Memoire Titre développeur d’Application Web

et Web Mobile

Christopher CHIARANDINI

Change ton climat

**Table des matières**

[**Introduction**](#_bookmark0)

[**Presentation**](#_bookmark0)[**Remerciement**](#_bookmark1)

[**Analyse du besoin**](#_bookmark2)

[**Présentation du Projet**](#_bookmark2)[**Contexte**](#_bookmark2)

[**Langages utilisés**](#_bookmark2)[**Contraintes techniques**](#_bookmark3)[**Outils utilisés**](#_bookmark3)

[Logiciels](#_bookmark3)

[Serveur local et Conception SQL](#_bookmark4) [Éditeur de code](#_bookmark4)

[Langages](#_bookmark5) [**Arborescence du projet**](#_bookmark6) **Arborescence du projet**

[**Spécification fonctionelles**](#_bookmark7)

[**Use Case**](#_bookmark7)

[**Diagramme d’activité**](#_bookmark8)[**Diagramme de séquence**](#_bookmark9)

[**Conception**](#_bookmark10)

[**MCD**](#_bookmark10)[**MLD**](#_bookmark11)

[**Maquettage**](#_bookmark12)

[**SQL**](#_bookmark13)

[**Front End**](#_bookmark14)

[**Présentation du code HTML/CSS/JS : Carte intéractive**](#_bookmark15)

[HTML](#_bookmark16) [CSS](#_bookmark17)

[Javascript](#_bookmark18)

[**Back End**](#_bookmark18)

[**Messages.php**](#_bookmark18)

[**messagerieController.php**](#_bookmark19)

[**3**](#_bookmark0)

[**3**](#_bookmark0)

[**4**](#_bookmark1)

[**5**](#_bookmark2)

[**5**](#_bookmark2)

[**5**](#_bookmark2)

[**5**](#_bookmark2)

[**6**](#_bookmark3)

[**6**](#_bookmark3)

[6](#_bookmark3)

[7](#_bookmark4)

[7](#_bookmark4)

[8](#_bookmark5)

[**9**](#_bookmark6)

**10**

[**11**](#_bookmark7)

[**11**](#_bookmark7)

[**13**](#_bookmark8)

[**16**](#_bookmark9)

[**19**](#_bookmark10)

[**19**](#_bookmark10)

[**21**](#_bookmark11)

[**23**](#_bookmark12)

[**26**](#_bookmark13)

[**28**](#_bookmark14)

[**29**](#_bookmark15)

[30](#_bookmark16)

[32](#_bookmark17)

[32](#_bookmark17)

[**36**](#_bookmark18)

[**36**](#_bookmark18)

[**39**](#_bookmark19)

[**Conclusion**](#_bookmark20)[**Annexes**](#_bookmark21)

[**41**](#_bookmark20)

[**42**](#_bookmark21)

**Introduction**

**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 pro- fessionnelle pour une start up basée à Paris.

Quelques années plus tard, j’ai quitté la région parisienne pour m’installer à Tou- louse où j’ai travaillé dans l’agro-alimentaire jusqu’à évoluer au poste de Respon- sable Manager.

J’ai toujours eu un attrait pour le monde du numérique, j’ai suivi quelques forma- tions 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’Applica- tion 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 uti- lisés ou sur le travail en équipe sur un même projet

**3**

**Remerciement**

Avant de débuter ce mémoire, je tiens à remercier l’ADRAR et tout particuliè- rement 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’en- semble 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 énormé- ment 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 parta- gés lors de ce stage.

Pour finir, merci à Julie CHIARANDINI BOLIOLI, mon épouse, pour tout le sou- tien qu’elle m’a apporté durant les phases de doutes et de difficultés.

**4**

**Analyse du besoin**

**Présentation du Projet**

Pour pouvoir valider notre titre, il nous a été proposé, au début de notre formation, de 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 est un moment charnière pour l’environnement et ceux qui en dé- pendent.

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 em- preinte carbone, ce qui entraine un réchauffement anormal et rapide des températures sur l’en- semble 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’informer les utilisateurs de façon simple et les ludique sur les enjeux du changement climatique.

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 des bonnes pratiques pour réduire leurs émissions de CO2,
* **S’entraider** au travers d’un forum, qui permettra aux utilisateurs de discuter entre eux et de leur apporter des éléments de réponses.

**Langages utilisés**

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.

**5**

En ce qui concerne la partie Back End : **PHP** sera le modèle de langage le plus adapté aux be- soins. De plus, j’utiliserai **SQL** pour la gestion et l’interrogation des bases de données.

**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’acces- sibilités :

•

•

•

•

**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 utilisés**

Logiciels

Maquettage

**Figma** est une application de conception d’Interface Utilisateur (UI) et d’Ex- périence 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

**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.

**6**

**Looping** est un logiciel de modélisation Merise pour les bases de données relationnelle. Il va permettre de concevoir des bases de données à l’aide des schémas MCD (modèle conceptuel des données), MLD (modèle logique des données).

Serveur local et Conception SQL

**XAMPP** est un logiciel qui fournit un environnement de développement web local regroupant des outils comme **Apache, MySQL, PHP et Perl** permet- tant de créer, développer et tester des applications web directement sur son ordinateur sans avoir besoin d’une connexion internet et d’un serveur distant.

**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.

Éditeur de 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.

**7**

Langages

**HTML (HyperText Markup Language)** est un langage de balisage permet- tant de créer et agencer le contenu d’une page web. Il rend possible la struc- turation logique du contenu textuel d’une page grâce à un système de ba- lises (< >), hiérarchisant ainsi les éléments tels que les titres, les textes, les images, …

**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.

**Bootstrap** est un framework de développement web front-end qui offre des composants prêts à l’emploi pour la création d’interfaces utilisateur. Il facilite la conception de sites web réactifs et est basé sur HTML, CSS et JavaScript. En utilisant Bootstrap, les développeurs peuvent rapidement construire des sites web esthétiques et fonctionnels avec une mise en page adaptable.

**Javascript** 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.

**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’ap- plications web dynamiques et interactives.

**8**

**Arborescence du projet**

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, mon- trant 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.

**9**

**Spécification fonctionelles**

**Use Case**

Le **Use Case** (Cas d’Utilisation en français) est une **modélisation** qui permet de décrire les interac- tions entre les **utilisateurs (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.

Ceci est la représentation de l’**acteur** dans le schéma, ici le modé- rateur.

Ceci est un **héritage.** Le système d’héritage est une relation entre un cas d’utilisation spécifique et un cas d’utilisation plus général.

La **généralisation** permet d’hériter (partager) des droits sur les cas d’utilisa- tion d’un acteur avec d’autres acteurs. Dans le cas précis Le modérateur va récupérer l’accès aux cas d’utilisation détenus par l’acteur Commun.

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

*Ici, dans l’exemple, le modèrateur hérite des fonctionnalités de l’acteur Com- mun.*

Le **cas d’utilisation** est représenté par une bulle qui va contenir la description d’une fonctionnalité, commen- çant par un verbe à l’infinitif.

*Ex : Ecrire un commentaire*

**10**

**L’include (ou inclusion)** est une relation qui va imposer une exécution obli- gatoire d’un cas d’utilisation avec un ou plusieurs autres cas. la fléche va partir du cas d’utilisation vers le cas qui va être obligatoire.

*Exemple sur le use case : L’utilisateur peut supprimer son compte ce qui <<in- clude>> d’aller modifier son compte*

**L’extend ou extension**est une relation qui permet une exécution d’un cas d’utilisation avec un ou plusieurs autres cas. La flèche va partir du cas d’uti- lisation qui va étendre vers le cas qui va être étendu.

Exemple sur le use case :Le modérateur a possibilité de supprimer une dis- cussion quand il affiche une discussion

**11**

**Diagramme d’activité**

**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 pro- cessus.

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 :

**L’état initial** : il représente le début du processus.

**Accept Signal** : Il représente l’étape qui reçoit un signal entrant ou un évènement externe qui lui permet de pou- voir continuer son exécution, il permet de spécifier le déclencheur du cas d’utilisation

*Exemple dans le diagramme d’activité : Afficher une dis- cussion*

**Action** : représente les différentes étape du processus.

tâches de chaque

**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.

**12**

**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.

**Le Send signal** représente une action qui déclenche l’en- voi d’un signal ou d’un évènement à un autre élément du système.

**L’état final** représente, quant à lui, la fin du processus.

**13**

Dans le présent mémoire, j’ai opté de considérer que l’utilisateur est déjà connecté, et ai décidé de présenter l’affichage et l’envoi de message dans une messagerie interne.

**14**

**Diagramme de séquence**

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 :

**La Lifeline** représente l’**existence temporelle** d’un objet ou d’une entité par- ticipant 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 déroule- ment du scénario d’utilisation.

**Le message synchrone** est un message où l’émetteur at- tend une réponse du système.

**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.

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

**15**

**Le scénario alternatif** représente une séquence d’action qui se produit selon des conditions spécifiques qui dif- fèrent 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 com- plète et détaillée des interactions.

*Dans cet exemple, le Si informe que la discussion n’existe*

*pas. L’utilisateur peut choisir une scénario nominal peut reprendre.*

*autre*

*discussion*

*et le*

**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 mo- déliser les réponses du système à des situations excep- tionnelles ou à des conditions d’erreur telles que les er- reurs de traitement, les cas non gérés ou les défaillances matérielles

*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 ré- soudre le problème*

**16**

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.

**17**

**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 com- posent 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 com- préhension de leur gestion.

Définissons les différents éléments d’un MCD :

**Entité** : Elle représente un objet ou un concept du « monde réel » sur lequel on souhaite stocker des infor- mations (**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 **DATETIME** (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)

**read\_messages** et **is\_hidden\_messages** de type **TINYINT(1)** (un type de booléen qui renvoie 0 ou 1)

•

•

•

•

•

•

**18**

**Association** : Une association représente une **relation/ connexion** entre deux ou plusieurs entités. Elle sert à décrire comment les entités sont liées entre elles. **Les cardinalités** vont jouer un rôle important dans l’asso- ciation 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 mes- sages vers Users .*

*C’est-à-dire qu’un message doit être attribué , au mini- mum (1), et maximum (1) à un user.*

*Le minimum (1) correspond à la suppression en cascade*

*: si on supprime un utilisateur, tout ce qui lui sera lié sera supprimé*

*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).*

**19**

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

**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 couleur 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 identi- fient, 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**

**20**

**et la manière dont les données sont organisées en base de données relationnelles.**

Ci-dessous, la représentation du MLD du projet :

**21**

**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 or- ganiser les différents éléments fonctionnels.

*Dans cet exemple, on peut découvrir les différents em- placements choisis pour les éléménts de l’application en version mobile*

**22**

**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’inter- face** 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.

*Dans notre exemple, nous découvrons la position des éléments dans la page basée sur le zoning.*

**23**

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

Il va servir à la présentation finale pour validation auprès des clients avant la mise en dévelop- pement du projet.

*Dans notre exemple, la représentation finale de la page d’index de l’applica-*

*tion change Desktop.*

*ton*

*climat*

*dans*

*sa*

*version*

**24**

**SQL**

Afin de créer les différentes tables de la base de données dont j’aurai 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 le- quel 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.

La commande **CREATE DATABASE nw\_change\_ton\_cli-**

**mat** permet de créer une base de données 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.

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 PRI-**

**MARY KEY** crée un attribut qui prend différents para- mètres **INT** pour indiquer qu’il s’agira d’un entier, **AUTO\_ INCREMENT** pour indiquer que la colonne id\_messages doit insérer à chaque nouvelle ligne une nouvelle valeur, **NOT NULL** informe que la colonne ne doit jamais avoir la valeur **NULL** et **PRIMARY\_KEY** informe que cette co- lonne 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 ca- ractères et jusqu’à 65535 caractères.

**date\_messages DATETIME** crée un attribut de type **DA- TETIME** 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** in- dique que par défaut l’entier sera 0 (nous nous servirons 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 indi- quant le moteur de stockage à utiliser. Ici, on utilisera In-

noDB qui permet plusieurs fonctionnalités avancées.

**25**

**La commande ALTER TABLE** messages indique quelle table nous voulons modifier , **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 lier la clé étrangère à la colonne **id\_users** de la table users **(users(id\_users))**

**26**

**Front End**

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-des- sous.

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

**App** : Ce dossier regroupe toute la logique de l’applica- tion.

**Controller** : Dans ce répertoire, on retrouve l’ensemble des fichiers qui sont responsables de la **gestion des re- quêtes des utilisateurs**, de la **coordination des interac- tions 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 **réa- gir 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 fi- chiers de codes pouvant être **réutilisables de façon in- dépendante** 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 **res- ponsables 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’applica- tion) 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 web Apache. Il est utilisé pour **définir des règles et des directives** qui affectent le comportement du ser- veur tel que les réécritures d’urls, la gestion des erreurs etc…

**Index.php** : C’est le **point d’entrée** de l’application qui contiendra le **router**.

**27**

**Présentation du code HTML/CSS/JS : Carte intérac- tive**

Dans un premier temps, j’ai initialisé le **controller chiffre- Controller** 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, **renvoie l’ensemble des informa- tions** 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 à afficher à l’utilisateur,

**les fichiers javascript et css.**

•

•

•

**28**

HTML

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 utili- sé car il prend en charge la plupart des caractères et langues les plus parlées dans le monde.

Les balises **<link>** vont permettre d’initialiser les fichiers **CSS** dont j’aurai besoin, ici je dé- livre 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>** ont le même rôle de la balise **<link>** mais pour appeler des fichiers **ja- vascript**.

Pour finir, la balise **<title>** va permettre d’afficher le titre de la page dans l’onglet du navi- gateur (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 1 et 2
* La balise **< ?= $content ?>** est présentée ci-dessous.

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

**29**

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 divisée 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 Ja- vascript 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**.

**30**

CSS

J’ai choisi d’utiliser **Boostrap** car c’est un **Framework CSS Open Source** très utilisé par les dé- veloppeurs.

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 3

Javascript

Dans cette partie je vous explique comment j’appelle les différents éléments HTML dont j’ai be- soin 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.

Je commence par initialiser plusieurs variables en récupérant les éléments du **DOM (Docu- ment)** la méthode **parseInt** permet de trans- former 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’ins- tancie une nouvelle date au 1er janvier de l’an- née en cours avec une **new Date** qui prendra en paramètre l’**année (currentYear)**, le **mois (0 *ndlr => index de la première valeur d’un ta- bleau*)** et le **jour (1)**.

Je continue sur l’initialisation des chiffres par pays regroupés au format **objet (nom de la va- riable : { clé : valeur } )**. La logique est la même pour chaque pays instancié. Les chiffres sont obtenus grâce au tableau des émissions trans- mis par l’ADEME

**31**

Pour finir, j’instancie deux variables qui servi- ront un peu plus tard.

L’écouteur d’évènements **window.addEvent- Listener()** (ligne 151) va permettre de lancer, au chargement de la page **(load)**, la fonction **loo- pFr** et la fonction **emSinceNYear**, qui prennent en paramètres les chiffres par pays instanciés auparavant, ce qui aura pour **résultat de lan- cer 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 dé- clenchera 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 **clearInter- val()** 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 para- mètres les informations du pays concerné (ré- cupérées par la variable cible) grâce à un **swit- ch**.

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 sui- vant 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.

**32**

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

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 inté- ressante 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) fonc- tionne 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 renvoie le résultat à **showResultCountry** tous les cen- tièmes de secondes.

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

**33**

La fonction **emSinceYear**, quant à elle renvoie les émissions de CO2 depuis le début de l’an- née :

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, donnés en millisecondes, je le convertis en jours grâce au calcul **par- seInt((tInstant.getTime() – startYear.get- Time())/1000/3600/24 )**

Je calcule le taux de C02 rejeté grâce au calcul (ligne 327), que je retourne grâce à une incré- mentation dans un setInterval initialisé à la se- conde (ligne 328).

Ci dessous, le résultat final de la présentation du code HTML, CSS et Javascript.

Il represente le résultat au chargement de la page(sous la carte) et au clic sur un pays.

**34**

**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 distincts :

* **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 re- quêtes 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, essen- tiellement basés sur du Front-End seront présentés dans les annexes 4 et 5 .

**Messages.php**

Le **namespace App/Model** va permettre d’orga- niser et de regrouper les classes. Cette instan- ciation va permettre de faire comprendre au sys- tème 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 fi- chiers. Ici j’utilise la fonction déportée de la connexion à la base de données : **App\utils\**

**BddConnect** (Annexe 6) et j’appelle le Model User qui me sera nécessaire pour référencer l’ex- péditeur et le réceptionnaire du message : **App\ Model\Users**.

Une fois les appels recensés je crée la class Mes- sages qui héritera des méthodes de la classe **BddConnect** grâce à la méthode **extends**.

Cette notion d’héritage permet la réutilisation des méthodes publiques créées 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 né- cessaires 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 en- capsulés au sein de la classe et ne doivent pas en sortir.

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

**35**

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 re- tourner, 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 7.

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 don- nées dans la base de données.

Pour, mon explication, je vais vous présenter la méthode d’insertion de message en base de don- nées. Toutes les autres méthodes de cette classe sont présentées en Annexe 8 à 12.

**36**

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 récupérées** *(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

**37**

**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**

Je commence par instancier mon **namespace**, les use. Le fonctionnement est exactement le même que celui décrit plus tôt.

La

classe

**messagerieController**

étend

**(extends)** la classe **Messages** afin de pouvoir accéder à l’ensemble des getter et setter pu- blics.

$error = ‘’ permet d’instancier une variable **$er- ror** 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()** pré- sentes dans le model Users.

Cette fonction est un ensemble d’imbrica- tions de **If ..Else** afin de permettre de gérer un ensemble de cas de figures et de pouvoir ré- pondre le cas échéant.

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

Je commence par vérifier si le bouton est bien défini grâce à la méthode **isset()**, cette mé- thode 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 vé- rifie 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 **« Mer- ci de remplir les champs »**.

Si tous les champs sont bien remplis, le pro- gramme 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

**38**

**$\_POST[‘reciper’] != $\_SESSION[‘id’]**. Si la condition n’est pas remplie, le programme ren- voie un message d’erreur, ici : **« Veuillez ren- seigner un destinataire »**

Si le destinataire est bien différent, le pro- gramme continue et vient nettoyer les champs grâce à la fonction **Utilitaire ::CleanInput** (pré- sentée dans l’Annexe 13)

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->in- sertMessage() )** je profite de la variable **«error**

**»** pour afficher un message de succès.

Ci dessous le resultat final côté Front.

**39**

**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 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’avan- cement du projet, j’ai découvert la partie Back End et mon appétence pour ces langages qui me plaisent aussi énormément.

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

Mon stage m’a aussi apporté énormément de compétences grâce aux nouvelles technologies découvertes, a renforcé mes soft skills grâce au travail en équipe et m’a permis de découvrir comment une agence fonctionne.

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 en- treprise qui pourra me tutorer tout au long de l’année.

**40**

**Annexes**

Annexe 1 : Navbar

**41**

Annexe 2 : Footer

Annexe 3 : CSS Personnalisé

Annexe 4 - vueMessagerie.php

Annexe 5 - vueReception.php

Annexe 6 - Class BddConnect.php

Annexe 7 - Getter et Setter (suite)

Annexe 8 - Recupération de tous les messages par utilisateur

Annexe 9 - Récupération des messages par utilisateur regroupé par id de discussion

Annexe 10 - Récupération d’un message par son id

Annexe 11 - Enregistrement d’un message par id de discussion

Annexe 12 - Suppression d’un message par id

Annexe 13 - CleanInput