’apparence visuelle des éléments comme la couleur, la taille et la disposition grâce à l’application de règles de styles à l’aide de sélecteurs et de propriétés CSS.

[Logo Javascript] Javascript : C’est un langage de programmation côté client utilisé afin de rendre les pages interactives et dynamiques. 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 pour se connecter et 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’applications 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 la manière dont 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 système 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 les relations 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 un cas d’utilisation spécifique et un cas d’utilisation plus général.

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

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 système d’information.

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

Exemple 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 vers le cas étendu avec la mention <<extend>>

* DIAGRAMME D’ACTIVITE

Le diagramme d’activité est un 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.

Généralement, il représente un seul scénario mais il peut arriver qu’il en représente plusieurs en fonction de la complexité du dit 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 : représente les différentes tâches de chaque étape du processus.

[screen nœud de décision] Le Nœud de décision : C’est un point de décision dans le processus où le flux d’activité 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’activité 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 système.

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

Dans le présent mémoire, 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.

À contrario 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 l’état de l’objet et ses interactions avec les autres objets au fil du temps, ce qui va aider à la compréhension du déroulement du scénario d’utilisation.

[screen message synchrone] Le message synchrone est un 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 selon des conditions spécifiques qui diffèrent du scénario initial. Il permet de prendre en compte les différentes possibilités et les cas exceptionnels dans le comportement du système en offrant une vue complète et détaillée des interactions.

[screen scenario alternatif] Dans cet exemple, le Si informe que la discussion n’existe pas. L’utilisateur peut 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 ont lieu pendant le scénario principal. Il est utilisé pour modéliser les réponses du système à des situations exceptionnelles ou à des conditions d’erreur telles que les erreurs de traitement, les cas non gérés ou les défaillances matérielles

[screen scenario d’erreur] Dans cet exemple, le scénario d’erreur indique qu’une défaillance liée au serveur s’est produite. Un 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 défini par le fait que le système suit un **chemin normal** sans rencontrer 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 clé 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 composent 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 structurée 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** (chaîne de plus de 255 caractères), data\_message de type **DATATIME** (qui renvoie la date et l’heure) **sender** et **recipient** qui sont de type **INTEGER** (un nombre), id \_talk de type **VARCHAR(100)** (chaîne de 100 caractères) ainsi que read\_messages et is\_hidden\_messages de type **TINYINT(0)** (un type de booléen qui renvoie 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 permettra de mieux comprendre la relation entre les tables.

**Les cardinalités** : Les cardinalités définissent le nombre d’occurrences d’une entité qui peuvent être associées 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 à, 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 message (0) mais a pu aussi en écrire plusieurs (n).

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

[screen du MCD]

* MLD

Le MLD (Modèle Logique de Données) est une représentation structurée des données d’un système où les entités, attributs et relations sont spécifiés de manière détaillée sous forme d’un schéma des bases de données relationnelles.

A contrario du MCD qui se concentre sur les concepts et les relations entre les 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 donner plusieurs informations comme :

* les clés primaires : représentées par des **attributs de couloir noire, en gras et surlignés**, elles identifient de manière unique chaque occurrence de l’entité dans une table de la base de données,
* les clés étrangères : représentées par *des attribut de couleur bleue en italique*, elles identifient, quant à elles, des clés primaires d’une 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 schéma :

* 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’une autre entité,
* une flèche pointant vers l’intérieur d’une entité signifie qu’une clé étrangère dans une autre entité est liée 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 la manière dont les données sont organisées 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 représentations graphiques ou interactives 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 simplifiée de l’interface utilisateur (UI), il se concentre principalement sur la structure et la disposition des éléments de l’interface sans se soucier, des détails esthétiques (couleurs, images, polices, etc..). Il va permettre d’architecturer l’information, la navigation, les zones fonctionnelles et les interactions de base donnant une fondation solide 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 éléments 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 esthétiques comme les couleurs (reprise de la charte graphique), les polices, les images et les éléments graphiques.

Il va servir afin à 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 pour le projet, j’ai utilisé le logiciel Workbench.

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

Pour me permettre d’accéder à mon administration, j’utilise en parallèle l’outil XAMPP avec lequel je crée une connexion à 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 celle 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 placer 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 les parenthèses, nous allons pouvoir insérer nos différents attributs, ici :

* **Id\_messages INT AUTO\_INCREMENT NOT NULL PRIMARY KEY** crée un attribut qui prend différents paramètres **INT** pour indiquer qu’il s’agira d’un entier, **AUTO\_INCREMENT** pour indique que la colonne id\_messages doit doit insérer à 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 **TEXT** qui correspond à une chaine de caractères de plus de 255 caractères.
* **date\_messages** **DATETIME** crée un attribut de type **DATETIME** qui insère dans la BDD une date et une heure.
* **sender** **INT** et **recipient** **INT** créent des attributs de type **ENTIER** (ils représentent les foreign key)
* **read\_messages TINYINT(1) DEFAULT 0** et **is\_hidden\_messages TINYINT(1) DEFAULT** **0** créent un attribut de type **TINYINT** c’est-à-dire un petit entier, **DEFAULT 0** indique que par défaut l’entier sera 0 (nous nous serviront de cet 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 respectées 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 créé sous le modèle **MVC (Model View Controller).** C’est un type d’architecture logicielle qui divise une application en trois composants interconnectés que je détaillerai 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 models 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 réutilisables de façon indépendante du model, du view ou du controller. 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 et donc d’éviter de 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 : l’ensemble des images, des fichiers **CSS** (pour l’habillage de l’application) ainsi que le fichier **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’urls, 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 informations envoyées au navigateur de l’utilisateur.

* La commande **namespace** va me permettre d’instancier le chemin de mon 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 renvoie :
  + le fichier contenant la navigation (navbar.php),
  + le titre de la page,
  + le fichier contenant la partie informative de la page (vueChiffre.php),
  + le Footer de la page (footer.php),
  + l’initialisation de la variable d’erreur, au cas où il y ait une erreur a afficher à l’utilisateur,
  + les fichiers javascript et css.

[screen 1-HTML-Head] La première partie est envoyée au Template qui me sert de page HTML. Il est composé en deux 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 supplémentaires sur la page. Dans notre cas, je lui passe plusieurs données :
  + Le **charset** qui va permettre d’informer sur l’encodage des caractères, UTF-8 est le plus utilisé car il prend en charge la plupart des caractères Footeret langues les plus parlées dans le monde.
  + Les balises **<link>** vont permettre d’initialiser les fichiers CSS dont j’aurai besoin, ici je délivre le CDN de Bootstrap qui me permettra d’utiliser l’ensemble des outils proposés par ce Framework. On peut remarquer une boucle **foreach** qui appelle des fichiers, ici 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 passé 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 trois balises qui sont passées par l’intermédiaire du Controller :
  + Les balises < ?= $navbar ?> et < ?= $footer ?> sont présentées en Annexe (NUMERO ANNEXE A INDEXER)
  + La balise < ?= $content  ?> est présentée ci-dessous.

À 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 deux balises **<article>**.

La première va servir à afficher :

* Une **<div> « Tooltip »** qui, au chargement n’est pas visible mais sera appelée grâce à du code Javascript que nous verrons ultérieurement. Cette balise comporte des paragraphes dans lesquels des informations seront injectées 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’informations au sein même de l’image. Ici je récupère 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’utiliser **Boostrap** car c’est un **Framework CSS Open Source** 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 les adapter à nos besoins.

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

* PRESENTATION DU JAVASCRIPT

Dans cette partie je vous explique comment j’appelle 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 deux compteurs en « temps réel » lorsque l’utilisateur cliquera sur la mappemonde et un 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 éléments 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 récupérée, 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).

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

[screen 6- JS ]

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

L’écouteur d’évènements window.**addEventListener()** (ligne 151) va permettre de lancer, au chargement de la page **(load),** la fonction **loopFr** et la fonction **emSinceNYear,** qui prennent en paramètres les chiffres par pays instanciés auparavant, ce qui aura pour résultat de lancer deux compteurs distincts sur le navigateur de l’utilisateur

Un second écouteur d’évènements (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 utilisé 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 compteurs je réinitialise l’ensemble des fonctions de ces derniers pour les remettre à 0 avec la méthode **clearInterval()** et vider les éléments du DOM pour avoir des paragraphes vides (ligne 166 – 167 -168).

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

[screen 7 - js ] Toujours dans notre écouteur d’évènements, 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** (ligne 240) permet, quant à elle d’afficher ou d’enlever la div du DOM.

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

La fonction **cumulUser** (ligne 254) qui prend deux paramètres en compte, permet d’afficher le compteur d’émissions 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 appelées 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 sont extrêmement petits (on parle, par exemple de 0.2gr de CO2 /sec pour un Français).

La fonction renvoie 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, à ceci près qu’ici, les chiffres sont bien plus élevés (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** tous les centièmes de secondes.

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

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

J‘instancie le taux à 0 (ce qui correspond aux émissions le 1er janvier à 0h00). Ensuite, je calcule le nombre de jours écoulés depuis le début de l’année, donné en millisecondes, je le convertis en jours grâce au calcul **parseInt((tInstant.getTime() – startYear.getTime())/1000/3600/24 )**

Je calcule le taux de C02 rejeté 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 autre utilisateur, et comment ce dernier peut y répondre.

Pour réaliser cette fonction du site, j’ai eu besoin de quatre 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** : fichiers basés dans la **Vue**, ils vont permettre l’affichage sur le navigateur des éléments nécessaires pour l’utilisateur. Ces fichiers, essentiellement basés 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’autres fonctions se situant dans d’autres fichiers. 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 classe 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 créées au sein de la classe enfant.

Je crée ensuite les attributs qui me seront nécessaires dans la classe Messages. Ces attributs font référence aux colonnes de la table messages en base de données et sont déclarés en **private** pour informer le système qu’ils doivent être encapsulés au sein de la classe 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 classe 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 classe. Grâce au typage des functions **: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 personne est amenée à intervenir sur le code) .

La suite des getter et setter est représentée dans l’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 classe sont présentées en 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 majeure 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 récupérés (*ndlr : dans le formulaire côté front*) dans des variables locales faisant référence à mes attributs de classe.

Une fois fait, j’instancie une requête **$req** qui va appeler **connexion()** (*ndlr : fonction héritée de la classe BddConnect*) et **prépare** la requête SQL.

Afin de passer mes données stockées dans les variables dans la base de données via ma requête SQL, j’utilise bindParam qui va permettre le traitement des données de manière sécurisée et flexible, ce qui est renforcé par l’argument **\PDO ::PARAM\_STR** qui indique la 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 expliqué comment passer les données à la BDD, je vais maintenant vous décrire comment les données sont envoyées à la classe **Messages**

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

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

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

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

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

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

Je commence par vérifier si le bouton est bien défini 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($\_POST[« reciper »]** que l’ensemble des champs ne soient pas vides. Si le cas se présente, le programme sort de la condition et renvoie vers le **else**, qui affichera, grâce à **$error,** un message d’erreur, ici « Merci de remplir les champs ».

Si tous les champs sont bien remplis, le programme procède à deux vérifications : que le destinataire soit bien différent du champ « pseudo » (*ndlr : pseudo est la valeur du champ vide*) **$\_POST[‘reciper’] != « pseudo »** et qu’il soit différent de la variable de login **$\_POST[‘reciper’] != $\_SESSION[‘id’]**. Si la condition n’est pas remplie, le programme renvoie un message d’erreur, 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** (présentée dans l’Annexe [A PRECISER])

Une fois les champs nettoyés, je passe aux différents attributs de la classe Messages, les variables récupérées dans le formulaire (**$this->getSender()->setId($\_Session[‘id’]**) et je lance la fonction **insertMessages** ( **$this->insertMessage()** ) je profite 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 des 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 mon choix de reconversion professionnelle.

Je connaissais déjà mon attrait concernant la partie Front End, mais au fur et à mesure de l’avancement du projet, j’ai découvert la partie Back End et mon appétence pour ces langages qui me plait aussi énormément.

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

Mon stage m’aura aussi apporté énormément de compétences grâce aux nouvelles technologies découvertes au travail en équipe, y compris les difficultés que cela pourrait amener.

J’ai décidé de poursuivre sur la formation de Concepteur Développeur d’Application afin de pouvoir approfondir l’apprentissage des différents langages informatiques. Cette formation, je souhaite la faire en alternance afin de pouvoir aussi monter en compétences au sein d’une entreprise qui pourra me tutorer tout au long de l’année.