

**Stage en entreprise**

Effectué du 21 Janvier 2021 au 16 Avril 2021



Société Pidev - Dury

Mission de stage : Conception et réalisation d’une application web pour l’envoi de « bon à tirer » au client.

**Maître de stage** : Mr Delrue Pierre

**SOMMAIRE**

**Remerciements** …………………………………………………………………………………………… 4

**Présentation anglais** …………………………………………………………………………………………… 5

**Présentation**

**de l’entreprise** ………………………………………………………………………………………….. 6

**Présentation**

**du projet** ………….……………………………………………………………………………….. 7

**Outils** …………..……………………………………………………………………………….. 8

**Initialisation**

**du projet** …………..……………………………………………………………………………….. 11

**Elaboration**

**de la base**

**de données** …………..……………………………………………………………………………….. 14

**Développement**

**et programmation** …………..……………………………………………………………………………….. 20

**La gestion**

**des informations** …………..……………………………………………………………………………….. 26

**Remerciements**

Je

**Présentation anglais**

**Présentation de l’entreprise**



Pidev est une agence de publicité rayonnant en local sur Amiens et sur Lille, le dénouement de compétences convergentes en communication digitale et print.

Depuis plus de 7 ans, elle initie et concrétise une ère faite d’innovations, de conseils et de solutions. Sa formule ? Le rapport constant des projets de notre clientèle diversifiée à notre force de proposition conjuguée à tous les temps.

En filigrane, les fractions de Pidev dévoilent les passions complémentaires de ses acteurs. Des passions qui riment avec imagination, valorisation et décision. Pidev, c’est aussi cette aire de liberté dans laquelle prend forme votre identité visuelle, votre message publicitaire, l’optimisation de vos impressions et de votre référencement on web. Pidev, c’est enfin ce champ des possibles toujours plus vaste. C’est l’allié de votre différence, qui transcende les réalités de vos cibles à chaque instant.

**Présentation du projet**

Mon principal objectif lors de mon stage au sein de l’entreprise fut de concevoir et d’initier une application web permettant au graphiste de communiquer avec les clients désirant un produit de l’entreprise (logo brodé sur divers vêtements, autocollants sur un véhicule, flyers, cartes de visites…). En effet, ayant comme principale activité la communication, l’entreprise se doit de fournir un travail et une finition reflétant le souhait du client au plus proche de la perfection.

Pour cela, le projet qui m’a été confié devait répondre à cette exigence.

**Outils**

Afin de concevoir le projet dans les meilleures conditions possibles, un certain nombre d’outils et de langages m’ont été utiles durant cette période de stage, voici une liste pour les présenter :

**Environnement général**



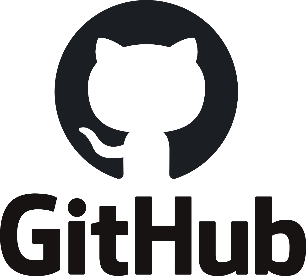
Asana est un logiciel permettant la gestion de la communication au sein d’une équipe. Nous pouvons y retrouver notamment des espaces de travail, des tâches à réaliser, des étiquettes, des notes ou encore des commentaires.

La mise à jour des informations se fait en temps réel : la planification et la gestion des projets en ressortent facilisées.



FileZilla est un client F.T.P. (File Transfer Protocol), F.T.P.S. (File Transfer Protocol Secure). C’est un logiciel libre préconisé par l’Etat français dans le cadre de la modernisation globale de ses systèmes d’informations.

Dans le cadre du stage, j’ai surtout utilisé le serveur associé : FileZilla Server, un serveur F.T.P. et F.T.P.S.



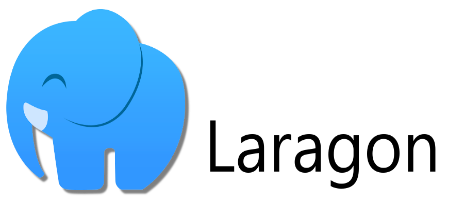
GitHub est un service web d’hébergements et de gestion de développement de logiciels.

Ce dernier a été utile durant mon stage pour tout ce qui a été enregistrement et sauvegarde du projet et m’a permis de travailler sur différentes machines.



Développé par JetBrains, IntelliJ IDEA est un environnement de développement intégré de technologie Java destiné au développement de logiciels informatiques.

Ce dernier a été un point essentiel au développement du projet en programmant toutes les fonctionnalités de l’application par ce biais.



Laragon est un serveur local qui a permis de tester l’application avant de l’envoyer sur un serveur F.T.P.

Ainsi, l’application a pu être testée et les erreurs ont pu être décelées au préalable.



phpMyAdmin est une application web de gestion pour les systèmes de gestion de base de données. Cette dernière permet d’exécuter très rapidement des requêtes et de sauvegarder les bases de données conçues.

**Langages**

Les trois principaux langages de bases de la programmation, ils sont essentiels au niveau front-end afin de structurer, d’aménager l’application et interagir avec l’utilisateur.

PHP (Hypertext Preprocessor) est un langage orienté objet (représentation d’un concept) permettant de produire des pages dynamiques via un serveur http. C’est un langage de script utilisé le plus souvent côté serveur (Back-end).

SQL (Structured Query Langage) est un langage informatique « normalisé » permettant d’exploiter des bases de données relationnelles. En d’autres termes, cela permet de rechercher, d’ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

**Frameworks**

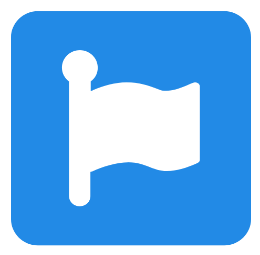
**** Bootstrap est une librairie contenant une collection d’outils utiles à la création du design des applications web. On peut y retrouver des modèles de graphismes, d’animations et d’interactions pour l’élaboration d’un projet, ou bien encore des exemples de formulaires et de boutons divers.

**Conception de projet**

draw.io est un logiciel de réalisation de diagramme en ligne permettant de visualiser le projet de façon théorique avant de procéder à la programmation.

 Looping est un logiciel de modélisation conceptuelle de données gratuit et libre d’utilisation. Ce dernier a été fortement utile pour concevoir les premières esquisses de la base de données.

**Divers**



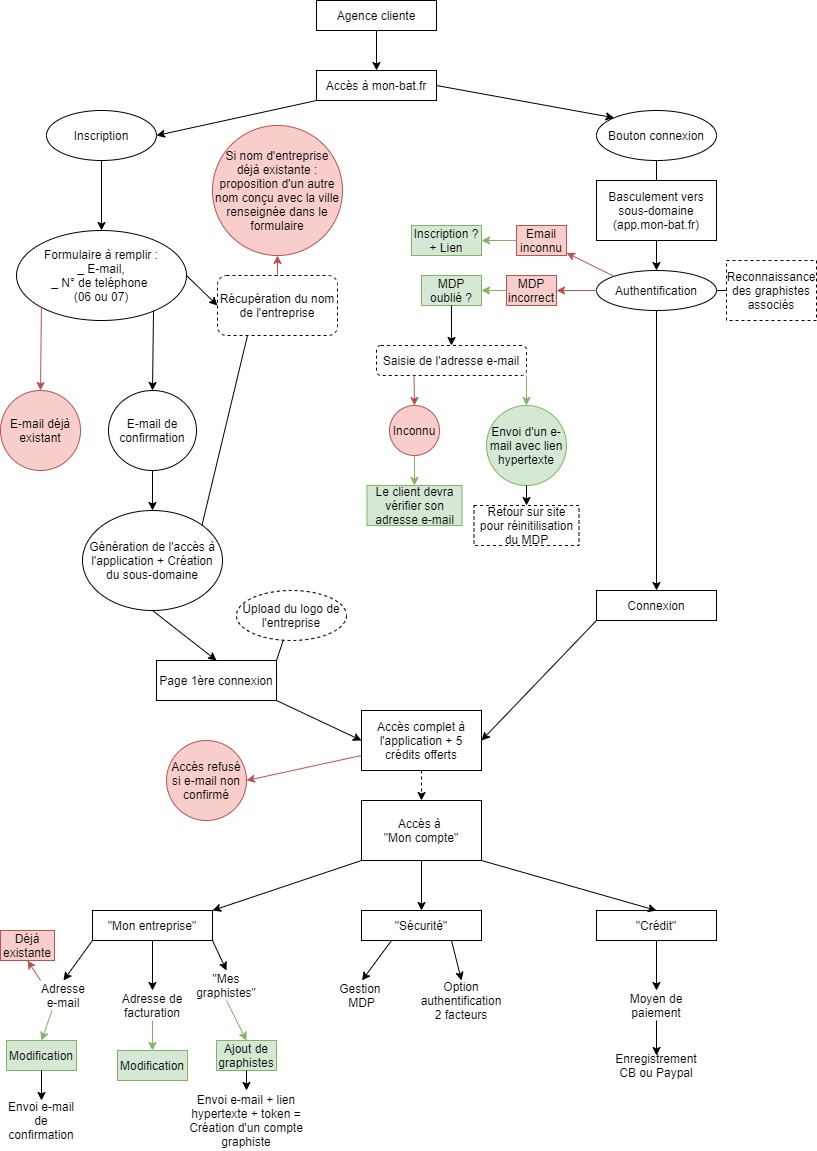
FontAwesome propose diverses icônes permettant d’illustrer une application sur les formulaires par exemple.

**Initialisation du projet**

Les premiers jours du stage ont été consacrés à la présentation du projet et à la compréhension de celui-ci.

Avec les explications qui m’ont été fournies, j’ai pu réaliser le schéma de l’utilisation de l’application web sur un tableau blanc (réalisé ensuite grâce au logiciel draw.io).

Pour la partie « Inscription / Connexion » que l’entreprise devra effectuer pour accéder à l’application web, voici la procédure imaginée :



Les parties de l’application y sont insérées et le schéma présente surtout les interactions que l’utilisateur pourrait et devra effectuer pour interagir avec l’application.

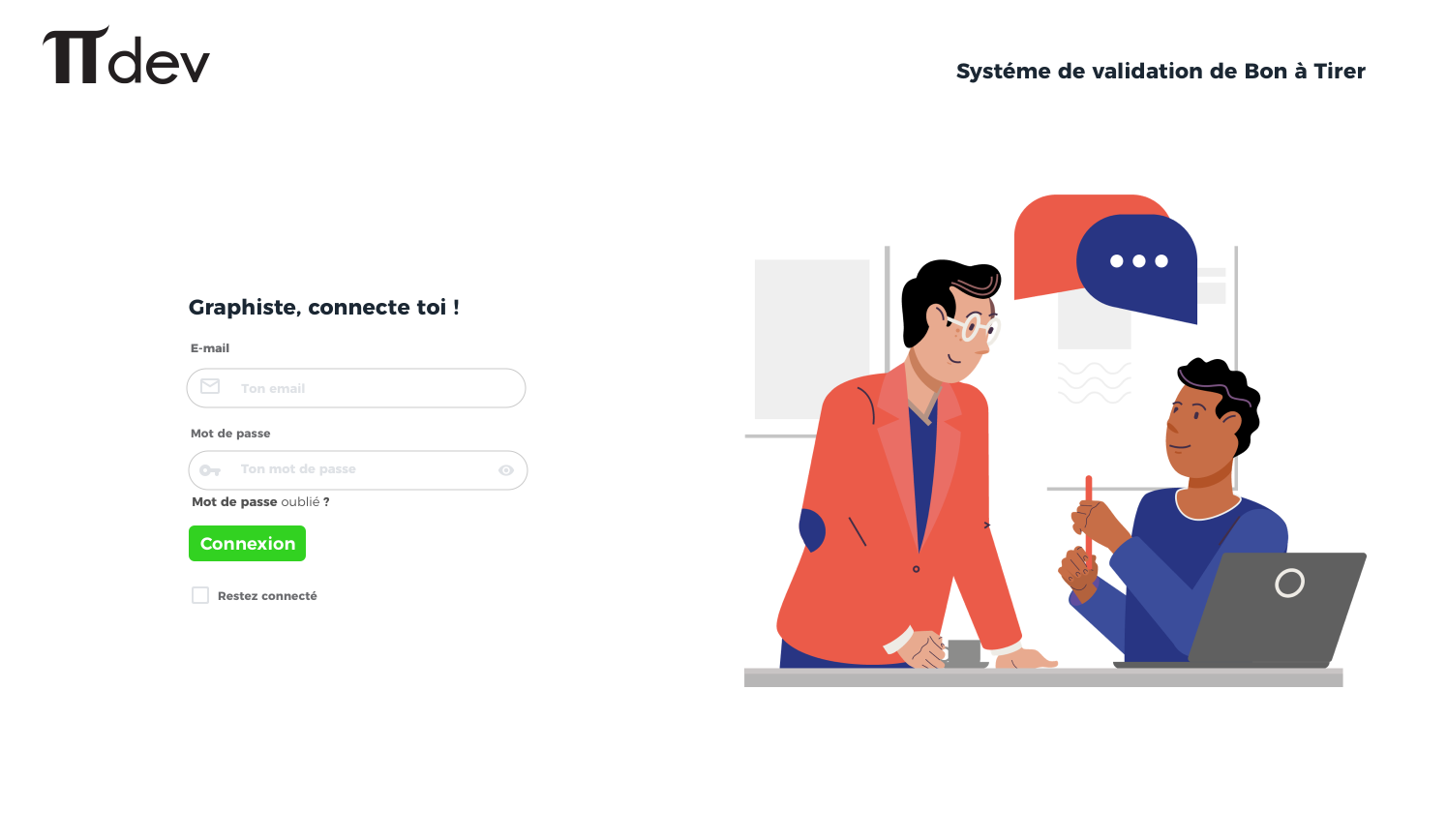
Grâce à cela, le concepteur peut visualiser de façon plus objective le projet et ainsi anticiper les différents problèmes que cela pourrait engendrer et proposer les solutions les plus adaptés pour résoudre cela (par exemple dans le cas au-dessus : nous avons pris le cas des différents cas de figures que l’authentification pourrait présenter comme un e-mail inconnu et un mot de passe oublié, ceux-ci figureront en rouge et la solution sera coloré en vert).

De là, il suffit de suivre le cheminement pour comprendre l’utilisation de l’application.

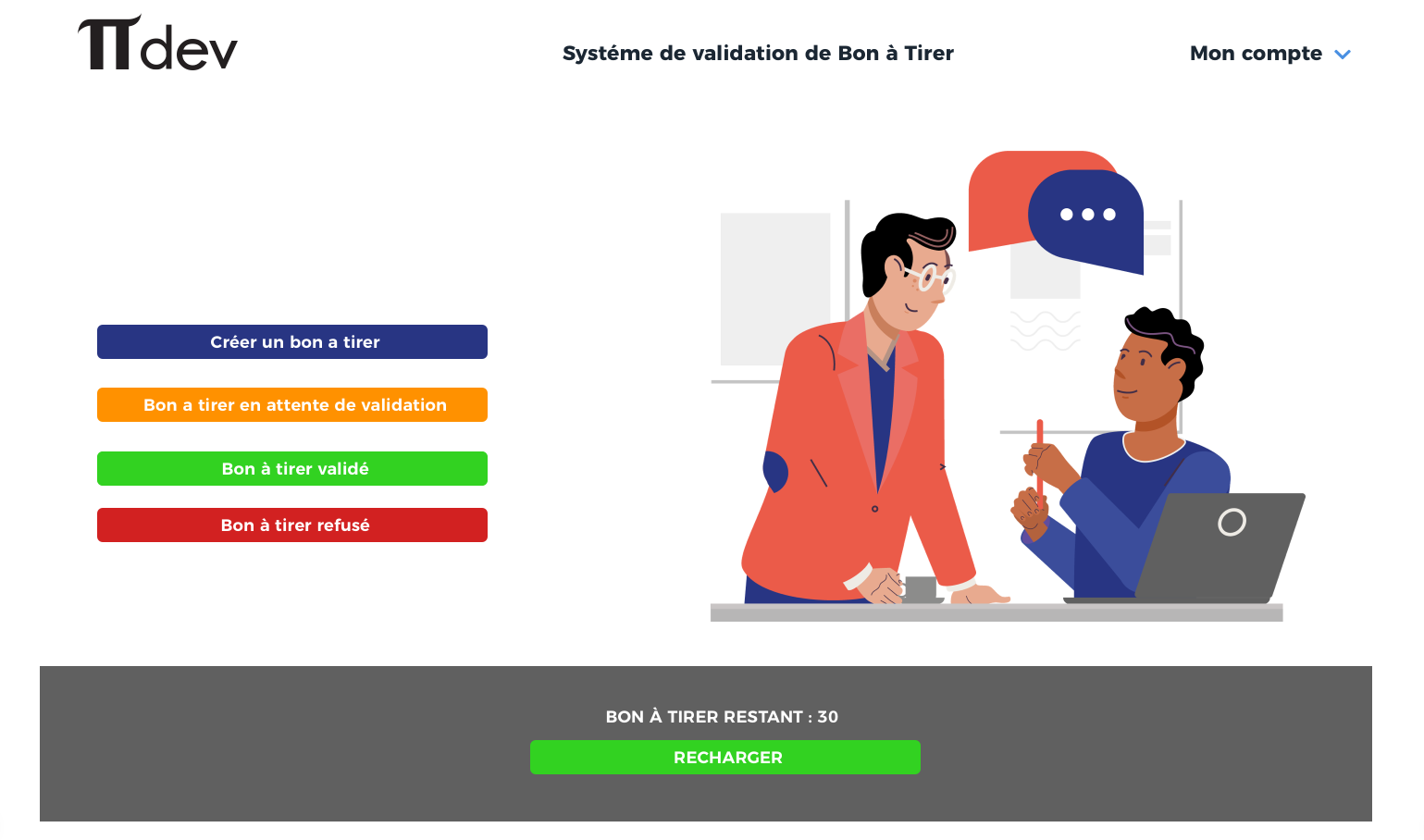
D’autres parties du projet dans ce format-ci ont été effectuées, celles-ci se trouveront en annexe de ce rapport de stage.

A partir de là, mon tuteur m’a dévoilé les premières maquettes non définitives de l’application. Celles-ci donnent un rendu non exhaustif de cette dernière est ma tâche fut de réaliser, au plus proche possible, les mêmes intégrations.

Voici par exemple ce que devrait présenter la page d’accueil du graphiste de l’entreprise voulant se connecter sur le site :



Puis ce que ce dernier pourra voir une fois l’authentification réussie :



Nous avons donc ici une partie de la maquette qui présente l’interaction entre l’application et l’utilisateur ainsi que les divers services que celle-ci devra présenter et proposer au graphiste.

Le reste de la maquette qui m’a été donnée se trouveront, elles aussi, en annexe.

**Elaboration de la base de données**

En parallèle à cette intégration, dépourvu de bases de données pourtant essentielle pour le stockage des futures informations, j’ai dressé une première esquisse de cette dernière pour le projet dont je fus en charge.

Pour cela, j’ai repris la méthode appelé « schéma entité-association » (vue et acquise lors de mes formations de Développeur Web et Web Mobile et retravaillée lors de ma formation actuelle) afin de concevoir celle-ci sous forme de schéma.

En plus de cela, et afin de réfléchir et anticiper les besoins d’informations de l’entreprise, j’ai créé un « dictionnaire de données » permettant ainsi d’alimenter les champs des différentes tables que la base de données sera pourvue.

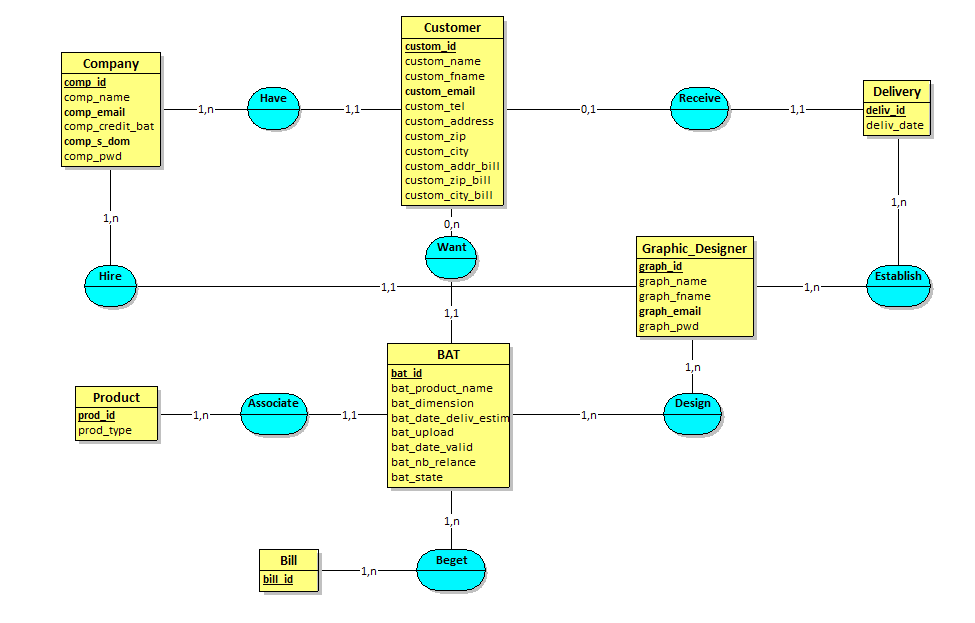
Grâce aux explications fournies lors de la présentation du projet, j’ai pu initier ces premiers prototypes :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code** | **Libellé** | **Type** | **Contraintes** | **Règles de calcul** |
| **Utilisateur (entreprise)** | | | | |
| id\_user | Numéro identifiant de l’entreprise | VARCHAR (255) | Unique |  |
| nom\_user | Nom de l’entreprise | VARCHAR (255) |  |  |
| email\_user | Adresse email de l’entreprise | VARCHAR (255) | Adresse mail conforme (@) |  |
| adresse\_user | Adresse postale de l’entreprise | VARCHAR (255) |  |  |
| cp\_user | Code postal de l’entreprise | VARCHAR (255) |  |  |
| ville\_user | Ville de l’entreprise | VARCHAR (255) |  |  |
| credit\_bat\_user | Nombre de BAT restants pour l’entreprise cliente | INT |  | BAT crédité(s) – Nombre de BAT envoyé(s) |
| **Client** | | | | |
| id\_cli | Numéro identifiant du client | VARCHAR (255) | Unique |  |
| nom\_cli | Nom du client | VARCHAR (255) |  |  |
| prenom\_cli | Prénom du client | VARCHAR (255) |  |  |
| email\_clli | Adresse email du client | VARCHAR (255) | Adresse mail conforme (@) |  |
| tel\_cli | Numéro de téléphone du client | VARCHAR (255) | Commence par 06 ou 07 pour envoi de SMS |  |
| adresse\_cli | Adresse postale du client | VARCHAR (255) |  |  |
| cp\_cli | Code postal du client | VARCHAR (255) |  |  |
| ville\_cli | Ville du client | VARCHAR (255) |  |  |
| adresse\_fact\_cli | Adresse de facturation du client | VARCHAR (255) |  |  |
| cp\_fact\_cli | Code postal de facturation du client | VARCHAR (255) |  |  |
| ville\_fact\_cli | Ville de facturation du client | VARCHAR (255) |  |  |
| bat\_cli | Le Bon à Tirer relié au client de l’entreprise | VARCHAR (255) |  |  |
| **Graphiste** | | | | |
| id\_graph | Numéro identifiant du graphiste | VARCHAR (255) | Unique |  |
| nom\_graph | Nom du graphiste | VARCHAR (255) |  |  |
| prenom\_graph | Prénom du graphiste | VARCHAR (255) |  |  |
| email\_graph | Email du graphiste | VARCHAR (255) |  |  |
| bat\_graph | Bon à tirer relié au graphiste | VARCHAR (255) |  |  |
| **Livraison** | | | | |
| id\_livr | Numéro de livraison | VARCHAR (255) | Unique |  |
| date\_livr | Date à laquelle le BAT a été livré | DATETIME |  |  |
| **Bon à Tirer** | | | | |
| id\_bat | Numéro identifiant du BAT | VARCHAR (255) | Unique |  |
| nom\_produit\_bat | Nom du produit associé au BAT | VARCHAR (255) |  |  |
| dimension\_bat | Dimension du BAT (H x L) | VARCHAR (255) |  |  |
| date\_livr\_est\_bat | Date de livraison estimée par le graphiste | DATETIME |  |  |
| upload\_bat | Illustration du BAT à tirer à présenter au client | VARCHAR (255) | PDF ou JPG  Pas plus de 100 Mo |  |
| date\_validation\_max\_bat | Date de validation maximale pour le client | DATETIME |  |  |
| nbr\_relance\_bat | Nombre de relances envoyée(s) au client pour validation | INT |  |  |
| **Facture** | | | | |
| id\_fact | Numéro identifiant de la facture | VARCHAR (255) | Unique |  |
| bat\_fact | BAT associé à la facture | VARCHAR (255) |  |  |

C’est ici la première version du dictionnaire de données établie pour initier la base de données. Bien entendu, cette dernière a reçu d’autres champs selon les besoins de l’application et de l’entreprise.

A partir de cela, j’ai pu commencer à établir le « schéma entité-association » cité précédemment. Voici comment ce dernier s’est présenté :

**Schéma Entité / Association**



Cette méthode m’a été forte utile pour appréhender les besoins de l’entreprise par rapport aux clients potentiels qui utiliseront cette application.

En effet, cette dernière permet d’utiliser les cardinalités, permettant par la suite d’associer clés primaires aux clés étrangères de façon logique.

Par exemple, et avec l’aval de l’entreprise, il a été décidé qu’une entreprise *(Company)* disposerait d’un ou plusieurs *(1,n)* graphistes *(Graphic\_Designer)* mais qu’un graphiste ne serait employé que par une et une seule *(1,1)* entreprise.

A partir de cela, le script SQL permettant de créer la base de données sur le gestionnaire de base de données devenait plus facile à construire.

En prenant pour exemple la table « *customer* », son script se composera ainsi :

CREATE TABLE customer(

custom\_id INT AUTO\_INCREMENT,

custom\_name VARCHAR(255) NOT NULL,

custom\_fname VARCHAR(255),

custom\_email VARCHAR(255) NOT NULL,

custom\_tel VARCHAR(10) NOT NULL,

custom\_address VARCHAR(255) NOT NULL,

custom\_zip VARCHAR(20) NOT NULL,

custom\_city VARCHAR(255) NOT NULL,

custom\_addr\_bill VARCHAR(255) NOT NULL,

custom\_zip\_bill VARCHAR(20) NOT NULL,

custom\_city\_bill VARCHAR(255) NOT NULL,

comp\_id INT NOT NULL,

PRIMARY KEY(custom\_id),

UNIQUE(custom\_email),

FOREIGN KEY(comp\_id) REFERENCES Company(comp\_id)

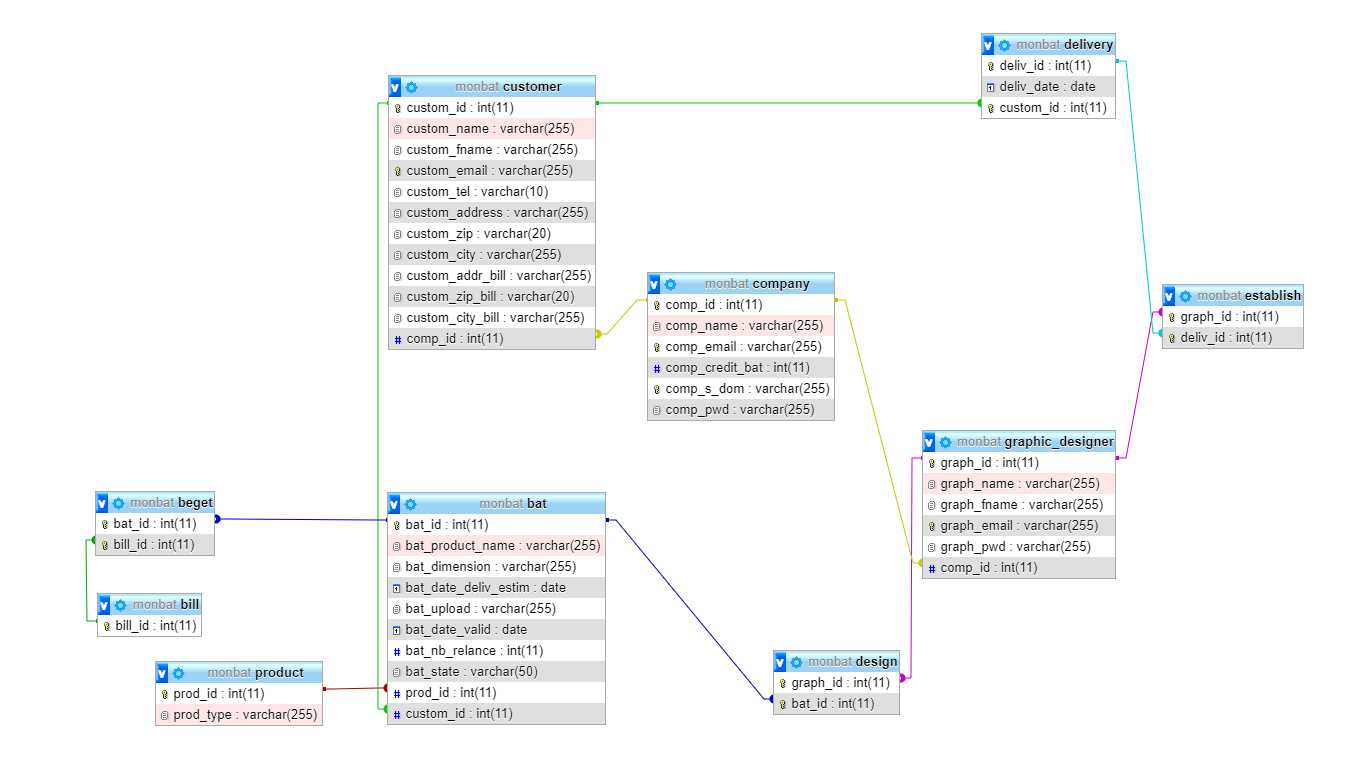
);

Nous remarquerons ici que le champ id du client sera auto-incrementé et chaque client aura son propre id, ce dernier sera aussi la clé primaire de la table, permettant les associations avec les autres tables comme nous pouvons le voir au niveau de la ligne de la FOREIGN KEY où cette clé étrangère relie donc la table « *customer* » à la table « *company*».

De plus, nous avons indiquer que le champ « custom\_email » se devra d’être unique. Cela sera considéré come d’ores et déjà une sécurité afin d’éviter les inscriptions « doublons » et perturber l’alimentation de la base de données.

De là, cette dernière a pu être créer entièrement comme le montre le schéma suivant en mode « concepteur ».

**Base de données en mode concepteur**



**Développement et programmation**

1. **L’index de la page de connexion du graphiste**

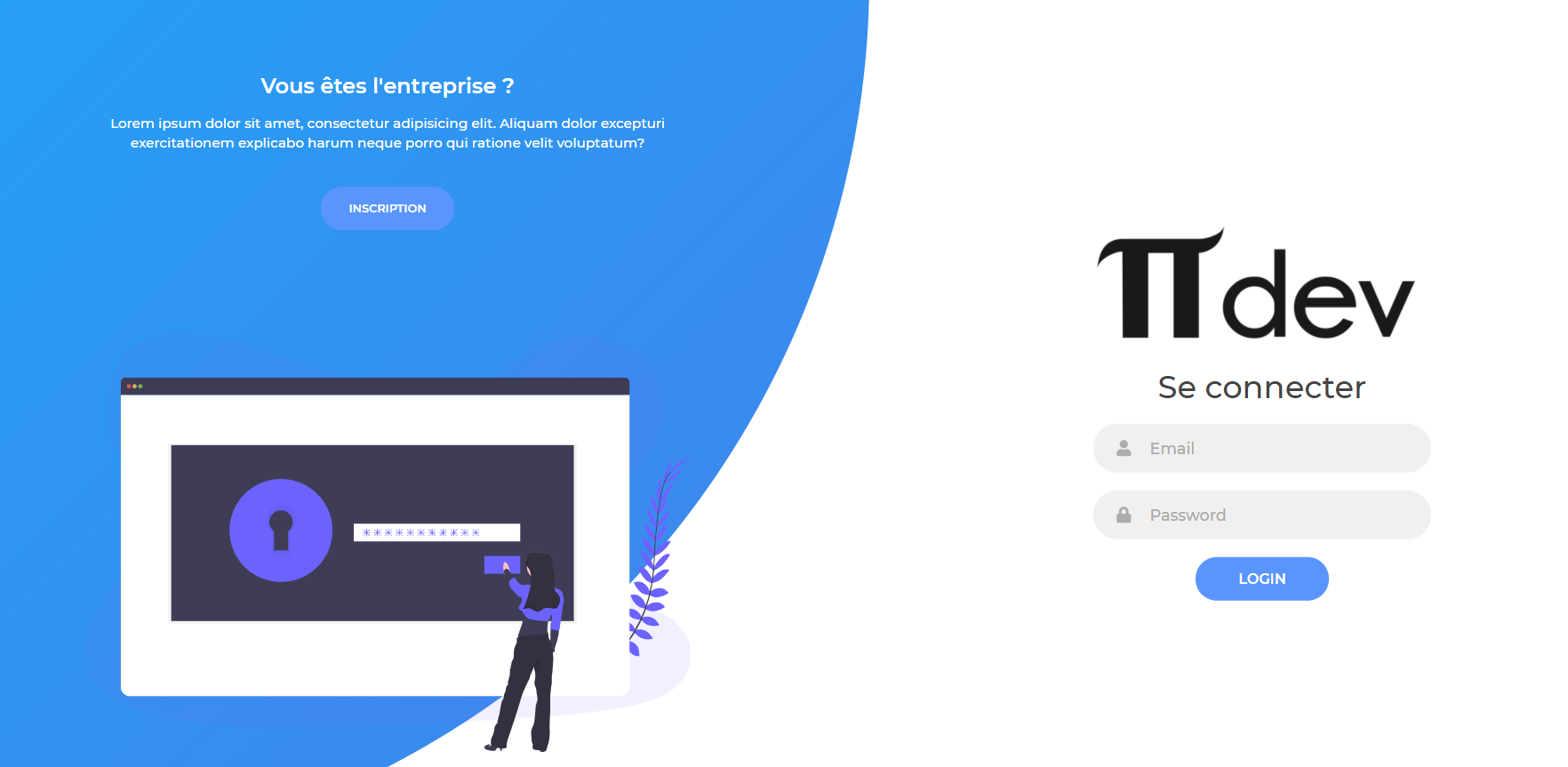
La première étape abordée fut le développement de la page « index » de l’application web. Cette dernière est, pour l’instant, la page qui apparaîtra en tout premier lieu lorsque l’utilisateur appellera le site.

Cette dernière est plus utile au graphiste car elle permet à celui-ci de se connecter afin de visualiser les bons à tirer sur lesquels il travaillera ainsi que d’autres options. Plus tard, cette page ne sera plus un « index » car au final, la première page que l’entreprise veut montrer en premier lieu sera celle où l’entreprise cliente pourra s’inscrire ou s’authentifier.

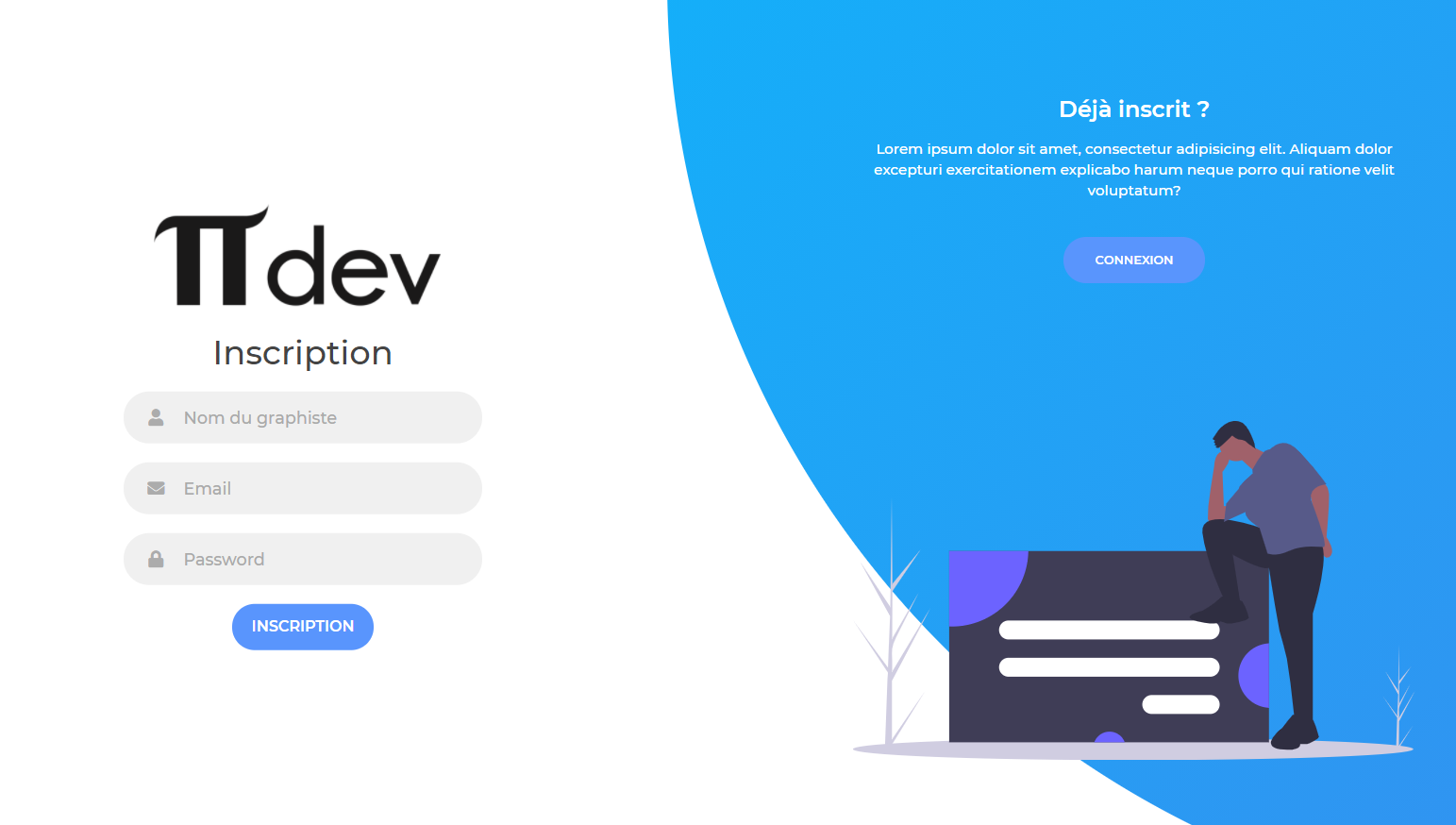
Ce fut sur l’étape de la connexion du graphiste que l’entreprise voulait que je travaille en premier lieu.

Comme vu précédemment dans la partie « Initialisation du projet », la maquette m’a guidé sur la vision globale de la future page de connexion du graphiste.

Après quelques essais et l’aval du tuteur de stage, j’ai proposé ce design en finalité :



De plus, je me suis permis de rajouter une fonctionnalité qui permettra, plus tard, aux entreprises de rajouter les graphistes par le biais d’un autre formulaire en cliquant sur le bouton « Inscription prévu dans le paragraphe « Vous êtes l’entreprise ? ».



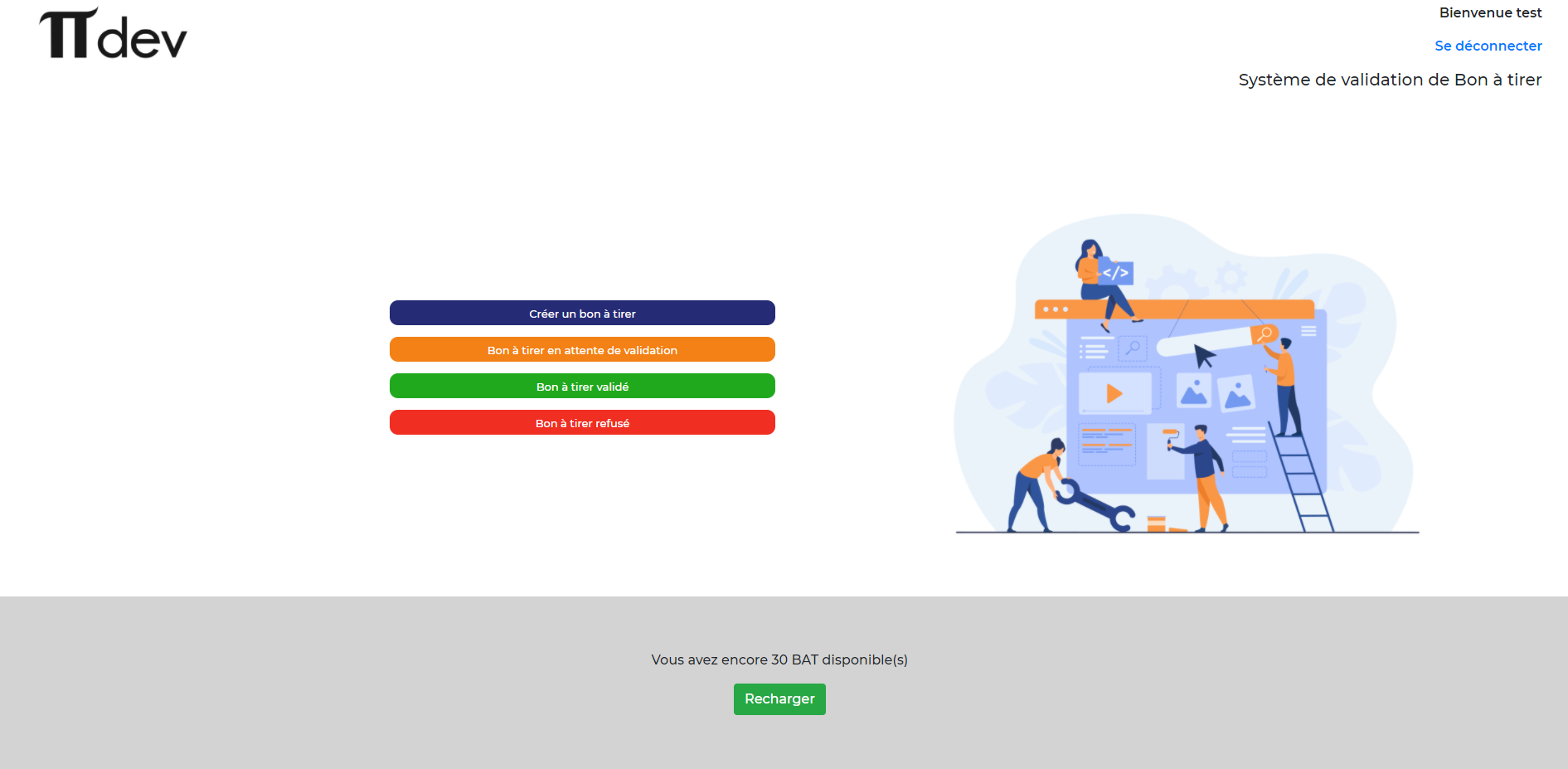
Il faut cependant savoir que ce n’est encore là qu’une intégration et que la fonctionnalité « back-end » n’est pas encore disponible pour cette partie.

La partie « Connexion » fut une priorité par rapport à la partie « Inscription » pour la durée de mon stage.

1. **L’accueil du graphiste**

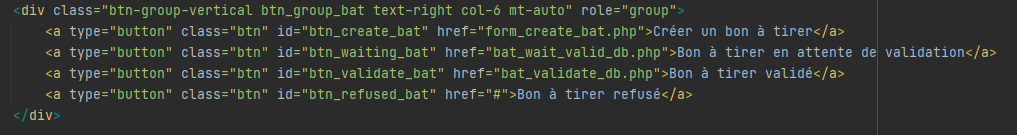
Une fois l’authentification de l’utilisateur effectué, celui-ci se retrouvera sur une page lui permettant d’effectuer ses diverses tâches ou encore de prendre connaissances de l’état de ses bons à tirer envoyés à ses clients.

En m’inspirant de la maquette fournie, voici comment cette dernière apparaît aujourd’hui :



L’essentiel fut d’intégrer les boutons permettant d’accéder aux différentes parties de l’application et de constater le nombre de bons à tirer restants à l’entreprise.

Afin d’intégrer au mieux les boutons, j’ai utilisé une « class » déjà prédéfinie sur Bootstrap pour mettre ces derniers en colonne : « btn-group-vertical » :



Plus tard, en ce qui concerne le nombre de bons à tirer restants, ceux-ci devront s’afficher par rapport aux nombres qui sera stockée dans la base de données (ce qui ne sera pas encore effectif aujourd’hui).

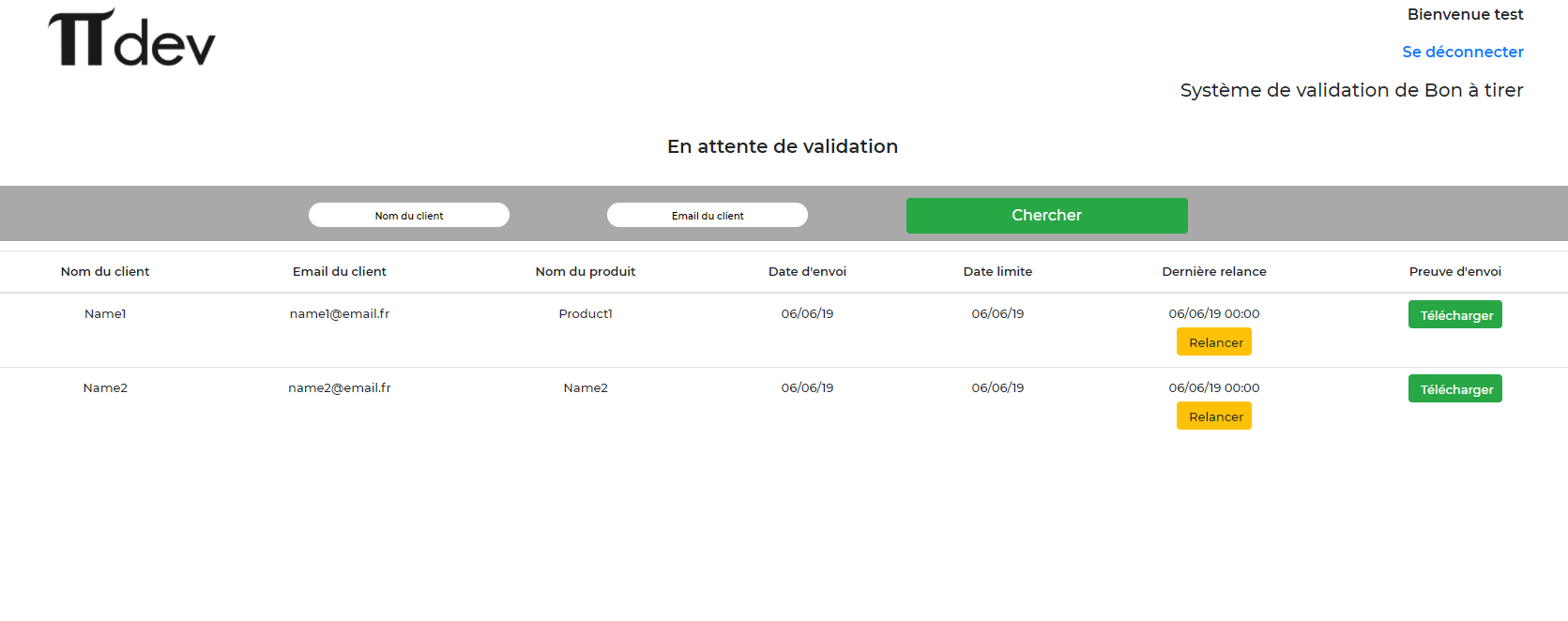
Avec le restant de maquettes mis à ma disposition, j’ai pu rendre fonctionnel les trois premiers boutons permettant ainsi de naviguer vers la création de bons à tirer, visualiser le tableau de bord des bons à tirer en attente ou ceux déjà validés par le client de l’entreprise.

Enfin, nous pouvons constater qu’un message de bienvenue avec le nom de l’utilisateur est apparent sur l’écran en haut à droite, réalisé grâce au langage PHP (dont nous aborderons le sujet plus tard) et un lien permettant de se déconnecter de la session en cours.

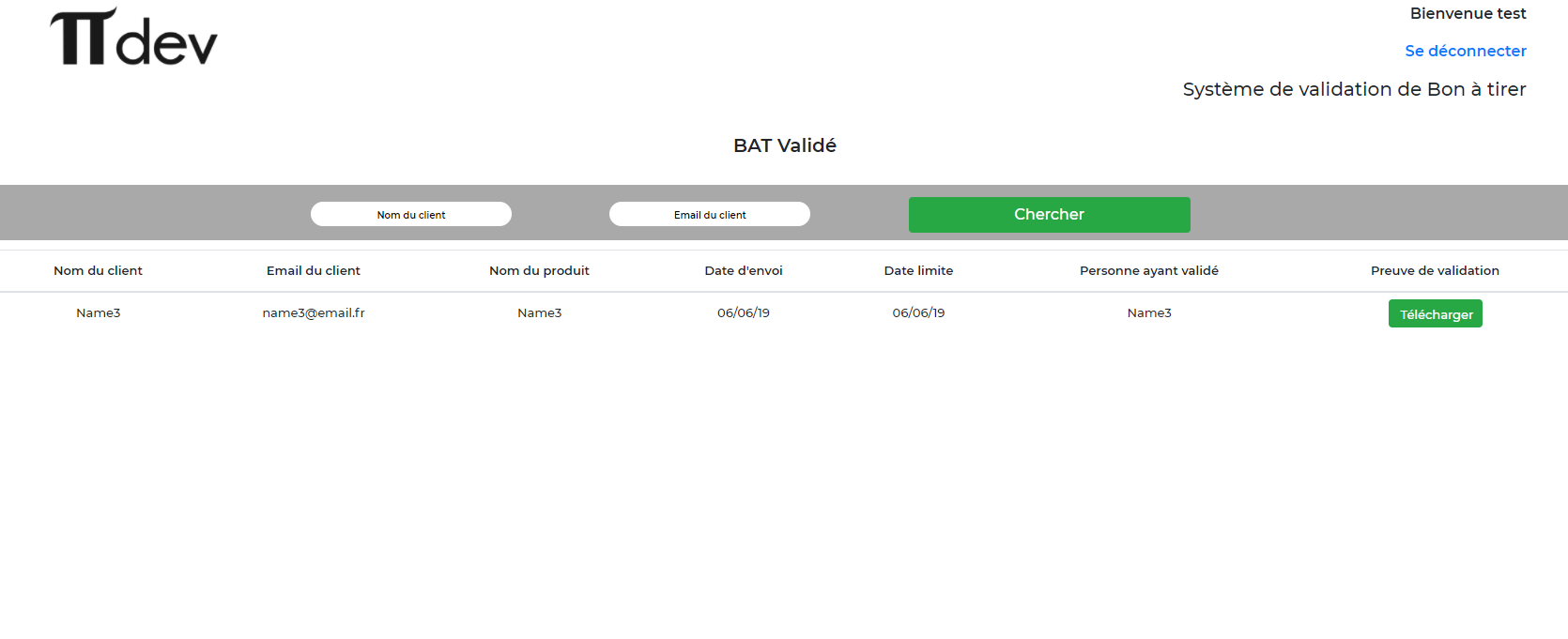
1. **Les tableaux de bord du graphiste**

Ces tableaux sont accessibles en cliquant sur le bouton « Bon à tirer en attente de validation » ou « Bon à tirer validé ».

Le tuteur de stage désirait une page simple et claire prenant toute la largeur de l’écran comme l’on peut le voir sur les maquettes en annexe. J’ai donc proposé une intégration proche de la version souhaitée :



*Le tableau de bord pour les bons en attente de validation*



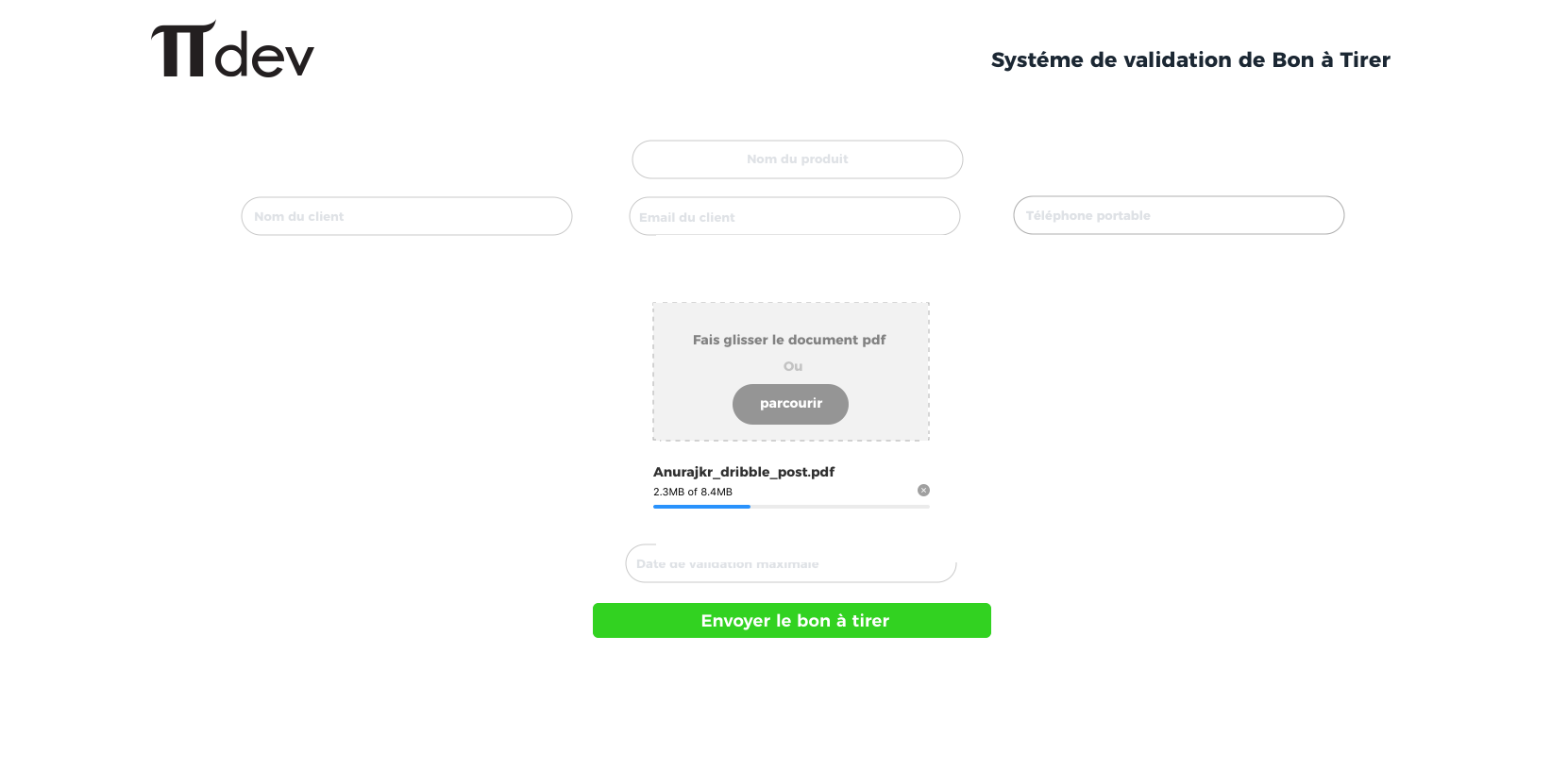
*Le tableau de bord pour les bons validé par le client*

La barre de recherche, les relances et le téléchargement des preuves ne sont cependant pas encore actifs mais l’intégration était souhaitable.

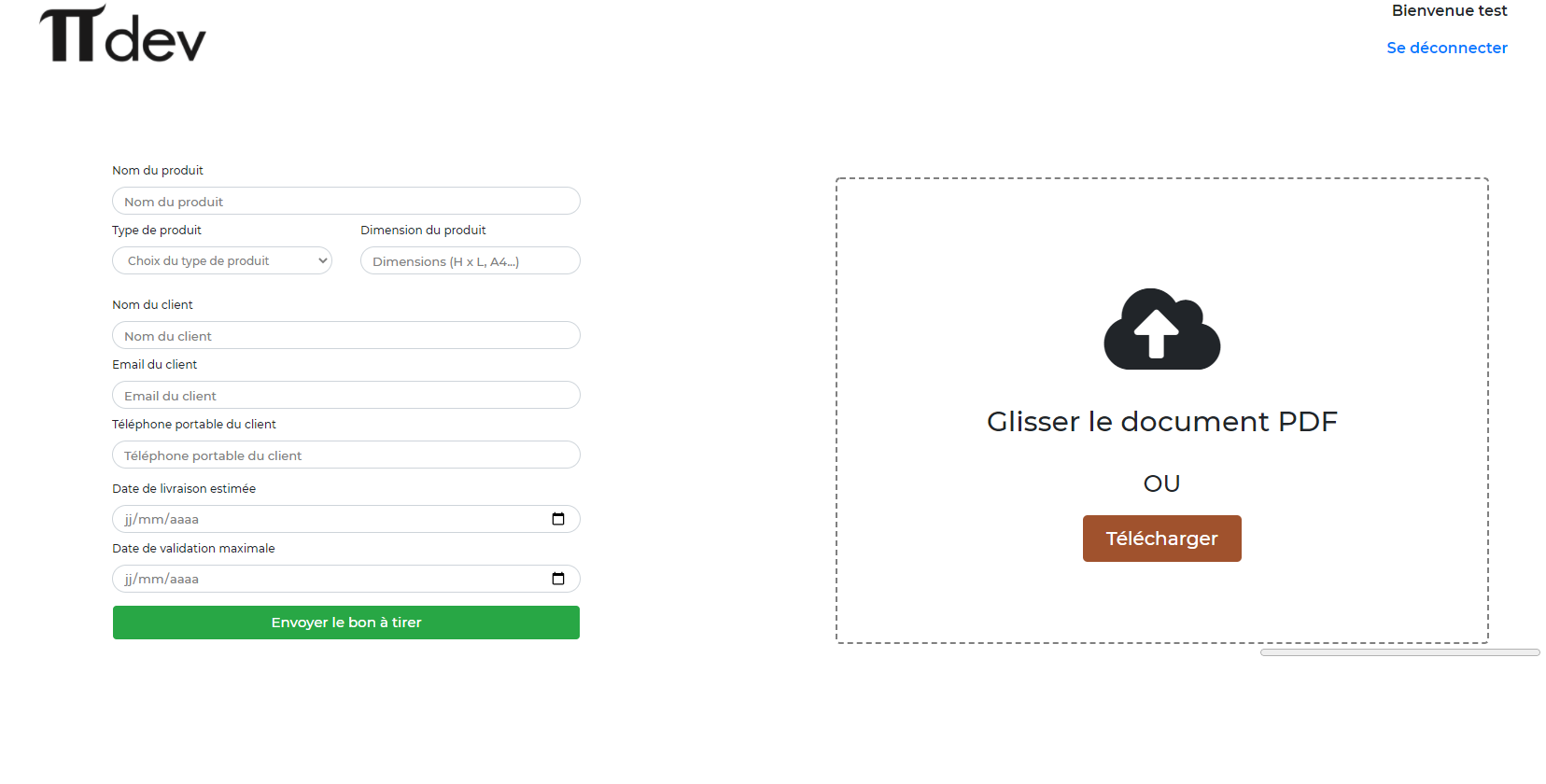
L’apparition des données se font essentiellement grâce au langage PHP et à l’« état » du bon à tirer (c’est-à-dire si celui-ci est validé ou non par le client), ainsi nous avons les bonnes informations souhaitées dans les bons tableaux correspondants.

1. **Le formulaire de création de BAT**

Malgré son inachèvement, le formulaire de création a bénéficié de quelques mesures d’aménagement. J’ai dû partir d’un modèle proposé au tout début de mon stage qui ressemblait à cela :

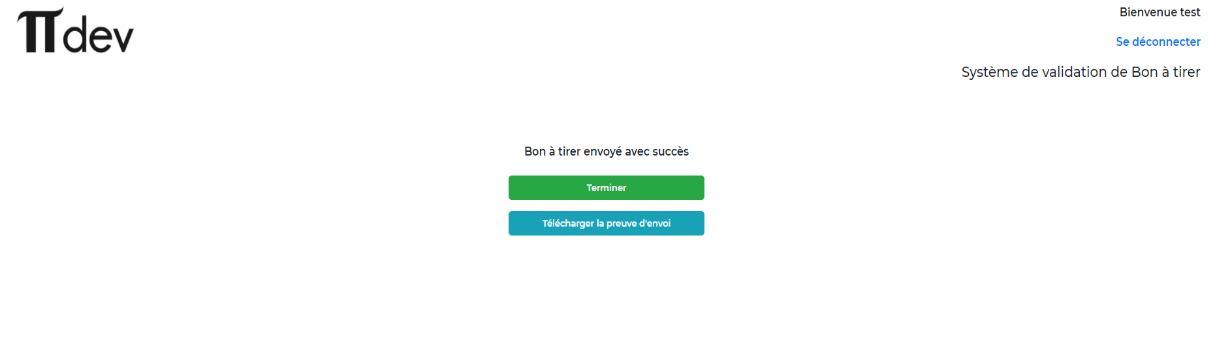


Après quelques réflexions de la part de l’entreprise, des changements ont été souhaités lors de mon insertion dans cette dernière : le formulaire devait désormais se trouver à gauche de l’écran, quelques champs du formulaire devaient être rajoutés et le téléchargement devait finalement se trouver à droite de l’écran. J’ai dû retravailler le design et la mise en page de ce formulaire malgré avoir réussi la première intégration, le résultat n’a finalement pas abouti malgré une bonne approche :



1. **Les diverses intégrations**

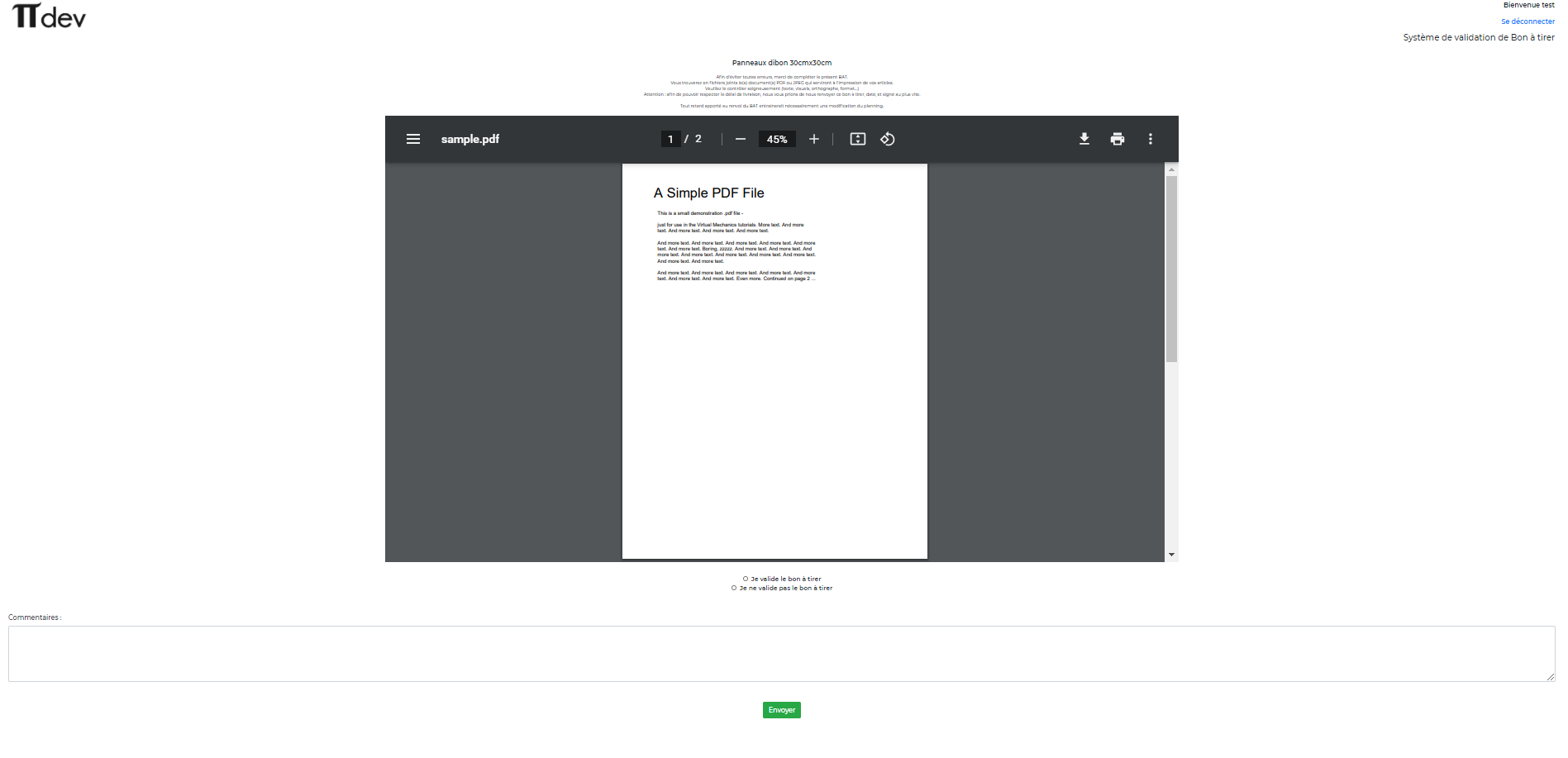
Le restant des intégrations désirées n’a pas été trop difficile à réaliser, notamment en ce qui concerne les fenêtres alertant la réussite de certaines actions dont voici quelques exemples :



*Fenêtre attestant de la réussite de l’envoi du bon à tirer au client*



*Fenêtre confirmant l’envoi du bon à tirer non validé au graphiste*



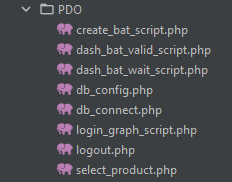
*Formulaire permettant au client de valider ou non le bon à tirer*

**La gestion des informations**

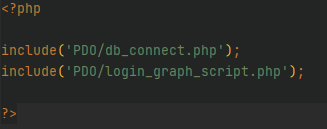
Après l’intégration des différentes fenêtres grâce à la maquette, mon objectif a été de travailler sur la partie « back-end » grâce au langage PHP et ainsi interagir avec la base de données. Pour cela, un C.R.U.D. (Create Read Update Delete) a été nécessaire à réaliser car comme son nom l’indique, cela permettra de créer, lire, mettre à jour ou encore supprimer des informations.

Afin de communiquer avec la base de données concernée et après quelques recherches sur le meilleur moyen pour effectuer cela, il s’est avéré que j’ai opté pour utiliser l’extension P.D.O. (PHP Data Objects). Il s’agit là d’une interface qui facilite l’accession à la base de données tout en étant la plus sécuritaire possible.

A partir de mon environnement de travail, j’ai décidé de créer un dossier « PDO » réunissant toutes les pages servant à la communication avec la base de données :



De ce fait, la facilité d’intégrer ces pages de connexions a été effective car il ne me restait plus qu’à appeler ces dernières lorsque je le désirais grâce à l’expression de langage « include » tout en spécifiant le fichier voulu :



1. **L’instauration de la connexion à la base de données**

La première démarche effectuée a été de configurer l’accès à cette base de données. Pour cela, j’ai décidé de fragmenter les fichiers afin de sécuriser au maximum l’accès à la base de données. J’ai créé en premier lieu le fichier *db\_config.php* contenant les informations nécessaires pour s’authentifier à la base :

<?php  
  
$DB\_DSN = 'mysql:host=localhost;charset=utf8;dbname=monbat';  
$DB\_USER = 'root';  
$DB\_PASS = '';

Trois variables sont initiées dans ce fichier :

\_ $DB\_DSN : contient le Data Source Name, c’est-à-dire toute la structure de données utilisée pour se connecter à une base de données notamment (ici nous aurons le nom de l’hôte, le charset et le nom de la base données renseignés),

\_$DB\_USER : contient l’identifiant de l’utilisateur voulant se connecter,

\_$DB\_PASS : contient le mot de passe servant à s’authentifier correctement et à sécuriser l’accès.

Grâce à ces dernières, nous pourrons donc interagir avec un autre fichier permettant de s’authentifier, il sera nommé *db\_connect.php*:

<?php  
include*(*'db\_config.php'*)*;  
  
try*{* $options =  
 *[* PDO::*MYSQL\_ATTR\_INIT\_COMMAND* => 'SET NAMES utf8', /\*Work with utf8 for sure\*/  
 PDO::*ATTR\_ERRMODE* => PDO::*ERRMODE\_EXCEPTION  
 ]*;  
  
 $PDO = new PDO*(*$DB\_DSN, $DB\_USER, $DB\_PASS, $options*)*;  
   
*}*catch*(*PDOException $pe*){* echo "Error : " . $pe->getMessage*()* . "*<*br*>*";  
 echo "N° : " . $pe->getCode*()*;  
 die*(*"Fin du script"*)*;  
*}*

Nous pouvons en premier lieu constater que nous appelons, grâce à l’expression « *include* », le fichier contenant les variables vu précédemment. C’est ici que ces dernières vont être utiles.

La méthode utilisée ici pour se connecter est un « *try/catch* ». En d’autres termes, nous demandons à la machine de faire une tentative de connexion (*try*) mais si cela venait à échouer, alors une interaction serait effectuée (*catch*) (ici nous demandons à la machine de récupérer l’erreur et de nous la communiquer afin de tenter de résoudre le problème grâce au système de PDOException représentant l’erreur émise par PDO et définit dans une variable dans les paramètres du *catch*).

Pour en revenir à la méthode *try* permettant de se connecter, nous retrouvons une variable intéressante qui permettra d’instaurer la connexion. En effet, par le biais du « *new PDO* » (et des paramètres instaurés) contenu dans cette variable, il sera possible de faire la liaison entre la page et la base de données à chaque fois que nous ferons appel à cette dernière. C’est ici que le système de sécurité avec les « *include* » œuvre. Au lieu d’inscrire de manière brute les identifiants de connexion à la base de données, le système devra aller les chercher via le fichier db\_config.php où se trouve les variables contenant les informations nécessaires.

1. **L’authentification du graphiste**

Mon premier travail a été de pouvoir authentifier le graphiste voulant se connecter sur sa session. Pour cela, je devais faire lire (Read) les informations de la table « Graphic\_Designer » dans la base de données et ainsi les faire comparer aux informations renseignées par l’utilisateur. Tout cela se passera au niveau de la page *index.php* où le graphiste devra se connecter.

En parallèle à cela, j’ai créé la page *login\_graph\_script.php* qui assurera les fonctionnalités « back-end » lorsque cela sera demandé sur la page *index.php*.

Voici les principales fonctions qui ont été initiées :

<?php  
if*(*isset*(*$\_POST*[*"btn\_co\_graph"*]))  
{* if*(*empty*(*$\_POST*[*"email"*])* || empty*(*$\_POST*[*"password"*])){* $message = '*<*label*>* Tous les champs sont obligatoires *</*label*>*';  
 *}* else  
 *{* $email = $\_POST*[*"email"*]*;  
 $pwd = $\_POST*[*"password"*]*;  
 $query = "SELECT \* FROM graphic\_designer WHERE graph\_email=:graph\_email AND graph\_pwd = :graph\_pwd";  
 $statement = $PDO->prepare*(*$query*)*;  
 $statement->bindValue*(*':graph\_email', $email, PDO::PARAM\_STR*)*;  
 $statement->bindValue*(*':graph\_pwd', $pwd, PDO::PARAM\_STR*)*;  
 $statement->execute*()*;  
  
 $query\_name= "SELECT \* FROM graphic\_designer WHERE graph\_email=:graph\_email";  
 $stat\_name = $PDO->prepare*(*$query\_name*)*;  
 $stat\_name->bindValue*(*':graph\_email', $email, PDO::PARAM\_STR*)*;  
 $stat\_name->execute*()*;  
 $graph\_des = $stat\_name->fetch*(*PDO::*FETCH\_OBJ)*;  
  
 $count = $statement->rowCount*()*;  
 if*(*$count > 0*)  
 {* $\_SESSION*[*"graph"*]* = $graph\_des->graph\_name;  
 header*(*"location:graph\_choice.php"*)*;  
 *}* else*{* $message = '*<*label*>*Email ou mot de passe erroné(s)*</*label*>*';  
 *}  
 }  
}*

Nous pouvons remarquer que la communication ici cible essentiellement la base de données car nous retrouvons une requête SQL destinée à cette dernière.

Quelques points de sécurité ont été ajoutés dans ce fichier. Lorsque le bouton contenant l’id « btn\_co\_graph » a été cliqué, cette sécurité se déclenche :

\_ Nous vérifions que tous les champs soient bien remplis,

\_Nous soumettons une « requête préparée ». Celle-ci sera stockée dans la variable *$query* et les conditions de sélection des données seront retranscrites de manière sécurisée : WHERE graph\_email=:graph\_email AND graph\_pwd = : graph\_pwd"; . *:graph\_email* et *:graph\_pwd* sont transcrits de cette manière car ce sont là des expressions qui seront définies par la suite grâce aux *bindValue* qui verront affecter une valeur. Ici, il s’agira de la variable *$email* contenant $\_POST[« email »], autrement dit : la valeur de ce qu’à rentrer l’utilisateur dans l’input contenant *name= email*. Il s’agira de la même procédure en ce qui concerne le mot de passe.

Une fois la « requête préparée » exécutée, le système vérifiera la correspondance des données.

Si aucune correspondance n’est trouvée, un message d’erreur sera initialisé dans la variable *$message*. Celui-ci sera alors déclenché dans la page correspondante en l’initialisant ainsi :

<?php  
if*(*isset*(*$message*)){* echo '*<*label class="text-danger" id="alert\_create\_bat"*>*'.$message.'*</*label*>*';  
*}*?>

L’initialisation de la variable étant déclenchée, le code sera appliqué là aussi et le message apparaîtra ainsi :



Par ailleurs, le message d’erreur donnera une indication floue d’où se trouve l’erreur pour éviter de cibler une éventuelle attaque malsaine.

L’erreur se déclenchera de la même manière si un ou plusieurs champs sont vide(s) :



Si toutefois la correspondance venait à réussir sans souci, alors un système de recherche se déclenche. C’est grâce essentiellement au *rowCount* qu’il est possible de faire cela : nous initialisons une requête préparée permettant de rechercher la correspondance entre l’email renseigné et tous les emails stockés. Si une correspondance s’effectue, le système va stocker une ligne dans un tableau pour mémoriser cela. De là, il suffit tout simplement de compter ces lignes : si une ligne est stockée (donc s’il y a une correspondance), le *rowCount* correspondra au chiffre du nombre de lignes trouvées (en l’occurrence ici : 1). A l’inverse, si aucune correspondance n’est trouvée, ce dernier aura la valeur de 0.

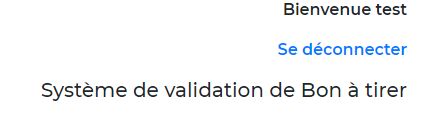
Il faudra ensuite élaborer une condition permettant des actions si le *rowCount* est supérieur (ou dans ce cas « différent ») de 0.

Ici, l’utilisateur se verra initié une session grâce à la superglobale *$\_SESSION* permettant de stocker des informations (ici le nom du graphiste) et sera redirigé vers la page d’accès aux différents choix de travail du graphiste grâce au *header*.

Avec la deuxième requête préparée présente dans ce fichier, nous avons réussi à récupérer le nom de l’utilisateur grâce à l’expression *fetch* permettant de récupérer la ligne voulue dans un jeu de résultat PDO (ici la ligne correspondante à l’email renseigné).

Dès lors, il est aussitôt possible d’afficher un message avec l’information souhaitée dans ce dernier.

echo '*<*p class="text-right text\_welcome"*>* Bienvenue ' .htmlspecialchars*(*$\_SESSION*[*"graph"*])*.'*</*p*>*';



Afin de rendre cela effectif, PHP a besoin qu’on lui précise en début de fichier qu’une session est en cours et qu’il devra effectuer toutes les démarches associées à celle-ci. Pour cela, il faudra placer la fonction *session\_start()*  pour que la superglobale *$\_SESSION* fasse effet :

<?php  
session\_start*()*;

Une sécurité a encore été apportée ici. En effet, la fonction *htmlspecialchars* a été préconisée afin d’éviter à certains individus malveillants de rentrer des caractères spéciaux pouvant créer des injections malsaines pour le site. Cette fonction aura pour but de convertir ces caractères spéciaux et de ne pas les faire apparaître tels que certains auraient voulus.

Une fois que tout cela soit fonctionnel, il suffit de rajouter sur la page *index.php* quelques lignes de code permettant la communication entre le formulaire de connexion et la base de données :

<?php  
  
include*(*'PDO/db\_connect.php'*)*;  
include*(*'PDO/login\_graph\_script.php'*)*;  
  
?>

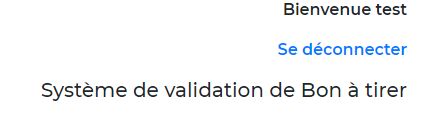
1. **La déconnexion**

L’entreprise souhaitait aussi que le graphiste puisse se déconnecter par lui-même ou après un certain temps d’inactivité. Un fichier *logout.php* fut donc créer pour permettre cela :

<?php  
  
unset*(*$\_SESSION*[*"email"*])*;  
if *(*ini\_get*(*"session.use\_cookies"*))  
{* setcookie*(*session\_name*()*, '', time*()*-42000*)*;  
*}*session\_destroy*()*;  
  
header*(*'Location:../index.php'*)*;

Lorsque l’utilisateur cliquera sur le lien « Se déconnecter » apparu en même temps que la phrase de bienvenue, celui-ci déclenchera le mécanisme du fichier :

echo '*<*a href="PDO/logout.php"*><*p class="text-right text\_logout"*>*Se déconnecter*</*p*></*a*>*';



En parallèle à cela, une condition est mise en place avec la fonction time(), qui dans ce cas présent et avec l’aide de cookies paramétrés, servira à compter le temps écoulé. Une fois 42000 secondes passées, soit l’équivalent de douze heures à peu près, l’utilisateur sera déconnecté automatiquement.

1. **Le formulaire de création de bon à tirer**

Une attention particulière a été donné pour ce formulaire où beaucoup d’informations circuleront plus tard sur ce dernier, notamment des renseignements sur les produits et plus importants encore, sur les coordonnées des potentiels clients de l’entreprise.

Il a fallu réfléchir au préalable sur les diverses sécurités que pouvait se procurer l’application afin d’éviter tout acte malveillant.

Pour le moment, deux sécurités ont été établies : une première du point de vue du langage PHP (donc en back-end, au niveau du système) et une deuxième, configurée cette fois-ci grâce à JQuey, permettant d’interdire l’envoi de certaines données non désirées dans la base de données.

En ce qui concerne la première mesure de sécurité (établie grâce à PHP), voici la procédure créée dans le fichier contenant le script de vérifications de données et de communication avec la base de données :

* 1. La vérification des champs remplis

Comme pour le formulaire de connexion, la première sécurité élaborée fut d’empêcher à l’utilisateur d’envoyer des champs vides à la base de données. Ceci s’est tout simplement fait avec des fonctions *empty ()* dans les paramètres de conditions qui empêcheront l’envoi des données si un champ désigné est vide (empty) :

if*(*empty*(*$\_POST*[*"product\_name"*])* || empty*(*$\_POST*[*"product\_dim"*])* || empty*(*$\_POST*[*"date\_estim\_del"*])* || empty*(*$\_POST*[*"date\_valid\_max"*])* || empty*(*$\_POST*[*"product\_name\_cust"*])* || empty*(*$\_POST*[*"product\_email\_cust"*])* || empty*(*$\_POST*[*"product\_tel\_cust"*])) {* $message = '*<*label*>* Tous les champs sont à renseigner *</*label*>*';*}*

* 1. L’insertion et la sécurisation des données envoyées côté « back-end »

Pour l’envoi des données, là encore une requête préparée a été la solution la plus préférable pour la sécurisation. Celle-ci a été divisée en plusieurs parties :

\_ Tout d’abord, une variable a été créée afin de stocker la requête :

$query\_bat = "INSERT INTO bat(bat\_product\_name, bat\_dimension, bat\_date\_deliv\_estim, bat\_date\_valid, bat\_state,   
 prod\_id, custom\_id)  
 VALUES(:bat\_product\_name, :bat\_dimension, :bat\_date\_estim\_del, :bat\_date\_valid, :bat\_state,   
 :bat\_product\_type, :bat\_custom\_id)";

Les champs de la table ayant été précisés (bat\_product\_name, bat\_dimension…) et les valeurs devant être associées aussi dans l’ordre des champs définis ( :bat\_product\_name, :bat\_dimension…).

Une fois cette requête établie, nous demandons au système de la préparer sans toutefois l’exécuter pour l’instant et de la stocker dans une variable :

$statement\_bat = $PDO->prepare*(*$query\_bat*)*;

\_ De là, nous pouvons faire appel aux mesures de sécurité établies au préalable afin de vérifier les données. En effet, un tableau dit « d’erreurs » a été défini au début du fichier afin de récupérer toutes les incohérences que le système pourrait rencontrer :

$formError = *[]*;

Ces erreurs seront donc stockées après être passés dans une condition permettant ou non l’insertion d’une ligne dans ce tableau. Voici un exemple avec la vérification des données rentrées par l’utilisateur par rapport au nom du produit :

if*(*preg\_match*(*$prod\_name\_pat, $\_POST*[*"product\_name"*])){* $product\_name = $\_POST*[*"product\_name"*]*;*}*else*{* $formError*[*'product'*]* = 'product=true';  
*}*

Nous constatons ici la fonction *preg\_match()* qui nous a été d’une forte utilité durant la vérification. En effet, cette dernière fera appel aux Regex (Expressions Régulières) qui permet au système de comprendre les caractères de notre langage qui sont autorisés ou non à être envoyés en base de données. Pour le cas du nom du produit, une Regex a été prédéfinie au début du fichier, stockée dans une variable et appelé dans les paramètres de la fonction *preg\_match()* :

$prod\_name\_pat = "/^[A-Za-z0-9\/éèàçâêûîôäëüïö\:\\_\'\-\s]+$/";

On peut comprendre ici que les majuscules, les minuscules, les chiffres et les accents sont autorisés mais pas les caractères spéciaux tels que « < » ou « > » qui peuvent permettre des injections SQL entre autres et ainsi mettre en danger les données stockées.

De là, la fonction *preg\_match()* comparera cette Regex avec ce qui a été posté dans le champ correspondant.

Si toutes les mesures sont respectées, alors une variable stockera les données renseignées ($\_POST[« product\_name »] => ce qui a été renseigné dans l’input portant le name « product\_name »).

Au contraire, si une faille est détectée, en plus d’empêcher l’envoi des données en n’initialisant pas une variable portant la valeur de ce qu’à renseigner l’utilisateur, une ligne dans le tableau d’erreur sera stockée qui portera le nom [‘product’] et aura comme action de rajouter la valeur « product=true » dans la barre URL après le nom du fichier correspondant.

A partir de cela, nous avons inclus des lignes de codes en PHP dans le fichier contenant le formulaire permettant de vérifier si cette erreur se trouve dans le tableau et les conséquences que cela implique :

<?php  
if *(*isset*(*$\_GET*[*"product"*]))  
{* ?>  
 *<*div class="alert alert-danger"*>*Caractère(s) interdit(s) ou nombre trop important*</*div*>* <?php  
*}*?>

Ici, nous pourrions expliquer ces lignes en disant que : s’il est avéré que le mot product existe et a comme valeur « true » dans la barre URL ($\_GET), alors un message d’alerte sera visible (grâce à l’insertion d’une « div » permettant cela) et expliquant les raisons de l’erreur.

Une attention particulière a été demandé par rapport à la Regex permettant de vérifier le numéro de téléphone du client.

En effet, l’entreprise voulant absolument envoyer des SMS d’alerte au client concerné, celle-ci m’a demandé de n’accorder l’envoi des données qu’aux clients ayant un numéro de téléphone mobile, commençant donc par 06 ou 07. La conception de la Regex a donc produit cela :

$cust\_tel\_pat = "/^(0)(6|7)[0-9]{8}$/";

Ainsi, le graphiste ne peut rentrer le numéro de téléphone des clients que si ce dernier commence par 06 ou 07 et comporte ni plus ni moins de huit chiffres.

* 1. L’insertion et la sécurisation des données envoyées côté « front-end »

Une autre sécurité a donc été instauré grâce à JQuery.

Pour cela, il a fallu créer un fichier de type JavaScript que nous appellerons dans la page concernée :

*<*script src="js/jquery/create\_bat.js"*></*script*>*

Comme il a été conçu au niveau de la sécurité niveau « back-end », des Regex ont été créées afin de déterminer les bons caractères et surtout d’enclencher les erreurs qui apparaîtront sur l’écran de l’utilisateur :

let regProdName = /^*[*A**-**Za**-**z0**-**9**\/**éèàçâêûîôäëüïö:\_'\-\s*]*+$/;  
let regProdDim = /^*[*A**-**Za**-**z0**-**9**\/**éèàçâêûîôäëüïö:\_'\-\s*]*+$/;  
let regCustName = /^*[*A**-**Za**-**z0**-**9**\/**éèàçâêûîôäëüïö:\_'\-\s*]*+$/;  
let regCustEmail= /^*([*a**-**zA**-**Z0**-**9\_\-\.*]*+*)*@*([*a**-**zA**-**Z0**-**9\_\-\.*]*+*)*\.*([*a**-**zA**-**Z*]{*2,5*})*$/;  
let regCustTel = /^*(*0*)(*6|7*)[*0**-**9*]{*8*}*$/;

Et une variable « result » a un booléen tourné vers « true ». Ce booléen permettra à la fin des vérifications de continuer le processus ou non. En effet, si le *return result* placé à la fin de la fonction contient le booléen true, alors le processus continuera, à l’inverse, si le booléen est à false, cela stoppera tout processus en cours. Il a donc fallu penser à changer cette variable à chaque condition.

Une fois le formulaire soumis, des conditions ont donc été mises en place pour la vérification des données dont voici un exemple par rapport au nom du produit :

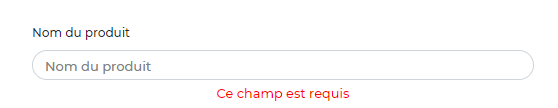
if*(*$*(*'#input\_product\_name*')*.val*()* === ""*){* /\*$('#input\_product\_name').addClass('is-invalid');\*/  
 $*(*'#alert1*')*.text*(*'Ce champ est requis'*)*.css*(*"color", "#FF0000"*)*;  
 result = false;  
*}*else if*(*regProdName.test*(*$*(*'#input\_product\_name*')*.val*())* === false*){* /\*$('#input\_product\_name').addClass('is-invalid');\*/  
 $*(*'#alert1*')*.text*(*'Le format est invalide'*)*.css*(*"color", "#FF702D"*)*;  
 result = false;  
*}*

Nous constatons dans ces dernières qu’un appel d’un id nommé « alert1 » dans le fichier correspondant. En effet, afin d’afficher le message d’erreur à l’endroit voulu, une balise *small* a été mise en place avec l’id correspondant :

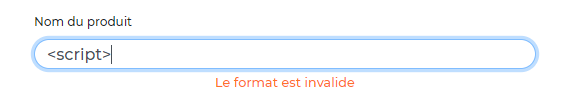
*<*small class="form-text" id="alert1"*></*small*>*

Ne contenant rien pour le moment, le déclenchement sera effectif si les données renseignées répondent aux critères de la condition du fichier JavaScript.

Dans la première condition, nous demandons à vérifier si le champ est vide ou non. Si tel est le cas, l’alerte se déclenchera avec un texte et une couleur définis :



Au niveau de l’autre condition, celle-ci testera la Regex stockée dans la variable prédéfinie au préalable (*regProdName.test*) avec la valeur (*val()*) stockée dans l’input portant l’id *input\_product\_name*. Si cette Regex détecte une incohérence, l’alerte se transformera ainsi :



* 1. L’enregistrement des clients

Une particularité m’a été demandé de la part de l’entreprise lorsque l’enregistrement d’un bon à tirer sera demandé. En indiquant le client correspond au bon à tirer effectué, le système devra rechercher si ce même client existe ou non dans la base de données. Si le client est absent de la base de données, alors ces coordonnées devront être envoyés dans la table prévue à cet effet.

J’ai donc élaboré un système de recherche et d’enregistrement du client avant même d’envoyer les données du bon à tirer :

$query\_search\_cust = "SELECT \* FROM customer WHERE custom\_name = :custom\_name";  
  
$statement\_search\_cust = $PDO->prepare*(*$query\_search\_cust*)*;  
$statement\_search\_cust->bindValue*(*'custom\_name', $custom\_name, $PDO::*PARAM\_STR)*;  
  
if*(*count*(*$formError*)* === 0*){*$statement\_search\_cust->execute*()*;*}*else*{* $sUrl = implode*(*"&", $formError*)*;  
 header*(*"Location:form\_create\_bat.php?".$sUrl*)*;  
 exit;  
*}*

Pour l’instant la recherche s’effectue par le biais du nom du client, mais cela est à titre provisoire car étant conscient que le même nom de famille peut être associé à différents clients, les recherches et le mécanisme pourrait se voir être erroné. L’entreprise réfléchit encore par quel biais la recherche s’effectuera plus tard.

Ici, une autre requête préparée est appelé afin de rechercher si le client existe ou non dans la base de données.

Si tel n’est pas le cas, nous nous en apercevrons en comptant les lignes du tableau créé par le système ainsi :

$count = $statement\_search\_cust->rowCount*()*;

if*(*$count === 0*){* /\* The customer will be register \*/  
 $query\_custom = "INSERT INTO customer(custom\_name, custom\_email, custom\_tel)   
 VALUES(:custom\_name, :custom\_email, :custom\_tel)";  
 $statement\_cust = $PDO->prepare*(*$query\_custom*)*;  
  
 if*(*count*(*$formError*)* === 0*){* $statement\_cust->bindValue*(*':custom\_name', $custom\_name, PDO::PARAM\_STR*)*;  
 $statement\_cust->bindValue*(*':custom\_email', $custom\_email, PDO::PARAM\_STR*)*;  
 $statement\_cust->bindValue*(*':custom\_tel', $custom\_tel, PDO::PARAM\_STR*)*;  
 $statement\_cust->execute*()*;  
 *}* else*{* $sUrl = implode*(*"&", $formError*)*;  
 header*(*"Location:form\_create\_bat.php?".$sUrl*)*;  
 exit;  
 *}*

De plus, et pour faciliter le mécanisme de recherche et d’association client/bon à tirer, il a été nécessaire de retrouver l’id du client non enregistré avant de continuer l’envoi des données.

/\* Looking for his id with his name \*/  
$query\_search\_cust = "SELECT \* FROM customer WHERE custom\_name = :custom\_name";  
$statement\_search\_cust = $PDO->prepare*(*$query\_search\_cust*)*;  
$statement\_search\_cust->bindValue*(*'custom\_name', $custom\_name, $PDO::*PARAM\_STR)*;  
if*(*count*(*$formError*)* === 0*){* $statement\_search\_cust->execute*()*;*}*else*{* $sUrl = implode*(*"&", $formError*)*;  
 header*(*"Location:form\_create\_bat.php?".$sUrl*)*;  
 exit;  
*}*$cust\_name\_id = $statement\_search\_cust->fetch*(*PDO::*FETCH\_OBJ)*;  
$cust\_id = $cust\_name\_id->custom\_id;  
  
/\* Associate the id with the new BAT \*/  
$statement\_bat->bindValue*(*':bat\_custom\_id', $cust\_id, PDO::*PARAM\_INT)*;  
$statement\_bat->execute*()*;

Toutefois, si le client existe d’ores et déjà dans la base de données, alors une simple association du bon à tirer avec le client déjà existant s’effectuera sans pour autant rajouter d’information sur ce client :

/\* Associate directly the id of the customer without add it in the database \*/  
$cust\_name\_id = $statement\_search\_cust->fetch*(*PDO::*FETCH\_OBJ)*;  
$cust\_id = $cust\_name\_id->custom\_id;  
$statement\_bat->bindValue*(*':bat\_custom\_id', $cust\_id, PDO::*PARAM\_INT)*;  
$statement\_bat->execute*()*;

* 1. L’envoi du mail (premières embauches)

Après l’exécution de l’envoi des données, j’ai réalisé une simulation d’envoi d’e-mails aux futurs clients une fois que le bon à tirer sera réalisé par le graphiste.

Je me suis donc inspiré de la documentation de PHP afin de produire cela. Ce qui a donné ce système :

$to = $custom\_email;  
$subject = 'Bon à tirer à valider';  
$message = "Lorem ipsum dolor sit amet, consectetur adipiscing elit,   
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud   
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit   
in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non   
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.";  
$headers = array*(* 'From' => 'webmaster@example.com',  
 'Reply-To' => 'webmaster@example.com',  
 'X-Mailer' => 'PHP/' . phpversion*()  
)*;  
  
mail*(*$to, $subject, $message, $headers*)*;

PHP prend en charge un envoi d’e-mail grâce à la fonction nommée tout simplement *mail()* . Cette dernière doit cependant être accompagné de plusieurs paramètres que nous pouvons constater sur le schéma ci-dessus.

Quatre paramètres ont été instaurés ici : le destinataire (*$to*), le sujet (*$subject*), le message (*$message*) et les différentes options qui devront être visibles dans l’en-tête de l’email et répertoriées dans un tableau (*array()*) pour l’occasion(*$headers*).

Les quatre parties ont été fragmentées en quatre variables qui contiendront les informations souhaitées pour l’envoi de l’e-mail. Ainsi, un appel de ces quatre variables dans les paramètres de la fonction a suffi pour rendre cette dernière opérationnelle.

Dans le cas présent, une simulation d’envoi d’e-mail est effective car la réalisation s’est encore produite au niveau local afin de tester tout cela.

* 1. La sélection du produit

Afin de guider au mieux les utilisateurs et leur proposer les produits dont l’entreprise peut fabriquer, la sélection du type de produit a été mise en place sous forme de liste déroulante dans le formulaire de création.

Pour cela, plusieurs outils ont été élaborés :

\_ La création d’une table spécifique nommée « product » dans la base de données, ainsi à chaque produit est associé à un id permettant de l’identifier à travers toute la base.

\_A partir de là, un fichier a pu être créer dans le dossier PDO de l’environnement de travail (appelé *select\_product.php*). Ce dernier permettra lui aussi d’interroger la base de données comme nous le souhaitons :

<?php  
include*(*'PDO/db\_connect.php'*)*;  
  
$request\_search\_prod = $PDO->query*(* "SELECT \* FROM product"*)*;  
$search\_prod = $request\_search\_prod->fetchAll*(*PDO::*FETCH\_OBJ)*;  
$request\_search\_prod->closeCursor*()*;

Après avoir inclus le processus de connexion, nous élaborons un requête permettant de sélectionner tous les produits de la table « product » et de les afficher dans leur intégralité grâce à la fonction *fetchAll()* retournant un tableau contenant toutes les lignes du jeu d’enregistrements.

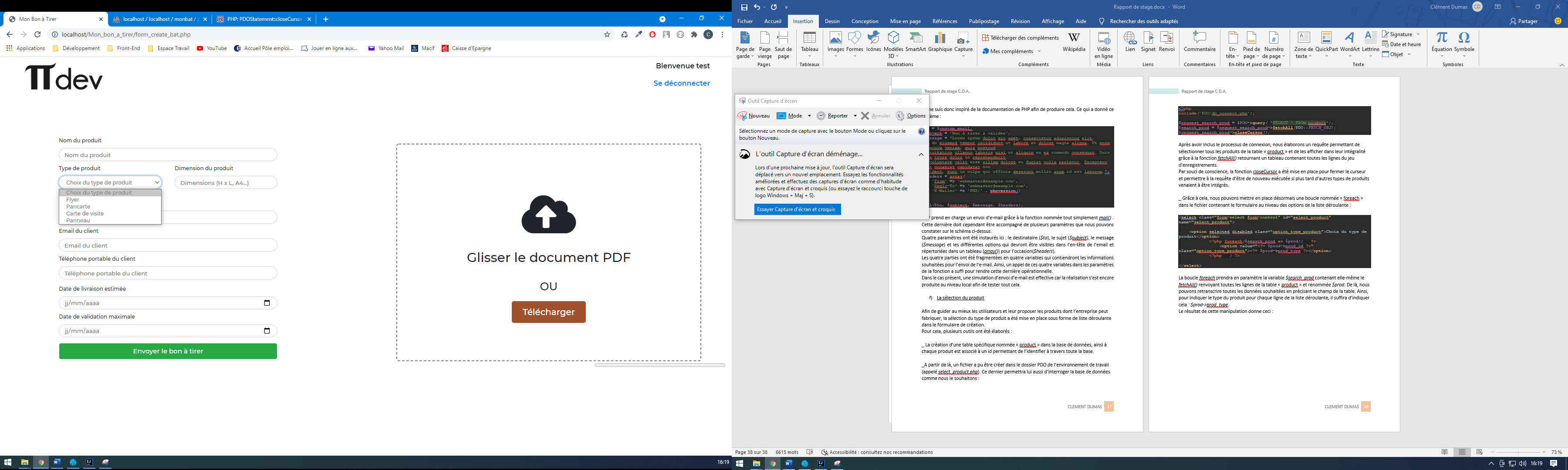
Par souci de conscience, la fonction closeCursor a été mise en place pour fermer le curseur et permettre à la requête d’être de nouveau exécutée si plus tard d’autres types de produits venaient à être intégrés.

\_ Grâce à cela, nous pouvons mettre en place désormais une boucle nommée « foreach » dans le fichier contenant le formulaire au niveau des options de la liste déroulante :

*<*select class="form-select form-control" id="select\_product" name="select\_product"*>  
  
 <*option selected disabled class="option\_type\_product"*>*Choix du type de produit*</*option*>* <?php foreach*(*$search\_prod as $prod*){* ?>  
 *<*option value="<?= $prod->prod\_id ?>" class="option\_type\_product"*>*<?= $prod->prod\_type ?>*</*option*>* <?php *}* ?>  
  
*</*select*>*

La boucle *foreach* prendra en paramètre la variable *$search\_prod* contenant elle-même le *fetchAll()* renvoyant toutes les lignes de la table « product » et renommée *$prod*. De là, nous pouvons retranscrire toutes les données souhaitées en précisant le champ de la table. Ainsi, pour indiquer le type du produit pour chaque ligne de la liste déroulante, il suffira d’indiquer cela : *$prod->prod\_type*.

Le résultat de cette manipulation donne ceci :



* 1. Le Drag and Drop

Le dernier travail qui me fut donné, mais qui n’a pas pu aboutir, a été de configuré le système de drag and drop pour uploader les fichier jpg, jpeg et pdf que le graphiste aurait élaboré et envoyé au client.

L’intégration au formulaire fut réalisée avec ces quelques lignes en HTML :

*<*div class="drag-area col-md-5" ondrop="uploadFile*()*"*>  
 <*div class="icon"*><*i class="fas fa-cloud-upload-alt"*></*i*></*div*>  
 <*p id="text\_upload"*>*Glisser le document PDF*</*p*>  
 <*span*>*OU*</*span*>  
 <*button id="btn\_upload"*>*Télécharger*</*button*>  
 <*input id="input\_upload" name="input\_upload" type="file" onchange="uploadFile*()*" hidden multiple*>  
</*div*>*

Et avec de nombreuses lignes de codes dans le CSS associé, voilà le premier rendu de cet espace :



Ce dernier permet alors de déposer le fichier de façon automatique en prenant le fichier dans un dossier, puis de le glisser et le lâcher dans la zone prévue à cet effet (DROP), ou bien en cliquant sur le bouton « Télecharger » qui ouvrira la fenêtre d’accès au système et permettant de choisir le fichier voulu (DRAG).

Le plus lourd du travail a été de rendre cet espace fonctionnel par le biais du langage JavaScript. Pour cela, un fichier spécifique a été créé et associé à la page du formulaire.

Pour permettre les actions voulues, des conditions et des fonctions ont été mises en place dont voici quelques exemples :

const ***dropArea*** = ***document***.querySelector*(*".drag-area"*)*,  
***dragText*** = ***dropArea***.querySelector*(*"#text\_upload"*)*,  
***buttonUpload*** = ***dropArea***.querySelector*(*"#btn\_upload"*)*,  
***inputUpload*** = ***dropArea***.querySelector*(*"#input\_upload"*)*;  
let ***file***; //This is a global variable and we'll use it inside multiple functions

*La sélection de tous les éléments requis*

// If user Drag File Over DropArea  
***dropArea***.addEventListener*(*"dragover", *(*event*)*=>*{* event.preventDefault*()*; //Preventing from Default behavior  
 ***dropArea***.classList.add*(*"active"*)*;  
 ***dragText***.textContent = "Relâcher pour uploader"  
*})*;  
  
// If user leaves dragged File from DropArea  
***dropArea***.addEventListener*(*"dragleave", *()*=>*{* ***dropArea***.classList.remove*(*"active"*)*;  
 ***dragText***.textContent = "Fais glisser le document PDF"  
*})*;  
  
// If user drop File on DropArea  
***dropArea***.addEventListener*(*"drop", *(*event*)*=>*{* event.preventDefault*()*; //Preventing from Default behavior  
 //getting user select file and [0] this means if user select multiple files then we'll select only the first one  
 ***file*** = event.dataTransfer.files*[*0*]*;  
 showFile*()*; //Calling function  
*})*;

*Les conséquences des actions de l’utilisateur sur le Drag and Drop*

function showFile*(){* let fileType = ***file***.type;  
 let validExtensions = *[*"application/pdf", "image/jpg", "image/jpeg"*]*; //adding some valid image extensions in array  
 if*(*validExtensions.includes*(*fileType*)){* let fileReader = new ***FileReader****()*; //creating new FileReader object  
 fileReader.onload = *()*=>*{* let fileURL = fileReader.result; //passing user file source in fileURL variable  
 let imgTag = `*<*img src="$*{*fileURL*}*" alt="upload\_file"*>*`; //Creating an img tag and passing user selected file source inside src attribute  
 ***dropArea***.innerHTML = imgTag; //adding that created img tag inside dropArea container  
 *}* fileReader.readAsDataURL*(****file****)*;  
 *}*else*{* alert*(*"Le fichier ne peut être accepté"*)*;  
 ***dropArea***.classList.remove*(*"active"*)*;  
 ***dragText***.textContent = "Fais glisser le document PDF"  
 *}  
}*

*La fonction permettant d’afficher le document directement dans la zone*

Pour l’instant, cette fonction ne permet d’afficher que les documents sous format jpg et jpeg et l’upload ne fonctionne qu’avec le bouton « Télécharger » qui renvoi à un fichier nommé « *file\_upload\_parser.php* » permettant le cheminement du document vers un dossier « uploads » :

<?php  
$fileName = $\_FILES*[*"input\_upload"*][*"name"*]*; // The file name  
$fileTmpLoc = $\_FILES*[*"input\_upload"*][*"tmp\_name"*]*; // File in the PHP tmp folder  
$fileType = $\_FILES*[*"input\_upload"*][*"type"*]*; // The type of file it is  
$fileSize = $\_FILES*[*"input\_upload"*][*"size"*]*; // File size in bytes  
$fileErrorMsg = $\_FILES*[*"input\_upload"*][*"error"*]*; // 0 for false... and 1 for true  
if *(*!$fileTmpLoc*) {* // if file not chosen  
 echo "ERROR: Please browse for a file before clicking the upload button.";  
 exit*()*;  
*}*if*(*move\_uploaded\_file*(*$fileTmpLoc, "uploads/$fileName"*)){* echo "$fileName upload is complete";  
*}* else *{* echo "move\_uploaded\_file function failed";  
*}*